MarkLogic サーバー

Application Developer’s Guide

MarkLogic 9
2017 年 5 月

最終更新：9.0-3、2017 年 9 月

目次

Application Developer’s Guide

1.0 MarkLogic サーバーでのアプリケーション開発 14
1.1 MarkLogic サーバーアプリケーション開発の概要 14
1.2 MarkLogic サーバーアプリケーションの開発に必要なスキル 15
1.3 詳細情報の入手先 15

2.0 スキーマの読み込み 17
2.1 データベースの設定 17
2.2 スキーマの読み込み 18
2.3 スキーマの参照 19
2.4 スキーマの利用 20
2.5 スキーマを基準とした XML の検証 21

3.0 MarkLogic サーバーのトランザクションとは 22
3.1 用語と定義 23
3.2 MarkLogic サーバートランザクションの概要 26
3.2.1 主要なトランザクション属性 27
3.2.2 ステートメント境界とは 28
3.2.3 シングルステートメントトランザクションの概要 30
3.2.4 マルチステートメントトランザクションの概要 31
3.3 コミットモード 32
3.4 トランザクションのタイプ 33
3.4.1 トランザクションのタイプの概要 34
3.4.2 XQuery におけるトランザクションのタイプの制御 35
3.4.3 JavaScript におけるトランザクションのタイプの制御 38
3.4.4 クエリトランザクション：ポイントインタイム評価 40
3.4.5 更新トランザクション：読み取り / 書き込みロック 41
3.4.6 例：クエリトランザクションと更新トランザクションの関係 44
3.5 シングルステートメントトランザクションとマルチステートメントトランザクション 45
3.5.1 シングルステートメントトランザクション（自動コミット） 45
3.5.2 マルチステートメントトランザクション（明示的コミット） 46
3.5.3 ステートメントの区切り文字としてのセミコロン 52
3.6 トランザクションモード 54
3.6.1 トランザクションモードの概要 54
3.6.2 Auto トランザクションモード 56
3.6.3 クエリトランザクションモード 57
3.6.4 更新トランザクションモード 58

3.7 xdmp:eval/invoke とのインタラクション 58
3.7.1 xdmp:eval/invoke の isolation 分離オプション 59
3.7.2 デッドロックの回避 61
3.7.3 eval/invoke からの更新をトランザクションの後半で認識 62
3.7.4 xdmp:eval/invoke でのマルチステートメントトランザクションの実行 64
3.8 非トランザクショナル副作用のある関数 65
3.9 Multi-Version Concurrency Control によるブロックの削減 66
3.10 トランザクションの管理 67
3.11 トランザクションの例 68
3.11.1 例：マルチステートメントトランザクションと same-statement 分離 68
3.11.2 例：マルチステートメントトランザクションと different-statement 分離 69
3.11.3 例：xdmp:host-status を使用したトランザクションレポート生成 71

4.0 バイナリドキュメントの使用 73
4.1 用語 73
4.2 バイナリドキュメントの読み込み 74
4.3 バイナリコンテンツ向けの MarkLogic サーバーの設定 74
4.3.1 ラージサイズしきい値の設定 74
4.3.2 バイナリコンテンツのサイズ設定と拡張性 75
4.3.3 バイナリコンテンツの位置の選択 77
4.3.4 フォレスト内にあるラージバイナリデータの合計サイズのモニタリング 78
4.3.5 孤立バイナリの検出と削除 79
4.4 バイナリドキュメントを使用するアプリケーションの開発 80
4.4.1 プロパティを使用した、バイナリドキュメントへのメタデータの追加 81
4.4.2 HTTP レンジリクエストを含むバイナリコンテンツのダウンロード 82
4.4.3 バイナリメール添付ファイルの作成 85
4.5 バイナリドキュメントの操作に役立つビルトイン 86

5.0 XQuery モジュールおよび XSLT スタイルシートのインポートとパスの解決 87
5.1 XQuery のライブラリモジュールとメインモジュール 87
5.1.1 メインモジュール 87
5.1.2 ライブラリモジュール 88
5.2 import/invoke/spawn パスの解決のルール 88
5.3 モジュールキャッシュ処理の注意事項 90
5.4 モジュールインポートのシナリオの例 91

6.0 ライブラリサービスアプリケーション 92
6.1 ライブラリサービスとは 92
6.2 ライブラリサービスを使用したアプリケーションの構築 94
6.3 必要なレンジ要素インデックス 94
6.4 ライブラリサービス API 95
6.4.1 ライブラリサービス API のカテゴリ 96
6.4.2 マネージドドキュメント更新ラッパー関数 96
6.5 ライブラリサービスアプリケーションのセキュリティ上の考慮事項 97
6.5.1 dls-admin ロール 97
6.5.2 dls-user ロール 98
6.5.3 dls-internal ロール 98
6.6 トランザクションとライブラリサービス 98
6.7 マネージドバージョン管理下へのドキュメントの配置 98
6.8 マネージドドキュメントのチェックアウト 99
6.8.1 マネージドドキュメントのチェックアウトステータスの表示 100
6.8.2 マネージドドキュメントのチェックアウトの取り消し 100
6.9 マネージドドキュメントのチェックイン 101
6.10 マネージドドキュメントの更新 101
6.11 保持ポリシーの定義 102
6.11.1 マネージドドキュメントのバージョンのパージ 102
6.11.2 保持規則について 102
6.11.3 保持規則の作成 103
6.11.4 特定のドキュメントバージョンの保持 105
6.11.5 複数の保持規則 105
6.11.6 保持規則の削除 108
6.12 ライブラリサービスでのモジュラードキュメントの管理 108
6.12.1 マネージドモジュラードキュメントの作成 108
6.12.2 マネージドモジュラードキュメントの展開 110
6.12.3 モジュラードキュメントのバージョンの管理 112

7.0 再帰的 typeswitch 式による XML 構造の変換 115
7.1 XML 変換 115
7.1.1 XQuery と XSLT 115
7.1.2 XHTML または XSL-FO への変換 115
7.1.3 typeswitch 式 116
7.2 サンプルの XQuery 変換コード 116
7.2.1 シンプルな例 117
7.2.2 cts:highlight を使用するシンプルな例 118
7.2.3 XHTML へのサンプル変換 119
7.2.4 typeswitch デザインパターンの拡張 121

8.0 ドキュメントとディレクトリのロック 123
8.1 ロックの概要 123
8.1.1 書き込みロック 123
8.1.2 永続的 124
8.1.3 検索可能 124
8.1.4 排他または共有 124
8.1.5 階層型 124
8.1.6 ロックと WebDAV 124
8.1.7 ロックの他の使用方法 124
8.2 ロック API 125
8.3 例：ロックが設定されたドキュメントの URI 検索 125
8.4 例：ドキュメントへのロックの設定 126
8.5 例：ドキュメントのロックの解除 127
8.6 例：ロックの所有者であるユーザーの検索 127

9.0 プロパティドキュメントとディレクトリ 128
9.1 プロパティドキュメント 128
9.1.1 プロパティドキュメントの名前空間とスキーマ 128
9.1.2 プロパティドキュメントの API 131
9.1.3 XPath のプロパティ軸 131
9.1.4 保護されているプロパティ 132
9.1.5 プロパティドキュメント要素に対する要素インデックスの作成 133
9.1.6 サンプルのプロパティドキュメント 133
9.1.7 スタンドアロンのプロパティドキュメント 133
9.2 ドキュメント処理のためのプロパティの使用 134
9.2.1 プロパティ軸の使用によるドキュメント状態の判断 135
9.2.2 ドキュメント処理に関する問題 136
9.2.3 ドキュメント処理のソリューション 136
9.2.4 モジュールを実行するための基本コマンド 137
9.3 ディレクトリ 138
9.3.1 プロパティとディレクトリ 138
9.3.2 ディレクトリと WebDAV サーバー 139
9.3.3 ディレクトリとコレクション 139
9.4 プロパティやディレクトリのパーミッション 140
9.5 例：ディレクトリおよびドキュメントブラウザ 140
9.5.1 ディレクトリブラウザのコード 140
9.5.2 ディレクトリブラウザの設定 142

10.0 ポイントインタイムクエリ 144
10.1 ポイントインタイムクエリを理解する 144
10.1.1 ログ構造化データベースに格納されるフラグメント 145
10.1.2 システムタイムスタンプとマージのタイムスタンプ 145
10.1.3 ポイントインタイムクエリ用のフラグメントが格納される仕組み 146

10.1.4 クエリステートメントだけで利用でき、更新ステートメントでは利用できない 147
10.1.5 すべての補助データベースが最新バージョンを使用する 147
10.1.6 データベース設定の変更はポイントインタイムフラグメントに適用されない 147
10.2 クエリでタイムスタンプを使用する 147
10.2.1 管理画面でポイントインタイムクエリを有効化する 148
10.2.2 xdmp:request-timestamp 関数 149
10.2.3 xdmp:timestamp の実行権限が必要 150
10.2.4 xdmp:eval/xdmp:invoke/xdmp:spawn の timestamp パラメータ 150
10.2.5 XCC におけるリクエストのタイムスタンプ 150
10.2.6 スコア計算に関する考慮事項 151
10.3 xdmp:eval/xdmp:invoke/xdmp:spawn/XCC でポイントインタイムクエリを指定する 152
10.3.1 例：XCC を使用して古いバージョンのドキュメントをクエリする 152
10.3.2 例：削除済みのドキュメントをクエリする 152
10.4 システムタイムスタンプを追跡する 153
10.5 特定のタイムスタンプへのフォレストのロールバック 155
10.5.1 フォレストのロールバックで考慮すべきトレードオフとシナリオ 156
10.5.2 マージのタイムスタンプの設定 157
10.5.3 xdmp:forest-rollback 操作の実行に関する注意 157
10.5.4 1 つあるいは複数のフォレストをロールバックする一般的な手順 158

11.0 システムプラグインフレームワーク 159
11.1 MarkLogic サーバープラグインの仕組み 159
11.1.1 システムプラグインの概要 159
11.1.2 システムプラグインとアプリケーションプラグイン 160
11.1.3 plugin API 160
11.2 システムプラグインモジュールの記述 161
11.3 パスワードプラグインのサンプル 161
11.3.1 パスワードプラグインについて 161
11.3.2 パスワードプラグインの変更 163

12.0 map 関数を使用した名前 / バリューマップの作成 165
12.1 マップ：XQuery で操作するインメモリ構造 165
12.2 map:map XQuery プリミティブ型 166
12.3 XML ノードへのマップのシリアライズ 166
12.4 map API 166
12.5 マップ演算子 167
12.6 例 168
12.6.1 シンプルなマップの作成 168
12.6.2 マップ内の値を返す 168

12.6.3 シリアライズされたマップの作成 169
12.6.4 シーケンス値の追加 169
12.6.5 マップの和集合の作成 170
12.6.6 マップの積集合の作成 171
12.6.7 マップの差演算子の適用 172
12.6.8 単項の否定演算子の適用 173
12.6.9 Div 演算子の適用 174
12.6.10Mod 演算子の適用 175

13.0 関数値 176
13.1 関数値の概要 176
13.2 xdmp:function XQuery プリミティブ型 176
13.3 関数値用の XQuery API 177
13.4 適用した関数がクエリステートメントによる更新である場合 177
13.5 関数値の使用例 177

14.0 モジュラードキュメントアプリケーションによるコンテンツの再利用 180
14.1 モジュラードキュメント 180
14.2 XInclude と XPointer 181
14.2.1 例：シンプルな ID 182
14.2.2 例：xpath() スキーム 182
14.2.3 例：element() スキーム 182
14.2.4 例：xmlns() と xpath() スキーム 183
14.3 CPF XInclude アプリケーションと API 183
14.3.1 XInclude コードと CPF パイプライン 184
14.3.2 必須のセキュリティ権限ー xinclude ロール 184
14.4 モジュラードキュメントアプリケーションで使用する XML の作成 185
14.4.1 <xi:include> 要素 185
14.4.2 <xi:fallback> 要素 185
14.4.3 シンプルな例 186
14.5 モジュラードキュメントアプリケーションの設定 188

15.0 アプリケーションサーバーのアクセス、出力、およびエラーの制御 190
15.1 カスタム HTTP サーバーエラーページの作成 190
15.1.1 カスタム HTTP エラーページの概要 190
15.1.2 エラー XML の形式 191
15.1.3 カスタムエラーページの設定 192
15.1.4 Modules データベース用のエラーハンドラドキュメントには実行パーミッションが必要 192
15.1.5 カスタムエラーページの例 192
15.2 HTTP アプリケーションサーバーの URL 書き換えの設定 193
15.2.1 URL 書き換えの概要 193

15.2.2 シェークスピアの XML コンテンツの読み込み 194
15.2.3 シンプルな URL リライタ 195
15.2.4 URL 書き換えモジュールの作成 198
15.2.5 内部 URL へのアクセスの禁止 199
15.2.6 URL 書き換えとページ相対 URL 199
15.2.7 URL Rewrite トレースイベントの使用 200
15.3 SGML エンティティの出力 202
15.3.1 さまざまな SGML マッピングの設定について 202
15.3.2 アプリケーションサーバー設定での SGML マッピング設定 203
15.3.3 XQuery プログラムにおける SGML マッピングの指定 204
15.4 出力のエンコーディングの指定 204
15.4.1 アプリケーションサーバーの出力のエンコーディング設定 205
15.4.2 出力のエンコーディングを指定するための XQuery ビルトイン 205
15.5 アプリケーションサーバーレベルで出力オプションの指定 206

16.0 インタプリタ型 XQuery リライタの作成による REST Web サービスのサポート 207
16.1 この章で使用する用語 207
16.2 REST ライブラリの概要 208
16.3 シンプルな XQuery リライタとエンドポイント 209
16.4 リライタのマッチ基準に関する注意 211
16.5 options ノード 212
16.6 options ノード要素の検証 215
16.7 URL からの複数のコンポーネントの抽出 216
16.8 エラーの処理 218
16.9 リダイレクトの処理 218
16.10 HTTP 動詞の処理 220
16.10.1 OPTIONS リクエストの処理 221
16.10.2 POST リクエストの処理 223
16.11 パラメータの定義 225
16.11.1 パラメータの型 225
16.11.2 URL 内で指定されたパラメータのサポート 226
16.11.3 必須パラメータ 227
16.11.4 パラメータのデフォルト値 227
16.11.5 値のリストの指定 227
16.11.6 リピータブルパラメータ 228
16.11.7 パラメータキーのエイリアス 228
16.11.8 パラメータ内の正規表現と match/pattern 属性とのマッチング 228
16.12 条件の追加 229
16.12.1 認証条件 230
16.12.2 accept ヘッダ条件 231
16.12.3 ユーザーエージェント条件 231
16.12.4 関数条件 231

16.12.5 and 条件 232
16.12.6 or 条件 232
16.12.7 content-type 条件 232

17.0 宣言型 XML リライタの作成による REST Web サービスのサポート 233
17.1 XML リライタの概要 233
17.2 XML リライタを使用するためのアプリケーションサーバー設定 234
17.3 入出力コンテキスト 234
17.3.1 入力コンテキスト 235
17.3.2 出力コンテキスト 236
17.4 正規表現（Regex） 237
17.5 マッチルール 238
17.5.1 rewriter 239
17.5.2 match-accept 240
17.5.3 match-content-type 241
17.5.4 match-cookie 242
17.5.5 match-execute-privilege 243
17.5.6 match-header 244
17.5.7 match-method 246
17.5.8 match-path 247
17.5.9 match-query-param 250
17.5.10 match-role 252
17.5.11 match-string 253
17.5.12 match-user 254
17.6 システム変数 255
17.7 評価ルール 257
17.7.1 add-query-param 258
17.7.2 set-database 259
17.7.3 set-error-format 260
17.7.4 set-error-handler 261
17.7.5 set-eval 262
17.7.6 set-modules-database 263
17.7.7 set-modules-root 264
17.7.8 set-path 264
17.7.9 set-query-param 265
17.7.10 set-transaction 266
17.7.11 set-transaction-mode 266
17.7.12 set-var 267
17.7.13 trace 268
17.8 終了ルール 270
17.8.1 dispatch 270
17.8.2 error 272
17.9 シンプルなリライタの例 273

18.0 テンプレートに基づく抽出（TDE）.....276
18.1 TDE ドキュメントのセキュリティ.....277
18.2 テンプレートビュー要素.....279
18.2.1 コレクション.....280
18.2.2 ディレクトリ.....281
18.2.3 path-namespaces.....281
18.2.4 コンテキスト.....282
18.2.5 変数.....283
18.3 テンプレートダイアレクトとデータ変換関数.....284
18.3.1 日付 / 時間関数.....285
18.3.2 論理関数とデータ検証.....286
18.3.3 文字列関数.....286
18.3.4 型キャスト処理.....287
18.3.5 算術関数.....288
18.3.6 その他の関数.....288
18.4 テンプレートの検証と挿入.....290
18.5 テンプレートと非準拠ドキュメント.....292
18.6 テンプレートの有効化と無効化.....292
18.7 テンプレートの削除.....293

19.0 リレーショナル操作のためのオプティック API.....294
19.1 JavaScript と XQuery のオプティック API の相違点.....296
19.2 オプティックパイプライン内のオブジェクト.....297
19.3 データアクセス関数.....300
19.3.1 fromView の例.....301
19.3.2 fromTriples の例.....305
19.3.3 fromLexicons の例.....306
19.3.4 fromLiterals の例.....309
19.4 オプティッククエリの種類.....311
19.4.1 基本クエリ.....311
19.4.2 集約とグループ化.....312
19.4.3 行結合.....314
19.4.4 ドキュメントの結合.....321
19.4.5 union、intersect、および except.....325
19.4.6 ドキュメントのクエリ.....330
19.5 オプティック出力の処理.....331
19.6 カラム値を処理するための式関数.....331
19.6.1 式関数に必要な XQuery ライブラリ.....337
19.7 ブール、数値、および文字列演算子に相当する関数.....339
19.8 ノードコンストラクタ関数.....341
19.9 ベストプラクティスとパフォーマンスに関する考慮事項.....343
19.10 オプティック実行プラン.....344
19.11 プランのパラメータ化.....344
19.12 シリアライズされたオプティッククエリ
のエクスポートとインポート.....345

20.0 JSON の使用 347
20.1 JSON、XML、および MarkLogic 347
20.2 MarkLogic が JSON ドキュメントを表現する方法 348
20.3 XPath を使用した JSON ドキュメントのトラバーサル 349
20.3.1 XPath とは 350
20.3.2 XPath の例 350
20.3.3 ノードおよびノード値の選択 351
20.3.4 ノードテスト演算子 353
20.3.5 配列および配列メンバーの選択 354
20.4 JSON ドキュメントに対するインデックスおよびレキシコンの作成 357
20.5 JSON と XML のフィールドクエリの相違点 358
20.6 地理情報データ、テンポラルデータ、およびセマンティックデータの表現 359
20.6.1 地理情報データ 359
20.6.2 日付 / 時間データ 360
20.6.3 セマンティックデータ 360
20.7 ドキュメントのプロパティ 361
20.8 大きな整数値のシリアライゼーション 361
20.9 XQuery における JSON の使用 362
20.9.1 JSON ノードの構築 362
20.9.2 マップからの JSON オブジェクトの構築 364
20.9.3 fn:data との関係 365
20.9.4 JSON ドキュメントの操作 366
20.9.5 例：JSON ドキュメントの更新 367
20.9.6 JSON ドキュメントの検索 369
20.10 サーバーサイド JavaScript における JSON の使用 370
20.11 JSON から XML へ、および XML から JSON への変換 371
20.11.1 変換の方針 371
20.11.2 XML と JSON の間の変換を行う関数 372
20.11.3 JSON から XML への変換 372
20.11.4 カスタム変換のための設定方式について 372
20.11.5 例：basic 方式を使用した変換 373
20.11.6 例：full 方式を使用した変換 374
20.11.7 例：custom 方式を使用した変換 375
20.12 低レベル JSON XQuery API とプリミティブ型 377
20.12.1 使用可能な関数とプリミティブ型 377
20.12.2 例：JSON ノードへのシリアライゼーション 379
20.12.3 例：JSON ノードの項目リストへのパース 379
20.13 JSON ドキュメントの読み込み 381
20.13.1 mlcp を使用した JSON ドキュメントの読み込み 381
20.13.2 Java クライアント API を使用した JSON ドキュメントの読み込み 381
20.13.3 Node.js クライアント API を使用した JSON ドキュメントの読み込み 381
20.13.4 REST クライアント API を使用した JSON の読み込み 382

21.0 トリガーを使用したアクションのスポーン 384
21.1 トリガーの概要 384
21.1.1 トリガーのコンポーネント 384
21.1.2 トリガーで使用されるデータベース 385
21.2 トリガーと Content Processing Framework 386
21.3 プリコミットトリガーとポストコミットトリガー 386
21.3.1 プリコミットトリガー 387
21.3.2 ポストコミットトリガー 387
21.4 トリガーイベント 388
21.4.1 データベースイベント 388
21.4.2 データイベント 388
21.5 トリガーのスコープ 389
21.6 トリガーによって呼び出されるか、スポーンされるモジュール 390
21.6.1 プリコミットトリガーとポストコミットトリガーによるモジュール動作の相違点 390
21.6.2 モジュール外部変数 trgr:uri および trgr:trigger 391
21.7 triggers.xqy によるトリガーの作成および管理 391
21.8 シンプルなトリガーの例 392
21.9 トリガーの無限ループ（トリガーストーム）の回避 394

22.0 ネイティブプラグインの使用 397
22.1 ネイティブプラグインとは 397
22.2 MarkLogic サーバーによるネイティブプラグインの管理方法 398
22.3 ネイティブプラグインライブラリの構築 398
22.4 ネイティブプラグインのパッケージング 399
22.5 ネイティブプラグインのインストール 400
22.6 ネイティブプラグインのアンインストール 402
22.7 実行時のネイティブプラグイン登録 402
22.8 ネイティブプラグインのバージョン管理 403
22.9 読込済プラグインのステータスの確認 404
22.10 プラグインマニフェスト 406
22.11 ネイティブプラグインのセキュリティ上の考慮事項 407
22.12 ネイティブプラグインの例 408

23.0 集計ユーザー定義関数 409
23.1 集計ユーザー定義関数とは 409
23.2 インデータベース MapReduce の概念 409
23.2.1 MapReduce とは 410
23.2.2 インデータベース MapReduce の動作 410
23.3 集計ユーザー定義関数の実装 411
23.3.1 集計 UDF の作成および導入 411
23.3.2 AggregateUDF::map の実装 413
23.3.3 AggregateUDF::reduce の実装 415
23.3.4 AggregateUDF::finish の実装 416
23.3.5 集計 UDF の登録 417

23.3.6 集計 UDF のメモリ管理 419
23.3.7 AggregateUDF::encode および AggregateUDF::decode の実装 420
23.3.8 集計 UDF のエラー処理とロギング 421
23.3.9 集計 UDF の引数処理 422
23.3.10 集計 UDF の型変換 424

24.0 ドキュメントコンテンツのリダクション 427
24.1 用語と定義 428
24.2 リダクションの概要 429
24.2.1 リダクションとは 429
24.2.2 ルールによるリダクション要件の表現 431
24.2.3 複数のインターフェイスを使用したルール適用 431
24.2.4 リダクションロジックの保護 432
24.3 例：リダクションの基本的な使用方法 432
24.3.1 ソースドキュメントのインストール 432
24.3.2 ルールのインストール 434
24.3.3 ルールについて 436
24.3.4 ルールの適用 437
24.4 セキュリティ上の考慮事項 439
24.5 リダクションルールの定義 442
24.5.1 ルール定義の基本 442
24.5.2 リダクション方式の選択 444
24.5.3 リダクション関数の選択 445
24.5.4 XML 名前空間プレフィックスバインドの定義 446
24.5.5 リダクションルールにおける XPath 式の制限 447
24.5.6 複数のドキュメント形式で使用可能なルールの定義 447
24.5.7 XML ルールシンタックスのリファレンス 450
24.5.8 JSON ルールシンタックスのリファレンス 452
24.6 リダクションルールのインストール 454
24.7 リダクションルールの適用 456
24.7.1 概要 456
24.7.2 mlcp を使用したルール適用 457
24.7.3 XQuery を使用したルール適用 458
24.7.4 JavaScript を使用したルール適用 459
24.7.5 ルールの順序は保証されない 460
24.8 リダクションルールの検証 460
24.9 ビルトインリダクション関数のリファレンス 461
24.9.1 mask-deterministic 462
24.9.2 mask-random 466
24.9.3 conceal 469
24.9.4 redact-number 471
24.9.5 redact-us-ssn 473
24.9.6 redact-us-phone 476
24.9.7 redact-email 480
24.9.8 redact-ipv4 482
24.9.9 redact-regex 483

24.10 例：ビルトインリダクション関数の使用 487
24.10.1 ルール例のまとめ 487
24.10.2 XML ルールのインストール 488
24.10.3 JSON ルールのインストール 492
24.10.4 ルールの適用 495
24.10.5 結果の確認 497
24.11 ユーザー定義のリダクション関数 498
24.11.1 ユーザー定義のリダクション関数の実装 499
24.11.2 ユーザー定義のリダクション関数のインストール 500
24.12 例：カスタムリダクションルールの使用 503
24.12.1 例：JavaScript を使用したカスタムリダクション 504
24.12.2 例：XQuery を使用したカスタムリダクション 511
24.13 辞書ベースのマスキングの使用 517
24.13.1 リダクション辞書の定義 517
24.13.2 リダクション辞書のインストール 518
24.13.3 リダクション辞書の使用 519
24.14 例：辞書ベースのマスキング 519
24.14.1 辞書のインストール 520
24.14.2 ルールのインストール 522
24.14.3 ルールの適用 525
24.15 サンプルの実行準備 528

1.0 MarkLogic サーバーでのアプリケーション開発

この章では、MarkLogic サーバーによるアプリケーション開発の概要を説明します。次のセクションで構成されています。

- MarkLogic サーバーアプリケーション開発の概要
- MarkLogic サーバーアプリケーションの開発に必要なスキル
- 詳細情報の入手先

この『*Application Developer's Guide*』では、MarkLogic サーバーを使用してアプリケーションを作成する方法について、概要を説明しています。MarkLogic サーバーの強力な XQuery 検索機能を使用して検索アプリケーションを開発する方法については、『*Search Developer's Guide*』を参照してください。

1.1 MarkLogic サーバーアプリケーション開発の概要

MarkLogic サーバーは、コンテンツ、地理情報データ、数値データ、バイナリデータなど、あらゆる種類のデータを格納するアプリケーションを構築するためのプラットフォームを提供します。開発者は、アプリケーションを開発するプログラミング言語として、コンテンツを検索するための XQuery やサーバーサイド JavaScript を使用してアプリケーションを構築します。これらのアプリケーションは、クライアント API（Java、Node.js、および REST）やその他の Web サービスを介して、または Java から XCC インターフェイスを介して、その他の環境と統合できます。MarkLogic サーバーのみを使用してアプリケーション全体を作成し、XQuery またはサーバーサイド JavaScript で完全にプログラミングすることもできます。

この『*Application Developer's Guide*』では、主に、MarkLogic サーバーでコンテンツアプリケーションおよび検索アプリケーションを構築するために XQuery またはサーバーサイド JavaScript を使用するうえで必要な手法、デザインパターン、および概念に重点を置いて説明します。Java クライアント API、Node.js クライアント API、または REST API を使用している場合は、このガイドで説明する概念の一部が役立ちますが、具体的なガイダンスについては、それぞれの API に関するガイドを参照してください。Java クライアント API を使用してアプリケーションを開発する方法については、『*Java Application Developer's Guide*』を参照してください。REST API を使用してアプリケーションを開発する方法については、『*REST Application Developer's Guide*』を参照してください。Node.js クライアント API を使用してアプリケーションを開発する方法については、『*Node.js Application Developer's Guide*』を参照してください。

1.2 MarkLogic サーバーアプリケーションの開発に必要なスキル

MarkLogic サーバーでアプリケーションを開発するときに役に立つスキルと経験は、以下のとおりです。はじめからこれらのスキルをすべて備えている必要はありません。MarkLogic アプリケーション開発の経験を積むにつれて、これらを身に付けることができます。

- Web 開発のスキル（XHTML、HTTP、ブラウザ間の問題、CSS、Javascript など）。特に HTTP アプリケーションサーバー上で実行されるアプリケーションを開発する場合。
- XML の全体的な理解と知識。
- XQuery のスキル。XQuery を使い始めるには、『*XQuery and XSLT Reference Guide*』を参照してください。
- JavaScript のスキル。MarkLogic のサーバーサイド JavaScript については、『*JavaScript Reference Guide*』を参照してください。
- 検索エンジンおよび全文クエリの理解。
- Java（Java クライアント API または XCC を使用してアプリケーションを開発する場合）。詳細については、『*Java Application Developer's Guide*』または『*XCC Developer's Guide*』を参照してください。
- Node.js（Node.js クライアントを使用してアプリケーションを開発する場合）。詳細については、『*Node.js Application Developer's Guide*』を参照してください。
- 一般的なアプリケーション開発技法（アプリケーション要件の確定、ソースコード管理など）。
- 大規模アプリケーションを導入する場合は、運用手法の管理（大規模ファイルシステムの作成や管理、複数マシンの管理、ネットワーク帯域幅の問題など）。

1.3 詳細情報の入手先

MarkLogic サーバーには、フルセットのドキュメントが用意されており、http://docs.marklogic.com で入手できます。この『*Application Developer's Guide*』では、MarkLogic サーバーアプリケーションの開発に使用される概念やデザインパターンについて説明します。技術的な情報の入手先について、リストを次に示します。

- MarkLogic サーバーをインストールおよびアップグレードする方法については、『*Installation Guide*』を参照してください。また、新機能、および他のリリースとの既知の非互換性のリストについては、『*Release Notes*』を参照してください。
- データベース、フォレスト、アプリケーションサーバー、ユーザー、権限などを作成する方法については、『Administrator's Guide』を参照してください。

- MarkLogic サーバーのセキュリティを使用する方法については、『*Security Guide*』を参照してください。
- ドキュメントの変換やその他の目的でパイプライン処理を作成する方法については、『*Content Processing Framework Guide*』を参照してください。
- XQuery の標準関数、MarkLogic サーバーの組み込み拡張関数（更新、検索、HTTP サーバー機能用）、その他の XQuery ライブラリ関数など、個々の XQuery 関数の構文および使用方法については、『*MarkLogic XQuery and XSLT Function Reference*』を参照してください。
- MarkLogic のサーバーサイド JavaScript については、『*JavaScript Reference Guide*』を参照してください。
- XCC を使用して Java から MarkLogic サーバーのコンテンツにアクセスする方法については、『*XCC Developer's Guide*』を参照してください。
- 言語が検索に与える影響については、『*Search Developer's Guide*』の Language Support in MarkLogic Server を参照してください。コンテンツの言語に関係なく、言語が検索に与える影響を理解することは重要です。
- 検索アプリケーションを開発する方法については、『*Search Developer's Guide*』を参照してください。
- MarkLogic サーバーのトランザクションを構成する要素については、この『*Application Developer's Guide*』の「MarkLogic サーバーのトランザクションとは」（22 ページ）を参照してください。
- その他の開発者向けトピックについては、この『*Application Developer's Guide*』の内容を確認してください。
- パフォーマンス関連の問題については、『*Query Performance and Tuning Guide*』を参照してください。

2.0 スキーマの読み込み

MarkLogic サーバーは、「スキーマデータベース」というコンセプトに基づいています。スキーマデータベースには、MarkLogic サーバーの同一クラスタ内のさまざまなデータベースで共有できるスキーマドキュメントが格納されます。この章では、MarkLogic サーバーへのスキーマドキュメントの読み込みの基本原則を紹介します。以下のセクションで構成されています。

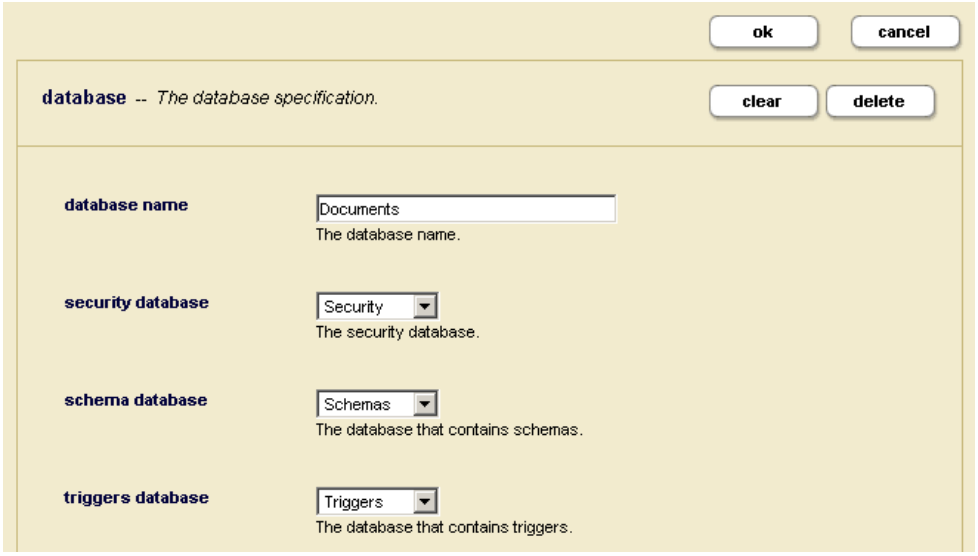
- データベースの設定
- スキーマの読み込み
- スキーマの参照
- スキーマの利用
- スキーマを基準とした XML の検証

管理画面でスキーマを構成する方法の詳細については、『*Administrator's Guide*』の「Understanding and Defining Schemas」の章を参照してください。

2.1 データベースの設定

MarkLogic サーバーでは、インストール時に「Schemas」という名前の空のスキーマデータベースが自動的に作成されます。

作成されたドキュメントデータベースはすべて、Schema データベースと Security データベースの両方を参照するデフォルトでは、新しいデータベースが作成されると、自動的に「Schemas」をスキーマデータベースとして参照します。ほとんどの場合、このデフォルト設定（以下を参照）で問題ありません。



ただし場合によっては、別のスキーマデータベースを参照したほうがいいこともあります。例えば、2 つの別々のデータベースが同じスキーマの別バージョン（名前は同じ）を参照しなければならないことがあります。この場合は、デフォルトの「Schemas」データベースの代わりに使用するデータベースをドロップダウンメニューから選択します。システム内の任意のデータベースをスキーマデータベースとして使用できます。

場合によっては、自分自身をスキーマデータベースとして設定した方が効率的なこともあります。これは、まったく問題のない設定であり、同じドロップダウンメニューから設定できます。この場合は、同一データベース内にアプリケーションに関するコンテンツとスキーマの両方が格納されることになります。

> **注**：自身をスキーマデータベースとして参照するデータベースを作成するには、まず、デフォルトの「Schemas」データベースを参照するデータベースを設定で作成する必要があります。新しいデータベースを作成したら、スキーマデータベースのドロップダウンメニューからそのデータベース自体を指定します。

2.2 スキーマの読み込み

ドキュメントデータベースに、HTTP ならびに XDBC サーバーを接続します。これらの HTTP および XDBC サーバーによって実行されるドキュメントの挿入操作（`xdmp:document-load`、`xdmp:document-insert`、およびさまざまな XDBC ドキュメント挿入メソッドによる）により、これらのサーバーに接続されたドキュメントデータベースにドキュメントが挿入されます。

これにより、スキーマのロードにはやや注意が必要になります。なぜなら、スキーマドキュメントをリクエストする際にシステムが調べるのは、現在のドキュメントデータベースが参照しているスキーマデータベースであるためです。したがって、スキーマドキュメントが必ず、現在の「ドキュメントデータベース」ではなく、現在のデータベースの「スキーマデータベース」に読み込まれるようにする必要があります。

このための方法はいくつかあります。

1. 管理画面のロード機能で、スキーマドキュメントをスキーマデータベースに直接読み込む。ドキュメントをロードするスキーマデータベースの［Database］画面に移動します。右上で［load］タブを選択し、その他のドキュメントと同じようにスキーマをロードします。

2. `xdmp:eval` 組み込み関数を使用する XQuery プログラムを作成できます。現在のデータベースのスキーマデータベースにスキーマを直接読み込むように `<database>` オプションを指定します。

```
xdmp:eval('xdmp:document-load("sample.xsd")', (),
        <options xmlns="xdmp:eval">
            <database>{xdmp:schema-database()}</database>
        </options>)
```

3. 当該スキーマデータベースをドキュメントデータベースとして直接参照する XDBC または HTTP サーバーを作成し、任意のドキュメント挿入関数を使用して 1 つ以上のスキーマをスキーマデータベースにロードできます。この方法は、通常は必要ありません。

4. Schemas データベースを参照する WebDAV サーバーを作成し、次に WebDAV クライアントを使用してスキーマドキュメントをドラッグアンドドロップできます。

2.3 スキーマの参照

ドキュメントをロードするとき（コンテンツのリペアのため）およびクエリを評価するとき（適切なデータ型定義のため）、スキーマはサーバーによって自動的に呼び出されます。ドキュメントに対して、サーバーは現在のドキュメントデータベースが参照するスキーマデータベース内で一致するスキーマを検索します。

1. ターゲット名前空間がマッチするスキーマが見つからない場合、ドキュメントの処理にスキーマは使用されません。

2. 一致するスキーマが 1 つ見つかった場合は、そのスキーマがドキュメントの処理に使用されます。

3. 一致するスキーマが複数見つかった場合は、以下の優先順序に基づいてスキーマが 1 つ選択されます。

   a. ドキュメントのルート要素の `xsi:schemaLocation` または `xsi:noNamespaceSchemaLocation` 属性で URI を指定している場合は、指定された URI のスキーマが使用される。

   b. ターゲット名前空間がマッチする import schema プロローグ式がある場合は、指定された URI のスキーマが使用される。import schema 式のターゲット名前空間と、その式が参照するスキーマドキュメントのターゲット名前空間が一致しない場合は、import schema 式は適用されない。

   c. 現在の HTTP または XDBC サーバーの［Schema］パネル内で設定された名前空間とマッチするスキーマがある場合は、そのスキーマが使用される。設定パネルで指定されたターゲット名前空間がスキーマドキュメントのターゲット名前空間とマッチしない場合、管理画面のスキーマ設定情報は使用されない。

d. これらのルールのいずれも該当しない場合、サーバーは検出した最初のスキーマを使用する。データベース内のドキュメントの順序が定義されていないため、この方法では何が選択されるのか予想できない。このためこの方法は推奨されない。

2.4 スキーマの利用

データベース内のスキーマを指定して明示的に読み込んだほうが便利な場合もあります。例えば、スキーマを外部で利用したい場合やスキーマ駆動型のクエリ処理を行う場合です。

スキーマは、このシステムでは他のドキュメントと同じように扱われます。つまり、挿入、読み込み、更新、および削除ができます。スキーマが異なっている点は、通常は、ドキュメントデータベース自体ではなく、二次的なスキーマデータベースに格納されることです。

スキーマに関して最も多い処理は、スキーマの読み込みです。サーバーからスキーマを明示的に取得する方法は 2 つあります。

1. `xdmp:eval` を使用する XQuery プログラムを作成できます。現在のデータベースのスキーマデータベースからスキーマを直接読み込むように `<database>` オプションを指定します。例えば、以下の式は前述のコード例でロードされたスキーマドキュメントを返します。

```
xdmp:eval('doc("sample.xsd")', (),
  <options xmlns="xdmp:eval">
    <database>{xdmp:schema-database()}</database>
  </options>)
```

   ビルトイン関数である `xdmp:schema-database` を使用することで、`sample.xsd` ドキュメントが必ず、現在のデータベースのスキーマデータベースから読み取られるようになります。

2. 当該スキーマデータベースをドキュメントデータベースとして直接参照する XDBC または HTTP サーバーを作成し、XQuery で、スキーマデータベースに格納されたスキーマの読み取り、分析、更新などができます。この方法は、ほとんどの場合必要ありません。

スキーマに関するその他のタスクも、同様に行うことができます。例えば、スキーマを削除する必要がある場合は、前述の 2 つのいずれかで（`xdmp:document-delete("sample.xsd")` を使用して）モデル化されたアプローチを使用できます。

2.5 スキーマを基準とした XML の検証

XQuery の `validate` 式を使用して、要素がスキーマに従っているか（妥当であるか）を検証できます。validate 式の詳細については、『*XQuery and XSLT Reference Guide*』の Validate Expression、および W3C の XQuery に関する勧告（http://www.w3.org/TR/xquery/#id-validate）を参照してください。

ドキュメントをロード前に検証する場合は、まずドキュメントのノードを取得し、このノードを検証してから、データベースに挿入します。例：

```
xquery version "1.0-ml";

(:
   this will validate against the schema if it is in scope,
   but will validate it without a schema if there is no
   in-scope schema
:)
let $node := xdmp:document-get("c:/tmp/test.xml")
return
try { xdmp:document-insert("/my-valid-document.xml",
        validate lax { $node } )
    }
catch ($e) { "Validation failed: ",
             $e/error:format-string/text() }
```

次の例では strict 検証を使用し、検証元のスキーマをインポートします。

```
xquery version "1.0-ml";
import schema "my-schema" at "/schemas/my-schema.xsd";

(:
   this will validate against the specified schema, and
   will fail if the schema does not exist (or if it is
   not valid according to the schema)
:)
let $node := xdmp:document-get("c:/tmp/test.xml")
return
try { xdmp:document-insert("/my-valid-document.xml",
        validate strict { $node } )
    }
catch ($e) { "Validation failed: ",
             $e/error:format-string/text() }
```

3.0 MarkLogic サーバーのトランザクションとは

MarkLogic サーバーは、データの整合性を保証するトランザクショナルシステムです。この章では、MarkLogic サーバーのトランザクションモデルについて説明します。以下のセクションで構成されています。

- 用語と定義
- MarkLogic サーバートランザクションの概要
- コミットモード
- トランザクションのタイプ
- シングルステートメントトランザクションとマルチステートメントトランザクション
- トランザクションモード
- xdmp:eval/invoke とのインタラクション
- 非トランザクショナル副作用のある関数
- Multi-Version Concurrency Control によるブロックの削減
- トランザクションの管理
- トランザクションの例

XCC Java アプリケーションでマルチステートメントおよび XA/JTA トランザクションを使用する方法の詳細については、『*XCC Developer's Guide*』を参照してください。

3.1 用語と定義

トランザクション（データの一貫性）は、ほとんどのデータベースにおいて核となる機能ですが、その意味するところはシステムによって若干異なります。これらの違いを適切にまた包括的に理解するためには、用語を明確に定義しておく必要があります。これらの用語の意味が混乱しないように、このセクションでは、この章や MarkLogic サーバー関連文書で使用される用語のいくつかを定義しておきます。また、これらの用語の定義は、MarkLogic サーバーにおけるトランザクションの説明に役立つ開始点となります。

用語	定義
ステートメント	W3C XQuery 標準で定義されている XQuery のメインモジュールの 1 つです。MarkLogic サーバーによって評価されます。メインモジュールは、オプションのプロローグと完全な XQuery 式で構成されています。 サーバーサイド JavaScript プログラム（または「スクリプト」）は、トランザクションに使用される 1 つのステートメントであるとみなされます。 ステートメントには、「クエリステートメント」と「更新ステートメント」の 2 種類があります。これは、ステートメントの評価前に静的分析によって判断されます。
クエリステートメント	更新呼び出しを含まないステートメントです。 クエリステートメントでは、読み込みに関してデータベースの一貫性が保持されます。クエリステートメントはデータベースの状態を変更しないため、サーバーは扱われているトランザクションに応じて、ロックあるいはライトウェイトロックを適用しないように最適化します。
更新ステートメント	更新を実行する可能性のある（つまり、更新呼び出しが 1 つ以上含まれる）ステートメントです。 このステートメントは、実行時に更新を実行するかどうかを問わず、更新ステートメントに分類できます。更新ステートメントの実行の際には「読み取り／書き込みロック」が使用され、アクセス対象ドキュメントに必要なロックを適用します。

用語	定義
トランザクション	1 つあるいは複数のステートメントで、「すべて失敗」あるいは「すべて成功」のいずれかになるもの。 トランザクションには、「更新トランザクション」と「クエリ（読み取りのみ）トランザクション」の 2 種類があります。これは、トランザクションのタイプと、トランザクション内のステートメントの種類によって決まります。トランザクションには、「シングルステートメントトランザクション」と「マルチステートメントトランザクション」の 2 種類があります。これは、作成時のコミットモードによって決まります。
トランザクションモード	新規作成されるトランザクションのトランザクションタイプとコミット方法を制御します。モードには、`auto`、`query`、`update` の 3 種類があります。デフォルトである `auto` モードでは、トランザクションはすべてシングルステートメントトランザクションです。`query` モードまたは `update` モードでは、トランザクションはすべてマルチステートメントトランザクションです。
シングルステートメントトランザクション	`auto` コミットモードで作成されたトランザクション。シングルステートメントトランザクションに含まれるステートメントは、常に 1 つだけです。正常完了時には自動的にコミットされ、エラー時にはロールバックされます。
マルチステートメントトランザクション	明示的なコミットモードで作成されたトランザクション。まとめて「コミット」または「ロールバック」される 1 つまたは複数のステートメントで構成されます。このトランザクションでは、1 つのステートメントによる変更は、コミット前であっても同一トランザクション内の後続のステートメントによって認識されます。マルチステートメントトランザクションは、`xdmp:commit` または `xdmp.commit` を呼び出して明示的にコミットする必要があります。
クエリトランザクション	更新を一切行わないトランザクション（読み取りのみのトランザクション）。これは、`auto` 更新モードの 1 つの「クエリステートメント」、あるいはクエリトランザクションタイプで作成されたあらゆるトランザクションです。クエリトランザクションで更新しようとすると、`XDMP-UPDATEFUNCTIONFROMQUERY` が発生します。 クエリトランザクションではロックを使用する代わりに、ある種のシステムタイムスタンプを使用し、読み込みに関してデータベースの一貫性が保持されます。

用語	定義
更新トランザクション	更新（データベースの変更) を実行できるトランザクション。auto コミットモードの 1 つの更新ステートメント、あるいは更新トランザクションタイプで作成されたあらゆるトランザクションです。 更新トランザクションの実行の際には「読み取り・書き込みロック」が使用され、アクセス対象ドキュメントに必要なロックを適用します。
コミット	トランザクションを終了し、トランザクションで加えられた変更がデータベース内で認識されるようにします。シングルステートメントトランザクションの場合は、ステートメントが正常に完了した時点で自動的にコミットされます。マルチステートメントトランザクションは、xdmp:commit を使って明示的にコミットされますが、実際にコミットされるのは、呼び出しステートメントが正常に完了した場合のみです。
ロールバック	トランザクションをただちに終了し、このトランザクションによるすべての更新を破棄します。エラーが発生した場合、すべてのトランザクションは自動的にロールバックされます。マルチステートメントトランザクションは、xdmp:rollback を使って明示的にロールバックすることもできます。あるいは、タイムアウトにより、または xdmp:commit を使用せずにセッションの最後に到達したことにより暗黙でロールバックすることもできます。
システムタイムスタンプ	MarkLogic サーバーが管理する数値です。システム内のいずれかのデータベースにおいて変更（クラスタ内のホストからの設定変更を含む）が発生するたびに増加します。データベース内の各フラグメントにはシステムタイムスタンプが付けられます。タイムスタンプの範囲によって、フラグメントの有効期間がわかります。

用語	定義
読み取り / 書き込みロック	ドキュメントの読み取りならびに書き込みに対するロックです。 MarkLogic サーバーでは、更新ステートメントの際に読み取り・書き込みロックを使用します。更新トランザクションでは必要な場合しかロックを使用しないので、更新ステートメントは常にドキュメントの最新バージョンを認識します。ドキュメントはロックされた時点から、一貫性が保持されます。いったんドキュメントがロックされると、他のトランザクションに含まれる更新ステートメントは、ロックが解除されるまで待機し、解除後にドキュメントを更新します。詳細については、「更新トランザクション：読み取り / 書き込みロック」（41 ページ）を参照してください。
プログラム	評価のために MarkLogic サーバーに送信される、XQuery コードの拡張バージョン。`.xqy` ファイル内のクエリ式、`xdmp:eval` ステートメント内の送信された XQuery コードなどがあります。プログラムの構成要素としては、呼び出しモジュール内のコードだけではなく、インポート済みモジュール（呼び出しモジュールが呼び出すもの）、また呼び出されるあらゆるモジュールなどがあります。
セッション	MarkLogic サーバーインスタンスにおける、データベースとの「会話」。セッションは、接続情報、資格情報、トランザクション設定などの状態情報をカプセル化します。セッションの詳細な内容は、会話のコンテキストによって変わってきます。詳細については、「セッション」（51 ページ）を参照してください。
リクエスト	アプリケーションサーバー、タスクサーバー、`xdmp:eval` など、あらゆる手段によるプログラムの呼び出し。また、アプリケーションサーバーへのある種のクライアント呼び出しもリクエストです（例：XCC による　XML の読み込み、HTTP　による画像のダウンロード、WebDAV によるドキュメントのロックなど）。

3.2 MarkLogic サーバートランザクションの概要

このセクションでは、MarkLogic サーバーにおける以下の主要なトランザクションの概念の概要を説明します。

- 主要なトランザクション属性
- ステートメント境界とは

- シングルステートメントトランザクションの概要
- マルチステートメントトランザクションの概要

その後、この章の残りの部分ではこれらの概念について詳細に説明します。

3.2.1 主要なトランザクション属性

MarkLogic では、次のトランザクションモデルがサポートされます。

- シングルステートメントトランザクション。ステートメントの終了時に自動的にコミットされます。これは、MarkLogic のデフォルトトランザクションモデルです。
- マルチステートメントトランザクション。複数のリクエストやステートメントを含むことができ、明示的にコミットされる必要があります。

マルチステートメントトランザクション内の 1 つのステートメントによる更新は、同じトランザクション内の後続のステートメントによって認識されますが、トランザクション外部で実行されているコードによっては認識されません。

アプリケーションは、どちらかまたは両方のトランザクションモデルを使用できます。シングルステートメントトランザクションは、ほとんどのアプリケーションに適しています。マルチステートメントトランザクションは強力ですが、アプリケーションが複雑になります。自分のトランザクションモデルに適している概念を重視してください。

トランザクションには、「シングルステートメント」と「マルチステートメント」の区別だけではなく、「更新」と「クエリ」のタイプがあります。トランザクションタイプによって、許可されている操作、ロックできるかどうか、およびロックのタイミングと方法が決まります。デフォルトでは、トランザクションタイプが自動的に検出されますが、タイプを明示的に指定することもできます。

トランザクションモデル（シングル / マルチステートメント）、コミットモード（auto/explicit）、およびトランザクションタイプ（auto/query/update）は、トランザクション作成時に決定されます。例えば、コードのブロックが、`same-statement` 分離を使用した `xdmp:eval`（XQuery）または `xdmp.eval`（JavaScript）の呼び出しによって評価される場合は、呼び出し元のトランザクションコンテキストでコードブロックが実行されるため、呼び出したコードが設定を変更しようとしても、トランザクション設定は呼び出し元によって決定されます。

XQuery とサーバーサイド JavaScript では、デフォルトのトランザクションセマンティックがわずかに異なります。各言語でのデフォルトの動作、および動作の変更の情報を次の表に示します。詳細については、「トランザクションのタイプ」（33 ページ）を参照してください。

言語	デフォルトのトランザクション動作	代替
XQuery	シングルステートメント、自動コミット、トランザクションタイプの自動検出	トランザクションタイプを明示的に `update` または `query` に設定するには、`update` プロローグオプションを使用します。コミットモードを `auto`（シングルステートメント）または `explicit`（マルチステートメント）に設定するには、`commit` オプションを使用します。`xdmp:eval` や `xdmp:invoke` などの関数でオプションを指定して同様に制御することもできます。
サーバーサイド JavaScript	シングルステートメント、自動コミット、クエリトランザクションタイプ	明示的にトランザクションタイプを update に設定し、コミットモードが auto（シングルステートメント）と explicit（マルチステートメント）のどちらであるかを制御するには、`declareUpdate` 関数を使用します。トランザクションタイプの自動検出は使用できません。`xdmp.eval` や `xdmp.invoke` などの関数でオプションを指定して同様に制御することもできます。

ステートメントは、クエリステートメント（読み取りのみ）または更新ステートメントです。XQuery では、トランザクションのタイプを明示的に指定しない限り、最初の（または唯一の）ステートメントタイプによってトランザクションのタイプが決まります。ステートメントのタイプは、静的分析によって判断されます。JavaScript では、明示的にトランザクションを update に設定しない限り、クエリステートメントタイプであるとみなされます。

トランザクションのコンテキストでは、「ステートメント」の意味が XQuery と JavaScript で異なります。詳細については、「ステートメント境界とは」（28 ページ）を参照してください。

3.2.2 ステートメント境界とは

トランザクションは「ステートメント」との関連で言及されることが多いため、サーバーサイドプログラミング言語でステートメントを構成する要素について理解しておく必要があります。

- XQuery におけるトランザクショナルステートメント
- サーバーサイド JavaScript におけるトランザクショナルステートメント

3.2.2.1 XQuery におけるトランザクショナルステートメント

XQuery では、トランザクション用のステートメントは、メインモジュールとして実行できる 1 つの完全な XQuery ステートメントです。区切り文字としてセミコロンを使用して、1 つのコードブロックに複数のステートメントを含めることができます。

例えば、次のコードブロックには 2 つのステートメントが含まれています。

```
xquery version "1.0-ml";
xdmp:document-insert('/some/uri/doc.xml', <data/>);
(: end of statement 1 :)

xquery version "1.0-ml";
fn:doc('/some/uri/doc.xml');
(: end of statement 2 :)
```

デフォルトでは、上記のコードが、2 つの自動検出自動コミットトランザクションとして実行されます。

このコードをマルチステートメントトランザクションとして評価した場合、両方のステートメントは同一トランザクション内で実行されます。トランザクションは、評価のコンテキストに応じて、オープンのままになるか、明示的なコミットがないためにコードの最後でロールバックされます。

詳細については、「ステートメントの区切り文字としてのセミコロン」（52 ページ）を参照してください。

3.2.2.2 サーバーサイド JavaScript におけるトランザクショナルステートメント

JavaScript では、含まれる JavaScript ステートメントの数にかかわらず、スクリプトの全体またはメインモジュールが、トランザクションのための 1 つのステートメントであるとみなされます。例えば、次のコードは複数の JavaScript ステートメントを含んでいますが、1 つのトランザクショナル「ステートメント」です。

```
'use strict';
declareUpdate();
xdmp.documentInsert('/some/uri/doc.json',
{property: 'value'});
console.log('I did something!');
// end of module
```

デフォルトでは、上記のコードは単一のトランザクション内で実行され、スクリプトの最後で完了します。このコードをマルチステートメントトランザクションのコンテキストで評価した場合、トランザクションはスクリプトの完了後もオープンのままになります。

3.2.3 シングルステートメントトランザクションの概要

デフォルトモデル（シングルステートメント、自動コミット）を使用する場合は、以下の概念を理解しておく必要があります。

シングルステートメントトランザクションのコンセプト	詳細情報
ステートメントは、1 つのトランザクション内で実行される。	シングルステートメントトランザクションとマルチステートメントトランザクション
このトランザクションに含まれるのは、ステートメントが 1 つだけである。	シングルステートメントトランザクション（自動コミット）
トランザクションは、各ステートメントが終わるごとに自動的にコミットされる。	シングルステートメントトランザクション（自動コミット）
トランザクションは、「更新」タイプと「クエリ」タイプのいずれかである。 • クエリトランザクションは、ロックの代わりにシステムタイムスタンプを使用。 • 更新トランザクションではロックを使用。	トランザクションのタイプ
XQuery クエリの場合、トランザクションのタイプは MarkLogic によって自動的に検出するか、明示的に設定することができる。 JavaScript では、明示的に update に設定しない限り、クエリであるとみなされる。自動検出は使用できない。	トランザクションのタイプ
ステートメントによる更新は、ステートメント（トランザクション）が完了するまで認識されない。	更新トランザクション：読み取り / 書き込みロック
XQuery では、1 つのメインモジュールに複数のステートメントを含める場合は、ステートメント / トランザクションの区切り文字としてセミコロンを使用できる。 各 JavaScript プログラムは、含まれる JavaScript ステートメントの数にかかわらず、トランザクションのための 1 つのステートメントであるとみなされる。	ステートメント境界とは シングルステートメントトランザクション（自動コミット） ステートメントの区切り文字としてのセミコロン

3.2.4 マルチステートメントトランザクションの概要

マルチステートメントトランザクションを使用する場合は、以下の概念を理解しておく必要があります。

マルチステートメントトランザクションのコンセプト	詳細情報
ステートメントは、1 つのトランザクション内で実行される。	シングルステートメントトランザクションとマルチステートメントトランザクション
トランザクションには 1 つあるいは複数のステートメントがあり、「すべて失敗」あるいは「すべて成功」のいずれかになる。	マルチステートメントトランザクション（明示的コミット）
マルチステートメントトランザクションは、`xdmp:commit` で明示的にコミットする必要がある。	マルチステートメントトランザクション（明示的コミット） マルチステートメントトランザクションのコミット
ロールバックは暗黙で、または明示的（`xdmp:rollback`）に行う。	マルチステートメントトランザクション（明示的コミット） マルチステートメントトランザクションのロールバック
トランザクションは、「更新」タイプと「クエリ」タイプのいずれかである。 • クエリトランザクションでは、ロックの代わりにシステムタイムスタンプを使用。 • 更新トランザクションではロックを使用。	トランザクションのタイプ
XQuery クエリの場合、トランザクションのタイプは MarkLogic によって自動的に検出するか、明示的に設定することができる。 JavaScript では、明示的に update に設定しない限り、クエリであるとみなされる。自動検出は使用できない。	トランザクションのタイプ
トランザクションは、1 つのセッション内で実行される。	セッション

マルチステートメントトランザクションのコンセプト	詳細情報
セッションには、トランザクションモードプロパティとして `auto`、`query`、または `update` がある。各モードでは以下の内容が異なる。 • トランザクションのタイプ • コミットの方法 • トランザクション内のステートメントの個数	トランザクションモード
コミットモードを explicit に設定すると、常にマルチステートメントトランザクション、明示的コミットが必要なトランザクションが作成される。	シングルステートメントトランザクションとマルチステートメントトランザクション マルチステートメントトランザクション（明示的コミット）
ステートメントによる更新は、ステートメントが完了するまで認識されない。	更新トランザクション：読み取り / 書き込みロック
1 つのステートメントによる更新は、トランザクションがまだオープンである間は、同一トランザクション内の後続のステートメントによって認識される。	マルチステートメントトランザクション（明示的コミット）
XQuery では、1 つのトランザクションに複数のステートメントを含める場合は、ステートメントの区切り文字としてセミコロンを使用できる。	ステートメント境界とは ステートメントの区切り文字としてのセミコロン

3.3 コミットモード

トランザクションは、auto または explicit コミットモードで実行できます。

デフォルトの動作は auto コミットであり、「ステートメント境界とは」（28 ページ）に記載されているように、ステートメントの最後でトランザクションがコミットされます。

explicit コミットモードは、マルチステートメントトランザクション用です。このモードでは、`xdmp:commit`（XQuery）または `xdmp.commit`（JavaScript）を呼び出してトランザクションを明示的にコミットしたり、`xdmp:rollback`（XQuery）または `xdmp.rollback`（JavaScript）を呼び出してトランザクションを明示的にロールバックしたりする必要があります。これにより、複数のステートメントまたはリクエストでトランザクションをオープンしたままにすることができます。

コミットモードは次の方法で制御できます。

- XQuery プロローグオプション `xdmp:commit` を auto または explicit に設定します。
- `explicitCommit` オプションを指定して JavaScript 関数 `declareUpdate` を呼び出します。これは、コミットモードとトランザクションのタイプの両方に影響します。詳細については、「JavaScript におけるトランザクションのタイプの制御」（38 ページ）を参照してください。
- `xdmp:eval`（XQuery）、`xdmp.eval`（JavaScript）、または eval/invoke ファミリの他の関数でコードを評価するときに `commit` オプションを設定します。このオプションをサポートする関数の完全なリストについては、以下の表を参照してください。
- `xdmp.set-transaction-mode`（XQuery）または `xdmp.setTransactionMode`（JavaScript）を呼び出します。これは、コミットモードとクエリタイプの両方に影響します。詳細については、「トランザクションモード」（54 ページ）を参照してください。

次の関数は、`commit` および `update` オプションをサポートします。これらのオプションを使用すると、コミットモード（explicit または auto）およびトランザクションのタイプ（update、query、または auto）を制御できます。詳細については、`xdmp:eval` または `xdmp.eval` の関数リファレンスを参照してください。

XQuery	JavaScript
`xdmp:eval`	`xdmp.eval`
`xdmp:javascript-eval`	`xdmp.xqueryEval`
`xdmp:invoke`	`xdmp.invoke`
`xdmp:invoke-function`	`xdmp.invokeFunction`
`xdmp:spawn`	`xdmp.spawn`
`xdmp:spawn-function`	

3.4 トランザクションのタイプ

このセクションでは、トランザクションのタイプに関して以下のような内容を取り上げます。これは、シングルステートメントならびにマルチステートメントトランザクションのどちらにも該当します。

- トランザクションのタイプの概要
- XQuery におけるトランザクションのタイプの制御

- JavaScript におけるトランザクションのタイプの制御
- クエリトランザクション：ポイントインタイム評価
- 更新トランザクション：読み取り / 書き込みロック
- 例：クエリトランザクションと更新トランザクションの関係

3.4.1 トランザクションのタイプの概要

トランザクションのタイプごとに、許可される操作とロックの使用の有無が異なります。トランザクションは、「更新」タイプと「クエリ」タイプのいずれかです。ステートメントにも「更新」タイプと「クエリ」タイプがあり、実行できる操作が異なります。

更新のトランザクションとステートメントは、クエリと更新の両方を実行できます。クエリのトランザクションとステートメントは読み取りのみで、更新は実行しません。クエリトランザクションは更新ステートメントを含むことができますが、このステートメントが実行時に更新を試みると、エラーが発生します。詳細は、「クエリトランザクションモード」（57 ページ）を参照してください。

MarkLogic サーバーでは、トランザクションのタイプを次の方法で決定します。

- auto：トランザクションのタイプが静的分析によって判断されます。Auto は、XQuery におけるデフォルトの動作です。
- explicit：トランザクションのタイプが更新またはクエリであることをコードで指定します。指定するには、オプションを使用するか、`xdmp:set-transaction-mode`（XQuery）または `xdmp.setTransactionMode`（JavaScript）を呼び出すか、または `declareUpdate`（JavaScript のみ）を呼び出します。

詳細については、「XQuery におけるトランザクションのタイプの制御」（35 ページ）または「JavaScript におけるトランザクションのタイプの制御」（38 ページ）を参照してください。

クエリトランザクションではシステムタイムスタンプを使って、特定の時点におけるデータベースの一貫性のあるスナップショットを使用します（ロックは使用しません）。更新トランザクションでは、読み取り・書き込みロックを使用します。詳細は「クエリトランザクション：ポイントインタイム評価」（40 ページ）および「更新トランザクション：読み取り / 書き込みロック」（41 ページ）を参照してください。

以下の表で、トランザクションのタイプ、ステートメント、ロックの動作の関係をまとめてあります。これは、シングルステートメントならびにマルチステートメントトランザクションのどちらにも該当します。

トランザクション タイプ	ステートメント	動作
query	query	ドキュメントの特定時点（ポイントインタイム）の状態を提示。ロックは不要です。
	update	ランタイムエラー。
update	query	適宜、読み取りロックが使用されます。
	update	適宜、読み取り・書き込みロックが使用されます。

3.4.2 XQuery におけるトランザクションのタイプの制御

デフォルトの自動検出がアプリケーションに適していない場合を除き、トランザクションのタイプを明示的に設定する必要はありません。トランザクションのタイプが「auto」（デフォルト）である場合、コードの静的分析によってトランザクションのタイプが決まります。マルチステートメントトランザクションでは、トランザクションのタイプを自動検出するときに、最初のステートメントのみが調べられます。

トランザクションのタイプを明示的に設定するには、次の手順に従います。

- XQuery プロローグで `xdmp:update` オプションを宣言します。または、
- そのモードで実行する必要のあるトランザクションを作成する前に、`xdmp:set-transaction-mode` を呼び出します。または、
- 関数（`xdmp:eval`、`xdmp:invoke`、`xdmp:spawn` など）に渡される options ノードで `update` オプションを設定します。

メインモジュールの先頭など、最初のトランザクションの作成前にトランザクションのタイプを設定する必要がある場合は、`xdmp:update` プロローグオプションを使用します。例えば、次のコードは、プロローグオプションを使用しているため、マルチステートメントの更新トランザクションとして実行されます。

```
declare option xdmp:commit "explicit";
declare option xdmp:update "true";

let $txn-name := "ExampleTransaction-1"
return (
  xdmp:set-transaction-name($txn-name),
  fn:concat($txn-name, ": ",
```

```
    xdmp:host-status(xdmp:host())
      //hs:transaction[hs:transaction-name eq $txn-name]
        /hs:transaction-mode)
);
xdmp:commit();
```

詳細については、『*XQuery and XSLT Reference Guide*』の xdmp:update および xdmp:commit を参照してください。

xdmp:set-transaction-mode を使用してトランザクションモードを設定すると、コミットのセマンティック（auto または explicit）とトランザクションのタイプ（query または update）の両方に影響します。トランザクションの途中でトランザクションモードを設定しても、現行のトランザクションに影響はありません。トランザクションモードを設定すると、セッション全体のトランザクション作成セマンティックに影響します。

次の例は、トランザクションモードを別の値に設定しても現在実行中のトランザクションには影響しないことを示すために、xdmp:set-transaction-mode を使用します。この例では、xdmp:host-status を使用して、現在のトランザクションのモードを調べます（例では、xdmp:host-status の結果で、関連するトランザクションを簡単に選択できるように xdmp:set-transaction-name のみを使用します）。

```
xquery version "1.0-ml";

declare namespace
hs="http://marklogic.com/xdmp/status/host";

(: The first transaction created will run in update mode :)
declare option xdmp:commit "explicit";
declare option xdmp:update "true";

let $txn-name := "ExampleTransaction-1"
return (
  xdmp:set-transaction-name($txn-name),
  xdmp:set-transaction-mode("query"),  (: no effect on
  current txn :)
  fn:concat($txn-name, ": ",
    xdmp:host-status(xdmp:host())
      //hs:transaction[hs:transaction-name eq $txn-name]
        /hs:transaction-mode)
  );

(: complete the current transaction :)
xdmp:commit();
```

```
(: a new transaction is created, inheriting query
mode from above :)
declare namespace hs="http://marklogic.com/xdmp/status/host";
let $txn-name := "ExampleTransaction-2"
return (
  xdmp:set-transaction-name($txn-name),
    fn:concat($txn-name, ": ",
      xdmp:host-status(xdmp:host())
        //hs:transaction[hs:transaction-name eq $txn-name]
          /hs:transaction-mode)
  );
```

上記の例を Query Console に貼り付けて実行すると、結果がテキストで表示されます。最初のトランザクションが、xdmp:transaction-mode の指定に従って更新モードで実行され、2 番目のトランザクションが、`xdmp:set-transaction-mode` の指定に従ってクエリモードで実行されます。

```
ExampleTransaction-1: update
ExampleTransaction-2: query
```

プログラムには、option 宣言と `xdmp:set-transaction-mode` 呼び出しを複数含めることができます。ただし、設定が考慮されるのは、トランザクションの作成時のみです。最初のステートメントが評価される直前に、トランザクションが暗黙で作成されます。例：

```
xquery version "1.0-ml";
declare option xdmp:commit "explicit";
declare option xdmp:update "true";

(: begin transaction :)
"this is an update transaction";
xdmp:commit();
(: end transaction :)

xquery version "1.0-ml";
declare option xdmp:commit "explicit";
declare option xdmp:update "false";

(: begin transaction :)
"this is a query transaction";
xdmp:commit();
(: end transaction :)
```

次の関数は、commit および update オプションをサポートします。これらのオプションを使用すると、コミットモード（explicit または auto）およびトランザクションのタイプ（update、query、または auto）を制御できます。詳細については、xdmp:eval または xdmp.eval の関数リファレンスを参照してください。

XQuery	JavaScript
xdmp:eval	xdmp.eval
xdmp:javascript-eval	xdmp.xqueryEval
xdmp:invoke	xdmp.invoke
xdmp:invoke-function	xdmp.invokeFunction
xdmp:spawn	xdmp.spawn
xdmp:spawn-function	

3.4.3 JavaScript におけるトランザクションのタイプの制御

デフォルトでは、サーバーサイド JavaScript は、シングルステートメントで自動コミットのクエリトランザクションで実行されます。トランザクションのタイプは次の方法で制御できます。

- declareUpdate 関数を使用して、トランザクションのタイプを update に設定したり、コミットのセマンティックを指定したりします。または、
- 関数（xdmp.eval、xdmp.invoke、xdmp.spawn など）に渡される options ノードで update オプションを設定します。または、
- そのモードで実行する必要のあるトランザクションを作成する前に。xdmp.setTransactionMode を呼び出します。

3.4.3.1 declareUpdate を使用したトランザクションの設定

デフォルトでは、JavaScript は、auto コミットモードとクエリトランザクションタイプで実行されます。declareUpdate 関数を使用すると、トランザクションタイプを update に変更したり、コミットモードを auto から explicit に変更したりすることができます。

MarkLogic では、JavaScript コードが更新を実行するかどうかを、静的分析によって判断することができません。JavaScript コードが更新を加える場合は、次のいずれかの要件を満たす必要があります。

- declareUpdate 関数を呼び出して、コードが更新を加えることを示します。

- コードの呼び出し元がトランザクションタイプを、更新を許容するタイプに設定します。

引数なしで `declareUpdate` を呼び出すと、auto コミットモードで更新トランザクションタイプと同等になります。つまり、コードが更新を実行でき、シングルステートメントトランザクションとして実行されます。更新は、JavaScript コードの完了時に自動的にコミットされます。

また、以下に示すように `explicitCommit` オプションを `declareUpdate` に渡すこともできます。`explicitCommit` のデフォルト値は false です。

```
declareUpdate({explicitCommit: boolean});
```

`explicitCommit` を `true` に設定した場合、コードは、新しいマルチステートメント更新トランザクションを開始します。トランザクションは、JavaScript コードから復帰する前に、または他のコンテキスト（JavaScript コードの呼び出し元や同一トランザクション内で実行中の別のリクエストなど）で、明示的にコミットまたはロールバックする必要があります。

例えば、`explicitCommit` を使用して、XCC からアドホッククエリリクエスト内でマルチステートメントトランザクションを開始し、その後、他のリクエストからそのトランザクションをコミットします。

呼び出し元がトランザクションのタイプを update に設定した場合、更新を実行するためにコードで `declareUpdate` を呼び出す必要はありません。このような状況で `declareUpdate` を呼び出す場合は、その結果のモードが、呼び出し元の設定したモードと競合しないようにしてください。

詳細については、『*JavaScript Reference Guide*』の declareUpdate Function を参照してください。

3.4.3.2 呼び出し元でのトランザクションの設定

コードが呼び出される前にトランザクションのタイプとコミットのモードが設定される場合の例を次に示します。

- コードは、XQuery 関数 `xdmp:javascript-eval` や JavaScript 関数 `xdmp.eval` などの eval/invoke 関数経由で呼び出され、呼び出し元は、`commit`、`update`、または `transaction-mode` オプションを指定します。
- コードはサーバーサイドのインポート変換であり、mlcp コマンドラインツールで使用します。
- コードは、サーバーサイドの変換、拡張、またはその他のカスタマイズであり、Java、Node.js、または REST クライアント API で呼び出されます。事前に設定されるモードは、コードの実行のきっかけとなった操作によって異なります。

- コードは、XCC セッションのコンテキストで実行され、そのコミットモードやトランザクションのタイプはクライアントが設定します。

次の関数は、`commit` および `update` オプションをサポートします。これらのオプションを使用すると、コミットモード（explicit または auto）およびトランザクションのタイプ（update、query、または auto）を制御できます。詳細については、`xdmp:eval`（XQuery）または `xdmp.eval`（JavaScript）の関数リファレンスを参照してください (JavaScript)。

XQuery	JavaScript
`xdmp:eval`	`xdmp.eval`
`xdmp:javascript-eval`	`xdmp.xqueryEval`
`xdmp:invoke`	`xdmp.invoke`
`xdmp:invoke-function`	`xdmp.invokeFunction`
`xdmp:spawn`	`xdmp.spawn`
`xdmp:spawn-function`	

3.4.4 クエリトランザクション：ポイントインタイム評価

クエリトランザクションは読み取りのみで、ドキュメントをロックすることはありません。このセクションでは、クエリトランザクションに関して以下のような内容を取り上げます。

- システムタイムスタンプとフラグメントのバージョン管理
- クエリトランザクションはタイムスタンプを使って実行（ロック不使用）
- クエリトランザクションは、タイムスタンプに基づいてドキュメントの最新バージョンを認識

3.4.4.1 システムタイムスタンプとフラグメントのバージョン管理

MarkLogic サーバーにおけるトランザクションを理解するには、ドキュメントがどのように格納されているかを理解しておく必要があります。ドキュメントは、1 つあるいは複数のフラグメントから構成されています。ドキュメントが作成されると、そのフラグメントはそれぞれ、1 つあるいは複数のスタンドに格納されます。スタンドが集まってフォレストとなり、フォレストが集まってデータベースとなります。データベースには、フォレストが 1 つあるいは複数含まれています。

スタンド内の各フラグメントにはシステムタイムスタンプが付けられます。システムタイムスタンプの範囲は、このフラグメントのバージョンの有効期間に対応しています。ドキュメントが更新される際には、変更されるフラグメントの新しいバージョンが作成されます。新バージョンのフラグメントは、新しいスタンドに格納され、有効なタイムスタンプが新しく付けられます。その後、古いスタンドと新しいスタンドがマージされ、最新バージョンのフラグメントのみを含んだ新しいスタンドが作成されます。ポイントインタイムクエリも、マージの際にどのバージョンのフラグメントが格納・保持されるに関係しています。マージ後、古いスタンドは削除されます。

フラグメントに付けられた有効なシステムタイムスタンプの範囲によって、ステートメントはトランザクションの際にどのバージョンのドキュメントを使用したらよいのかが分かります。マージの詳細については、『*Administrator's Guide*』の Understanding and Controlling Database Merges を参照してください。格納されるドキュメントのバージョンにポイントインタイムクエリが及ぼす影響の詳細については、「ポイントインタイムクエリ」（144 ページ）を参照してください。

3.4.4.2 クエリトランザクションはタイムスタンプを使って実行（ロック不使用）

クエリトランザクションは、トランザクション作成時のシステムタイムスタンプを使って実行されます。`xdmp:request-timestamp` を呼び出すと、クエリトランザクション中のどの時点でも、同じシステムタイムスタンプが返されます。空のシーケンスが返されることはありません。クエリトランザクションは、どんなドキュメントに対してもロックをしません。このためこのトランザクションの実行中に、他のトランザクションがこのドキュメントを読み出したり更新したりできます。

3.4.4.3 クエリトランザクションは、タイムスタンプに基づいてドキュメントの最新バージョンを認識

MarkLogic サーバーでは、クエリトランザクションが作成されると、その時点でのシステムタイムスタンプ（`xdmp:request-timestamp` 関数を呼び出すと返される数値）が付けられ、ドキュメントの最新バージョン（タイムスタンプの値がクエリトランザクションより小さい）だけが使われます。タイムスタンプを使うことで、トランザクションの処理中に対象となるドキュメントが更新・削除された場合でも、このトランザクション内のすべてステートメントはドキュメントの同一バージョンに一貫してアクセスしていることが保証されます。

3.4.5 更新トランザクション：読み取り / 書き込みロック

更新トランザクションによってデータベースが変更されてしまう可能性があるので、トランザクションの整合性を保つために、ドキュメントをロックします。更新トランザクションでは、読み取り・書き込みロックが使用されます（クエリトランザクションのようにタイムスタンプは使用しません）。このセクションには、次の内容が含まれます。

- 更新トランザクションの特定
- ロックはオンデマンドで適用され、トランザクションの終了まで保持される
- 更新の認識

3.4.5.1 更新トランザクションの特定

シングルステートメントトランザクションのうち、静的分析によってステートメントに更新が発見されたものは、更新トランザクションとして扱われることになります。このトランザクションは、その固有のロジックによっては実際には何も更新しない可能性があります。しかし、(静的分析によって) 更新トランザクションであると判断されたシングルステートメントトランザクションは、更新トランザクションとして実行されます (クエリトランザクションとしてではなく)。例えば、以下のトランザクションは、`xdmp:document-insert` が決して実行されないにもかかわらず、更新トランザクションとして実行されます。

```
if ( 1 = 2 )
then ( xdmp:document-insert("fake.xml", <a/>) )
else ()
```

マルチステートメントトランザクションでは、トランザクションのタイプは常に、トランザクション作成の時点で有効なトランザクションのタイプ設定に対応しています。トランザクションのタイプが明示的に `update` に設定されている場合、更新を行うステートメントが含まれていなくても、このトランザクションは更新トランザクションとなります。更新トランザクションでは、すべてのステートメントに対してロックが使用されます (更新が実行されるかどうかを問わず)。

更新トランザクション中は、`xdmp:request-timestamp` を呼び出すと必ず、空のシーケンスが返されます。つまり、`xdmp:request-timestamp` が値を返す場合、このトランザクションは更新トランザクションではなく、クエリトランザクションです。

3.4.5.2 ロックはオンデマンドで適用され、トランザクションの終了まで保持される

更新トランザクションではタイムスタンプを利用しません。このため、対象ドキュメントは、このトランザクション内のいずれかのステートメントが最初にアクセスした時点のものが最新版として認識されます。更新トランザクションでは評価を完了させるため、読み込みや書き込みの対象となるすべてのドキュメントに必ずロックをかけます。このため、他のトランザクションが行った更新の「半分」や「一部」が、更新トランザクションに提示されることは絶対にありません。ステートメントは必ず「トランザクション」なのです。

ロックはいったん適用されると、このトランザクションが完了するまで保持されます。こうすることで、読み取りロックがかかったドキュメントを他のトランザクションが更新できなくなります。このため読み取りに関してこのドキュメントの一貫性が保持されます。クエリ (読み取り) 処理では、読み取りロックが必要です。更新処理では、読み取り・書き込みロックが必要です。

更新トランザクション内のステートメントが更新を実行する場合、ドキュメントに対して読み取り・書き込みロックがかけられます（あるいはすでに読み取りロックがかかっている場合は、これが読み取り・書き込みロックに変換されます）。読み取り・書き込みロックは排他ロックです。読み取り・書き込みロックは、他のトランザクションで使用されているロックがすべて解除されない限り、使用できません。

ロックの有効期間は、特にマルチステートメントトランザクションの場合に考慮しておく必要があります。例えば以下のシングルステートメントでは、読み取り・書き込みロックが使用されるのは、`xdmp:node-replace` まで来てからです。

```
  (: query statement, no locks needed :)
fn:doc("/docs/test.xml");
  (: update statement, readers/writers lock acquired :)
fn:node-replace(fn:doc("/docs/test.xml")/node(),
<a>hello</al>);
  (: readers/writers lock released :)
  (: query statement, no locks needed :)
fn:doc("/docs/test.xml");
```

この例をマルチステートメントトランザクションとして書き直すと、ロックは 3 つのステートメントすべてに対して使用されます。

```
declare option xdmp:transaction-mode “update”;

  (: read lock acquired :)
fn:doc("/docs/test.xml");
  (: the following converts the lock to a readers/writers
  lock :)
fn:node-replace(fn:doc("/docs/test.xml")/node(),
<a>hello</al>);
  (: readers/writers lock still held :)
fn:doc("/docs/test.xml");

  (: after the following statement, txn ends and locks
  released :)
xdmp:commit()
```

3.4.5.3 更新の認識

更新は、トランザクションの更新ステートメントが完了しないと認識されません。つまり更新ステートメントの時点では認識されません。更新トランザクションがコミットされてから、この更新は他のトランザクションに認識されるようになります。更新トランザクションの一部として実行されるプリコミットトリガーでは、コミットされる前に更新を認識します。トランザクションモデルによって、更新の認識方法が変わります。これは、コミットのタイミングが異なるためです。

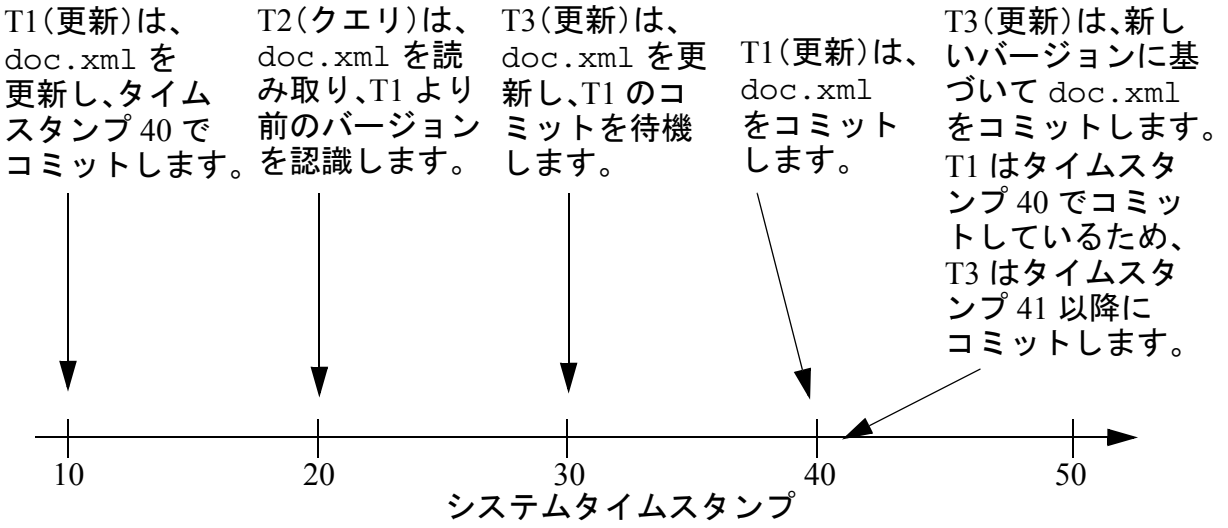
デフォルトのシングルステートメントトランザクションモデルでは、ステートメントが完了した時点で、自動的にコミットされます。新しく更新されたドキュメントを使用するには、更新とアクセスを分けて、それぞれに対してシングルステートメントトランザクションを作成します。あるいはマルチステートメントトランザクションを使用します。

マルチステートメントトランザクションでは、1 つのステートメントによる変更は、この更新ステートメントが完了した時点で、同一トランザクション内の後続のステートメントによって認識されます。変更は、`xdmp:commit` を呼び出すまでこのトランザクションの外では認識されません。

同一ドキュメントに対して複数の更新が競合する場合、更新ステートメントは実行されません。例えば、同一ステートメントにおいて、あるノードを更新し、これに子ノードを追加することはできません。シングルステートメントにおいて競合する更新を同一ドキュメントに実行しようとすると、`XDMP-CONFLICTINGUPDATES` 例外となり、失敗します。

3.4.6 例：クエリトランザクションと更新トランザクションの関係

以下の図では 3 つのトランザクション（T1、T2、T3）の関係を表しています。



T1 は更新トランザクションで、その実行には長い時間がかかるものとします。タイムスタンプ 10 で開始し、タイムスタンプ 40 でコミットします（つまり、この更新ステートメントの実行中に更新などの変更が 30 件あったということです）。

T2 は、T1 が更新しているドキュメント（`doc.xml`）を読み取るときに、タイムスタンプが 20 以下の最新バージョンであると認識しますが、これは、T1 が使用している更新前のバージョンです。

T3 がこのドキュメントを更新しようとしますが、T1 によって読み取り・書き込みロックがかけられているため、ロックが解除されるまで待機します。T1 がコミットされ、ロックが解除されます。その後、T3 はこのドキュメントの新しく更新されたバージョンを対象にして、更新を実行します。この更新がコミットされるのは、タイムスタンプ 41 の時点です。

3.5 シングルステートメントトランザクションとマルチステートメントトランザクション

このセクションでは、MarkLogic サーバーがサポートする 2 つのトランザクションモデル（シングルステートメントとマルチステートメント）の詳細と違いを説明します。次の内容が含まれます。

- シングルステートメントトランザクション（自動コミット）
- マルチステートメントトランザクション（明示的コミット）
- ステートメントの区切り文字としてのセミコロン

3.5.1 シングルステートメントトランザクション（自動コミット）

MarkLogic サーバーでは、デフォルトでは、トランザクションはすべてシングルステートメントで自動的にコミットされます。このデフォルトモデルでは、各ステートメントを評価するために、それぞれトランザクションが作成されます。ステートメントが完了すると、トランザクションは自動的にコミットされ（あるいはエラーの場合はロールバック）トランザクションが終わります。

サーバーサイド JavaScript では、JavaScript プログラム（または「スクリプト」）が、トランザクションの観点からはシングル「ステートメント」であるとみなされます。詳細については、「ステートメント境界とは」（28 ページ）を参照してください。

シングルステートメントトランザクションでは、ステートメントが完了し、トランザクションがコミットされるまで、このステートメントによる更新が、このステートメント以外には認識されません。

ほとんどのアプリケーションでは、シングルステートメントのモデルが適しています。このモデルではトランザクションを詳しく知らなくても大丈夫ですし、またアプリケーションがそれほど複雑になりません。

- ステートメントとトランザクションは、ほぼ同じ意味
- トランザクションのタイプは静的分析によって判断される
- ステートメントが正常に完了した場合、トランザクションは自動的にコミットされる
- エラーが発生した場合には、このステートメントによる更新は自動的にロールバックされる

シングルステートメントトランザクションによる更新は、ステートメントが完了するこのステートメント以外には認識されない詳細については、「更新の認識」（43 ページ）を参照してください。

XQuery では、プログラムにシングルステートメントトランザクションを複数含める場合に、区切り文字としてセミコロンを使用します。詳細については、「ステートメントの区切り文字としてのセミコロン」（52 ページ）を参照してください。

> **注：** サーバーサイド JavaScript では、更新を実行するために `declareUpdate()` 関数を使用する必要があります。詳細については、「JavaScript におけるトランザクションのタイプの制御」（38 ページ）を参照してください。

3.5.2 マルチステートメントトランザクション（明示的コミット）

コミットモードが「explicit」のコンテキストでトランザクションが作成されると、トランザクションはマルチステートメントトランザクションになります。このセクションでは、次の関連トピックについて取り上げます。

- マルチステートメントトランザクションの特徴
- マルチステートメントトランザクションのコミット
- マルチステートメントトランザクションのロールバック
- セッション

トランザクションのタイプおよびコミットモードの設定の詳細については、「トランザクションのタイプ」（33 ページ）を参照してください。

Java でマルチステートメントトランザクションを利用する場合の詳細については、『*XCC Developer's Guide*』の「マルチステートメントトランザクション」を参照してください。

3.5.2.1 マルチステートメントトランザクションの特徴

マルチステートメントトランザクションでは、アプリケーションがより複雑になるため、MarkLogic サーバーにおけるトランザクションを詳しく理解している必要があります。マルチステートメントトランザクションでは、以下のようになります。

- XQuery では、同一トランザクション内の複数のステートメントをセミコロンで区切ります。
- サーバーサイド JavaScript では、含まれる JavaScript ステートメントの数に関係なく、プログラム全体（スクリプト）がシングルトランザクションステートメントであるとみなされます。

- 各ステートメントは、同一トランザクション内の先行するステートメントの変更を認識します。
- 同一トランザクション内のステートメントは、すべてコミットされるか、すべて失敗するかのいずれかになります。
- トランザクションをコミットするには、`xdmp:commit`（XQuery）または `xdmp.commit`（JavaScript）を使用する必要があります。
- トランザクションは、`xdmp:rollback`（XQuery）または `xdmp.rollback`（JavaScript）を使用してアボートできます。

マルチステートメントトランザクションは、作成されたデータベースにバインドされます。あるデータベースコンテキストで作成されたトランザクション ID を使用して、別のデータベース上の同一トランザクションで操作を実行することはできません。

マルチステートメントトランザクションのステートメントには順番があります（別個のリクエストで実行されるものであっても）。つまり、トランザクション内の 1 つのステートメントが終了してから、次のステートメントが開始されます。これらのステートメントが実行されるのは別個のリクエストの場合でもそうなります。

マルチステートメントトランザクションが終了するのは、`xdmp:commit` または `xdmp.commit` で明示的にコミットされた場合、`xdmp:rollback` または `xdmp.rollback` で明示的にロールバックされた場合、タイムアウト、エラー、セッション完了によって暗黙でロールバックされた場合のみです。マルチステートメントトランザクションを明示的にコミットあるいはロールバックしなかった場合、ロックが保持され、リソースが占有されたままになる可能性があります。この状態は、トランザクションがタイムアウトするか、そのセッションが終了するまで持続します。タイムアウトあるいはセッション終了の時点で、このトランザクションはロールバックされます。このため、マルチステートメントトランザクションは常に明示的にコミットあるいはロールバックすることを推奨します。

以下の例には、マルチステートメントトランザクションが 3 つ含まれています（`commit` プロローグオプションを使用しているため）。1 つめのトランザクションは明示的にコミットされます。2 つめは明示的にロールバックされます。3 つめは、コミットやロールバックがなくセッションが終了した時点で暗示的にロールバックされます。この例を Query Console で実行すると、`xdmp:eval` で `different transaction` 分離を使用した場合と同等になります。そのため、最後のトランザクションは、セッションの終了によりクエリの最後まで到達したときにロールバックされます。セッションにおけるマルチステートメントトランザクションの詳細については、「セッション」（51 ページ）を参照してください。

```
xquery version "1.0-ml";

declare option xdmp:commit "explicit";
(: Begin transaction 1 :)
```

```
xdmp:document-insert('/docs/mst1.xml', <data/>);
(: This statement runs in the same txn, so sees
/docs/mst1.xml :)
xdmp:document-insert('/docs/mst2.xml', fn:doc
('/docs/mst1.xml'));
xdmp:commit();
(: Transaction ends, updates visible in database :)

declare option xdmp:commit "explicit";
(: Begin transaction 2 :)
xdmp:document-delete('/docs/mst1.xml');
xdmp:rollback();
(: Transaction ends, updates discarded :)

declare option xdmp:commit "explicit";
(: Begin transaction 3 :)
xdmp:document-delete('/docs/mst1.xml');
(: Transaction implicitly ends and rolls back due to
 :   reaching end of program without a commit :)
```

「更新トランザクション：読み取り / 書き込みロック」（41 ページ）で説明したように、マルチステートメントの更新トランザクションではロックを使用します。マルチステートメントトランザクションには、クエリと更新の両方が含まれる可能性があります。マルチステートメントの更新トランザクションに含まれるクエリでは、必要に応じて読み取りロックを使用します。このトランザクションに含まれる更新では、必要に応じてこのロックを読み取り・書き込みロックに置き換えるか、あるいは新しい読み取り・書き込みロックを使用します。

マルチステートメントのクエリトランザクションでは、ロックの代わりにタイムスタンプを使用し、このトランザクション内のすべてのステートメントに対して、このデータベースの読み込みに関する一貫性を提供します。これについては、「クエリトランザクション：ポイントインタイム評価」（40 ページ）で説明します。システムタイムスタンプの値は、クエリトランザクションの作成時に確認されます。このため、このトランザクション内のすべてのステートメントにおいて、対象ドキュメントのバージョンは同じになります。

3.5.2.2 マルチステートメントトランザクションのコミット

マルチステートメントトランザクションは、`xdmp:commit` によって明示的にコミットします。マルチステートメントの更新トランザクションが `xdmp:commit` を呼び出さなかった場合は、トランザクションの終了時に更新がすべて失われます。更新をコミットしないでトランザクションをそのままにしておいた場合、ロックが継続されたり、リソースを取られたりします。

更新がコミットされると、このトランザクションは終了し、新しいトランザクション内の後続するステートメントの評価に移ります。例：

```
xquery version "1.0-ml";

declare option xdmp:commit "explicit";

(: Begin transaction 1 :)
xdmp:document-insert('/docs/mst1.xml', <data/>);
(: This statement runs in the same txn, so sees
/docs/mst1.xml :)
xdmp:document-insert('/docs/mst2.xml', fn:doc
('/docs/mst1.xml'));
xdmp:commit();
(: Transaction ends, updates visible in database :)
```

xdmp:commit を呼び出すと、ステートメントの呼び出しが正常に終了した場合にのみ更新がコミットされ、トランザクションが終了します。つまり、xdmp:commit を呼び出した後でも、ステートメントのコミットの完了前にエラーになった場合は、更新内容が失われる可能性があります。このため、ステートメントの最後には xdmp:commit を使用することを推奨します。

以下の例では、エラーが発生しても更新が保持されています。これは、xdmp:commit を呼び出すステートメントが必ず完了するためです。

```
xquery version "1.0-ml";

declare option xdmp:commit "explicit";

(: transaction created :)
xdmp:document-insert("not-lost.xml", <data/>)
, xdmp:commit();
fn:error(xs:QName("EXAMPLE-ERROR"), "An error occurs here");
(: end of session or program :)

(: ==> Insert is retained because the statement
       calling commit completes sucessfully.:)
```

対照的に、この例では更新が失われます。これは、xdmp:commit と同じステートメント内でエラーが発生しているために、ステートメントが正常にコミットされないためです。

```
xquery version "1.0-ml";

declare option xdmp:commit "explicit";

(: transaction created :)
xdmp:document-insert("lost.xml", <data/>)
```

```
, xdmp:commit()
, fn:error(xs:QName("EXAMPLE-ERROR"), "An error occurs
here");
(: end of session or program :)

(: ==> Insert is lost because the statement
       terminates with an error before commit can occur.:)
```

例外がキャッチされなかった場合、トランザクションがロールバックされます。マルチステートメントトランザクション内のコードが例外を引き起こしてトランザクションがアボートされることを避けるには、このコードを try-catch ブロックでラップし、catch ハンドラで適切に対応します。例：

```
xquery version "1.0-ml";

declare option xdmp:commit "explicit";

xdmp:document-insert("/docs/test.xml", <a>hello</a>);
try {
  xdmp:document-delete("/docs/nonexistent.xml")
} catch ($ex) {
  (: handle error or rethrow :)
  if ($ex/error:code eq 'XDMP-DOCNOTFOUND') then ()
  else xdmp:rethrow()
}, xdmp:commit();
(: start of a new txn :)
fn:doc("/docs/test.xml")//a/text()
```

3.5.2.3 マルチステートメントトランザクションのロールバック

マルチステートメントトランザクションのロールバックは、暗黙で（エラーやセッション終了により）、あるいは明示的に（xdmp:rollback または xdmp.rollback を使用して）行われます。xdmp:rollback を呼び出すと、現在のトランザクションがすぐに終了します。後続のステートメントの評価は、新しいトランザクション内で継続されます。例：

```
xquery version "1.0-ml";

declare option xdmp:commit "explicit";
                                (: begin transaction :)
xdmp:document-insert("/docs/mst.xml", <data/>);
xdmp:commit()
, "this expr is evaluated and committed";
                                (: end transaction :)
                                (:begin transaction :)
declare option xdmp:commit "explicit";
```

```
xdmp:document-insert("/docs/mst.xml", <data/>);
xdmp:rollback()
                                    (: end transaction :)
, "this expr is never evaluated";
                                    (:begin transaction :)
"execution continues here, in a new transaction"
                                    (: end transaction :)
```

最後が xdmp:rollback のステートメントの結果は必ず、空のシーケンスとなります。

xdmp:rollback で明示的にロールバックすることを推奨します。セッションの終了を待つ暗示的なロールバックをしようとして、トランザクションをそのままにしておくと、セッションがタイムアウトするまでロックがかかり、リソースを取られてしまいます。これはかなり長い時間がかかる可能性があります。例えば、1 つの HTTP セッションには複数の HTTP リクエストが含まれる場合があります。詳細については、「セッション」（51 ページ）を参照してください。

3.5.2.4 セッション

セッションは、MarkLogic サーバーインスタンスにおける、データベースとの「会話」です。セッションは、接続情報、資格情報、トランザクション設定などの、この「会話」の情報をカプセル化します。マルチステートメントトランザクションを使う場合、以下のような理由から、それぞれのセッションのどの時点で評価が行われるかを理解しておく必要があります。

- トランザクションは、セッションの属性の 1 つです。
- トランザクションはコミットされなかった場合、セッション終了時に自動的にロールバックされます。

例えば、different-transaction 分離を使って xdmp:eval（XQuery）または xdmp.eval（JavaScript）で評価されるクエリは、それ自身のセッション内で実行され、呼び出し元のトランザクションモード設定を継承することはありません。また、eval クエリの評価が最後までいってもクエリトランザクションがまだオープン（＝コミットされていない）の場合、このトランザクションは自動的にロールバックされます。

これとは対照的に、HTTP セッションでは、複数の HTTP リクエストによって実行されているクエリにトランザクション設定が適用されることがあります。コミットされていないトランザクションは、HTTP セッションがタイムアウトするまで、オープンのままになります（タイムアウトまでは、かなり時間がかかる場合があります）。

セッションの詳細な内容は、「会話」のコンテキストによって変わってきます。以下の表は、MarkLogic サーバーのアプリケーションで一般的に見られるセッションとのその寿命をまとめたものです。

セッションのタイプ	セッションの寿命
HTTP HTTP アプリケーションサーバーとやり取りする HTTP クライアント。	セッションがまだ存在しないクライアントから最初の HTTP リクエストを受信すると、セッションが作成されます。セッションは、タイムアウトするまで複数のリクエストにわたって存続します。
XCC XDBC アプリケーションサーバーとやり取りする XCC Java アプリケーション。	Session オブジェクトがインスタンス化されるとセッションが作成され、Session オブジェクトのファイナライズ、Session.close() の呼び出し、またはセッションのタイムアウトまで存続します。
以下のように評価されるスタンドアロンクエリ。 • `xdmp:eval` または `xdmp:invoke` で `different-transaction` 分離を使って • `xdmp:spawn` によって • タスクサーバー上のタスクとして	eval/invoke/spawn が実行されたクエリまたはタスクを評価するためにセッションが作成され、そのクエリまたはタスクが完了すると終了します。

セッションタイムアウトは、アプリケーションサーバーの設定です。詳細については、『*XQuery and XSLT Reference Guide*』の `admin:appserver-set-session-timeout`、またはアプリケーションサーバーの管理画面の［Session Timeout］設定を参照してください。

3.5.3 ステートメントの区切り文字としてのセミコロン

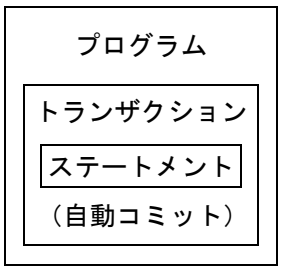
MarkLogic サーバーは、ステートメント間の区切り文字としてセミコロン（ ; ）を XQuery ボディに含めるように XQuery 言語を拡張しています。ステートメントは、出現する順序で評価されます。トランザクションにおいてセミコロンで区切られた各ステートメントは、次のステートメントが開始する前に完全に評価されます。

シングルステートメントトランザクションでは、ステートメントの区切り文字がトランザクションセパレータでもあります。セミコロンで区切られた各ステートメントは、それ自身のトランザクションとして評価されます。クエリステートメントとして評価されるセミコロン区切り部分と、更新ステートメントとして評価されるセミコロン区切り部分がプログラム内で混在することが可能です。ステートメントは、出現する順序で評価され、更新ステートメントの場合は、各ステートメントが次のステートメントの開始前にコミットします。

`auto` コミットモード（デフォルト）のセミコロン区切りステートメントは、マルチステートメントトランザクションではありません。各ステートメントはシングルステートメントトランザクションです。最初の更新がコミットされたあとに、次がランタイムエラーとなった場合でも、最初のトランザクションはロールバックされません。後続のトランザクションが失敗した場合にロールバックが必要になるロジックがある場合は、そのロジックを XQuery コードに追加するか、マルチステートメントトランザクションを使用するか、プリコミットトリガーを使用する必要があります。トリガーについては、「トリガーを使用したアクションのスポーン」（384 ページ）を参照してください。

マルチステートメントトランザクションでは、区切り文字のセミコロンはトランザクションセパレータとして機能しません。マルチステートメントトランザクションにおいてセミコロン区切りステートメントは、同じトランザクションの以前のステートメントによる更新を認識します。ただし、トランザクションが明示的にコミットされるまで、更新はコミットされません。トランザクションがロールバックされると、そのトランザクションで以前に評価されたステートメントによる更新は破棄されます。

次の図は、シングルステートメントトランザクションとマルチステートメントトランザクションでのステートメントとトランザクションの関係を比較したものです。



デフォルトのモデル: 自動コミットのシングルステートメントトランザクションを 1 つ含むプログラム。

デフォルトのモデル: 複数のシングルステートメントトランザクションを含むプログラム。

マルチステートメントトランザクション: 複数のマルチステートランザクションを含むプログラム。

3.6 トランザクションモード

このセクションでは、トランザクションモードに関して以下のような内容を取り上げます。

- トランザクションモードの概要
- Auto トランザクションモード
- クエリトランザクションモード
- 更新トランザクションモード

3.6.1 トランザクションモードの概要

トランザクションモードは、コミットモード（auto または explicit）とトランザクションタイプ（auto、update、または query）の概念を組み合わせたものです。トランザクションモード設定は、セッション全体で有効です。トランザクションモードは次の方法で制御できます。

- `xdmp:set-transaction-mode` XQuery 関数または `xdmp.setTransactionMode` JavaScript 関数を呼び出す。
- 非推奨：`xdmp:eval`（XQuery）、`xdmp.eval`（JavaScript）、または関連する関数の eval/invoke/spawn 関数で `transaction-mode` オプションを使用する。代わりに、`commit` および `update` オプションを使用してください。
- 非推奨：XQuery プロローグオプション `xdmp:transaction-mode` を使用する。代わりに、xdmp:commit および xdmp:update XQuery プロローグオプションを使用してください。

一般的には、トランザクションモードを設定するのではなく、具体的なコミットモードとトランザクションタイプの制御を使用してください。このような制御を使用すると、トランザクション設定を細かく制御できます。

例えば、次の表を使用して、`xdmp:transaction-mode` XQuery プロローグオプションを `xdmp:commit` および `xdmp:update` プロローグオプションにマッピングします。詳細については、「XQuery におけるトランザクションのタイプの制御」（35 ページ）を参照してください。

xdmp:transaction-mode の値	xdmp:commit および xdmp:update オプションの同等の設定
"auto"	declare option xdmp:commit "auto"; declare option xdmp:update "auto";
"update-auto-commit"	declare option xdmp:commit "auto"; declare option xdmp:update "true";
"update"	declare option xdmp:commit "explicit"; declare option xdmp:update "true";
"query"	declare option xdmp:commit "explicit"; declare option xdmp:update "false";

xdmp:commit および xdmp:update は、その宣言後に作成される次のトランザクションのみに影響し、セッション全体には影響しません。セッションレベルで設定を変更する必要がある場合は、xdmp:set-transaction-mode または xdmp.setTransactionMode を使用してください。

次の表を使用して、xdmp:eval および関連する eval/invoke/spawn 関数の commit および update オプションと、transaction-mode オプションをマッピングします。

transaction-mode オプションの値	commit および update オプションの同等の値
auto	commit: "auto" update: "auto"
update-auto-commit	commit: "auto" update: "true"
update	commit: "explicit" update: "true"
query	commit "explicit" update "false"

サーバーサイド JavaScript アプリケーションは、declareUpdate 関数を使用して、トランザクションモードが update-auto-commit または update である時期を示します。詳細については、「JavaScript におけるトランザクションのタイプの制御」（38 ページ）を参照してください。

マルチステートメントトランザクションを使用するには、トランザクションモードを query または update に明示的に設定する必要があります。これにより、コミットモードが暗黙で「explicit」に設定されます。読み取り専用トランザクションには query を使用してください。更新を実行する可能性があるトランザクションには update モードを使用してください。適切なモードを選択すると、サーバーがクエリを適切に最適化できます。詳細については、「マルチステートメントトランザクション（明示的コミット）」（46 ページ）を参照してください。

トランザクションモードは、トランザクションの作成時にのみ考慮されます。モードを変更しても、現在のトランザクションには影響しません。

トランザクションモードを明示的に設定した場合は、現在のセッションのみに影響します。xdmp:eval、xdmp.eval、または同様の関数で different-transaction 分離を指定して実行した関数、または xdmp:spawn で実行した関数は、呼びたし元のコンテキストからトランザクションモードを継承しません。「xdmp:eval/invoke とのインタラクション」（58 ページ）を参照してください。

3.6.2 Auto トランザクションモード

デフォルトのトランザクションモードは auto です。これは、「auto」コミットモードと「auto」トランザクションタイプの組み合わせと同等です。このモードでは、トランザクションはすべてシングルステートメントトランザクションです。「シングルステートメントトランザクション（自動コミット）」（45 ページ）を参照してください。

ほとんどの XQuery アプリケーションでは、auto トランザクションモードを使用してください。auto トランザクションモードを使用すると、サーバーは各ステートメントを独立して最適化でき、ファイルのロックを最小限にすることができます。ほとんどの場合に、パフォーマンスが向上し、デッドロックの可能性が減少します。

ほとんどのサーバーサイド JavaScript アプリケーションでは、更新を実行しないコードでは auto モードを使用し、更新を実行するコードでは update-auto-commit モードを使用してください。引数なしで declareUpdate を呼び出すと、update-auto-commit モードがアクティブになります。詳細については、「JavaScript におけるトランザクションのタイプの制御」（38 ページ）を参照してください。

auto トランザクションモードでは、次のようになります。

- トランザクションはすべてシングルステートメントトランザクションです。ステートメントごとに新しいトランザクションが作成されます。
- 評価前にステートメントの静的分析を行うことにより、作成したトランザクションを、更新モードとクエリモードのどちらで実行するかが判断されます。
- ステートメントに関連付けられたトランザクションは、ステートメントの実行が完了すると自動的にコミットされ、エラーが発生した場合は自動的にロールバックされます。

update-auto-commit は、トランザクションが常に更新トランザクションである点のみが異なります。

XQuery では、xdmp:set-transaction-mode または xdmp:transaction-mode プロローグオプションを使用してモードを明示的に auto に設定できますが、以前にモードを update または query に明示的に設定したことがある場合を除き、必須ではありません。

3.6.3 クエリトランザクションモード

クエリトランザクションモードは、explicit コミットモードとクエリトランザクションタイプの組み合わせと同等です。

XQuery では、クエリトランザクションモードが有効なのは、xdmp:set-transaction-mode または xdmp:transaction-mode プロローグオプションを使用してモードを明示的に設定した場合のみです。このモードで作成されるトランザクションは、常にマルチステートメントトランザクションです。「マルチステートメントトランザクション（明示的コミット）」（46 ページ）を参照してください。

マルチステートメントクエリトランザクションはサーバーサイド JavaScript からは作成できません。

クエリトランザクションモードでは、以下のようになります。

- トランザクションは、複数のステートメントにわたる場合もあります。
- トランザクションは読み取り専用とみなされるため、ロックが取得されません。トランザクション内のすべてのステートメントは、ポイントインタイムクエリとして実行され、トランザクション開始時のシステムタイムスタンプが使用されます。
- トランザクション内のすべてのステートメントは、クエリ（読み取り専用）ステートメントとして機能する必要があります。更新操作が試みられると、実行時にエラーが発生します。
- xdmp:commit を使用して明示的にコミットされないトランザクションは、セッションがタイムアウトするとロールバックします。ただし、コミット対象の更新がないため、ロールバックを識別できるのは、ステートメントの完了前に明示的なロールバックが発生した場合のみです。

更新ステートメントはマルチステートメントトランザクション内に出現できますが、実行時に更新の呼び出しが実際に行われないようにしてください。クエリモードで実行中のトランザクションが更新操作を試みると、XDMP-UPDATEFUNCTIONFROMQUERY が発生します。例えば、次のコードは、プログラムのロジックにより更新操作が実行されないようになっているため、例外が発生しません。

```
xquery version "1.0-ml";
declare option xdmp:transaction-mode "query";

if (fn:false())then
  (: XDMP-UPDATEFUNCTIONFROMQUERY only if this executes :)
  xdmp:document-insert("/docs/test.xml", <a/>)
else ();
xdmp:commit();
```

3.6.4 更新トランザクションモード

更新トランザクションモードは、explicit コミットモードと更新トランザクションタイプの組み合わせと同等です。

XQuery では、更新トランザクションモードが有効なのは、`xdmp:set-transaction-mode` または xdmp:transaction-mode プロローグオプションを使用してモードを明示的に設定した場合のみです。`update` モードで作成されるトランザクションは、常にマルチステートメントトランザクションです。「マルチステートメントトランザクション（明示的コミット）」（46 ページ）を参照してください。

サーバーサイド JavaScript では、`declareUpdate` の呼び出し時に `explicitCommit` を true に設定すると、トランザクションが更新モードになります。

更新トランザクションモードでは、次のようになります。

- トランザクションは、複数のステートメントにわたる場合もあります。
- トランザクションはデータベースを変更すると想定されるため、必要に応じて読み取り / 書き込みロックが取得されます。
- 更新トランザクション内のステートメントは、更新ステートメントまたはクエリステートメントです。
- `xdmp:commit` を使用して明示的にコミットされないトランザクションは、セッションがタイムアウトするとロールバックします。

更新トランザクションは、クエリステートメントと更新ステートメントの両方を含むことができますが、更新トランザクション内のクエリステートメントは引き続き、システムタイムスタンプを使用せずに読み取りロックを適用します。詳細については、「更新トランザクション：読み取り / 書き込みロック」（41 ページ）を参照してください。

3.7 xdmp:eval/invoke とのインタラクション

`xdmp:eval` および `xdmp:invoke` 関数ファミリを使用すると、トランザクションのコンテキストから別のトランザクションを開始できます。`xdmp:eval` XQuery 関数および `xdmp.eval` JavaScript 関数は、評価対象の文字列を送信します。`xdmp:invoke` XQuery 関数および `xdmp.invoke` JavaScript 関数は、格納されているモジュールを評価

します。eval および invoke のセマンティックは、これらの関数のオプションで制御でき、それによって、プログラムのトランザクショナルセマンティックがわずかに変更されることがあります。このセクションでは、これらの微細な点について説明します。このセクションは、次の部分から構成されます。

- xdmp:eval/invoke の isolation 分離オプション
- デッドロックの回避
- eval/invoke からの更新をトランザクションの後半で認識
- xdmp:eval/invoke でのマルチステートメントトランザクションの実行

3.7.1 xdmp:eval/invoke の isolation 分離オプション

`xdmp:eval` および `xdmp:invoke` XQuery 関数、およびそれぞれに対応する JavaScript 関数は、options ノードをオプションの第 3 パラメータとして使用します。`isolation` オプションは、eval/invoke 操作の結果としてのトランザクションの動作を決定します。次のいずれかの値にする必要があります。

- `same-statement`
- `different-transaction`

`same-statement` 分離では、eval または invoke によって実行されるコードは、呼び出し元ステートメントと同じトランザクション内で同じステートメントの一部として実行されます。`same-statement` 分離を使用して eval/invoke 操作で実行される更新は、呼び出し元ステートメントの後続の部分によって認識されません。ただし、マルチステートメントトランザクションを使用するときは、この更新が、同じトランザクション内の後続のステートメントで認識されます。

`same-statement` 分離においてクエリトランザクションから呼び出される eval/invoke で実行されるコードでは、更新操作を実行できないことがあります。クエリトランザクションはタイムスタンプに従って実行されるため、更新を実行するには、トランザクションの途中でタイムスタンプモードと読み取り / 書き込みロックを切り替える必要がありますが、これは許可されません。これを行うステートメントやトランザクションは、`XDMP-UPDATEFUNCTIONFROMQUERY` をスローします。

eval または invoke で `database` オプションを使用して、呼び出し元ステートメントのコンテキストのデータベースとは別のデータベースを指定しているときは、`same-statement` 分離を使用できないことがあります。eval/invoke コードで別のデータベースを使用する必要がある場合は、`different-transaction` 分離を使用してください。

isolation を different-transaction に設定すると、eval/invoke によって実行されるコードは、呼び出し元ステートメントとは別のセッションおよび別のトランザクションで実行されます。eval/invoke セッションおよびトランザクションは、呼び出し元トランザクションの残りの部分を続行する前に完了します。呼び出し元トランザクションが更新トランザクションである場合、different-transaction 分離を使用して eval/invoke 操作で実行されるコミット済み更新は、呼び出し元ステートメントの後続の部分で認識されます。ただし、different-transaction 分離（デフォルトの分離レベル）を使用する場合は、デッドロック状態に陥らないようにする必要があります（「デッドロックの回避」（61 ページ）を参照）。

次の表は、クエリステートメントと更新ステートメントで許可される分離オプションを示しています。

呼び出し元ステートメント	呼び出し先ステートメント（xdmp:eval、xdmp:invoke）			
	same-statement 分離		different-transaction 分離	
	クエリステートメント	更新ステートメント	クエリステートメント	更新ステートメント
クエリステートメント（タイムスタンプモード）	はい	更新が発生しない場合は、はい。更新が発生する場合は、例外がスローされます。	はい	はい
更新ステートメント（読み取り / 書き込みロックモード）	はい（「注」を参照）	はい	はい	はい（ロックされているドキュメントを更新すると、デッドロックが発生する可能性があります）

注： この表は、若干単純化されています。例えば、更新ステートメントが same-statement 分離を使用してクエリステートメントを呼び出す場合、この「クエリステートメント」は実際には、更新ステートメントの一部として実行されます（呼び出し元更新ステートメントと同じトランザクションの一部として実行されるため）。したがって、タイムスタンプモードではなく、読み取り / 書き込みロックを使用して実行されます。

3.7.2 デッドロックの回避

デッドロックとは、2 つのプロセスまたはスレッドがそれぞれ、他方のロックの解除を待機しており、ロックが解除されるまではどちらのプロセスも続行できない状態です。デッドロックは、データベース操作の標準的な部分です。サーバーは、検出したデッドロックに対処できます（例えば、いずれかのトランザクションを再試行したり、いずれかまたは両方のリクエストをキルしたりします）。

ただし、トランザクションがタイムアウトするまで、MarkLogic サーバーが待機以外に何もできないようなデッドロック状態もあります。`xdmp:eval` または `xdmp:invoke` ステートメントを呼び出す更新ステートメントを実行し、その eval/invoke が更新ステートメントである場合は、デッドロック条件が生じるおそれがあります。このようなデッドロックは、更新ステートメントのみで発生します。クエリステートメントではデッドロックが発生することはありません。

トランザクションがドキュメントで何らかのロックを適用し、そのトランザクションから呼び出される eval/invoke ステートメントが同じドキュメントの書き込みロックを適用しようとすると、デッドロック条件が発生します。デッドロック条件を解決するには、クエリをキャンセルするか、クエリがタイムアウトするまで待つ必要があります。

次の例に示すように、念のために `prevent-deadlocks` オプションを `true` に設定すると、デッドロックの発生を回避できます。

```
xquery version "1.0-ml";
(: the next line ensures this runs as an update
statement :)
declare option xdmp:update "true";
xdmp:eval("xdmp:node-replace(doc('/docs/test.xml')/a,
<b>goodbye</b>)",
          (),
          <options xmlns="xdmp:eval">
            <isolation>different-transaction</isolation>
            <prevent-deadlocks>true</prevent-deadlocks>
          </options>) ,
doc("/docs/test.xml")
```

その後、このステートメントは、次の例外をスローします。

```
XDMP-PREVENTDEADLOCKS: Processing an update from an update
  with different-transaction isolation could deadlock
```

この例では、このステートメントが、`xdmp:document-insert` 呼び出しによって更新ステートメントとして実行された結果、読み取り / 書き込みロックを使用するため、デッドロックの発生が実際に回避されます。2 行目で、読み取りロックが URI `/docs/test.xml` のドキュメントに設定されます。次に、`xdmp:eval` ステートメン

トが、同じドキュメントに書き込みロックを適用しようとしますが、読み取りロックが解放されるまで書き込みロックを適用できません。これにより、デッドロック条件が生じます。そこで、prevent-deadlocks オプションによってデッドロックの発生が中止されます。

prevent-deadlocks オプションを使用しない場合は、デフォルトが false（デッドロックを「許容」）になります。そのため、次のステートメントではデッドロックが発生します。

警告 このコードはデモンストレーションを目的としており、実行すると、デッドロックが発生します。デッドロックを解消するには、クエリをキャンセルするか、タイムアウトするまで待機する必要があります。

```
(: the next line ensures this runs as an update
statement :)
if ( 1 = 2) then ( xdmp:document-insert("foobar", <a/>) )
else (),
doc("/docs/test.xml"),
xdmp:eval("xdmp:node-replace(doc('/docs/test.xml')/a,
<b>goodbye</b>)",
          (),
          <options xmlns="xdmp:eval">
            <isolation>different-transaction</isolation>
          </options>) ,
doc("/docs/test.xml")
```

これがデッドロック条件であり、トランザクションがタイムアウトするか、手動でキャンセルされるか、または MarkLogic が再起動されるまで、デッドロックのままになります。上記の例の 2 行目にある doc("/docs/test.xml") の最初の呼び出しを削除すると、このステートメントによりデッドロックが発生しません。これは、/docs/test.xml の読み取りロックが、xdmp:eval ステートメントの完了まで呼び出されないためです。

3.7.3 eval/invoke からの更新をトランザクションの後半で認識

eval/invoke 操作内の更新ステートメントが、呼び出し元ステートメントですでに参照されているドキュメントを更新しようとしない（したがって、デッドロック条件が発生しない（「デッドロックの回避」（61 ページ）を参照））ことが確実である場合は、eval/invoke からの更新が呼び出し元トランザクションで認識されるようにステートメントを設定できます。これは、新しく更新されたドキュメントにアクセスするコードの前に eval/invoke ステートメントがあるトランザクションで非常に役立ちます。

注： eval/invoke 操作からの更新をステートメントの後半で認識するには、トランザクションが更新トランザクションである必要があります。トランザクションがクエリトランザクションである場合は、タイムスタンプモードで実行されるため、コミットされた eval/invoke 操作より前に存在するドキュメントバージョンが常に認識されます。

例えば、次の例では、doc("/docs/test.xml") がトランザクションの開始前に <a>hello を返します。

```
(: doc("/docs/test.xml") returns <a>hello</a> before
running this :)
(: the next line ensures this runs as an update
statement :)
if ( 1 = 2 ) then ( xdmp:document-insert("fake.xml", <a/>)
) else (),
xdmp:eval("xdmp:node-replace(doc('/docs/test.xml')/node(),
<b>goodbye</b>)", (),
          <options xmlns="xdmp:eval">
            <isolation>different-transaction</isolation>
            <prevent-deadlocks>false</prevent-deadlocks>
          </options>) ,
doc("/docs/test.xml")
```

この例の最後の行にある doc("/docs/test.xml") の呼び出しは、<a>goodbye を返します。これは、xdmp:eval 操作によって更新された新しいバージョンです。

同様の問題は、多くの場合、マルチステートメントトランザクションを使用して解決できます。マルチステートメントトランザクションでは、1 つのステートメントによる更新が、同一トランザクション内の後続のステートメントによって認識されます。上記の例をマルチステートメントトランザクションに書き換えるとします。トランザクションモードを update に設定すると、ステートメントを強制的に更新として分類する「fake」コードが不要になりますが、データベース内で更新が認識されるように xdmp:commit を呼び出すことが必要になります。

```
declare option xdmp:transaction-mode "update";

(: doc("/docs/test.xml") returns <a>hello</a> before
running this :)
xdmp:eval("xdmp:node-replace(doc('/docs/test.xml')/node(),
<b>goodbye</b>)", (),
          <options xmlns="xdmp:eval">
            <isolation>different-transaction</isolation>
            <prevent-deadlocks>false</prevent-deadlocks>
          </options>);
(: returns <a>goodbye</b> within this transaction :)
doc("/docs/test.xml"),
(: make updates visible in the database :)
xdmp:commit()
```

3.7.4 xdmp:eval/invoke でのマルチステートメントトランザクションの実行

different-transaction 分離を指定して xdmp:eval、xdmp:invoke、またはそれらに対応する JavaScript 関数を使用して、あるいは xdmp:spawn または xdmp.spawn 経由でクエリを実行すると、クエリを実行するための新しいトランザクションが作成され、新しく作成されたセッション内で実行されます。このことは、xdmp:eval または xdmp:invoke で評価されるマルチステートメントトランザクションに関する 2 つの重要な点を示唆しています。

- トランザクションモードは呼び出し元から継承されません。
- コミットされていない更新は、eval/invoke/spawn が実行されたクエリが完了すると、自動的にロールバックされます。

そのため、different-transaction 分離を指定して eval/invoke で評価されるコード、あるいは xdmp:spawn または xdmp.spawn で評価されるコードでマルチステートメントトランザクションを使用するときは、次のようにします。

- トランザクションをマルチステートメントトランザクションとして実行する必要がある場合、または XQuery xdmp:commit プロローグオプションまたは JavaScript declareUpdate 関数を使用して explicit コミットモードを使用する必要がある場合は、options ノードでコミットオプションを「explicit」に設定します。
- 更新を維持する必要がある場合は必ず、eval/invoke で実行されたマルチステートメントクエリ内部から xdmp:commit を呼び出します。

eval/invoke で実行されたクエリのプロローグでコミットモードを設定することは、commit を explicit に設定して options ノードを xdmp:eval/invoke に渡して設定することと同等です。options ノードを使用してモードを設定すると、eval/invoke で実行されたクエリを変更せずにコミットモードを設定できます。

different-transaction 分離を指定してマルチステートメントトランザクションを使用する例については、「例：マルチステートメントトランザクションと different-statement 分離」（69 ページ）を参照してください。

同様の考慮事項は、xdmp:spawn または xdmp.spawn を使用して評価されるマルチステートメントクエリにも当てはまります。

same-statement 分離で実行されるトランザクションは呼び出し元のコンテキストで実行されるため、同じトランザクションモードを使用し、呼び出し元トランザクションのコミットによるメリットを得られます。詳細な例については、「例：マルチステートメントトランザクションと same-statement 分離」（68 ページ）を参照してください。

3.8 非トランザクショナル副作用のある関数

更新トランザクションは、各種の更新ビルトイン関数を使用します。これらの関数は、トランザクションのコミット時にデータベース内のドキュメントを更新します。このような更新は、技術的には「副作用」として知られています。トランザクション内のステートメントが返す内容の範囲外で変更が発生するためです。更新ビルトイン関数（`xdmp:node-replace`、`xdmp:document-insert` など）による副作用は、本質的にトランザクショナルです。つまり、完全に完了するか、更新ステートメントの開始時の状態までロールバックされます。

一部の関数は、更新トランザクションとクエリトランザクションのどちらに呼び出された場合でも、呼び出されるとすぐに非同期で評価されます。このような関数には、呼び出し元のステートメント、または関数を含むトランザクションの範囲外の副作用（「非トランザクショナル」副作用）があります。次に、非トランザクショナル副作用が考えられる関数の例を示します。

- xdmp:spawn（XQuery）または xdmp.spawn（JavaScript）
- xdmp:http-get（XQuery）または xdmp.httpGet（JavaScript）
- xdmp:log（XQuery）または xdmp.log（JavaScript）

更新トランザクションを実行するモジュールを評価するときは、更新が失敗するか、再試行される可能性があります。これは正常なトランザクショナル動作であり、トランザクションが失敗したか、再試行された場合でも、データベースは一貫性のある状態を維持します。ただし、更新トランザクションが非トランザクショナル副作用のある関数を呼び出すと、呼び出し元の更新トランザクションが失敗してロールバックされた場合でも、その関数が評価されます。

更新トランザクションからのこれらの関数の呼び出しは、注意して行うか、避けてください。これは、評価が 1 回のみであること（または、トランザクションがロールバックされた場合は評価されないこと）が保証されていないためです。トランザクションで `xdmp:log` または `xdmp.log` を使用して情報をログに記録している場合は、再試行時にログが記録されることが適切な場合と不適切な場合があります（デッドロックが検出されたためにトランザクションが再試行される場合など）。これが意図した状況ではない場合でも、支障はありません。

ただし、その他の副作用によって、更新に問題が生じることがあります。例えば、このコンテキストで `xdmp:spawn` または `xdmp.spawn` を使用する場合、呼び出し元トランザクションが再試行されるとアクションが複数回生成されたり、トランザクションが失敗した場合でもアクションが生成されたりします。spawn 呼び出しは、呼び出されるとすぐに非同期で評価されます。同様に、`xdmp:http-get` または `xdmp.httpGet` を使用して更新トランザクションから Web サービスを呼び出している場合は、評価対象でなかった場合でも評価される可能性があります。

これらの関数を更新で使用する場合は、アプリケーションロジックで副作用を適切に処理する必要があります。通常、このような種類のユースケースは、トリガーや Content Processing Framework に適しています。詳細については、「トリガーを使用したアクションのスポーン」（384 ページ）および『*Content Processing Framework Guide*』マニュアルを参照してください。

3.9 Multi-Version Concurrency Control によるブロックの削減

「Multi-Version Concurrency Control」（マルチバージョン同時実行制御）アプリケーションサーバー設定パラメータを `nonblocking` に設定すると、クエリがデータベースを参照するタイミングが最適ではなくなりますが、トランザクションのブロックを最小限に抑えることができます。このオプションは、ロックなしのクエリのタイムスタンプが選択される方法を制御します。タイムスタンプがクエリに与える影響の詳細については、「クエリトランザクション：ポイントインタイム評価」（40 ページ）を参照してください。

非ブロックモードは、次の場合にアプリケーションで役立ちます。

- クエリのレイテンシが低いことが更新のレイテンシよりも重要である。
- アプリケーションが XA トランザクションに関与している。XA トランザクションには、複数の参加者および非 MarkLogic サーバーリソースが関与する場合があるため、通常よりも時間がかかることがあります。
- アプリケーションがレプリカデータベースにアクセスするため、マスターに大幅な遅れが発生することが想定される。マスターがしばらく到達不可能になる場合など。

デフォルトの Multi-Version Concurrency Control は `contemporaneous` です。このモードでは、MarkLogic サーバーは、何らかのトランザクションがコミットされたことがわかっている最新のタイムスタンプを選択します。そのタイムスタンプでは他のトランザクションが完全にはコミットされていない場合でも、このタイムスタンプが選択されます。クエリは、同時発生したトランザクションが完全にコミットされるまで待機する間、ブロックできますが、最もタイミングのよい結果を認識します。ブロック時間は、同時発生した最遅トランザクションによって決まります。

`nonblocking` モードでは、サーバーは、すべてのトランザクションがコミットされたことがわかっている最新のタイムスタンプを選択します。少し後のタイムスタンプで別のトランザクションがコミットされている場合でも、最新のタイムスタンプが選択されます。このモードでは、クエリは、同時発生したトランザクションを待機する間、ブロックしませんが、最新の結果を認識しない可能性があります。

同一のデータベースに対して、異なる Multi-Version Concurrency Control 設定でアプリケーションサーバーを実行できます。

3.10 トランザクションの管理

MarkLogic サーバー XQuery API には、トランザクションのデバッグ、モニタリング、および管理に役立つビルトイン関数が含まれています。

実行中のトランザクションに関する情報を取得するには、xdmp:host-status を使用します。ステータス情報には <transactions> 要素が含まれます。この要素には、ホストで実行中のすべてのトランザクションに関する詳細情報が格納されます。例：

```
<transactions xmlns="http://marklogic.com/xdmp/
status/host">
  <transaction>
    <transaction-id>10030469206159559155</transaction-id>
    <host-id>8714831694278508064</host-id>
    <server-id>42127728720393659 46</server-id>
    <name/>
    <mode>query</mode>
    <timestamp>11104</timestamp>
    <state>active</state>
    <database>108286083390327344 79</database>
    <canceled>false</canceled>
    <start-time>2011-05-03T09:14:11-07:00</start-time>
    <time-limit>600</time-limit>
    <max-time-limit>3600</max-time-limit>
    <user>153014186478447595 56</user>
    <admin>true</admin>
  </transaction>
  ...
</transactions>
```

クラスタ化インストールでは、トランザクションがリモートホストで実行されることがあります。リモートトランザクションが正常に終了しない場合は、xdmp:transaction-commit や xdmp:transaction-rollback を使用してリモートでコミットまたはロールバックできます。これらの関数は、xdmp:host がホスト ID パラメータとして渡されるときに xdmp:commit や xdmp:rollback を呼び出すことと同等です。管理画面の［Host Status］ページからトランザクションをロールバックすることもできます。詳細については、『*Administrator's Guide*』の Rolling Back a Transaction を参照してください。

xdmp:transaction-commit の呼び出しはすぐに復帰しますが、コミットは、ターゲットトランザクション内で実行中のステートメントが正常に完了した後にのみ発生します。xdmp:transaction-rollback を呼び出すと、ターゲットトランザクション内で実行中のステートメントが直ちに中断され、トランザクションが終了します。

これら機能の使用例については、「例：xdmp:host-status を使用したトランザクションレポート生成」（71 ページ）を参照してください。ビルトインの詳細については、『*XQuery & XSLT API Reference*』を参照してください。

3.11 トランザクションの例

このセクションでは、以下の各例について説明します。

- 例：マルチステートメントトランザクションと same-statement 分離
- 例：マルチステートメントトランザクションと different-statement 分離

実際の時刻に関連してシステムタイムスタンプを追跡する例については、「システムタイムスタンプを追跡する」（153 ページ）を参照してください。

3.11.1 例：マルチステートメントトランザクションと same-statement 分離

次の例では、マルチステートメントトランザクションと `same-statement` 分離のインタラクションについて示します（「xdmp:eval/invoke とのインタラクション」（58 ページ）を参照）。

このサンプルの目標は、`xdmp:eval` を使用してデータベースでドキュメントを挿入し、その結果を呼び出しモジュールで調べ、修正することです。挿入したドキュメントは、呼び出しモジュールから直ちに認識される必要がありますが、トランザクション完了まではモジュール外部から認識されないようにします。

```
xquery version "1.0-ml";
declare option xdmp:transaction-mode "update";

(: insert a document in the database :)
let $query :=
  'xquery version "1.0-ml";
   xdmp:document-insert("/examples/mst.xml", <myData/>)
  '
return xdmp:eval(
  $query, (),
  <options xmlns="xdmp:eval">
    <isolation>same-statement</isolation>
  </options>);

(: demonstrate that it is visible to this transaction :)
if (fn:empty(fn:doc("/examples/mst.xml")//myData))
then ("NOT VISIBLE")
else ("VISIBLE");

(: modify the contents before making it visible in the
database :)
```

```
xdmp:node-insert-child(doc('/examples/mst.xml')/myData,
<child/>),
xdmp:commit()

(: result: VISIBLE :)
```

同じ操作（ドキュメントを挿入して変更した後、データベースで認識されるようにする）を、デフォルトのトランザクションモデルを使用して容易に実行することはできません。モジュールが、ドキュメントの挿入および子の挿入を同一シングルステートメントトランザクション内で試みると、XDMP-CONFLICTINGUPDATES エラーが発生します。この 2 つの操作を個別のシングルステートメントトランザクション内で実行すると、挿入したドキュメントが、子ノードを挿入するより前に、直ちにデータベース内で認識されます。プリコミットトリガーを使用して子の挿入を実行しようとすると、トリガーストームが生じます。「トリガーの無限ループ（トリガーストーム）の回避」（394 ページ）を参照してください。

eval は same-statement 分離を使用するため、eval で実行されるクエリは、呼び出しモジュールのマルチステートメント更新トランザクションの一部として実行されます。トランザクションモードは、別のコンテキストで作成されたトランザクションからは継承されないため、different-transaction 分離を使用すると、eval で実行されるクエリがシングルステートメントトランザクションとして評価され、ドキュメントが他のトランザクションから直ちに認識されるようになります。

モジュールによって実行される更新を維持するには、xdmp:commit の呼び出しが必要です。xdmp:commit を省略した場合は、評価がモジュールの最後に達すると、すべての更新が失われます。この例では、eval で実行されるクエリ内ではなく、呼び出しモジュール内でコミットが発生する必要があります。xdmp:commit が eval で実行されるクエリ内で発生する場合、トランザクションは、xdmp:eval 呼び出しが含まれるステートメントが完了すると完了し、子ノードの挿入前にドキュメントがデータベース内で認識されます。

3.11.2 例：マルチステートメントトランザクションと different-statement 分離

次の例では、different-transaction 分離がマルチステートメントトランザクションのトランザクションモードとインタラクションする方法について示します。同様のインタラクションが、xdmp:spawn を使用して実行されるクエリにも当てはまります。詳細な背景については、「トランザクションモード」（54 ページ）および「xdmp:eval/invoke とのインタラクション」（58 ページ）を参照してください。

この例では、xdmp:eval を使用して、ドキュメントを挿入する新しいトランザクションを作成します。このドキュメントには、xdmp:transaction を使用して現在のトランザクション ID が格納されます。呼び出し元クエリは、自身のトランザクション ID、および eval で実行されるクエリのトランザクション ID を出力します。

```
xquery version "1.0-ml";

(: init to clean state; runs as single-statement txn :)
xdmp:document-delete("/docs/mst.xml");

(: switch to multi-statement transactions :)
declare option xdmp:transaction-mode "query";

let $sub-query :=
  'xquery version "1.0-ml";
   declare option xdmp:transaction-mode "update";   (: 1 :)

   xdmp:document-insert("/docs/mst.xml", <myData/>);

   xdmp:node-insert-child(
     fn:doc("/docs/mst.xml")/myData,
     <child>{xdmp:transaction()}</child>
   );
   xdmp:commit()                                    (: 2 :)
  '
return xdmp:eval($sub-query, (),
  <options xmlns="xdmp:eval">
    <isolation>different-transaction</isolation>
  </options>);

(: commit to end this transaction and get a new system
 : timestamp so the updates by the eval'd query are visible.:)
xdmp:commit();                                       (: 3 :)

(: print out my transaction id and the eval'd query
transaction id :)
fn:concat("My txn id: 「, xdmp:transaction())       (: 4 :)
fn:concat("Subquery txn id: ",
fn:doc("/docs/mst.xml")//child)
```

ステートメント `(: 1 :)` でトランザクションモードを設定する必要があります。これは、`different-transaction` 分離により、eval で実行されるクエリが、自身のセッション内で実行される新しいトランザクションとなるためです。したがって、このトランザクションは、呼び出し元コンテキストのトランザクションモードを継承しません。eval で実行されるクエリで xdmp:transaction-mode を省略すると、そのクエリはデフォルトの `auto` トランザクションモードで実行されます。

同様に、xdmp:commit の呼び出し（ステートメント（: 2 :））も、different-transaction 分離により必要になります。新しいトランザクションとそれを含むセッションは、eval で実行されるクエリの最後で終了します。トランザクションまたはそれを含むセッションがコミットせずに終了すると、変更は暗黙でロールバックされます。

xdmp:commit の呼び出し（ステートメント（: 3 :））は、xdmp:eval を呼び出したマルチステートメントクエリトランザクションを終了し、新しいトランザクションを開始して結果を出力します。これにより、ステートメント（: 4 :）の最後のトランザクションは、新しいタイムスタンプで実行されるため、xdmp:eval によって挿入されるドキュメントを認識します。システムタイムスタンプはトランザクションの開始時に固定されているため、このコミットを省略すると、挿入したドキュメントが認識されません。詳細については、「クエリトランザクション：ポイントインタイム評価」（40 ページ）を参照してください。

xdmp:eval を呼び出すクエリがクエリトランザクションではなく、更新トランザクションである場合は、xdmp:commit（ステートメント（: 3 :））を省略できます。更新トランザクションは、ドキュメントに最初にアクセスした時点で最新バージョンのドキュメントを認識します。この例のドキュメントは、xdmp:eval 呼び出しの後まではアクセスされないため、この例を更新トランザクションとして実行すると、eval で実行されるクエリからの更新が認識されます。詳細については、「更新トランザクション：読み取り / 書き込みロック」（41 ページ）を参照してください。

3.11.3 例：xdmp:host-status を使用したトランザクションレポート生成

ホストで実行されているトランザクションのリストを生成するには、ビルトイン xdmp:host-status を使用します。管理画面の［Host Status］ページには、同様のリストが表示されています。

この例は、ローカルホストにおけるすべてのトランザクションの継続時間に関するシンプルな HTML レポートを生成します。

```
xquery version "1.0-ml";

declare namespace html = "http://www.w3.org/1999/xhtml";
declare namespace hs="http://marklogic.com/xdmp/
status/host";

<html>
  <body>
    <h2>Running Transaction Report for {xdmp:host-name()}</h2>
    <table border="1" cellpadding="5">
    <tr>
      <th>Transaction Id</th>
      <th>Database</th><th>State</th>
```

```
        <th>Duration</th>
      </tr>
      {
        let $txns:= xdmp:host-status(xdmp:host())
        //hs:transaction
        let $now := fn:current-dateTime()
        for $t in $txns
        return
          <tr>
            <td>{$t/hs:transaction-id}</td>
            <td>{xdmp:database-name($t/hs:database-id)}</td>
            <td>{$t/hs:transaction-state}</td>
            <td>{$now - $t/hs:start-time}</td>
          </tr>
      }
      </table>
    </body>
  </html>
```

上記のクエリを Query Console に貼り付け、HTML 出力として実行すると、次のようなレポートが生成されます。

Running Transaction Report for mumble.marklogic.com

Transaction Id	Database	State	Duration
6335215186646946533	Documents	active	PT1M15.666S
11159175550762420347	App-Services	active	PT0.666S
17156013413075008219	Documents	active	PT0.666S

各トランザクションに関する多くの詳細情報は、`xdmp:host-status` レポートで利用できます。詳細については、『*XQuery & XSLT API Reference*』の `xdmp:host-status` を参照してください。

レポート内の最初のトランザクションでデッドロックが発生することが想定される場合は、`xdmp:transaction-rollback` を呼び出してトランザクション ID を指定することで、手動でキャンセルできます。以下に例を示します。

```
xquery version "1.0-ml";
xdmp:transaction-rollback(xdmp:host(), 6335215186646946533)
```

管理画面の［Host Status］ページからトランザクションをロールバックすることもできます。

4.0 バイナリドキュメントの使用

このセクションでは、MarkLogic サーバーにおけるバイナリドキュメントの設定および管理について説明します。バイナリドキュメントは、テキストや XML コンテンツよりも非常に大きくなることが多いため、特別な配慮が必要です。次のトピックから構成されます。

- 用語
- バイナリドキュメントの読み込み
- バイナリコンテンツ向けの MarkLogic サーバーの設定
- バイナリドキュメントを使用するアプリケーションの開発
- バイナリドキュメントの操作に役立つビルトイン

4.1 用語

次の表は、MarkLogic サーバーにおけるバイナリドキュメントのサポートに関連して使用される用語とその説明です。

用語	定義
スモールバイナリドキュメント	コンテンツがサーバーで管理され、サイズがラージサイズしきい値を超えないバイナリドキュメント。
ラージバイナリドキュメント	コンテンツがサーバーで管理され、サイズがラージサイズしきい値を超えるバイナリドキュメント。
外部バイナリドキュメント	コンテンツがサーバーで管理されないバイナリドキュメント。
ラージサイズしきい値	スモールバイナリドキュメントのサイズの上限を定義するデータベース設定。このしきい値より大きいバイナリドキュメントは、自動的にラージバイナリドキュメントとして分類されます。
ラージデータディレクトリ	ラージバイナリドキュメントのコンテンツが格納される、フォレストごとの領域。
静的コンテンツ	アプリケーションサーバーの modules データベースに格納されるコンテンツ。MarkLogic サーバーは、静的コンテンツの HTTP レンジリクエスト（部分的 GET）に直接応答します。「HTTP レンジリクエストを含むバイナリコンテンツのダウンロード」（82 ページ）を参照してください。

用語	定義
動的コンテンツ	動的コンテンツとは、アプリケーションによって生成されるコンテンツ（XQuery モジュールが返す結果など）です。MarkLogic サーバーは、動的コンテンツの HTTP レンジリクエスト（部分的 GET リクエスト）に直接応答しません。「HTTP レンジリクエストを含むバイナリコンテンツのダウンロード」（82 ページ）を参照してください。

4.2 バイナリドキュメントの読み込み

スモール / ラージバイナリドキュメントの MarkLogic データベースへの読み込みには、特別な処理は不要です。ドキュメント形式を明示的に設定することも可能です。『*Loading Content Into MarkLogic Server Guide*』の Choosing a Binary Format を参照してください。

外部バイナリは MarkLogic によって管理されないため、その読み込みには特別な処理が必要です。詳細については、Loading Binary Documents を参照してください。

4.3 バイナリコンテンツ向けの MarkLogic サーバーの設定

このセクションでは、バイナリドキュメントの MarkLogic サーバー設定および管理について説明します。

- ラージサイズしきい値の設定
- バイナリコンテンツのサイズ設定と拡張性
- バイナリコンテンツの位置の選択
- フォレスト内にあるラージバイナリデータの合計サイズのモニタリング
- 孤立バイナリの検出と削除

4.3.1 ラージサイズしきい値の設定

`large size threshold` データベース設定により、スモールバイナリの最大サイズ（キロバイト）が定義されます。このしきい値より大きいすべてのバイナリドキュメントは「ラージ」であり、ラージデータディレクトリに格納されます。Choosing a Binary Format を参照してください。このしきい値は外部バイナリには影響しません。

例えば、しきい値が 1024 の場合、サイズしきい値は 1MB に設定されます。1MB を超える任意の（管理対象）バイナリドキュメントは、自動的にラージバイナリオブジェクトとして扱われます。

64 ビットマシンで利用できるしきい値の範囲は 32KB ～ 512MB です。

ラージサイズしきい値の選択時には、データ特性、アプリケーションのアクセスパターン、基盤となるハードウェアやオペレーティングシステムなど、多くの要因を考慮する必要があります。サムネイルや顔写真など、サイズが小さく、頻繁にアクセスされるバイナリコンテンツは、アクセス効率を高めるために「スモール」に分類され、ムービーや音楽など、アプリケーションからストリーミングされる可能性のある、大きなサイズのドキュメントは、メモリ使用効率を高めるために「ラージ」に分類されるようにしきい値を設定することが理想的です。

しきい値は、管理画面から設定することも、Admin API 関数を呼び出して設定することもできます。管理画面からしきい値を設定するには、データベース設定ページの [large size threshold] 設定を使用します。プログラムでしきい値を設定するには、XQuery ビルトイン admin:database-set-large-size-threshold を使用します。

```
xquery version "1.0-ml";

import module namespace admin = "http://marklogic.com/xdmp/
admin"
      at "/MarkLogic/admin.xqy";

let $config := admin:get-configuration()
return
  admin:save-configuration(
    admin:database-set-large-size-threshold(
        $config, xdmp:database("myDatabase"),2048)
```

しきい値が変更されると、新しい設定に適合するように、再インデックス付けプロセスによってバイナリドキュメントがラージデータディレクトリの中または外に移動されます。

4.3.2 バイナリコンテンツのサイズ設定と拡張性

このセクションには、次の内容が含まれます。

- インメモリツリーサイズの決定
- E ノード圧縮ツリーキャッシュサイズへの外部バイナリの効果
- フォレストのスケーリングに関する考慮事項

サイズ設定と拡張性の詳細については、『*Scalability, Availability, and Failover Guide*』および『*Query Performance and Tuning Guide*』を参照してください。

4.3.2.1 インメモリツリーサイズの決定

in memory tree size データベース設定は、large size threshold 設定と、データベースに読み込もうとする最大の非バイナリドキュメントのいずれか大きい方の値より、少なくとも 1 ～ 2MB 大きい値にします。つまり、以下の値より 1 ～ 2MB 大きい値にします。

```
max(large-size-threshold, largest-expected-non-
binary-document)
```

「バイナリコンテンツの位置の選択」（77 ページ）で説明するように、スモールバイナリドキュメントの最大サイズは、64 ビットシステムで 512MB です。ラージバイナリならびに外部バイナリの場合、ドキュメントの最大サイズは OS がサポートするファイルの最大サイズとなります。

in memory tree size 設定を変更するには、管理画面のデータベース設定ページ、または『*XQuery and XSLT Reference Guide*』の admin:database-set-in-memory-limit を参照してください。

4.3.2.2 E ノード圧縮ツリーキャッシュサイズへの外部バイナリの効果

アプリケーションが外部バイナリドキュメントを多用する場合は、compressed tree cache size グループ設定の値の増加が必要になることがあります。

スモールバイナリがキャッシュされる場合、ドキュメント全体がメモリにキャッシュされる。ラージバイナリや外部バイナリがキャッシュされる場合、必要に応じてコンテンツが一塊として圧縮ツリーキャッシュ内に取り込まれます。

この一塊のラージバイナリは、そのフラグメントやドキュメントを含む D ノードの圧縮ツリーキャッシュに取り込まれます。この一塊の外部バイナリは、クエリのアクセスを評価する E ノードの圧縮ツリーキャッシュに取り込まれます。そのため、アプリケーションが外部バイナリドキュメントを多用する場合は、E ノードの圧縮ツリーキャッシュサイズの増加が必要になることがあります。

compressed tree cache size を変更するには、管理画面の［Groups］設定ページ、または『*XQuery and XSLT Reference Guide*』の admin:group-set-compressed-tree-cache-size を参照してください。

4.3.2.3 フォレストのスケーリングに関する考慮事項

フォレストのスケーリングのガイドラインを考慮する際は、フラグメント数の見積もりに、すべてのタイプのバイナリドキュメントを含めます。ラージバイナリおよび外部バイナリは、アクセス時にメモリに完全にはキャッシュされないため、メモリ要件の値が低くなります。ラージバイナリおよび外部バイナリはマージ時にコピーされないため、ラージバイナリおよび外部バイナリのコンテンツサイズを除外して最大フォレストサイズを計算できます。

サイズ設定と拡張性の詳細については、『*Scalability, Availability, and Failover Guide*』の Scalability Considerations in MarkLogic Server を参照してください。

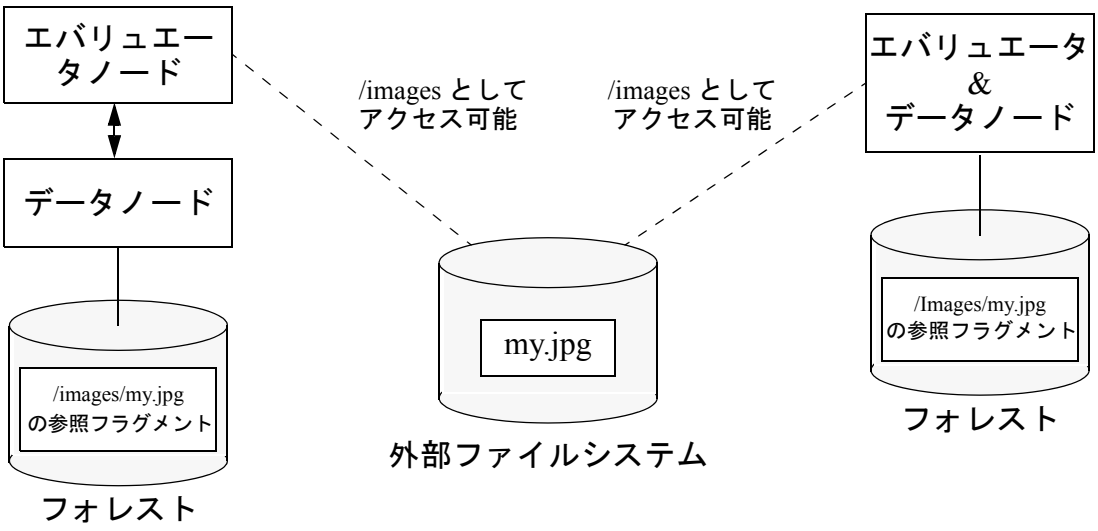
4.3.3 バイナリコンテンツの位置の選択

各フォレストにはラージデータディレクトリがあります。ここにフォレスト内のすべてのラージバイナリドキュメントのバイナリコンテンツが格納されます。ラージデータディレクトリの物理的な位置のデフォルトは、フォレスト内です。位置はフォレストの作成時に設定可能です。この柔軟性により、スモールならびにラージバイナリドキュメントを他のハードウェアに扱わせることができます。ラージデータディレクトリには、フォレストを含むサーバーインスタンスからアクセス可能である必要があります。任意の場所をラージデータディレクトリに指定するには、`admin:forest-create` の `$large-data-directory` パラメータ、または管理画面の `large data directory` フォレスト設定を使用します。

外部バイナリドキュメントに関連付けられている外部ファイルは、そのドキュメントが含まれるフォレストの外部に存在する必要があります。外部ファイルは、ドキュメントを操作するクエリを評価するサーバーインスタンスからアクセス可能である必要があります。つまり、`external-binary` ノードの作成時に使用される外部ファイルパスは、ドキュメントに対してクエリを実行するあらゆるサーバーインスタンス上で解決可能である必要があります。

外部バイナリファイルは、外部バイナリドキュメントに対してクエリを実行する任意の E ノードから同じパスでアクセスできる限り、ネットワーク共有ファイルシステム上に配置することでクラスタ全体で共有できます。関連付けられた `external-binary` ノードが含まれている参照フラグメントは、外部ストレージにアクセスできないリモート D ノード上に配置できます。

以下の図は、ホスト設定の異なるクラスタでの外部バイナリコンテンツ共有の例を示しています。左側では、エバリュエータノード（E ノード）とデータノード（D ノード）が別のホストです。右側では、同一ホストが、エバリュエータノードとデータノードの両方として機能しています。どちらの設定のデータベースにも、`/images/my.jpg` を参照している外部バイナリドキュメントが含まれています。JPEG コンテンツは、共有外部ストレージ上に格納され、データベース内の外部バイナリドキュメントに格納された外部ファイルパスを介してエバリュエータノードからアクセスできます。



4.3.4 フォレスト内にあるラージバイナリデータの合計サイズのモニタリング

フォレスト内でラージバイナリドキュメントが消費しているディスク領域を確認するには、`xdmp:forest-status` または管理画面を使用します。サイズは、MB 単位で表示されます。MarkLogic サーバーのモニタリング機能の詳細については、『*Monitoring MarkLogic Guide*』を参照してください。

管理画面を使用してラージデータディレクトリのサイズを確認するには、次の手順に従います。

1. ブラウザで管理画面を開きます。例：http:yourhost:8001。
2. 左側のツリーメニューで、[Forests] をクリックします。フォレストの概要が表示されます。
3. フォレスト設定ページに表示するフォレストの名前をクリックします。
4. 上部にある［Status］タブをクリックして、フォレストステータスページを表示します。
5. [Large Data Size] のステータスを確認します。これは、ラージデータディレクトリのコンテンツの合計サイズを示しています。

次の例では、xdmp:forest-status を使用してラージデータディレクトリのサイズを取得します。

```
xquery version "1.0-ml";
declare namespace fs = "http://marklogic.com/xdmp/
  status/forest";
fn:data(
  xdmp:forest-status(
    xdmp:forest("samples-1"))/fs:large-data-size)
```

4.3.5 孤立バイナリの検出と削除

ラージバイナリコンテンツおよび外部バイナリコンテンツでは、データベース内のドキュメントと関連付けられなくなった孤立バイナリデータを検出して削除する、特別な処理が必要になることがあります。このセクションでは、孤立バイナリコンテンツの管理に関する次のトピックについて説明します。

- 孤立ラージバイナリコンテンツの検出と削除
- 孤立外部バイナリコンテンツの検出と削除

4.3.5.1 孤立ラージバイナリコンテンツの検出と削除

『*Loading Content Into MarkLogic Server Guide*』の Choosing a Binary Format で説明するように、ラージバイナリドキュメントのバイナリコンテンツはラージデータディレクトリに格納されます。

通常、該当するフォレストにバイナリコンテンツデータへの参照が含まれなくなると、サーバーによってそのバイナリコンテンツが削除されます。ただし、バイナリドキュメントの挿入中にフェイルオーバーが発生した場合など、状況によっては、ラージデータディレクトリ内にコンテンツが残されてしまうことがあります。ラージデータディレクトリ内に残されてしまったコンテンツのうち、対応するデータベース参照フラグメントのないものが孤立バイナリです。

データにラージバイナリドキュメントが含まれる場合は、孤立バイナリを定期的に確認して削除してください。このクリーンアップを実行するには、xdmp:get-orphaned-binaries および xdmp:remove-orphaned-binary を使用します。例：

```
xquery version "1.0-ml";

for $fid in xdmp:forests()
  for $orphan in xdmp:get-orphaned-binaries($fid)
  return xdmp:remove-orphaned-binary($fid, $orphan)
```

4.3.5.2 孤立外部バイナリコンテンツの検出と削除

外部バイナリドキュメントに関連付けられている外部ファイルは MarkLogic サーバーによって管理されないため、存在しない外部ファイルに関連付けられている可能性があります。例えば、外部ファイルが社外の組織によって削除される可能性があります。XQuery API には、データベース内のそのようなドキュメントを確認して削除するのに役立つビルトインが用意されています。

例えば、外部バイナリファイル `/external/path/sample.jpg` に関連付けられているすべての外部バイナリドキュメントを削除するには、`xdmp:external-binary-path` を使用します。

```
xquery version "1.0-ml";
for $doc in fn:collection()/binary()
where xdmp:external-binary-path($doc) = "/external/path/
  sample.jpg"
return xdmp:document-delete(xdmp:node-uri($doc))
```

存在しない外部ファイルに関連付けられている外部バイナリドキュメントを識別するには、`xdmp:filesystem-file-exists` を使用します。ただし、`xdmp:filesystem-file-exists` は、基盤となるファイルシステムをクエリするため、比較的コストのかかる操作です。次の例では、外部ファイルのない外部バイナリドキュメントのドキュメント URI のリストを生成します。

```
xquery version "1.0-ml";
for $doc in fn:collection()/binary()
where xdmp:binary-is-external($doc)
return
  if (xdmp:filesystem-file-exists(xdmp:external-binary-path
    ($doc)))
  then xdmp:node-uri($doc)
  else ()
```

4.4 バイナリドキュメントを使用するアプリケーションの開発

このセクションでは、バイナリコンテンツを操作するアプリケーションを作成する開発者にとって役立つ、次の内容について説明します。

- プロパティを使用した、バイナリドキュメントへのメタデータの追加
- HTTP レンジリクエストを含むバイナリコンテンツのダウンロード
- バイナリメール添付ファイルの作成

4.4.1 プロパティを使用した、バイナリドキュメントへのメタデータの追加

スモール / ラージ / 外部バイナリドキュメントには、プロパティを使用して、メタデータによる注釈を付けることができます。データベース内のドキュメントには、それに関連するプロパティドキュメントが存在します。これは追加の XML データを格納するためです。バイナリデータの場合と違って、プロパティドキュメントは要素のインデックス付けに関係しています。プロパティの使用の詳細については、「プロパティドキュメントとディレクトリ」（128 ページ）を参照してください。

MarkLogic サーバーサーバーには、バイナリドキュメントへのメタデータ追加に役立つ XQuery ビルトイン `xdmp:document-filter` および JavaScript メソッド `xdmp.documentFilter` が用意されています。これらの関数は、バイナリドキュメントからテキストやメタデータをノードとして抽出します。このノードの子要素のそれぞれが、1 つのメタデータを表します。この結果は、ドキュメントのプロパティとして使用できます。抽出されたテキストには、書式や構造がほとんどないため、その最適な利用法は、検索、分類、その他のテキスト処理です。

例えば、次のコードは、`xdmp:document-filter` によって Microsoft Word ドキュメントから抽出されたメタデータに対応するプロパティを作成します。

```
xquery version "1.0-ml";
let $the-document := "/samples/sample.docx"
return xdmp:document-set-properties(
  $the-document,
  for $meta in xdmp:document-filter(fn:doc($the-document))
  //*:meta
  return element {$meta/@name} {fn:string($meta/@content)}
)
```

結果として得られるプロパティドキュメントには、Author、AppName、Creation_Date など、`xdmp:document-filter` から抽出されたプロパティが含まれます。

```
<prop:properties xmlns:prop="http://marklogic.com/xdmp/
property">
  <content-type>application/msword</content-type>
  <filter-capabilities>text subfiles HD-HTML
  </filter-capabilities>
  <AppName>Microsoft Office Word</AppName>
  <Author>MarkLogic</Author>
  <Company>Mark Logic Corporation</Company>
  <Creation_Date>2011-09-05T16:21:00Z</Creation_Date>
  <Description>This is my comment.</Description>
  <Last_Saved_Date>2011-09-05T16:22:00Z</Last_Saved_Date>
  <Line_Count>1</Line_Count>
  <Paragraphs_Count>1</Paragraphs_Count>
```

```
    <Revision>2</Revision>
    <Subject>Creating binary doc props</Subject>
    <Template>Normal.dotm</Template>
    <Typist>MarkLogician</Typist>
    <Word_Count>7</Word_Count>
    <isys>SubType: Word 2007</isys>
    <size>10047</size>
    <prop:last-modified>2011-09-05T09:47:10-07:00
    </prop:last-modified>
  </prop:properties>
```

4.4.2 HTTP レンジリクエストを含むバイナリコンテンツのダウンロード

HTTP アプリケーションでは、レンジリクエスト（「部分的 GET」とも呼ばれる）を使用して、ラージデータ（ビデオなど）を処理することがよくあります。MarkLogic サーバーは、「静的バイナリコンテンツ」の HTTP レンジリクエストを直接サポートします。静的バイナリコンテンツとは、アプリケーションサーバーの modules データベースに格納されたバイナリコンテンツです。「動的バイナリコンテンツ」のレンジリクエストは直接サポートされていませんが、そのようなリクエストを処理するアプリケーションコードを記述できます。動的バイナリコンテンツは、アプリケーションコードによって生成されるあらゆるコンテンツです。

このセクションでは、レンジリクエストに応答したバイナリコンテンツの処理に関する次の内容について説明します。

- 静的バイナリコンテンツによるレンジリクエストへの応答
- 動的バイナリコンテンツによるレンジリクエストへの応答

4.4.2.1 静的バイナリコンテンツによるレンジリクエストへの応答

HTTP アプリケーションサーバーは、modules データベース内のバイナリドキュメントに対するレンジリクエストを受信すると、直接応答します。追加のアプリケーションコードは必要ありません。modules データベース内のコンテンツは「静的」コンテンツとみなされます。任意のデータベースを modules データベースとして使用するようにアプリケーションサーバーを設定できます。これにより、MarkLogic は、静的バイナリコンテンツのレンジリクエストに直接応答できるようになります。

例えば、URI が「/images/really_big.jpg」であるラージバイナリドキュメントがデータベースに存在するとします。このデータベースを modules データベースとして使用する HTTP アプリケーションサーバーをポート 8010 で作成します。次の形式の GET リクエストをポート 8010 に送信すると、バイナリドキュメントが直接フェッチされます。

```
GET http://host:8010/images/really_big.jpg
```

リクエストにレンジを含めている場合は、データベースからドキュメントを段階的にストリーミングできます。例：

```
GET http://host:8010/images/really_big.jpg
Range: bytes=0-499
```

MarkLogic は、ドキュメント `/images/really_big.jpg` の最初の 500 バイトを、`Partial Content` 応答と 206（コンテンツの一部）ステータスで返します。例えば、次のようになります（簡潔にするために一部のヘッダを省略)。

```
HTTP/1.0 206 Partial Content
Accept-Ranges: bytes
Content-Length: 500
Content-Range: bytes 0-499/3980
Content-Type: image/jpeg

[first 500 bytes of /images/really_big.jpg]
```

重複していない複数のレンジがレンジリクエストに含まれる場合、アプリケーションサーバーは、206 と、メディアタイプ「multipart/byteranges」のマルチパートメッセージ本文で応答します。

レンジリクエストが満たされない場合、アプリケーションサーバーは 416 ステータス（満たすことができないレンジリクエスト）で応答します。

静的コンテンツでは、次のリクエストタイプが直接サポートされます。

- 単一のレンジリクエスト
- 複数のレンジリクエスト
- HTTP-date がある、If-Range リクエスト

エンティティタグ付きの If-Range リクエストはサポートされません。

4.4.2.2 動的バイナリコンテンツによるレンジリクエストへの応答

HTTP アプリケーションサーバーは、動的コンテンツの HTTP レンジリクエストに直接応答しません。動的コンテンツとは、アプリケーションコードによって生成されるコンテンツのことです。アプリケーションサーバーは動的コンテンツのレンジリクエストを無視しますが、アプリケーションの XQuery コードで Range ヘッダを処理し、適切なコンテンツで応答することはできます。

次のコードは、Range ヘッダを解釈し、レンジリクエストへの応答で動的に生成されたコンテンツを返す方法を示しています。

```
xquery version "1.0-ml";

(: This code assumes a simple range like 1000-2000; your :)
(: application code may support more complex ranges.     :)

let $data := fn:doc(xdmp:get-request-field("uri"))/binary()
let $range := xdmp:get-request-header("Range")
return
  if ($range)
  then
    let $range := replace(normalize-space($range),
    "bytes=", "")
    let $splits := tokenize($range, "-")
    let $start := xs:integer($splits[1])
    let $end := if ($splits[2] eq "")
                then xdmp:binary-size($data)-1
                else xs:integer($splits[2])
    let $ranges  :=
        concat("bytes ", $start, "-", $end, "/",
               xdmp:binary-size($data))
    return (xdmp:add-response-header("Content-Range",
            $ranges),
            xdmp:set-response-content-type("image/JPEG"),
            xdmp:set-response-code(206, "Partial Content"),
            xdmp:subbinary($data, $start+1, $end -
            $start + 1))
    else $data
```

上記のコードが XQuery モジュール fetch-bin.xqy 内にある場合、次のようなリクエストは、バイナリの最初の 100 バイトを返します（`curl` コマンドの `-r` オプションで、バイトのレンジが指定されます）。

```
$ curl -r "0-99" http://myhost:1234/fetch-bin.xqy?uri=
sample.jpg
```

このリクエストに対する応答は、次のようになります。

```
HTTP/1.1 206 Partial Content
Content-Range: bytes 0-99/1442323
Content-type: image/JPEG
Server: MarkLogic
Content-Length: 100

[first 100 bytes of sample.jpg]
```

4.4.3 バイナリメール添付ファイルの作成

バイナリファイルが添付されたメールメッセージを生成するには、xdmp:email を使用し、メッセージのコンテンツタイプを multipart/mixed に設定します。次の例は、JPEG ファイルが添付されたメールメッセージを生成します。

```
xquery version "1.0-ml";

(: generate a random boundary string :)
let $boundary := concat("blah", xdmp:random())
let $newline := "&#13;&#10;"
let $content-type := concat("multipart/mixed; boundary=",
$boundary)
let $attachment1 := xs:base64Binary(doc("/images/
sample.jpeg"))
let $content := concat(
  "--",$boundary,$newline,
  $newline,
  "This is a test email with an image attached.", $newline,
  "--",$boundary,$newline,
  "Content-Type: image/jpeg", $newline,
  "Content-Disposition: attachment; filename=sample.jpeg",
  $newline,
  "Content-Transfer-Encoding: base64", $newline,
  $newline,
  $attachment1, $newline,
  "--",$boundary,"--", $newline)

return
  xdmp:email(
  <em:Message
    xmlns:em="URN:ietf:params:email-xml:"
    xmlns:rf="URN:ietf:params:rfc822:">
    <rf:subject>Sample Email</rf:subject>
    <rf:from>
      <em:Address>
        <em:name>Myself</em:name>
        <em:adrs>me@somewhere.com</em:adrs>
      </em:Address>
    </rf:from>
    <rf:to>
      <em:Address>
        <em:name>Somebody</em:name>
        <em:adrs>somebody@somewhere.com</em:adrs>
      </em:Address>
    </rf:to>
```

```
    <rf:content-type>{$content-type}</rf:content-type>
    <em:content xml:space="preserve">
      {$content}
    </em:content>
  </em:Message>)
```

4.5 バイナリドキュメントの操作に役立つビルトイン

バイナリコンテンツを操作するために、次の XQuery ビルトインが用意されています。詳細については、『*XQuery and XSLT Reference Guide*』を参照してください。

- `xdmp:subbinary`
- `xdmp:binary-size`
- `xdmp:external-binary`
- `xdmp:external-binary-path`
- `xdmp:binary-is-small`
- `xdmp:binary-is-large`
- `xdmp:binary-is-external`

また、次の XQuery ビルトインは、外部バイナリコンテンツの作成や、その整合性のテストの際に役立つ可能性があります。

- `xdmp:filesystem-file-length`
- `xdmp:filesystem-file-exists`

5.0 XQuery モジュールおよび XSLT スタイルシートのインポートとパスの解決

XQuery モジュールは、MarkLogic サーバーの他の XQuery モジュールからインポートできます。同様に、XSLT スタイルシートを他のスタイルシートにインポートしたり、XQuery モジュールを XSLT スタイルシートにインポートしたり、XSLT スタイルシートを XQuery モジュールにインポートしたりできます。この章では、2 種類の XQuery モジュールについて説明し、モジュールのインポートや URI 参照の解決のためのルールを指定します。以下のセクションで構成されています。

- XQuery のライブラリモジュールとメインモジュール
- import/invoke/spawn パスの解決のルール
- モジュールキャッシュ処理の注意事項
- モジュールインポートのシナリオの例

XQuery ライブラリモジュールの XSLT スタイルシートへのインポートとその逆のインポートの詳細については、『*XQuery and XSLT Reference Guide*』の Notes on Importing Stylesheets With <xsl:import> および Importing a Stylesheet Into an XQuery Module を参照してください。

5.1 XQuery のライブラリモジュールとメインモジュール

XQuery モジュールは 2 種類あります（XQuery 仕様 http://www.w3.org/TR/xquer//#id-query-prolog を参照）。

- メインモジュール
- ライブラリモジュール

XQuery 言語の詳細については、『*XQuery and XSLT Reference Guide*』を参照してください。

5.1.1 メインモジュール

メインモジュールは、XQuery プログラムとして実行でき、XQuery 式を含むクエリボディを含んでいる必要があります（XQuery 式は、さらに XQuery 式を含む XQuery 式を含むことができます）。メインモジュールのシンプルな例を次に示します。

```
"hello world"
```

メインモジュールではプロローグを使用できますが、プロローグはオプションです。メインモジュールには、プロローグの一部として関数定義を含めることができます。ただし、メインモジュール内の関数定義は、そのモジュールでのみ利用可能であり、他のモジュールにはインポートできません。

5.1.2 ライブラリモジュール

ライブラリモジュールには名前空間があり、関数を定義するために使用されます。ライブラリモジュールは、直接評価されることはありません。他のライブラリモジュールやメインモジュールから `import` ステートメントを使用してインポートされます。ライブラリモジュールのシンプルな例を次に示します。

```
xquery version "1.0-ml";
module namespace hello = "helloworld";

declare function helloworld()
{
  "hello world"
};
```

このモジュールをファイル `c:/code/helloworld.xqy` に保存した場合、アプリケーションサーバーのファイルシステムルートが `c:/code` であると、次のように、メインモジュールまたはライブラリモジュールでこのモジュールをインポートして関数を呼び出すことができます。

```
xquery version "1.0-ml";
import module namespace hw="helloworld" at
"/helloworld.xqy";

hw:helloworld()
```

この XQuery プログラムは、`helloworld` 名前空間でこのライブラリモジュールをインポートし、その `helloworld()` 関数を評価します。

5.2 import/invoke/spawn パスの解決のルール

XQuery ライブラリモジュール内にある関数を呼び出すには、その名前空間でモジュールをインポートする必要があります。MarkLogic サーバーは、他の HTTP サーバーやアプリケーションサーバーがパスを解決する場合と同様の方法でライブラリパスを解決します。同様に、`xdmp:invoke` または `xdmp:spawn` を使用してモジュールを実行する場合は、モジュールへのアクセスをパスで指定します。これらのルールは、`xdmp:xslt-invoke` を使用するときの XSLT スタイルシートのパス、および `<xsl:import>` または `<xsl:include>` 指示でのスタイルシートのインポートにも適用されます。

import/invoke/spawn が使用された XQuery モジュールは、次のいずれかの場所に配置できます。

- Modules ディレクトリ内。
- 呼び出しモジュールに対して相対的に指定されたディレクトリ内。

- アプリケーションサーバーのルート下。ルートとは、Modules データベース内の指定されたディレクトリ（アプリケーションサーバーが Modules データベースに設定されている場合）、またはファイルシステム上の指定されたディレクトリ（アプリケーションサーバーがファイルシステム内のモジュールを見つけるように設定されている場合）です。

import/invoke/spawn パスを解決するときは、MarkLogic が最初にパスのルートを解決します。次に、パスに一致する、最初に見つかったモジュールを使用して、Modules ディレクトリ、アプリケーションサーバールートの順に調べてモジュールを探します。

import/invoke/spawn 式内のパスは、次のように解決されます。

1. import/invoke/spawn パスの先頭がスラッシュである場合は、最初に Modules ディレクトリ（Windows では通常 `c:\Program Files\MarkLogic\Modules`）を調べます。例：

   ```
   import module "foo" at "/foo.xqy";
   ```

   この例では、名前空間 `foo` のモジュールファイルを `c:\Program Files\MarkLogic\Modules\foo.xqy` で探します。

2. import/invoke/spawn パスの先頭がスラッシュであり、Modules ディレクトリで見つからない場合は、アプリケーションサーバールートを最初に調べます。例えば、アプリケーションサーバールートが `/home/mydocs/` であり、import が次のとおりであるとします。

   ```
   import module "foo" at "/foo.xqy";
   ```

   この場合は、名前空間 `foo` のモジュールを `/home/mydocs/foo.xqy` で探します。

   ファイルシステムルートの場合も、Modules データベースルートの場合も、最初に調べるのはアプリケーションサーバールートです。例えば、modules データベースを使用して設定されているアプリケーションサーバーで、ルートが `http://foo/` であるとします。

   ```
   import module "foo" at "/foo.xqy";
   ```

   この場合は、名前空間 `foo` のモジュールを、URI `http://foo/foo.xqy`（アプリケーションサーバールートを `foo.xqy` に付加して解決される）の modules データベースで探します。

3. import/invoke/spawn パスの先頭がスラッシュでない場合は、最初に Modules ディレクトリを調べます。そこでモジュールが見つからない場合は、関数を呼び出したモジュールの場所に相対的な場所で探します。例えば、/home/mydocs/bar.xqy にあるモジュールに次の import があるとします。

```
import module "foo" at "foo.xqy";
```

この場合は、名前空間 foo のモジュールを /home/mydocs/foo.xqy で探します。

ファイルシステムを使用するように設定されたアプリケーションサーバーの場合も、modules データベースを使用するように設定されたアプリケーションサーバーの場合も、最初に調べるのは、呼び出しモジュールの場所です。例えば、URI が http://foo/bar.xqy であるモジュールが modules データベース内に存在し、次の import ステートメントがあるとします。

```
import module "foo" at "foo.xqy";
```

この場合は、URI http://foo/foo.xqy のモジュールを modules データベースで探します。

4. import/invoke/spawn パスに、スキームまたはネットワークロケーションが含まれる場合、サーバーは例外をスローします。例：

```
import module "foo" at "http://foo/foo.xqy";
```

この場合は、無効パス例外がスローされます。以下も同様です。

```
import module "foo" at "c:/foo/foo.xqy";
```

この場合は、無効パス例外がスローされます。

5.3　モジュールキャッシュ処理の注意事項

XQuery モジュール（または XSLT ファイル）が、MarkLogic サーバーで設定されているアプリケーションサーバーのルートに格納される場合は、初回アクセス時に各モジュールが解析され、メモリ内にキャッシュされます。これで、モジュールへの以降のアクセスが高速になります。モジュールが更新されると、キャッシュは無効化され、そのアプリケーションサーバーに対する各モジュールには、次回の評価時に再解析が必要になります。モジュールキャッシュ処理は自動的に実行されるため、開発者にとって透過的です。ただし、モジュールの名前を考えるときは、次の点に注意してください。

- ベストプラクティスは、application/vnd.marklogic-xdmp または application/xslt+xml MIME タイプに対応するモジュールファイル拡張子を使用することです。デフォルトでは、拡張子 xqy、xq、および xslt です。他の拡張子をこれらの MIME タイプに追加するには、管理画面の MIME タイプ設定を使用します。
- application/vnd.marklogic-xdmp または application/xslt+xml に対応する MIME タイプ拡張子を持たないモジュールに変更を加えても、モジュールキャッシュは無効になりません。そのため、適切な拡張子を持たないモジュールに加えた変更を認識するには、各ホストでキャッシュを再度読み込む（サーバーを再起動する、適切な拡張子を持つモジュールを変更する、など）必要があります。

5.4 モジュールインポートのシナリオの例

次のシナリオについて考えてみましょう。

- c:/mydir というルートが定義された HTTP サーバーがあります。
- ファイル c:/mydir/lib.xqy 内には、インポート対象の関数が含まれるライブラリモジュールがあります。ライブラリモジュールのコンテンツは次のとおりです。

```
xquery version "1.0-ml";
module namespace hw="http://marklogic.com/me/my-module";

declare function hello()
{
"hello"
};
```

- ファイル c:/mydir/main.xqy 内には、上記のライブラリモジュールから関数をインポートする XQuery メインモジュールがあります。そのコードは次のとおりです。

```
xquery version "1.0-ml";

declare namespace my="http://marklogic.com/me/my-module";
import module "http://marklogic.com/me/my-module" at
"lib.xqy";

my:hello()
```

ライブラリモジュール lib.xqy は、アプリケーションサーバールートに相対的に（この例では c:/mydir に相対的に）インポートされます。

6.0 ライブラリサービスアプリケーション

この章では、ライブラリサービスを使用する方法について説明します。ライブラリサービスを使用すると、コンテンツ管理システム（CMS）と類似の方法で、バージョン設定されたコンテンツを MarkLogic サーバーで作成および管理できます。この章は、次のセクションで構成されています。

- ライブラリサービスとは
- ライブラリサービスを使用したアプリケーションの構築
- 必要なレンジ要素インデックス
- ライブラリサービス API
- ライブラリサービスアプリケーションのセキュリティ上の考慮事項
- トランザクションとライブラリサービス
- マネージドバージョン管理下へのドキュメントの配置
- マネージドドキュメントのチェックアウト
- マネージドドキュメントのチェックイン
- マネージドドキュメントの更新
- 保持ポリシーの定義
- ライブラリサービスでのモジュラードキュメントの管理

6.1 ライブラリサービスとは

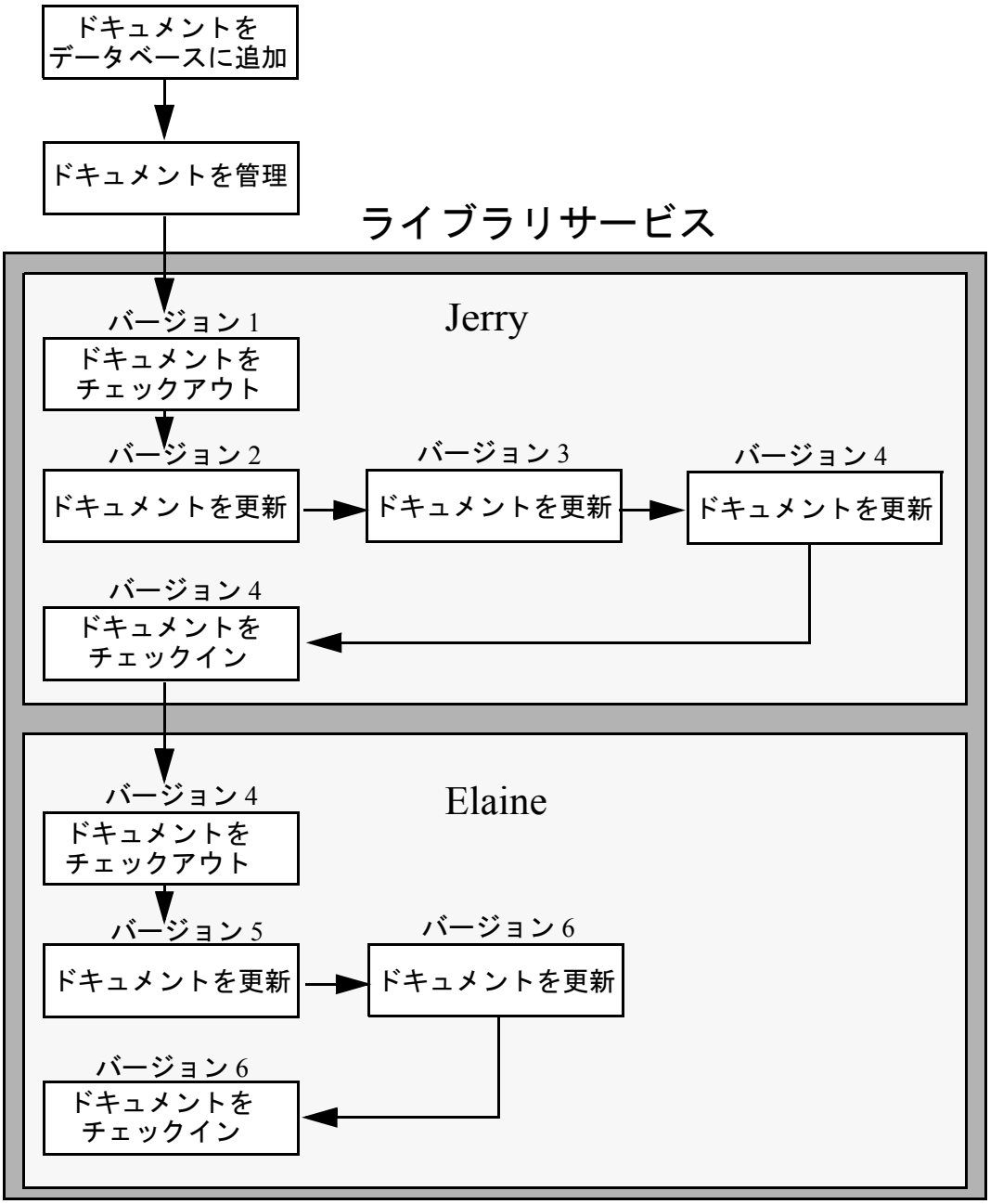
ライブラリサービスを使用すると、MarkLogic サーバーでマネージドドキュメントのバージョンを作成および維持できます。マネージドドキュメントへのアクセスは、チェックアウト / チェックインモデルを使用して制御されます。マネージドドキュメントに対して更新を行う場合は、まずこのドキュメントをチェックアウトする必要があります。チェックアウトされたドキュメントは、これをチェックアウトしたユーザーしか更新できません。他のユーザーがこのドキュメントを更新したい場合は、このドキュメントが再度チェックインされたあとにチェックアウトする必要があります。

> **注：** バージョンを設定するには、ドキュメントをデータベースに格納する必要があります。ドキュメントが CPF アプリケーション（エンティティエンリッチメント、モジュラードキュメント、変換、カスタム CPF アプリケーションなど）によって作成されている場合、これをデータベースに挿入する際に CPF アプリケーションがライブラリサービスを使っている場合にのみバージョンを付けることが可能。デフォルトでは、MarkLogic が提供する CPF アプリケーションはマネージドドキュメントを作成しません。

最初にドキュメントをライブラリサービス管理下に置くと、ドキュメントのバージョン 1 が作成されます。ドキュメントを更新するたびに、このドキュメントの新しいバージョンが作成される。更新されたドキュメントの旧バージョンは、保持ポリシーに基づいて保持されます（「保持ポリシーの定義」（102 ページ）を参照）。

ライブラリサービスには、モジュラードキュメントを管理するための関数が含まれているため、リンクされたドキュメントの各種バージョンを作成および管理できます（「ライブラリサービスでのモジュラードキュメントの管理」（108 ページ）を参照）。

次の図は、一般的なマネージドドキュメントのワークフローを表しています。この例では、ドキュメントがデータベースに追加され、ライブラリサービスで管理されるようになります。マネージドドキュメントは、Jerry によってチェックアウトされ、何度か更新された後に、チェックインされます。ドキュメントがチェックインされると、Elaine がこれをチェックアウトし、更新した後、同じマネージドドキュメントとしてチェックインします。更新されるたび、このドキュメントの前のバージョンは、保持規則に基づいて消去（パージ）されます。



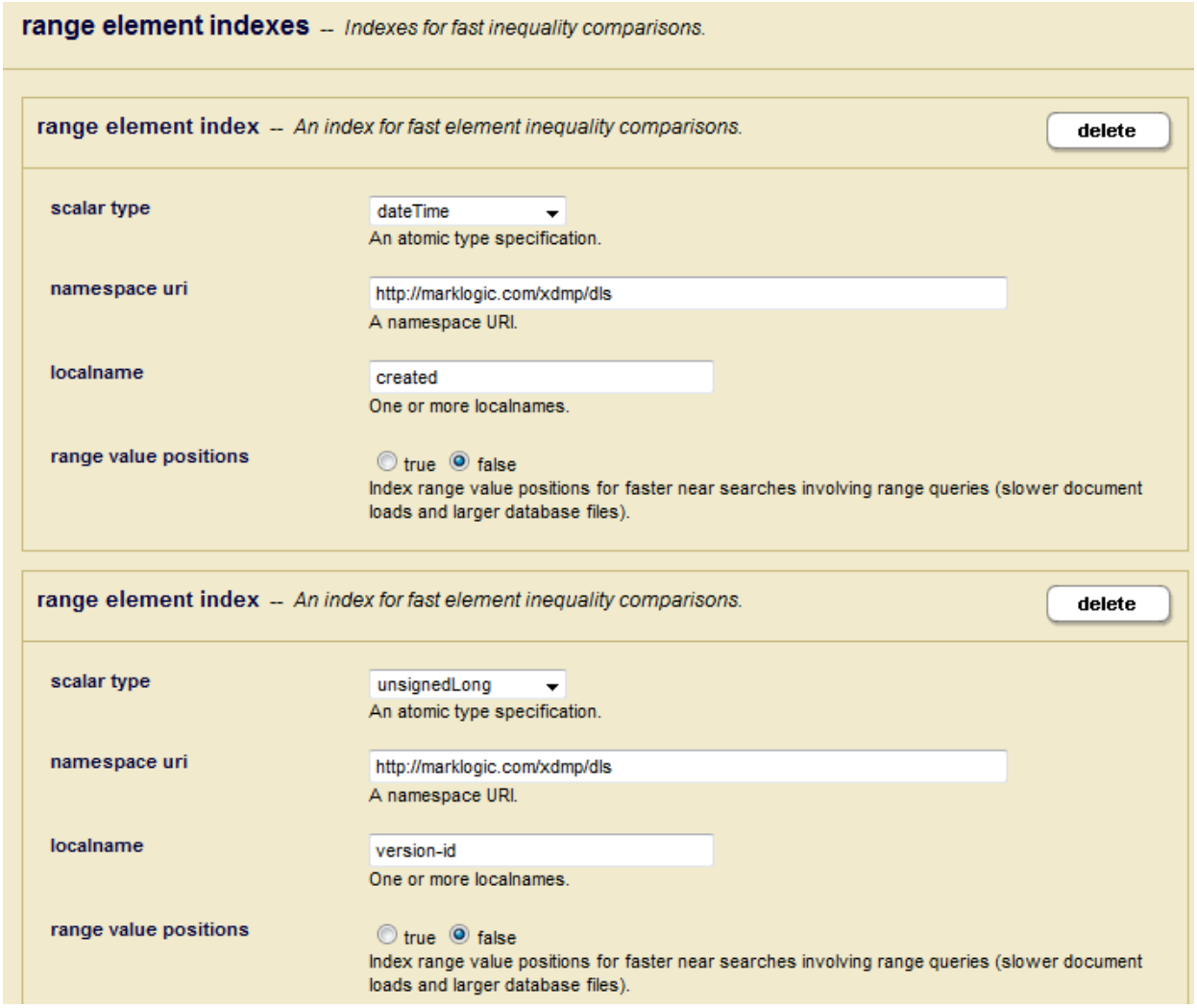
6.2 ライブラリサービスを使用したアプリケーションの構築

ライブラリサービス API には、特定の日付またはバージョンの時点におけるドキュメントの特定のドラフトを格納および抽出するアプリケーションを実装するための基本ツールが用意されています。また、他の MarkLogic サーバー API とともにライブラリサービス API を使用して、構造化ワークフローやバージョン管理を提供できます。
また、個別に管理されるコンポーネントにドキュメントを分割できるようにすることもできます。セキュリティ API は、ユーザーのロールや責任を、さまざまなドキュメントタイプやコレクションと関連付ける機能を提供します。また、search API で、強力なコンテンツ取得機能を実装できます。

6.3 必要なレンジ要素インデックス

以下の表と図に示すレンジ要素インデックスは、ライブラリサービスによって管理されるドキュメントを含むデータベースに対して必ず設定する必要があります。これらのインデックスは、データベースを新規作成する際に自動的に設定されます。ただし、以前のバージョンの MarkLogic サーバーで作成されたデータベースでライブラリサービスを利用する場合は、ライブラリサービスを手動で設定する必要があります。

scalar type	namespace uri	localname	range value positions
dateTime	http://marklogic.com/xdmp/dls	created	false
unsignedLong	http://marklogic.com/xdmp/dls	version-id	false



6.4 ライブラリサービス API

このセクションでは、ライブラリサービス API について説明します。以下のセクションで構成されています。

- ライブラリサービス API のカテゴリ
- マネージドドキュメント更新ラッパー関数

6.4.1 ライブラリサービス API のカテゴリ

ライブラリサービス関数については、『*MarkLogic XQuery and XSLT Function Reference*』を参照してください。ライブラリサービス関数は、次のカテゴリに分類されます。

- ドキュメント管理関数。ドキュメントをバージョン管理下に配置します。また、ドキュメントをバージョン管理にチェックインしたり、バージョン管理からチェックアウトしたりします。使用方法については、「マネージドバージョン管理下へのドキュメントの配置」（98 ページ）、「マネージドドキュメントのチェックアウト」（99 ページ）、および「マネージドドキュメントのチェックイン」（101 ページ）を参照してください。
- ドキュメント更新関数。ドキュメントのコンテンツとそのプロパティを更新します。使用方法については、「マネージドドキュメントの更新」（101 ページ）、および「マネージドドキュメント更新ラッパー関数」（96 ページ）を参照してください。
- 保持ポリシー関数。特定のドキュメントバージョンがパージされるタイミングを管理します。使用方法については、「保持ポリシーの定義」（102 ページ）を参照してください。
- XInclude 関数。リンクされたドキュメントを作成および管理します。使用方法については、「ライブラリサービスでのモジュラードキュメントの管理」（108 ページ）を参照してください。
- `cts:query` コンストラクタ関数。`cts:search`、ライブラリサービスの XInclude 関数、および保持規則の定義時に使用します。使用方法については、「保持ポリシーの定義」（102 ページ）を参照してください。

6.4.2 マネージドドキュメント更新ラッパー関数

マネージドドキュメントに対するすべての更新および削除の操作は、ライブラリサービス API を使用して実行する必要があります。ライブラリサービス API には、次の「ラッパー」関数が含まれています。これらの関数を使用すると、対応する XDMP 関数を使用して非マネージドドキュメントを更新する場合と同様に、マネージドドキュメントを更新できます。

- `dls:document-add-collections`
- `dls:document-add-permissions`
- `dls:document-add-properties`
- `dls:document-set-collections`
- `dls:document-set-permissions`
- `dls:document-set-properties`
- `dls:document-remove-properties`

- `dls:document-remove-permissions`
- `dls:document-remove-collections`
- `dls:document-set-property`
- `dls:document-set-quality`

注： ドキュメントでコレクションまたはプロパティの設定のみを変更すると、これらの設定は、ドキュメントがチェックインされるときにバージョン履歴で維持されません。コレクションやプロパティに対する変更のバージョン管理を行うには、ドキュメントのコンテンツも変更する必要があります。

6.5 ライブラリサービスアプリケーションのセキュリティ上の考慮事項

ライブラリサービスアプリケーションに関しては、事前定義されたロールが 2 つあります。また、ライブラリサービス API が使用する「インターナル」ロールもあります。

- dls-admin ロール
- dls-user ロール
- dls-internal ロール

注： マネージドドキュメントをデータベースに挿入するとき、またはライブラリサービスをテストするときは、管理者ロールとしてログインしないでください。代わりに、`dls-user` ロールを持つテストユーザーを作成し、このユーザーに、マネージドドキュメントへのアクセスに必要なさまざまなパーミッションを付与します。自分が書いたコードを Query Console でテストする場合も、テストユーザーに `qconsole-user` ロールを割り当てる必要があります。

6.5.1 dls-admin ロール

`dls-admin` ロールは、ライブラリサービスアプリケーションの管理者に、ライブラリサービス API を使用するために必要なすべての権限を付与するために用意されています。このロールには、保持ポリシーの挿入やチェックアウトの取り消しなどの操作を実行するために必要な権限が含まれています。そのため、`dls-admin` ロールは、信頼できるユーザー（悪意を持っておらず、適切なトレーニングを受けており、正しい管理手続きに従うことが想定されるユーザー）のみに付与する必要があります。`dls-admin` ロールは、ライブラリサービスアプリケーションの管理者に割り当ててください。

6.5.2 dls-user ロール

dls-user ロールは、最低限の権限を持つロールです。このロールをライブラリサービス API で使用すると、通常のライブラリサービスアプリケーションユーザー（dls-admin ユーザーではない）がライブラリサービス API でコードを実行できます。ユーザーにはドキュメント更新パーミッションが付与され、マネージドドキュメントの管理、チェックアウト、チェックインが可能になります。

dls-user ロールには、ライブラリサービス API の実行に必要な権限だけが含まれています。ライブラリサービス API のスコープ外の関数に対する実行権限は含まれません。ライブラリサービス API では、上位の権限が必要な操作を、制御された方法で強化するメカニズムとして、dls-user ロールを使用しています。そのため、このロールは、ライブラリサービスアプリケーションを使用してもかまわないと考えられるどのユーザーに割り当てても、それほど危険ではありません。dls-user ロールは、ライブラリサービスアプリケーションのすべてのユーザーに割り当ててください。

6.5.3 dls-internal ロール

dls-internal ロールは、ライブラリサービス API によって内部的に使用されるロールです。このロールは、どのユーザーやロールにも明示的に付与しないでください。このロールは、ライブラリサービスの特定の関数のコンテクストにおいて特別な権限を強化するために使用されます。これをユーザーに割り当ててしまうと、通常は望ましくない権限をシステムに関して与えてしまうことになります。このロールは、どのユーザーにも割り当てないでください。

6.6 トランザクションとライブラリサービス

dls:document-checkout、dls:document-update、dls:document-checkin 関数は別々のトランザクションとして実行される必要があります。チェックアウト、更新、チェックインを単一のトランザクション内で完了する場合は、dls:document-checkout-update-checkin 関数を使用します。

6.7 マネージドバージョン管理下へのドキュメントの配置

マネージドバージョン管理下に置くには、ドキュメントがコンテンツデータベース内にある必要があります。ドキュメントがこのデータベース内に配置されると、dls-user ロールが割り当てられたユーザーは、dls:document-manage 関数を使用してドキュメントを管理下に置くことができます。または、dls:document-insert-and-manage 関数を使用すると、ドキュメントのデータベースに挿入すると同時に管理下に置くことができます。

マネージドドキュメントを挿入するときは、ドキュメントを管理するユーザーに割り当てられたロールに、少なくとも読み取りと更新のパーミッションを指定する必要があります。パーミッションを指定していない場合は、マネージドドキュメントを挿入するユーザーのデフォルトのパーミッションが適用されます。デフォルトのパーミッションを取得するには、xdmp:default-permissions 関数を呼び出します。以下の例に示すようにコレクションをドキュメントに追加するときは、ユーザーに unprotected-collections 権限も必要です。

例えば、次のクエリでは、新しいドキュメントがデータベースに挿入され、ライブラリサービス管理下に置かれます。このドキュメントは、writer ロールや editor ロールを割り当てられ、http://marklogic.com/engineering/specs コレクションの読み取りおよび更新のパーミッションを持つユーザーのみが読み取りまたは更新を行うことができます。

```
(: Insert a new managed document into the database.:)
xquery version "1.0-ml";

import module namespace dls = "http://marklogic.com/
xdmp/dls"
        at "/MarkLogic/dls.xqy";

dls:document-insert-and-manage(
        "/engineering/beta_overview.xml",
        fn:true(),
        <TITLE>Project Beta Overview</TITLE>,
        "Manage beta_overview.xml",
        (xdmp:permission("writer", "read"),
         xdmp:permission("writer", "update"),
         xdmp:permission("editor", "read"),
         xdmp:permission("editor", "update")),
        ("http://marklogic.com/engineering/specs"))
```

6.8 マネージドドキュメントのチェックアウト

更新操作を実行する前に、最初に dls:document-checkout 関数を使用してマネージドドキュメントをチェックアウトする必要があります。例えば、beta_overview.xml ドキュメント、およびそのリンクされたドキュメントすべてをチェックアウトするには、次のように指定します。

```
xquery version "1.0-ml";

import module namespace dls = "http://marklogic.com/xdmp/dls"
at "/MarkLogic/dls.xqy";
```

```
dls:document-checkout(
        "/engineering/beta_overview.xml",
        fn:true(),
        "Updating doc")
```

オプションとして、timeout パラメータを dls:document-checkout に指定できます。これで、ドキュメントのチェックアウトの最大時間（秒）を指定します。例えば、beta_overview.xml ドキュメントを 1 時間チェックアウトするには、次のように指定します。

```
dls:document-checkout(
        "/engineering/beta_overview.xml",
        fn:true(),
        "Updating doc",
        3600)
```

6.8.1 マネージドドキュメントのチェックアウトステータスの表示

dls:document-checkout-status 関数を使用して、チェックアウトされたドキュメントのステータスを報告できます。例：

```
dls:document-checkout-status("/engineering/
beta_overview.xml")
```

次のような出力が返されます。

```
<dls:checkout xmlns:dls="http://marklogic.com/xdmp/dls">
  <dls:document-uri>/engineering/beta_overview.xml
  </dls:document-uri>
  <dls:annotation>Updating doc</dls:annotation>
  <dls:timeout>0</dls:timeout>
  <dls:timestamp>1240528210</dls:timestamp>
  <sec:user-id xmlns:sec="http://marklogic.com/xdmp/
  security">
    10677693687367813363
  </sec:user-id>
</dls:checkout>
```

6.8.2 マネージドドキュメントのチェックアウトの取り消し

dls-admin ロールを持つユーザーは、dls:break-checkout を呼び出してドキュメントの「チェックアウトを取り消す」ことができます。例えば、あるドキュメントをチェックアウトした人が他のプロジェクトに異動になっている場合、管理者は現在進行中のチェックアウトを終了し、他のユーザーがチェックアウト可能にできます。

6.9 マネージドドキュメントのチェックイン

ドキュメントの更新が完了したら、dls:document-checkin 関数を使用して、ドキュメントとそのリンクされたドキュメントのすべてをチェックインします。

```
dls:document-checkin(
        "/engineering/beta_overview.xml",
        fn:true() )
```

6.10 マネージドドキュメントの更新

dls:document-update 関数を呼び出して、既存のマネージドドキュメントのコンテンツを置き換えることができます。ドキュメントに対して dls:document-update 関数を呼び出すたびに、ドキュメントのバージョンが 1 ずつ増加します。また、パージ操作が開始されて、保持ポリシーによって保持されないドキュメントのバージョンが削除されます（「保持ポリシーの定義」（102 ページ）を参照）。

注： マネージドドキュメントには、ノード更新の関数（xdmp:node-replace など）を使用できません。ドキュメントへの更新をメモリ内で行ってから、dls:document-update 関数を呼び出す必要があります。ドキュメントノードでインメモリ更新を実行する方法については、「再帰的 typeswitch 式による XML 構造の変換」（115 ページ）を参照してください。

例えば、「Project Beta Overview」ドキュメントを更新するには、次のように入力します。

```
let $contents :=
<BOOK>
  <TITLE>Project Beta Overview</TITLE>
  <CHAPTER>
    <TITLE>Objectives</TITLE>
    <PARA>
    The objective of Project Beta, in simple terms, is to
    corner the widget market.
    </PARA>
  </CHAPTER>
</BOOK>

return
    dls:document-update(
       "/engineering/beta_overview.xml",
       $contents,
       "Roughing in the first chapter",
       fn:true())
```

注： dls:document-update 関数は、ドキュメントのコンテンツ全体を置き換えます。

6.11 保持ポリシーの定義

「保持ポリシー」で、パージ操作後にデータベースに保持されるドキュメントバージョンが指定されます。保持ポリシーは、1 つあるいは複数の「保持規則」で構成されます。保持ポリシーを定義しない場合、以前のドキュメントバージョンはまったく保持されません。

このセクションでは、次の内容を取り上げます。

- マネージドドキュメントのバージョンのパージ
- 保持規則について
- 保持規則の作成
- 特定のドキュメントバージョンの保持
- 複数の保持規則
- 保持規則の削除

6.11.1 マネージドドキュメントのバージョンのパージ

マネージドドキュメントが更新されるたびに、パージ操作が開始されます。これによって、保持ポリシーによって保持されないドキュメントバージョンが削除されます。また、`dls:purge` を呼び出してすべてのドキュメントをパージすることも、`dls:document-purge` を呼び出して特定のマネージドドキュメントでパージを実行することもできます。

`dls:purge` または `dls:document-purge` を使用して、保持ポリシーによって削除される「予定」のドキュメントを、実際には削除せずに特定できます。このオプションは、保持規則を作成するときに役立ちます。例えば、保持ポリシーを変更した結果、削除されることになるドキュメントバージョンを具体的に特定する場合は、次のように使用できます。

```
xquery version "1.0-ml";
import module namespace dls="http://marklogic.com/xdmp/dls"
              at "/MarkLogic/dls.xqy";

dls:purge(fn:false(), fn:true())
```

6.11.2 保持規則について

保持規則は、パージ（消去）操作の際に保持されるドキュメントのバージョンを指定します。`dls:document-update` あるいは `dls:document-extract-part` を使ってドキュメントの新しいバージョンを作成した場合は、旧バージョンのうち、保持規則を満たさないものがパージされます。

保持規則を定義することで、さまざまな数のバージョンを保持したり、cts-query 式に一致するドキュメントを保持したり、特定期間だけドキュメントを保持したりできます。保持規則内の制限は、論理 AND によって結合されます。つまり、ドキュメントバージョンが保持されるには、保持規則内のすべての式が true である必要があります。個別の保持規則を結合した保持ポリシーでは、各規則が OR によって結合されています（つまり、規則のいずれかに一致すると、ドキュメントバージョンが保持されます）。複数のルールには、演算順序はありません。

警告 保持ポリシーでは、パージされるものではなく、「保持される」ものが指定されます。したがって、保持ポリシーに一致しないものがすべて削除されます。

6.11.3 保持規則の作成

保持規則を作成するには、dls:retention-rule 関数を呼び出します。dls:retention-rule-insert 関数は、1 つあるいは複数の保持規則をデータベースに挿入します。

例えば、次の保持規則では、すべてのドキュメントのすべてのバージョンが保持されます。空の cts:and-query 関数は、すべてのドキュメントに一致するためです。

```
xquery version "1.0-ml";
import module namespace dls="http://marklogic.com/xdmp/dls"
              at "/MarkLogic/dls.xqy";

dls:retention-rule-insert(
dls:retention-rule(
    "All Versions Retention Rule",
    "Retain all versions of all documents",
    (),
    (),
    "Locate all of the documents",
    cts:and-query(()) ) )
```

次の保持規則では、/engineering/ ディレクトリ下にあるすべてのドキュメントの最新の 5 バージョンが保持されます。

```
xquery version "1.0-ml";
import module namespace dls="http://marklogic.com/xdmp/dls"
              at "/MarkLogic/dls.xqy";

dls:retention-rule-insert(
dls:retention-rule(
    "Engineering Retention Rule",
    "Retain the five most recent versions of Engineering
    docs",
```

```
    5,
    (),
    "Locate all of the Engineering documents",
    cts:directory-query("/engineering/", "infinity") ) )
```

次の保持規則では、Jim が作成した、タイトルに「Project Alpha」が含まれる技術ドキュメントの最新の 3 バージョンが保持されます。

```
xquery version "1.0-ml";
import module namespace dls="http://marklogic.com/xdmp/dls"
             at "/MarkLogic/dls.xqy";

dls:retention-rule-insert(
dls:retention-rule(
    "Project Alpha Retention Rule",
    "Retain the three most recent engineering documents
     with the title 'Project Alpha' and authored by Jim.",
    3,
    (),
    "Locate the engineering docs with 'Project Alpha' in
     the title authored by Jim",
    cts:and-query((
       cts:element-word-query(xs:QName("TITLE"),
       "Project Alpha"),
       cts:directory-query("/engineering/", "infinity"),
       dls:author-query(xdmp:user("Jim")) )) ) )
```

次の保持規則では、「specs」コレクションに含まれる、作成後 30 日以内のドキュメントのうち、直近の 5 バージョンが保持されます。

```
xquery version "1.0-ml";
import module namespace dls="http://marklogic.com/xdmp/dls"
             at "/MarkLogic/dls.xqy";

dls:retention-rule-insert(
dls:retention-rule(
    "Specs Retention Rule",
    "Keep the five most recent versions of documents in the
    'specs'collection that are 30 days old or newer",
    5,
    xs:duration("P30D"),
    "Locate documents in the 'specs' collection",
    cts:collection-query("http://marklogic.com/documents/
    specs") ) )
```

6.11.4 特定のドキュメントバージョンの保持

保持規則で dls:document-version-query および dls:as-of-query コンストラクタ関数を使用して、ある時点でのドキュメントの「スナップショット」を保持できます。特定の日付の時点での、特定のドキュメントバージョンまたはドキュメントのスナップショットを保持できます。

例えば、次の保持規則では、2009 年 4 月 23 日午後 5:00 より前に作成された、技術ドキュメントの最新バージョンが保持されます。

```
xquery version "1.0-ml";
import module namespace dls="http://marklogic.com/xdmp/dls"
              at "/MarkLogic/dls.xqy";

dls:retention-rule-insert(
dls:retention-rule(
    "Draft 1 of the Engineering Docs",
    "Retain each engineering document that was update
     before 5:00pm, 4/23/09",
    (),
    (),
    (),
    cts:and-query((
       cts:directory-query("/documentation/", "infinity"),
       dls:as-of-query(xs:dateTime("2009-04-23T17:00:00-
       07:00")) )) ))
```

技術ドキュメントのスナップショットを 2 つ保持する場合は、別の cts:or-query 関数を含む保持規則を追加できます。例：

```
cts:and-query((
    cts:directory-query("/documentation/", "infinity"),
    dls:as-of-query(xs:dateTime("2009-25-12T09:00:01-
    07:00")) ))
```

6.11.5 複数の保持規則

組織によっては、複数の保持規則が必要な場合もあります。例えば、エンジニア部やドキュメント管理部が管理する共有データベースを複数の部署が利用する場合、各部署ごとに独自の保持規則がある場合などです。

以下のような 2 つの規則を考えてみましょう。1 つ目の規則では、/engineering/ ディレクトリ下にあるすべてのドキュメントの最新の 5 バージョンが保持されます。2 つ目の規則では、/documentation/ ディレクトリ下にあるすべてのドキュメントの最新の 10 バージョンが保持されます。これらの 2 つの規則の OR の結果は、各規則の意図に影響しません。また、各規則は、互いに影響することなく更新できます。

```
xquery version "1.0-ml";
import module namespace dls="http://marklogic.com/xdmp/dls"
              at "/MarkLogic/dls.xqy";

dls:retention-rule-insert((
dls:retention-rule(
    "Engineering Retention Rule",
    "Retain the five most recent versions of Engineering
    docs",
    5,
    (),
    "Apply to all of the Engineering documents",
    cts:directory-query("/engineering/", "infinity") ),

dls:retention-rule(
    "Documentation Retention Rule",
    "Retain the ten most recent versions of the
    documentation",
    10,
    (),
    "Apply to all of the documentation",
    cts:directory-query("/documentation/", "infinity") ) ))
```

前述のとおり、複数の保持規則には論理 OR が定義されます。したがって、同一のドキュメントセットに対して、希望する保持ポリシーを定義するために、複数の保持規則が必要になる場合があります。

例えば、すべての技術ドキュメントの最新の 5 バージョンと、2009 年 4 月 24 日午前 8:00 より前および 2009 年 5 月 12 日午前 9:00 より前に更新されたすべての技術ドキュメントを保持するとします。希望する保持ポリシーを定義するには、次に示す 2 つの保持規則が必要です。

```
xquery version "1.0-ml";
import module namespace dls="http://marklogic.com/xdmp/dls"
              at "/MarkLogic/dls.xqy";

dls:retention-rule-insert((
dls:retention-rule(
    "Engineering Retention Rule",
    "Retain the five most recent versions of Engineering
    docs",
    5,
    (),
    "Retain all of the Engineering documents",
    cts:directory-query("/engineering/", "infinity") ),
```

```
dls:retention-rule(
    "Project Alpha Retention Rule",
    "Retain the engineering documents that were updated
     before the review dates below.",
    (),
    (),
    "Retain all of the Engineering documents updated before
     the two dates",
    cts:and-query((
      cts:directory-query("/engineering/", "infinity"),
      cts:or-query((
        dls:as-of-query(xs:dateTime("2009-04-24T08:00:
        17.566-07:00")),
        dls:as-of-query(xs:dateTime("2009-05-12T09:00:
        01.632-07:00"))
        ))
     )) ) ))
```

上記の 2 つの保持規則を論理 OR で結合することと、1 つの規則内で論理 AND を使用することの違いを理解することが重要です。例えば、上記の 2 つの保持規則の OR 結合は、以下の 1 つの規則と同じではありません。この 1 つの規則は、最新の 5 バージョンの保持と、指定時点でのバージョンの保持の AND 結合であるためです。この規則の最終結果では、最新の 5 バージョンは保持されず、指定時点でのバージョンのうち、最新の 5 バージョンに含まれるもののみが保持されます。最新の 5 つのドキュメントのリビジョンが指定時点の日付以降に移動されると、この AND ロジックは true ではなくなり、有効な保持ポリシーではなくなるため、ドキュメントバージョンはまったく保持されません。

```
xquery version "1.0-ml";
import module namespace dls="http://marklogic.com/xdmp/dls"
              at "/MarkLogic/dls.xqy";

dls:retention-rule-insert(
dls:retention-rule(
    "Project Alpha Retention Rule",
    "Retain the 5 most recent engineering documents",
    5,
    (),
    "Retain all of the Engineering documents updated before
     the two dates",
    cts:and-query((
      cts:directory-query("/engineering/", "infinity"),
      cts:or-query((
        dls:as-of-query(xs:dateTime("2009-04-24T08:56:
        17.566-07:00")),
```

```
            dls:as-of-query(xs:dateTime("2009-05-12T08:59:
            01.632-07:00"))
          )) )) ) )
```

6.11.6 保持規則の削除

保持規則は、dls:retention-rule-remove 関数を使用して削除できます。例えば、「Project Alpha Retention Rule」を削除するには、次のように使用します。

```
xquery version "1.0-ml";
import module namespace dls="http://marklogic.com/xdmp/dls"
              at "/MarkLogic/dls.xqy";

dls:retention-rule-remove("Project Alpha Retention Rule")
```

データベース内のすべての保持規則を削除するには、次のように使用します。

```
xquery version "1.0-ml";
import module namespace dls="http://marklogic.com/xdmp/dls"
              at "/MarkLogic/dls.xqy";

dls:retention-rule-remove(fn:data(dls:retention-rules("*")
//dls:name))
```

6.12 ライブラリサービスでのモジュラードキュメントの管理

「モジュラードキュメントアプリケーションによるコンテンツの再利用」（180 ページ）で説明するように、1 つあるいは複数のリンクされたドキュメントに格納されているコンテンツからモジュラードキュメントを作成できます。このセクションでは、次の内容を取り上げます。

- マネージドモジュラードキュメントの作成
- マネージドモジュラードキュメントの展開
- モジュラードキュメントのバージョンの管理

6.12.1 マネージドモジュラードキュメントの作成

「モジュラードキュメントアプリケーションによるコンテンツの再利用」（180 ページ）で説明するように、1 つあるいは複数のリンクされたドキュメントに格納されているコンテンツからモジュラードキュメントを作成できます。dls:document-extract-part 関数を使用すると、モジュラーマネージドドキュメントを簡単に作成できます。この関数は、マネージドドキュメントから子要素を抽出し、新しいマネージドドキュメントに子要素を配置し、抽出した子要素を XInclude 参照に置換します。

例えば、次の関数は「Project Beta Overview」ドキュメントから第 1 章を抽出します。

```
dls:document-extract-part("/engineering/beta_overview_
chap1.xml",
    fn:doc("/engineering/beta_overview.xml")//CHAPTER[1],
    "Extracting Chapter 1",
    fn:true() )
```

/engineering/beta_overview.xml のコンテンツは、次のようになります。

```
<BOOK>
  <TITLE>Project Beta Overview</TITLE>
  <xi:include href="/engineering/beta_overview_chap1.xml"/>
</BOOK>
```

/engineering/beta_overview_chap1.xml のコンテンツは、次のようになります。

```
<CHAPTER>
  <TITLE>Objectives</TITLE>
  <PARA>
   The objective of Project Beta, in simple terms, is to
   corner the widget market.
  </PARA>
</CHAPTER>
```

注: 抽出した子要素が含まれる、新しく作成したマネージドドキュメントは、最初からチェックインされているため、更新を加えるには、チェックアウトする必要があります。

dls:document-extract-part 関数は、同一ドキュメントに対してトランザクション内で 1 回だけ呼び出すことができます。1 つのドキュメントから複数の要素を抽出して XInclude ステートメントに置換する必要がある場合があります。例えば、次のクエリは、「Project Beta Overview」ドキュメントのすべての章に対して個別のドキュメントを作成し、それを XInclude ステートメントに置換します。

```
xquery version "1.0-ml";
import module namespace dls="http://marklogic.com/xdmp/dls"
              at "/MarkLogic/dls.xqy";

declare namespace xi="http://www.w3.org/2001/XInclude";

let $includes := for $chap at $num in
   doc("/engineering/beta_overview.xml")/BOOK/CHAPTER
```

```
return (
   dls:document-insert-and-manage(
     fn:concat("/engineering/beta_overview_chap",
     $num, ".xml"),
     fn:true(),
     $chap),

   <xi:include href="/engineering/beta_overview_chap
   {$num}.xml"
    xmlns:xi="http://www.w3.org/2001/XInclude"/>
  )

let $contents :=
   <BOOK>
      <TITLE>Project Beta Overview</TITLE>
      {$includes}
   </BOOK>

return
   dls:document-update(
      "/engineering/beta_overview.xml",
      $contents,
      "Chapters are XIncludes",
      fn:true() )
```

このクエリは、次のような「Project Beta Overview」ドキュメントを生成します。

```
<BOOK>
   <TITLE>Project Beta Overview</TITLE>
   <xi:include href="/engineering/beta_overview_chap1.xml"
    xmlns:xi="http://www.w3.org/2001/XInclude"/>
   <xi:include href="/engineering/beta_overview_chap1.xml"
    xmlns:xi="http://www.w3.org/2001/XInclude"/>
   <xi:include href="/engineering/beta_overview_chap2.xml"
    xmlns:xi="http://www.w3.org/2001/XInclude"/>
</BOOK>
```

6.12.2 マネージドモジュラードキュメントの展開

リンクされた各ノードを含むノード全体、またはリンクされた特定のノードを表示するために、モジュラードキュメントを「展開」できます。`dls:node-expand` を使用してモジュラードキュメントを展開したり、`dls:link-expand` を使用してモジュラードキュメント内のリンクされたノードを展開したりできます。

注： バージョン設定された特定のドキュメントへの XInclude リンクが含まれるドキュメントを展開するために `dls:node-expand` 関数を使用する場合は、`$restriction` パラメータを空のシーケンスとして指定します。

例えば、展開した `beta_overview.xml` ドキュメントを返すには、次のように使用します。

```
xquery version "1.0-ml";
import module namespace dls="http://marklogic.com/xdmp/dls"
              at "/MarkLogic/dls.xqy";

let $node := fn:doc("/engineering/beta_overview.xml")

return dls:node-expand($node, ())
```

`beta_overview.xml` ドキュメント内の、リンクされた最初のノードを返すには、次のように使用します。

```
xquery version "1.0-ml";
import module namespace dls="http://marklogic.com/xdmp/dls"
              at "/MarkLogic/dls.xqy";

declare namespace xi="http://www.w3.org/2001/XInclude";

let $node := fn:doc("/engineering/beta_overview.xml")

return dls:link-expand(
               $node,
               $node/BOOK/xi:include[1],
               () )
```

`dls:node-expand` 関数および `dls:link-expand` 関数を使用すると、`cts:query` コンストラクタを指定して、展開するドキュメントバージョンを制限できます。例えば、2009 年 4 月 6 日の午後 1:30 より前に作成された「Project Beta Overview」ドキュメントの最新バージョンを展開するには、次のように使用します。

```
xquery version "1.0-ml";
import module namespace dls="http://marklogic.com/xdmp/dls"
              at "/MarkLogic/dls.xqy";

let $node := fn:doc("/engineering/beta_overview.xml")

return dls:node-expand(
         $node,
         dls:as-of-query(
          xs:dateTime("2009-04-06T13:30:33.576-07:00")) )
```

6.12.3 モジュラードキュメントのバージョンの管理

ライブラリサービスではモジュラードキュメントを管理できるため、リンクされたドキュメントに対して各種バージョンを作成できます。モジュラードキュメントのリンクされたドキュメントは更新されるため、ノード全体のスナップショットを定期的に作成する必要がある場合があります。

例えば、「マネージドモジュラードキュメントの作成」（108 ページ）に示すように、「Project Beta Overview」ドキュメントには 3 つの章が含まれており、それぞれが別のドキュメントとしてリンクされています。次のクエリは、各章の最新バージョンのスナップショットを作成し、バージョン設定された章が含まれる「Project Beta Overview」ドキュメントの新しいバージョンを作成します。

```
xquery version "1.0-ml";
import module namespace dls="http://marklogic.com/xdmp/dls"
              at "/MarkLogic/dls.xqy";

declare namespace xi="http://www.w3.org/2001/XInclude";

(: For each chapter in the document, get the URI :)
let $includes :=
  for $chap at $num in doc("/engineering/beta_overview.xml")
  //xi:include/@href

(: Get the latest version of each chapter :)
let $version_number :=
  fn:data(dls:document-history($chap)//dls:version-id)
  [last()]

let $version :=  dls:document-version-uri($chap,
$version_number)

(: Create an XInclude statement for each versioned chapter :)
  return
    <xi:include href="{$version}"/>

(: Update the book with the versioned chapters :)
let $contents :=
  <BOOK>
    <TITLE>Project Beta Overview</TITLE>
    {$includes}
  </BOOK>

return
  dls:document-update(
    "/engineering/beta_overview.xml",
```

```
    $contents,
    "Latest Draft",
    fn:true() )
```

上記のクエリによって、次のような「Project Beta Overview」ドキュメントの新しいバージョンが作成されます。

```
<BOOK>
  <TITLE>Project Beta Overview</TITLE>
  <xi:include
href="/engineering/beta_overview_chap1.xml_versions/
4-beta_overview_chap1.xml" xmlns:xi="http://www.w3.org/
2001/XInclude"/>
  <xi:include
href="/engineering/beta_overview_chap2.xml_versions/3-beta
_overview_chap2.xml" xmlns:xi="http://www.w3.org/
2001/XInclude"/>
  <xi:include
href="/engineering/beta_overview_chap3.xml_versions/3-beta
_overview_chap3.xml" xmlns:xi="http://www.w3.org/
2001/XInclude"/>
</BOOK>
```

注： バージョン設定された特定のドキュメントへの XInclude リンクが含まれるモジュラードキュメントを展開するために `dls:node-expand` 関数を使用する場合は、`$restriction` パラメータを空のシーケンスとして指定します。

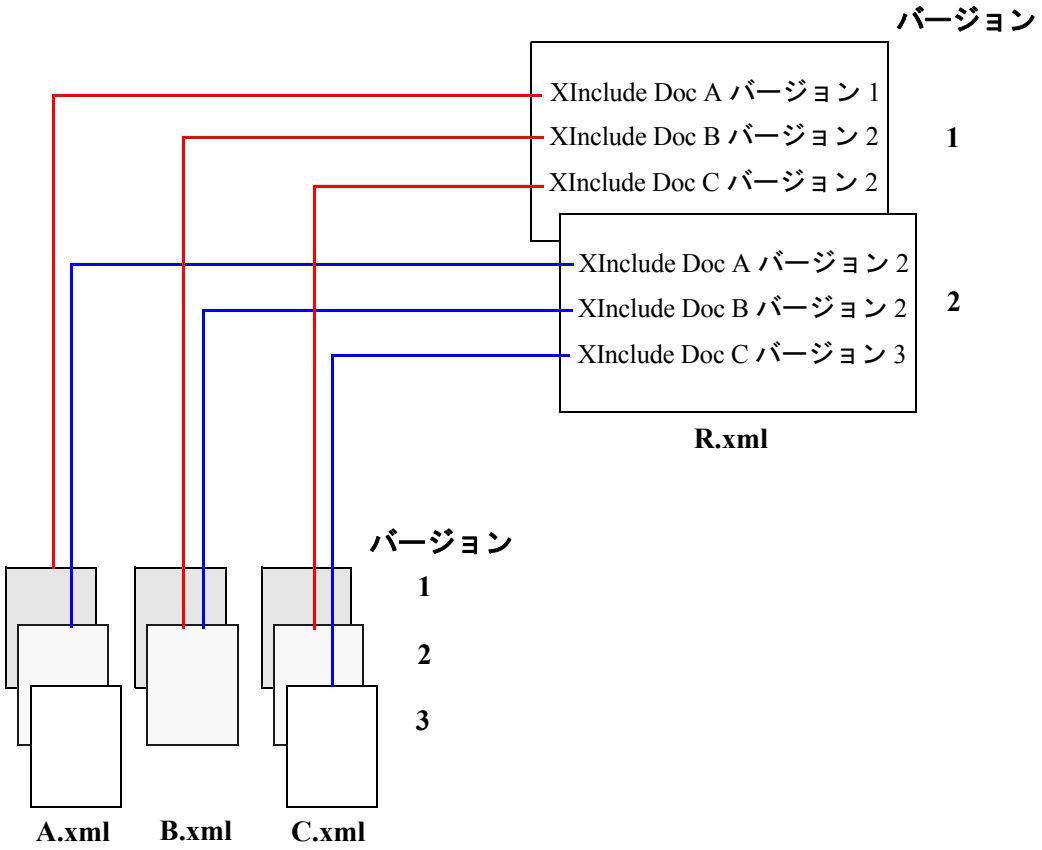
リンクされたドキュメントのさまざまなバージョンが含まれるモジュラードキュメントを作成することもできます。例えば、下の図のように、Doc R.xml バージョン 1 に次のコンテンツが含まれているとします。

- Doc A.xml バージョン 1
- Doc B.xml バージョン 2
- Doc C.xml バージョン 2

Doc X バージョン 2 に次のコンテンツが含まれているとします。

- Doc A.xml バージョン 2
- Doc B.xml バージョン 2

- Doc C.xml バージョン 3



7.0 再帰的 typeswitch 式による XML 構造の変換

XML では、構造を別の構造に変換するための一般的な作業が必要です。この章では、XQuery `typeswitch` 式を使用したデザインパターンについて説明します。この式を使用すると、高いパフォーマンスで、複雑な XML 変換を簡単に実行できます。また、このデザインパターンを図示したサンプルも示します。以下のセクションで構成されています。

- XML 変換
- サンプルの XQuery 変換コード

7.1 XML 変換

プログラマは、各種の XML 構造間の変換タスクに直面することがよくあります。このような変換は、要素名変更などの非常にシンプルな変換から、XML 構造の再形成や他のドキュメントまたはソースのコンテンツとの結合などの極めて複雑な変換まで、多岐に渡ります。このセクションでは、XML 変換のいくつかの側面について説明します。このセクションは、次の部分から構成されます。

- XQuery と XSLT
- XHTML または XSL-FO への変換
- typeswitch 式

7.1.1 XQuery と XSLT

変換では、一般に XSLT が使用され、多くの変換で適切に機能します。ただし、一部の変換タイプの場合は難点があります。特に、変換が大規模な XQuery アプリケーションの一部である場合が該当します。

XQuery は強力なプログラミング言語であり、MarkLogic サーバーは、コンテンツへの高速アクセスを提供します。これらを組み合わせることで、変換を非常に効率よく処理できます。MarkLogic サーバーは、特に、変換を必要とするコンテンツを取得するための検索を必要とする変換に適しています。例えば、元の XML 値に置換する値を取得するためにレキシコンルックアップを使用する変換が挙げられます。また、特定のコレクション内で作成者の数を数える必要のある変換も考えられます。

7.1.2 XHTML または XSL-FO への変換

一般的な XML 変換では、ドキュメントをプロプライエタリの XML 構造から HTML に変換します。XQuery は XML を生成するため、XHTML（HTML の XML バージョン）を返す XQuery プログラムを記述することは非常に簡単です。XHTML は、その大部分が整形式の HTML であり、小文字のタグ名と属性名を使用します。そのため、XHTML を返す XQuery プログラムを記述することが一般的です。

同様に、XSL-FO を返す XQuery プログラムを記述できます。XSL-FO は、PDF 出力をビルドするための共通パスです。また、XSL-FO も単なる XML 構造であるため、その構造で XML を返す XQuery は簡単に記述できます。

7.1.3 typeswitch 式

XQuery で変換を実行する方法は他にもありますが、再帰関数で使用される `typeswitch` 式は、簡便でパフォーマンスが高く、変換コードを変更および保守しやすいデザインパターンです。

`typeswitch` 式のシンタックスについては、『*XQuery and XSLT Reference Guide*』の The typeswitch Expression を参照してください。`case` 節を使用すると、`typeswitch` への入力をテストし、何らかを返すことができます。変換では、多くの場合、「kind テスト」と呼ばれるテストが実行されます。kind テストは、あるもの（指定された QName を持つ要素ノードなど）が、どの種類のノードであるかを調べるために実行します。そのテストが true を返した場合、`return` 節内のコードが実行されます。`return` 節は、任意の XQuery にすることができるため、関数を呼び出すことができます。

XML は順序付きツリー構造であるため、XML ノード全体を再帰的に調べる関数を作成できます。この関数は、調べるたびにノードで何らかの変換を実行し、その子ノードを関数に返送します。その結果、XML ノードの構造やコンテンツを変換する便利なメカニズムが得られます。

7.2 サンプルの XQuery 変換コード

このセクションでは、`typeswitch` 式を使用するコード例をいくつか示します。各サンプルでは、コードをカット＆ペーストし、アプリケーションサーバーに対して実行できます。この手法に関する複雑な例については、developer.marklogic.com/code にある Shakespeare デモアプリケーションを参照してください。

内容は、次のとおりです。

- シンプルな例
- cts:highlight を使用するシンプルな例
- XHTML へのサンプル変換
- typeswitch デザインパターンの拡張

7.2.1 シンプルな例

次のサンプルコードは、入力ノードの簡単な変換を実行しますが、typeswitch 式の default 節でシンプルな関数を呼び出して子ノードを元の関数に返送するという基本的なデザインパターンを示しています。

```
xquery version "1.0-ml";

(: This is the recursive typeswitch function :)
declare function local:transform($nodes as node()*)
as node()*
{
for $n in $nodes return
typeswitch ($n)
  case text() return $n
  case element (bar) return <barr>{local:transform
  ($n/node())}</barr>
  case element (baz) return <bazz>{local:transform
  ($n/node())}</bazz>
  case element (buzz) return
     <buzzz>{local:transform($n/node())}</buzzz>
  case element (foo) return <fooo>{local:transform
  ($n/node())}</fooo>
  default return <temp>{local:transform($n/node())}</temp>
};

let $x :=
<foo>foo
  <bar>bar</bar>
  <baz>baz
    <buzz>buzz</buzz>
  </baz>
  foo
</foo>
return
local:transform($x)
```

この XQuery プログラムは次のような内容を返します。

```
<fooo>
  foo
  <barr>bar</barr>
  <bazz>baz
    <buzzz>buzz</buzzz>
  </bazz>
  foo
</fooo>
```

7.2.2 cts:highlight を使用するシンプルな例

次のサンプルコードは前の例と同じですが、変換の結果に対して cts:highlight も実行します。この方法で cts:highlight を使用すると、検索の結果を表示し、cts:query 式にマッチする語句を強調表示するときに便利である場合があります。cts:highlight の詳細については、『*Search Developer's Guide*』の Highlighting Search Term Matches を参照してください。

```
xquery version "1.0-ml";

(: This is the recursive typeswitch function :)
declare function local:transform($nodes as node()*)
as node()*
{
for $n in $nodes return
typeswitch ($n)
  case text() return $n
  case element (bar) return <barr>{local:transform
  ($n/node())}</barr>
  case element (baz) return <bazz>{local:transform
  ($n/node())}</bazz>
  case element (buzz) return
     <buzzz>{local:transform($n/node())}</buzzz>
  case element (foo) return <fooo>{local:transform
  ($n/node())}</fooo>
  default return <booo>{local:transform($n/node())}</booo>
};

let $x :=
<foo>foo
  <bar>bar</bar>
  <baz>baz
    <buzz>buzz</buzz>
  </baz>
  foo
</foo>
return
cts:highlight(local:transform($x), cts:word-query("foo"),
   <b>{$cts:text}</b>)
```

この XQuery プログラムは次のような内容を返します。

```
<fooo>
  <b>foo</b>
  <barr>bar</barr>
  <bazz>baz
    <buzzz>buzz</buzzz>
  </bazz>
  <b>foo</b>
</fooo>
```

7.2.3 XHTML へのサンプル変換

次のサンプルコードは、XML 構造から XHTML への非常にシンプルな変換を実行します。前の例と同じデザインパターンを使用しますが、ここでは、XQuery コードに HTML マークアップが含まれます。

```
xquery version "1.0-ml";
declare default element namespace
"http://www.w3.org/1999/xhtml";

(: This is the recursive typeswitch function :)
declare function local:transform($nodes as node()*)
as node()*
{
for $n in $nodes return
typeswitch ($n)
  case text() return $n
  case element (a) return local:transform($n/node())
  case element (title) return <h1>{local:transform
  ($n/node())}</h1>
  case element (para) return <p>{local:transform
  ($n/node())}</p>
  case element (sectionTitle) return
      <h2>{local:transform($n/node())}</h2>
  case element (numbered) return <ol>{local:transform
  ($n/node())}</ol>
  case element (number) return <li>{local:transform
  ($n/node())}</li>
  default return <tempnode>{local:transform
  ($n/node())}</tempnode>
};
```

```
let $x :=
<a>
 <title>This is a Title</title>
 <para>Some words are here.</para>
 <sectionTitle>A Section</sectionTitle>
 <para>This is a numbered list.</para>
 <numbered>
   <number>Install MarkLogic Server.</number>
   <number>Load content.</number>
   <number>Run very big and fast XQuery.</number>
 </numbered>
</a>
return
<html xmlns="http://www.w3.org/1999/xhtml">
<head><title>MarkLogic Sample Code</title></head>
<body>{local:transform($x)}</body>
</html>
```

これは、次の XHTML コードを返します。

```
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>MarkLogic Sample Code</title>
  </head>
<body>
<h1>This is a Title</h1>
<p>Some words are here.</p>
<h2>A Section</h2>
<p>This is a numbered list.</p>
  <ol>
      <li>Install MarkLogic Server.</li>
      <li>Load content.</li>
      <li>Run very big and fast XQuery.</li>
  </ol>
</body>
</html>
```

HTTP アプリケーションサーバーに対してこのコードを実行すると（例えば、アプリケーションサーバールートにあるファイルにこのコードをコピーし、ブラウザでそのページにアクセスすると）、次のような結果が表示されます。

この例の `typeswitch case` ステートメントの `return` 節は単純化されており、次のようになっています。

```
case element (sectionTitle) return
<h2>{local:passthru($x)}</h2>
```

より一般的な例では、`return` 節で関数を呼び出します。

```
case element (sectionTitle) return local:myFunction($x)
```

その後、この関数は、任意の複雑なロジックを実行できます。通常、各 case ステートメントは、その要素に必要な変換方法に適したコードを使用して関数を呼び出します。

7.2.4 typeswitch デザインパターンの拡張

上記のシンプルな例の他に、このデザインパターンを拡張できる方法は多数あります。例えば、前の例に示したシンプルな `transform` 関数に、2 番目のパラメータを追加できます。2 番目のパラメータは、変換しているノードに関するその他の情報を渡します。

例えば、XML 階層内の要素の出現位置に基づいて特定の要素を除外する変換が必要だとします。この場合、次のコードの一部分に示すように、渡された要素を除外するロジックを関数に追加できます。

```
declare function transform($nodes as node()*, $excluded
as element()*)
  as node()*
{
```

```
(: Test whether each node in $nodes is an excluded element,
   if so return empty, otherwise run the typeswitch
   expression.
:)
for $n in $nodes return
if ( some $node in $excluded satisfies $n )
then ( )
else ( typeswitch ($n) .....)
};
```

このデザインパターンに対して使用できる拡張は、他にも多数あります。拡張の使用手順は、アプリケーション要件に応じて異なります。XQuery は強力なプログラミング言語であり、このようなタイプのデザインパターンは、新しい要件に対する非常に高い拡張性を備えています。

8.0 ドキュメントとディレクトリのロック

この章では、ドキュメントとディレクトリのロックについて説明します。次のセクションから構成されます。

- ロックの概要
- ロック API
- 例：ロックが設定されたドキュメントの URI 検索
- 例：ドキュメントへのロックの設定
- 例：ドキュメントのロックの解除
- 例：ロックの所有者であるユーザーの検索

注： この章は、明示的に設定するドキュメントとディレクトリのロックについて説明するものであり、MarkLogic によって暗黙で設定されるトランザクションロックに関するものではありません。トランザクションについては、「MarkLogic サーバーのトランザクションとは」（22 ページ）を参照してください。

8.1 ロックの概要

それぞれのドキュメントとディレクトリには「ロック」を設定できます。ロックは、ロックドキュメントとして MarkLogic サーバーデータベース内に格納されます。ロックドキュメントは、関連付けられているドキュメントやディレクトリからは分離されています。ロックには次のような特性があります。

- 書き込みロック
- 永続的
- 検索可能
- 排他または共有
- 階層型
- ロックと WebDAV
- ロックの他の使用方法

8.1.1 書き込みロック

ロックとは書き込みロックであり、ロックを持たないすべてのユーザーによる更新を制限します。あるユーザーが排他的ロックを持つ場合、他のユーザーはロックを取得できず、そのドキュメントを更新することも削除することもできません。ロックが設定されたドキュメントを更新または削除しようとすると、エラーが発生します。ただし、他のユーザーは、ロックが設定されたドキュメントを読み取ることはできます。

8.1.2 永続的

ロックは、データベース内で永続的です。トランザクションには関連付けられていません。ロックは、指定した期間だけ存続するように設定することも、無期限に存続するように設定することもできます。永続的であるため、ロックを使用して、マルチトランザクション操作中にドキュメントが変更されないようにすることができます。

8.1.3 検索可能

ロックは永続的な XML ドキュメントであるため、検索可能な XML ドキュメントであり、データベース内のロックに関する情報を取得するためのクエリを記述できます。例については、「例：ロックが設定されたドキュメントの URI 検索」（125 ページ）を参照してください。

8.1.4 排他または共有

ロックは `exclusive` として設定できます。この場合、ロックを設定したユーザーのみが、関連付けられたデータベースオブジェクト（ドキュメント、ディレクトリ、またはコレクション）を更新できます。また、`shared` として設定することもできます。この場合、他のユーザーがデータベースオブジェクトの共有ロックを取得できます。オブジェクトの `shared` ロックを取得したユーザーは、そのオブジェクトを更新できます。

8.1.5 階層型

ディレクトリをロックするときに、ロックするディレクトリ階層構造の深さを指定できます。「`0`」を指定すると、指定した URI のみがロックされ、「`infinity`」を指定すると、URI（ディレクトリなど）とそのすべての子がロックされます。

8.1.6 ロックと WebDAV

WebDAV クライアントは、ロックを使用して、ドキュメントやディレクトリをロックしてから更新します。ロックすることで、ドキュメントの保存中に他のクライアントがドキュメントを変更できないようにします。ロックの設定方法は、WebDAV クライアントの実装により異なります。一部のクライアントは、一定の期間後は無効になるようにロックを設定します。また、ドキュメントのロックを明示的に解除するまで存続するように設定するクライアントもあります。

8.1.7 ロックの他の使用方法

どのアプリケーションも、更新戦略の一環としてロックを使用できます。例えば、ドキュメントやディレクトリの更新を実行する前に 30 秒間のロックを開発者が設定するというポリシーを策定できます。ロックは柔軟性が非常に高いため、環境に応じたポリシーを設定することも、ポリシーをまったく使用しないようにすることもできます。

データベース内のすべてのドキュメントとディレクトリにロックを設定した場合、データベース内のあらゆるデータの変更の禁止（ロックを所有しているユーザーによる変更は除く）という効果が得られることがあります。ロックのアプリケーション開発プラクティスとセキュリティパーミッションの使用を効果的に組み合わせることで、堅牢なマルチユーザー開発環境を実現することができます。

8.2 ロック API

ロック用の API は基本的に 2 種類です。ロックを表示する API と、ロックを設定または削除する API です。これらの API の詳細なシンタックスについては、オンラインの『XQuery Built-In and Module Function Reference』を参照してください。

ロックを表示する API は次のとおりです。

- `xdmp:document-locks`
- `xdmp:directory-locks`
- `xdmp:collection-locks`

引数が指定されていない `xdmp:document-locks` 関数は、ドキュメントロックごとに 1 つずつロックのシーケンスを返します。引数として URI のシーケンスが指定された `xdmp:document-locks` 関数は、指定されたドキュメントのロックを返します。`xdmp:directory-locks` 関数は、指定されたディレクトリ内にあるすべてのドキュメントロックを返します。`xdmp:collection-locks` 関数は、指定されたコレクション内にあるすべてのドキュメントロックを返します。

ディレクトリやドキュメントのロックを設定および削除する場合は、次の関数を使用できます。

- `xdmp:lock-acquire`
- `xdmp:lock-release`

ドキュメントやディレクトリにロックを設定する基本的な手順では、`xdmp:lock-acquire` 関数を使用したクエリを送信します。このクエリでは、URI、リクエストされたロック（`exclusive` または `shared`）の範囲、ロックの影響を受ける階層（その URI のみ、またはその URI とすべての子）、ロックの所有者、ロックの期間を指定します。

> **注：** ロックの `owner` は、ロックの `sec:user-id` と同じではありません。`owner` は、`xdmp:lock-acquire` のオプションとして指定できます。所有者を明示的に指定しない場合は、ロックコマンドを発行したユーザーの名前が、デフォルトの所有者として設定されます。例については、「例：ロックの所有者であるユーザーの検索」（127 ページ）を参照してください。

8.3 例：ロックが設定されたドキュメントの URI 検索

ロックドキュメントで XQuery ビルトイン `xdmp:node-uri` 関数を呼び出すと、ロックされているドキュメントの UR が返されます。次のクエリは、データベース内でロックの設定されているすべてのドキュメントの URI のリストを含むドキュメントを返します。

```
<root>
{
for $locks in xdmp:document-locks()
return <document-URI>{xdmp:node-uri($locks)}
</document-URI>
}
</root>
```

例えば、データベース内でロックの設定された唯一のドキュメントの URI が /document/myDocument.xml である場合、上記のクエリは次のように返します。

```
<root>
  <document-URI>/documents/myDocument.xml</document-URI>
</root>
```

8.4 例：ドキュメントへのロックの設定

次の例では、xdmp:lock-acquire 関数を使用して、指定された URI のドキュメントに 2 分間（120 秒）のロックを設定します。

```
xdmp:lock-acquire("/documents/myDocument.xml",
                  "exclusive",
                  "0",
                  "Raymond is editing this document",
                  xs:unsignedLong(120))
```

次のように xdmp:document-locks 関数を使用すると、結果のロックドキュメントを表示できます。

```
xdmp:document-locks("/documents/myDocument.xml")

=>

<lock:lock xmlns:lock="http://marklogic.com/xdmp/lock">
  <lock:lock-type>write</lock:lock-type>
  <lock:lock-scope>exclusive</lock:lock-scope>
  <lock:active-locks>
    <lock:active-lock>
      <lock:depth>0</lock:depth>
      <lock:owner>Raymond is editing this document
      </lock:owner>
      <lock:timeout>120</lock:timeout>
      <lock:lock-token>
       http://marklogic.com/xdmp/locks/4d0244560cc3726c
      </lock:lock-token>
```

```
      <lock:timestamp>1121722103</lock:timestamp>
      <sec:user-id
xmlns:sec="http://marklogic.com/xdmp/security">
         8216129598321388485
      </sec:user-id>
    </lock:active-lock>
  </lock:active-locks>
</lock:lock>
```

8.5 例：ドキュメントのロックの解除

次の例では、xdmp:lock-release 関数を使用して、ドキュメントのロックを明示的に解除します。

```
xdmp:lock-release("/documents/myDocument.xml")
```

タイムアウト期間のないロックを取得する場合は、処理が完了したときに、必ずロックを解除してください。ロックを解除しないと、他のユーザーが、xdmp:lock-acquire アクションによってロックされたドキュメントやディレクトリを更新できません。

8.6 例：ロックの所有者であるユーザーの検索

ロックはドキュメントであるため、ロックの所有者であるユーザーを検索するクエリを記述できます。例えば、次のクエリは、ロックドキュメントの sec:user-id 要素を検索し、各ロックの所有者ユーザーのユーザー ID と URI 名の集まりを返します。

```
for $x in xdmp:document-locks()//sec:user-id
return  <lock>
           <URI>{xdmp:node-uri($x)}</URI>
           <user-id>{data($x)}</user-id>
        </lock>
```

結果の例を、以下に示します（この結果では、データベース内のロックは 1 つのみと想定しています）。

```
<lock>
  <URI>/documents/myDocument.xml</URI>
  <user-id>15025067637711025979</user-id>
</lock>
```

9.0 プロパティドキュメントとディレクトリ

この章では、MarkLogic サーバーのプロパティドキュメントとディレクトリについて説明します。以下のセクションで構成されています。

- プロパティドキュメント
- ドキュメント処理のためのプロパティの使用
- ディレクトリ
- プロパティやディレクトリのパーミッション
- 例：ディレクトリおよびドキュメントブラウザ

9.1 プロパティドキュメント

「プロパティドキュメント」は、データベース内のドキュメントと同じ URI を共有する XML ドキュメントです。それぞれのドキュメントには、対応するプロパティドキュメントが存在する可能性がありますが、プロパティドキュメントは、プロパティが作成される場合にのみ作成されます。一般に、プロパティドキュメントは、対応するドキュメントに関連するメタデータを格納するために使用されますが、プロパティドキュメントのスキーマに準拠している限り、どの XML データもプロパティドキュメントに格納できます。通常、ドキュメントは、プロパティドキュメントを作成するために特定の URI に存在しますが、ドキュメントの作成とプロパティの追加を単一のトランザクションで実行できます。また、ドキュメントが存在しないプロパティを作成することもできます。プロパティドキュメントは、対応するドキュメントとは別のフラグメントに格納されます。このセクションでは、プロパティドキュメント、およびプロパティドキュメントにアクセスするための API について説明します。このセクションは、次のサブセクションで構成されます。

- プロパティドキュメントの名前空間とスキーマ
- プロパティドキュメントの API
- XPath のプロパティ軸
- 保護されているプロパティ
- プロパティドキュメント要素に対する要素インデックスの作成
- サンプルのプロパティドキュメント
- スタンドアロンのプロパティドキュメント

9.1.1 プロパティドキュメントの名前空間とスキーマ

プロパティドキュメントは、`properties.xsd` スキーマに準拠する必要がある XML ドキュメントです。`properties.xsd` スキーマは、インストール時に `<install_dir>/Config` ディレクトリにコピーされます。

プロパティスキーマには、prop 名前空間プレフィックスが割り当てられます。このプレフィックスはサーバーで事前定義されています。

```
http://marklogic.com/xdmp/property
```

properties.xsd スキーマを次に示します。

```
<xs:schema targetNamespace="http://marklogic.com/xdmp/
property"
 xsi:schemaLocation="http://www.w3.org/2001/XMLSchema
                     XMLSchema.xsd
                     http://marklogic.com/xdmp/security
                     security.xsd"
 xmlns="http://marklogic.com/xdmp/property"
 xmlns:xs="http://www.w3.org/2001/XMLSchema"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xhtml="http://www.w3.org/1999/xhtml"
 xmlns:sec="http://marklogic.com/xdmp/security">

  <xs:complexType name="properties">
    <xs:annotation>
      <xs:documentation>
        A set of document properties.
      </xs:documentation>
      <xs:appinfo>
      </xs:appinfo>
    </xs:annotation>
    <xs:choice minOccurs="1" maxOccurs="unbounded">
      <xs:any/>
    </xs:choice>
  </xs:complexType>

  <xs:element name="properties" type="properties">
    <xs:annotation>
      <xs:documentation>
        The container for properties.
      </xs:documentation>
      <xs:appinfo>
      </xs:appinfo>
    </xs:annotation>
  </xs:element>

  <xs:simpleType name="directory">
    <xs:annotation>
      <xs:documentation>
        A directory indicator.
```

```
        </xs:documentation>
        <xs:appinfo>
        </xs:appinfo>
      </xs:annotation>
      <xs:restriction base="xs:anySimpleType">
      </xs:restriction>
    </xs:simpleType>

    <xs:element name="directory" type="directory">
      <xs:annotation>
        <xs:documentation>
          The indicator for a directory.
        </xs:documentation>
        <xs:appinfo>
        </xs:appinfo>
      </xs:annotation>
    </xs:element>

    <xs:element name="last-modified" type="last-modified">
      <xs:annotation>
        <xs:documentation>
          The timestamp of last document modification.
        </xs:documentation>
        <xs:appinfo>
        </xs:appinfo>
      </xs:annotation>
    </xs:element>

    <xs:simpleType name="last-modified">
      <xs:annotation>
        <xs:documentation>
          A timestamp of the last time something
          was modified.
        </xs:documentation>
        <xs:appinfo>
        </xs:appinfo>
      </xs:annotation>
      <xs:restriction base="xs:dateTime">
      </xs:restriction>
    </xs:simpleType>

  </xs:schema>
```

9.1.2 プロパティドキュメントの API

プロパティドキュメントの API は、プロパティドキュメントでプロパティのリストアップ、追加、および設定するために使用できる XQuery 関数です。プロパティ API を使用すると、プロパティドキュメントの最上位要素にアクセスできます。プロパティは XML 要素であるため、XPath を使用すると、最上位プロパティ要素の任意の子や子孫にナビゲートできます。プロパティドキュメントは、対応するドキュメントに関連付けられており、URI を共有しています。ドキュメントを削除すると、そのプロパティドキュメントも削除されます。

プロパティドキュメントのアクセスおよび操作に次の API を使用できます。

- `xdmp:document-properties`
- `xdmp:document-add-properties`
- `xdmp:document-set-properties`
- `xdmp:document-set-property`
- `xdmp:document-remove-properties`
- `xdmp:document-get-properties`
- `xdmp:collection-properties`
- `xdmp:directory`
- `xdmp:directory-properties`

これらの API のシグネチャおよび説明については、『*MarkLogic XQuery and XSLT Function Reference*』を参照してください。

9.1.3 XPath のプロパティ軸

MarkLogic には、「プロパティ軸」を含む拡張 XPath（XQuery と XSLT の両方で使用可能）があります。プロパティ軸（`property::`）を使用すると、プロパティドキュメントの項目で、特定の URI を検索する XPath 式を記述できます。これらの式を使用して、ドキュメントおよびプロパティ軸で結合を実行できます。これは、プロパティにドキュメントの状態情報を格納するときに便利です。このようなアプローチの詳細については、「ドキュメント処理のためのプロパティの使用」（134 ページ）を参照してください。

プロパティ軸は、XPath 式の順方向軸や逆方向軸に似ています。例えば、`child::` 順方向軸を使用すると、ドキュメント内で子要素にトラバースできます。XPath 軸の詳細については、XPath 2.0 仕様、および『*XQuery and XSLT Reference Guide*』の XPath Quick Reference を参照してください。

プロパティ軸には、特定の URI のプロパティドキュメントノードの子がすべて含まれます。

次の例は、ドキュメントをクエリするときにプロパティ軸を使用してドキュメントのプロパティにアクセスする方法を示しています。

次のようにしてテストドキュメントを作成します。

```
xdmp:document-insert("/test/123.xml",
  <test>
    <element>123</element>
  </test>)
```

/test/123.xml ドキュメントのプロパティドキュメントにプロパティを追加します。

```
xdmp:document-add-properties("/test/123.xml",
  <hello>hello there</hello>)
```

/test/123.xml ドキュメントのプロパティをリストアップすると、直前に追加したプロパティが表示されます。

```
xdmp:document-properties("/test/123.xml")
=>
<prop:properties
xmlns:prop="http://marklogic.com/xdmp/property">
  <hello>hello there</hello>
</prop:properties>
```

これで、次のようにして /test/123.xml ドキュメントのプロパティ軸で検索できます。

```
doc("test/123.xml")/property::hello
=>
<hello>hello there</hello>
```

9.1.4 保護されているプロパティ

次のプロパティは保護されており、作成または変更できるのはシステムのみです。

- prop:directory
- prop:last-modified

これらのプロパティは、MarkLogic サーバーが使用するために直接予約されています。これらの名前のプロパティを追加または削除しようとすると、失敗して例外が発生します。

9.1.5 プロパティドキュメント要素に対する要素インデックスの作成

プロパティドキュメントは XML ドキュメントであるため、プロパティドキュメント内の要素に対して要素（レンジ）インデックスを作成できます。例えば、プロパティを使用して、プロパティドキュメントが対応するドキュメントに関する数値または日付のメタデータを格納する場合に、要素インデックスを作成して、メタデータにアクセスするクエリを高速化できます。

9.1.6 サンプルのプロパティドキュメント

プロパティドキュメントは、「プロパティドキュメントの名前空間とスキーマ」（128 ページ）で説明するスキーマに準拠する XML ドキュメントです。プロパティドキュメントのコンテンツをリストアップするには、`xdmp:document-properties` 関数を使用します。指定した URI にプロパティドキュメントが存在しない場合、この関数は空のシーケンスを返します。ディレクトリのプロパティドキュメントには、空の `prop:directory` 要素が 1 つあります。例えば、URI `http://myDirectory/` にディレクトリが存在する場合、`xdmp:document-properties` コマンドは次のようにプロパティドキュメントを返します。

```
xdmp:document-properties("http://myDirectory/")
=>
<prop:properties xmlns:prop="http://marklogic.com/xdmp/
property">
  <prop:directory/>
</prop:properties>
```

プロパティドキュメントには任意の内容を追加できます（プロパティスキーマに準拠している限り）。関数 `xdmp:document-properties` を引数なしで実行すると、データベース内にあるすべてのプロパティドキュメントのシーケンスが返されます。

9.1.7 スタンドアロンのプロパティドキュメント

通常、プロパティドキュメントは、対応ドキュメントとともに作成され、このドキュメントと URI を共有しています。ただし、同じ URI の対応ドキュメントなしで、特定の URI のプロパティドキュメントを作成できます。そのようなプロパティドキュメントは、「スタンドアロンのプロパティドキュメント」と呼ばれます。スタンドアロンのプロパティドキュメントを作成するには、`xdmp:document-add-properties` または `xdmp:document-set-properties` API を使用し、必要に応じて、プロパティドキュメントのパーミッション、コレクション、クオリティなどを設定する `xdmp:document-set-permissions`、`xdmp:document-set-collections`、`xdmp:document-set-quality` などの API を追加します。

次の例では、プロパティドキュメントを作成してパーミッションを設定します。

```
xquery version "1.0-ml";
```

```
xdmp:document-set-properties("/my-props.xml",
<my-props/>),
xdmp:document-set-permissions("/my-props.xml",
   (xdmp:permission("dls-user", "read"),
    xdmp:permission("dls-user", "update")))
```

そして、その URI で `xdmp:document-properties` を実行すると、新しいプロパティドキュメントが返されます。

```
xquery version "1.0-ml";

xdmp:document-properties("/my-props.xml")
(: returns:
<?xml version="1.0" encoding="ASCII"?>
<prop:properties xmlns:prop="http://marklogic.com/xdmp/
property">
  <my-props/>
  <prop:last-modified>2010-06-18T18:19:10-07:00
  </prop:last-modified>
</prop:properties>
:)
```

同様に、スタンドアロンのプロパティドキュメントの作成時または作成後に、関数を渡して、そのプロパティドキュメントのコレクションやクオリティを設定できます。

9.2 ドキュメント処理のためのプロパティの使用

大量のドキュメントを更新する必要がある場合、特にマルチステッププロセスでは、各ドキュメントの現在の状態の追跡が必要になることがよくあります。例えば、3 つのステップで数百万ものドキュメントを更新するコンテンツ処理アプリケーションがある場合は、まったく処理されていないドキュメント、ステップ 1 で処理が完了したドキュメント、ステップ 2 で処理が完了したドキュメントなどをプログラムを使用して判断する方法が必要です。

このセクションでは、ドキュメント処理パイプラインで使用するメタデータを格納するためにプロパティを使用する方法を説明します。このセクションは、次のサブセクションで構成されています。

- プロパティ軸の使用によるドキュメント状態の判断
- ドキュメント処理に関する問題
- ドキュメント処理のソリューション
- モジュールを実行するための基本コマンド

9.2.1 プロパティ軸の使用によるドキュメント状態の判断

プロパティドキュメントを使用して、マルチステップ処理の対象であるドキュメントの状態情報を格納できます。プロパティドキュメント間の結合によって、処理されたドキュメントと処理されていないドキュメントを特定できます。このような結合を実行するクエリは、`property::` 軸を使用します（詳細については、「XPath のプロパティ軸」（131 ページ）を参照）。

プロパティ軸間の結合に述語がある場合は、パフォーマンスを上げるために結合が最適化されます。次の例は、プロパティ `bar` を持つドキュメントから `foo` ルート要素を返します。

```
foo[property::bar]
```

パフォーマンスを上げるために最適化されるクエリの種類の例を次に示します（`/a/b/c` は何らかの XPath 式）。

- 述語内のプロパティ軸：

  ```
  /a/b/c[property::bar]
  ```

- プロパティ軸の否定テスト：

  ```
  /a/b/c[not(property::bar = "baz")]
  ```

- `property` 述語の後に続くパス式：

  ```
  /a/b/c[property::bar and bob = 5]/d/e
  ```

- 等価な FLWOR 式

  ```
  for $f in /a/b/c
  where $f/property::bar = "baz"
  return $f
  ```

次のように、他の種類の式も使用できますが、パフォーマンスを上げるために最適化されません。

- ルート要素が `foo` であるドキュメントの bar プロパティが必要な場合：

  ```
  /foo/property::bar
  ```

9.2.2 ドキュメント処理に関する問題

このセクションで説明するアプローチは、次のような状況に適しています。

- 「すでに 100 万のドキュメントを読み込んでいるので、それらをすべて更新したい。」この場合の疑似コードは次のようになります。

```
for $d in fn:doc()
return some-update($d)
```

  このようなタイプのクエリは、最終的にツリーキャッシュメモリが不足して失敗します。

- 次の形式の反復的な呼び出しが次第に低速になる場合：

```
for $d in fn:doc()[k to k+10000]
return some-update($d)
```

このようなシナリオでは、プロパティを使用して、ドキュメントの処理が必要かどうかをテストすると、更新を管理可能な塊（チャンク）に効果的にまとめることができます。

9.2.3 ドキュメント処理のソリューション

このコンテンツ処理手法は、さまざまな状況で使用できます。このアプローチは次の要件を満たします。

- 大規模な既存データセットを操作する。
- データセットを読み込む前に、後でさらに処理が必要になることを認識している必要はない。
- データが到着し続ける（新しいデータが毎日追加されるなど）ような状況で、このアプローチが有効である。
- 最終的に、定常状態である「コンテンツ処理」可能環境へと遷移できることが必要である。

ドキュメント処理アプローチの基本手順を次に示します。

1. 反復的な戦略を採用します。ただし、次第に低速にならないものにします。
2. 再処理アクティビティを複数の更新に分割します。
3. プロパティ（または、プロパティがないこと）で、処理が（引き続き）必要なドキュメントを識別します。
4. 同じモジュールを繰り返し呼び出して、プロパティだけでなく、ドキュメントも更新します。

```
for $p in fn:doc()/root[not(property::some-update)]
[1 to 10000]
return some-update($d)
```

5. 引き続き処理が必要なドキュメントがある場合は、モジュールを再度呼び出します。

6. 特定のプロパティを持たないドキュメントを処理するモジュールの疑似コードを次に示します。

```
let $docs := get n documents that have no properties
return
for $processDoc in $docs
return if (empty $processDoc)
       then ()
       else ( process-document($processDoc),
              update-property($processDoc) )
,
xdmp:spawn(process_module)
```

この疑似コードは次の処理を実行します。

- 特定のプロパティを持たないドキュメントの URI を取得します。
- URI ごとに、特定のプロパティが存在するかどうかを確認します。
- そのプロパティが存在する場合は、ドキュメントに対して何も実行しません（すでに更新されているため）。
- そのプロパティが存在しない場合は、ドキュメントに対する更新とプロパティに対する更新を実行します。
- これをすべての URI に対して継続します。
- すべての URI が処理されたら、モジュールを再度呼び出して、新しいドキュメント（プロパティのないもの）があれば取得します。

7. （省略可能）Content Processing Pipeline プロセスを自動化します。

9.2.4 モジュールを実行するための基本コマンド

自動化されたコンテンツ処理を実行するには、次のビルトイン関数が必要です。

- タスクサーバーのキューにモジュールを入れるには、次の手順に従います。

```
xdmp:spawn($database, $root, $path)
```

- モジュール全体を評価するには、次の手順に従います（xdmp:eval に似ているが対象はモジュール）。

```
xdmp:invoke($path, $external-vars)

xdmp:invoke-in($path, $database-id, $external-vars)
```

9.3 ディレクトリ

ディレクトリには多くの用途があります。ドキュメント URI の整理や、WebDAV サーバーでのドキュメント URI の使用などです。このセクションは、ディレクトリに関する次の内容で構成されます。

- プロパティとディレクトリ
- ディレクトリと WebDAV サーバー
- ディレクトリとコレクション

9.3.1 プロパティとディレクトリ

ディレクトリを作成すると、MarkLogic サーバーによって prop:directory 要素を持つプロパティドキュメントが作成されます。ディレクトリに対応する URI で xdmp:document-properties コマンドを実行すると、次の例に示すように、空の prop:directory 要素を持つプロパティドキュメントが返されます。

```
xdmp:directory-create("/myDirectory/");

xdmp:document-properties("/myDirectory/")
=>
<prop:properties xmlns:prop="http://marklogic.com/xdmp/
property">
  <prop:directory/>
</prop:properties>
```

注： 一意の URI でディレクトリを作成できますが、ディレクトリ URI の末尾はフォワードスラッシュ（/）であるという規則があります。ディレクトリと同じ URI でドキュメントを作成できますが、これは推奨されません。ベストプラスティスは、末尾がスラッシュの URI をディレクトリ用に予約することです。

xdmp:document-properties を引数なしで実行すると、データベース内にあるすべてのプロパティドキュメントに関するプロパティドキュメントが返されます。各ディレクトリには prop:directory 要素があるため、データベース内にあるすべてのディレクトリを返すクエリを簡単に記述できます。これを行うには、次のように xdmp:node-uri 関数を使用します。

```
xquery version "1.0-ml";

for $x in
xdmp:document-properties()/prop:properties/prop:directory
return <directory-uri>{xdmp:node-uri($x)}</directory-uri>
```

9.3.2 ディレクトリと WebDAV サーバー

ディレクトリは、WebDAV サーバーで使用するために必要です。WebDAV クライアントからアクセスできるドキュメントを作成するには、親ディレクトリが存在する必要があります。ドキュメントの親ディレクトリは、URI がドキュメントのプレフィックスであるディレクトリです（例えば、URI `http://myserver/doc.xml` のディレクトリは `http://myserver/` です）。WebDAV サーバーでデータベースを使用するときは、データベース設定の `directory creation` 設定が `automatic`（デフォルト設定）に設定されていることを確認してください。これにより、ドキュメントが作成されるときに親ディレクトリが作成されます。WebDAV サーバーでディレクトリを使用する方法については、『*Administrator's Guide*』の WebDAV Servers を参照してください。

9.3.3 ディレクトリとコレクション

データベース内のドキュメントの整理には、ディレクトリならびにコレクションが利用できます。ディレクトリとコレクションには、次のような重要な違いがあります。

- ディレクトリは階層型（ファイルシステムのディレクトリと同様）です。コレクションではこの必要はありません。このため、ディレクトリ URI には親ディレクトリが含まれている必要があります。コレクション URI は、このコレクションに含まれるドキュメントと何の関係がなくてもよいです。例えば、`http://marklogic.com/a/b/c/d/e/`（`http://marklogic.com/` がルート）という名前のディレクトリには、親ディレクトリ `d`、`c`、`b`、`a` が存在している必要があります。コレクションの場合、どのドキュメントも（その URI に関係なく）、任意の URI を持つコレクションに属することができます。
- ディレクトリは、WebDAV クライアントでドキュメントを表示させる際に必要です。つまり、WebDAV サーバで、ルートが `/a/b/` である URI `/a/b/hello/goodbye` を持つドキュメントを表示するには、次のような URI を持つディレクトリがデータベース内に存在している必要があります。

  ```
  /a/b/

  /a/b/hello/
  ```

ドキュメントの整理には、ディレクトリとコレクションの両方を使用できますが、ディレクトリはコレクションとは無関係です。コレクションの詳細については、『*Search Developer's Guide*』の Collections を参照してください。WebDAV サーバーの詳細については、『*Administrator's Guide*』の WebDAV Servers を参照してください。

9.4 プロパティやディレクトリのパーミッション

MarkLogic サーバーデータベース内のあらゆるドキュメントと同様に、プロパティドキュメントにもパーミッションを設定できます。ディレクトリにはプロパティドキュメント（および空の prop:directory 要素）があるため、ディレクトリにもパーミッションを設定できます。プロパティドキュメントへのパーミッションは、それに対応するドキュメントに設定されたパーミッションと同じです。これらのパーミッションは、xdmp:document-get-permissions 関数を使って一覧表示できます。同様に、ディレクトリに設定されたパーミッションを、xdmp:document-get-permissions 関数を使って一覧表示できます。パーミッションおよびセキュリティの詳細については、『*Security Guide*』を参照してください。

9.5 例：ディレクトリおよびドキュメントブラウザ

プロパティドキュメントを使用して、ある URI 下にあるドキュメントおよびディレクトリをリストアップするシンプルなアプリケーションを構築できます。次のサンプルコードは、あるディレクトリの子（ディレクトリ内にあるドキュメントの URI に対応する）をリストアップするために xdmp:directory 関数を使用し、prop:directory 要素を探すために xdmp:directory-properties 関数を使用して、その URI がディレクトリであることを示します。この例には、2 つの部分があります。

- ディレクトリブラウザのコード
- ディレクトリブラウザの設定

9.5.1 ディレクトリブラウザのコード

非常にシンプルなディレクトリブラウザのサンプルコードを次に示します。

```
xquery version "1.0-ml";
(:   directory browser
           Place in Modules database and give execute
           permission :)

declare namespace prop="http://marklogic.com/xdmp/property";

(: Set the root directory of your AppServer for the
   value of $rootdir :)
let $rootdir := (xdmp:modules-root())
(: take all but the last part of the request path, after
   the initial slash :)
let $dirpath := fn:substring-after(fn:string-join
                (fn:tokenize(xdmp:get-request-path(),
                "/")[1 to last() - 1], "/"), "/")
let $basedir := if ( $dirpath eq "" )
                then ( $rootdir )
                else fn:concat($rootdir, $dirpath, "/")
```

```
let $uri := xdmp:get-request-field("uri", $basedir)
return if (ends-with($uri, "/")) then
<html xmlns="http://www.w3.org/1999/xhtml">
   <head>
     <title>MarkLogic Server Directory Browser</title>
   </head>
   <body>
     <h1>Contents of {$uri}</h1>
<h3>Documents</h3>
{
   for $d in xdmp:directory($uri, "1")
   let $u := xdmp:node-uri($d)
   (: get the last two, and take the last non-empty
   string :)
   let $basename :=
     tokenize($u, "/")[last(), last() - 1][not(. = "")]
     [last()]
   order by $basename
   return element p {
     element a {

(:  The following should work for all $basedir values, as
    long as the string represented by $basedir is unique
    in the document URI :)
        attribute href { substring-after($u,$basedir) },
        $basename
     }
   }
}
       <h3>Directories</h3>
{
   for $d in xdmp:directory-properties($uri, "1")//prop:
   directory
   let $u := xdmp:node-uri($d)
   (: get the last two, and take the last non-empty
   string :)
   let $basename :=
     tokenize($u, "/")[last(), last() - 1][not(. = "")]
     [last()]
   order by $basename
   return element p {
     element a {
       attribute href { concat(
                                xdmp:get-request-path(),
                                "?uri=",
                                $u) },
       concat($basename, "/")
```

```
        }
      }
    }
    </body>
    </html>
    else doc($uri)

    (: browser.xqy :)
```

このアプリケーションは、サーバーのルートにあるドキュメントおよびディレクトリへのリンクを含む HTML ドキュメントを書き出します。xdmp:directory 関数を使用して、ルートディレクトリにあるドキュメントを検索し、xdmp:directory-properties 関数を使用してディレクトリを検索し、文字列操作を実行して、表示する URI の最後の部分を取得し、アプリケーションサーバーの request オブジェクトビルトイン XQuery 関数（xdmp:get-request-field および xdmp:get-request-path）を使用して状態を保持します。

9.5.2 ディレクトリブラウザの設定

このディレクトリブラウザアプリケーションを実行するには、次の操作を実行します。

1. HTTP サーバーを作成し、次のように設定します。

   a. ドキュメントデータベースと同じデータベースになるように Modules データベースを設定します。例えば、database 設定がデータベース my-database に設定されている場合は、modules データベースも my-database に設定します。

   b. HTTP サーバールートを http://myDirectory/ に設定します。または、ルートを別の値に設定し、HTTP サーバールートと一致するように、ディレクトリブラウザのコードの $rootdir 変数を変更します。

   c. ポートを 9001、または現在使用されていないポート番号に設定します。

2. サンプルコードをファイル browser.xqy にコピーします。必要に応じて、HTTP サーバールートと一致するように $rootdir 変数を変更します。サンプルコードのように xdmp:modules-root 関数を使用すると、アプリケーションサーバールートの値が自動的に取得されます。

3. browser.xqy ファイルを、HTTP サーバールートの最上位で Modules データベースに読み込みます。例えば、HTTP サーバールートが http://myDirectory/ である場合は、browser.xqy ファイルを URI http://myDirectory/browser.xqy でデータベースに読み込みます。ドキュメントの読み込みには、WebDAV クライアント（このルートを指す WebDAV サーバーもある場合）または xdmp:document-load 関数を使用できます。

4. browser.xqy ドキュメントに実行パーミッションが設定されていることを確認します。次の関数を使用して、パーミッションを確認できます。

```
xdmp:document-get-permissions("http://myDirectory/browser.
xqy")
```

このコマンドは、ドキュメントのすべてのパーミッションを返します。このコマンドは、アプリケーションを実行するユーザーが所有するロールの「実行」機能を持っている必要があります。持っていない場合は、次のようなコマンドを使用してパーミッションを追加できます。

```
xdmp:document-add-permissions("http://myDirectory/
browser.xqy",
                              xdmp:permission("myRole",
                              "execute"))
```

ここで myRole は、アプリケーションを実行するユーザーが所有するロールです。

5. 他のドキュメントを HTTP サーバールートに読み込みます。例えば、ドキュメントやフォルダを WebDAV クライアントにドラッグアンドドロップします（このルートを指す WebDAV サーバーもある場合）。

6. HTTP サーバーのホストおよびポート番号を使用して、Web ブラウザで browser.xqy ファイルにアクセスします。例えば、ローカルマシン上で実行しており、HTTP サーバーポートを 9001 に設定している場合は、このアプリケーションを URL http://localhost:9001/browser.xqy から実行できます。

データベースに読み込んだドキュメントとディレクトリへのリンクが表示されます。他のドキュメントを読み込んでいない場合は、browser.xqy ファイルへのリンクのみが表示されます。

10.0 ポイントインタイムクエリ

MarkLogic サーバーは、ドキュメントの古いバージョンを保持するように設定できます。これにより、過去の特定時点に戻ったかのようにクエリステートメントを評価できます。クエリステートメントの評価対象とするタイムスタンプを指定すると、そのステートメントは、指定したタイムスタンプ以前で（そのタイムスタンプを超えない範囲で）最新バージョンのデータベースに対して評価されます。

この章では、ポイントインタイムクエリと、ポイントインタイムへのフォレストのロールバックについて説明します。次のセクションで構成されています。

- ポイントインタイムクエリを理解する
- クエリでタイムスタンプを使用する
- xdmp:eval/xdmp:invoke/xdmp:spawn/XCC でポイントインタイムクエリを指定する
- システムタイムスタンプを追跡する
- 特定のタイムスタンプへのフォレストのロールバック

10.1 ポイントインタイムクエリを理解する

ポイントインタイムクエリを深く理解するには、各バージョンのフラグメントがどのように格納され、MarkLogic サーバーからマージされるのかについて、多少は理解しておく必要があります。このセクションでは、フラグメントが格納される仕組みと、それによってどのようにポイントインタイムクエリが実現されるのかについて説明します。また、ポイントインタイムクエリでできることとできないことを理解するために重要なその他の情報をいくつか示します。

- ログ構造化データベースに格納されるフラグメント
- システムタイムスタンプとマージのタイムスタンプ
- ポイントインタイムクエリ用のフラグメントが格納される仕組み
- クエリステートメントだけで利用でき、更新ステートメントでは利用できない
- すべての補助データベースが最新バージョンを使用する
- データベース設定の変更はポイントインタイムフラグメントに適用されない

マージ動作の詳細については、『*Administrator's Guide*』の「Understanding and Controlling Database Merges」の章を参照してください。また、「MarkLogic サーバーのトランザクションとは」（22 ページ）は、この章を理解するための背景情報として活用できます。

10.1.1 ログ構造化データベースに格納されるフラグメント

MarkLogic サーバーデータベースは、1 つ以上のフォレストで構成されています。また、各フォレストは 1 つ以上のスタンドで構成され、各スタンドには 1 つ以上のフラグメントが格納されています。フラグメントの数は、ドキュメントの数やデータベース設定で定義されたフラグメントのルートなど、複数の要因によって決まります。

効率性を最大限に高め、パフォーマンスを向上させるために、フラグメントは「ログ構造化ファイルシステム」と同様の方法を使用して管理されています。ログ構造化ファイルシステムは、ファイルを追加、削除、および変更する非常に効率的な方法であり、使用されなくなったバージョンのファイルを定期的に削除するガベージコレクション処理を使用します。MarkLogic サーバーでは、フラグメントはログ構造化データベースに格納されます。MarkLogic サーバーでは、複数のスタンドが定期的にマージされ、単一のスタンドが形成されます。このマージ処理は、ログ構造化ファイルシステムのガベージコレクションに相当します。

既存のドキュメントやノードを変更したり削除したりすると、1 つ以上のフラグメントに影響します。ドキュメントを変更する場合（`xdmp:node-replace` 操作など）、MarkLogic サーバーはその操作に関与するフラグメントの新しいバージョンを作成します。フラグメントの古いバージョンは廃棄対象としてマークされますが、この時点ではまだ削除されません。同様に、フラグメントを削除した場合も、単に廃棄対象としてマークされるだけで、ただちにディスクから削除されるわけではありません（ただしポイントインタイムクエリを使用しない限り、そのフラグメントをクエリできなくなります）。

10.1.2 システムタイムスタンプとマージのタイムスタンプ

マージが発生すると、廃棄対象のフラグメントによって占有されていたディスク領域が回復します。ただし、「システムタイムスタンプ」（MarkLogic サーバーの管理対象が変更されるたびに増加する数値）は維持されます。デフォルトでは、マージが完了した時点で、新しいスタンドにはその時点のタイムスタンプが設定されます（「マージのタイムスタンプ」）。マージのタイムスタンプよりも前に廃棄対象となったフラグメント（つまり、古いバージョンのフラグメントや削除されたフラグメント）は、マージ処理中に除去されます。

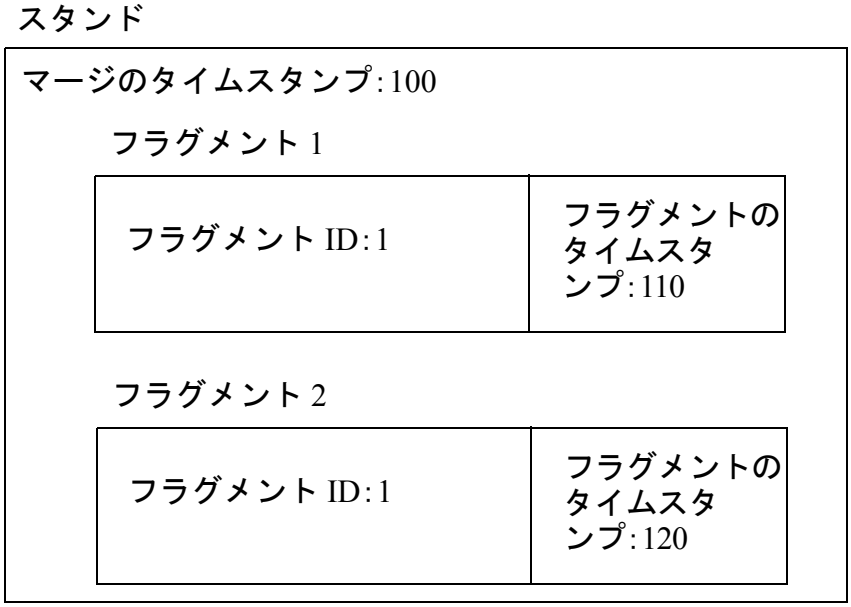
システムには `merge timestamp` というデータベースレベルのコントロールが用意されており、管理画面経由で設定します。デフォルトでは `merge timestamp` は 0 に設定されています。この場合、マージのタイムスタンプは、マージが完了した時点に対応するタイムスタンプに設定されます。ポイントインタイムクエリを使用する場合は、`merge timestamp` を特定時点に対応する静的値に設定できます。その時点以降に発生するマージでは、`merge timestamp` で指定した時点以降のタイムスタンプを持つすべてのフラグメント（廃棄対象のフラグメントを含む）が保持されます。

廃棄対象のフラグメントを保持することによって、古いデータベースを参照するクエリを実行できるという利点が生まれます。つまり、過去の特定時点においてデータベースをクエリしたかのような操作が可能です。マージのタイムスタンプの詳細については、「管理画面でポイントインタイムクエリを有効化する」（148 ページ）を参照してください。

10.1.3 ポイントインタイムクエリ用のフラグメントが格納される仕組み

古いタイムスタンプのフラグメントは、他のフラグメントと同様にスタンドに格納されます。そしてスタンドはフォレストに格納されます。唯一の違いは、古いタイムスタンプが関連付けられている点です。バージョンが異なるフラグメントは、別のスタンドにも同じスタンドにも格納できます。これは、同じスタンドにマージされたかどうかによって決まります。

以下の図は、マージのタイムスタンプが 100 であるスタンドを示したものです。フラグメント 1 はタイムスタンプ 110 で変更されたバージョン、フラグメント 2 は同じフラグメントですが、タイムスタンプ 120 で変更されたバージョンです。



このシナリオでは、現在のタイムスタンプが 200 の場合、現在の時点で実行したクエリはフラグメント 2 を認識しますが、フラグメント 1 を認識しません。また、タイムスタンプ 115 のポイントインタイムクエリを実行すると、フラグメント 1 が認識されますが、フラグメント 2 は認識されません（タイムスタンプ 115 の時点では、フラグメント 2 が存在していないため）。

保持できるバージョンの数に制限はありません。`merge timestamp` が現在の時点または過去の時点に設定されている場合、それ以降に変更されたフラグメントはすべて保持され、ポイントインタイムクエリで利用できます。

10.1.4 クエリステートメントだけで利用でき、更新ステートメントでは利用できない

指定できるのは、ポイントインタイムクエリのステートメントだけです。更新ステートメントのポイントインタイムクエリを指定すると、例外がスローされます。更新ステートメントは、MarkLogic サーバーに対して発行される XQuery であり、更新関数（`xdmp:document-load`、`xdmp:node-replace` など）が含まれます。クエリステートメントおよび更新ステートメントを構成する要素の詳細については、「MarkLogic サーバーのトランザクションとは」（22 ページ）を参照してください。

10.1.5 すべての補助データベースが最新バージョンを使用する

データベースリクエストに関連付けられた補助データベース（Security データベース、Schemas データベース、Modules データベース、および Triggers データベース）は、ポイントインタイムクエリの処理時であってもすべて最新のタイムスタンプで処理が行われます。そのため、ポイントインタイムクエリで指定した時点以降にセキュリティオブジェクトやスキーマなどに対して行われた変更はすべて、クエリに反映されます。例えば、ポイントインタイムクエリで指定した時点と最新のタイムスタンプとの間に、実行中のユーザーが削除された場合、そのクエリは認証に失敗します（ユーザーが存在しなくなったため）。

10.1.6 データベース設定の変更はポイントインタイムフラグメントに適用されない

データベースの設定を変更した場合（例えばデータベースインデックスの設定を変更した場合）、そのような変更はフラグメントの最新バージョンだけに適用されます。例えば、インデックスオプションに変更を加え、古いバージョンのフラグメントが保持されているデータベースのインデックスを再生成すると、最新バージョンのフラグメントだけが再インデックス化されます。また、古いバージョンのフラグメント（ポイントインタイムクエリで使用）では、妥当でなくなった時点のタイムスタンプ（フラグメントで更新または削除が発生した時点のタイムスタンプ）のインデックス化プロパティが保持されます。データベース設定の変更と、`merge timestamp` データベースパラメータが 0 以外に設定されているデータベースの再インデックス付けは、いずれも実行しないことを推奨します。

10.2 クエリでタイムスタンプを使用する

デフォルトでは、クエリステートメントはステートメントが開始した時点のシステムタイムスタンプで実行されます。クエリステートメントを別のシステムタイムスタンプで実行するには、古いバージョンのドキュメントを格納するようにシステムを設定しておき、ポイントインタイムクエリのステートメントを発行するときにタイムスタンプを指定します。このセクションでは、この処理の概要について説明します。以下のように構成されています。

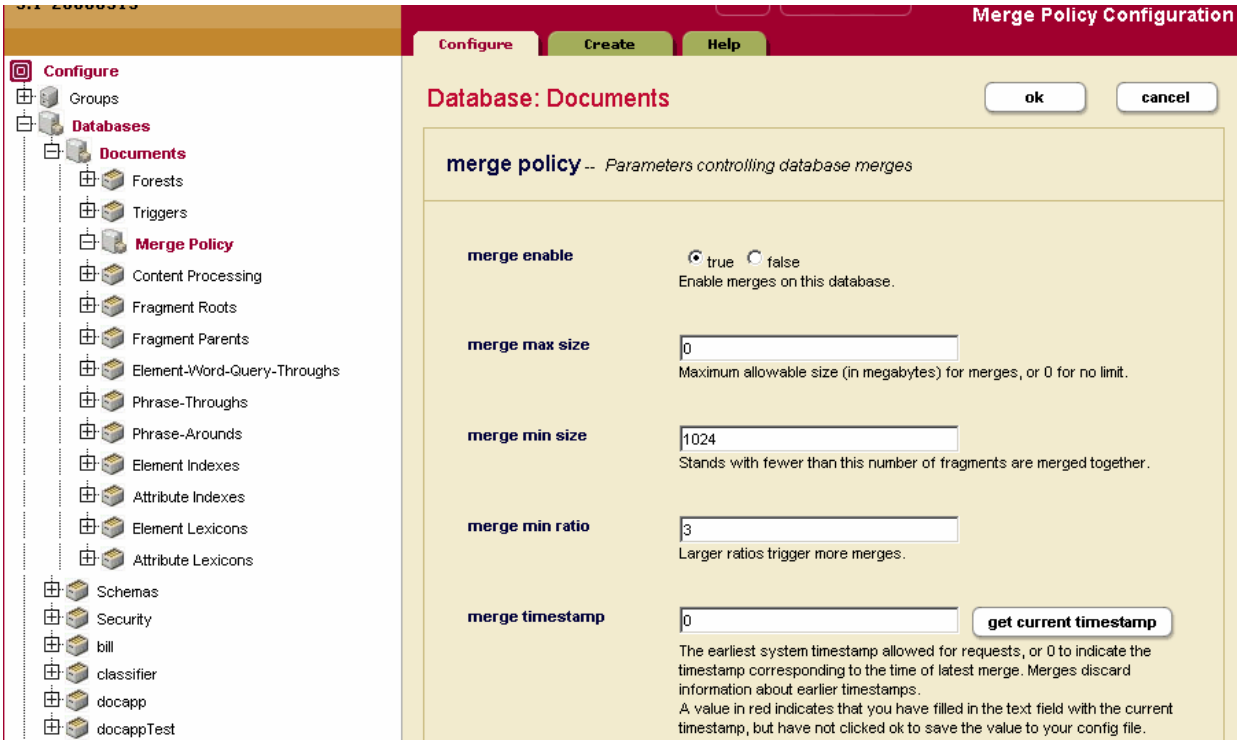
- 管理画面でポイントインタイムクエリを有効化する
- xdmp:request-timestamp 関数

- xdmp:timestamp の実行権限が必要
- xdmp:eval/xdmp:invoke/xdmp:spawn の timestamp パラメータ
- XCC におけるリクエストのタイムスタンプ
- スコア計算に関する考慮事項

10.2.1 管理画面でポイントインタイムクエリを有効化する

データベースでポイントインタイムクエリを使用するには、古いバージョンのフラグメントを保持するようにマージを設定する必要があります。デフォルトでは、古いバージョンのフラグメントはマージ後にデータベースから削除されます。マージ動作の詳細については、『*Administrator's Guide*』の「Understanding and Controlling Database Merges」の章を参照してください。

管理画面の［Merge Policy Configuration］ページには、`merge timestamp` というパラメータが用意されています。このパラメータが 0（デフォルト）に設定されていて、マージが有効な場合、ポイントインタイムクエリは実質的に無効になります。［Merge Policy Configuration］ページにアクセスするには、管理画面のツリーメニューから［Databases］> ［db_name］> ［Merge Policy］リンクをクリックします。



merge timestamp パラメータに設定する値としては、現在のシステムタイムスタンプが最もよく使われます。このパラメータの値を現在のシステムタイムスタンプに設定すると、現時点以降に生成されたバージョンのフラグメントが保持されるようになります。merge timestamp パラメータを現在のタイムスタンプに設定するには、[Merge Control Configuration] ページの [get current timestamp] ボタンをクリックし、[OK] をクリックします。

merge timestamp パラメータに現在のタイムスタンプよりも後の値を設定すると、MarkLogic サーバーではマージのときに現在のタイムスタンプが使用されます（デフォルトの 0 に設定した場合と同じ動作）。このように指定した場合は、システムタイムスタンプが merge timestamp の数値よりも後の値になると、指定した merge timestamp が使用されるようになります。同様に、データベースに保持された最も前のタイムスタンプよりも前の値を merge timestamp に設定すると、データベース内に保持されているすべてのフラグメントのうち最も前のタイムスタンプと現在のタイムスタンプのうちいずれか前のほうが使用されます。

システムタイムスタンプは長期間にわたって追跡することがあるため、ポイントインタイムクエリを実行するときには、実際の時刻とシステムタイムスタンプとをマップできます。そのようなタイムスタンプレコードを作成する方法の例については、「システムタイムスタンプを追跡する」（153 ページ）を参照してください。

注： merge timestamp を 0 に設定している場合は、マージ後に、すべての廃棄対象バージョンのフラグメントが削除されます。つまり、フラグメントの最新バージョンのみがデータベースに残ります。merge timestamp を現在のタイムスタンプよりも前の値に設定している場合、廃棄対象バージョンのフラグメントはデータベース内に存在しなくなるため、すべて利用できなくなります。そのため、フラグメントのバージョンを保持する場合は、先にそのようにシステムを設定してからコンテンツを更新する必要があります。

10.2.2 xdmp:request-timestamp 関数

MarkLogic サーバー には、XQuery ビルトイン関数 xdmp:request-timestamp が用意されています。この関数は、現在のリクエストのシステムタイムスタンプを返します。MarkLogic サーバーは、システムタイムスタンプの値を使用してフラグメントのバージョンを追跡します。また、ユーザーは merge timestamp パラメータでシステムタイムスタンプを使用して（「管理画面でポイントインタイムクエリを有効化する」（148 ページ）を参照）マージ後にデータベース内に残すフラグメントのバージョンを指定します。xdmp:request-timestamp 関数の詳細については、『*MarkLogic XQuery and XSLT Function Reference*』を参照してください。

10.2.3 xdmp:timestamp の実行権限が必要

現在のタイムスタンプ以外のタイムスタンプでクエリを実行するには、クエリを実行するユーザーが xdmp:timestamp の実行権限を持つグループに属している必要があります。セキュリティと実行権限の詳細については、『*Security Guide*』を参照してください。

10.2.4 xdmp:eval/xdmp:invoke/xdmp:spawn の timestamp パラメータ

xdmp:eval、xdmp:invoke、および xdmp:spawn 関数は、どれもオプションの 3 つめのパラメータとして options ノードを取ります。options ノードは、xdmp:eval 名前空間に含まれている必要があります。options ノードには timestamp 要素があり、クエリを実行するシステムタイムスタンプを指定できます。現在のタイムスタンプよりも前の timestamp の値を指定すると、ポイントインタイムクエリを指定していることになります。

指定するタイムスタンプは、データベースに対して妥当なものでなければなりません。データベースに保持されている最も古いタイムスタンプよりも前のシステムタイムスタンプを指定すると、XDMP-OLDSTAMP 例外がスローされます。現在のタイムスタンプよりも新しいタイムスタンプを指定すると、XDMP-NEWSTAMP 例外がスローされます。

> **注**：マージのタイムスタンプがデフォルトの 0 に設定されていて、データベースで最後に実行された更新または削除以降にすべてのマージが完了している場合は、現在のシステムタイムスタンプよりも古いタイムスタンプを指定するクエリステートメントを実行すると XDMP-OLDSTAMP 例外がスローされます。これは、マージのタイムスタンプの値が 0 であるために、廃棄対象のフラグメントが保持されていないからです。

以下の例は、timestamp パラメータを指定した xdmp:eval ステートメントを示したものです。

```
xdmp:eval("doc('/docs/mydocument.xml')", (),
  <options xmlns="xdmp:eval">
    <timestamp>99225</timestamp>
  </options>)
```

このステートメントでは、/docs/mydocument.xml ドキュメントのうち、システムタイムスタンプ 99225 の時点で存在していたバージョンが返されます。

10.2.5 XCC におけるリクエストのタイムスタンプ

xdmp:eval、xdmp:invoke、および xdmp:spawn 関数を使用すると、XQuery レベルでクエリステートメントのタイムスタンプを指定できます。XML Content Connector（XCC）ライブラリを使用して MarkLogic サーバーとやり取りする場合は、Java レベルでタイムスタンプを指定することもできます。

XCC for Java では、RequestOptions クラスを使用してリクエストのオプションを設定できます。このクラスでは、リクエストが実行される環境を変更できます。また、setEffectivePointInTime メソッドを使用して、リクエストが実行されるタイムスタンプを設定します。核となるデザインパターンは、リクエストのオプションを設定しておき、リクエストを MarkLogic サーバーに送って評価するときにそのオプションを使用するというものになります。また、Session オブジェクトでリクエストのオプションを設定することもできます。基本的なデザインパターンは、以下の Java コードの一部分で示すとおりです。

```
// create a class and methods that use code similar to
// the following to set the system timestamp for requests

Session session = getSession();
BigInteger timestamp = session.getCurrentServerPointInTime();
RequestOptions options = new RequestOptions();

options.setEffectivePointInTime (timestamp);
session.setDefaultRequestOptions (options);
```

Java 環境を使用してポイントインタイムクエリを実行する方法の例については、「例：XCC を使用して古いバージョンのドキュメントをクエリする」（152 ページ）を参照してください。

10.2.6 スコア計算に関する考慮事項

複数バージョンのフラグメントをデータベースに格納すると、cts:search の結果で返されるスコアに若干の影響が及びます。スコアの計算では、スコア計算式内の変数としてドキュメントの頻度が使用されます（デフォルトの score-logtfidf スコア計算方法の場合）。古いバージョンのフラグメントを保持する影響の大きさは、以下の 2 つの要因に依存します。

- 複数のバージョンを持つフラグメントの数
- データベース内に存在するフラグメントの総数

複数のバージョンを持つフラグメントの数がデータベース内に存在するフラグメントの総数と比べて小さい場合、影響は比較的少なくて済みます。この比率が大きくなると、スコアにおける影響も大きくなります。

スコアおよびスコア計算方法の詳細については、『*Search Developer's Guide*』の Relevance Scores: Understanding and Customizing を参照してください。

10.3 xdmp:eval/xdmp:invoke/xdmp:spawn/XCC でポイントインタイムクエリを指定する

前述のとおり、`xdmp:eval`、`xdmp:invoke`、または `xdmp:spawn` 関数の `options` ノードで、有効な `timestamp` 要素を指定すると、ポイントインタイムクエリが開始されます。また、XCC を使用して、XCC リクエスト全体をポイントインタイムクエリとして指定することもできます。クエリは、指定したタイムスタンプで実行され、指定したタイムスタンプに対応する特定時点に存在したバージョンのデータベースを参照します。このセクションでは、ポイントインタイムクエリのシナリオ例を取り上げます。以下のように構成されています。

- 例：XCC を使用して古いバージョンのドキュメントをクエリする
- 例：削除済みのドキュメントをクエリする

10.3.1 例：XCC を使用して古いバージョンのドキュメントをクエリする

システムでコンテンツに更新を加えたら、新しいコンテンツをユーザーに公開する前に、新しいバージョンのコンテンツを追加してテストできます。テスト中は、ユーザーは古いバージョンのコンテンツを参照し続けることになります。その後、新しいコンテンツを十分にテストしたら、ユーザーが新しいコンテンツを参照するように切り替えることができます。

ポイントインタイムクエリでは、この処理をすべて同一データベース内で実行できます。アプリケーションで変更する必要があるのは、クエリステートメントが実行されるタイムスタンプだけです。XCC には、この目標を実現するのに役立つメカニズムが用意されています。

10.3.2 例：削除済みのドキュメントをクエリする

ドキュメントを削除すると、そのドキュメントのフラグメントは廃棄対象としてマークされます。ただし、マージが完了するまで、そのフラグメントが実際に削除されることはありません。また、`merge timestamp` がドキュメントの削除時点に対応するタイムスタンプよりも前に設定されている場合、マージしても廃棄対象フラグメントは保持されます。

この例では、ポイントインタイムクエリによって削除済みドキュメントをクエリする方法を示します。簡潔に説明するため、この例の実行中は、システムでその他のクエリまたは更新アクティビティは発生しないものと仮定します。この例を理解するために、ここに示す順序で以下のコードサンプルを実行してください。

1. まず、ドキュメントを作成します。

```
xdmp:document-insert("/docs/test.xml", <a>hello</a>))
```

2. このドキュメントをクエリすると、挿入したノードが返されます。

```
doc("/docs/test.xml")
(: returns the node <a>hello</a> :)
```

3. ドキュメントを削除します。

```
xdmp:document-delete("/docs/test.xml")
```

4. ドキュメントを再度クエリします。削除されているため、空のシーケンスが返されます。

5. 現在のタイムスタンプを指定してポイントインタイムクエリを実行します(これはタイムスタンプを指定せずにドキュメントをクエリすることと同じ意味を持ちます)。

```
xdmp:eval("doc('/docs/test.xml')", (),
<options xmlns="xdmp:eval">
  <timestamp>{xdmp:request-timestamp()}</timestamp>
</options>)
(: returns the empty sequence because the document
has been deleted :)
```

6. 現在のタイムスタンプよりも 1 小さいタイムスタンプを指定して、ポイントインタイムクエリを実行します。この場合、データベースに発生した変更は 1 つだけであるため、古いタイムスタンプということになります。以下のクエリステートメントを実行すると、古いドキュメントが返されます。

```
xdmp:eval("doc('/docs/test.xml')", (),
<options xmlns="xdmp:eval">
  <timestamp>{xdmp:request-timestamp()-1}</timestamp>
</options>)
(: returns the deleted version of the document :)
```

10.4 システムタイムスタンプを追跡する

システムタイムスタンプは、更新が発生した実際の時点を記録しません。システムタイムスタンプは、システムで更新や設定の変更が発生するたびに増加する番号にすぎません。システムタイムスタンプを実際の時刻にマッピングするには、その情報をどこかに格納するか、または `xdmp:timestamp-to-wallclock` および `xdmp:wallclock-to-timestamp` XQuery 関数を使用する必要があります。このセクションでは、いくつかのサンプルコードを示します。これは、実際の時間間隔でシステムタイムスタンプをアーカイブするアプリケーションを作成する際の基本的な考え方を示すデザインパターンとなるものです。

注：アプリケーションによっては、システムタイムスタンプを実際の時刻にマップすることがそれほど重要でない場合もあります。例えば、マージのタイムスタンプを現在のタイムスタンプに設定し、その後のすべてのタイムスタンプが保持される場合などです。システムタイムスタンプを追跡する必要がない場合は、このアプリケーションを作成する必要はありません。

最初のステップは、タイムスタンプが格納されるドキュメントを作成し、現在のタイムスタンプの初期エントリを含めることです。将来のポイントインタイムクエリにおける混乱を避けるため、このドキュメントは、ポイントインタイムクエリを実行するデータベースとは別のデータベースに作成します。このドキュメントは、以下のように作成できます。

```
xdmp:document-insert("/system/history.xml",
<timestamp-history>
  <entry>
    <datetime>{fn:current-dateTime()}</datetime>
    <system-timestamp>{
(: use eval because this is an update statement :)
      xdmp:eval("xdmp:request-timestamp()")}
    </system-timestamp>
  </entry>
</timestamp-history>)
```

これにより、以下のようなドキュメントが生成されます。

```
<timestamp-history>
  <entry>
    <datetime>2006-04-26T19:35:51.325-07:00</datetime>
    <system-timestamp>92883</system-timestamp>
  </entry>
</timestamp-history>
```

このコードでは `xdmp:eval` を使用して、現在のタイムスタンプを取得しています。ステートメントは更新ステートメントであり、更新ステートメントは、`xdmp:request-timestamp` の呼び出しに対して常に空のシーケンスを返すため、`xdmp:eval` を使用する必要があります。詳細については、「MarkLogic サーバーのトランザクションとは」（22 ページ）を参照してください。

次に、以下のようなコードを定期的に実行するプロセスを設定します。例えば、以下のコードを 15 分ごとに実行するとします。

```
xdmp:node-insert-child(doc("/system/history.xml")/timestam
p-history,
<entry>
  <datetime>{fn:current-dateTime()}</datetime>
```

```
  <system-timestamp>{
(: use eval because this is an update statement :)
   xdmp:eval("xdmp:request-timestamp()")}
  </system-timestamp>
</entry>)
```

これにより、以下のようなドキュメントが生成されます。

```
<timestamp-history>
  <entry>
    <datetime>2006-04-26T19:35:51.325-07:00</datetime>
    <system-timestamp>92883</system-timestamp>
  </entry>
  <entry>
    <datetime>2006-04-26T19:46:13.225-07:00</datetime>
    <system-timestamp>92884</system-timestamp>
  </entry>
</timestamp-history>
```

このコードを定期的に呼び出すには、cron ジョブの設定やシェルスクリプトの記述、Java または .Net プログラムの記述など、利用する環境で有効な方法を用います。タイムスタンプの履歴が記載されたドキュメントがあれば、そのドキュメントをクエリすることで、特定時点のシステムタイムスタンプを簡単に調べることができます。

10.5 特定のタイムスタンプへのフォレストのロールバック

マージのタイムスタンプを設定し、削除したフラグメントを保持すると、特定のポイントインタイムにおけるデータベースの状態をクエリできるだけでなく、1 つあるいは複数のフォレストの状態を、保持されたタイムスタンプにロールバックすることもできます。1 つあるいは複数のフォレストを特定のタイムスタンプにロールバックするには、`xdmp:forest-rollback` 関数を使用します。このセクションには、xdmp:forest-rollback を使用した、1 つあるいは複数のフォレストの状態のロールバックに関する次の内容が含まれます。

- フォレストのロールバックで考慮すべきトレードオフとシナリオ
- マージのタイムスタンプの設定
- xdmp:forest-rollback 操作の実行に関する注意
- 1 つあるいは複数のフォレストをロールバックする一般的な手順

10.5.1 フォレストのロールバックで考慮すべきトレードオフとシナリオ

フォレストを以前のタイムスタンプにロールバックするには、古いバージョンのフラグメントをデータベース内で保持していたマージのタイムスタンプが以前に設定されていることが必要です。削除したフラグメントを保持すると、データベースのサイズが短期間で増大するため、より多くのディスク領域や他のシステムリソースが使用されます。古いフラグメントを保持する利点は、古いフラグメントをクエリできること（前のセクションの説明に従ってポイントインタイムクエリを使用）、およびデータベースを以前のタイムスタンプにロールバックできることです。システムにとって有意義であるかどうかを判断するときは、これらの利点（フォレストの状態を以前の時点に戻す利便性とスピード）およびコスト（ディスク領域、システムリソース、システムタイムスタンプの追跡など）を考慮する必要があります。

フォレストのロールバックの典型的なユースケースは、データベースのフルリストアを実行することなく、その事故以前のポイントインタイムに復帰できるようにして、ある種のデータ破損事故から保護することです。例えば、先週のいずれかの時点にアプリケーションが戻れるようにする場合は、マージのタイムスタンプを、7 日前のシステムタイムスタンプへと毎日更新するプロセスを作成できます。これにより、過去 7 日間の任意のポイントインタイムに戻すことができます。このプロセスを設定するには、次の手順に従う必要があります。

- システムタイムスタンプと実際の時点とのマッピングを維持します（「システムタイムスタンプを追跡する」（153 ページ）を参照）。
- データベースのマージのタイムスタンプを 7 日に 1 回更新するスクリプト（手動プロセス、または Admin API を使用した XQuery スクリプト）を作成します。このスクリプトは、マージのタイムスタンプを、7 日前にアクティブであったシステムタイムスタンプに更新します。
- ロールバックが必要になった場合は、データベース内のすべてのフォレストを、現在のタイムスタンプとマージのタイムスタンプの間の時点にロールバックします。例：

```
xdmp:forest-rollback(
     xdmp:database-forests(xdmp:database("my-db")),
     3248432)
(: where 3248432 is the timestamp to which you want to roll
back :)
```

フォレストのロールバック操作を使用する環境を設定する他のユースケースは、コードやコンテンツの新しい集まりをアプリケーションにプッシュしている場合に、以前の状態にロールバックできるようにすることです。このシナリオを設定するには、次の手順に従う必要があります。

- 新しいコンテンツ / コードをプッシュする前にシステムが定常状態にある場合は、マージのタイムスタンプを現在のタイムスタンプに設定します。
- 新しいコンテンツ / コードを読み込みます。

- 変更に満足していますか。
    - 「はい」の場合は、マージのタイムスタンプを 0 に戻すことができます。これにより、最終的には古いコンテンツ / コードがマージされます（削除されたフラグメントであるため）。
    - 「いいえ」の場合は、データベース内のすべてのフォレストを、マージのタイムスタンプで設定したタイムスタンプにロールバックします。

10.5.2 マージのタイムスタンプの設定

前述のとおり、データベースのマージのタイムスタンプが設定されていないフォレストはロールバックできません。デフォルトでは、マージのタイムスタンプが 0 に設定されます。これにより、マージ操作中に、古いバージョンのフラグメントが削除されます。詳細については、「システムタイムスタンプとマージのタイムスタンプ」（145 ページ）を参照してください。

10.5.3 xdmp:forest-rollback 操作の実行に関する注意

このセクションでは、フォレストをロールバックできる環境をセットアップする前に理解する必要のある `xdmp:forest-rollback` の動作について説明します。`xdmp:forest-rollback` 操作については、次の点に注意してください。

- `xdmp:forest-rollback` は、指定されたフォレストを再起動します。その結果、フェイルオーバーされたフォレストはすべて、プライマリホストをマウントしようとします。つまり、そのフォレストがフェイルオーバーされている場合は、フェイルオーバー解除操作が行われることになります。フェイルオーバーの詳細については、『*Scalability, Availability, and Failover Guide*』ガイドの High Availability of Data Nodes With Failover を参照してください。
- コンテキストデータベース内にある 1 つあるいは複数のフォレスト（つまり、クエリの評価時に基準となるデータベースに属しているフォレスト）をロールバックするときは注意が必要です。コンテキストデータベース内にあるフォレストでは、`xdmp:forest-rollback` 操作が非同期で実行されます。フォレストの新しい状態は、フォレストの再起動が発生するまで認識されません。フォレストがアンマウントされるまで、古い状態が反映され続けます。また、ロールバック操作の一環として発生する可能性のあるエラーは、そのロールバック操作を実行するクエリには報告されません（ただし、可能な場合は `ErrorLog.txt` ファイルにログが記録されます）。ベストプラクティスとして、コンテキストデータベースにアタッチされていないフォレストに対して `xdmp:forest-rollback` 操作を実行することを推奨します。
- データベース内の一部のフォレストをロールバック対象として指定しなかった場合、ロールバックされたフォレストが他のフォレストと一貫性のない状態になる可能性があります。ほとんどの場合、データベース内にあるすべてのフォレストをロールバックすることを推奨します。ただし、他のフォレストが同じ状態にロールバックされないとしても、ロールバックしているフォレストのコンテンツは、一貫性のない状態にはならないと確信している場合（ロールバックしている

すべてのコンテンツが 1 つのフォレストだけに存在していることがわかっている場合など）を除きます。

- ロールバック先のポイントインタイム以降にデータベースのインデックス付け設定が変更されている場合、および再インデックス付けが有効な場合は、ロールバック操作が完了するとすぐに再インデックス付けが開始されます。再インデックス付けが有効でない場合、ロールバックされたフラグメントには、最後に更新された時点のインデックスが付いたままになります。このため、現在のデータベース設定と一貫性のない状態になる可能性があります。
- ベストプラクティスとして、操作の時点で更新アクティビティのないフォレストのみでロールバック操作を実行することを推奨します（つまり、フォレストが休止している必要があります）。

10.5.4 1 つあるいは複数のフォレストをロールバックする一般的な手順

1 つあるいは複数のフォレストの状態をロールバックするには、次の一般的な手順を実行します。

1. ロールバック先にすることができるデータベース状態で、マージのタイムスタンプを現在のタイムスタンプに設定します。

2. システムタイムスタンプを追跡します（「システムタイムスタンプとマージのタイムスタンプ」（145 ページ）を参照）。

3. アプリケーションに対する更新を通常どおり実行します。古いバージョンのドキュメントはデータベースに残ります。

4. 現時点より前にロールバックする必要がない場合は、手順 1 に戻ります。

5. ロールバックする場合、マージのタイムスタンプと現在のタイムスタンプの間の任意の時点にロールバックできます。ロールバックを実行するときは、別のデータベースのコンテキストから実行することを推奨します。例えば、`my-db` データベース内にあるすべてのフォレストをロールバックするには、次のような操作を実行します。これにより、データベースのコンテキストが、ロールバックしているフォレストとは異なるコンテキストに設定されます。

```
xdmp:eval(
 'xdmp:forest-rollback(
    xdmp:database-forests(xdmp:database("my-db")),
    3248432)
 (: where 3248432 is the timestamp to which you want
    to roll back :)',
 (),
 <options xmlns="xdmp:eval">
   <database>{xdmp:database("Documents")}</database>
 </options>)
```

11.0 システムプラグインフレームワーク

この章では、MarkLogic サーバーのシステムプラグインフレームワークについて説明します。次のセクションから構成されます。

- MarkLogic サーバープラグインの仕組み
- システムプラグインモジュールの記述
- パスワードプラグインのサンプル

11.1 MarkLogic サーバープラグインの仕組み

プラグインを使用すると、MarkLogic サーバークラスタ内のすべてのアプリケーションに機能を提供できます。アプリケーションがコードを呼び出す必要はありません。このセクションでは、MarkLogic サーバーのシステムプラグインフレームワークについて説明します。以下のように構成されています。

- システムプラグインの概要
- システムプラグインとアプリケーションプラグイン
- plugin API

11.1.1 システムプラグインの概要

プラグインは、リクエストが評価される前に一部の機能を自動的に実行するために使用されます。プラグインは、XQuery メインモジュールであるため、任意の作業を実行できます。プラグインフレームワークは、各リクエストが評価される前に、`<marklogic-dir>/Plugins` ディレクトリ内にあるメインモジュールを評価します。

プラグインフレームワークの仕組みについては、次の点を考慮してください。

- MarkLogic の起動後に、`Plugins` ディレクトリ内にある各モジュールが評価されてから、クラスタ内の各ノードで各アプリケーションサーバーに対する最初のリクエストが評価されます。このプロセスは、`Plugins` ディレクトリが変更された後に繰り返されます。
- クラスタを使用するときは、`Plugins` ディレクトリに追加したすべてのファイルを、MarkLogic サーバークラスタ内の各ノードにある `Plugins` ディレクトリに追加する必要があります。
- プラグインモジュールのエラー（シンタックスエラーなど）は、クラスタ内でアプリケーションサーバー（管理画面を含む）に対してリクエストが実行されるたびに必ずスローされます。そのため、*`<marklogic-dir>`*`/Plugins` ディレクトリへの導入前にプラグインモジュールをテストすることが非常に重要です。プラグインモジュールにエラーがある場合は、アプリケーションサーバーに対するリクエストの評価が正常に終了する前に、必ず修正する必要があります。

- プラグインはキャッシュされますが、パフォーマンス上の理由から、MarkLogic サーバーが更新を確認するのは 1 秒間に 1 回だけです。また、キャッシュをリフレッシュするのは、Plugins ディレクトリが更新された後のみです。Plugins ディレクトリ内にある個々のファイルに変更が発生したかどうかは確認しません。更新のたびに新しいファイルを作成する（そして、ディレクトリを変更する）エディタを使用してプラグインを変更している場合、MarkLogic サーバーは、次の 1 秒以内に更新を認識します。ファイルをその場で変更するエディタの場合、最新の変更内容が読み込まれるように、ディレクトリに touch コマンドを実行して変更日を変更する必要があります（または、MarkLogic サーバーを再起動することもできます）。Plugins ディレクトリからプラグインを削除しても、MarkLogic サーバーを再起動するか、同名の他プラグインを各アプリケーションサーバーに登録するまでは、そのプラグインをすでに評価しているアプリケーションサーバーでは登録されたままになります。

11.1.2 システムプラグインとアプリケーションプラグイン

MarkLogic サーバーにはプラグインが 2 種類あります。「システムプラグイン」と「アプリケーションプラグイン」です。

システムプラグインは、MarkLogic サーバーのビルトインプラグインフレームワークを、xdmp:set-server-field 関数および xdmp:get-server-field 関数とともに使用します。「システムプラグインの概要」（159 ページ）で説明するとおり、システムプラグインは *<marklogic-dir>*/Plugins ディレクトリに格納され、これらのプラグインでエラーが発生した場合は、クラスタ内のすべてのアプリケーションサーバーでスローされます。

アプリケーションプラグインは、システムプラグインの上に構築されます。また、アプリケーションで使用されるように設計されています。アプリケーションプラグインは *<marklogic-dir>*/Assets/plugins/marklogic/appservices ディレクトリに格納され、システムプラグインとは異なり、プラグインコードにエラーが含まれていても、他のアプリケーションでエラーを引き起こすことはありません。

11.1.3 plugin API

plugin:register 関数は、MarkLogic サーバークラスタ内の任意の場所でプラグイン機能を使用できるようにするためにプラグインモジュールが使用するメカニズムです。plugin API の他の機能は、登録機能を実装するために使用されます。plugin API は、サーバーフィールド（xdmp:set-server-field および xdmp:get-server-field 関数ファミリ）を使用して、各プラグインの ID と機能を登録します。Plugins ディレクトリをスキャンするプラグインフレームワークとこの API を組み合わせると、MarkLogic サーバークラスタのすべてのアプリケーションサーバーで使用できる機能を作成できます。

plugin API を使用して、プラグインの集合を登録できます。その後、特定の機能を持つすべてのプラグインを要求すると、各プラグインが提供する機能をアプリケーションで使用できるようになります。plugin API の詳細については、『*MarkLogic XQuery and XSLT Function Reference*』を参照してください。

11.2 システムプラグインモジュールの記述

プラグインモジュールは XQuery メインモジュールにすぎないため、その点では、どのメインモジュールも、`Plugins` ディレクトリに配置するとプラグインになります。つまり、プラグインモジュールは、実現しようとする機能によって異なります。

警告 システムプラグインモジュールにエラーが発生すると必ず、すべてのリクエストでエラーが発生します。そのため、実稼働環境に導入する前にプラグインをテストすることが非常に重要です。

システムプラグインを使用するには、プラグインメインモジュールを `Plugins` ディレクトリに導入する必要があります。MarkLogic サーバークラスタにプラグインを導入するには、クラスタ内にある各ホストのプラグインディレクトリにプラグインメインモジュールをコピーする必要があります。

警告 記述するシステムプラグインモジュールは、一意のファイル名にする必要があります。MarkLogic から出荷された *<marklogic-dir>*`/Plugins` ディレクトリ内のプラグインファイルは、どれも変更しないでください。MarkLogic がインストールした、`Plugins` ディレクトリ内のファイルに変更を加えても、MarkLogic サーバーをアップグレードするたびにオーバーライドされます。

11.3 パスワードプラグインのサンプル

このセクションでは、パスワードプラグインと、その変更方法の例について説明します。以下のように構成されています。

- パスワードプラグインについて
- パスワードプラグインの変更

11.3.1 パスワードプラグインについて

システムプラグインのユースケースの 1 つは、パスワードに含まれている文字や特殊文字などの数の確認です。`<marklogic-dir>/Samples/Plugins` ディレクトリには、パスワード確認用のサンプルプラグインモジュールが含まれています。

セキュリティ XQuery ライブラリ（`security.xqy`）を使用してパスワードを設定すると、このプラグインが呼び出され、このプラグイン機能と次の URI を使用してパスワードが確認されます。

```
http://marklogic.com/xdmp/security/password-check
```

上記の機能に登録されたプラグインが *<marklogic-dir>*/Plugins ディレクトリにない場合、パスワードを設定するときに他の処理は実行されません。上記の password-check 機能に登録されたプラグインが *<marklogic-dir>*/Plugins ディレクトリにある場合は、パスワードを設定するときにモジュールが実行されます。この機能に複数のプラグインが登録されている場合は、そのすべてが実行されます。実行する順序は不定であるため、順序が影響しないようにコードを設計してください。

最短長を確認するサンプルと、パスワードに数字が含まれているかどうかを確認するサンプルが含まれています。必要なパスワード確認機能（特定の長さであるか、各種の特殊文字が存在するか、文字の繰り返しがあるか、大文字や小文字があるかの確認など）を実行する独自のプラグインモジュールを作成できます。

また、Security データベースユーザードキュメントに追加の履歴を保存するためのプラグインを作成できます。このドキュメントには、パスワード確認コード内で使用または更新できる情報が格納されます。パスワード確認アプリケーションの情報を格納するために使用できる要素は sec:password-extra です。sec:user-set-password-extra 関数および sec:user-set-password-extra 関数（security.xqy 内）を使用して、ユーザードキュメント内の sec:password-extra 要素を変更できます。これらの API を使用して、sec:password-extra 要素の子として要素を作成します。

<marklogic-dir>/Samples/Plugins/password-check-minimum-length.xqy ファイルを調べると、このファイルが、成功時に空（から）を返し、パスワードが最小文字数より短い場合はエラーメッセージを返す関数を含むメインモジュールであることがわかります。メインモジュールのボディでは、その機能（複数の機能を登録できるが、ここでは 1 つのみ登録）と一意の名前（この例では、次の xqy ファイルの名前）が含まれるマップに、このプラグインが登録されます。

```
let $map := map:map(),
      $_ := map:put($map,
"http://marklogic.com/xdmp/security/password-check",
                    xdmp:function(xs:QName
                    ("pwd:minimum-length")))
return
  plugin:register($map, "password-check-minimum-length.xqy")
```

これにより、pwd:minimum-length 関数が http://marklogic.com/xdmp/security/password-check 機能に登録されます。このプラグインは、パスワードが設定されるたびに呼び出されます。

注： プラグインを登録するには、一意の名前（plugin:register の 2 番目のプラグイン）を使用する必要があります。その名前が他のプラグインで使用されている場合、最終的に登録されるのはいずれか 1 つのみです（登録が上書きされるため）。

パスワードの確認時（ユーザー作成とパスワード変更の両方）に実行されるロジックを独自に実装する場合は、次のセクションで説明するように、プラグインを記述できます。

11.3.2 パスワードプラグインの変更

次の例は、最短パスワード長を確認し、1 文字以上の数字が必ず含まれているようにする方法を示しています。

警告 プラグインモジュールにエラーがあると、すべてのリクエストでエラーが発生します。そのため、実稼働環境に導入する前にプラグインをテストすることが非常に重要です。

サンプルパスワードプラグインを使用および変更するには、次の手順を実行します。

1. `<marklogic-dir>Samples/Plugins/password-check-*.xqy` ファイルを `Plugins` ディレクトリにコピーします。例：

   ```
   cd /opt/MarkLogic/Plugins
   cp ../Samples/Plugins/password-check-*.xqy .
   ```

   必要に応じて、コピーする場合にファイル名を変更します。

2. ファイル（`password-check-minimum-length` 等）を変更する場合は、そのファイルをテキストエディタで開きます。

3. 必要な変更を加えます。例えば、最短長を変更するには、`pwd:minimum-length` 関数を見つけ、4 から 6（または希望する任意の値）へ変更します。完了すると、関数のボディは次のようになります。

   ```
   if (fn:string-length($password) < 6)
   then "password too short"
   else ()
   ```

   これで、パスワードが 6 文字以上であることが確認されます。

4. 必要に応じて、ファイル名を変更した場合は、`plugin:register` の 2 番目のパラメータを、最初のステップでプラグインファイルに付けた名前に変更します。例えば、プラグインファイルの名前を `my-password-plugin.xqy` にした場合は、`plugin:register` の呼び出しを次のように変更します。

   ```
   plugin:register($map, "my-password-plugin.xqy")
   ```

5. 変更をファイルに保存します。

   警告 プラグインでシンタックスエラーを引き起こすような入力ミスや誤りがあった場合は、アプリケーションサーバーに対するすべてのリクエストが例外をスローします。その場合は、ファイルを編集してエラーを修正してください。

6. クラスタを使用している場合は、クラスタ内にある各ホストの `Plugins` ディレクトリにプラグインをコピーします。

7. コードをテストして、意図したとおりに動作することを確認します。

次回にパスワードを変更しようとしたときに、新しい確認処理が実行されます。
例えば、パスワードを 1 文字にしようとすると、却下されます。

12.0 map 関数を使用した名前 / バリューマップの作成

この章では、map 関数の使い方を説明します。以下のセクションで構成されています。

- マップ : XQuery で操作するインメモリ構造
- map:map XQuery プリミティブ型
- XML ノードへのマップのシリアライズ
- map API
- マップ演算子
- 例

12.1 マップ : XQuery で操作するインメモリ構造

「マップ」は、ユーザーが作成および操作できる名前 / バリューペアを含むインメモリ構造です。マップは、プログラミング言語によってはハッシュ テーブルを使って実現されているものもあります。これはプログラミングに便利なツールで、名前 / バリューのペアを格納・更新し、プログラム内で後で使えます。またデータへのアクセスが速くかつ楽になります。

MarkLogic サーバーには、マップを作成および操作するための一連の XQuery があります。`xdmp:set` 関数と同様に、マップには副作用があり、プログラムが変更される可能性があります。このため厳密には、マップは他の大部分の XQuery のような関数型ではありません。マップがメモリ内にある場合、開発者にとってその構造はブラックボックス化します。これにアクセスするにはビルトインの XQuery 関数を使用します。しかしマップを保存して後で使いたい場合には、マップの構造を XML ノードとして保持できます。マップはノードなので、アイデンティティがあります。このアイデンティティはマップがメモリ内にある限り同じものになります。しかしマップを XML としてシリアライズしてドキュメント内に格納した場合、これを抽出するとノードのアイデンティティは変わってしまいます（つまり、マップのアイデンティティとシリアライズされたマップのアイデンティティを比較した場合、false が返されます）。同様に、データベースから抽出した XML 値をマップに格納した場合、インメモリマップのノードのアイデンティティは、このデータベースのノードと同じアイデンティティになります（マップがメモリにある限り）。しかしこのマップが XML ドキュメントとしてシリアライズされデータベースに格納されると、アイデンティティは変わってしまいます。これは、他の XQuery におけるノードのアイデンティティの場合と同じです。

キーは `xs:string` 型を取り、値は `item()*` 値を取ります。このため、値としては、文字列、要素、アイテムのシーケンスを使用できます。マップは、インメモリの XML ノードに値を格納したものに XPath でアクセスする方法の代替策となります。マップによって、値の更新がとても楽にできます。

12.2 map:map XQuery プリミティブ型

マップは、map:map XQuery プリミティブ型として定義されます。この型は、関数や変数の定義において使うことができます。また他のプリミティブ型と同様に XQuery で使えます。またこれを XML にシリアライズしてデータベース内に格納できます。これについては以下のセクションで説明します。

12.3 XML ノードへのマップのシリアライズ

マップの構造は、cts:query のシリアライズとほぼ同じ方法で、XML 要素のコンテキストにマップを配置することで、XML ノードにシリアライズできます（詳細については、『*Search Developer's Guide*』の Composing cts:query Expressions の章の Serializing a cts:query as XML を参照）。マップの中身をデータベースに格納して保存したい場合には、シリアライズするといいでしょう。この XML は、*<marklogic-dir>*/Config/map.xsd スキーマに準拠しており、その名前空間は http://marklogic.com/xdmp/map です。

例えば以下のようにして、構成されたマップを XML でシリアライズできます。

```
let $map := map:map()
let $key := map:put($map, "1", "hello")
let $key := map:put($map, "2", "world")
let $node := <some-element>{$map}</some-element>
return $node/map:map
```

以下の XML が返されます。

```
<map:map xmlns:map="http://marklogic.com/xdmp/map"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <map:entry key="1">
    <map:value xsi:type="xs:string">hello</map:value>
  </map:entry>
  <map:entry key="2">
    <map:value xsi:type="xs:string">world</map:value>
  </map:entry>
</map:map>
```

12.4 map API

map API はとてもシンプルです。マップは、map:map 関数を使って最初から作ることも、マップの XML 表現（map:map）から作ることもできます。map 関数には以下のものがあります。各関数のシグネチャおよび説明については、『*MarkLogic XQuery and XSLT Function Reference*』を参照してください。

- map:clear
- map:count
- map:delete

- `map:get`
- `map:keys`
- `map:map`
- `map:put`

12.5 マップ演算子

マップ演算子は、セット演算子と同じような機能を持っています。セットをさまざまな方法で組み合わせて他のセットを作ることができるように、マップもマップ演算子を使って組み合わせることができます。以下のテーブルで、さまざまなマップ演算子について説明しています。

マップ演算子	説明
`+`	2 つのマップの和集合（union）。1 つめのマップ（Map A）と 2 つめのマップ（Map B）のキーと値をすべて含みます。例については、「マップの和集合の作成」（170 ページ）を参照してください。
`*`	2 つのマップの積集合（intersection）。セットの積集合と同様です。結果として、両方のマップ（Map A と Map B）に共通するキー / バリューペアが返されます。例については、「マップの積集合の作成」（171 ページ）を参照してください。
`-`	2 つのマップの差集合（difference）。セットの差集合と同様です。結果として、1 つ目のマップ（Map A）には存在するが、2 つ目のマップ（Map B）には存在しないキー / バリューペアが返されます。例については、「マップの差演算子の適用」（172 ページ）を参照してください。 この演算子は、単項の否定演算子としても使用できます。その場合、キーとバリューが逆になります。例については、「単項の否定演算子の適用」（173 ページ）を参照してください。
`div`	あるマップの値が他のマップのキーとマッチするという推論。結果として、1 つ目のマップ（Map A）のキーと、2 つ目のマップ（Map B）の値が返されます。Map A の値は Map B のキーと同じです。例については、「Div 演算子の適用」（174 ページ）を参照してください。
`mod`	単項の否定演算子とマップ間の推論を組み合わせたもの。1 つめのマップ（Map A）のキーと 2 つめのマップ（Map B）の値が入れ替わったものが返されます（Map A の値が Map B のキーと等しい場合）。 つまり、`Map A mod Map B` は `-Map A div Map B` と同等です。例については、「Mod 演算子の適用」（175 ページ）を参照してください。

12.6 例

このセクションでは、マップを使ったコードの例を紹介します。以下の例があります。

- シンプルなマップの作成
- マップ内の値を返す
- シリアライズされたマップの作成
- シーケンス値の追加
- マップの和集合の作成
- マップの積集合の作成
- マップの差演算子の適用
- 単項の否定演算子の適用
- Div 演算子の適用
- Mod 演算子の適用

12.6.1 シンプルなマップの作成

以下の例ではマップを作成し、キー / バリューペアを 2 つ入れ、このマップを返します。

```
let $map := map:map()
let $key := map:put($map, "1", "hello")
let $key := map:put($map, "2", "world")
return $map
```

これにより、キー / バリューペアが 2 つあるマップが返されます。キー「1」の値は「hello」、キー「2」の値は「world」になっています。

12.6.2 マップ内の値を返す

以下の例ではマップを作成し、キーでソートした値を返します。

```
let $map := map:map()
let $key := map:put($map, "1", "hello")
let $key := map:put($map, "2", "world")
return
  for $x in map:keys($map)
  order by $x return
  map:get($map, $x)
(: returns hello world :)
```

12.6.3 シリアライズされたマップの作成

以下の例では前の例と同様にマップを作成し、このマップを XML ノードにシリアライズします。その後、この XML ノードから新しいマップを作り、新しいキー / バリューペアを入れます。最後にこの新しいマップを返します。

```
let $map := map:map()
let $key := map:put($map, "1", "hello")
let $key := map:put($map, "2", "world")
let $node := <some-element>{$map}</some-element>
let $map2 := map:map($node/map:map)
let $key := map:put($map2, "3", "fair")
return $map2
```

これにより、キー / バリューペアが 3 つあるマップが返されます。キー「1」の値は「hello」、キー「2」の値は「world」、キー「3」の値は「fair」になっています。`$map` 変数にバインドされたマップは、`$map2` にバインドされたマップと同じではないことに注意してください。XML にシリアライズされた後、新しいマップが `$map2` 変数内に構築されています。

12.6.4 シーケンス値の追加

マップに入れる値の型は、`item()*` です。つまり、任意のシーケンスをキーの値として追加できます。以下の例には、いくつかの文字列値と 1 つのシーケンス値が含まれ、それぞれの結果が `<result>` 要素に出力されます。

```
let $map := map:map()
let $key := map:put($map, "1", "hello")
let $key := map:put($map, "2", "world")
let $seq := ("fair",
             <some-xml>
                 <another-tag>with text</another-tag>
             </some-xml>)
let $key := map:put($map, "3", $seq)
return
  for $x in map:keys($map) return
  <result>{map:get($map, $x)}</result>
```

これは以下の要素を返します。

```
<result>fair
        <some-xml>
            <another-tag>with text</another-tag>
        </some-xml>
</result>
<result>world</result>
<result>hello</result>
```

12.6.5 マップの和集合の作成

以下の例では 2 つのマップの和集合を作成し、そのキー / バリューペアを返します。

```
let $mapA := map:map(
<map:map xmlns:map="http://marklogic.com/xdmp/map">
        <map:entry>
            <map:key>1</map:key>
            <map:value>1</map:value>
        </map:entry>
        <map:entry>
            <map:key>3</map:key>
            <map:value>3</map:value>
        </map:entry>
</map:map>)
let $mapB := map:map(
<map:map xmlns:map="http://marklogic.com/xdmp/map">
        <map:entry>
            <map:key>2</map:key>
            <map:value>2</map:value>
        </map:entry>
        <map:entry>
            <map:key>3</map:key>
            <map:value>3</map:value>
            <map:value>3.5</map:value>
        </map:entry>
        </map:map>)
return $mapA + $mapB
```

両方のマップに共通のキー / バリューペアが、1 回だけ含まれます。これにより、以下が返されます。

```
<xml version="1.0" encoding="UTF-8">
<results warning="atomic item">
    <map:map xmlns:map="http://marklogic.com/xdmp/map"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xs="http://www.w3.org/2001/XMLSchema">
        <map:entry key="1">
            <map:value>1</map:value>
        </map:entry>
        <map:entry key="2">
            <map:value>2</map:value>
        </map:entry>
        <map:entry key="3">
            <map:value>3</map:value>
            <map:value>3.5</map:value>
        </map:entry>
    </map:map>
</results>
```

12.6.6 マップの積集合の作成

以下の例では 2 つのマップの積集合を作成します。

```
xquery version "1.0-ml";
let $mapA := map:map(
<map:map xmlns:map="http://marklogic.com/xdmp/map">
        <map:entry>
            <map:key>1</map:key>
            <map:value>1</map:value>
        </map:entry>
        <map:entry>
            <map:key>3</map:key>
            <map:value>3</map:value>
        </map:entry>
        </map:map>)
let $mapB := map:map(
<map:map xmlns:map="http://marklogic.com/xdmp/map">
        <map:entry>
            <map:key>2</map:key>
            <map:value>2</map:value>
        </map:entry>
        <map:entry>
            <map:key>3</map:key>
            <map:value>3</map:value>
            <map:value>3.5</map:value>
        </map:entry>
</map:map>)
return $mapA * $mapB
```

両方のマップに共通のキー / バリューペアだけが返されます。これにより、以下が返されます。

```
<xml version="1.0" encoding="UTF-8">
<results warning="atomic item">
    <map:map xmlns:map="http://marklogic.com/xdmp/map"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xs="http://www.w3.org/2001/XMLSchema">
        <map:entry key="3">
            <map:value>3</map:value>
        </map:entry>
    </map:map>
</results>
```

12.6.7 マップの差演算子の適用

以下の例では、Map A には存在するが、Map B には存在しないキー / バリューペアが返されます。

```
let $mapA := map:map(
<map:map xmlns:map="http://marklogic.com/xdmp/map">
        <map:entry>
            <map:key>1</map:key>
            <map:value>1</map:value>
        </map:entry>
        <map:entry>
            <map:key>3</map:key>
            <map:value>3</map:value>
        </map:entry>
</map:map>)
let $mapB := map:map(
<map:map xmlns:map="http://marklogic.com/xdmp/map">
        <map:entry>
            <map:key>2</map:key>
            <map:value>2</map:value>
        </map:entry>
        <map:entry>
            <map:key>3</map:key>
            <map:value>3</map:value>
            <map:value>3.5</map:value>
        </map:entry>
        </map:map>)
return $mapA - $mapB
```

これにより、以下が返されます。

```
<xml version="1.0" encoding="UTF-8">
<results warning="atomic item">
    <map:map xmlns:map="http://marklogic.com/xdmp/map"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xs="http://www.w3.org/2001/XMLSchema">
        <map:entry key="1">
            <map:value>1</map:value>
        </map:entry>
    </map:map>
</results>
```

12.6.8 単項の否定演算子の適用

以下の例では、マップの差演算子を単項の否定演算子として使用し、マップの内のキーと値を入れ替えます。

```
xquery version "1.0-ml";
let $mapA := map:map(
<map:map xmlns:map="http://marklogic.com/xdmp/map">
        <map:entry>
            <map:key>1</map:key>
            <map:value>1</map:value>
        </map:entry>
        <map:entry>
            <map:key>3</map:key>
            <map:value>3</map:value>
        </map:entry>
        </map:map>)
let $mapB := map:map(
<map:map xmlns:map="http://marklogic.com/xdmp/map">
        <map:entry>
            <map:key>2</map:key>
            <map:value>2</map:value>
        </map:entry>
        <map:entry>
            <map:key>3</map:key>
            <map:value>3</map:value>
            <map:value>3.5</map:value>
        </map:entry>
</map:map>)
return -$mapB
```

これにより、以下が返されます。

```
<xml version="1.0" encoding="UTF-8">
<results warning="atomic item">
    <map:map xmlns:map="http://marklogic.com/xdmp/map"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xs="http://www.w3.org/2001/XMLSchema">
        <map:entry key="3.5">
            <map:value>3</map:value>
        </map:entry>
        <map:entry key="2">
            <map:value>2</map:value>
        </map:entry>
        <map:entry key="3">
            <map:value>3</map:value>
        </map:entry>
    </map:map>
</results>
```

12.6.9 Div 演算子の適用

以下の例では推論の規則を使用し、Map A のキーと Map B の値を返します（Map A の値が Map B のキーと等しい場合）。

```
xquery version "1.0-ml";
let $mapA := map:map(
<map:map xmlns:map="http://marklogic.com/xdmp/map">
        <map:entry>
            <map:key>1</map:key>
            <map:value>1</map:value>
        </map:entry>
        <map:entry>
            <map:key>3</map:key>
            <map:value>3</map:value>
        </map:entry>
        </map:map>)
let $mapB := map:map(
<map:map xmlns:map="http://marklogic.com/xdmp/map">
        <map:entry>
            <map:key>2</map:key>
            <map:value>2</map:value>
        </map:entry>
        <map:entry>
            <map:key>3</map:key>
            <map:value>3</map:value>
            <map:value>3.5</map:value>
        </map:entry>
</map:map>)
return $mapA div $mapB
```

これは以下を返します。

```
<xml version="1.0" encoding="UTF-8">
<results warning="atomic item">
    <map:map xmlns:map="http://marklogic.com/xdmp/map"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xs="http://www.w3.org/2001/XMLSchema">
        <map:entry key="3">
            <map:value>3</map:value>
            <map:value>3.5</map:value>
        </map:entry>
    </map:map>
</results>
```

12.6.10 Mod 演算子の適用

以下の例では、上述の演算 2 つが実行されます。まず、Map A のキーと値が入れ替えられます。次に推論規則が適用され、Map A の値と Map B のキーがマッチされ、Map B の値が返されます。

```
xquery version "1.0-ml";
let $mapA := map:map(
<map:map xmlns:map="http://marklogic.com/xdmp/map">
        <map:entry>
            <map:key>1</map:key>
            <map:value>1</map:value>
        </map:entry>
        <map:entry>
            <map:key>3</map:key>
            <map:value>3</map:value>
        </map:entry>
        </map:map>)
let $mapB := map:map(
<map:map xmlns:map="http://marklogic.com/xdmp/map">
        <map:entry>
            <map:key>2</map:key>
            <map:value>2</map:value>
        </map:entry>
        <map:entry>
            <map:key>3</map:key>
            <map:value>3</map:value>
            <map:value>3.5</map:value>
        </map:entry>
</map:map>)
return $mapA mod $mapB
```

これは以下を返します。

```
<xml version="1.0" encoding="UTF-8">
<results warning="atomic item">
    <map:map xmlns:map="http://marklogic.com/xdmp/map"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xs="http://www.w3.org/2001/XMLSchema">
        <map:entry key="3.5">
            <map:value>3</map:value>
        </map:entry>
        <map:entry key="3">
            <map:value>3</map:value>
        </map:entry>
    </map:map>
</results>
```

13.0 関数値

この章では、関数値の使用方法について説明します。この方法で、関数値をパラメータとして XQuery 関数に渡すことができます。以下のセクションで構成されています。

- 関数値の概要
- xdmp:function XQuery プリミティブ型
- 関数値用の XQuery API
- 適用した関数がクエリステートメントによる更新である場合
- 関数値の使用例

13.1 関数値の概要

XQuery 関数はパラメータを取ります。このパラメータは、任意の XQuery 型です。通常、パラメータは文字列、日付、数値などです。XQuery には多くの型があり、堅牢な型付けサポートを提供します。ただし、場合によっては、指定した関数へのポインタを、別の関数のパラメータとして渡す方が簡単です。このような関数ポインタは「関数値」と呼ばれ、これを使用すると、より堅牢で、保守が容易なコードを記述できます。関数をパラメータとして渡すことをサポートするプログラミング言語では、それらのより高位の関数を呼び出すことがあります。MarkLogic サーバーの関数値は、他の言語で高位の関数が実行する処理の大半を実行しますが、関数を出力したり、匿名関数を作成したりすることはできません。代わりに、関数値を入出力することはできます。関数値は XQuery のプリミティブ型として実装されているためです。

関数値を他の関数に渡すには、渡す関数の名前を相手の関数に通知します。関数から返される実際の値は、クエリ実行時に動的に評価されます。このように関数値を渡すことで、関数に対するインターフェイスを定義し、そのデフォルト実装を用意できます。同時に、その関数の呼び出し元は、その関数の独自のバージョンを実装し、デフォルトバージョンの代わりに指定できます。

13.2 xdmp:function XQuery プリミティブ型

関数値は、`xdmp:function` XQuery プリミティブ型として定義されます。この型は、関数や変数の定義において使うことができます。また他のプリミティブ型と同様に XQuery で使えます。他のいくつかの MarkLogic サーバー XQuery プリミティブ型（`cts:query` や `map:map` など）とは異なり、`xdmp:function` XQuery プリミティブ型には XML シリアライゼーションはありません。

13.3 関数値用の XQuery API

関数値を渡すには、次の XQuery ビルトイン関数が使用されます。

- `xdmp:function`
- `xdmp:apply`

渡す関数を指定するには `xdmp:function` を使用し、渡した関数を実行するには `xdmp:apply` を使用します。これらの API の詳細およびシグネチャについては、『*MarkLogic XQuery and XSLT Function Reference*』を参照してください。

13.4 適用した関数がクエリステートメントによる更新である場合

`xdmp:function` 関数を使用して関数を適用するときに、MarkLogic サーバーは、クエリのコンパイル時に適用した関数のコンテンツを認識していません。そのため、`xdmp:apply` を呼び出すステートメントがクエリステートメントであり（つまり、更新式を含んでいないためにタイムスタンプで実行され）、適用される関数が更新を実行している場合は、`XDMP-UDATEFUNCTIONFROMQUERY` 例外をスローします。

適用するコードが更新を実行する場合、および呼び出し元クエリに更新ステートメントが含まれない場合は、呼び出し元クエリを更新ステートメントにする必要があります。クエリステートメントを更新ステートメントに変更するには、`xdmp:update` プロローグオプションを使用するか、更新の呼び出しをステートメント内に配置します。例えば、クエリを強制的に更新ステートメントとして実行するには、XQuery プロローグに以下を追加します。

```
declare option xdmp:update "true";
```

プロローグオプションを使用しない場合は、クエリ内の更新式が強制的に更新ステートメントとして実行されます。例えば、次の式は、クエリを強制的に更新ステートメントとして実行し、クエリに関するその他の部分は何も変更しません。

```
if ( fn:true() )
then ()
else xdmp:document-insert("fake.xml", <fake/>)
```

更新ステートメントとクエリステートメントの違いの詳細については、「MarkLogic サーバーのトランザクションとは」（22 ページ）を参照してください。

13.5 関数値の使用例

次の例は、再帰関数 `my:sum:sequences` を示しています。この関数は、`xdmp:function` 型を取り、シーケンスの最後に到達するまでその関数呼び出しを再帰的に適用します。この例は、呼び出し元が `my:add` 関数の独自の実装を提供して、`my:sum-sequences` 関数の動作を変更する方法を示します。`/sum.xqy` という名前の、次のようなライブラリモジュールを考えてみます。

```
xquery version "1.0-ml";
module namespace my="my-namespace";

(: Sum a sequence of numbers, starting with the
   starting-number (3rd parameter) and at the
   start-position (4th parameter).:)
declare function my:sum-sequence(
  $fun as xdmp:function,
  $items as item()*,
  $starting-number as item(),
  $start-position as xs:unsignedInt)
as item()
{
  if ($start-position gt fn:count($items)) then
  $starting-number
  else
    let $new-value := xdmp:apply($fun,$starting-number,
                        $items[$start-position])
    return
      my:sum-sequence($fun,$items,$new-value,
      $start-position+1)
};

declare function my:add($x,$y) {$x+ $y};
(: /sum.xqy :)
```

ここで、この関数を、次のメインモジュールを使用して呼び出します。

```
xquery version "1.0-ml";
import module namespace my="my-namespace" at "/sum.xqy";

let $fn := xdmp:function(xs:QName("my:add"))
return my:sum-sequence($fn,(1 to 100), 2, 1)
```

これにより、2 から 100 までのすべての数値の合計である 5052 が返されます。

別の式を使用して数値を加算する場合は、これと同じ関数の別の実装で XQuery ライブラリモジュールを作成し、それを代わりに指定できます。例えば、別の式を使用して数値を加算するとします。次のコード（2 番目の数値に 2 を乗算してから 1 番目の数値に加算する）を含む、/my.xqy という名前の別のライブラリモジュールを作成します。

```
xquery version "1.0-ml";
module namespace my="my-namespace";

declare function my:add($x,$y) {$x+ (2 * $y)};
(: /my.xqy :)
```

これで、次のように、my:add 関数の新しい実装を指定する my:sum-sequence 関数を呼び出すことができます。

```
xquery version "1.0-ml";
import module namespace my="my-namespace" at "/sum.xqy";

let $fn := xdmp:function(xs:QName("my:add"), "/my.xqy")
return my:sum-sequence($fn,(1 to 100), 2, 1)
```

これにより、新しい式を使用して 10102 が返されます。この手法により、呼び出し元が、渡される指定関数のまったく異なる実装を指定できます。

14.0 モジュラードキュメントアプリケーションによるコンテンツの再利用

この章では、他のコンテンツを含む XML を使用して、コンテンツを再利用するアプリケーションを作成する方法について説明します。以下のセクションで構成されています。

- モジュラードキュメント
- XInclude と XPointer
- CPF XInclude アプリケーションと API
- モジュラードキュメントアプリケーションで使用する XML の作成
- モジュラードキュメントアプリケーションの設定

14.1 モジュラードキュメント

「モジュラードキュメント」とは、その全体あるいは一部のコンテンツとして他のドキュメントの全体あるいは一部を参照する XML ドキュメントのことです。参照されたドキュメントの部分を取得し、そのコンテンツを、参照元の要素の子要素として配置することを、ドキュメントの「展開」と呼びます。すべての参照（展開されたドキュメント内のすべての参照を含む）を、展開する参照がなくなるまで繰り返し展開した場合、結果として得られたドキュメントは「展開ドキュメント」と呼ばれます。その後、展開ドキュメントは検索に利用できます。これで、1 つのドキュメント内のコンテンツ全体に基づいて、関連度で結果をランキングできます。モジュラードキュメントは、参照されたドキュメントおよびドキュメントの部分を指定する方法として、W3C 勧告である XInclude を使用します。

モジュラードキュメントを使用すると、コンテンツを管理および再利用できます。MarkLogic サーバーには、すべての XInclude 参照に基づいてドキュメントを展開する Content Processing Framework（CPF）アプリケーションが含まれています。この CPF アプリケーションは、元のドキュメントを変更することなく、新しいドキュメントとして展開ドキュメントを作成します。いずれかの部分が更新されると、展開ドキュメントが再作成されるため、展開ドキュメントでは自動的に最新の状態が維持されます。

ドキュメントの展開に関係するすべての作業は、モジュラードキュメント用の CPF アプリケーションが処理します。必要な作業は、XInclude 参照の含まれるドキュメントをデータベースで追加または更新することだけです。これで、CPF のドメインにあるすべてのドキュメントが自動的に展開されます。CPF の詳細については、『*Content Processing Framework Guide*』を参照してください。

コンテンツは、複数のドキュメントで参照することにより再利用できます。例えば、多くのさまざまなタイトルに含まれる常用文（法的免責表明、会社情報など）を使用する出版社であるとします。各書籍は、常用文を含むドキュメントを参照できます。CPF アプリケーションを使用している場合は、常用文が更新されると、すべてのドキュメントが自動的に更新されます。CPF アプリケーションを使用していない場合でも、シンプルな API 呼び出しを使用してドキュメントを更新できます。

14.2 XInclude と XPointer

モジュラードキュメントは、XInclude ならびに XPointer を使用します。

- XInclude: http://www.w3.org/TR/xinclude/
- XPointer: http://www.w3.org/TR/WD-xptr

XInclude は、XML ドキュメントを他の XML ドキュメントに含めるためのシンタックスを提供します。これにより、含めるドキュメントの相対または絶対 URI を指定できます。XPointer は、XML ドキュメントの一部を指定するためのシンタックスを提供します。これにより、XPath に基づいた（ただし、Xpath とまったく同じではない）シンタックスを使用してドキュメント内のノードを指定できます。MarkLogic サーバーは、XPointer フレームワークと、XPointer の `element()` および `xmlns()` スキーム、および `xpath()` スキームをサポートしています。

- `element()` スキーム : http://www.w3.org/TR/2002/PR-xptr-element-20021113/
- `xmlns()` スキーム : http://www.w3.org/TR/2002/PR-xptr-xmlns-20021113/
- `xpath()` スキームは、W3C 勧告ではありませんが、このスキームを使用すると、単純な XPath でドキュメントの一部を指定できます。

`xmlns()` スキームは、XPointer フレームワークで名前空間プレフィックスのバインドに使用されます。`element()` スキームは、XInclude `href` 属性内のドキュメントから選択する要素を指定するために使用されるシンタックスの 1 つです。`xpath()` スキームは、ドキュメントから要素を選択するための、もう 1 つのシンタックス（`element()` スキームよりも XPath に似ている）です。

これらの各スキームは、`xpointer` という名前の属性内で使用されます。`xpointer` 属性は、`<xi:include>` 要素の属性の 1 つです。`idref` に対応する文字列を指定すると、その ID 属性を持つ要素が選択されます（「例：シンプルな ID」（182 ページ）を参照）。

以降の例は、XPointer を使用してドキュメントの一部を選択する XInclude を示します。

- 例：シンプルな ID
- 例：xpath() スキーム
- 例：element() スキーム
- 例：xmlns() と xpath() スキーム

14.2.1　例：シンプルな ID

次のようなコンテンツを持つドキュメント /test2.xml があるとします。

```
<el-name>
  <p id="myID">This is the first para.</p>
  <p>This is the second para.</p>
</el-name>
```

次のコードは、id 属性の値が myID である要素を /test2.xml ドキュメントから選択します。

```
<xi:include href="/test2.xml" xpointer="myID" />
```

この <xi:include> 要素を展開すると、次のようになります。

```
<p id="myID" xml:base="/test2.xml">This is the first
para.</p>
```

14.2.2　例：xpath() スキーム

次のようなコンテンツを持つドキュメント /test2.xml があるとします。

```
<el-name>
  <p id="myID">This is the first para.</p>
  <p>This is the second para.</p>
</el-name>
```

次のコードは、ルート要素 el-name の子である 2 番目の p 要素を /test2.xml ドキュメントから選択します。

```
<xi:include href="/test2.xml" xpointer="xpath
(/el-name/p[2])" />
```

この <xi:include> 要素を展開すると、次のようになります。

```
<p xml:base="/test2.xml">This is the second para.</p>
```

14.2.3　例：element() スキーム

次のようなコンテンツを持つドキュメント /test2.xml があるとします。

```
<el-name>
  <p id="myID">This is the first para.</p>
  <p>This is the second para.</p>
</el-name>
```

次のコードは、ルート要素 el-name の子である 2 番目の p 要素を /test2.xml ドキュメントから選択します。

```
<xi:include href="/test2.xml" xpointer="element(/1/2)" />
```

この <xi:include> 要素を展開すると、次のようになります。

```
<p xml:base="/test2.xml">This is the second para.</p>
```

14.2.4　例：xmlns() と xpath() スキーム

次のようなコンテンツを持つドキュメント /test2.xml があるとします。

```
<pref:el-name xmlns:pref="pref-namespace">
  <pref:p id="myID">This is the first para.</pref:p>
  <pref:p>This is the second para.</pref:p>
</pref:el-name>
```

次のコードは、ルート要素 pref:el-name の子である 1 番目の pref:p 要素を /test2.xml ドキュメントから選択します。

```
<xi:include href="/test2.xml"
            xpointer="xmlns(pref=pref-namespace)
                      xpath(/pref:el-name/pref:p[1])" />
```

この <xi:include> 要素を展開すると、次のようになります。

```
<pref:p id="myID" xml:base="/test2.xml"
   xmlns:pref="pref-namespace">This is the first
   para.</pref:p>
```

xmlns() スキームには、XPointer の名前空間プレフィックスを入力する必要があります。プレフィックスはクエリコンテキストから継承されません。

14.3　CPF XInclude アプリケーションと API

このセクションでは、XInclude CPF アプリケーションコードについて説明します。以下のように構成されています。

- XInclude コードと CPF パイプライン
- 必須のセキュリティ権限 — xinclude ロール

14.3.1　XInclude コードと CPF パイプライン

自分でモジュラードキュメントのアプリケーションを作成するか、CPF アプリケーションの XInclude 処理パイプランを利用することができます。CPF の詳細については、『*Content Processing Framework Guide*』を参照してください。モジュラードキュメントのアプリケーションを作成する際に使用される XQuery ライブラリと CPF コンポーネントは、次のとおりです。

- XQuery モジュールライブラリ `xinclude.xqy`。このライブラリの主要な関数は `xinc:node-expand` 関数です。この関数は、1 つのノードであらゆる XInclude 参照を再帰的に展開し、完全に展開されたノードを返します。
- XQuery モジュールライブラリ `xpointer.xqy`。
- XInclude パイプラインとそれに関連するアクション。
- XInclude パイプラインに基づいて、XInclude パイプラインへの次の `<options>` を使用するカスタムパイプラインを作成できます。これらのオプションは、パイプラインがアタッチされているドメインにあるドキュメントの XInclude 参照の展開を制御します。
    - `<destination-root>` は、ドキュメントの展開バージョンの保存先となるディレクトリを指定します。これは、データベース内のディレクトリパスにする必要があります。展開ドキュメントは、このルートと未展開ドキュメントのベース名を連結した URI に保存されます。例えば、未展開ドキュメントの URI が `/mydocs/unexpanded/doc.xml` で、destination-root が `/expanded-docs/` に設定されている場合、このドキュメントは、URI が `/expanded-docs/doc.xml` であるドキュメントに展開されます。
    - `<destination-collection>` は、展開バージョンの配置先のコレクションを指定します。パイプライン内で複数の `<destination-collection>` 要素を指定することで、複数のコレクションを指定できます。
    - `<destination-quality>` は、展開バージョンのドキュメントクオリティを指定します。これは、整数値にする必要があります。大きい正の値にすると、ドキュメントの一致に関する関連度スコアが増加します。小さい負の値にすると、関連度スコアが減少します。ドキュメントにおけるクオリティのデフォルト値は 0 です。つまり、関連度スコアは変化しません。
    - デフォルトでは、展開前のソースと同じ値を使用します。

14.3.2　必須のセキュリティ権限ー xinclude ロール

XInclude コードでは、次の権限が必要です。

- `xdmp:with-namespaces`
- `xdmp:value`

したがって、ドキュメントを展開するすべてのユーザーには、これらの権限が必要です。xinclude という事前定義されたロールがあり、このコードの実行に必要な権限が与えられています。XInclude CPF アプリケーションで使用されている XInclude コードを実行するためには、ユーザーにこの xinclude ロールが割り当てられているか、あるいは上記の権限が必要です。

14.4　モジュラードキュメントアプリケーションで使用する XML の作成

XInclude を使用するための基本シンタックスは比較的シンプル。参照されたドキュメントごとに、参照されたドキュメントの URI 値を含む <xi:include> 要素と href 属性を含めます。この URI 値は、<xi:include> 要素を持つドキュメントの相対 URI、またはデータベース内のドキュメントの絶対 URI です。ドキュメントが展開されると、<xi:include> 要素が、参照されたドキュメントに置換されます。このセクションは、次のように構成されています。

- <xi:include> 要素
- <xi:fallback> 要素
- シンプルな例

14.4.1　<xi:include> 要素

他のドキュメント内のコンテンツへの参照を含む要素は、<xi:include> 要素です。xi は、http://www.w3.org/2001/XInclude 名前空間にバインドされています。各 xi:include 要素には href 属性があり、この属性には、含まれるドキュメントの URI が含まれます。この URI は、<xi:include> 要素を含むドキュメントの相対 URI、またはデータベース内ドキュメントの絶対 URI です。

14.4.2　<xi:fallback> 要素

XInclude 仕様には、「フォールバック」コンテンツを指定するメカニズムがあります。フォールバックコンテンツとは、ドキュメント展開時に XInclude 参照が見つからなかった場合に使用されるコンテンツのことです。フォールバックコンテンツを指定するには、<xi:include> 要素の子要素として <xi:fallback> 要素を追加します。フォールバックコンテンツはオプションですがグッドプラクティスとしては指定しておいたほうがよいです。xi:include の href 属性が適切に解決される限り、<xi:fallback> 要素がないドキュメントは適切に展開されます。ただし、xi:include の href 属性が適切に解決されない場合、および解決されない参照用の <xi:fallback> 要素がない場合は、展開が失敗し、XI-BADFALLBACK 例外がスローされます。

<xi:include> 要素と <xi:fallback> 要素を指定した例を次に示します。

```
<xi:include href="/blahblah.xml">
  <xi:fallback><p>NOT FOUND</p></xi:fallback>
</xi:include>
```

URI が /blahblah.xml であるドキュメントが見つからなかった場合は、この <xi:include> 要素を持つドキュメントを展開するときに NOT FOUND が代用されます。

また、次のように <xi:include> 要素を <xi:fallback> 要素内に置いて、データベース内のいずれかのコンテンツにフォールバックすることもできます。

```
<xi:include href="/blahblah.xml">
  <xi:fallback><xi:include href="/fallback.xml" />
  </xi:fallback>
</xi:include>
```

この要素は、ドキュメントを展開するときに URI が /blahblah.xml であるドキュメントを含め、それが見つからない場合は /fallback.xml 内のコンテンツを使用するように指示しています。

14.4.3　シンプルな例

2 つのドキュメントを作成してから、一方を XInclude 参照で展開するシンプルな例を次に示します。

```
xquery version "1.0-ml";
declare namespace xi="http://www.w3.org/2001/XInclude";

xdmp:document-insert("/test1.xml", <document>
  <p>This is a sample document.</p>
  <xi:include href="test2.xml"/>
</document>);

xquery version "1.0-ml";

xdmp:document-insert("/test2.xml",
  <p>This document will get inserted where
     the XInclude references it.</p>);

xquery version "1.0-ml";
import module namespace
xinc="http://marklogic.com/xinclude"
     at "/MarkLogic/xinclude/xinclude.xqy";

xinc:node-expand(fn:doc("/test1.xml"))
```

xinc:node-expand 呼び出しから返された展開ドキュメントは、次のようになります。

```
<document>
  <p>This is a sample document.</p>
```

```
  <p xml:base="/test2.xml">This document will get inserted
     where the XInclude references it.</p>
</document>
```

含まれるコンテンツの URI からのベース URI が、展開されたノードの `xml:base` 属性として追加されています。

次の例に示すように、フォールバックコンテンツを含めることができます。

```
xquery version "1.0-ml";
declare namespace xi="http://www.w3.org/2001/XInclude";

xdmp:document-insert("/test1.xml", <document>
  <p>This is a sample document.</p>
  <xi:include href="/blahblah.xml">
    <xi:fallback><p>NOT FOUND</p></xi:fallback>
  </xi:include>
</document>);

xquery version "1.0-ml";

xdmp:document-insert("/test2.xml",
  <p>This document will get inserted where the XInclude
     references it.</p>);

xquery version "1.0-ml";

xdmp:document-insert("/fallback.xml",
  <p>Sorry, no content found.</p>);

xquery version "1.0-ml";
import module namespace
xinc="http://marklogic.com/xinclude"
     at "/MarkLogic/xinclude/xinclude.xqy";

xinc:node-expand(fn:doc("/test1.xml"))
```

`xinc:node-expand` 呼び出しから返された展開ドキュメントは、次のようになります。

```
<document>
  <p>This is a sample document.</p>
  <p xml:base="/test1.xml">NOT FOUND</p>
</document>
```

14.5 モジュラードキュメントアプリケーションの設定

モジュラードキュメント CPF アプリケーションを設定するには、CPF をインストールし、XInclude リンクを持つドキュメントの展開場所となるドメインを作成する必要があります。Content Processing Framework に関する詳細（設定手順や機能情報など）については、『*Content Processing Framework Guide*』ガイドを参照してください。

XInclude モジュラードキュメントアプリケーションを設定するには、次の手順を実行します。

1. Content Processing がインストールされていない場合は、データベースにインストールします。例えば、データベース名が `modular` である場合は、管理画面で［Databases］>［modular］>［Content Processing］リンクをクリックします。インストールされていない場合は、そのことが［Content Processing Summary］ページに表示されます。インストールされていない場合は、［Install］タブをクリックし、［install］をクリックします（変換を有効にするかどうかに関わらずインストールできます）。

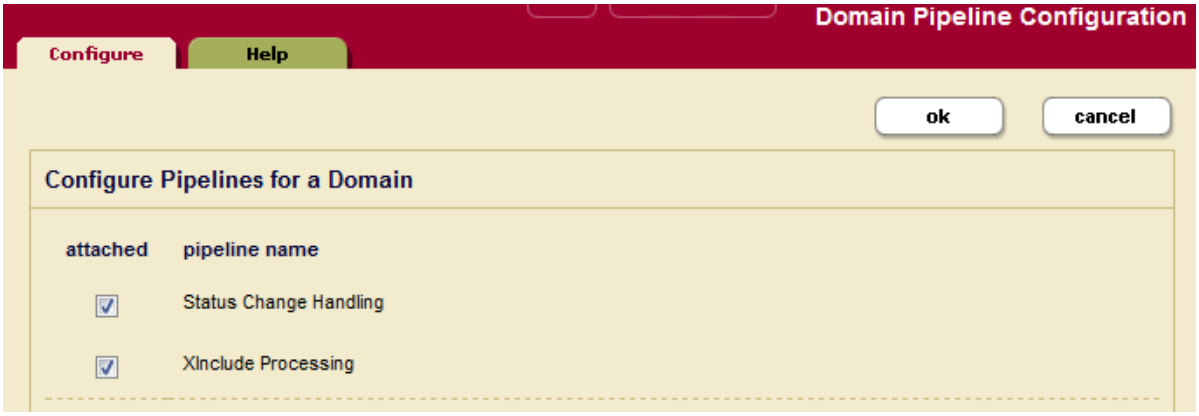
2. 左側のツリーメニューで［domains］リンクをクリックします。新しいドメインを作成するか、既存のドメインを変更して、XInclude 処理で処理するドキュメントの範囲が含まれるようにします。ドメインの詳細については、『*Content Processing Framework Guide*』ガイドを参照してください。

3. 選択したドメインで、左側のツリーメニューから［Pipelines］リンクをクリックします。

4. ［`Status Change Handling`］および［`XInclude Processing`］パイプラインをチェックします。アプリケーションでの必要性に応じて、他のパイプラインをアタッチまたはデタッチすることもできます。

   注： `XInclude Processing` パイプラインのいずれかの `<options>` 設定を変更する場合は、そのパイプラインを別のファイルにコピーし、変更を加え（`<pipeline-name>` 要素の値も必ず変更する）、パイプラインの XML ファイルを読み込みます。これでドメインにアタッチできるようになります。XInclude パイプラインのオプションの詳細については、「CPF XInclude アプリケーションと API」（183 ページ）を参照してください。

5. [OK] ボタンをクリックします。アタッチされたパイプラインが［Domain Pipeline Configuration］画面に表示されます。



これで、ドメインで挿入または更新された、XInclude を含むすべてのドキュメントが展開されます。展開ドキュメントの URI は、末尾が `_expanded.xml` になります。例えば、URI が `/test.xml` であるドキュメントを挿入すると、URI が `/test_xml_expanded.xml` の展開ドキュメントが作成されます（XInclude パイプラインオプションを変更しなかった場合）。

注：ドメインの範囲内に既存の XInclude ドキュメントがある場合、それらのドキュメントは更新されるまで展開されません。

15.0 アプリケーションサーバーのアクセス、出力、およびエラーの制御

MarkLogic サーバーは、アプリケーションサーバーに対して XQuery プログラムを評価します。この章では、出力を、アプリケーションサーバー設定で制御する方法と、XQuery ビルトイン関数を使用して制御する方法を説明します。この章で説明する機能は、主に HTTP アプリケーションサーバーに適用されますが、一部は XDBC サーバーやタスクサーバーにも当てはまります。この章は、以下のセクションで構成されています。

- カスタム HTTP サーバーエラーページの作成
- HTTP アプリケーションサーバーの URL 書き換えの設定
- SGML エンティティの出力
- 出力のエンコーディングの指定
- アプリケーションサーバーレベルで出力オプションの指定

15.1 カスタム HTTP サーバーエラーページの作成

このセクションでは、HTTP サーバーエラーページを使用する方法について説明します。以下の部分で構成されています。

- カスタム HTTP エラーページの概要
- エラー XML の形式
- カスタムエラーページの設定
- Modules データベース用のエラーハンドラドキュメントには実行パーミッションが必要
- カスタムエラーページの例

15.1.1 カスタム HTTP エラーページの概要

カスタム HTTP サーバーエラーページを使用すると、アプリケーション例外を XQuery プログラムにリダイレクトできます。400 または 500HTTP 例外（503 エラーを除く）がスローされた場合は必ず、XQuery モジュールが評価され、その結果がクライアントに返されます。カスタムエラーページでは、通常、ユーザーフレンドリーなメッセージがエンドユーザーに表示されます。ただし、エラーページは XQuery モジュールであるため、任意の処理を実行できます。

XQuery モジュールは、`xdmp:get-response-code` API を使用して、HTTP エラーコード、および HTTP レスポンスのコンテンツを取得できます。エラーハンドラ用の XQuery モジュールは、XQuery スタックトレースが存在する場合はその XQuery スタックトレースにもアクセスできます。XQuery スタックトレースは、`$error:errors`

（XQuery `1.0-ml` ダイアレクトの場合）および `$err:errors`（XQuery `0.9-ml` ダイアレクトの場合）という名前で外部変数としてモジュールに渡されます。どちらも同じ名前空間にバインドされていますが、`err` プレフィックスは `0.9-ml` で、`error` プレフィックスは `1.0-ml` で事前定義されています。

エラーが 503（使用不可）エラーである場合は、エラーハンドラが呼び出されず、503 例外がクライアントに返されます。

エラーページ自体が例外をスローした場合、その例外は、エラーページのエラーコードとともにクライアントに渡されます。また、元のエラーコードと例外が含まれているスタックトレースも含まれます。

15.1.2　エラー XML の形式

エラーメッセージは、`error.xsd` スキーマを使用する XML エラースタックトレースとともにスローされます。スタックトレースには、スローされる例外、行番号、および XQuery バージョンが含まれます。スタックトレースは、`$error:errors` 外部変数を使用してカスタムエラーページからアクセスできます。シンタックスエラーのある XQuery モジュールのエラー XML 出力例を次に示します。

```
<error:error xsi:schemaLocation=
   "http://marklogic.com/xdmp/error
    error.xsd"
    xmlns:error="http://marklogic.com/xdmp/error"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <error:code>XDMP-CONTEXT</error:code>
  <error:name>err:XPDY0002</error:name>
  <error:xquery-version>1.0-ml</error:xquery-version>
  <error:message>Expression depends on the context where
                 none is defined</error:message>
  <error:format-string>XDMP-CONTEXT: (err:XPDY0002)
     Expression depends on the context where none
     is defined</error:format-string>
  <error:retryable>false</error:retryable>
  <error:expr/> <error:data/>
  <error:stack>
    <error:frame>
     <error:uri>/blaz.xqy</error:uri>
     <error:line>1</error:line>
     <error:xquery-version>1.0-ml</error:xquery-version>
    </error:frame>
  </error:stack>
</error:error>
```

15.1.3　カスタムエラーページの設定

HTTP アプリケーションサーバー用のカスタムエラーページを設定するには、HTTP サーバーの［Error Handler］フィールドに XQuery モジュールの名前を入力します。パスの先頭がスラッシュ（/）でない場合は、アプリケーションサーバーからの相対パスです。パスの先頭がスラッシュ（/）の場合は、「XQuery モジュールおよび XSLT スタイルシートのインポートとパスの解決」（87 ページ）で説明するインポートルールが適用されます。

15.1.4　Modules データベース用のエラーハンドラドキュメントには実行パーミッションが必要

Modules データベースを使用するようにアプリケーションサーバーが設定されている（つまり、XQuery コードをデータベースに格納して実行する）場合は、エラーハンドラモジュールドキュメントに実行パーミッションを付与する必要があります。実行パーミッションはロールとペアになっており、エラーハンドラを実行するには、アプリケーションサーバーのすべてのユーザーがそのロールを持つ必要があります。そのロールを持っていないユーザーは、エラーハンドラモジュールを実行できません。また、エラーハンドラがエラーを検出して処理するのではなく、401（許可されていない）エラーが発生します。

エラーハンドラに実行パーミッションが必要であるため、実際にはエラーハンドラの実行が許可されていないユーザーがアプリケーションサーバーにアクセスしようとすると、認証されるまでそのユーザーは、アプリケーションサーバーに設定されたデフォルトユーザーとして実行されます。認証に失敗すると、エラーハンドラがデフォルトユーザーとして呼び出されますが、デフォルトユーザーにはエラーハンドラを実行するパーミッションがないため、エラーハンドラを見つけられず、404 エラー（見つからない）が返されます。そのため、すべてのユーザー（許可されていないユーザーを含む）にエラーハンドラを実行するパーミッションを設定する場合は、デフォルトユーザーにロール（権限が付与されていなくてもよい）を設定し、そのロールとペアになっているエラーハンドラに実行パーミッションを割り当てる必要があります。

15.1.5　カスタムエラーページの例

次の XQuery モジュールは、非常にシンプルな XQuery エラーハンドラです。

```
xquery version "1.0-ml";

declare variable $error:errors as node()* external;

xdmp:set-response-content-type("text/plain"),
xdmp:get-response-code(),
$error:errors
```

これは、例外をスローするページからすべての情報を返すだけです。通常のエラーページでは、一部またはすべての情報を使用し、ユーザーフレンドリーな表現でユーザーに表示します。エラーページには任意の XQuery を記述できるため、アプリケーション管理者へのメール送信や、別のページへのリダイレクトなど、さまざまな処理を実行できます。

15.2 HTTP アプリケーションサーバーの URL 書き換えの設定

このセクションでは、HTTP サーバーの URL リライタ機能を使用する方法を説明します。URL 書き換えの詳細については、「インタプリタ型 XQuery リライタの作成による REST Web サービスのサポート」（207 ページ）を参照してください。

このセクションでは、次の内容を取り上げます。

- URL 書き換えの概要
- シェークスピアの XML コンテンツの読み込み
- シンプルな URL リライタ
- URL 書き換えモジュールの作成
- 内部 URL へのアクセスの禁止
- URL 書き換えとページ相対 URL
- URL Rewrite トレースイベントの使用

15.2.1 URL 書き換えの概要

URL を使用して、任意の MarkLogic サーバーリソースにアクセスできます。これは、Representational State Transfer（REST）サービスの基本的な特性です。生の形式の URL は、リソースの物理的な場所を反映している（データベース内のドキュメントの場合）か、または次の形式である必要があります。

```
http://<dispatcher-program.xqy>?instructions=foo
```

一般に Web アプリケーションのユーザーは、生のクエリ文字列パラメータより、短くすっきりした URL を好みます。簡潔な URL は、「クリーンな URL」とも呼ばれ、憶えやすく、より短時間で入力できます。URL がページのコンテンツと明確に関連している場合は、エラーが起こりにくくなります。また、クローラや検索エンジンは、多くの場合、Web ページの URL を使用して、URL にインデックス付けするかどうか、およびその URL に設定するランキングを判断します。例えば、次のような URL があるとします。

```
http://marklogic.com/technical/features.html
```

検索エンジンは、このように適切な構造の URL に対しては、次のようなわかりにくい形式の URL よりも高いランキングを設定すると考えられます。

```
http://marklogic.com/document?id=43759
```

「RESTful」な環境では、URL は適切な構造で作成され、予測可能であり、ドキュメントやプログラムの物理的な位置とは切り離されている必要があります。HTTP サーバーは、適切な構造の外部 URL による HTTP リクエストを受信したときに、ドキュメントやプログラムの内部 URL に透過的にマッピングできることが必要です。

URL リライタ機能を使用すると、外部 URL を内部 URL に書き換えることができるように HTTP アプリケーションサーバーを設定できます。これにより、任意の URL を使用して任意のリソース（Web ページ、ドキュメント、XQuery プログラム、および引数）を柔軟にポイントできます。MarkLogic サーバーで実装されている URL リライタの動作は Apache mod_rewrite モジュールに似ており、管理者のみが、書き換え操作を実行する XQuery プログラムを記述します。

URL 書き換えは、内部リダイレクトメカニズムから発生するため、クライアントは、URL がどのように書き換えられたかを認識しません。このため、Web サイトにアクセスした人には、そのアドレスが内部でどのように機能しているかはわかりません。内部 URL は、存在しない URL に書き換えることで、必要に応じてブロックしたり、アクセス不可にしたりすることもできます（「内部 URL へのアクセスの禁止」（199 ページ）を参照）。

XSLT スタイルシートを直接呼び出す URL リライタを作成する方法については、『*XQuery and XSLT Reference Guide*』の Invoking Stylesheets Directly Using the XSLT Rewriter を参照してください。

> **注**：アプリケーションコードが modules データベースにある場合は、モジュールを実行するために、URL リライタにデフォルトアプリケーションサーバーユーザー（デフォルトでは nobody）のパーミッションが必要です。これは、データベースに格納されているエラーハンドラの場合と同様です（「Modules データベース用のエラーハンドラドキュメントには実行パーミッションが必要」（192 ページ）を参照）。

15.2.2　シェークスピアの XML コンテンツの読み込み

この章の例では、シェークスピア劇が XML ファイルの形式でデータベースに読み込まれていることを前提としています。XML コンテンツを `Documents` データベースに読み込む最も簡単な方法は、以下のとおりです。

- Query Console を開き、[Content Source] を `Documents` に設定します。
- Query Console ウィンドウに以下のクエリをコピーします。

- ［Run］をクリックして、クエリを実行します。

次のクエリで、zip ファイルとして取得した、シェークスピア劇が含まれている XML ファイルが現在のデータベースに読み込まれます。

```
xquery version "1.0-ml";

import module namespace ooxml= "http://marklogic.com/openxml"
          at "/MarkLogic/openxml/package.xqy";

xdmp:set-response-content-type("text/plain"),
let $zip-file :=
 xdmp:document-get("http://www.ibiblio.org/bosak/xml/eg/
 shaks200.zip")

return for $play in ooxml:package-uris($zip-file)
  where fn:contains($play , ".xml")
    return (let $node := xdmp:zip-get ($zip-file, $play)
      return xdmp:document-insert($play, $node) )
```

注： シェークスピア劇の XML ソースは、zip ファイルに含まれている shaksper.htm ファイルに記載されている著作権の対象です。

15.2.3　シンプルな URL リライタ

URL を書き換える最も簡単な方法は、ブラウザによってサーバーに設定された外部 URL を読み取り、サーバーが認識する生の URL に変換する URL 書き換えスクリプトを作成することです。

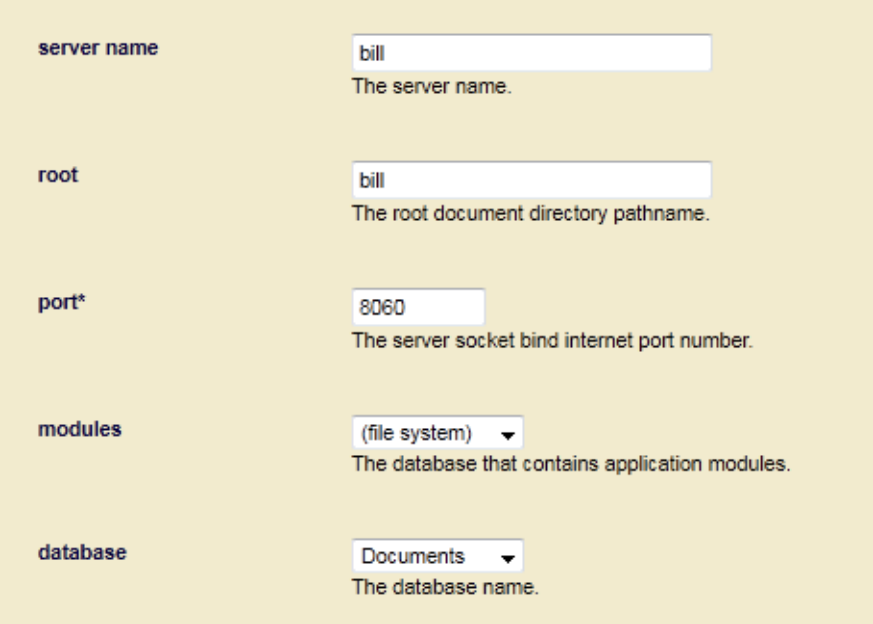
次の手順では、このセクションで説明するシンプルな URL リライタの例のために MarkLogic サーバーを設定する方法について説明します。

1. 管理画面の左側のツリーメニューで、［Groups］アイコンをクリックします。

2. HTTP サーバーを定義するグループ（［Default］など）をクリックします。

3. 左側のツリーメニューにある［App Servers］アイコンをクリックし、新しい HTTP アプリケーションサーバーを作成します。

4. HTTP アプリケーションサーバーに `bill` という名前を付け、ポート `8060` を割り当てます。ルートディレクトリとして `bill` を、データベースとして `Documents` を指定します。



5. MarkLogic ルートディレクトリの下に、`bill` という名前の新しいディレクトリを作成します。

6. `mac.xqy` という名前のモジュールを作成します。これは、`fn:doc` 関数を使用して、データベース内の `macbeth.xml` ファイルを呼び出すシンプルなモジュールです。

```
xdmp:set-response-content-type("text/html")
fn:doc("macbeth.xml")
```

7. `mac.xqy` を */<MarkLogic_Root>/bill* ディレクトリに保存します。

8. ブラウザを開き、次の URL を入力して `macbeth.xml` を（生の XML 形式で）表示します。

```
http://localhost:8060/mac.xqy
```

「よりクリーン」でわかりやすい URL は次のようになります。

```
http://localhost:8060/macbeth
```

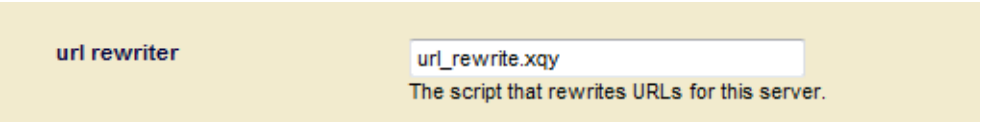
この URL 書き換えを実行するには、次の手順に従います。

1. スクリプト `url_rewrite.xqy` を作成します。このスクリプトは、`xdmp:get-request-url` 関数を使用してユーザー指定の URL を読み取り、`fn:replace` 関数を使用して URL の `/macbeth` 部分を `/mac.xqy` に変換します。

```
xquery version "1.0-ml";

let $url := xdmp:get-request-url()
return fn:replace($url, "^/macbeth$", "/mac.xqy")
```

2. `url_rewrite.xqy` スクリプトを `/<MarkLogic_Root>/bill` ディレクトリに保存します。

3. 管理画面で、`bill` アプリケーションサーバーを開き、`[url rewriter]` フィールドで `url_rewrite.xqy` を指定します。



4. 次の URL をブラウザに入力します。

```
http://localhost:8060/macbeth
```

注： URL は `fn:replace` 関数によって `/mac.xqy` に変換されますが、ページが開いた後のブラウザの URL フィールドには `/Macbeth` が表示されます。

xdmp:get-request-url 関数は、URL のスキームとネットワークロケーション（ドメイン名または *host_name:port_number*）に続く部分を返します。上記の例では、xdmp:get-request-url は /Macbeth を返します。リクエストパスのみ（パラメータなし）を返す xdmp:get-request-path とは異なり、xdmp:get-request-url 関数は、リクエストパスと、URL 内のクエリパラメータ（リクエストフィールド）を返します。これらはすべて、URL 書き換えスクリプトで変更できます。

より複雑な URL 書き換えモジュールを作成できます。「URL 書き換えモジュールの作成」（198 ページ）および「インタプリタ型 XQuery リライタの作成による REST Web サービスのサポート」（207 ページ）を参照してください。

15.2.4　URL 書き換えモジュールの作成

このセクションでは、シンプルな URL 書き換えモジュールを作成する方法を説明します。堅牢な URL 書き換えソリューションについては、「インタプリタ型 XQuery リライタの作成による REST Web サービスのサポート」（207 ページ）を参照してください。

正規表現でパターンマッチング機能を使用して、柔軟性のある URL 書き換えモジュールを作成できます。例えば、ユーザーが URL のスキームとネットワークロケーションの後に / を入力した（例：http://localhost:8060/）だけで、それが /mac.xqy に書き換えられるようにするとします。

```
xquery version "1.0-ml";

let $url := xdmp:get-request-url()
return fn:replace($url,"^/$", "/mac.xqy")
```

この例では、ブラウザのアドレスバーに .xqy 拡張子が表示されないようにし、静的 URL を動的 URL（? 文字を含む）に変換します。例えば、次のようにします。

```
let $url := xdmp:get-request-url()

return fn:replace($url,
      "^/product-([0-9]+)\.html$",
      "/product.xqy?id=$1")
```

製品 ID は任意の数値です。例えば、URL /product-12.html は /product.xqy?id=12 に、/product-25.html は /product.xqy?id=25 に変換されます。

検索エンジン最適化の専門家は、メインのキーワードを URL に表示することを推奨しています。次の URL 書き換え手法では、URL に製品名を表示できます。

```
let $url := xdmp:get-request-url()

return fn:replace($url,
      "^/product/([a-zA-Z0-9_-]+)/([0-9]+)\.html$",
      "/product.xqy?id=$2")
```

製品名は任意の文字列です。例えば、/product/canned_beans/12.html は /product.xqy?id=12 に、/product/cola_6_pack/8.html は /product.xqy?id=8 に変換されます。

HTTP サーバー上の複数のページを書き換える必要がある場合は、次のような URL 書き換えスクリプトを作成できます。

```
let $url := xdmp:get-request-url()

let $url := fn:replace($url, "^/Shrew$", "/tame.xqy")
let $url := fn:replace($url, "^/Macbeth$", "/mac.xqy")
let $url := fn:replace($url, "^/Tempest$", "/tempest.xqy")

return $url
```

15.2.5　内部 URL へのアクセスの禁止

URL リライタ機能を使用して、ユーザーによる内部 URL へのアクセスをブロックすることもできます。例えば、customer_list.html への直接アクセスを禁止するには、URL リライタスクリプトを次のように記述します。

```
let $url := xdmp:get-request-url()

return if (fn:matches($url,"^/customer_list.html$"))
       then "/nowhere.html"
       else fn:replace($url,"^/price_list.html$",
        "/prices.html")
```

ここで /nowhere.html は、存在しないページです。ブラウザには「404 Not Found」エラーが表示されます。または、xdmp:random を使用して生成されるランダムな数値が含まれる URL へリダイレクトします。この数値は、存在しない URL を確実に生成できる他のスキームを使用して生成することもできます。

15.2.6　URL 書き換えとページ相対 URL

URL を、ページ相対 URL を使用するページに書き換えると、問題が発生することがあります。これは、相対 URL がクライアントによって解決されるためです。クライアントが使用する外部 URL のディレクトリパスが、サーバーにおける内部 URL と異なる場合は、ページ相対リンクが正しく解決されません。

ページ相対 URL を使用するページに URL を書き換える場合は、ページ相対 URL をサーバー相対 URL または標準（カノニカル）URL に変換してください。例えば、アプリケーションが `C:\Program Files\MarkLogic\myapp` にあり、次のように、ページ相対 URL を持つフレームセットがページで構築されているとします。

```
<frame src="top.html" name="headerFrame">
```

この URL は、次のようなサーバー相対 URL に変更する必要があります。

```
<frame src="/myapp/top.html" name="headerFrame">
```

または次のように標準（カノニカル）URL にします。

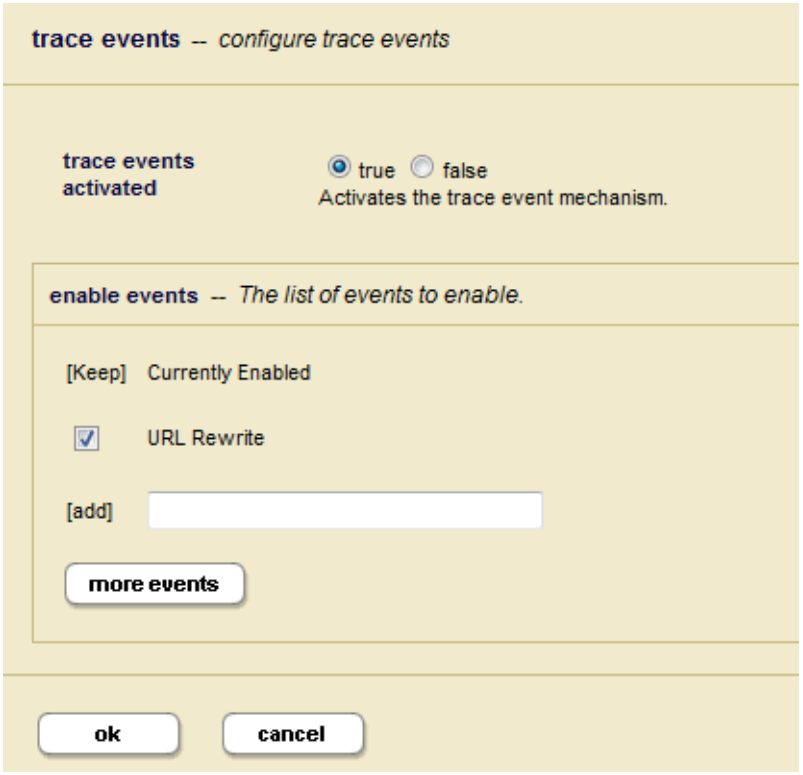
```
<frame src="http://127.0.0.1:8000/myapp/top.html"
name="headerFrame">
```

15.2.7　URL Rewrite トレースイベントの使用

URL Rewrite トレースイベントを使用すると、URL 書き換えモジュールのデバッグ時に役立ちます。URL Rewrite トレースイベントを移用するには、設定でトレースを（グループレベルで）有効にし、イベントを設定する必要があります。

1. 管理画面にログインします。
2. [Groups] > [「*グループ名*」] > ［Diagnostics］を選択します。

   ［Diagnostics Configuration］ページが表示されます。
3. ［`trace events activated`］の［`true`］ボタンをクリックします。
4. ［add］フィールドで、次のように入力します。`URL Rewrite`
5. ［ok］ボタンをクリックして、イベントをアクティブ化します。



URL Rewrite トレースイベントを設定した後、URL Rewrite スクリプトが呼び出されると必ず、以下に示すような行が `ErrorLog.txt` ファイルに追加されます。これは、クライアントから受信した URL と、URL リライタによる変換後の URL を示しています。

```
2009-02-11 12:06:32.587 Info: [Event:id=URL Rewrite]
Rewriting URL /Shakespeare to /frames.html
```

注： トレースイベントは、開発およびデバッグのツールとして設計されており、MarkLogic サーバーの全体的なパフォーマンスの低下を引き起こす場合があります。また、多くのトレースイベントを有効にすると、多数のドキュメントを処理している場合は特に、大量のメッセージが生成されます。デバッグ中でない場合は、パフォーマンスを最大にするために、トレースイベントを無効にしてください。

15.3 SGML エンティティの出力

このセクションでは、MarkLogic サーバーの SGML エンティティ出力制御について説明します。以下のように構成されています。

- さまざまな SGML マッピングの設定について
- アプリケーションサーバー設定での SGML マッピング設定
- XQuery プログラムにおける SGML マッピングの指定

15.3.1 さまざまな SGML マッピングの設定について

SGML の文字エンティティは、先頭のアンパサンド（`&`）文字と末尾のセミコロン（`;`）文字で区切られた名前です。エンティティは、特定の文字にマッピングされます。このマークアップは、SGML で使用され、XML でも利用されることがあります。MarkLogic サーバーでは、[Output SGML Character Entites] ドロップダウンリストを使用してアプリケーションサーバーレベルで、またはビルトイン関数 `xdmp:quote` または `xdmp:save` で `<output-sgml-character-entities>` を使用して、出力での XML のシリアライゼーション時の SGML 文字エンティティを制御できます。SGML 文字が（アプリケーションサーバーまたはビルトイン関数で）マッピングされている場合、SGML マッピングが含まれる Unicode 文字はいずれも、対応する SGML エンティティとして出力されます。デフォルトでは `none` であり、どの文字も SGML エンティティとして出力されません。

このマッピングは、W3C XML Entities for Characters 仕様に基づいています。

- http://www.w3.org/TR/2008/WD-xml-entity-names-20080721/

この仕様には、次のような変更が加えられています。

- 代替の単一コードポイントマッピングが使用可能な場合を除き、複数のコードポイントにマッピングされているエンティティは出力されません。このようなエンティティのほとんどは、否定算術記号（例：`isoamsa` の `nrarrw`）です。
- `gcedil` セットも含まれます（これは仕様に含まれていません）。

次の表は、さまざまな SGML 文字マッピング設定を示しています。

SGML 文字マッピング設定	説明
none	デフォルト。出力で SGML エンティティマッピングが実行されません。
normal	出力で Unicode コードポイントを SGML エンティティに変換します。変換は、デフォルトの順序で実行されます。normal 設定と math および pub 設定の唯一の違いは、エンティティをマッピングするために選択される順序です。これは、エンティティのマッピングで、複数のエンティティが特定の 1 つのコードポイントにマッピングされている場合にのみ影響します。
math	出力で Unicode コードポイントを SGML エンティティに変換します。変換は、数学関連エンティティを優先する順序で実行されます。math 設定と normal および pub 設定の唯一の違いは、エンティティをマッピングするために選択される順序です。これは、エンティティのマッピングで、複数のエンティティが特定の 1 つのコードポイントにマッピングされている場合にのみ影響します。
pub	出力で Unicode コードポイントを SGML エンティティに変換します。変換は、出版社で一般的に使用されるエンティティを優先する順序で実行されます。pub 設定と normal および math 設定の唯一の違いは、エンティティをマッピングするために選択される順序です。これは、エンティティのマッピングで、複数のエンティティが特定の 1 つのコードポイントにマッピングされている場合にのみ影響します。

注： 一般に、xdmp:document-load の <repair>full</repair> オプション、および xdmp:unquote の "repair-full" オプションには、[Output SGML Character Entites] 設定とは逆の効果があります。これは、読み込み API が SGML エンティティを、それに相当するコードポイント（1 つ以上のコードポイント）にマッピングするためです。出力オプションの相違点は、出力オプションは単一のコードポイントとエンティティのマッピングを実行するが、複数のコードポイントとエンティティのマッピングは実行しないことです。

15.3.2　アプリケーションサーバー設定での SGML マッピング設定

アプリケーションサーバーの SGML 出力マッピングを設定するには、次の手順を実行します。

1. 管理画面で、設定するアプリケーションサーバーに移動します（例：［Groups］> ［Default］> ［App Servers］> ［MyAppServer］）。

2. 左側のツリーメニューから［Output Options］ページを選択します。［Output Options Configuration］ページが表示されます。

3. ［Output SGML Entity Characters］ドロップダウンリストを見つけます（ページ上部にあります）。

4. 必要な設定を選択します。設定については、前のセクションの表を参照してください。

5. [OK] ボタンをクリックします。

これで、このアプリケーションサーバーに対するリクエストについて SGML エンティティにマッピングされるコードポイントは、デフォルトでエンティティとしてシリアライズされます。

15.3.3　XQuery プログラムにおける SGML マッピングの指定

XQuery プログラムの XML 出力での SGML マッピングを指定するには、次の XML シリアライズ API で `<output-sgml-character-entities>` オプションを使用します。

- `xdmp:quote`
- `xdmp:save`

これらの関数の詳細については、『*MarkLogic XQuery and XSLT Function Reference*』を参照してください。

15.4　出力のエンコーディングの指定

デフォルトでは、MarkLogic サーバーは UTF-8 でコンテンツを出力します。アプリケーションサーバー単位とクエリ単位の両方で、異なる出力のエンコーディングを指定できます。このセクションでは、これらの手法について説明します。以下のように構成されています。

- アプリケーションサーバーの出力のエンコーディング設定
- 出力のエンコーディングを指定するための XQuery ビルトイン

15.4.1　アプリケーションサーバーの出力のエンコーディング設定

管理画面または Admin API を使用して、アプリケーションサーバーの出力のエンコーディングを設定できます。この設定は、サポートされる任意の文字セットに設定できます（『*Search Developer's Guide*』の章 Encodings and Collations の Collations and Character Sets By Language を参照）。

管理画面を使用してアプリケーションサーバーの出力のエンコーディングを設定するには、次の手順を実行します。

1. 管理画面で、設定するアプリケーションサーバーに移動します（例：[Groups] > [Default] > [App Servers] > [MyAppServer]）。
2. 左側のツリーメニューから［Output Options］ページを選択します。［Output Options Configuration］ページが表示されます。
3. ［Output Encoding］ドロップダウンリストを見つけます（ページ上部にあります）。
4. 必要なエンコーディングを選択します。設定は、さまざまな言語に対応しています（『*Search Developer's Guide*』の章 Encodings and Collations で Collations and Character Sets By Language を参照）。
5. [OK] ボタンをクリックします。

これで、このアプリケーションサーバーに対するクエリがデフォルトで、指定されたエンコーディングで出力されます。

15.4.2　出力のエンコーディングを指定するための XQuery ビルトイン

リクエスト単位で出力のエンコーディングを取得および設定するには、次のビルトイン関数を使用します。

- `xdmp:get-response-encoding`
- `xdmp:set-response-encoding`

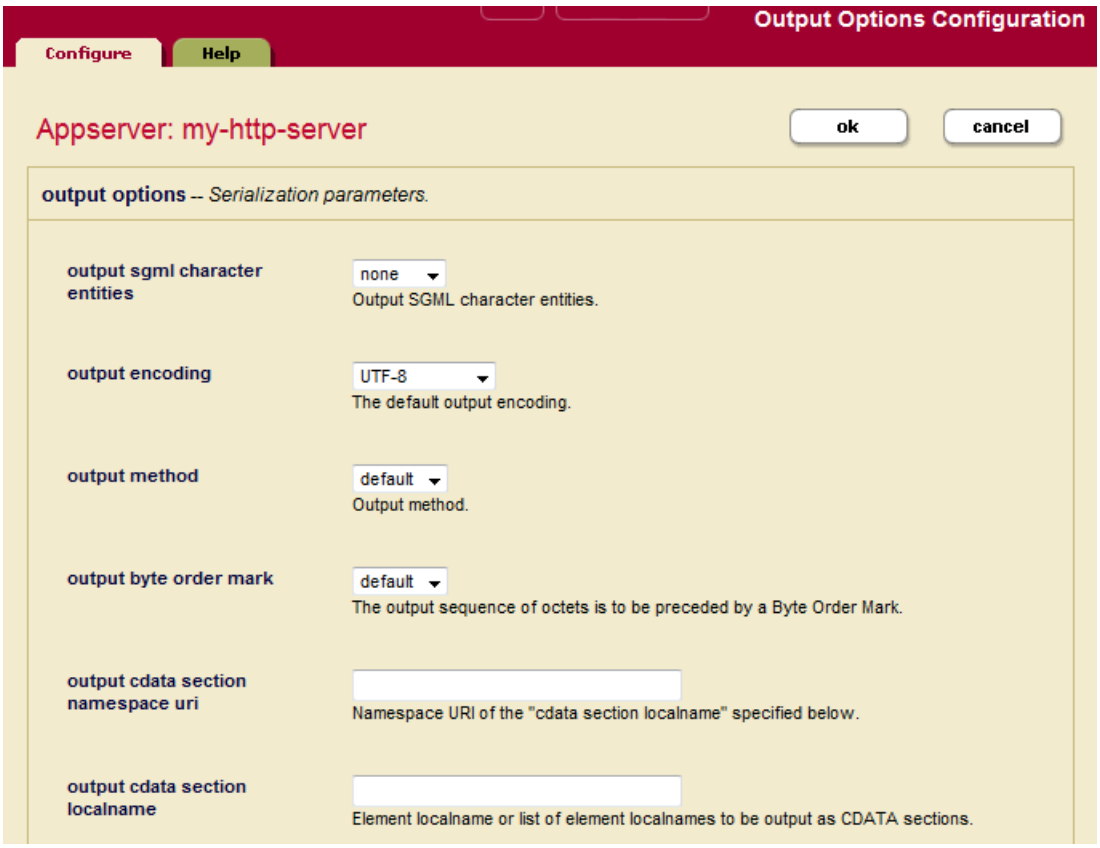
また、XQuery プログラムの XML 出力における出力のエンコーディングを指定するには、次の XML シリアライズ API で `<output-encoding>` オプションを使用します。

- `xdmp:quote`
- `xdmp:save`

これらの関数の詳細については、『*MarkLogic XQuery and XSLT Function Reference*』を参照してください。

15.5　アプリケーションサーバーレベルで出力オプションの指定

管理画面を使用して、一連の出力オプションにデフォルト値を指定できます。それぞれのアプリケーションサーバーに［Output Options Configuration］ページがあります。



この設定ページを使用すると、XSLT 出力オプション（http://www.w3.org/TR/xslt20#serialization）と一部の MarkLogic 固有オプションに対応するデフォルト値を指定できます。これらのオプションの詳細については、『*XQuery and XSLT Reference Guide*』の xdmp:output を参照してください。アプリケーションサーバーのデフォルトオプション設定の詳細については、『*Administrator's Guide*』の Setting Output Options for an HTTP Server を参照してください。

16.0 インタプリタ型 XQuery リライタの作成による REST Web サービスのサポート

REST ライブラリを使うと、アプリケーションで使用されている言語に関係なく、RESTful な機能を作成できます。

> **注：** この章に記載された手順は、「シェークスピアの XML コンテンツの読み込み」（194 ページ）で説明されている方法でシェークスピアの XML コンテンツをすでに読み込んでおり、「シンプルな URL リライタ」（195 ページ）で説明されている方法で「bill」アプリケーションサーバーとディレクトリを構成していることを前提としています。

このセクションで説明するトピックは次のとおりです。

- この章で使用する用語
- REST ライブラリの概要
- シンプルな XQuery リライタとエンドポイント
- リライタのマッチ基準に関する注意
- options ノード
- options ノード要素の検証
- URL からの複数のコンポーネントの抽出
- エラーの処理
- リダイレクトの処理
- HTTP 動詞の処理
- パラメータの定義
- 条件の追加

16.1 この章で使用する用語

- 「*REST*」は「*Representational State Transfer*」の略であり、MarkLogic サーバーのモニタリングのコンテキストでは、モニタリングを行うアプリケーションとモニターホストの間で呼び出しを行うための HTTP の使用法を記述するアーキテクチャスタイルです。
- 「リライタ」は、着信リクエストの URL を解釈し、要求にサービスを提供する内部的な URL に書き換えます。リライタは、XQuery モジュール（この章を参照）または XML ファイル（「宣言型 XML リライタの作成による REST Web サービスのサポート」（233 ページ）を参照）として実装できます。
- 「エンドポイント」は MarkLogic サーバーの XQuery モジュールであり、HTTP リクエストによって呼び出され、HTTP リクエストに応答します。

16.2 REST ライブラリの概要

REST ライブラリは一連の XQuery 関数から構成されており、URL の書き換え（リライト）とエンドポイントの検証などを行います。また MarkLogic の REST ボキャブラリは、web サービスのエンドポイントの記述作業をシンプルにします。この REST ボキャブラリは、エンドポイントの宣言的な記述に使われます。この説明には、URL の一部のパラメータへのマッピングや、入ってくるリクエストをエンドポイントにマッピングするのに必要な条件などが含まれます。

この REST ライブラリに含まれる関数によって、以下のことがシンプルになります。

- URL リライタを作成し、入ってくるリクエストをエンドポイントにマッピングする。
- リソースをリクエストしているアプリケーションが、必要なアクセス権限を持っていることを検証する。
- 入ってくるリクエストをエンドポイントが処理することを検証する。
- エラーの報告。

REST ボキャブラリでは、リライタとエンドポイントに同じ記述を使えます。

MarkLogic サーバーのリソースへの RESTful なアクセスを有効にした場合、アプリケーションは、対象となる MarkLogic サーバーのホスト上のエンドポイントモジュールを呼び出す URL を使用して、これらのリソースにアクセスします。

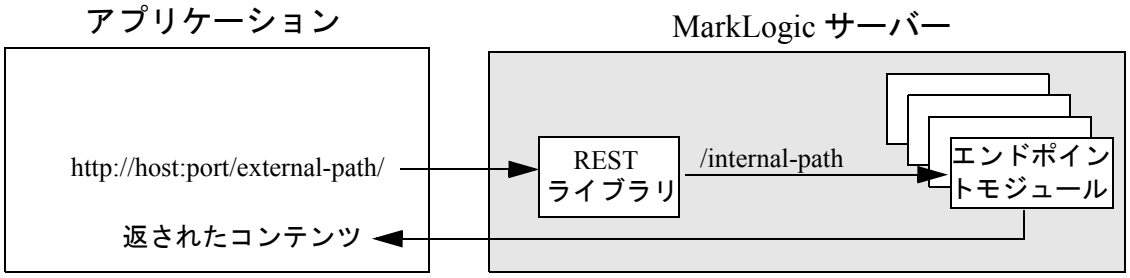
この REST ライブラリでは、以下のことを行います。

1. 入ってくる HTTP リクエストを検証する。

2. ユーザーを認証する。

3. エンドポイントを呼び出す前に、内部的にサーバーが理解できるようにリソースパスを書き換える。

リクエストが有効な場合、エンドポイントモジュールは要求された操作を実行し、アプリケーションにデータを返します。それ以外の場合は、エンドポイントモジュールはエラーメッセージを返します。



注： REST ライブラリの API シグネチャは、『*MarkLogic XQuery and XSLT Function Reference*』に記載されています。URL 書き換えの詳細については、「HTTP アプリケーションサーバーの URL 書き換えの設定」（193 ページ）を参照してください。

16.3 シンプルな XQuery リライタとエンドポイント

このセクションでは、1 つのエンドポイントを呼び出すシンプルなリライタについて説明します。

/<MarkLogic_Root>/bill ディレクトリに移動し、以下の内容のファイルを作成してください。

以下の内容を含む、requests.xqy という名前のモジュールを作ります。

```
xquery version "1.0-ml";

module namespace
    requests="http://marklogic.com/appservices/requests";

import module namespace rest = "http://marklogic.com/
appservices/rest"
    at "/MarkLogic/appservices/utils/rest.xqy";

declare variable $requests:options as element(rest:options)
:=
  <options xmlns="http://marklogic.com/appservices/rest">
    <request uri="^/(.+)$" endpoint="/endpoint.xqy">
      <uri-param name="play">$1.xml</uri-param>
    </request>
  </options>;
```

以下の内容を含む、url_rewriter.xqy という名前のモジュールを作成します。

```
xquery version "1.0-ml";

import module namespace rest = "http://marklogic.com/
appservices/rest"
     at "/MarkLogic/appservices/utils/rest.xqy";

import module namespace requests =   "http://marklogic.com/
appservices/requests"
     at "requests.xqy";

rest:rewrite($requests:options)
```

以下の内容を含む、endpoint.xqy という名前のモジュールを作成します。

```
xquery version "1.0-ml";

import module namespace rest = "http://marklogic.com/
appservices/rest"
    at "/MarkLogic/appservices/utils/rest.xqy";

import module namespace requests =
  "http://marklogic.com/appservices/requests"
  at "requests.xqy";

let $request := $requests:options/rest:request
                  [@endpoint = "/endpoint.xqy"][1]

let $map  := rest:process-request($request)
let $play := map:get($map, "play")

return
  fn:doc($play)
```

次の URL を入力します。これは、「シンプルな URL リライタ」（195 ページ）内に作成された、bill というアプリケーションサーバーを使用しています。

```
http://localhost:8060/macbeth
```

リライタ内の rest:rewrite 関数は、options ノードを使用して着信リクエストをエンドポイントにマッピングします。options ノードには、request 要素が含まれます。この要素の uri 属性は正規表現を指定し、endpoint 属性は、着信リクエストの URL が正規表現にマッチした場合に呼び出すエンドポイントモジュールを指定します。マッチした場合は、(.+) にマッチする URL の一部分が $1 変数にバインドされます。request 要素内にある uri-param 要素は、$1 変数の値と .xml 拡張子を play パラメータに割り当てます。

```
<rest:options xmlns="http://marklogic.com/
appservices/rest">
  <rest:request uri="^/(.+)" endpoint="/endpoint.xqy">
    <uri-param name="play">$1.xml</uri-param>
  </rest:request>
</rest:options>
```

上述のリライタモジュールの例では、この options ノードが rest:rewrite 関数に渡され、パラメータ play=macbeth.xml を持つエンドポイントモジュールを呼び出す URL が出力されます。

```
/endpoint.xqy?play=macbeth.xml
```

エンドポイント内の rest:process-request 関数は、endpoint.xqy モジュールに関連付けられた最初の request 要素を見つけ、これを使用して、リライタが定義するパラメータを特定します。この例では、play という 1 つのパラメータのみですが、「URL からの複数のコンポーネントの抽出」（216 ページ）に記載された理由により、同一エンドポイントに request 要素が複数ある場合は、URL から最大個数のパラメータを抽出する request 要素が、options ノード内のリストにおいて、より少数のパラメータをを抽出するものよりも先に配置されている必要があります。

```
let $request := $requests:options/rest:request
                  [@endpoint = "/endpoint.xqy"][1]
```

エンドポイント内の rest:process-request 関数は、request 要素を使用して着信リクエストをパースし、すべてのパラメータを型付き値として含むマップを返します。map:get 関数は、このマップから各パラメータ（この例では 1 つのみ）を抽出します。

```
let $map  := rest:process-request($request)
let $play := map:get($map, "play")
```

16.4　リライタのマッチ基準に関する注意

このリライタはデフォルトでは、リクエストをすべての基準にマッチさせます。基準としては、URI、Accept ヘッダ、コンテンツタイプ（content-type）、条件、メソッド、パラメータがあります。ここで前提となるのは、期待されているものと完全にマッチしない限り、エンドポイントが呼び出されないということです。このため、結果がちょっとわかりにくい場合があります。例えば、以下のようなリクエストノードがあるとします。

```
<request uri="/path/to/resource" endpoint="/endpoint.xqy">
  <param name="limit" as="decimal"/>
</request>
```

フォームから入ってきたリクエストである

```
/path/to/resource?limit=test
```

はリクエストノードとマッチしません（「limit」が「decimal」ではないため）。マッチするリクエストノードが存在しない場合、このリクエストには「404 (not found)」が返されます。

これは不思議な感じがするかもしれません。他のリクエストノードを追加してマッチをもっと緩くすることで、この問題を解決できます。しかしながら、リクエストの数と複雑さが増すにつれて、このやり方はあまりうまくいかなくなります。その代わりとして、リクエストの特定部分にだけマッチするようにリライタを設定できます。こうすることで、呼び出されたモジュールでエラーを処理できます。

マッチ基準は、rest:rewrite への呼び出しのなかで指定されます。例：

```
rest:rewrite($options, ("uri", "method"))
```

この場合、URI と HTTP メソッドだけがマッチに使われています。

利用できる基準には、以下のものがあります。

- uri = URI でマッチ
- accept = Accept ヘッダでマッチ
- content-type = コンテンツタイプでマッチ
- conditions = 条件でマッチ
- method = HTTP メソッドでマッチ
- params = パラメータでマッチ

リクエストがマッチしていると認識されるためには、すべての基準とマッチする必要があります。

16.5 options ノード

REST ライブラリは、options ノードを使用して着信リクエストをエンドポイントにマッピングします。この options ノードは element(rest:options) 型として宣言する必要があり、http://marklogic.com/appservices/rest 名前空間内にある必要があります。

以下は、とてもシンプルな options ノードの宣言の例です。

```
declare variable $options as element(rest:options) :=
  <rest:options xmlns="http://marklogic.com/
  appservices/rest">
    <rest:request uri="^/(.+)" endpoint="/endpoint.xqy">
      <uri-param name="play">$1.xml</uri-param>
    </rest:request>
  </rest:options>;
```

以下のテーブルには、options ノードで使用できる要素と属性がすべて示されています。一番左は要素で、これは属性や子要素を持ちます。各要素の属性は「属性」列に表示されています。属性はオプションです（「required」とある場合は必須です）。一番左に表示されている要素の、第一レベル要素が「子要素」列に表示されています。user-params="allow" と user-params="allow-dups" の属性の値の違いは、allow では、指定された名前に対して 1 つのパラメータを使用できるのに対し、allow-dups では複数を使用できることです。

要素	属性	子要素	子要素の個数	詳細
options	user-params = ignore | allow | allow-dups | forbid	request	0..n	「シンプルな XQuery リライタとエンドポイント」（209 ページ）
request	uri=*string* endpoint=*string* user-params = ignore | allow | allow-dups | forbid	uri-param param http auth function accept user-agent and or	0..n 0..n 0..n 0..n 0..n 0..n 0..n 0..n 0..n	「シンプルな XQuery リライタとエンドポイント」（209 ページ） 「URL からの複数のコンポーネントの抽出」（216 ページ） 「条件の追加」（229 ページ）
uri-param	name=*string* (required) as=*string*			「URL からの複数のコンポーネントの抽出」（216 ページ） 「パラメータの定義」（225 ページ）

要素	属性	子要素	子要素の個数	詳細
param	name=*string* (required) as=*string* values=*string* match=*string* default=*string* required = true | false repeatable = true | false pattern = <regex>			「パラメータの定義」（225 ページ） 「URL 内で指定されたパラメータのサポート」（226 ページ）. 「パラメータ内の正規表現と match/pattern 属性とのマッチング」（228 ページ）
http	method = *string* (required) user-params = ignore | allow | allow-dups | forbid	param auth function accept user-agent and or	0..n 0..n 0..n 0..n 0..n 0..n 0..n	「HTTP 動詞の処理」（220 ページ） 「条件の追加」（229 ページ）.
auth		privilege kind	0..n 0..n	「認証条件」（230 ページ）
function	ns=*string* (required) apply=*string* (required) at=*string* (required)			「関数条件」（231 ページ）

16.6 options ノード要素の検証

rest:check-options 関数を使用して、REST スキーマを基準にして options ノードを検証できます。例えば、「シンプルな XQuery リライタとエンドポイント」（209 ページ）で説明されているように、requests.xqy モジュールで定義された options ノードを検証するには、以下のようにします。

```
xquery version "1.0-ml";

import module namespace rest = "http://marklogic.com/
appservices/rest" at "/MarkLogic/appservices/utils/
rest.xqy";

import module namespace requests =
  "http://marklogic.com/appservices/requests"
  at "requests.xqy";

rest:check-options($requests:options)
```

options ノードが有効である場合は、空のシーケンスが返されます。それ以外の場合は、エラーが返されます。

また、rest:check-request 関数を使用して、options ノード内の request 要素を検証できます。例えば、「シンプルな XQuery リライタとエンドポイント」（209 ページ）で説明されているような requests.xqy モジュールで定義された options ノード内のすべての request を検証するには、以下のようにします。

```
xquery version "1.0-ml";

import module namespace rest = "http://marklogic.com/
    appservices/rest"  at "/MarkLogic/appservices/
   utils/rest.xqy";

declare option xdmp:mapping "false";

rest:check-request($requests:options/rest:request)
```

request ノードが有効である場合は、空のシーケンスが返されます。それ以外の場合は、エラーが返されます。

> **注：** rest:check-request 関数を呼び出す前に、xdmp:mapping を false に設定して、関数のマッピングを無効にする必要があります。

16.7 URL からの複数のコンポーネントの抽出

options ノードには 1 つあるいは複数の request 要素が含まれ、それぞれに、リクエスト URL の一部をパラメータに割り当てる uri-param 要素が 1 つあるいは複数含まれます。各 request 要素の目的は、特定の URL パターンを検知し、1 つあるいは複数のパラメータでエンドポイントを呼び出すことです。URL から複数のコンポーネントを抽出することは、単に、特定の URL パターンを認識する正規表現で request 要素を定義した後に、必要な URL 部分を変数にバインドすることです。

例えば、「シンプルな XQuery リライタとエンドポイント」（209 ページ）で説明されているリライタの機能を拡張し、以下のような URL を使ってシェークスピアの戯曲内の「幕（act）」を表示する機能を追加するとします。

```
http://localhost:8060/macbeth/act3
```

requests.xqy 内の options ノードは、以下のようになります。ここには request 要素が 2 つ含まれています。リライタは「ファーストマッチ」ルールを使用します。つまり、並べられている順に着信 URL を request 要素にマッチさせ、この URL にマッチする正規表現を含む最初のものを選択します。以下の例では、URL で「幕（act）」が指定されている場合に、リライタが最初の request を使用します。戯曲だけが URL で指定されている場合、最初の request 要素にはマッチするものがありませんが、2 つめの request 要素にはあります。

注： パラメータの型はデフォルトでは文字列になっています。文字列以外のパラメータは、以下の act パラメータのように、明示的に型付け（タイピング）する必要があります。型付けパラメータの詳細については、「パラメータの型」（225 ページ）を参照してください。

```
<options>
  <request uri="^/(.+)/act(\d+)$" endpoint="/endpoint.xqy">
    <uri-param name="play">$1.xml</uri-param>
    <uri-param name="act" as="integer">$2</uri-param>
  </request>
  <request uri="^/(.+)/?$" endpoint="/endpoint.xqy">
    <uri-param name="play">$1.xml</uri-param>
  </request>
</options>
```

着信 URL で「幕」が指定されている場合、最初の request は、macbeth と 3 をそれぞれ、変数 $1 と $2 にバインドします。その後、play および act という名前のパラメータに割り当てます。rest:rewrite 関数によって書き換えられた URL は、以下のようになります。

```
/endpoint.xqy?play=macbeth.xml&act=3
```

以下は、前述の options ノードを使用するリライタで呼び出すことのできるエンドポイントモジュールの例です。「シンプルな XQuery リライタとエンドポイント」（209 ページ）で説明するように、エンドポイント内の rest:process-request 関数は、request 要素を使用して着信リクエストをパースし、すべてのパラメータを型付き値として含むマップを返します。その後、各パラメータは map:get 関数によってマップから抽出されます。このエンドポイントを呼び出す URL に act パラメータが含まれていない場合、$num 変数の値は空の文字列になります。

注： この例では、endpoint.xqy モジュールを呼び出す最初の request 要素が使われています。これは、ファーストマッチルールに基づいて、play および act パラメータの両方をサポートする要素であるためです。

```
xquery version "1.0-ml";

import module namespace rest = "http://marklogic.com/
appservices/rest"
    at "/MarkLogic/appservices/utils/rest.xqy";

import module namespace requests =
  "http://marklogic.com/appservices/requests"
  at "requests.xqy";

  let $request := $requests:options/rest:request
                    [@endpoint = "/endpoint.xqy"][1]

  let $map  := rest:process-request($request)
  let $play := map:get($map, "play")
  let $num  := map:get($map, "act")

return
  if (empty($num))
  then
    fn:doc($play)
  else
    fn:doc($play)/PLAY/ACT[$num]
```

16.8 エラーの処理

REST エンドポイントライブラリには、MarkLogic サーバーのエラーのマークアップから HTML への単純な変換を実行する `rest:report-error` 関数が含まれています。モジュール内でこれを呼び出し、エラーの報告に使用できます。

```
try {
  let $params := rest:process-request($request)
  return
    ...the non-error case...
} catch ($e) {
  rest:report-error($e)
}
```

リクエストしているユーザーエージェントがテキスト /HTML をアクセプトする場合、シンプルな HTML 形式の反応が返されます。それ以外の場合は、エラーの XML がそのまま返されます。

またこの関数をエラーハンドラ内で使用し、特定のアプリケーションにおけるエラーをすべて処理することもできます。

16.9 リダイレクトの処理

この章の前のセクションで示したように、URL リライタは要求された URL を新しい URL にトランスレートして、サーバー内でディスパッチできるようにします。リクエストしているユーザーエージェントはこのトランスレートにまったく気づくことはありません。REST ライブラリが成熟し拡大してきた場合、リクエストに対応するために、リダイレクトを行い、リクエストを新しい URL で再発行するようにユーザーエージェントに指示したほうがいいでしょう。

例えば、このユーザーは以下の URL パターンを利用して戯曲「マクベス」にアクセスしていました。

```
http://localhost:8060/Shakespeare/macbeth
```

これを以下の URL にリダイレクトしたいとします。

```
http://localhost:8060/macbeth
```

ユーザーはこのリダイレクトが起こったことがわかります。というのも、アドレスバーの内容が古い URL から新しい URL に変わるからです。ユーザーはこの新しい URL をブックマークできます。

このようなリダイレクトは、以下のような redirect.xqy をアプリケーションに追加することで実現できます。

```
xquery version "1.0-ml";

import module namespace rest="http://marklogic.com/
appservices/rest"
     at "/MarkLogic/appservices/utils/rest.xqy";

import module namespace requests =
  "http://marklogic.com/appservices/requests"
  at "requests.xqy";

(: Process requests to be handled by this endpoint module.:)
let $request := $requests:options/rest:request
                    [@endpoint = "/redirect.xqy"][1]

let $params  := rest:process-request($request)

(: Get parameter/value map from request.:)
let $query  := fn:string-join(
    for $param in map:keys($params)
    where $param != "__ml_redirect__"
    return
      for $value in map:get($params, $param)
      return
        fn:concat($param, "=", fn:string($value)),
  "&amp;")

(: Return the name of the play along with any parameters.:)
let $ruri   := fn:concat(map:get($params, "__ml_redirect__"),
                      if ($query = "") then ""
                      else fn:concat("?", $query))

(: Set response code and redirect to new URL.:)
return
  (xdmp:set-response-code(301, "Moved permanently"),
   xdmp:redirect-response($ruri))
```

requests.xqy 内の options ノードに、以下のリクエスト要素を追加してリダイレクトを実行します。

```
<request uri="^/shakespeare/(.+)/(.+)"
endpoint="/redirect.xqy">
  <uri-param name="__ml_redirect__">/$1/$2</uri-param>
</request>
```

```
<request uri="^/shakespeare/(.+)"
endpoint="/redirect.xqy">
  <uri-param name="__ml_redirect__">/$1</uri-param>
</request>
```

これで、options ノードは以下のようになります。redirect.xqy モジュール用の request 要素は、endpoint.xqy モジュール用の要素の前にあります。これは、「URL からの複数のコンポーネントの抽出」（216 ページ）で説明する「ファーストマッチ」ルールによるものです。

```
<options xmlns="http://marklogic.com/appservices/rest">
  <request uri="^/shakespeare/(.+)/(.+)"
  endpoint="/redirect.xqy">
    <uri-param name="__ml_redirect__">/$1/$2</uri-param>
  </request>
  <request uri="^/shakespeare/(.+)" endpoint="/redirect.xqy">
    <uri-param name="__ml_redirect__">/$1</uri-param>
  </request>
  <request uri="^/(.+)/act(\d+)" endpoint="/endpoint.xqy">
    <uri-param name="play">$1.xml</uri-param>
    <uri-param name="act" as="integer">$2</uri-param>
  </request>
  <request uri="^/(.+)$" endpoint="/endpoint.xqy">
    <uri-param name="play">$1.xml</uri-param>
  </request>
</options>
```

これと同じ redirect.xqy モジュールで、__ml_redirect パラメータの値を変更することで、必要な数だけリダイレクトを使用できます。

16.10 HTTP 動詞の処理

動詞を指定していないリクエストは、HTTP GET リクエストとのみマッチされます。他の動詞をマッチさせる場合は、request 要素内の method 属性で http 要素を使用して、それらの動詞をリストに含めます。

```
<request uri="^/(.+?)/?$" endpoint="/endpoint.xqy">
  <uri-param name="play">$1.xml</uri-param>
  <http method="GET"/>
  <http method="POST"/>
</request>
```

このリクエストは、リクエストメソッドが HTTP GET か HTTP POST の場合にマッチ（ならびに検証）されます。

以下のトピックでは、マッピングのリクエスト（動詞と単純なエンドポイントによる）のユースケースを紹介しています。

- OPTIONS リクエストの処理
- POST リクエストの処理

16.10.1 OPTIONS リクエストの処理

options ノード、または options ノード内の特定の request 要素を返す仕組みがあると便利です。例えば、特定の URL にマッチする request 要素を見つけるユニットテストの一部を自動化できます。このような機能は、OPTIONS メソッドをサポートすることで実現できます。

以下は、OPTIONS メソッドを指定するリクエストを処理するシンプルな options.xqy モジュールの例です。リクエスト URL が / の場合、options.xqy モジュールは options 要素全体を返し、エンドポイントの集まり全体が公開されます。リクエスト URL が / ではない場合、このモジュールは、この URL にマッチする request 要素を返します。

```
xquery version "1.0-ml";

import module namespace rest="http://marklogic.com/
appservices/rest"
  at "/MarkLogic/appservices/utils/rest.xqy";

import module namespace requests =
  "http://marklogic.com/appservices/requests"
  at "requests.xqy";

(: Process requests to be handled by this endpoint module.:)
let $request := $requests:options/rest:request
                  [@endpoint = "/options.xqy"][1]

(: Get parameter/value map from request.:)
let $params  := rest:process-request($request)
let $uri     := map:get($params, "__ml_options__")
let $accept  := xdmp:get-request-header("Accept")
let $params  := map:map()

(: Get request element that matches the specified URL.:)
let $request := rest:matching-request($requests:options,
                                      $uri,
                                      "GET",
                                      $accept,
                                      $params)
```

```
(: If URL is '/', return options node.Otherwise, return
   request element that matches the specified URL.:)
return
  if ($uri = "/")
  then
    $requests:options
  else
    $request
```

以下の request 要素を requests.xqy に追加して、OPTIONS メソッドを含むあらゆる HTTP リクエストにマッチさせます。

```
<request uri="^(.+)$" endpoint="/options.xqy"
user-params="allow">
  <uri-param name="__ml_options__">$1</uri-param>
  <http method="OPTIONS"/>
</request>
```

Query Console を開き、以下のクエリを入力します。「name」と「password」は、自分のログイン情報に置き換えてください。

```
xdmp:http-options("http://localhost:8011/",
<options xmlns="xdmp:http">
  <authentication method="digest">
    <username>name</username>
    <password>password</password>
  </authentication>
</options>)
```

リクエスト URL が / であるため、options ノード全体が返されます。他の URL を使った場合の結果を見るためには、Query Console に以下のクエリを入力してください。

```
xdmp:http-options("http://localhost:8011/shakespeare/
macbeth",
<options xmlns="xdmp:http">
  <authentication method="digest">
    <username>name</username>
    <password>password</password>
  </authentication>
</options>)
```

options ノード全体を返すのではなく、指定された URL にマッチする request 要素が返されます。

```
<request uri="^/shakespeare/(.+)" endpoint="/redirect.xqy"
  xmlns="http://marklogic.com/appservices/rest">
    <uri-param name="__ml_redirect__">/$1</uri-param>
</request>
```

以下のリクエストをオプションの最後に追加することで、これを使えます。

```
<request uri="^(.+)$" endpoint="/options.xqy"
user-params="allow">
  <uri-param name="__ml_options__">$1</uri-param>
  <http method="OPTIONS"/>
</request>
```

以前のリクエストの中に、直接 OPTIONS をサポートしているものがある場合、このリソースに関しては、それが優先されます。

16.10.2 POST リクエストの処理

POST メソッドをサポートして、RESTful なコンテンツマネジメント機能（データベースへのコンテンツの読み込みなど）を実装することが必要な場合もあります。

以下に示すのは、シンプルな post.xqy モジュールの例です。この例は、POST メソッドを含むリクエストを受け入れ、このリクエストで指定された URL のデータベースにリクエストのボディを挿入します。

```
xquery version "1.0-ml";

import module namespace rest="http://marklogic.com/
appservices/rest"
       at "/MarkLogic/appservices/utils/rest.xqy";

import module namespace requests =
  "http://marklogic.com/appservices/requests"
  at "requests.xqy";

(: Process requests to be handled by this endpoint module.:)
let $request := $requests:options/rest:request
                  [@endpoint = "/post.xqy"][1]

(: Get parameter/value map from request.:)
let $params  := rest:process-request($request)
let $posturi := map:get($params, "_ml_post_")
let $type    := xdmp:get-request-header('Content-Type')

(: Obtain the format of the content.:)
let $format :=
```

```
  if ($type = 'application/xml' or ends-with($type,
'+xml'))
  then
    "xml"
  else
    if (contains($type, "text/"))
    then "text"
    else "binary"

(: Insert the content of the request body into the
database.:)
let $body := xdmp:get-request-body($format)

return
  (xdmp:document-insert($posturi, $body),
  concat("Successfully uploaded: ", $posturi, "&#10;"))
```

以下のリクエスト要素を `requests.xqy` に追加します。リクエスト URL が `/post/`*`filename`* の場合、リライタは、`POST` メソッドを含む HTTP リクエストを `post.xqy` モジュールに発行します。

```
<request uri="^/post/(.+)$" endpoint="/post.xqy">
  <uri-param name="_ml_post_">$1</uri-param>
  <http method="POST"/>
</request>
```

`post.xqy` エンドポイントをテストするには、Query Console を開き、以下のクエリを入力します。「name」と「password」を、MarkLogic サーバーへのログイン認証情報に置き換えてください。

```
let $document:= xdmp:quote(
  <html>
    <title>My Document</title>
    <body>
      This is my document.
    </body>
  </html>)

return
  xdmp:http-post("http://localhost:8011/post/mydoc.xml",
    <options xmlns="xdmp:http">
      <authentication method="digest">
        <username>name</username>
        <password>password</password>
      </authentication>
```

```
    <data>{$document}</data>
    <headers>
      <content-type>text/xml</content-type>
    </headers>
  </options>)
```

Query Console の［Explore］ボタンをクリックし、Documents データベース内の /mydoc.xml を見つけます。

16.11 パラメータの定義

このセクションでは、request 要素内にある uri-param および param 要素を詳細に説明します。このセクションで説明するトピックは次のとおりです。

- パラメータの型
- URL 内で指定されたパラメータのサポート
- 必須パラメータ
- パラメータのデフォルト値
- 値のリストの指定
- リピータブルパラメータ
- パラメータキーのエイリアス
- パラメータ内の正規表現と match/pattern 属性とのマッチング

16.11.1　パラメータの型

デフォルトでは、パラメータは文字列型になっています。パラメータの他の型（整数やブール型など）の場合は、リクエスト要素において明示的に型を指定する必要があります。「URL からの複数のコンポーネントの抽出」（216 ページ）の request 要素の例を使用して、act パラメータを整数として明示的に定義する必要があります。

```
<request uri="^/(.+)/act(\d+)$" endpoint="/
endpoint.xqy">
  <uri-param name="play">$1.xml</uri-param>
  <uri-param name="act" as="integer">$2</uri-param>
</request>
```

パラメータの型としては、XQuery でサポートしているものをすべて利用できます。これについては仕様書である『XML Schema Part 2: Datatypes Second Edition』を参照してください。

http://www.w3.org/TR/xmlschema-2/

16.11.2 URL 内で指定されたパラメータのサポート

REST ライブラリでは、URL パスの後に入力したパラメータをサポートしています。以下のようなフォーマットになります。

```
http://host:port/url-path?param=value
```

例えば、「scene」パラメータをサポートする endpoint.xqy モジュールが必要であるとします。このモジュールを使用すると、以下の URL を入力することで「Macbeth, Act 4, Scene 2（マクベス、第 4 幕、第 2 場）」を返すことができます。

```
http://localhost:8011/macbeth/act4?scene=2
```

scene パラメータをサポートするには、以下のように、エンドポイントモジュールの最初の request 要素を変更します。param 要素の match 属性は部分式を定義しているため、パラメータ値は $1 変数に割り当てられます。この変数は、uri_param 要素が使用する $1 変数とは別のものです。

```
<request uri="^/(.+)/act(\d+)$" endpoint="/endpoint.xqy">
  <uri-param name="play">$1.xml</uri-param>
  <uri-param name="act" as="integer">$2</uri-param>
  <param name="scene" as="integer" match="(.+)">$1</param>
</request>
```

endpoint.xqy モジュールを以下のように書き換えて、scene（「場」）パラメータのサポートを追加します。

```
xquery version "1.0-ml";

import module namespace rest = "http://marklogic.com/
appservices/rest"
    at "/MarkLogic/appservices/utils/rest.xqy";

import module namespace requests =
  "http://marklogic.com/appservices/requests"
  at "requests.xqy";

let $request := $requests:options/rest:request
                  [@endpoint = "/requests.xqy"][1]

let $map  := rest:process-request($request)
let $play := map:get($map, "play")
let $num  := map:get($map, "act")
let $scene := map:get($map, "scene")
```

```
return
    if (empty($num))
    then
      fn:doc($play)
    else if (empty($scene))
    then
      fn:doc($play)/PLAY/ACT[$num]
    else
      fn:doc($play)/PLAY/ACT[$num]/SCENE[$scene]
```

これで、リライタとエンドポイントの両方で scene パラメータが認識されます。request 要素には、任意の個数のパラメータを定義できます。

16.11.3 必須パラメータ

デフォルトでは、param 要素で定義されるパラメータは省略可能です。パラメータを必須にするには、required 属性を使用します。例えば、以下のように required 属性を使用して、scene パラメータを必須にすることができます。こうすると、scene がないリクエスト URL はマッチしなくなり、scene がないエンドポイントを呼び出そうとするとエラーが発生します。

```
<param name="scene" as="integer" match="(.+)"
required="true">
  $1
</param>
```

16.11.4 パラメータのデフォルト値

パラメータのデフォルト値を指定することもできます。以下の例では、場（scene）パラメータなしで幕（act）をリクエストした場合、この幕の第 1 場が返されます。

```
<param name="scene" as="integer" match="(.+)" default="1">
  $1
</param>
```

16.11.5 値のリストの指定

scene のようなパラメータにおいて、値のリストを指定したい場合があります。例えば、第 1 場、第 2 場、第 3 場へのリクエストだけに対応したい場合は、以下のようにします。

```
<param name="scene" as="integer" values="1|2|3"
default="1"/>
```

16.11.6　リピータブルパラメータ

パラメータをリピータブルにできます。例えば、css パラメータを使って、ある戯曲に対してスタイルシートを追加したいとします。複数使いたい場合は、css パラメータを以下のように追加できます。

```
<param name="css" repeatable="true"/>
```

リライタでは、任意の個数の css パラメータを使用できます。エンドポイントでは、パラメータマップには css キーが 1 つありますが、この値はリストにすることもできます。

16.11.7　パラメータキーのエイリアス

場合によっては、URL 内の複数のキー / バリューを 1 つのキー / バリューとして解釈したいことがあります。

例えば jQuery は、キーの値が配列の場合に、キーの名前を変更します。つまり、jQuery で { "a": "b", "c": ["d", "e"] } を呼び出した場合、以下の URL が得られます。

```
http://whatever/endpoint?a=b&amp;c[]=d&amp;c[]=e
```

ここでは、以下のような `alias` 属性を使用できます。こうすると、`rest:process-request` 関数から得られるマップでは、`c=` と `c[]=` のどちらが着信 URL のパラメータで使用されている場合でも、キーの値が「c」になります。

```
<param name="c" alias="c[]" repeatable="true"/
```

16.11.8　パラメータ内の正規表現と match/pattern 属性とのマッチング

「URL 内で指定されたパラメータのサポート」（226 ページ）で紹介したように、`match` 属性を使用して、パラメータ値に対するマッチ / 置換操作を実行できます。これは、`request` 要素の `uri` 属性を使用して、URL の各部分に対して実行できる操作です。`pattern` 属性で、パラメータの名前をテストできます。このセクションでは、`match` 属性と `pattern` 属性ついて、さらに詳しく説明します。このセクションには、次の部分があります。

- match 属性
- pattern 属性

16.11.8.1　match 属性

`param` 要素の `match` 属性は、パラメータの値のテストに使用する部分式を定義します。これにより、正規表現内でキャプチャされたグループが `$1` 変数に割り当てられます。

match 属性を使用してパラメータを変換できます。例えば、インターネットのメディアタイプを含むパラメータを変換し、match 属性を使用して、この値の一部を抽出するとします。以下は、format=application/xslt+xml を format=xslt に変換します。

```
<param name="format" match="^application/(.*?)(\+xml)?$">
    $1
</param>
```

パラメータのマッチと検証を組み合わせている場合には、変換された値が検証されるので注意してください。例えば、以下のパラメータは絶対にマッチしません。

```
<param name="test" values="foo|bar" match="^(.+)$">
    baz-$1
</param>
```

この代わりに以下のようにします。

```
<param name="test" values="baz-foo|baz-bar" match="^(.+)$">
    baz-$1
</param>
```

つまり、変換後の値が検証されます。

16.11.8.2 pattern 属性

param 要素は pattern 属性をサポートします。この属性は、特定の正規表現を使用してパラメータの名前をマッチさせます。これによってパラメータ名のマッチに関して、正規表現を指定できます。例えば以下のようになります。

```
pattern='xmlns:.+'

pattern='val[0-9]+'
```

name あるいは pattern を 1 つだけ指定する必要があります。エンドポイントに渡されるパラメータの名前が複数のパターンにマッチする場合は、エラーになります。

16.12 条件の追加

条件を追加することができますが、リクエストのボディ部分に追加した場合、すべての動詞が対象となります。あるいは特定の動詞に対してだけ追加できます。例えば、以下の request 要素には、POST 動詞用の auth 条件と、GET 動詞と POST 動詞の両方用の user-agent 条件があります。

```
<request uri="^/slides/(.+?)/?$" endpoint="/slides.xqy">
    <uri-param name="play">$1.xml</uri-param>
```

```
      <http method="GET"/>
      <http method="POST">
        <auth>
          <privilege>http://example.com/privs/editor</
          privilege>
          <kind>execute</kind>
        </auth>
      </http>
      <user-agent>ELinks</user-agent>
    </request>
```

このリクエストでは、指定された実行権限を持つユーザーだけが、この URL に POST できます。権限がないユーザーが post しようとしても、このリクエストはマッチせず、次のリクエストへフォールスルーされます。こうすることで、必要な場合にフォールバックを提供できます。

リライタでは、条件にマッチしない場合、リクエストがマッチしなくなります。エンドポイントでは、条件にマッチしない場合、エラーが起こります。

このセクションで取り上げるトピックは以下のとおりです。

- 認証条件
- accept ヘッダ条件
- ユーザーエージェント条件
- 関数条件
- and 条件
- or 条件
- content-type 条件

16.12.1　認証条件

特定の権限をチェックできる auth 条件を、以下の形式で追加できます。

```
<auth>
  <privilege>privilege-uri</privilege>
  <kind>kind</kind>
</auth>
```

例えば、「POST リクエストの処理」（223 ページ）で説明されている POST リクエストの request 要素を使用すると、あらゆるユーザーがデータベースにドキュメントを読み込むことができます。この POST 機能を、Information Studio の実行権限があるユーザーのみに制限するには、このリクエスト要素に以下の auth 条件を追加します。

```
<request uri="^/post/(.+)$" endpoint="/post.xqy">
  <uri-param name="_ml_post_">$1</uri-param>
  http method="POST">
    <auth>
      <privilege>
         http://marklogic.com/xdmp/privileges/infostudio
      </privilege>
      <kind>execute</kind>
    </auth>
  </http>
</request>
```

権限としては、あらゆる実行権限あるいは URL 権限が利用できます。指定されていない場合、`kind` はデフォルトで `execute` になります。

16.12.2　accept ヘッダ条件

ユーザーエージェントが URL をリクエストする際に、アクセプトできるレスポンスを指定できます。これでメディアのタイプを指定します。アクセプトできるメディアのタイプを、accept ヘッダで指定できます。

例えば、JSON のレスポンスをアクセプトするユーザーエージェントのリクエストだけをマッチする場合には、以下のようなアクセプト条件をリクエスト内で指定します。

```
<accept>application/json</accept>
```

16.12.3　ユーザーエージェント条件

ユーザーエージェント文字列をマッチさせることもできます。以下のように、`user-agent` を指定するリクエストは、ELinks ブラウザとして特定されたユーザーエージェントのみにマッチします。

```
<user-agent>ELinks</user-agent>
```

16.12.4　関数条件

`function` 条件を使うと、任意の条件でテストできます。名前空間、ローカル名、関数のモジュールなどを指定することで、あらゆるコードを実行できます。

```
<function ns="http://example.com/module"
          apply="my-function"
          at="utils.xqy"/>
```

上述のような関数を指定したリクエストでは、指定した関数が true となるリクエストだけをマッチします。この関数には、URL 文字列と関数条件の要素が渡されます。

16.12.5 and 条件

and 条件には、条件のみを含めることができます。これが true を返すのは、それ以下の階層に含まれる条件がすべて true の場合だけです。

```
<and>
  ...conditions...
</and>
```

複数の条件がリクエストの最上位のレベルにある場合、これらの条件は and 内にあるとして扱われます。

例えば以下の条件では、ELinks ブラウザからの HTML のレスポンスをアクセプトするユーザーエージェントのリクエストだけをマッチします。

```
<and>
  <accept>text/html</accept>
  <user-agent>ELinks</user-agent>
</and>
```

注： これらの条件を特定の順番で評価することや、すべての条件が評価されるかどうかは保証されません。

16.12.6 or 条件

or 条件には、条件のみを含めることができます。これが true を返すのは、それ以下の階層に含まれる条件の少なくとも 1 つが true の場合だけです。

```
<or>
  ...conditions...
</or>
```

例えば以下の条件は、HTML あるいはプレーンテキストのレスポンスをアクセプトするユーザーエージェントのリクエストだけをマッチします。

```
<or>
  <accept>text/html</accept>
  <accept>text/plain</accept>
</or>
```

注： これらの条件を特定の順番で評価することや、すべての条件が評価されるかどうかは保証されません。

16.12.7 content-type 条件

content-type 条件は、リクエストにマッチするコンテンツタイプがある場合に true を返す条件です。content-type 条件は、条件を使用できるあらゆる場所で使用できます。

17.0 宣言型 XML リライタの作成による REST Web サービスのサポート

宣言型 XML リライタは、インタプリタ型 XQuery リライタ（「インタプリタ型 XQuery リライタの作成による REST Web サービスのサポート」（207 ページ）を参照）と同じ目的で使用できます。XML リライタには、リクエスト環境に影響するオプションが XQuery リライタよりも数多く用意されています。ただし、XML リライタは効率性を重視して設計されているため、XQuery リライタほどの表現力を備えておらず、システム関数呼び出しにもアクセスできません。その代わり、厳選されたマッチおよび評価ルールを使用できるため、数多くの一般的なケースに対応できます。

このセクションで説明するトピックは次のとおりです。

- XML リライタの概要
- XML リライタを使用するためのアプリケーションサーバー設定
- 入出力コンテキスト
- 正規表現（Regex）
- マッチルール
- システム変数
- 評価ルール
- 終了ルール
- シンプルなリライタの例

17.1 XML リライタの概要

XML リライタは、リクエスト値のマッチングおよびリクエストの環境の準備に関するルールが含まれている XML ファイルです。リクエストされたすべての更新が受け入れられた場合、リクエストの位置は、更新される環境より前です。それ以外の場合は、エラーまたは警告がログに記録されます。XQuery リライタは、リクエスト URI（パスおよびクエリパラメータ）のみに影響を及ぼすことができます。一方、XML リライタは、コンテンツデータベース、Modules データベース、トランザクション ID の他、XQuery アプリケーションで通常は `eval-into` を必要とする各種の設定を変更できます。静的コンテンツのリクエストなど、場合によっては、動的コンテンツのリクエストは引き続きインターセプトされるものの、リクエストに XQuery コードを使用する必要がまったくなくなることがあります。

XML リライタを使用すると、XCC クライアントが、REST および HTTP クライアントと同じポートで通信できるようになります。また、XCC と同じ機能でリクエストを実行できますが、XCC ライブラリは使用しません。

17.2 XML リライタを使用するためのアプリケーションサーバー設定

XML リライタを使用するには、HTTP サーバーのサーバー設定の［rewriter］フィールドに、XML リライタ（.xml 拡張子を持つファイル）を指定します。

例えば、App-Services サーバー（ポート 8000）の XML リライタは、次の場所にあります。

```
<marklogic-dir>/Modules/MarkLogic/rest-api/8000-rewriter.xml
```

17.3 入出力コンテキスト

リライタは、定義された入力コンテキストで呼び出されます。コンテキストに対して事前定義された変更の集まりが、出力コンテキストに適用されます。これらの変更は、リクエストハンドラに返されて、検証および適用されます。リライタ自体は、入出力コンテキストに対する変更を直接実装しません。

このセクションで説明するトピックは次のとおりです。

- 入力コンテキスト
- 出力コンテキスト

17.3.1 入力コンテキスト

リライタの入力コンテキストは、マッチルールからアクセスできるマッチングプロパティ（「マッチルール」（238 ページ）を参照）の組み合わせ、またはグローバルシステム変数（「システム変数」（255 ページ）を参照）から成ります。プロパティのマッチングルールが評価されると、マッチ結果に対して、スコープがローカルである変数が生成されます。これは、子ルールで使用できます。

以下の表に示すプロパティは、マッチルールのコンテキストとして使用可能です。「regex」と示されている場合、マッチは正規表現で実行されます。それ以外の場合、マッチは、プロパティの 1 つあるいは複数のコンポーネントと「等価」です。

プロパティ / 説明	マッチのサポート
path	regex
param	name [value]
HTTP header	name [value]
HTTP method	name in list
user	name or id
default user	is default user
execute privilege	in list

17.3.2　出力コンテキスト

出力コンテキストは、リライタによる実行が可能な（かつ許可されている）値とアクションから成ります。これらは、コンテキスト値と、その値で許可されたリライタコマンドの集まりとして表現できます。出力コンテキストプロパティは省略できます。省略した場合、対応する入力コンテキストは変更されません。シンプルなケースでは、リライタからの出力がなく、入力コンテキストは変更されません。例えば、新しいデータベース ID が出力で指定されているが、現在のデータベースと同じである場合、変更は必要ありません。競合する出力コンテキストをリライタが生成しないようにしてください。ただし、一貫性を保ち、その他の制約（パーミッションなど）に従うための変更を検証するのは、最終的にはリクエストハンドラです。結果的に、許可されていないアクションや無効なアクション（存在しないデータベースに対する設定や、ユーザーがパーミッションを持たないエンドポイントへの書き換えなど）をリライタが実行すると、エラー処理が実行されます。

入力コンテキストプロパティ、外部パス、および外部クエリは、出力コンテキストで変更できます。以下の表に示すように、特定のデータベースにリクエストを転送したり、特定のトランザクションを設定したりするために出力コンテキストに追加できるその他のプロパティもあります。

プロパティ	説明
path*	URI の書き換えられたパスコンポーネント
query*	書き換えられたクエリパラメータ
module-database	Modules データベース
root	Modules ルートパス
database	データベース
eval	パスを評価する場合は True 直接アクセスの場合は False
transaction	トランザクション ID
transaction mode	クエリまたは更新トランザクションモードを指定
error format	サーバーで生成されたエラーのエラー形式を指定

* これらは入力コンテキストから変更されます。

17.4 正規表現（Regex）

特にパス向けの一般的なユースケースは、正規表現を使用した「マッチさせて抽出」（または「マッチ / キャプチャ」）という概念です。

`fn:replace` XQuery 関数の正規表現ルールの場合と同様に、文字列で最初の（重複しない）マッチだけが処理され、残りは無視されます。

例えば、以下に示すパスがある場合、一般形式をマッチさせると同時に、1 つの正規表現にコンポーネントを抽出するとします。

```
/admin/v2/meters/databases/12345/total/file.xqy
```

次のパスマッチルール regex は、上記のパスをマッチすると同時に目的のコンポーネント（「マッチグループ」）を抽出し、以下の表に示すように番号付き変数としてローカルコンテキストに設定します。

```
<match-path matches="/admin/v(.)/([a-z]+)/([a-z]+)/([0-9]+)/
([a-z]+)/.+\.xqy">
```

変数	値
$0	/admin/v2/meters/databases/12345/total/file.xqy
$1	2
$2	meters
$3	databases
$4	12345
$5	total

抽出した値は、追加のクエリパラメータなどの出力値を構築するために使用できます。

> **注：** この例では、アンカー（「^ .....$」）が使用されていません。そのため、この正規表現は、以下のような 1 つの文字列にもマッチし、同じ結果となります。
>
> ```
> somestuff/admin/v2/meters/databases/12345/total/file.xqy/
> morestuff
> ```

ルールが regex（`matches` 属性が示す）をマッチするケースでは必ず、flags オプションを使用できます。現在サポートされているのは「i」フラグ（大文字と小文字を区別しない）のみです。

17.5 マッチルール

マッチルールは、エバリュエータ実行フローを制御します。マッチルールは複数のステップで評価されます。

1. ルールが評価され、マッチかどうかを判断します。

2. マッチである場合、ルールは 0 個以上の「評価式」（ローカル変数 $*,$0 ... $n）を生成する可能性があります。

3. マッチである場合、エバリュエータは、マッチルール内へと下降します。そうでない場合は、マッチが「マッチではない」とみなされ、エバリュエータは次の兄弟に進みます。

4. 最後の兄弟である場合、エバリュエータは親へと「上昇」します。

下降：マッチ時にマッチルールを下降するときは、次のステップが発生します。

1. 「スコープ設定されている」（属性 `scoped=true`）場合は、現在のコンテキスト（スコープ内のすべてのユーザー定義変数と現在アクティブなすべての変更リクエスト）がプッシュされます。

2. 親からのすべての評価式（$*,$0..$n）はクリアされ、マッチングノードによって生成された評価式に置換されます。

3. 評価は最初の子ノードに進みます。

上昇：上昇時（最後の兄弟を評価した後）、エバリュエータは親ノードへと上昇します。次のステップが発生します。

1. 親がスコープ設定（属性 `scoped=true`）されていた場合は、現在のコンテキストがポップされ、親ノードのコンテキストによって置換されます。それ以外の場合は、コンテキストが変更されません。

2. 評価式（$*,$0...）がポップされ、親のスコープ内評価式に置換されます。

   注： これは、スコープ設定された属性の影響を受けません。評価式は常に、その式を生成したマッチノードの直接の子のみにスコープ設定されます。

3. 評価は、親ノードの次の兄弟に進みます。

   注： 上昇は、まれなケースであり、できるだけ避けるようにしてください。

以下の表は、マッチルールの要約です。その後に、各ルールの詳しい説明があります。

要素	説明
rewriter	リライタルールツリーのルート要素です。
match-accept	HTTP Accept ヘッダでマッチします。
match-content-type	HTTP Content-Type ヘッダでマッチします。
match-cookie	Cookie でマッチします。
match-execute-privilege	ユーザー実行権限でマッチします。
match-header	HTTP ヘッダでマッチします。
match-method	HTTP メソッドでマッチします。
match-path	リクエストパスでマッチします。
match-role	ユーザーに割り当てられたロールでマッチします。
match-string	正規表現と照合して文字列値をマッチします。
match-query-param	URI パラメータ（クエリパラメータ）でマッチします。
match-user	ユーザー名、ID、またはデフォルトユーザーでマッチします。

17.5.1 rewriter

リライタルールツリーのルート要素です。

属性：なし

例：

/home/** にマッチするものをすべてモジュール gohome.xqy にリダイレクトし、マッチしない場合はリクエストを通過するシンプルなリライタ

```
<rewriter xmlns="http://marklogic.com/xdmp/rewriter">
     <match-path prefix="/home/">
           <dispatch>gohome.xqy</dispatch>
     </match-path>
</rewriter>
```

17.5.2 match-accept

Accept HTTP ヘッダでマッチします。

属性

名前	型	必須	目的
@any-of	list of strings	はい	指定したメディアタイプのいずれかが Accept ヘッダに含まれる場合にマッチします。
@scoped	boolean	いいえ デフォルトは **false**	このルールがその子のための新しい「スコープ」コンテキストを作成することを示します。
@repeated	boolean	いいえ デフォルトは **false**	false の場合は、マッチが繰り返されると即時エラーになります。

子コンテキストの変更

変数	型	値
$0	string	string としてマッチしたメディアタイプ。
$*	list of strings	list of strings としてマッチしたメディアタイプ。

注： マッチは、タイプ / サブタイプのリテラル文字列と照合して、大文字と小文字を区別するマッチとして実行されます。サブタイプ、メディアレンジ、品質要素のいずれの展開も評価されません。

例：

メディアタイプ `application/xml` または `text/plain` が Accept ヘッダ内で指定されている場合、`/handle-text.xqy` にディスパッチします。

```
<match-accept any-of="application/xml text/html">
      <dispatch>/handle-text.xqy</dispatch>
</match-accept>
```

17.5.3 match-content-type

Content-Type HTTP ヘッダでマッチします。

属性

名前	型	必須	目的
@any-of	list of strings	はい	指定した型のいずれかが Content-Type ヘッダに含まれる場合にマッチします。
@scoped	boolean	いいえ デフォルトは **false**	このルールがその子のための新しい「スコープ」コンテキストを作成することを示します。

子コンテキストの変更

変数	型	値
$0	string	string としてマッチした最初の型。

注： マッチは、タイプ / サブタイプのリテラル文字列と照合して、大文字と小文字を区別するマッチとして実行されます。サブタイプ、メディアレンジ、品質要素のいずれの展開も評価されません。

例：

メディアタイプ `application/xml` または `text/plain` が Content-Type ヘッダ内で指定されている場合、`/handle-text.xqy` にディスパッチします。

```
<match-content-type any-of="application/xml text/html">
      <dispatch>/handle-text.xqy</dispatch>
</match-content-type>
```

17.5.4 match-cookie

Cookie の名前でマッチします。Cookie は、よく知られている構造化形式の HTTP ヘッダです。

属性

名前	型	必須	目的
@name	string	はい	指定した名前の Cookie が存在する場合にマッチします。 Cookie 名は、大文字と小文字を区別せずにマッチします。
@scoped	boolean	いいえ デフォルトは **false**	このルールがその子のための新しい「スコープ」コンテキストを作成することを示します。

子コンテキストの変更

変数	型	値
$0	string	マッチする Cookie のテキスト値。

例：

Cookie 値 SESSIONID が存在する場合は、それを変数 `$session` に設定します。

```
<match-cookie name="SESSIONID">
       <set-var name="session">$0</set-var>
       ....
</match-cookie>
```

17.5.5 match-execute-privilege

ユーザー実行権限でマッチします。

属性

名前	型	必須	目的
@any-of	list of uris	いいえ *	指定された実行権限を 1 つ以上、ユーザーが持つ場合にマッチします。
@all-of	list of uris	いいえ *	指定された実行権限をすべて、ユーザーが持つ場合にマッチします。
@scoped	boolean	いいえ デフォルトは **false**	このルールがその子のための新しい「スコープ」コンテキストを作成することを示します。

*@any-of または @all-of のいずれか 1 つのみを必ず指定してください

注： 実行権限は、名前ではなく、URI です。例を参照してください。

子コンテキストの変更：

変数	型	値
$0	string	マッチする権限。マッチが複数ある場合は、スペース区切り文字列に変換されます。
$*	list of strings	list of strings としてマッチするすべての権限。

例：

ユーザーが `admin-module-read` または `admin-ui` 権限を持つ場合にディスパッチします。

```
<match-execute-privilege
     any-of="http://marklogic.com/xdmp/privileges/admin-mo
dule-read
          http://marklogic.com/xdmp/privileges/admin-ui">
     <dispatch/>
</match-execute-privilege>
```

注： XML 形式では、属性で復帰改行を使用できます。

17.5.6 match-header

HTTP ヘッダでマッチします。

属性

名前	型	必須	目的
@name	string	はい	その名前に等しいヘッダが存在する場合にマッチします。 HTTP ヘッダ名は、大文字と小文字を区別しない同等の文字列とマッチします。
@value	string	いいえ	その名前と値を持つヘッダが存在する場合にマッチします。 名前は、大文字と小文字を区別せずに比較されますが、値では大文字と小文字が区別されます。
@matches	regex	いいえ	regex でマッチします。
@flags	string	いいえ	必要に応じて使用する regex フラグ。 大文字と小文字を区別しない場合は「i」です。
@scoped	boolean	いいえ デフォルトは **false**	このルールがその子のための新しい「スコープ」コンテキストを作成することを示します。
@repeated	boolean	いいえ デフォルトは **false**	`false` の場合は、マッチが繰り返されるとエラーになります。

`@value` と `@matches` のいずれかのみを指定できますが、両方を省略することもできます。

子コンテキストの変更：

`@value` が指定された場合、`$0` は、マッチする値に設定されます。

@matches 属性と @value 属性のいずれもない場合、$0 は、その名前のヘッダのテキストコンテンツ全体になります。複数のヘッダがマッチする場合、@repeated は、この状態がエラーであるか、許可されるかを示します。 許可される場合（true）、$* は個々の値に設定され、$0 は、スペース区切りで連結したすべてのヘッダになります。false（デフォルト）の場合、マッチが複数であると、エラーが生成されます。

@matches が指定されている場合は、match-path や match-string と同様に、regex マッチの結果が $0 .. $N になります。

変数	型	値
$0	string	マッチしたヘッダの値。
$1....$N	string	マッチする各グループ。

例：

ユーザーエージェントに FireFox/20.0 または FireFox/21.0 が含まれる場合に、クエリパラメータを追加します。

```
<match-header name="User-Agent"
matches="FireFox/2[01]\.0">
     <add-query-param name="do-firefox">yes</add-query-param>
     ...
</match-header>
```

17.5.7 match-method

HTTP メソッドでマッチします。

属性

名前	型	必須	目的
@any-of	list of strings	はい	HTTP メソッドがリスト内のいずれかの値である場合にマッチします。メソッド名のマッチでは、大文字と小文字が区別されます。
@scoped	boolean	いいえ デフォルトは **false**	このルールがその子のための新しい「スコープ」コンテキストを作成することを示します。

少なくとも 1 つのメソッド名を指定する必要があります。

子コンテキストの変更： なし

HTTP メソッドの値はシステムグローバル変数 `$_method` です（「システム変数」（255 ページ）を参照）。

例：

メソッドが GET HEAD または OPTIONS であると同時にユーザーが実行権限 http://marklogic.com/xdmp/privileges/manage を持つ場合にディスパッチします。

```
<match-method any-of="GET HEAD OPTIONS">
     <match-execute-privilege
         any-of="http://marklogic.com/xdmp/privileges/manage">
         <set-path>/history/endpoints/resources.xqy</set-path>
         <dispatch/>
     </match-execute-privilege>
</match-method>
```

17.5.8　match-path

リクエストパスでマッチします。「パス」とは、RFC3986（https://tools.ietf.org/html/rfc3986）に従って、リクエスト URI の「パス」コンポーネントを指します。これは、URL で、スキームおよびオーソリティセクションの後にある部分のことです。先頭は「/」（何も指定されていない場合でも）で、末尾はクエリパラメータ区切り文字「?」の直前までです。フラグメント（「#」）は含まれません。

match-path で使用される場合、パスは URL でデコードされませんが、クエリパラメータ値はデコードされます（HTTP 仕様に従う）。これは意図的な動作です。これにより、この動作がない場合はパスコンポーネントの区切り文字であると解釈される文字を、パスコンポーネントに含めることができます。また、HTTP 仕様は、エンコードなしではパスの「目的」が曖昧である場合に限ってパスコンポーネントがエンコードされることを明確に規定しているため、パス内の文字が URL エンコードされるのは、パス区切りコンポーネント（または予約済みの URL 文字）であるとみなされないことが意図されている場合のみとなります。

例えば、次の URL を考えてみます。

```
http://localhost:8040//root%2Ftestme.xqy?name=%2Ftest
```

これは、次の HTTP リクエストとしてサーバーから受信しました。

```
GET /root%2Ftestme.xqy?name=%2Ftest
```

これは、次のようにパースされます。

```
PATH:  /root%2Ftestme.xqy
```

クエリ（名前 / 値ペアがデコードされている）: ("name", "/test")

これは、match-path を使用すると、次のような URL との区別が可能です。

```
http://localhost:8040//root/testme.xqy?name=%2Ftest
```

これは、次のようにパースされます。

```
PATH:  /root/testme.xqy
```

例えば、`<match-path matches="/root([^/].*)">` は、1 番目の URL にマッチしますが、同一パスへデコードされたとしても、2 番目とはマッチしません。

マッチ結果が $0..$n に代入されると、デフォルトの動作では結果がデコードされるため、上記の例では、$1 が「/testme.xqy」になります。 これは、やはりデコード形式である他のパラメータとの一貫性を処理するためであり、特に値がクエリパラメータとして設定される場合は、書き換えの一環として「URL エンコード」されます。 値がすでにエンコード形式であった場合は、2 回エンコードされるため、誤った値になります。

マッチ後の結果を path-match でデコードすることが望ましくない（まれな）場合、属性 @uri-decode を指定して false に設定できます。

属性

名前	型	必須	目的
@matches	regex	いいえ *	パスが正規表現にマッチする場合にマッチします。
@prefix	string	いいえ *	パスの先頭がプレフィックス（リテラル文字列）である場合にマッチします。
@flags	string	いいえ *	必要に応じて使用する regex フラグ。 大文字と小文字を区別しない場合は「i」です。
@any-of	list of strings	いいえ *	パスが完全一致のリストのいずれかである場合にマッチします。
@url-decode	boolean	いいえ デフォルトは **true**	true（デフォルト）の場合は、結果が、パスのマッチ部分から抽出後に URL デコードされます。
@scoped	boolean	いいえ デフォルトは **false**	このルールがその子のための新しい「スコープ」コンテキストを作成することを示します。

@matches、@prefix、@any-of のうち、1 つのみを指定できます。

指定した場合は、@matches、@prefix、または @any-of に、空でない値を指定する必要があります。

@flags は、（@prefix でも @any-of でもなく）@matches のみに適用されます。

@matches、@prefix、@any-of のいずれも指定しない場合、match-path はすべてのパスにマッチします。

空のパスにマッチするには、（任意のパスにマッチする matches="" ではなく）matches="^$" を使用します。

すべてのパスにマッチするには、@matches、@prefix、および @any-of を省略します。

子コンテキストの変更：

変数	型	値
$0	string	マッチしたパスの全体。マッチした場合は、マッチするテキストの全体です。 @prefix の場合は、プレフィックスのパターンです。 @any-of の場合は、リスト内でマッチした文字列です。
$1 ...$N	string	@matches のみで使用します。 XQuery 関数 fn:replace() で定義される、数値マッチグループの値です。

例：

```
<match-path
    matches="^/manage/(v2|LATEST)/meters/labels/
    ([^F$*/?&amp;]+)/?$">
       <add-query-param name="version">$1</add-query-param>
       <add-query-param name="label-id">$2</add-query-param>
       <set-path>/history/endpoints/labels-item.xqy</set-path>
       ...
</match-path>
```

17.5.9 match-query-param

クエリパラメータでマッチします。

クエリパラメータは、完全一致（名前および値クオリティによる）にすることも、部分一致（名前マッチのみを指定）にすることもできます。完全一致では、1 つの名前 / 値ペアのみがマッチします。部分一致では、クエリパラメータに同名のパラメータが複数ある場合、同じ名前の複数の名前 / 値ペアがマッチする可能性があります。repeated 属性は、これがエラーであるかどうかを指定します。デフォルト（false）では、マッチするパラメータが繰り返されるとエラーになります。

属性：

名前	型	必須	目的
@name	string	はい	その名前を持つクエリパラメータが存在する場合にマッチします。
@value	string	いいえ	その名前と値を持つクエリパラメータが存在する場合にマッチします。
@scoped	boolean	いいえ デフォルトは **false**	このルールがその子のための新しい「スコープ」コンテキストを作成することを示します。
@repeated	boolean	いいえ デフォルトは **false**	false の場合は、マッチが繰り返されると即時エラーになります。

注： 存在するものの空である @value パラメータが有効であると同時に、空の値のクエリパラメータの存在にマッチする場合。

子コンテキストの変更：

変数	型	値
$0	String	マッチしたクエリパラメータの値。 クエリパラメータにマッチする値が複数ある（同名のパラメータが複数存在するため）場合、マッチする値はスペース区切りの文字列に変換されます。
$*	list of strings	マッチしたすべての値のリスト。$0 と同様ですが、list of strings である点が異なります。

例：

クエリパラメータ user の値が「admin」であると同時に、このユーザーが実行権限 http://marklogic.com/xdmp/privileges/manage を持つ場合は、/admin.xqy にディスパッチします。

```
<match-query-param name="user" value="admin">
      <match-execute-privilege
           any-of="http://marklogic.com/xdmp/privileges/manage">
           <dispatch>/admin.xqy</dispatch>
      /match-execute-privilege>
</match-query-param>
```

クエリパラメータにトランザクションが含まれる場合は、トランザクション ID を設定します。

```
<match-query-param name="transaction">
      <set-transaction>$0</set-transaction>
      ...
</match-query-param>
```

空のクエリパラメータが存在するかどうかをテストします。

URI の場合： /query.xqy?a=has-value&b=&c=cvalue

このルールは、「b」の値が空の場合は「default」に設定し、それ以外の場合は、

```
<match-query-param name="b" value="">
      <set-query-param name="b" value="default"/>
</match-query-param>
```

同名のクエリパラメータを複数使用する例については、match-string を参照してください。

17.5.10 match-role

ユーザーに割り当てられたロールでマッチします。

属性

名前	型	必須	目的
@any-of	list of role names (strings)	いいえ *	ユーザーが、指定されたロールを 1 つ以上持つ場合にマッチします。
@all-of	list of role names (strings)	いいえ *	ユーザーが、指定されたロールをすべて持つ場合にマッチします。
@scoped	boolean	いいえ デフォルトは **false**	このルールがその子のための新しい「スコープ」コンテキストを作成することを示します。

*@any-of と @all-of のいずれかを必ず指定してください。

子コンテキストの変更：

変数	型	値
$0	string	any-of の場合は、マッチした最初のロール。 それ以外の場合は、未設定（all-of がマッチした場合、それらのロールが何か、つまり、@all-of のコンテンツが既知である）。

例：

ユーザーが infostudio-user と rest-user の両方のロールを持つ場合にマッチします。

```
<match-role all-of="infostudio-user rest-user">
        ...
</match-role>
```

17.5.11 match-string

文字列式を正規表現と照合します。 値がマッチする場合、ルールは成功し、その子がさらに適用されます。

このルールは、現在のルールでは不十分であるか、すべてのルールにマッチする追加の正規表現を実装するには複雑すぎる場合に、ルールを補完するために使用します。このルールは、確実に必要である場合を除き、使用しないでください。

属性

名前	型	必須	目的
@value	string	はい	マッチの基準となる値。リテラル文字列または単一変数式です。
@matches	正規表現の string	はい	値が正規表現にマッチする場合にマッチします。
@flags	string	いいえ	必要に応じて使用する regex フラグ。大文字と小文字を区別しない場合は「i」です。
@scoped	boolean	いいえ デフォルトは **false**	このルールがその子のための新しい「スコープ」コンテキストを作成することを示します。
@repeated	boolean	false	false の場合は、マッチが繰り返されるとエラーになります。

子コンテキストの変更：

変数	型	値
$0	string	マッチした値の全体。
$1 ... $N	string	マッチする数値グループの値。

Regex（正規表現）を参照してください。

マッチの繰り返し：regex が文字列の先頭および末尾に固定されていない場合は、正規表現が、文字列の複数の非重複部分にマッチする可能性があります。

17.5.12 match-user

ユーザー名またはデフォルトユーザーでマッチします。

ユーザーにマッチするには、@name のいずれか 1 つのみを使用します。

ユーザーがデフォルトユーザーである場合にマッチするには、@default-user を使用します。

@name と @default-user の両方を指定すると、両方をマッチできます。

@default-user はデフォルトで false です。

@default-user が false である場合、デフォルトユーザーであるかどうかはチェックされません。

属性

名前	型	必須	目的
@name	string	いいえ *	ユーザー名にマッチします。
@default-user	boolean （true | false）	いいえ * デフォルトは **false**	true の場合は、ユーザーがデフォルトユーザーである場合にマッチします。
@scoped	boolean	いいえ デフォルトは **false**	このルールがその子のための新しい「スコープ」コンテキストを作成することを示します。

子コンテキストの変更：なし：name または default-user をシステム変数として使用できます。「システム変数」を参照してください。

例：

ユーザーが認証されている場合にマッチします。デフォルトユーザーではないことにより暗黙で示されます。

```
<match-user default-user="false">
       ...
</match-user>
```

17.6 システム変数

このセクションでは、初期状態の入力コンテキストを構成する、事前定義されたシステム変数について説明します。これらは、任意の変数置換式のコンテキストで使用できます。

システム変数は、XQuery リライタが使用するメカニズムの代わりとして使用され、XQuery API を呼び出して、この情報やその他の多くの情報を取得できます。宣言型リライタは、API 呼び出しを公開しません。そのため、出力内に値が必要な場合は、この呼び出しがグローバル変数として使用可能になります。 これらの変数とマッチルールは部分的に重複しており、マッチは不要で、値の設定が必要であるケースが単純化されています。例えば、set-database ルールで、データベースを現在の modules データベースに設定する（モジュールファイルでの GET および PUT 操作を許可するために）とします。modules データベース用のシステム変数（$_modules-database）を指定すると、値の抽出のみを目的とした、modules データベースに対するマッチングルールは不要になります。

システム変数は、予約済みのプレフィックス「_」を使用します。これにより、新しい変数が追加された場合に、現在や将来のコードで誤って使用されることが防止されます。 システム変数の上書きは、現在のスコープ内でのみ設定され、システムの変更は発生しません。

ピリオド（「.」）は、プロパティへのアクセスという概念を示唆する規則ですが、実際は変数名の一部にすぎません。 変数の先頭が、同じプレフィックスであるが、サフィックスが「.<name>」である場合、ドットを含まない名前が最も有用な値として評価され、ドットを含む名前は、その変数の特定のプロパティまたは型を指定します。例えば、$_database はデータベース名、$_database.id はデータベース ID です。

「変数」や「型」に示すように、すべての変数の実際の型は string（または list of strings）です。以下の表の「型」列は、その変数で指定可能な値の範囲を示しています。例えば、データベース ID は符号なし long として生成されているため、数値を想定しているすべての式で安全に使用できます。

注：

- [*name*] は、*name* がオプションであることを示します。
- *<name>* は、*name* が事前定義された定数ではないものの必須であることを示します。

変数	型	説明 / 注
`$_cookie.<name>`	string	Cookie の <name> 値。Cookie のテキスト値のみが返されます。その他のメタデータ（パス、ドメイン、有効期限など）は返されません。Cookie が存在しない場合は、"" として評価されます。Cookie 名は、大文字と小文字が区別されずにマッチおよび比較されます。 今後、Cookie ヘッダのサブ構造を公開する可能性があります。

変数	型	説明 / 注
$_database[.name]	string	コンテンツデータベースの名前。
$_database.id	integer	コンテンツデータベースの ID。
$_defaultuser	boolean	認証されたユーザーがデフォルトユーザーである場合は true です。
$_method	string	HTTP メソッド名。
$_modules-database[.name]	string	Modules データベース名。name が指定されない場合、ファイルシステムがモジュールに使用されます。
$_modules-database.id	integer	Modules データベース ID。ID を 0 に設定すると、ファイルシステムがモジュールに使用されます。
$_modules-root	string	Modules ルートパス。
$_path	string	HTTP リクエストパス。クエリ文字列は含みません。
$_query-param.<name>	list of strings	名前の list of strings にマッチするクエリパラメータ。
$_request-url	string	元のリクエスト URI。パスとクエリパラメータを含みます。
$_user[.name]	string	ユーザー名。
$_user.id	integer	ユーザー ID。

モジュール用のファイルシステムを設定します。

```
<set-database>$_modules-database</set-database>
```

トランザクションを Cookie の TRANSACTION_ID に設定します。

```
<set-transaction>$_cookie.TRANSACTION_ID</set-transaction>
```

17.7 評価ルール

評価ルールは、エバリュエータの実行制御に影響を及ぼしません。評価ルールは、到達したときに評価され、現在のコンテキストのみに影響を及ぼすことができます。実行フローは制御しません。

評価ルールには 2 種類あります。設定ルールと割り当てルールです。

設定ルールは、リライタコマンド（何らかの方法で出力コンテキストを変更するリクエスト）を作成するルールです。 割り当てルールは、ローカルでスコープ設定された変数を設定するルールであり、リライタコマンドを生成しません。

変数およびリライタコマンドは、現在のスコープに置かれます。

要素	説明
add-query-param	クエリパラメータ（名前 / 値）をクエリパラメータに追加します。
set-database	データベースを設定します。
set-error-format	システムで生成されるエラーのエラー形式を設定します。
set-error-handler	エラーハンドラを設定します。
set-eval	評価モード（eval または direct）を設定します。
set-modules-database	Modules データベースを設定します。
set-modules-root	Modules ルートパスを設定します。
set-path	URI パスを設定します。
set-query-param	クエリパラメータを設定します。
set-transaction	トランザクションを設定します。
set-transaction-mode	トランザクションモードを設定します。
set-var	ローカルスコープ内で変数を設定します。
trace	トレースメッセージをログに記録します。

17.7.1 add-query-param

クエリパラメータ（名前 / 値）をクエリパラメータに追加（付加）します。

属性

名前	型	必須	目的
@name	string	はい	パラメータの名前。

子供：

パラメータの値として評価される式。

この場合も、空の要素またはリストは、空の値のクエリパラメータを付加します（http://company.com?a= のような URL と等価）。

式がリストである場合、リストの値ごとに 1 回、クエリパラメータが複製されます。

例：

パスがマッチする場合、クエリパラメータに付加されます。

- version = マッチしたバージョン
- label-id = マッチしたラベル ID

```
<match-path
     matches="^/manage/(v2|LATEST)/meters/labels/
     ([^/?&amp;]+)/?$">
    <add-query-param name="version">$1</add-query-param>
    <add-query-param name="label-id">$2</add-query-param>
</match-path>
```

17.7.2 set-database

データベースを設定します。

リクエストの残りの部分用にコンテキストデータベースを変更します。

属性

名前	型	必須	目的
@checked	boolean [true,1 | false,0]	いいえ	true の場合、ユーザーの eval-in 権限がチェックされ、変更が許可されるかどうかが検証されます。

子供：

データベース ID またはデータベース名として評価される式。

値のリストとして評価される式を使用して値を設定すると、即時エラーになります。

データベース参照の解釈方法については、「データベース（名前または ID）」を参照してください。

@checked フラグの注意事項

@checked フラグは、リライタ変更結果フェーズで解釈されます。これは、ディスパッチの前に正常に評価された最後の set-database のみが使用されることを意味します。

@checked フラグが true であると同時に、データベースが、アプリケーションサーバーで定義されたデータベースと異なる場合、ユーザーには eval-in 権限が必要です。

例：

データベースを「SpecialDocuments」に設定します。

```
<set-database>SpecialDocuments</set-database>
```

データベースを現在の Modules データベースに設定します。

```
<set-database>$_modules-database</set-database>
```

17.7.3 set-error-format

システムで生成されるすべてのエラーに対して使用されるエラー形式を設定します。これは、成功しなかった HTTP 応答に対するエラーメッセージのボディの形式（content-type）です。

この設定は、アプリケーションサーバー設定による設定を上書きし、リライタルールの検証に成功すると直ちに有効になります。

属性：なし

子供：次のエラー形式のいずれかとして評価される式。

- html
- json
- xml
- compatible

「compatible」形式は、リクエストとエラーのタイプについて、以前のリリースで使用された形式に可能な限り近い形式に、システムがマッチすることを示します。 例えば、ディスパッチが「xdbc」を示す場合、「compatible」は、XCC クライアントライブラリと互換性のある HTML 形式でエラーを生成します。

値のリストとして評価される式を使用して値を設定すると、即時エラーになります。

> **注**： この設定は、ユーザー定義のエラーハンドラに影響を及ぼしません。このエラーハンドラは、任意の形式やボディを自由に出力できます。

例：

json 応答のエラー形式を設定します。

```
<set-error-format>json </set-database>
```

17.7.4 set-error-handler

エラーハンドラを設定します。

属性：なし

子供：パスとして評価される式（空白ではない文字列）。

例：

```
<set-error-handler >/myerror-handler.xqy</set-modules-root>
```

書き換えプロセス時にエラーが発生すると、アプリケーションサーバーに関連付けられているエラーハンドラを使用してエラーが処理されます。書き換えに成功した後、set-error-handler が新しいエラーハンドラを指定している場合は、そのエラーハンドラがエラー処理に使用されます。

エラーハンドラを見つけるために使用される Modules データベースおよび Modules ルートは、エラーの発生時に有効な Modules データベースおよびルートです。

エラーハンドラを空の文字列に設定すると、リクエストの残りに対して、ユーザー定義のエラーハンドラを使用できなくなります。

値のリストとして評価される式を使用して値を設定すると、即時エラーになります。

例えば、さらに set-modules-database ルールが使用された場合、書き換え後の Modules データベース（および set-modules-root で設定されたルート）で新しいエラーハンドラが検索されます。それ以外の場合は、アプリケーションサーバーで設定されている Modules データベースでエラーハンドラが検索されます。

17.7.5 set-eval

評価モード（eval または direct）を設定します。

評価モードは、パスが評価されるのか（XQuery または JavaScript）、直接アクセスされるのか（PUT/GET）を判断するために、リクエストハンドラで使用されます。

評価可能なドキュメント（Modules データベース内の）に読み書きを実行できるようにするには、評価モードを direct に設定し、データベースを Modules データベースに設定する必要があります。

属性：なし

子供：「eval」または「direct」として評価される式。

例：

ファイル名が .xqy で終わる場合は、評価ではなく、直接ファイルアクセスを強制的に実行します。

```
<match-path matches=".*\.xqy$">
     <set-eval>direct</set-eval>
</match-user>
```

17.7.6 set-modules-database

Modules データベースを設定します。

リクエスト用の Modules データベースを設定します。

属性

名前	型	必須	目的
@checked	boolean [true,1 | false,0] デフォルトは **false**	いいえ	true の場合、ユーザーのパーミッションに eval-in 権限が含まれるかどうかがチェックされ、変更が許可されるかどうかが検証されます。

子供：

データベース ID またはデータベース名として評価される式。空の値、式、または「0」として評価される式は、「ファイルシステム」を示します。それ以外の場合は、データベース名または ID として値が解釈されます。

データベース参照の解釈方法については、「データベース（名前または ID）」を参照してください。

値のリストとして評価される式を使用して値を設定すると、即時エラーになります。

@checked フラグの注意事項

@checked フラグは、リライタ変更結果フェーズで解釈されます。これは、ディスパッチの前に正常に評価された最後の set-database のみが使用されることを意味します。

@checked フラグが true であると同時に、データベースが、アプリケーションサーバーで定義された Modules データベースと異なる場合、ユーザーには eval-in 権限が必要です。

例：

データベースを「SpecialDocuments」に設定します。

```
<match-user name="admin">
      <set-modules-database>SpecialModules</set-modules-
       database>
      ...
</match-user>
```

17.7.7 set-modules-root

Modules ルートパスを設定します。

属性：なし

子供：パスとして評価される式（空白ではない文字列）。

値のリストとして評価される式を使用して値を設定すると、即時エラーになります。

例：

Modules ルートパスを /myapp に設定します。

```
<set-modules-root>/myapp</set-modules-root>
```

17.7.8 set-path

リクエストの URI パスを設定します。

多くの場合、これがリライタの主なユースケースです。

属性：なし

子供：

パスとして評価される式（空白ではない文字列）。

値のリストとして評価される式を使用して値を設定すると、即時エラーになります。

例：

ユーザー名が「admin」である場合、パスを /admin.xqy に設定します。

メソッドが GET、HEAD、OPTIONS のいずれかである場合はディスパッチします。メソッドが POST である場合は、クエリパラメータ「verified」を true に設定してからディスパッチします。

```
<match-user name="admin">
    <set-path>/admin.xqy</set-path>
    <match-method any-of="GET HEAD OPTIONS">
        <dispatch/>
    </match-method>
    <match-method any-of="POST">
        <set-query-param
name="verified">true</set-query-param>
```

```
            <dispatch/>
        </match-method>
    </match-user>
```

同一ルール内で set-path と dispatch を使用する方法については、4.1.5.6.1 を参照してください。

17.7.9 set-query-param

クエリパラメータを設定（上書き）します。そのクエリパラメータが以前に存在していた場合、そのすべての値が新しい値に置換されます。

属性

名前	型	必須	目的
@name	string	はい	パラメータの名前。

子供：

設定されるクエリパラメータの値として評価される式。式がリストである場合、リストの値ごとに 1 回、クエリパラメータが複製されます。

この場合も、空の要素、空の文字列値、または空のリスト値は、空の値のクエリパラメータを設定します（http://company.com?a= のような URL と等価）。

例：

ユーザーが admin である場合、クエリパラメータ user を admin に設定し、設定されていた以前の値を上書きします。

```
<match-user name="admin">
     <set-query-param name="user">admin</set-query-param>
</match-user>
```

クエリパラメータ「ids」のすべての値を新しいクエリパラメータ「app-ids」にコピーして、設定されていた以前の値を書き換えます。

```
<match-query-param name="ids">
     <set-query-param name="app-ids">$*</set-query-param>
</match-query-param>
```

これは、<dispatch> ルール内で @include-request-query-params が指定されているときに、クエリパラメータを名前で「パススルー」するために使用できます。

次のルールは、名前が「special」であるすべて（0 個以上）のクエリパラメータを結果にコピーし、他のパラメータはパススルーしません。

```
<match-query-param name="special" repeated="true">
      <set-query-param name=" special">$*</set-query-param>
</match-query-param>
<dispatch include-request-query-params="false"/>
```

17.7.10 set-transaction

現在のトランザクションを設定します。指定する場合は、set-transaction-mode も設定する必要があります。

属性：なし

子供：トランザクション ID として評価される式。

例：

Cookie の TRANSACTION_ID の値にトランザクションを設定します。

```
<set-transaction>$_cookie.TRANSACTION_ID</set-transaction>
```

注： set-transaction の式が空である場合（Cookie が存在しないときなど）、トランザクションは変更されません。

値のリストまたは 0 として評価される式を使用して値を設定すると、（リライタのパース時に）即時エラーになります。

17.7.11 set-transaction-mode

現在のトランザクションのトランザクションモードを設定します。指定する場合は、set-transaction も設定する必要があります。

属性：なし

子供：以下の 1 つの文字列で指定されたトランザクションモードとして評価される式。

（"auto" | "query" | "update"）

例：

クエリパラメータ「trmode」に値が存在する場合は、トランザクションモードをその値に設定します。

```
<match-query-param name="trmode">
      <set-transaction-mode>$0</set-transaction-mode>
</match-query-param>
```

注： トランザクションモードの値が "auto"、"query"、"update" のいずれでもない場合は、エラーです。値のリストとして評価される式を使用して値を設定した場合もエラーになります。

17.7.12 set-var

ローカルスコープ内で変数を設定します。

これは割り当てルールです。リライタコマンドを生成せず、変数を設定します。

割り当ては、現在のスコープ（親がプッシュする変数のリスト）のみに影響します。変数は、後続の兄弟とその子によって認識されます。

使用できるユーザー定義の変数名は、先頭が文字で、その後が 0 文字以上の文字、数字、アンダースコア、またはダッシュである必要があります。

具体的には、名前が正規表現パターン「[a-zA-Z][a-zA-Z0-9_-]*」にマッチする必要があります。

つまり、システム定義変数、プロパティコンポーネント、式変数のいずれも、set-var を使用して設定することはできません。

属性

名前	型	必須	目的
@name	string	はい	設定する変数の名前（「$」を含まない）

子供：

変数に設定する値として評価される式。

例：

マッチングパスの 1 番目のコンポーネントに変数 $dir1 を、2 番目のコンポーネントに $dir2 を設定します。

```
<match-path matches="^/([a-z]+)/([a-z]+)/.*">
    <set-var name="dir1">$1</set-var>
    <set-var name="dir2">$2</set-var>
    ...
</match-path
```

Modules データベース名に文字列「User」が含まれる場合は、変数 usedb を Modules DB の完全な名前に設定します。

```
<match-string  value="$_modules-database"
matches=".*User.*">
    <set-var name="usedb">$0</set-var>
</match-string>
```

クエリパラメータ「ids」のいずれかの値の全体が数字である場合は、そのすべての値にマッチします。

```
<match-query-param name="ids">
    <match-string  value="$*" matches="[0-9]+">
          ....
    </match-string>
</match-query-param>
```

17.7.13 trace

トレースメッセージをログに記録します。

トレースルールは、評価ルールを使用できるあらゆる場所で使用できます。`fn:trace` と同様のトレースメッセージを記録します。

event 属性は、トレースイベント ID を指定します。trace 要素のボディが、メッセージとしてログに記録されます。

属性

名前	型	必須	目的
@event	string	はい	トレースイベントを指定します。

子コンテンツ：トレースメッセージまたは式。

子要素：なし

子コンテキストの変更：なし

例：

```
<match-path prefix="/special">
    <trace event="AppEvent1">
        The following trace contains the matched path.
    </trace>
    <trace event="AppEvent2">
        $0
    </trace>
</match-path>
```

17.8 終了ルール

終了ルール（dispatch、error）は、現在のルールでエバリュエータを無条件に停止します。それ以降の検証は行われません。ディスパッチルールは、スコープ内で累積したすべてのリライタコマンドでエバリュエータから復帰します。エラールールは、すべてのコマンドを破棄し、エラー状態で復帰します。

要素	説明
dispatch	評価を停止し、すべての書き換えコマンドでディスパッチします。
error	エラーを生成して評価を終了します。

17.8.1 dispatch

評価を停止し、すべての書き換えコマンドでディスパッチします。

`dispatch` 要素は、マッチルールが含まれていない任意のマッチルールの最後の子として必要です。

属性

名前	型	必須	目的
@include-request-query-params	boolean デフォルトは **true**	いいえ	`true` の場合、リライタの適用前に、元のリクエストクエリパラメータがクエリパラメータの初期セットとして使用されます。
@xdbc	boolean デフォルトは **false**	いいえ	`true` の場合、リクエストに対してビルトイン XDBC ハンドラが使用されます。

属性 `include-request-query-params` は、初期のリクエストクエリパラメータがリライタの結果に含まれるかどうかを指定します。`true`（または未指定）の場合、リライタの変更は、初期のクエリパラメータによって開始され、ディスパッチ時にスコープ内にある set-query-param および add-query-param ルールによって増加し（追加またはリセットされ）ます。

`false` に設定すると、初期のリクエストパラメータが含まれず、set-query-param および add-query-param ルールによって設定または追加されるパラメータのみが結果に含まれます。

xdbc が指定され、true である場合、リクエストに対してビルトイン XDBC ハンドラが使用されます。XDBC サポートが有効な場合、最終的なパス（書き換え後の可能性あり）は、XDBC ビルトインハンドラでサポートされるパスの 1 つでなければなりません。

子コンテンツ：

空、または式。

子要素：

子要素が空または空白ではない場合、子要素が評価され、書き換えパスに使用されます。

子コンテキストの変更：

例：

```
<set-path>/a/path.xqy
     <dispatch/>
</set-path>
```

これは、以下と同等です。

```
<dispatch>/a/path.xqy</dispatch>
```

元の URL が /test?a=a&b=b である場合に、次のリライタを考えてみます。

```
<set-query-param name="a">a1</set-query-param>
<dispatch include-request-query-params="false">
  /run.xqy</dispatch>
```

これによりパス /run.xqy に書き換えられ、クエリパラメータは次のようになります。

```
a=a1
```

次のリライタがあるとします。

```
<set-query-param name="a">a1</set-query-param>
<dispatch>run.xqy</dispatch>
```

これによりパス /run.xqy に書き換えられ、クエリパラメータは次のようになります。

```
a=a1
b=b
```

XDBC にディスパッチする最小限の書き換えルールの例を次に示します。

```
<match-path  any-of="/eval /invoke /spawn /insert">
      <dispatch xdbc="true">$0</dispatch>
<match-path>
```

17.8.2 error

エラーを生成して評価を終了します。

error ルールは、リライタ全体を終了し、リクエストハンドラにエラーを返します。このエラーは、リクエストハンドラによって処理され、エラーハンドラが存在する場合は、そのエラーハンドラに渡されます。

code（省略可能）メッセージデータは、属性として提供されます。

属性：

名前	型	必須	目的
@code	string	はい	エラーコードを指定します。
@data1	string	いいえ	エラーメッセージの 1 番目の部分。
@data2	string	いいえ	エラーメッセージの 2 番目の部分。
@data3	string	いいえ	エラーメッセージの 3 番目の部分。
@data4	string	いいえ	エラーメッセージの 4 番目の部分。
@data5	string	いいえ	エラーメッセージの 5 番目の部分。

子コンテンツ：

なし

子要素：

なし

子コンテキストの変更：なし

例：

```
<error code="XDMP-BAD" data1="this" data2="that"/>
```

17.9　シンプルなリライタの例

シンプルなリライタの例をいくつか示します。

プレフィックス /dir を削除してリクエストをリダイレクトします。

```
<rewriter xmlns="http://marklogic.com/xdmp/rewriter">
   <match-path matches="^/dir(/.+)">
      <dispatch>$1</dispatch>
   </match-path>
</rewriter>
```

GET および PUT リクエストの場合にのみ、クエリパラメータ path が /admin に完全一致すると、/private/admin.xqy にリダイレクトし、それ以外の場合は、パラメータの値を使用してリダイレクトします。

path クエリパラメータがない場合は、リクエストを変更しません。

```
<rewriter xmlns="http://marklogic.com/xdmp/rewriter">
   <match-method any-of="GET PUT">
      <!-- match by name/value -->
      <match-query-param name="path" value="/admin">
         <dispatch>/private/admin.xqy</dispatch>:
      </match-query-param>
      <!-- match by name use value -->
      <match-query-param name="path">
         <dispatch>$0</dispatch>:
      </match-query-param>
   </match-method>
</rewriter>
```

パラメータ data が URI に存在する場合、データベースを UserData に設定します。クエリパラメータ module が存在する場合、Modules データベースを UserModule に設定します。パスの先頭が /users/ で、末尾が /version<versionID> の場合は、次のパスコンポーネント（$1）を抽出して /app に付加し、クエリパラメータ version を versionID に設定して追加します。

```
<rewriter xmlns="http://marklogic.com/xdmp/rewriter">
   <match-query-param name="data">
      <set-database>UserData</set-database>
   </match-query-param>
   <match-query-param name="module">
      <set-modules-database>UserModule</set-modules-database>
   </match-query-param>
   <match-path match="^/users/([^/]+)/version(.+)%">
      <set-path>/app/$1</set-path>
```

```
      <add-query-param name="version">$2</add-query-param>
   </match-path>
   <dispatch/>
</rewriter>
```

名前とデフォルトユーザーでユーザーをマッチし、クエリパラメータを設定または上書きします。

```
<rewriter xmlns="http://marklogic.com/xdmp/rewriter">
   <set-query-param name="default">
        default-user no match
   </set-query-param>
   <match-user name="admin">
       <add-query-param name="user">admin
matched</add-query-param>
   </match-user>
   <match-user name="infostudio-admin">
       <add-query-param name="user">
           infostudio-admin matced
       </add-query-param>
   </match-user>
   <match-user default-user="true">
      <set-query-param name="default">
         default-user matched
      </set-query-param>
   </match-user>
   <dispatch>/myapp.xqy</dispatch>
</rewriter>
```

マッチする Cookie。このプロパティは Cookie HTTP ヘッダ構造をパースするため、信頼性の高いマッチを実行できます。この例では、SESSIONID Cookie を使用して、現在のトランザクションを条件付きで設定します。

```
<rewriter xmlns="http://marklogic.com/xdmp/rewriter">
   <match-cookie name="SESSIONID">
      <set-transaction>$0</set-transaction>
   </match-cookie>
</rewriter>
```

ローカルにスコープ設定されたユーザー定義変数。ユーザー変数「test」に初期値を設定します。パスの先頭が /test/ で、2 つ以上のパスコンポーネントがパスに含まれる場合は、1 番目のマッチングパスを「test」変数に再設定し、2 番目のマッチングパスにクエリパラメータ「var1」を追加します。また、ユーザーのロールに「admin-builtins」または「app-builder」が含まれる場合は、パス「/admin/secret.xqy」に書き換えます。それ以外の場合は、クエリパラメータ「var2」を、「test」ユーザー変数の値とともに追加し、「/default.xqy」に書き換えます。

スコープ設定された属性を true から false に変更（または削除）すると、その条件内のすべての変更は、/admin/secret.xqy への最終的なディスパッチが到達しない場合に破棄され、「test」変数の初期値はそのままになり、「var1」クエリパラメータは追加されず、/default.xqy にディスパッチされます。

```
<rewriter xmlns="http://marklogic.com/xdmp/rewriter" >
   <set-var name="test">initial</set-var>
   <match-path matches="^/test/(\w+)/(\w+).*"
scoped="true">
      <set-var name="test">$1</set-var>
      <set-query-param name="var1">$2</set-query-param>
      <match-role any-of="admin-builtins app-builder">
         <dispatch>/admin/secret.xqy</dispatch>
      </match-role>
   </match-path>
   <add-query-param name="var2">$test</add-query-param>
   <dispatch>/default.xqy</dispatch>
</rewriter>
```

18.0 テンプレートに基づく抽出（TDE）

テンプレートに基づく抽出（TDE）では、ドキュメントデータに対してリレーションレンズを定義できるため、SQL またはオプティック API を使用してデータの各部分をクエリできます。テンプレートを使用すると、ドキュメントのどの部分がビューの行を構成するかを指定できます。また、テンプレートを使用して、ドキュメントのどの値がトリプルインデックスのトリプルを構成するかを指定することでセマンティックレンズを定義することもできます。

TDE では、事前定義されたテンプレートに基づいて、読み込んだドキュメントから行およびトリプルを生成できます。このテンプレートは、次の内容を説明します。

- マッチさせる入力データ
- マッチしたデータに適用されるデータ変換
- インデックス付けされたデータに変換される最終的なデータ射影

TDE を使用すると、ドキュメント自体を変更することなく、ドキュメント内のデータに複数の方法でアクセスできます。リレーショナルレンズは、SQL に詳しいユーザーにデータへのアクセスを許可する場合や、SQL を使用して通信するツールでユーザーがレポートや視覚化を作成する場合に便利です。また、エンティティを結合してドキュメントの集約を実行するときにも役立ちます。セマンティックレンズは、SPARQL を使用してトリプルとして自然に表現およびクエリされるデータがドキュメントに含まれている場合に便利です。

TDE は、読み込みの際にインデックス付け中に適用され、次の目的を果たします。

- SQL/ リレーションのインデックス付け。TDE では、XML または JSON ドキュメントの部分を SQL 行にマッピングできます。TDE テンプレートインスタンスを使用すると、ユーザーは、さまざまな行を作成し、ドキュメントから抽出したデータを使用して行の各行がどのように構築されているかを記述できます。詳細については、『*SQL Data Modeling Guide*』の Creating Template Views を参照してください。
- カスタム組み込みトリプル抽出。TDE では、`sem:triple` スキーマに従わないトリプルをユーザーが読み込むことができます。ユーザーは、1 つのテンプレート内で多くのトリプル射影を定義できます。このテンプレートでは、各射影が、主語、述語、または目的語にマップされるドキュメントのさまざまな部分を指定します。詳細については、『*Semantics Developer's Guide*』の Using a Template to Identify Triples in a Document を参照してください。
- エンティティサービスデータモデル。詳細については、『*Entity Services Developer's Guide*』の Creating and Managing Models を参照してください。

TDE データは、オプティック API でも使用されます（「リレーショナル操作のためのオプティック API」（294 ページ）を参照）。

> **注：** テンプレートをスキーマデータベースに挿入するには、`tde-admin` ロールが必要です。

このセクションで説明する主なトピックは次のとおりです。

- TDE ドキュメントのセキュリティ
- テンプレートビュー要素
- テンプレートダイアレクトとデータ変換関数
- テンプレートの検証と挿入
- テンプレートと非準拠ドキュメント
- テンプレートの有効化と無効化
- テンプレートの削除

18.1 TDE ドキュメントのセキュリティ

テンプレートドキュメントに対する操作は、以下によって制御されます。

`http://marklogic.com/xdmp/tde` コレクション。これは、TDE テンプレートドキュメントが含まれる protected コレクションです。

`tde-admin` ロール。TDE protected コレクションにアクセスするために必要です。

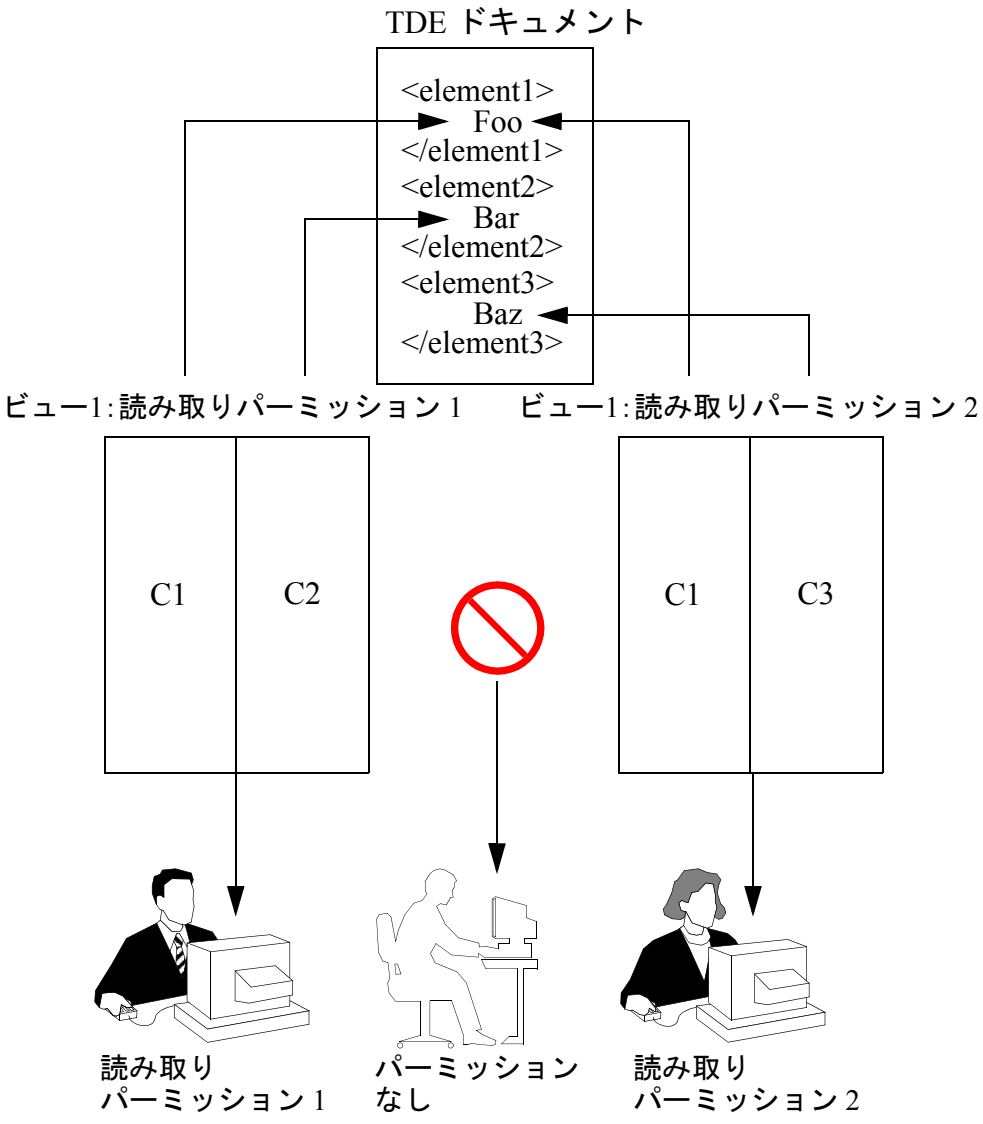
`tde-view` ロール。TDE protected コレクション内のドキュメントを表示するために必要です。ビューへのアクセスは、ビューを定義するテンプレートドキュメントに追加のパーミッションを設定することで詳細に制限できます。異なるパーミッションで読み込まれた複数のテンプレートで同一のビューを宣言できるため、ビューへのアクセスは、次のようにカラムレベルで制御する必要があります。

カラムレベルの読み取りパーミッションは暗黙的であり、テンプレートドキュメントに設定された読み取りパーミッションから得られるものです。カラムのパーミッションは、同一である必要はなく、OR で結合されます。1 つ以上の読み取りパーミッションがカラムに設定されたロールを持つユーザーには、そのカラムが表示されます。

ビューのカラムのどれにもパーミッションが設定されていないユーザーには、ビュー自体が表示されません。

例えば、下の図で説明します。

- テンプレートドキュメント `TD1` は、カラム `C1` および `C2` のあるビュー `View 1` を作成します。テンプレートドキュメント `TD1` は、Read Permission 1 で読み込まれました。
- テンプレートドキュメント `TD2` は、カラム `C1` および `C3` のあるビュー `View 1` を作成します。テンプレートドキュメント `TD2` は、Read Permission 2 で読み込まれました。
- `Permission 1` を持つユーザーは、クエリ時にカラム `C1` および `C2` にアクセスできます。
- `Permission 2` を持つユーザーは、クエリ時にカラム `C1` および `C3` にアクセスできます。
- `Permission 1` と `Permission 2` のいずれも持たないユーザーは、`View 1` にも、そのいずれのカラムにもアクセスできません。



このような設計では、次のようになります。

- ユーザーには、自分がアクセスできるテンプレート内で参照されているカラムが表示されます。
- ユーザーには、自分がアクセスできないテンプレート内で参照されている追加のカラムは表示されません。

TDE protected コレクション内のドキュメントが要素レベルのセキュリティを利用する場合は、ほとんどの状況で、保護されていない要素のみが抽出され、保護されている要素は無視されます。要素レベルのセキュリティの詳細については、『*Security Guide*』の Element Level Security を参照してください。

18.2 テンプレートビュー要素

テンプレートには、次の要素とその子要素が含まれます。

要素	説明
`description`	テンプレートの説明（省略可能）。

`collections`
`  collection`
`  collections-and`
`  collection`
	コレクションのスコープ（省略可能）。複数のコレクションのスコープを OR または AND で結合できます。「コレクション」（280 ページ）を参照してください。

`directories` `  directory`	ディレクトリのスコープ（省略可能）。複数のディレクトリのスコープを OR で結合できます。「ディレクトリ」（281 ページ）を参照してください。
`vars` `  var`	現在のコンテキストレベルで抽出された中間変数（省略可能）。「変数」（283 ページ）を参照してください。

`rows`
`  row`
`  schema-name`
`  view-name`
`  view-layout`
`  sparse`
`  identical`
`  columns`
`  column`
`  name`
`  scalar-type`
`  val`
`  nullable`
`  default`
`  invalid-values`
`  ignore`
`  reject`
`  reindexing`
`  hidden`
`  visible`
`  collation`
	これらの要素は、テンプレートビューで使用されます（『*SQL Data Modeling Guide*』の Creating Template Views を参照）。
	
	`rows` は、`row` の記述とマッピングのシーケンスです（『*SQL Data Modeling Guide*』の Row を参照）。
	
	`columns` は、`column` の記述とマッピングのシーケンスです（『*SQL Data Modeling Guide*』の Columns を参照）。
	
	scalar-type は、`val` の型です。「型キャスト処理」（287 ページ）を参照してください。

要素	説明

triples
  triple
  subject
  val
  invalid-values
  predicate
  val
  invalid-values
  object
  val
  invalid-values
	これらの要素は、トリプル抽出テンプレートで使用されます（『*Semantics Developer's Guide*』の Using a Template to Identify Triples in a Document を参照）。
	
	triples には、トリプル抽出記述のシーケンスが含まれます。各 triple 記述は、subject、predicate、および object のデータマッピングを定義します。
	
	抽出されるトリプルのグラフは、テンプレートでは指定できません。グラフは、組み込みトリプルに似たドキュメントのコレクションによって暗黙で定義されます。

templates template	サブテンプレートのシーケンス（省略可能）。詳細については、Creating Views from Multiple Templates および『*SQL Data Modeling Guide*』の Creating Views from Nested Templates を参照してください。
path-namespaces path-namespace	名前空間バインドのシーケンス（省略可能）。「path-namespaces」（281 ページ）を参照してください。
context	テンプレートのアクティブ化およびデータの抽出に使用されるルックアップノード。「コンテキスト」（282 ページ）を参照してください。
enabled	テンプレートが有効か（true）無効か（false）を指定するブール。デフォルト値は true です。

context、vars、および columns は、パス式を利用して XQuery 要素または JSON プロパティを識別します。パス式は、XPath に基づいています。『*XQuery and XSLT Reference Guide*』の XPath Quick Reference、および「XPath を使用した JSON ドキュメントのトラバーサル」（349 ページ）を参照してください。

18.2.1 コレクション

<collections> セクションは、テンプレートのスコープを特定のコレクション内のドキュメントのみに制限することを定義します。<collections> セクションは、次のシーケンスの最上位 OR です。

- テンプレートのスコープを特定のコレクションに設定する <collection>。
- AND で結合された <collection> のシーケンスが含まれる <collections-and>。

以下のコレクションの論理結合が可能です。

OR で結合されたコレクション：

```
<collections>
  <collection>A</collection>
  <collection>B</collection>
</collections>
```

AND で結合されたコレクション：

```
<collections>
  <collections-and>
    <collection>A</collection>
    <collection>B</collection>
  </collections-and>
</collections>
```

AND で結合されたコレクションの OR 結合：

```
<collections>
  <collection>A</collection>
  <collection>B</collection>
  <collections-and>
    <collection>C</collection>
    <collection>D</collection>
  </collections-and>
  <collections-and>
    <collection>E</collection>
    <collection>F</collection>
  </collections-and>
</collections>
```

18.2.2 ディレクトリ

<directories> セクションは、テンプレートのスコープを特定のディレクトリ内のドキュメントのみに制限することを定義します。<directories> セクションは、テンプレートのスコープを特定のディレクトリに設定する <directory> 要素のシーケンスの最上位 OR です。

18.2.3 path-namespaces

<path-namespaces> セクションは、1 つあるいは複数の <path-namespace> 要素の最上位です。各要素には、以下が含まれます。

- <prefix>：名前空間プレフィックス。
- <namespace-uri>：名前空間 URI。

例えば、パス名前空間バインドは、次のようにテンプレートで指定できます。

```
<path-namespaces>
  <path-namespace>
    <prefix>wb</prefix>
    <namespace-uri>http://marklogic.com/wb</namespace-uri>
  </path-namespace>
</path-namespaces>
```

名前空間プレフィックス定義は、対象データベースの設定内ではなく、テンプレートドキュメント内に格納されます。それ以外の場合は、テンプレートを使用する対象データベース設定がわかっていないと、テンプレートをコードにコンパイルできません。

18.2.4 コンテキスト

`context` タグは、テンプレートのアクティブ化およびデータの抽出に使用されるルックアップノードを定義します。`vars`、`rows`、または `triples` 内に出現するパス式は、親テンプレートの context 要素に相対的です。`context` は、XML/JSON ツリーを上下に移動してデータを収集するときに移動の基準となるアンカーを定義します。`context` 要素内では、インデックス付け可能なパス式はすべて有効であるため、述語を使用できます。サブテンプレートの `context` 要素は、親テンプレートの `context` 要素に相対的です。

例：

```
<context>/Employee</context>

<context>/MedlineCitation/Article</context>
```

単一行の抽出元になるノードがコンテキストで定義されることを理解することが重要です。ドキュメントから複数の行を抽出する場合は、コンテキストを、それらの行の親要素に設定する必要があります。例えば、次のような構造の「order」ドキュメントがあるとします。

```
<order>
  <order-num>10663</order-num>
  <order-date>2017-01-15</order-date>
  <items>
    <item>
      <product>SpeedPro Ultimate</product>
      <price>999</price>
      <quantity>1</quantity>
    </item>
    <item>
      <product>Ladies Racer Helmet</product>
      <price>115</price>
```

```
        <quantity>1</quantity>
      </item>
    </items>
  </order>
```

各 order ドキュメントには、1 つあるいは複数の <item> ノードが含まれます。<product>、<price>、および <quantity> の値を各 <item> ノードから抽出するビューテンプレートを作成する必要があります。/order のコンテキストとカラム値（items/item/product など）は、ドキュメント全体に対して単一行の抽出をトリガーします。そのため、これが機能するのは、ドキュメントの <item> ノードが 1 つのみである場合に限られます。すべての <item> ノードのコンテンツを複数行として抽出するには、コンテキストが /order/items/item である必要があります。この状況で、<order-num> も抽出する場合は、カラム値が ../../order-num になります。

注： context は、検索可能なパスでなければなりません。ワイルドカード（「*」など）を使用できますが、パフォーマンス上の理由から、その価値がパフォーマンスコストよりも大きい場合を除き、ワイルドカードは使用しないでください。ワイルドカードをコンテキストで使用するときは、コレクションまたはディレクトリのスコープ設定を使用することを推奨します。次のコンテキストは明示的に無効です。/ /. * /*

18.2.5 変数

変数は、データ変換に必要な中間データ射影であり、var 要素下で定義されます。カラム / トリプルへの最後の射影の前に複数の中間射影 / 変換が必要である場合、変数は、その変換セクション val 内で他の変数を参照できます。val コード内の式は、var が定義されている現在のテンプレートのコンテキスト要素に相対的です。val で使用可能な式の種類については、「テンプレートダイアレクトとデータ変換関数」（284 ページ）を参照してください。

例：

```
<context>/northwind/Orders/Order</context>

.......

<vars>
    <var>
      <name>OrderID</name>
      <val>./@OrderID</val>
    </var>
</vars>

.......
```

```
<column>
    <name>OrderID</name>
    <scalar-type>long</scalar-type>
    <val>$OrderID</val>
</column>
```

注： `var` の記述内には、変数の値を入力しません。変数の値は、`column` の記述内に入力してください。

18.3 テンプレートダイアレクトとデータ変換関数

テンプレートは、関数のサブセットのみが使用可能な制限付き機能を備えた XQuery のサブセットを使用してダイアレクトをサポートします。

テンプレートダイアレクトは、An XML Query Language 仕様の「Expressions」に記載されている、次の種類の式をサポートします。

- パス式
- シーケンス式
- 算術式
- 比較式
- 論理式
- 条件式
- SequenceType に対する式

ループ処理、FLWOR ステートメント、反復、XML コンストラクションなどの複雑な操作は、ダイアレクト内ではサポートされません。プロパティ軸 `property::` もサポートされません。

以降のセクションに、サポートされる XQuery 関数のリストがあります。

- 日付 / 時間関数
- 論理関数とデータ検証
- 文字列関数
- 型キャスト処理
- 算術関数
- その他の関数

注： テンプレートは XQuery 関数のみをサポートします。JavaScript 関数はサポートされません。

18.3.1 日付 / 時間関数

- fn:adjust-date-to-timezone
- fn:adjust-dateTime-to-timezone
- fn:adjust-time-to-timezone
- fn:month-from-date
- fn:month-from-dateTime
- fn:months-from-duration
- fn:seconds-from-dateTime
- fn:seconds-from-duration
- fn:seconds-from-time
- fn:minutes-from-dateTime
- fn:minutes-from-duration
- fn:minutes-from-time
- fn:timezone-from-date
- fn:timezone-from-dateTime
- fn:timezone-from-time
- fn:year-from-date
- fn:year-from-dateTime
- fn:years-from-duration
- fn:day-from-date
- fn:day-from-dateTime
- fn:days-from-duration
- fn:format-date
- fn:format-dateTime
- fn:format-time
- fn:hours-from-dateTime
- fn:hours-from-duration
- fn:hours-from-time
- xdmp:dayname-from-date
- xdmp:quarter-from-date
- xdmp:week-from-date
- xdmp:weekday-from-date
- xdmp:yearday-from-date
- sql:dateadd

- sql:datediff
- sql:datepart
- sql:day
- sql:seconds
- sql:dayname
- sql:timestampadd
- sql:hours
- sql:timestampdiff
- sql:minutes
- sql:week
- sql:month
- sql:weekday
- sql:monthname
- sql:year
- sql:quarter
- sql:yearday
- xdmp:parse-dateTime
- xdmp:parse-yymmdd

18.3.2 論理関数とデータ検証

- fn:boolean
- fn:empty
- fn:exists
- fn:false
- fn:not
- fn:true

18.3.3 文字列関数

- fn:codepoint-equal
- fn:codepoints-to-string
- fn:compare
- fn:concat
- fn:contains
- fn:encode-for-uri

- fn:ends-with
- fn:escape-html-uri
- fn:escape-uri
- fn:format-number
- fn:insert-before
- fn:iri-to-uri
- fn:last
- fn:lower-case
- fn:matches
- fn:normalize-space
- fn:normalize-unicode
- fn:position
- fn:remove
- fn:replace
- fn:reverse
- fn:starts-with
- fn:string-join
- fn:string-length
- fn:string-to-codepoints
- fn:subsequence
- fn:substring
- fn:substring-after
- fn:substring-before
- fn:tokenize
- fn:translate
- fn:upper-case

18.3.4 型キャスト処理

- number
- string
- decimal
- integer
- long
- int

- short
- byte
- float
- double
- boolean
- date
- time
- dateTime
- gDay
- gMonth
- gYear
- gYearMonth
- gMonthDay
- duration
- dayTimeDuration
- yearMonthDuration
- castable-as
- anyURI
- IRI (Internationalized Resource Identifier)

18.3.5 算術関数

- fn:abs
- fn:round
- fn:ceiling
- fn:round-half-to-even
- fn:floor

およびすべての数学（http://marklogic.com/xdmp/math namespace）ビルトイン関数（variance や stddev などの集計関数を除く）

18.3.6 その他の関数

- xdmp:node-uri
- xdmp:node-kind
- xdmp:path
- xdmp:type

- xdmp:node-metadata-value
- xdmp:node-metadata
- sem:uuid
- sem:uuid-string
- sem:bnode
- sem:datatype
- sem:sameTerm
- sem:lang
- sem:iri-to-QName
- sem:iri
- sem:QName-to-iri
- sem:unknown
- sem:unknown-datatype
- sem:invalid
- sem:invalid-datatype
- sem:typed-literal
- cts:point
- fn:head
- fn:tail
- fn:base-uri
- fn:document-uri
- fn:lang
- fn:local-name
- fn:name
- fn:namespace-uri
- fn:node-name
- fn:number
- fn:root
- fn:min
- fn:max
- fn:sum
- fn:count
- fn:avg

18.4 テンプレートの検証と挿入

注：テンプレートをスキーマデータベースに挿入するには、tde-admin ロールが必要です。

警告 最適なパフォーマンスを得るため、デフォルトの Schemas データベースを使用するようにコンテンツデータベースを設定するのではなく、テンプレートドキュメント用に専用のスキーマデータベースを作成することを推奨します。TDE で抽出するドキュメントを保持するために複数のコンテンツデータベースを作成する場合は、それぞれのコンテンツデータベースに専用のスキーマデータベースを設定してください。そうしないと、想定外のインデックス付け動作がコンテンツデータベースで発生することがあります。

スキーマデータベースにビューを挿入する前に、必ずテンプレートを検証してください。ビューを検証するには、次のように tde:validate 関数を使用します。

```
let $viewTemplate :=
<template xmlns="http://marklogic.com/xdmp/tde">
  .....
</template>

return tde:validate($viewTemplate)
```

有効なテンプレートは、次のような結果を返します。

```
<map:map xmlns:map="http://marklogic.com/xdmp/map"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <map:entry key="valid">
     <map:value xsi:type="xs:boolean">true</map:value>
  </map:entry>
</map:map>
```

注：テンプレートを検証するために xdmp:validate を使用しないでください。この関数は、一部の検証ステップを実行しない可能性があります。

ビューテンプレートが有効であることを確認したら、ドキュメントデータを保持しているコンテンツデータベースが使用するスキーマデータベースにビューテンプレートを挿入できます。ビューテンプレートを挿入するには、任意の方法を使用してドキュメントをデータベースに挿入できますが、テンプレートドキュメントは http://marklogic.com/xdmp/tde コレクションに挿入する必要があります。

tde:template-insert 関数を使用すると便利です。この関数は、テンプレートを検証し、デフォルトのパーミッションでスキーマデータベース内の tde コレクションにテンプレートドキュメントを挿入し（コンテンツデータベースで実行されている場合）、データベースの再インデックス付けをトリガーします。

注：テンプレートが挿入されると、テンプレートの影響を受けるドキュメントフラグメントのみが再インデックス付けされます。

例えば、ビューテンプレートを定義および挿入するには、次のように入力します。

```
xquery version "1.0-ml";

import module namespace tde = "http://marklogic.com/
xdmp/tde"
        at "/MarkLogic/tde.xqy";

let $ClinicalView :=
<template xmlns="http://marklogic.com/xdmp/tde">
  <description>populates patients' data</description>
  <context>/Citation/Article</context>
  <rows>
    <row>
      <schema-name>Medical2</schema-name>
      <view-name>Publications</view-name>
      <columns>
        <column>
          <name>ID</name>
          <scalar-type>long</scalar-type>
          <val>../ID</val>
        </column>
        <column>
          <name>ISSN</name>
          <scalar-type>string</scalar-type>
          <val>Journal/ISSN</val>
        </column>
      </columns>
    </row>
  </rows>
</template>

return tde:template-insert("/Template.xml", $ClinicalView)
```

別の挿入操作を使用する場合は、コンテンツデータベースが使用するスキーマデータベースの `http://marklogic.com/xdmp/tde` コレクションにテンプレートドキュメントを明示的に挿入する必要があります。例：

```
return xdmp:document-insert(
     "/Template.xml",
      $ClinicalView,
      (),
      "http://marklogic.com/xdmp/tde")
```

18.5 テンプレートと非準拠ドキュメント

コンテンツデータベース用の TDE テンプレートを挿入した後に、テンプレートに準拠しないドキュメントを挿入しようとすると、エラーが発生することがあります。

「準拠しない」とは、いずれかのパスに `price` 要素が必要であり、カラムを null にすることができないことがテンプレートで指定されており、デフォルト値がないという意味である場合があります。ただし、挿入したドキュメントのパスに `price` 要素がないか、ドキュメントに `price` があるもののカラムの型にキャストできない場合もあります。

ドキュメントがすでにデータベース内に存在し、テンプレートを追加する場合に、非準拠ドキュメントを削除するつもりはないが、その存在を知っておくようにすることがあります。ログレベルを `debug` に設定した場合、テンプレートを追加したが、既存ドキュメントのいくつかが非準拠であるという状況では、インデックス付けされないドキュメントごとにエラーがエラーログに記録されます。ログレベルの設定方法の詳細については、『Administrator's Guide』の Understanding the Log Levels を参照してください。

テンプレートがすでに存在する場合に非準拠ドキュメントを挿入しようとすると、考えられる結果が 2 つあります。

- 挿入が失敗してエラーが発生する。
- 挿入は成功するが、`price` カラムがない行はスキップされる（インデックスに追加されない）。

どちらの結果になるかは、テンプレートで `invalid-values` を `reject`（非準拠ドキュメントを却下し、エラーをスローする）または `ignore`（ドキュメントの挿入を許可し、その行をインデックス付けで無視する）に設定して制御できます。

18.6 テンプレートの有効化と無効化

テンプレートを有効化または無効化するには、テンプレートの `<enabled>` フラグを変更します。`<enabled>` フラグを `true` に設定するとテンプレートが有効になり、`false` に設定すると無効になります。

例えば、テンプレートを無効にするには、次のように `<enabled>` フラグを `false` に設定します。

```
<template xmlns="http://marklogic.com/xdmp/tde">
    <context>/foo/bar</context>
    <enabled>false</enabled>
        ...
</template>
```

テンプレートが有効または無効になるたびに、再インデックス付けが自動的に開始されます。

18.7 テンプレートの削除

テンプレートを無効にした後、無効にしたテンプレートに関連する再インデックス付けが確実に完了するために十分な時間が経過したら、テンプレートドキュメントを安全に削除できます。

テンプレートを誤って削除した場合は、次のようにして修正できます。

1. 無効な状態でテンプレートを再挿入します。

2. 新しいテンプレートで、同じテンプレートドキュメント URI を再利用します。

3. データベースを手動で再インデックス付けします。

19.0 リレーショナル操作のためのオプティック API

MarkLogic オプティック API を使用すると、インデックス付けされた値およびドキュメントでリレーショナル操作を実行できます。オプティック API は単体の API ではなく、XQuery、JavaScript、および Java 言語内で公開されている API の集まりです。

オプティック API は、レンジインデックス内、トリプルインデックス内、テンプレートで抽出された行内のいずれにある場合でも、インデックス付きの値をすべて読み取ることができます。抽出テンプレート（『*SQL Data Modeling Guide*』の Creating Template Views で説明する、テンプレートビューを作成するために使用されるものなど）を使用すると、シンプルかつ強力な方法でドキュメントに対してリレーショナルレンズを指定できます。また、ドキュメントデータの各部分が SQL 経由でアクセス可能になります。オプティックを使用すると、結合や集約など、行に対する同じリレーショナル操作にアクセスできます。オプティック API により、ドキュメントから射影された行、行内のカラムとして結合されたドキュメント、および動的なドキュメント構造をドキュメント検索で一致として検出することもできます。これらはすべて、データベース内で効率的に実行され、アプリケーションからプログラムによってアクセスされます。

オプティック API を使用すると、データをそのまま使用でき、JavaScript または XQuery のシンタックスを使用して MarkLogic ドキュメントと検索機能を利用できるようになり、データの構造に関係なく、一般的な SQL の概念が活用されます。SQL とは異なり、オプティックは、アプリケーションの構築と MarkLogic NoSQL 機能へのフルアクセスに適しています。オプティックは、一般的なアプリケーション言語に統合されています。そのため、データを更新し、エンドユーザーへのプレゼンテーション用に結果を処理する広範なアプリケーションのコンテキスト内でクエリを実行できます。

オプティック API は、次の操作をサポートします。

- 結合－頻繁に更新されるドキュメントや多数の関係があるドキュメントを、非正規化された書き込みではなく、宣言型クエリに結合します。
- グループ化－多数のドキュメントに対して集約プロパティをまとめます。
- ドキュメント内で繰り返される構造に対する完全一致。
- トリプル結合－セマンティックトリプルを組み込んで、行データをエンリッチしたり、ドキュメントと行をリンクしたりします。
- ドキュメント結合－ソースドキュメント全体を返して、行データにコンテキストを提供します。
- ドキュメントクエリ－リレーショナルフィルタリングに加え、リッチな全文検索を実行して行に制約を加えます。

SQL および SPARQL インターフェイスと同様に、オプティック API を使用して標準操作（`where`、`groupBy`、`orderBy`、`union`、`join` など）からクエリを構築できますが、JavaScript および XQuery 関数の呼び出しを介して操作を表現します。オプティック API ではプログラミング言語の環境で作業できるため、変数と関数を利用して、プラン構成のモジュラー化や、文字列連結でクエリを作成する場合のパースエラーやインジェクション攻撃などの回避などのメリットが得られます。

> **注：** SQL とは異なり、オプティックではカラム順序が不定です。注意すべき例外として、優先度を指定する `orderby` のソート順キーと `groupby` のグループ化キーがあります。

オプティック Java クライアント API も用意されています。『*Developing Applications With the Java Client API*』ガイドの Optic Java API for Relational Operations を参照してください。

この章は、次のメインセクションで構成されています。

- JavaScript と XQuery のオプティック API の相違点
- オプティックパイプライン内のオブジェクト
- データアクセス関数
- オプティッククエリの種類
- オプティック出力の処理
- カラム値を処理するための式関数
- ブール、数値、および文字列演算子に相当する関数
- ノードコンストラクタ関数
- ベストプラクティスとパフォーマンスに関する考慮事項
- オプティック実行プラン
- プランのパラメータ化
- シリアライズされたオプティッククエリのエクスポートとインポート

19.1 JavaScript と XQuery のオプティック API の相違点

XQuery オプティック API と JavaScript オプティック API の機能は同等です。それぞれの言語規則の特徴や慣例に適合していますが、それ以外の点では両方とも、可能な限りの一貫性があり、パフォーマンスの特徴は同じです。自分のスキルやプログラミング環境に最適な言語を使用してください。

次の表では、オプティック API の JavaScript バージョンと XQuery バージョンの主要な相違点を説明しています。

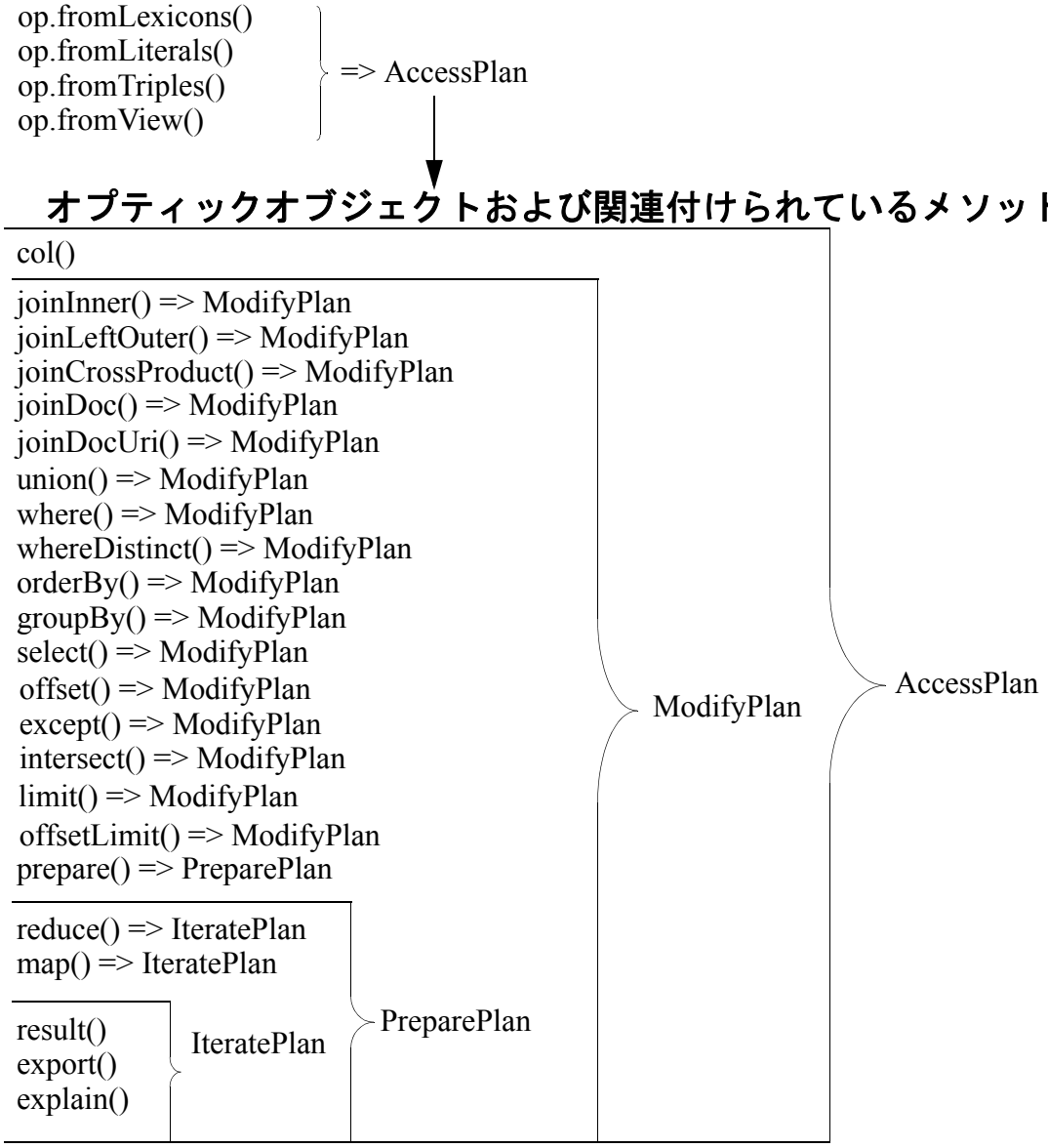
特性	JavaScript	XQuery

プロキシ関数用の名前空間	ネストされた名前空間（`op.fn.min` など）	次のテンプレートに準拠する、別の名前空間内のモジュール（プレフィックスの場合は `ofn:min` など）。
		
		`import module namespace`
		`  ofn="http://marklogic.com/optic/`
		`  expression/fn"`
		`  at "/MarkLogic/optic/optic-fn.xqy";`
		
		詳細については、「式関数に必要な XQuery ライブラリ」（337 ページ）を参照してください。

スムーズなオブジェクトチェーン	オブジェクトを返すメソッド	関数は、状態オブジェクトを 1 番目のパラメータとして取り、状態オブジェクトを返します。これで、XQuery => チェーン演算子の使用が可能になります。これらのブラックボックスオブジェクトは、マップ形式で構築されるプランの状態を保持します。これらの状態オブジェクトは将来のリリースで変更される可能性があるため、変更、シリアル化、永続化のいずれも行わないでください。連結された関数は、既存のマップを変更するのではなく、常に新しいマップを作成します。
命名規則	キャメルケース（camelCase）	ハイフン区切りの命名規則。元の関数がキャメルケースであるプロキシ関数は例外です（`fn:current-dateTime` 関数など）。
バインドされていないパラメータ	許可	単一のシーケンスパラメータとしてサポートされます。現時点の唯一の例は、`fn:concat` および `sem:coalesce` のプロキシ関数です。
結果の型	オブジェクトのシーケンスを返します。配列のシーケンスを返すオプションがあります。	`sql:rows` のマップを返します。ヘッダと行で構成される配列を返すオプションがあります。

19.2 オプティックパイプライン内のオブジェクト

以下の図は、オプティックパイプラインのメソッドで入出力として使用されるオブジェクトを示しています。



オプティッククエリは、行セットに対するリレーショナル操作のシーケンスを適用するパイプラインを作成します。オプティッククエリで使用される関数とメソッドの基本的な特性を次に示します。

- すべてのデータアクセス関数（あらゆる `from*` 関数）は、`AccessPlan` オブジェクトの形式で出力行セットを生成します。
- すべての修飾子操作（`ModifyPlan.prototype.where` など）は、入力行セットを使用し、`ModifyPlan` オブジェクトの形式で出力行セットを生成します。

- すべての構成子操作（ModifyPlan.prototype.joinInner など）は、2 つの入力行セットを使用し、ModifyPlan オブジェクトの形式で 1 つの出力行セットを生成します。
- 最後の出力行セットは、プランの結果です。
- 操作の順序に対する制約は、パイプラインがアクセサ操作で始まることだけです。例えば、次のように指定できます。
    - groupBy の前に select を指定し、2 つのカラムに数式を適用して sum 関数の入力を指定します。
    - groupBy の後に select を指定し、2 つの sum 集約から出力されたカラムで数式を適用します。

ビュー内の行から特定のカラムを選択して特定の順序で出力するシンプルな例を次に示します。このクエリで作成されるパイプラインを以下に示します。

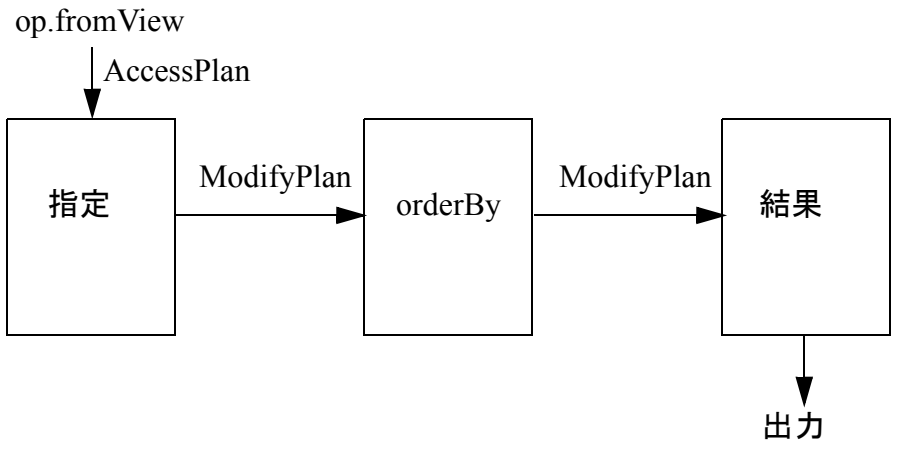
```
const op = require('/MarkLogic/optic');

op.fromView('main', 'employees')
   .select(['EmployeeID', 'FirstName', 'LastName'])
   .orderBy('EmployeeID')
   .result();
```

1. op.fromView 関数は、すべての API メソッドで使用できる AccessPlan オブジェクトを出力します。

2. AccessPlan.prototype.select メソッドは、ModifyPlan オブジェクトを出力します。

3. ModifyPlan.prototype.orderBy メソッドは、もう 1 つの ModifyPlan オブジェクトを出力します。

4. ModifyPlan.prototype.result メソッドは、ModifyPlan オブジェクトを使用し、プランを実行します。



次の例では、従業員ごとに経費の合計を計算し、その結果を従業員番号の順で返します。

```
const op = require('/MarkLogic/optic');
const employees = op.fromView('main', 'employees');
const expenses = op.fromView('main', 'expenses');

const Plan =
employees.joinInner(expenses, op.on(employees.col('EmployeeID'),
                                    expenses.col('EmployeeID')))
   .groupBy(employees.col('EmployeeID'), ['FirstName',
     'LastName',
            op.sum('totalexpenses', expenses.col('Amount'))])
   .orderBy('EmployeeID')
Plan.result();
```

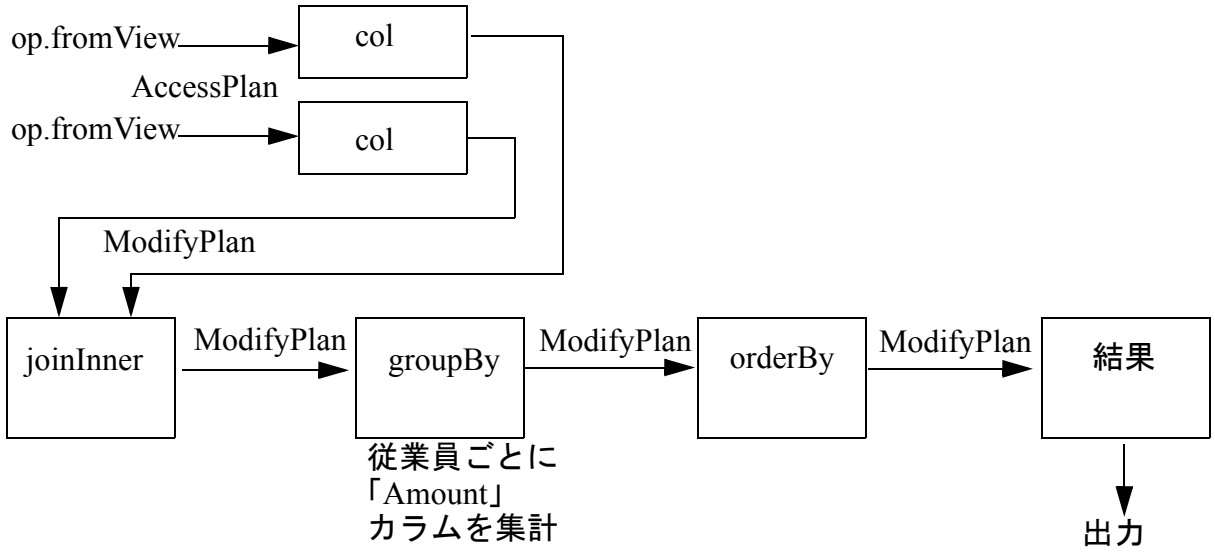
注： `.select` がないことは、SQL の `SELECT *` と同等であるため、ビュー内のすべてのカラムが取得されます。

1. `op.fromView` 関数は、`AccessPlan` オブジェクトを出力します。このオブジェクトは、`op.on` 関数および `AccessPlan.prototype.col` メソッドによって使用され、`ModifyPlan.prototype.joinInner` メソッドに対し、両方のビューからの行セットを結合することが指示されます。結合された行セットは、`ModifyPlan` オブジェクトの形式で単一の行セットとして出力されます。

2. `ModifyPlan.prototype.groupBy` メソッドは、従業員ごとに経費の合計を計算し、結果を 1 行にまとめます。

3. `ModifyPlan.prototype.orderBy` メソッドは、結果をソートし、もう 1 つの `ModifyPlan` オブジェクトを出力します。

4. `ModifyPlan.prototype.result` メソッドは、`ModifyPlan` オブジェクトを使用し、プランを実行します。



19.3 データアクセス関数

次の関数は、行、トリプル、およびレキシコンとしてインデックス付けされたデータと、プログラム内で作成されたリテラル行セットにアクセスできます。

JavaScript	XQuery
`op.fromView`	`op:from-view`
`op.fromTriples`	`op:from-triples`
`op.fromLiterals`	`op:from-literals`
`op.fromLexicons`	`op:from-lexicons`

`op.fromView` 関数は、テンプレートビューによって作成されたインデックスにアクセスします（『*SQL Data Modeling Guide*』の Creating Template Views を参照）。

`op.fromTriples` 関数は、セマンティックトリプルインデックスにアクセスし、行およびカラムとして抽象化します。ただし、RDF グラフの行のカラムはさまざまなデータ型を持つ可能性があり、結合に影響を及ぼすことがあります。

`op.fromLexicons` 関数は、レンジインデックス、URI レキシコン、およびコレクションレキシコン上のカラムを使用してビューを動的に作成します。多くの場合、レキシコンは、ビュー内でインデックス付けされたデータをエンリッチするために結合されます。アプリケーションでレンジインデックスが定義済みである場合、または URI やコレクション情報がクエリに必要な場合は、オプティックからレキシコンにアクセスできると便利です。

`op.fromLiterals` 関数は、SQL VALUES または SPARQL VALUES ステートメントの結果と同様のリテラル行セットを作成します。これにより、既存のビューと結合する代替のカラムを提供できます。

次のセクションでは、さまざまなデータアクセス関数の例を示します。

- fromView の例
- fromTriples の例
- fromLexicons の例
- fromLiterals の例

19.3.1 fromView の例

`fromView` を使用したクエリは、ドキュメントで公開されているインデックス付けされた行を取得します。このセクションの例は、『*SQL Data Modeling Guide*』の章 SQL on MarkLogic Server Quick Start で説明するドキュメントとテンプレートビューに基づいています。

すべての従業員を ID 番号順にリストアップします。

JavaScript :

```
const op = require('/MarkLogic/optic');

op.fromView('main', 'employees')
   .select(['EmployeeID', 'FirstName', 'LastName'])
   .orderBy('EmployeeID')
   .result();
```

XQuery :

```
xquery version "1.0-ml";

import module namespace op="http://marklogic.com/optic"
     at "/MarkLogic/optic.xqy";

op:from-view("main", "employees")
   => op:select(("EmployeeID", "FirstName", "LastName"))
   => op:order-by("EmployeeID")
   => op:result()
```

オプティックを使用すると、行をフィルタリングして、関心がある特定のデータを抽出できます。例えば、次のクエリは、3 番の従業員の ID と名前を返します。

JavaScript :

```
const op = require('/MarkLogic/optic');

op.fromView('main', 'employees')
   .where(op.eq(op.col('EmployeeID'), 3))
   .select(['EmployeeID', 'FirstName', 'LastName'])
   .orderBy('EmployeeID')
   .result();
```

XQuery :

```
xquery version "1.0-ml";

import module namespace op="http://marklogic.com/optic"
     at "/MarkLogic/optic.xqy";

op:from-view("main", "employees")
   => op:where(op:eq(op:col("EmployeeID"), 3))
   => op:select(("EmployeeID", "FirstName", "LastName"))
   => op:order-by("EmployeeID")
   => op:result()
```

次のクエリは、各従業員のすべての経費と経費カテゴリを返しますが、結果は従業員番号の順に返します。経費レポートのみに含まれる情報と従業員レコードのみに含まれる情報があるため、`EmployeeID` での行結合を使用して、両方のドキュメントの集まりからデータを取り出し、統合された 1 つの行セットを生成します。

JavaScript :

```
const op = require('/MarkLogic/optic');

const employees = op.fromView('main', 'employees');
const expenses = op.fromView('main', 'expenses');

const Plan =
   employees.joinInner(expenses, op.on(employees.col
                      ('EmployeeID'), expenses.col
                      ('EmployeeID')))
   .select([employees.col('EmployeeID'), 'FirstName',
     'LastName','Category', 'Amount'])
   .orderBy(employees.col('EmployeeID'))
Plan.result();
```

XQuery :

```
xquery version "1.0-ml";

import module namespace op="http://marklogic.com/optic"
     at "/MarkLogic/optic.xqy";

let $employees := op:from-view("main", "employees")
let $expenses  := op:from-view("main", "expenses")
```

```
return $employees
   => op:join-inner($expenses, op:on(
          op:view-col("employees", "EmployeeID"),
          op:view-col("expenses",   "EmployeeID")))
   => op:select((op:view-col("employees", "EmployeeID"),
                 "FirstName", "LastName", "Category",
                 "Amount"))
   => op:order-by(op:view-col("employees", "EmployeeID"))
   => op:result()
```

許可されている上限を超える従業員の経費を見つけます。この例の `where` 操作は、先行するすべての行に適用され、オプティックのチェーンパイプラインの特性を示しています。

JavaScript :

```
const op = require('/MarkLogic/optic');

const employees = op.fromView('main', 'employees');
const expenses = op.fromView('main', 'expenses');
const expenselimit = op.fromView('main', 'expenselimit');

const Plan =
   employees.joinInner(expenses, op.on(employees.col
     ('EmployeeID'),expenses.col('EmployeeID')))
   .joinInner(expenselimit, op.on(expenses.col
     ('Category'),expenselimit.col('Category')))
   .where(op.gt(expenses.col('Amount'),
      expenselimit.col('Limit')))
   .select([employees.col('EmployeeID'), 'FirstName',
     'LastName',
           expenses.col('Category'), expenses.col('Amount'),
           expenselimit.col('Limit') ])
   .orderBy(employees.col('EmployeeID'))
Plan.result();
```

XQuery :

```
xquery version "1.0-ml";

import module namespace op="http://marklogic.com/optic"
     at "/MarkLogic/optic.xqy";

let $employees := op:from-view("main", "employees")
let $expenses  := op:from-view("main", "expenses")
let $expenselimit  := op:from-view("main", "expenselimit")

return $employees
   => op:join-inner($expenses, op:on(
               op:view-col("employees", "EmployeeID"),
               op:view-col("expenses", "EmployeeID")))
   => op:join-inner($expenselimit, op:on(
               op:view-col("expenses", "Category"),
               op:view-col("expenselimit", "Category")))
   => op:where(op:gt(op:view-col("expenses", "Amount"),
               op:view-col("expenselimit", "Limit")))
   => op:select((op:view-col("employees", "EmployeeID"),
               "FirstName", "LastName",
               op:view-col("expenses", "Category"),
               op:view-col("expenses", "Amount"),
               op:view-col("expenselimit", "Limit")))
   => op:order-by(op:view-col("employees", "EmployeeID"))
   => op:result()
```

19.3.2 fromTriples の例

次の例は、ブルックリンで生まれた人々のリストを、person と name という 2 つのカラムから成るテーブルとして返します。これは、『*Semantics Developer's Guide*』の Loading Triples で説明するデータセット例に対して実行されます。

JavaScript :

```
const op = require('/MarkLogic/optic');
// prefixer is a factory for sem:iri() constructors in a
   namespace
const resource = op.prefixer('http://dbpedia.org/resource/');
const foaf   = op.prefixer('http://xmlns.com/foaf/0.1/');
const onto = op.prefixer('http://dbpedia.org/ontology/');

const person = op.col('person');

const Plan =
    op.fromTriples([
        op.pattern(person, onto('birthPlace'),
          resource('Brooklyn')),
        op.pattern(person, foaf("name"), op.col("name"))
    ])
Plan.result();
```

XQuery :

```
xquery version "1.0-ml";

import module namespace op="http://marklogic.com/optic"
     at "/MarkLogic/optic.xqy";

let $resource := op:prefixer("http://dbpedia.org/resource/")
let $foaf     := op:prefixer("http://xmlns.com/foaf/0.1/")
let $onto     := op:prefixer("http://dbpedia.org/ontology/")
let $person   := op:col("person")

return op:from-triples((
   op:pattern($person, $onto("birthPlace"),
     $resource("Brooklyn")),
   op:pattern($person, $foaf("name"), op:col("name"))))
   => op:result()
```

19.3.3 fromLexicons の例

fromLexicons 関数は、アプリケーション内の他の場所で使用するレンジインデックスがすでに定義されている場合に役立つことがあります。このデータアクセス関数を使用すると、クエリパイプラインのもう 1 つのデータソースとしてレキシコンを利用できます。

このセクションの例は、『*SQL Data Modeling Guide*』の Load the Data で説明するドキュメントを操作します。

> **注：** fromLexicons 関数は、ビュー内のカラム名ではなく、レンジインデックス名に対してクエリします。例えば、employee ドキュメントの場合は、EmployeeID に対してクエリするのではなく、レンジインデックス、命名済み ID を作成し、ID に対してクエリします。

まず、データを保持しているデータベースで、次の要素に対して要素レンジインデックスを作成します。ID、Position、FirstName、および LastName。レンジインデックスの作成方法の詳細については、『*Administrator's Guide*』の Defining Element Range Indexes を参照してください。

次の例は、各従業員の EmployeeID を返します。各カラム名の前にはテキスト myview が付加されます。

JavaScript :

```
const op = require('/MarkLogic/optic');

const Plan =
op.fromLexicons(
      {EmployeeID: cts.elementReference(xs.QName('ID'))});
Plan.result();
```

XQuery :

```
xquery version "1.0-ml";

import module namespace op="http://marklogic.com/optic"
     at "/MarkLogic/optic.xqy";

op:from-lexicons(
   map:entry(
      "EmployeeID", cts:element-reference(xs:QName("ID"))),
   "myview")
=> op:result()
```

次の例は、各従業員の EmployeeID、FirstName、LastName、およびデータを保持しているドキュメントの URI を返します。

JavaScript :

```
const op = require('/MarkLogic/optic');

const Plan =
op.fromLexicons({
            EmployeeID: cts.elementReference
            (xs.QName('ID')),
            FirstName: cts.elementReference
            (xs.QName('FirstName')),
            LastName: cts.elementReference
            (xs.QName('LastName')),
            URI: cts.uriReference()});
Plan.result();
```

XQuery :

```
xquery version "1.0-ml";

import module namespace op="http://marklogic.com/optic"
     at "/MarkLogic/optic.xqy";

op:from-lexicons(
map:entry("EmployeeID", cts:element-reference
  (xs:QName("ID")))
=> map:with("FirstName", cts:element-reference
  (xs:QName("FirstName")))
=> map:with("LastName", cts:element-reference
  (xs:QName("LastName")))
=> map:with("uri", cts:uri-reference()))
=> op:result()
```

すべてのビューにはフラグメント ID が含まれます。op.fromLexicons から生成されるフラグメント ID は、ビューのフラグメント ID と結合するために使用できます。例えば、次のコードは各従業員の EmployeeID、FirstName、LastName、Position、およびドキュメント URI を返します。

JavaScript :

```
const op = require('/MarkLogic/optic');

const empldocid = op.fragmentIdCol('empldocid');
const uridocid = op.fragmentIdCol('uridocid');
const employees = op.fromView('main', 'employees', null,
empldocid);
const DFrags = op.fromLexicons({'URI': cts.uriReference()},
                                  null, uridocid)

const Plan =
    employees.joinInner(DFrags, op.on(empldocid,
uridocid))
             .select(['URI', 'EmployeeID', 'FirstName',
                      'LastName', 'Position']);
Plan.result() ;
```

XQuery :

```
xquery version "1.0-ml";

import module namespace op="http://marklogic.com/optic"
     at "/MarkLogic/optic.xqy";

let $empldocid := op:fragment-id-col("empldocid")
let $uridocid := op:fragment-id-col("uridocid")
let $employees := op:from-view("main", "employees", (),
$empldocid)
let $DFrags := op:from-lexicons(map:entry("URI",
cts:uri-reference()),(), $uridocid)

return $employees
   => op:join-inner($DFrags, op:on($empldocid, $uridocid))
   => op:select((op:view-col("employees", "EmployeeID"),
                ("URI", "FirstName", "LastName", "Position")))
   => op:result()
```

19.3.4 fromLiterals の例

`fromLiterals` 関数を使用すると、文字列の配列およびオブジェクトの実行時入力に基づいて行を動的に生成できます。このデータアクセス関数は、テストやデバッグに役立ちます。

2 つの行から成るテーブルを構築し、`id` カラムの値が 1 に一致する行を返します。

JavaScript :

```
const op = require('/MarkLogic/optic');
op.fromLiterals([
            {id:1, name:'Master 1', date:'2015-12-01'},
            {id:2, name:'Master 2', date:'2015-12-02'}
            ])
   .where(op.eq(op.col('id'),1))
   .result();
```

XQuery :

```
xquery version "1.0-ml";

import module namespace op="http://marklogic.com/optic"
     at "/MarkLogic/optic.xqy";

op:from-literals(
    map:entry("columnNames",
              json:to-array(("id", "name", "date")))
    => map:with("rowValues", (
                json:to-array((  1,  "Master 1",
"2015-12-01")),
                json:to-array((  2,  "Master 2",
"2015-12-02")))))
    => op:where(op:eq(op:col("id"), 1))
    => op:result()
```

5 つの行から成るテーブルを構築し、group 1 と group 2 の平均値を返します。

JavaScript :

```
const op = require('/MarkLogic/optic');
op.fromLiterals([
            {group:1, val:2},
            {group:1, val:4},
            {group:2, val:3},
            {group:2, val:5},
            {group:2, val:7}
            ])
   .groupBy('group', op.avg('valAvg', 'val'))
   .orderBy('group')
   .result()
```

XQuery :

```
xquery version "1.0-ml";

import module namespace op="http://marklogic.com/optic"
     at "/MarkLogic/optic.xqy";

op:from-literals((
            map:entry("group", 1) => map:with("val", 2),
            map:entry("group", 1) => map:with("val", 4),
            map:entry("group", 2) => map:with("val", 3),
            map:entry("group", 2) => map:with("val", 5),
            map:entry("group", 2) => map:with("val", 7)
            ))
    => op:group-by("group", op:avg("valAvg", "val"))
    => op:order-by("group")
    => op:result()
```

19.4 オプティッククエリの種類

このセクションでは、オプティッククエリのいくつかの種類について説明します。このセクションの例は、『*SQL Data Modeling Guide*』の章 SQL on MarkLogic Server Quick Start で説明するドキュメントとテンプレートビューに基づいています。

次のトピックで構成されています。

- 基本クエリ
- 集約とグループ化
- 行結合
- ドキュメントの結合
- union、intersect、および except
- ドキュメントのクエリ

19.4.1 基本クエリ

オプティック API を使用するには、まず、ドキュメントのビューに対して基本クエリを実行します。ビューをクエリすると、行が返されます。

例えば、次のコマンドは、すべての従業員の ID と名前を ID 番号順にリストアップします。

JavaScript :

```
const op = require('/MarkLogic/optic');

op.fromView('main', 'employees')
   .select(['EmployeeID', 'FirstName', 'LastName'])
   .orderBy('EmployeeID')
   .result();
```

XQuery :

```
xquery version "1.0-ml";

import module namespace op="http://marklogic.com/optic"
     at "/MarkLogic/optic.xqy";

op:from-view("main", "employees")
   => op:select(("EmployeeID", "FirstName", "LastName"))
   => op:order-by("EmployeeID")
   => op:result()
```

19.4.2 集約とグループ化

複数のドキュメントで値に対して集計関数を簡単に実行するには、MarkLogic オプティック API を使用します。次の例は、従業員の経費に関する基本的な統計情報を取得するために、いくつかの操作を実行しています。これらの例で使用している `op.math.trunc` および `omath:trunc` プロキシ関数については、「カラム値を処理するための式関数」（331 ページ）を参照してください。

オプティックにおけるグループ化は SQL とは異なります。SQL では、グループ化キーが GROUP BY ステートメント内にあり、集約は SELECT 内で別途宣言されます。オプティックの group-by 操作では、グループ化キーは 1 番目のパラメータであり、集約は、オプションである 2 番目のパラメータです。この方法により、オプティックでは、シーケンスと配列を group-by 操作で集約し、それらのシーケンスや配列を操作する式関数を呼び出すことができます。例えば、`math:*` 関数（「カラム値を処理するための式関数」（331 ページ）を参照）の多くはシーケンスを取ります。

オプティックでは、集計関数をグループに適用する代わりに、シンプルなカラムを提供できます。オプティックは、グループ内の任意の 1 行についてカラムの値をサンプリングします。これは、そのカラムの値が、グループ内のすべての行で同じである場合に便利です。例えば、部門番号でグループ化するが、部門名でサンプリングする場合です。

JavaScript :

```
const op = require('/MarkLogic/optic');

op.fromView('main', 'expenses')
  .groupBy(null, [
            op.count('ExpenseReports', 'EmployeeID'),
            op.min('minCharge', 'Amount'),
            op.avg('average', 'Amount'),
            op.max('maxCharge', 'Amount')
          ])
  .select(['ExpenseReports',
           'minCharge',
           op.as('avgCharge', op.math.trunc(op.col
             ('average'))),
           'maxCharge'])
  .result();
```

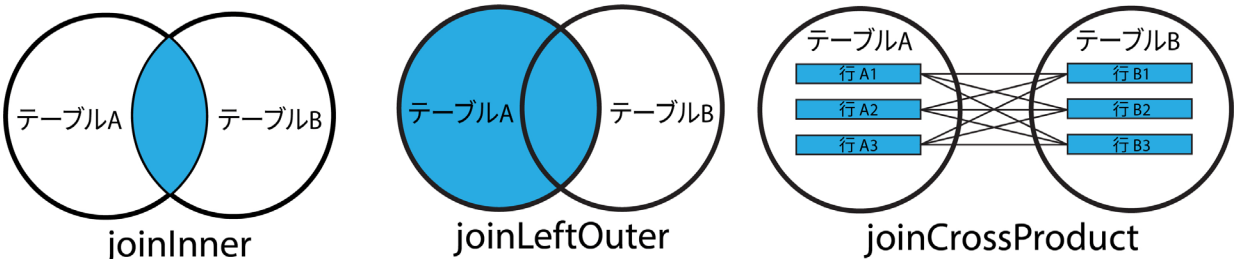
XQuery :

```
xquery version "1.0-ml";

import module namespace op="http://marklogic.com/optic"
     at "/MarkLogic/optic.xqy";

import module namespace
     omath="http://marklogic.com/optic/expression/math"
        at "/MarkLogic/optic/optic-math.xqy";

let $expenses  := op:from-view("main", "expenses")

return $expenses
   => op:group-by((), (
         op:count("ExpenseReports", "EmployeeID"),
         op:min("minCharge", "Amount"),
         op:avg("average", "Amount"),
         op:max("maxCharge", "Amount")
      ))
   => op:select(("ExpenseReports", "minCharge",
                op:as("avgCharge", omath:trunc(op:col
                ("average"))),"maxCharge"))
   => op:result();
```

19.4.3 行結合

オプティックでは、次の種類の行結合をサポートします。

メソッド	説明
joinInner	左右の行セットのキーの一致ごとに、左右の行を 1 つずつ連結した出力行セットを 1 つ作成します。
joinLeftOuter	左の行セットのすべての行と、右の行セットで一致する行で、出力行セットを 1 つ作成します。一致する行がない場合は NULL を作成します。
joinCrossProduct	左のすべての行と右のすべての行を連結して、出力行セットを 1 つ作成します。



このセクションの例は、employees（従業員）ビューと expenses（経費）ビューを結合し、従業員の経費とそのカテゴリに関して、個々のドキュメントで提供されるよりも詳細な情報を返します。

19.4.3.1 joinInner

次のクエリでは、AccessPlan.prototype.joinInner および op:join-inner 関数を利用して、各従業員のすべての経費および経費カテゴリを従業員番号順に返します。結合によって、別の expenses ドキュメントに格納されている情報で employee データが補完されます。内部結合はフィルタとして機能し、その employees と expenses のみが含まれます。

JavaScript :

```
const op = require('/MarkLogic/optic');

const employees = op.fromView('main', 'employees');
const expenses = op.fromView('main', 'expenses');

const Plan =
   employees.joinInner(expenses, op.on(employees.col
     ('EmployeeID'),expenses.col('EmployeeID')))
    .select([employees.col('EmployeeID'), 'FirstName',
    'LastName',
             expenses.col('Category'), 'Amount'])
    .orderBy(employees.col('EmployeeID'))
Plan.result();
```

XQuery :

```
xquery version "1.0-ml";

import module namespace op="http://marklogic.com/optic"
     at "/MarkLogic/optic.xqy";

let $employees := op:from-view("main", "employees")
let $expenses  := op:from-view("main", "expenses")

return $employees
   => op:join-inner($expenses, op:on(
          op:view-col("employees", "EmployeeID"),
          op:view-col("expenses",   "EmployeeID")))
   => op:select((op:view-col("employees", "EmployeeID"),
                 "FirstName", "LastName", "Category"))
   => op:order-by(op:view-col("employees", "EmployeeID"))
   => op:result()
```

`AccessPlan.prototype.where` および `op:where` 関数を使用して、許可されている上限を超える従業員の経費を見つけます。`employees`、`expenses`、および `category` の上限を結合して、従業員の経費の全体像を把握します。

JavaScript :

```
const op = require('/MarkLogic/optic');

const employees = op.fromView('main', 'employees');
const expenses = op.fromView('main', 'expenses');
const expenselimit = op.fromView('main', 'expenselimit');

const Plan =
   employees.joinInner(expenses,
op.on(employees.col('EmployeeID'),
                        expenses.col('EmployeeID')))
   .joinInner(expenselimit, op.on(expenses.col
       ('Category'),expenselimit.col('Category')))
   .where(op.gt(expenses.col('Amount'),
expenselimit.col('Limit')))
   .select([employees.col('EmployeeID'), 'FirstName',
     'LastName', expenses.col('Category'),
expenses.col('Amount'),
            expenselimit.col('Limit') ])
   .orderBy(employees.col('EmployeeID'))
Plan.result();
```

XQuery :

```
xquery version "1.0-ml";

import module namespace op="http://marklogic.com/optic"
     at "/MarkLogic/optic.xqy";

let $employees := op:from-view("main", "employees")
let $expenses  := op:from-view("main", "expenses")
let $expenselimit  := op:from-view("main", "expenselimit")

return $employees
   => op:join-inner($expenses, op:on(
               op:view-col("employees", "EmployeeID"),
               op:view-col("expenses", "EmployeeID")))
   => op:join-inner($expenselimit, op:on(
               op:view-col("expenses", "Category"),
               op:view-col("expenselimit", "Category")))
   => op:where(op:gt(op:view-col("expenses", "Amount"),
               op:view-col("expenselimit", "Limit")))
   => op:select((op:view-col("employees", "EmployeeID"),
               "FirstName", "LastName",
               op:view-col("expenses", "Category"),
               op:view-col("expenses", "Amount"),
               op:view-col("expenselimit", "Limit")))
   => op:order-by(op:view-col("employees", "EmployeeID"))
   => op:result()
```

19.4.3.2 joinLeftOuter

次のクエリは、`AccessPlan.prototype.joinLeftOuter` および `op:join-left-outer` 関数を使用して、各従業員のすべての経費および経費カテゴリを従業員番号順に返します。一致する経費レコードがない従業員については null 値を返します。

JavaScript :

```
const op = require('/MarkLogic/optic');

const employees = op.fromView('main', 'employees');
const expenses = op.fromView('main', 'expenses');

const Plan =
   employees.joinLeftOuter(expenses, op.on(employees.col
     ('EmployeeID'),
                        expenses.col('EmployeeID')))
    .orderBy(employees.col('EmployeeID'))
Plan.result();
```

XQuery :

```
xquery version "1.0-ml";

import module namespace op="http://marklogic.com/optic"
     at "/MarkLogic/optic.xqy";

let $employees := op:from-view("main", "employees")
let $expenses  := op:from-view("main", "expenses")

return $employees
   => op:join-left-outer($expenses, op:on(
          op:view-col("employees", "EmployeeID"),
          op:view-col("expenses",   "EmployeeID")))
   => op:order-by(op:view-col("employees", "EmployeeID"))
   => op:result()
```

19.4.3.3 joinCrossProduct

次のクエリは、`AccessPlan.prototype.joinCrossProduct` および `op:join-cross-product` 関数を利用して、従業員肩書（地位）ごとにすべての経費および経費カテゴリを経費カテゴリ順に返します。特定の地位にある従業員に、特定のカテゴリの経費がない場合、報告される経費は 0 になります。

JavaScript :

```
const op = require('/MarkLogic/optic');

const employees = op.fromView('main', 'employees');
const expenses = op.fromView('main', 'expenses');

expenses.groupBy ('Category')
   .joinCrossProduct(employees.groupBy('Position'))
   .select(null, 'all')
   .joinLeftOuter(
         expenses.joinInner(employees,
                            op.on(employees.col('EmployeeID'),
                            expenses.col('EmployeeID'))
                            )
        .groupBy(['Category', 'Position'],
                 op.sum('rawExpense', expenses.col('Amount'))
                )
        .select(null, 'expensed'),
        [op.on(op.viewCol('expensed', 'Category'),
               op.viewCol('all', 'Category')),
         op.on(op.viewCol('expensed', 'Position'),
               op.viewCol('all', 'Position'))]
   )
   .select([op.viewCol('all', 'Category'),
            op.viewCol('all', 'Position'),
            op.as('expense',
op.sem.coalesce(op.col('rawExpense'), 0))
           ])
   .orderBy(['Category', 'Position'])
   .result();
```

XQuery :

```
xquery version "1.0-ml";

import module namespace op="http://marklogic.com/optic"
     at "/MarkLogic/optic.xqy";
import module namespace
     osem="http://marklogic.com/optic/expression/sem"
         at "/MarkLogic/optic/optic-sem.xqy";

let $employees := op:from-view("main", "employees")
let $expenses  := op:from-view("main", "expenses")
let $rawExpense  := op:col("rawExpense")

return $expenses
   => op:group-by('Category')
   => op:join-cross-product($employees => op:group-by
        ("Position"))
   => op:select((), 'all')
   => op:join-left-outer(
         $expenses
         => op:join-inner($employees, op:on(
                op:col($employees, "EmployeeID"),
                op:col($expenses, "EmployeeID")
                ))
         => op:group-by(("Category", "Position"),
                  op:sum("rawExpense",  op:col($expenses,
"Amount")))
         => op:select((), "expensed"),
           (op:on(op:view-col("expensed", "Category"),
                  op:view-col("all", "Category")),
            op:on(op:view-col("expensed", "Position"),
                  op:view-col("all", "Position")))
         )
   => op:select((op:view-col("all", "Category"),
                 op:view-col("all", "Position"),
                 op:as("expense",
                        osem:coalesce((op:col("rawExpense"
), 0)))))
   => op:order-by(("Category", "Position"))
   => op:result();
```

19.4.4 ドキュメントの結合

オプティック API は、ビュー内の行にアクセスできるだけでなく、ドキュメント自体にもアクセスできます。

オプティックでは、次の種類のドキュメントの結合をサポートします。

メソッド	説明
joinDoc	ソースドキュメントを行で結合します（特に、行に射影されない詳細がソースドキュメントにある場合）。この場合、フラグメント ID カラムに名前を付け、それを結合で使用してください。
joinDocUri	関連するドキュメントをドキュメント URI に基づいて結合します。`AccessPlan.prototype.joinDocUri` メソッドを使用すると、簡単にドキュメントを URI で結合できます。ただし、細かい制御が必要な場合（関連するドキュメントの左外部結合の場合など）は、`cts.uriReference` レキシコンによる明示的結合を使用してフラグメント ID を取得し、そのフラグメント ID でドキュメントを結合できます。ドキュメントの結合後は、`op.xpath` 関数を使用して射影したり、`xdmp:*` 関数を使用してドキュメントのメタデータが含まれるカラムを追加したりできます。

注： ドキュメントを結合する前に、行をフィルタリングまたは制限して、取得されるドキュメントの数を最小限にしてください。

19.4.4.1 joinDoc

次の例は、AccessPlan.prototype.joinDoc または op:join-doc 関数によって、行データの後に「employee」および「expense」ソースドキュメントが返されます。結合は、op.fromView で返されたドキュメントフラグメント ID に対して実行されます。

JavaScript :

```
const op = require('/MarkLogic/optic');

const empldocid = op.fragmentIdCol('empldocid');
const expdocid = op.fragmentIdCol('expdocid');
const employees = op.fromView('main', 'employees', null,
empldocid);
const expenses = op.fromView('main', 'expenses', null,
expdocid);

const Plan =
   employees.joinInner(expenses, op.on(employees.col
     ('EmployeeID'), expenses.col('EmployeeID')))
            .joinDoc('Employee', empldocid)
            .joinDoc('Expenses', expdocid)
            .select([employees.col('EmployeeID'),'FirstNam
e',
                     'LastName', expenses.col('Category'),
'Amount',
                     'Employee', 'Expenses'])
            .orderBy(employees.col('EmployeeID'))
Plan.result();
```

XQuery :

```
xquery version "1.0-ml";

import module namespace op="http://marklogic.com/optic"
     at "/MarkLogic/optic.xqy";

let $empldocid := op:fragment-id-col("empldocid")
let $expdocid := op:fragment-id-col("expdocid")
let $employees := op:from-view("main", "employees", (),
$empldocid)
let $expenses := op:from-view("main", "expenses", (),
$expdocid)

return $employees
   => op:join-inner($expenses, op:on(
               op:view-col("employees", "EmployeeID"),
               op:view-col("expenses", "EmployeeID")))
   => op:join-doc("Employee", $empldocid)
   => op:join-doc("Expenses", $expdocid)
   => op:select((op:view-col("employees", "EmployeeID"),
               "FirstName", "LastName",
               op:view-col("expenses", "Category"),
               op:view-col("expenses", "Amount"),
               "Employee", "Expenses"))
   => op:order-by(op:view-col("employees", "EmployeeID"))
   => op:result()
```

19.4.4.2 joinDocUri

次の例は、`AccessPlan.prototype.joinDocUri` または `op:join-doc-uri` 関数を使用して、ドキュメント URI と行データを返す方法を示しています。

JavaScript :

```
const op = require('/MarkLogic/optic');
const empldocid = op.fragmentIdCol('empldocid');
const employees = op.fromView('main', 'employees', null,
empldocid);

employees.joinDocUri(op.col('uri'), empldocid)
         .result();
```

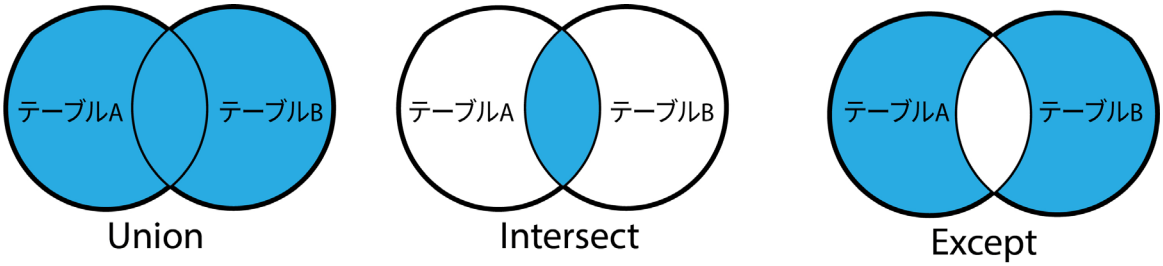
XQuery :

```
xquery version "1.0-ml";

import module namespace op="http://marklogic.com/optic"
      at "/MarkLogic/optic.xqy";

let $empldocid := op:fragment-id-col("empldocid")
return op:from-view("main", "employees", (), $empldocid)
     => op:join-doc-uri(op:col("uri"), $empldocid)
     => op:result()
```

19.4.5 union、intersect、および except

オプティックでは、データを結合して行を新規作成する次の方法をサポートしています。

メソッド	説明
union	入力行セットのすべての行を結合します。一部の入力行セットのみに存在するカラムの値は、それ以外の行セットの行では、実質的に null になります。
intersect	左右両方の行セットで同じカラムと同じ値を持つ行から、出力行セットを 1 つ作成します。
except	左右両方の行セットで同じカラムを持つ行から、出力行セットを 1 つ作成しますが、左の行セットのカラム値と右の行セットのカラム値は一致しません。



このセクションの例は、employees（従業員）ビューと expenses（経費）ビューを操作し、従業員の経費とそのカテゴリに関して、個々のドキュメントで提供されるよりも詳細な情報を返します。

19.4.5.1 union

次のクエリは、`AccessPlan.prototype.union` および `op:union` 関数を利用して、各従業員のすべての経費および経費カテゴリを従業員番号順に返します。

JavaScript :

```
const op = require('/MarkLogic/optic');

const employees = op.fromView('main', 'employees');
const expenses = op.fromView('main', 'expenses');

const Plan =
  employees.union(expenses)
           .whereDistinct()
           .orderBy([employees.col('EmployeeID')])
Plan.result();
```

XQuery :

```
xquery version "1.0-ml";

import module namespace op="http://marklogic.com/optic"
      at "/MarkLogic/optic.xqy";

let $employees := op:from-view("main", "employees")
let $expenses  := op:from-view("main", "expenses")

return $employees
   => op:union($expenses)
   => op:where-distinct()
   => op:order-by(op:view-col("employees", "EmployeeID"))
   => op:result()
```

19.4.5.2 intersect

次のクエリは、`AccessPlan.prototype.intersect` および `op:intersect` 関数を利用して、テーブル `tab1` と `tab2` 内の一致するカラムおよび値を返します。

> **注：** この例で `op.fromLiterals` 関数が使用されているのは、冗長なカラムおよび値がデータセットに含まれていないためです。

JavaScript :

```
const op = require('/MarkLogic/optic');

const tab1 = op.fromLiterals([
                     {id:1, val:'a'},
                     {id:2, val:'b'},
                     {id:3, val:'c'}
                     ]);

const tab2 = op.fromLiterals([
                     {id:1, val:'x'},
                     {id:2, val:'b'},
                     {id:3, val:'c'}
                     ]);

tab1.intersect(tab2)
        .orderBy('id')
        .result();
```

XQuery :

```
xquery version "1.0-ml";

import module namespace op="http://marklogic.com/optic"
      at "/MarkLogic/optic.xqy";

let $tab1 := op:from-literals((
    map:entry("id", 1) => map:with("val", "a"),
    map:entry("id", 2) => map:with("val", "b"),
    map:entry("id", 3) => map:with("val", "c")
    ))

let $tab2  := op:from-literals((
    map:entry("id", 1) => map:with("val", "x"),
    map:entry("id", 2) => map:with("val", "b"),
    map:entry("id", 3) => map:with("val", "c")
    ))

return $tab1
    => op:intersect($tab2)
    => op:order-by("id")
    => op:result()
```

19.4.5.3 except

次のクエリは、`AccessPlan.prototype.except` および `op:except` 関数を利用して、`tab2` のカラムと値に一致しない、`tab1` のカラムと値を返します。

> **注：** この例で `op.fromLiterals` 関数が使用されているのは、冗長なカラムおよび値がデータセットに含まれていないためです。

JavaScript :

```
const op = require('/MarkLogic/optic');

const tab1 = op.fromLiterals([
                     {id:1, val:'a'},
                     {id:2, val:'b'},
                     {id:3, val:'c'}
                     ]);

const tab2 = op.fromLiterals([
                     {id:1, val:'x'},
                     {id:2, val:'b'},
                     {id:3, val:'c'}
                     ]);

tab1.except(tab2)
        .orderBy('id')
        .result();
```

XQuery :

```
xquery version "1.0-ml";

import module namespace op="http://marklogic.com/optic"
      at "/MarkLogic/optic.xqy";

let $tab1 := op:from-literals((
    map:entry("id", 1) => map:with("val", "a"),
    map:entry("id", 2) => map:with("val", "b"),
    map:entry("id", 3) => map:with("val", "c")
    ))

let $tab2  := op:from-literals((
    map:entry("id", 1) => map:with("val", "x"),
    map:entry("id", 2) => map:with("val", "b"),
    map:entry("id", 3) => map:with("val", "c")
    ))

return $tab1
    => op:except($tab2)
    => op:order-by("id")
    => op:result()
```

19.4.6 ドキュメントのクエリ

MarkLogic オプティック API は、他の種類のクエリと結合できます。ドキュメントの複数の部分が行のビューに含まれていない場合でも、ドキュメントのクエリに基づいて行を制限できます。次の例は、`AccessPlan.prototype.where` および `op:where` 関数を使用して、オプティック API 内のドキュメントのクエリを表現しています。

JavaScript :

```
const op = require('/MarkLogic/optic');

op.fromView('main', 'employees')
  .where(cts.andQuery([cts.wordQuery('Senior'),
                       cts.wordQuery('Researcher')]))
  .select(['FirstName', 'LastName', 'Position'])
  .result();
```

XQuery :

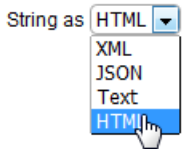
```
xquery version "1.0-ml";

import module namespace op="http://marklogic.com/optic"
     at "/MarkLogic/optic.xqy";

let $employees  := op:from-view("main", "employees")

return $employees
   => op:where(cts:and-query((cts:word-query("Senior"),
                              cts:word-query("Researcher"))))
   => op:select(("FirstName", "LastName", "Position"))
   => op:result()
```

19.5 オプティック出力の処理

Query Console のオプティック JavaScript クエリは、シリアライズされた JSON オブジェクトの形式で結果を出力します。ほとんどの場合は、オプティックの出力を使用するコードをいくつか記述します。例えば、次のクエリは、オプティックの出力を HTML のテーブルにマッピングします。

```
const op = require('/MarkLogic/optic');

let keys = null;
const rowItr = op.fromView('main', 'employees')
  .map(row => {
    if (keys === null) {
      keys = Object.keys(row);
    }
    return `<tr>${keys.map(key => `<td>${row[key]}</td>`)}
      </tr>`;
  })
  .result();

const rows = Array.from(rowItr).join('\n');
const header = `<tr>${keys.map(key =>
`<th>${key}</th>`)}</tr>`;
const report = `<table>\n${header}\n${rows}\n</table>`;
report;
```

Query Console で出力をテーブルとして表示するには、[String as] メニューで [HTML] を選択します。



19.6 カラム値を処理するための式関数

オプティックは、op.col および op:col から返されるカラム値を処理するビルトイン関数を表現するための式関数をサポートしています。これは、データ型コンストラクタ、日時、期間、数値、シーケンス、文字列関数などです。式関数の特徴は、次の 2 つです。

- 各行のカラム値に対するビルトイン関数の遅延呼び出しのプロキシです。
- ネストして、値を変換する強力な式を作成できます。

例えば、math.trunc 関数は、op.math.trunc 式関数（JavaScript）および omath:trunc（XQuery）として表現されます。

例えば、返された 'average' 値の小数部分を切り捨てるには、次の操作を実行します。

```
op.math.trunc(op.col('average')) // JavaScript

omath:trunc(op:col('average')) (: XQuery :)
```

式関数でサポートされる JavaScript 関数のリストを次の表に示します。これらと同等の XQuery 関数もサポートされますが、「式関数に必要な XQuery ライブラリ」（337 ページ）に示すそれぞれのモジュールライブラリをインポートする必要があります。

次のリストには、使用する可能性のあるほとんどすべての値処理ビルトイン関数が示されています。ここに含まれない関数を呼び出すことはほとんどありませんが、その場合に備えてオプティック API には、遅延呼び出し用に汎用の op.call コンストラクタが用意されています。

```
op.call(moduleUri, functionName, arg*) => expression

op.call({uri:..., name:..., args:*})=> expression
```

op.call 関数を使用すると、一部のビルトインのパフォーマンスに悪影響を与えることがあるため、注意が必要です。この関数は、JavaScript 関数および XQuery 関数の呼び出しに使用することはできません。代わりに、map 関数や reduce 関数を使用して結果を後処理してください。

オプティック式関数でサポートされるビルトイン関数

cts.tokenize	fn.roundHalfToEven	math.variance	xdmp.integerToOctal
cts.stem	fn.secondsFromDateTime	math.varianceP	xdmp.keyFromQName
fn.abs	fn.secondsFromDuration	rdf.langString	xdmp.lshift64
fn.adjustDateTimeToTimezone	fn.secondsFromTime	rdf.langStringLanguage	xdmp.md5
fn.adjustDateToTimezone	fn.startsWith	sem.bnode	xdmp.monthNameFromDate
fn.adjustTimeToTimezone	fn.string	sem.coalesce	xdmp.mul64

オプティック式関数でサポートされるビルトイン関数			
fn.analyzeString	fn.stringJoin	sem.datatype	xdmp.nodeCollections
fn.avg	fn.stringLength	sem.if	xdmp.nodeMetadata
fn.baseUri	fn.stringToCode points	sem.invalid	xdmp.nodeMetadata Value
fn.boolean	fn.subsequence	sem.invalidDatatype	xdmp.nodeKind
fn.ceiling	fn.substring	sem.iri	xdmp.nodePermissions
fn.codepointEqual	fn.substringAfter	sem.iriToQName	xdmp.nodeUri
fn.codepointsToString	fn.substringBefore	sem.isBlank	xdmp.not64
fn.compare	fn.sum	sem.isIRI	xdmp.octalToInteger
fn.concat	fn.tail	sem.isLiteral	xdmp.or64
fn.currentDateTime	fn.timezoneFromDate	sem.isNumeric	xdmp.parseDateTime
fn.currentDate	fn.timezoneFromDate Time	sem.lang	xdmp.parseYymmdd
fn.currentTime	fn.timezoneFromTime	sem.langMatches	xdmp.path
fn.contains	fn.tokenize	sem.QNameToIri	xdmp.position
fn.count	fn.translate	sem.random	xdmp.QNameFromKey
fn.dateTime	fn.true	sem.sameTerm	xdmp.quarterFromDate
fn.dayFromDate	fn.unordered	sem.timezoneString	xdmp.random
fn.dayFromDateTime	fn.upperCase	sem.typedLiteral	xdmp.resolveUri
fn.daysFromDuration	fn.yearFromDate	sem.unknown	xdmp.rshift64
fn.deepEqual	fn.yearFromDateTime	sem.unknownDatatype	xdmp.sha1
fn.defaultCollation	fn.yearsFrom Duration	sem.uuid	xdmp.sha256
fn.distinctValues	json.array	sem.uuidString	xdmp.sha384
fn.documentUri	json.arraySize	spell.double Metaphone	xdmp.sha512

オプティック式関数でサポートされるビルトイン関数			
fn.empty	json.arrayValues	spell.levenshteinDistance	xdmp.step64
fn.encodeForUri	json.object	spell.romanize	xdmp.strftime
fn.endsWith	json.objectDefine	sql.bitLength	xdmp.timestampToWallclock
fn.escapeHtmlUri	json.subarray	sql.dateadd	xdmp.toJSON
fn.exists	json.toArray	sql.datediff	xdmp.type
fn.false	map.contains	sql.datepart	xdmp.urlDecode
fn.floor	map.count	sql.day	xdmp.urlEncode
fn.formatDate	map.entry	sql.dayname	xdmp.wallclockToTimestamp
fn.formatDateTime	map.get	sql.hours	xdmp.weekdayFromDate
fn.formatNumber	map.keys	sql.insert	xdmp.weekFromDate
fn.formatTime	map.map	sql.left	xdmp.xor64
fn.generateId	map.new	sql.minutes	xdmp.yeardayFromDate
fn.head	math.acos	sql.month	xs.anyURI
fn.hoursFromDateTime	math.asin	sql.monthname	xs.boolean
fn.hoursFromDuration	math.atan	sql.octetLength	xs.byte
fn.hoursFromTime	math.atan2	sql.quarter	xs.date
fn.implicitTimezone	math.ceil	sql.rand	xs.dateTime
fn.indexOf	math.correlation	sql.repeat	xs.dayTimeDuration
fn.inScopePrefixes	math.cos	sql.right	xs.decimal
fn.insertBefore	math.cosh	sql.seconds	xs.double
fn.iriToUri	math.cot	sql.sign	xs.duration
fn.lang	math.covariance	sql.space	xs.float
fn.localName	math.covarianceP	sql.timestampadd	xs.gDay
fn.localNameFromQName	math.degrees	sql.timestampdiff	xs.gMonth

オプティック式関数でサポートされるビルトイン関数			
fn.lowerCase	math.exp	sql.week	xs.gMonthDay
fn.matches	math.fabs	sql.weekday	xs.gYear
fn.max	math.floor	sql.year	xs.gYearMonth
fn.min	math.fmod	sql.yearday	xs.hexBinary
fn.minutesFromDateTime	math.frexp	xdmp.add64	xs.int
fn.minutesFromDuration	math.ldexp	xdmp.and64	xs.integer
fn.minutesFromTime	math.linearModel	xdmp.base64Decode	xs.language
fn.monthFromDate	math.linearModelCoeff	xdmp.base64Encode	xs.long
fn.monthFromDateTime	math.linearModelIntercept	xdmp.castableAs	xs.Name
fn.monthsFromDuration	math.linearModelRsquared	xdmp.crypt	xs.NCName
fn.name	math.log	xdmp.crypt2	xs.NMTOKEN
fn.namespaceUriFromQName	math.log10	xdmp.daynameFromDate	xs.negativeInteger
fn.namespaceUri	math.median	xdmp.decodeFromNCName	xs.nonNegativeInteger
fn.namespaceUriForPrefix	math.mode	xdmp.describe	xs.nonPositiveInteger
fn.nilled	math.modf	xdmp.diacriticLess	xs.normalizedString
fn.nodeName	math.percentile	xdmp.elementContentType	xs.numeric
fn.normalizeSpace	math.percentRank	xdmp.encodeForNCName	xs.positiveInteger
fn.normalizeUnicode	math.pi	xdmp.formatNumber	xs.QName
fn.not	math.pow	xdmp.fromJSON	xs.short
fn.number	math.radians	xdmp.getCurrentUser	xs.string

オプティック式関数でサポートされるビルトイン関数			
fn.prefixFromQName	math.rank	xdmp.hash32	xs.time
fn.QName	math.sin	xdmp.hash64	xs.token
fn.remove	math.sinh	xdmp.hexToInteger	xs.unsignedByte
fn.replace	math.sqrt	xdmp.hmacMd5	xs.unsignedInt
fn.resolveQName	math.stddev	xdmp.hmacSha1	xs.unsignedLong
fn.resolveUri	math.stddevP	xdmp.hmacSha256	xs.unsignedShort
fn.reverse	math.tan	xdmp.hmacSha512	xs.yearMonthDuration
fn.root	math.tanh	xdmp.initcap	
fn.round	math.trunc	xdmp.integerToHex	

19.6.1 式関数に必要な XQuery ライブラリ

XQuery では、式関数を使用するために、それぞれのビルトイン関数用に次のライブラリをインポートする必要があります。

cts 関数：

```
import module namespace
octs="http://marklogic.com/optic/expression/cts"
    at "/MarkLogic/optic/optic-cts.xqy";
```

fn 関数：

```
import module namespace
ofn="http://marklogic.com/optic/expression/fn"
    at "/MarkLogic/optic/optic-fn.xqy";
```

json 関数：

```
import module namespace
ojson="http://marklogic.com/optic/expression/json"
    at "/MarkLogic/optic/optic-json.xqy";
```

map 関数：

```
import module namespace
omap="http://marklogic.com/optic/expression/map"
    at "/MarkLogic/optic/optic-map.xqy";
```

math 関数：

```
import module namespace
omath="http://marklogic.com/optic/expression/math"
    at "/MarkLogic/optic/optic-math.xqy";
```

rdf 関数：

```
import module namespace
ordf="http://marklogic.com/optic/expression/rdf"
    at "/MarkLogic/optic/optic-rdf.xqy";
```

sem 関数：

```
import module namespace
osem="http://marklogic.com/optic/expression/sem"
    at "/MarkLogic/optic/optic-sem.xqy";
```

spell 関数：

```
import module namespace
ospell="http://marklogic.com/optic/expression/spell"
    at "/MarkLogic/optic/optic-spell.xqy";
```

sql 関数：

```
import module namespace
osql="http://marklogic.com/optic/expression/sql"
    at "/MarkLogic/optic/optic-sql.xqy";
```

xdmp 関数：

```
import module namespace
oxdmp="http://marklogic.com/optic/expression/xdmp"
    at "/MarkLogic/optic/optic-xdmp.xqy";
```

xs 関数：

```
import module namespace
oxs="http://marklogic.com/optic/expression/xs"
    at "/MarkLogic/optic/optic-xs.xqy";
```

式関数を入れ子にして、値を変換する強力な式を作成できます。例：

```
.select(['countUsers', 'minReputation',
       op.as('avgReputation', op.math.trunc(op.col('aRep'))),
'maxReputation',
       op.as('locationPercent',
             op.fn.formatNumber(op.xs.double(
                 op.divide(op.col('locationCount'),
                           op.col('countUsers'))),'##%'))
))
```

19.7 ブール、数値、および文字列演算子に相当する関数

関数	SPARQL	SQL	コメント
eq(valueExpression, valueExpression) => booleanExpression eq({left:..., right:...})=> booleanExpression	=	= ==	式では、カラムを識別するために呼び出しで `op.col` 値を渡す必要があります。
gt(valueExpression, valueExpression) => booleanExpression gt({left:..., right:...})=> booleanExpression	>	>	
ge(valueExpression, valueExpression) => booleanExpression ge({left:..., right:...})=> booleanExpression	>=	>=	
lt(valueExpression, valueExpression) => booleanExpression lt({left:..., right:...})=> booleanExpression	<	<	
le(valueExpression, valueExpression) => booleanExpression le({left:..., right:...})=> booleanExpression	<=	<=	
ne(valueExpression, valueExpression) => booleanExpression ne({left:..., right:...})=> booleanExpression	!=	!=	
and(booleanExpression+) => booleanExpression and({list:...})=> booleanExpression	&&	AND	

関数	SPARQL	SQL	コメント
or(booleanExpression+) => booleanExpression or({list:...})=> booleanExpression	||	OR	
not(booleanExpression) => booleanExpression not({condition:...})=> booleanExpression	!	NOT	
case(whenExpression+, valueExpression) => valueExpression case({list:..., otherwise:...})=> valueExpression	IF	CASE WHEN ELSE	
when(booleanExpression, valueExpression) => whenExpression when({condition:..., value:...})=> whenExpression		WHEN	
isDefined(col) => booleanExpression isDefined({column: ...})=> booleanExpression	BOUND	IS NULL	
add(numericExpression, numericExpression) => numericExpression add({left:..., right:...})=> numericExpression	+	+	カラム名は、`op.col` 値にする必要があります。
divide(numericExpression, numericExpression) => numericExpression divide({left:..., right:...})=> numericExpression	/	/	

関数	SPARQL	SQL	コメント
modulo(numericExpression, numericExpression) => numericExpression modulo({left:..., right:...})=> numericExpression		%	
multiply(numericExpression, numericExpression) => numericExpression multiply({left:..., right:...})=> numericExpression	*	*	
subtract(numericExpression, numericExpression) => numericExpression subtract({left:..., right:...})=> numericExpression	-	-	

注： 初期リリースでは、サブプランから返された値を使用する式（SQL または SPARQL EXISTS に類似）はサポートされていません。

19.8 ノードコンストラクタ関数

オプティックには、ツリー構造を構築できるノードコンストラクタ関数が用意されています。ノードコンストラクタ関数には次の機能があります。

- プロパティがカラム値を使用する JSON オブジェクト、あるいは、コンテンツまたは属性値がカラム値を使用する XML 要素を作成します。
- `op.xpath` を使用して抽出したドキュメントまたはノードを、構築したノードに挿入します。
- ノードの集約配列から JSON 配列を、またはノードの集約シーケンスから XML 要素を作成します。

次の表は、オプティックノードコンストラクタ関数をまとめたものです。各関数の詳細については、オプティック API のリファレンスドキュメントを参照してください。

関数	説明
op.jsonArray	指定された JSON ノードを項目として持つ JSON 配列を構築します。
op.jsonBoolean	指定された値を持つ JSON ブールノードを構築します。
op.jsonDocument	ルートコンテンツを持つ JSON ドキュメントを構築します。ルートコンテンツは正確に 1 つの JSON オブジェクトまたは配列ノードである必要があります。
op.jsonNull	JSON null ノードを構築します。
op.jsonNumber	指定された値を持つ JSON 数値ノードを構築します。
op.jsonObject	指定されたプロパティを持つ JSON オブジェクトを構築します。 properties 引数は、prop() 関数で構築されます。
op.jsonString	指定された値を持つ JSON テキストノードを構築します。
op.prop	JSON オブジェクトの JSON プロパティのキー式と値コンテンツを指定します。
op.xmlAttribute	名前およびアトミック値を持つ XML 属性を構築します。
op.xmlComment	アトミック値を持つ XML コメントを構築します。
op.xmlDocument	ルートコンテンツを持つ XML ドキュメントを構築します。
op.xmlElement	名前、0 個以上の属性ノード、および子コンテンツを持つ XML 要素を構築します。
op.xmlPI	アトミック値を持つ XML 処理指示を構築します。
op.xmlText	XML テキストノードを構築します。
op.xpath	ノード値を持つカラムから子ノードのシーケンスを抽出します。

例えば、次のクエリは、その下に示すような JSON ドキュメントを構築します。

```
const op = require('/MarkLogic/optic');
const employees = op.fromView('main', 'employees');

employees.select(op.as('Employee', op.jsonDocument(
    op.jsonObject([op.prop('ID and Name',
                  op.jsonArray([
                    op.jsonNumber(op.col('EmployeeID')),
                    op.jsonString(op.col('FirstName')),
                    op.jsonString(op.col('LastName'))
                   ])),
                  op.prop('Position',
                    op.jsonString(op.col('Position')))
    ])
)))
.result();
```

このクエリによって、次のような出力が生成されます。

```
{
   「Employee": {
    "ID and Name": [
      42,
      "Debbie",
      "Goodall"
    ],
    "Position": "Senior Widget Researcher"
  }
}
```

19.9 ベストプラクティスとパフォーマンスに関する考慮事項

オプティックには、返される行またはドキュメントに関するデフォルト / 暗黙の制限はありません。数万行にも上る大規模な結果セットを返すプランを作成すると、パフォーマンスが低下する可能性があります。パフォーマンスの問題が発生した場合は、`AccessPlan.prototype.offsetLimit` メソッドを使用するか、または `AccessPlan.prototype.offset` メソッドと `AccessPlan.prototype.limit` メソッドを組み合わせて、結果をページ分割することを推奨します。

19.10 オプティック実行プラン

オプティック実行プランは、アトミック操作のシーケンスによる論理データフローを表現します。オプティック API を使用してプラン定義を構築し、パイプラインでオブジェクトを作成および変更してから、`PreparePlan.prototype.result` 関数でプランを実行します。

`PreparePlan.prototype.explain` 関数を使用すると、実行プランを表示または保存できます。実行プラン定義は、行セットに対する操作で構成されます。これらの操作は、次のカテゴリに分類されます。

- データアクセス－実行プランはビュー、グラフ、またはリテラルから行セットを読み取ることができます。この場合、ビューは、トリプルインデックス、またはドキュメントのレンジインデックス値の共起の外積にアクセスできます。
- 行セット変更－実行プランは、変更された行セットを得るために、`where`、`order by`、`group` を使用してフィルタしたり、`select` を使用して射影したり、`limit` で行セットを制限したりできます。
- 行セット合成－実行プランは、単一の行セットを得るために、`join`、`union`、`intersect`、または `except` を使用して複数の行セットを結合できます。
- 行結果処理－実行プランは、最終的な行セットに対して実行する処理（map 処理や reduce 処理など）を指定できます。

ビューが実行プランとして開かれているときは、ビューに特殊なプロパティがあります。このプロパティには、ビューのカラムごとのプロパティを持つオブジェクトがあります。プロパティの名前はカラム名であり、プロパティの値は名前オブジェクトです。異なるビューに同じ名前を持つカラムが存在するというあいまいさを避けるために、ビューカラムのカラム名の先頭に、ビュー名と区切りのピリオドが付加されます。

実行プランの結果は、出力の MIME タイプに応じて、CSV、行指向の XML または JSON にシリアライズできます。実行プランを読み取る方法の詳細については、『*SQL Data Modeling Guide*』の Execution Plan を参照してください。

19.11 プランのパラメータ化

任意の値に置換できるプレースホルダーを作成するには、`op.param` 関数を使用します。プランの実行時に、パラメータの値を指定する必要があります。

プランエンジンはプランをキャッシュするため、以前に実行されたプランをパラメータ化すると、新しいプランを送信するよりも効率が向上します。

例えば、次のクエリは、`start` および `length` パラメータを使用して、`EmployeeID` の値を増分するように offsetLimit および `increment` パラメータを設定します。

```
const op = require('/MarkLogic/optic');

const employees = op.fromView('main', 'employees');

employees.offsetLimit(op.param('start'), op.param('length'))
   .select(['EmployeeID',
             op.as('incremented',
           op.add(op.col('EmployeeID'),
                                op.param('increment')))])
   .result(null, {start:1, length:2, increment:1});
```

19.12 シリアライズされたオプティッククエリのエクスポートとインポート

`IteratePlan.prototype.export` メソッドまたは `op:export` 関数を使用して、シリアライズされた形式のオプティッククエリをエクスポートできます。これにより、プランをファイルとして格納し、その後に `op.import` または `op:import` 関数でインポートしたり、`/v1/rows` REST 呼び出しでペイロードとして使用したりできます。

例えば、オプティッククエリをファイルとしてエクスポートするには、次の手順に従います。

JavaScript :

```
const op = require('/MarkLogic/optic');

const EmployeePlan =
   op.fromView('main', 'employees')
     .select(['EmployeeID', 'FirstName', 'LastName'])
     .orderBy('EmployeeID')
const planObj = EmployeePlan.export();

xdmp.documentInsert("plan.json", planObj)
```

XQuery :

```
xquery version "1.0-ml";

import module namespace op="http://marklogic.com/optic"
     at "/MarkLogic/optic.xqy";

let $plan := op:from-view("main", "employees")
   => op:select(("EmployeeID", "FirstName", "LastName"))
   => op:order-by("EmployeeID")
   => op:export()

return xdmp:document-insert("plan.json",  xdmp:to-json
($plan))
```

オプティッククエリをファイルからインポートして結果を出力するには、次の手順に従います。

JavaScript :

```
const op = require('/MarkLogic/optic');

op.import(cts.doc('plan.json').toObject())
  .result();
```

XQuery :

```
xquery version "1.0-ml";

import module namespace op="http://marklogic.com/optic"
     at "/MarkLogic/optic.xqy";

op:import(fn:doc("plan.json")/node())
   => op:result()
```

20.0 JSON の使用

この章では、MarkLogic サーバーで JSON を使用する方法について説明します。以下のセクションで構成されています。

- JSON、XML、および MarkLogic
- MarkLogic が JSON ドキュメントを表現する方法
- XPath を使用した JSON ドキュメントのトラバーサル
- JSON ドキュメントに対するインデックスおよびレキシコンの作成
- JSON と XML のフィールドクエリの相違点
- 地理情報データ、テンポラルデータ、およびセマンティックデータの表現
- 大きな整数値のシリアライゼーション
- ドキュメントのプロパティ
- XQuery における JSON の使用
- サーバーサイド JavaScript における JSON の使用
- JSON から XML へ、および XML から JSON への変換
- 低レベル JSON XQuery API とプリミティブ型
- JSON ドキュメントの読み込み

20.1 JSON、XML、および MarkLogic

JSON（JavaScript Object Notation）は、本来、JavaScript 間でデータを受け渡すために設計されたデータ交換形式です。Web アプリケーションでは、データをアプリケーションとサーバー（MarkLogic サーバー）の間でやり取りすることがたびたび必要になりますが、その際によく使用される形式が JSON です。JSON は、XML と同様に、マシンと人間の両者が理解できるように設計されています。JSON の詳細については、json.org を参照してください。

MarkLogic サーバーは、JSON ドキュメントをサポートしています。JSON を使用して、ドキュメントを格納したり、クライアントに結果を提供したりできます。元のデータが JSON であったかどうかは関係ありません。MarkLogic の JSON サポートについて、次にいくつかの特長を示します。

- JavaScript、XQuery、または XSLT を使用して、MarkLogic サーバー内の JSON ドキュメントに対してドキュメント操作や検索を実行できます。Node.js、Java、および REST クライアント API を使用して、クライアントアプリケーションから JSON ドキュメントに対してドキュメント操作や検索を実行できます。

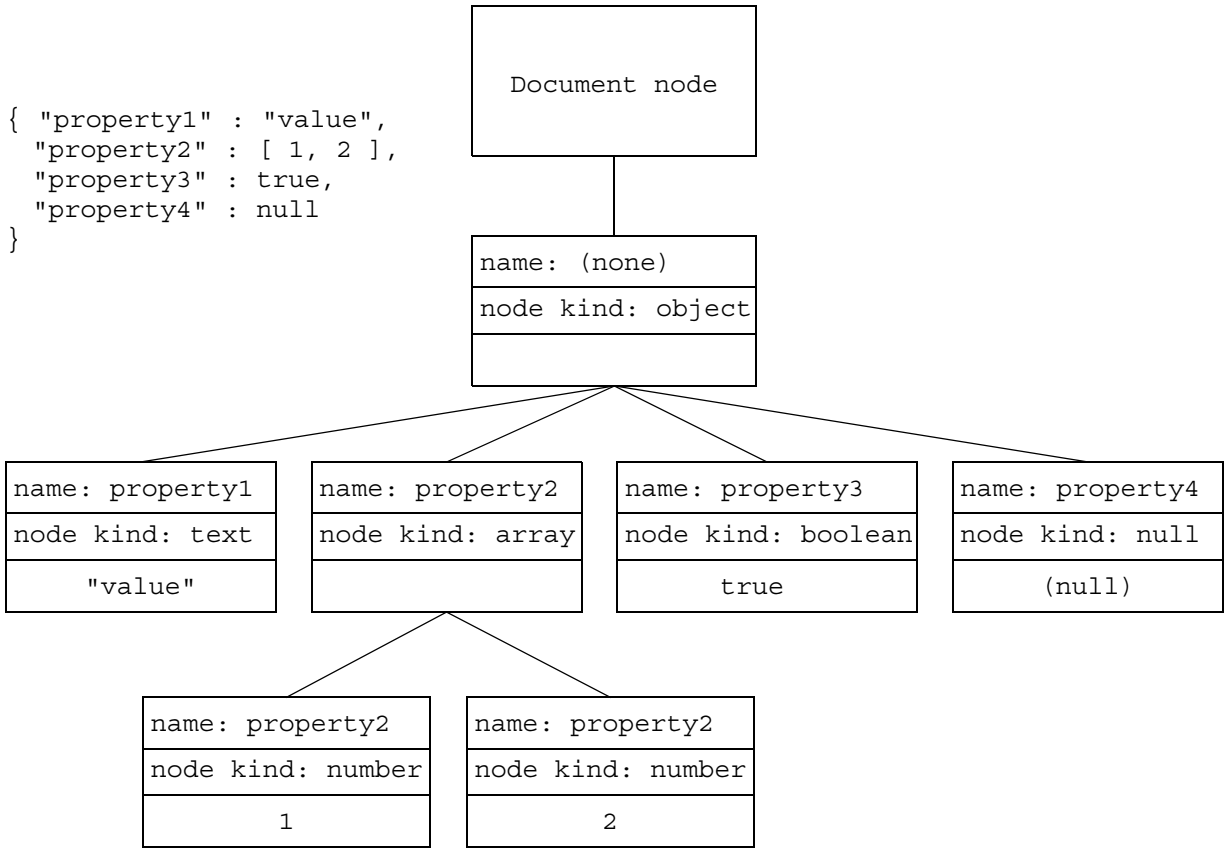
- すべてのクライアント API には、データを JSON で返すオプションがあります。このため、クライアントサイドアプリケーション開発者は、MarkLogic からのデータを簡単に処理できます。
- REST クライアント API および REST 管理 API は、JSON と XML の両方の入力を受け入れます。例えば、どちらの形式でも、クエリおよび設定情報を指定できます。
- MarkLogic クライアント API は、JSON ドキュメントの読み込みおよびクエリを完全にサポートしています。このため、JSON ドキュメントに対して細かく制御されたアクセスや、JSON コンテンツに対して検索およびファセットが可能です。
- JSON から XML に、または XML から JSON にデータを簡単に変換できます。このような変換を実行するための API の集まりが豊富に用意されており、変換後の XML 仕様や変換後の JSON 仕様に関して、高い柔軟性を備えています。これらをサポートする低レベル API が MarkLogic サーバーに組み込まれているため、非常に高速で変換されます。

20.2 MarkLogic が JSON ドキュメントを表現する方法

MarkLogic サーバーは、ノードのツリーとして JSON ドキュメントをモデル化します。このツリーのルートはドキュメントノードです。このモデルを理解すると、XPath を使用して JSON データを処理する方法や、ノードテストを実行する方法を理解しやすくなります。JavaScript で JSON ドキュメントを操作すると、通常は、JavaScript オブジェクトのようなコンテンツを処理できます。それでも、ドキュメントとオブジェクトの違いを認識しておくことは必要です。

JSON ドキュメントでは、ドキュメントノードの下にあるノードが JSON オブジェクト、配列、テキスト値、数値、ブール値、null 値を表します。オブジェクト、配列、数値、ブール型、null ノード型が含まれるのは JSON ドキュメントだけです。

例えば、次の図は、JSON オブジェクトと、JSON オブジェクトがデータベース内に JSON ドキュメントとして格納されているときのツリー表現を示しています（オブジェクトがドキュメントではなく、インメモリ構成体であったとしたら、ルートドキュメントノードは存在していません）。



ノードの名前は、最も内側にある JSON プロパティの名前です。例えば、前述のノードツリーで「property2」は配列ノードの名前であると同時に、各配列メンバーノードの名前です。

```
fn:node-name(fn:doc($uri)/property2/array-node()) ==>
"property2"
fn:node-name(fn:doc($uri)/property2[1]) ==> "property2"
```

上位のプロパティがないノードは、名前のないノードです。例えば、次の配列ノードには名前がないため、そのメンバーにも名前がありません。したがって、XQuery で fn:node-name を使用してノードの名前を取得しようとすると、空のシーケンスが返されます。

```
let $node := array-node { 1, 2 }
return fn:node-name($node//number-node[. eq 1])
==> an empty sequence
```

20.3 XPath を使用した JSON ドキュメントのトラバーサル

このセクションでは、XPath を使用して JSON ドキュメントの各部分やノードにアクセスする方法について説明します。JavaScript や XQuery のコードなど、XML データで使用できるあらゆる場所で、JSON データで XPath を使用できます。

次の内容が含まれます。

- XPath とは
- XPath の例
- ノードおよびノード値の選択
- ノードテスト演算子
- 配列および配列メンバーの選択

20.3.1 XPath とは

XPath は、本来は XML データ構造内のノードを処理するために設計された式言語です。MarkLogic サーバーでは、XPath を使用して JSON と XML をトラバースできます。XPath 式を使用すると、クエリの構成、インデックスの作成、フィールドの定義、JSON ドキュメント内のノード選択などを実行できます。

XPath は次の仕様で定義されています。

http://www.w3.org/TR/xpath20/#id-sequence-expressions

詳細については、『*XQuery and XSLT Reference Guide*』の XPath Quick Reference を参照してください。

20.3.2 XPath の例

XPath 式は、明示的なノードトラバーサル、クエリ構成、インデックス設定など、多くのさまざまなコンテキストで使用できます。このため、この章で示す XPath の例の多くは XPath 式のみを示し、実行コンテキストは示していません。

XPath 式で選択される対象を調べるためにノードトラバーサルを使用する場合は、テンプレートとして次のいずれかのパターンを Query Console で使用できます。

言語	テンプレート

XQuery	`xquery version "1.0-ml";`
	`let $node := xdmp:unquote(`*`json_literal`*`)`
	`return xdmp:describe($node/`*`xpath_expr`*`)`

サーバーサイド JavaScript	`const node = xdmp.toJSON(`*`jsObject`*`);` `xdmp.describe(node.xpath(`*`xpathExpr`*`));`

Query Console での書式に関係なく、結果のタイプを明確に示すために、`xdmp:describe`（XQuery）および `xdmp.describe`（JavaScript）の呼び出しで結果がラップされます。

XQuery では、XPath 式をノードに直接適用できますが、JavaScript では、Node.xpath メソッドを使用する必要があります。例えば、$node/a と node.xpath('/a/b') を比較してください。また、xpath メソッドは、JavaScript ではシーケンスを返します。したがって、このメソッドをアプリケーションで使用する場合は、結果に対して反復処理が必要になる可能性があります。

例えば、データ {"a": 1} が含まれる JSON ノードに XPath 式「/a」を適用した結果を調べる場合は、Query Console で次のいずれかの例を実行できます。

言語	テンプレート

XQuery	`xquery version "1.0-ml";`
	`let $node := xdmp:unquote('{"a": 1}')`
	`return xdmp:describe($node/a)`

サーバーサイド JavaScript	`const node = xdmp.toJSON({a: 1});` `xdmp.describe(node.xpath('/a'));`

20.3.3 ノードおよびノード値の選択

ほとんどの場合、XPath 式は 1 つあるいは複数のノードを選択します。ノードの値にアクセスするには、data() を使用します。例えば、次の XPath 式を比較してみましょう。JSON オブジェクトノードに { "a" : 1 } が含まれる場合、1 番目の式は、名前が「a」である数値ノードを選択し、2 番目の式は、そのノードの値を選択します。

```
(: XQuery :)
$node/a ==> number-node { 1 }
$node/a/data() ==> 1

// JavaScript
node.xpath('/a') ==> number-node { 1 }
node.xpath('/a/data()') ==> 1
```

ノードテスト演算子を使用すると、選択されるノードを、ノード型またはノード型と名前で制限できます。詳細については、「ノードテスト演算子」（353 ページ）を参照してください。

JSON 配列は、XPath を使用したアクセス時には、デフォルトでシーケンスのように扱われます。詳細については、「配列および配列メンバーの選択」（354 ページ）を参照してください。

次の JSON オブジェクトノードが変数 $node 内に存在するとします。

```
{ "a": {
    "b": "value",
    "c1": 1,
    "c2": 2,
    "d": null,
    "e": {
      "f": true,
      "g": ["v1", "v2", "v3"]
    }
} }
```

次の表は、このオブジェクトに適用された XPath 式によって、どのノードが選択されるかを示しています。「XPath の例」（350 ページ）で説明するパターンを使用して、これらの例を Query Console で試してみてください。

XPath 式	結果
`$node/a/b`	`"value"`

`$node/a/c1`	`A number node named "c1" with value 1:`
	
	`number-node{ 1 }`

`$node/a/c1/data()`	`1`
`$node/a/d`	`null-node { }`
`$node/a/e/f`	`boolean-node{ fn:true() }`
`$node/a/e/f/data()`	`true`

`$node/a/e/g`	`A sequence containing 3 values:`
	
	`("v1", "v2", "v3")`

`$node/a/e/g[2]`	`"s2"`

`$node/a[c1=1]`	`An object node equivalent to the following JSON:`
	
	`{`
	`  "b": "value”,`
	`  "c1": 1,`
	`  "c2": 2,`
	`...`
	`}`

20.3.4 ノードテスト演算子

次のノードテスト演算子を使用して、選択されるノードをノード型で制限できます。

- `object-node()`
- `array-node()`
- `number-node()`
- `boolean-node()`
- `null-node()`
- `text()`

すべてのノードテスト演算子では、JSON プロパティ名を指定するオプションの文字列パラメータを使用できます。例えば、次の式は、「a」という名前の任意のブール型ノードに一致します。

```
boolean-node("a")
```

次の JSON オブジェクトがインメモリオブジェクト `$node` 内に存在するとします。

```
{ "a": {
    "b": "value",
    "c1": 1,
    "c2": 2,
    "d": null,
    "e": {
      "f": true,
      "g": ["v1", "v2", "v3"]
    }
} }
```

ノードテスト演算子を使用した XPath 式の例を次の表に示します。「XPath の例」（350 ページ）で説明するパターンを使用して、これらの例を Query Console で試してみてください。

XPath 式	結果

`$node//number-node()`
 	
`$node/a/number-node()`
	`A sequence containing two number nodes, one named "c1" and one named "c2":`
	
	`(number-node{1}, number-node{2})`

XPath 式	結果

`$node//number-node()/data()`	`A sequence containing 2 numbers:`
	
	`(1,2)`

`$node/a/number-node("c2")`	`The number node named "c2":`
	
	`number-node{2}`

`$node//text()`	`("value", "v1", "v2", "v3")`
`$node/a/text("b")`	`"value"`

`$node//object-node()`	`A sequence of object nodes equivalent to the following JSON objects:`
	
	`{"a": {"b": "value", ...} }`
	`{"b": "value", "c1": 1, ...}`
	`{"f": true, "g": ["v1", "v2", "v3"]}`

`$node/a/e/array-node("g")`	`An array node equivalent to the following JSON array:`
	
	`[ "s1", "s2", "s3" ]`

`$node//node("g")`	`An array node, plus a text node for each item in the array.`
	
	`[ "s1", "s2", "s3" ]`
	`"s1"`
	`"s2"`
	`"s3"`

20.3.5 配列および配列メンバーの選択

XPath 式での配列の参照は、デフォルトではシーケンスとして扱われます。つまり、入れ子の配列はデフォルトでは平坦化されるため、`[1, 2, [3, 4]]` が `[1, 2, 3, 4]` として扱われ、`[]` 演算子はシーケンスメンバー値を返します。

シーケンスとしてではなく、配列として配列にアクセスするには、`array-node()` 演算子を使用します。関連ノードではなく、配列内の値にアクセスするには、`data()` 演算子を使用します。

ノード型が明確でない場合は、「descendant-or-self」軸（「//」）で、配列ノードと配列項目の両方を選択できます。例えば、ドキュメントが次のような形式であるとします。

```
{ "a" : [ 1, 2] }
```

XPath 式 `//node("a")` は、項目値 1 および 2 に対して、配列ノードと 2 つの数値ノードの両方を選択します。

次の JSON オブジェクトがインメモリオブジェクト `$node` 内に存在するとします。

```
{
   「a": [ 1, 2 ],
  "b": [ 3, 4, [ 5, 6 ] ],
  "c": [
    { "c1": "cv1" },
    { "c2": "cv2" }
  ]
}
```

配列と配列メンバーにアクセスする XPath 式の例を次の表に示します。「XPath の例」（350 ページ）で説明するパターンを使用して、これらの例を Query Console で試してみてください。

XPath 式	結果

`$node/a`	`A sequence of number nodes:`
	
	`(number-node{1}, number-node{2})`

`$node/a/data()`	`A sequence of numbers:`
	
	`(1, 2)`

`$node/array-node("a")`	`An array-node with number nodes children, equivalent to the following JSON array:`
	
	`[1, 2]`

`$node/node("a")`	`An array-node with number nodes children, equivalent to the following JSON array:`
	
	`[1, 2]`

XPath 式	結果
`$node//node("a")`	A sequence of nodes consisting of an array node, and a number node for each item in "a".
`$node/a[1]`	number-node{1}
`$node/a[1]/data()`	1
`$node/b/data()`	A sequence of numbers.The inner array is flattened when the value is converted to a sequence. (3, 4, 5, 6)
`$node/array-node("b")`	All array nodes with name "b".Equivalent to the following JSON array: [3, 4, [5, 6]]
`$node/array-node("b")/array-node()`	All array nodes contained inside the array named "b".Equivalent to the following JSON array: [5, 6]
`$node/b[3]`	number-node{5}
`$node/c`	An object node equivalent to the following JSON object: ({"c1": "cv1"}, {"c2": "cv2"})
`$node/c[1]`	An object node equivalent to the following JSON: { "c1": "cv1" }
`$node//array-node()/number-node()[data()=2]`	All number nodes with the value 2 that are children of an array node. number-node{2}

XPath 式	結果

`$node//array-node()[number-node()/data()=2]`	`All array nodes that contain a number node with a value of 2.`
	
	`[1, 2]`

`$node//array-node()[./node()/text() = "cv2"]`	`All array nodes that contain a member with a text value of "cv2".`
	
	`[ {"c1": "cv1"}, {"c2": "cv2"} ]`

20.4 JSON ドキュメントに対するインデックスおよびレキシコンの作成

JSON ドキュメントに対してパス、レンジ、およびフィールドインデックスを作成できます。インデックス付けに関しては、JSON プロパティ（名前 / バリューペア）は XML 要素とほぼ同等です。例えば、JSON プロパティレンジインデックスを作成するには、XML 要素レンジインデックスを作成するための API とインターフェイスを使用します。

JSON ドキュメントのインデックス付けは、XML ドキュメントのインデックス付けと次の点で異なります。

- JSON の文字列値は、XML テキストノードの場合と同様に、テキストノードとして表現され、テキストとしてインデックス付けされます。ただし、JSON の数値、ブール値、および null 値はテキストとしてインデックス付けされず、別々にインデックス付けされます。
- 各 JSON 配列メンバー値は、関連付けられたプロパティの値とみなされます。例えば、`{"a":[1,2]}` が含まれるドキュメントは、値 1 のプロパティ「a」の値クエリ、および値 2 のプロパティ「a」の値クエリと一致します。
- JSON ドキュメントにはフラグメントルートを定義できません。
- JSON ドキュメントには、phrase-through と phrase-around のいずれも定義できません。
- JSON ドキュメント内で言語を切り替えることはできません。また、JSON ドキュメントを読み込むときに、`xdmp:document-load`（XQuery）または `xdmp.documentLoad`（JavaScript）の `default-language` オプションは無視されます。
- JSON オブジェクトノードには文字列値は定義されません。つまり、フィールド値およびフィールドレンジクエリは、オブジェクトノード内にはトラバースしません。詳細については、「JSON と XML のフィールドクエリの相違点」（358 ページ）を参照してください。

詳細については、『Administrator's Guide』の Range Indexes and Lexicons を参照してください。

20.5 JSON と XML のフィールドクエリの相違点

XML と JSON の両方でフィールド語クエリの動作は同じですが、インデック付けに違いがあるため（「JSON ドキュメントに対するインデックスおよびレキシコンの作成」（357 ページ）を参照）、JSON と XML では、フィールド値クエリおよびフィールドレンジクエリの動作が異なります。

複雑な XML ノードには、インデックス付けのための文字列値があります。これは、すべての子孫ノードのテキストノードの連結です。JSON オブジェクトノードに等価な文字列値はありません。

例えば、XML では、フィールドがパス `/name` で定義されており、「middle」が除外される場合、「John Smith」のフィールド値クエリは次のドキュメントに一致します。含まれるテキストノードが連結されるため、次のドキュメントではフィールドの値が「John Smith」となります。

```
<name>
  <first>John</first>
  <middle>NMI</middle>
  <last>Smith</last>
<name>
```

JSON では、連結がないため、同様に動作するフィールドは作成できません。次の JSON ドキュメントの同じフィールドには、値「John」および「Smith」がありますが、「John Smith」はありません。

```
{ "name": {
    "first": "John",
    "middle": "NMI",
    "last": "Smith"
}
```

また、フィールド値およびフィールドレンジクエリは、JSON オブジェクトノード内にはトラバースしません。例えば、パスフィールド「myName」がパス `/a/b` に対して定義されている場合、次のクエリはドキュメント「my.json」に一致します。

```
xdmp:document-insert("my.json",
  xdmp:unquote('{"a": {"b": "value"}}'));
cts:search(fn:doc(), cts:field-value-query("myField",
"value"));
```

ただし、次のクエリは「my.json」に一致しません。`/a/b` はオブジェクトノード（`{"c":"example"}`）であり、文字列値ではないためです。

```
xdmp:document-insert("my.json",
  xdmp:unquote('{"a": {"b": {"c": "value"}}}'));
cts:search(fn:doc(), cts:field-value-query("myField",
"value"));
```

フィールドの詳細については、『Administrator's Guide』の Overview of Fields を参照してください。

20.6 地理情報データ、テンポラルデータ、およびセマンティックデータの表現

MarkLogic サーバーによる JSON ドキュメントの地理情報データ、テンポラルデータ、およびセマンティックデータのサポートを利用するには、データを特定の方法で表現する必要があります。

- 地理情報データ
- 日付 / 時間データ
- セマンティックデータ

20.6.1 地理情報データ

地理情報データは、ポイントや地域を定義する、緯度と経度による座標の集まりを表現します。インデックスを定義し、地理情報の値に対するクエリを実行できます。地理情報データは、MarkLogic で認識されるいずれかの座標系を使用する必要があります。

ポイントは、JSON で次のように表現できます。

- GeoJSON オブジェクト内の座標。http://geojson.org を参照してください。例：{"geometry": {"type": "Point", "coordinates": [37.52, 122.25]}}
- 値が数値の配列である JSON プロパティ。先頭のメンバー 2 つが緯度と経度（またはその逆）を表現し、残りのメンバーはすべて無視されます。例えば、次のオブジェクトの `coordinates` プロパティの値です。{"location": {"desc": "somewhere", "coordinates": [37.52, 122.25]}}
- JSON プロパティのペア。値が緯度を表すプロパティと、値が経度を表すプロパティです。例：`{"lat": 37.52, "lon": 122.25}`
- スペースで区切られた 2 つの数値が含まれる文字列。例：`"37.52 122.25"`。

JSON ドキュメント内の地理情報データにインデックスを作成できます。また、`cts:json-property-geospatial-query`、`cts:json-property-child-geospatial-query`、`cts:json-property-pair-geospatial-query`、`cts:path-geospatial-query`

（または同等の JavaScript）などのクエリを使用して、地理情報データを検索できます。Node.js、Java、および REST クライアント API も同様のクエリをサポートします。

2D ポイントのみがサポートされます。

GeoJSON の地域は、すべて同じ構造です（`type` および `coordinates` プロパティ）。地域の種類（ポイントと多角形など）を区別するのは、type プロパティのみです。そのため、GeoJSON データのインデックスを定義する場合、通常は、パス式の `type` に述語を含む地理情報パスレンジインデックスを使用してください。

例えば、GeoJSON ポイント（`"type": "Point"`）のみを対象とするインデックスを定義する場合は、インデックスの定義時に、次のようなパス式を使用できます。そして、`cts:path-geospatial-query` または同等の構造化クエリを使用して検索します（『*Search Developer's Guide*』の geo-path-query を参照）。

```
/whatever/geometry[type="Point"]/array-node("coordinates")
```

20.6.2 日付 / 時間データ

MarkLogic サーバーでは、テンポラル（時間）データ管理、階層型ストレージ、レンジインデックスなどの機能で、date、time、および dateTime データ型を使用します。

認識される日時形式の JSON 文字列値を、XML の同等のテキストと同じコンテキストで使用できます。MarkLogic サーバーは、XML スキーマ（ISO-8601 規則に基づく）で定義された日時形式を認識します。詳細については、次のドキュメントを参照してください。

http://www.w3.org/TR/xmlschema-2/#isoformats

テンポラルデータ型でレンジインデックスを作成するには、データが ISO-8601 標準の XSD データ形式で JSON ドキュメント内に格納されている必要があります。例えば、JSON ドキュメントに次の形式のデータが含まれているとします。

```
{ "theDate" : 「2014-04-21T13:00:01Z" }
```

このとき、`dateTime` の `theDate` で「element」型の要素レンジインデックスを定義でき、テンポラルデータを単に文字列として扱うのではなく、その特性を利用したクエリを `theDate` に対して実行できます。

20.6.3 セマンティックデータ

セマンティックトリプルは、『*Semantics Developer's Guide*』の Supported RDF Triple Formats で説明するいずれかの形式でデータベースに読み込むことができます（RDF/JSON を含む）。

JSON ドキュメントの組み込みトリプルは、次の形式である場合にインデックス化されます。

```
{ "triple": {
    "subject": IRI_STRING,
    "predicate": IRI_STRING,
    "object": STRING_PRESENTATION_OF_RDF_VALUE
} }
```

例：

```
{
  「my" : "data",
   "triple" : {
     "subject": "http://example.org/ns/dir/js",
     "predicate": "http://xmlns.com/foaf/0.1/firstname",
     "object": {"value": "John", "datatype": "xs:string"}
   }
}
```

詳細については、『*Semantics Developer's Guide*』の Loading Semantic Triples を参照してください。

20.7 ドキュメントのプロパティ

JSON ドキュメントでは、ドキュメントのプロパティのフラグメントを使用できますが、ドキュメントのプロパティは XML 形式である必要があります。

20.8 大きな整数値のシリアライゼーション

MarkLogic では、JSON がサポートする値よりも大きい整数値を表現できます。例えば、xs:unsignedLong XSD 型には、JSON では整数として表現できない値が含まれます。

MarkLogic が JSON で表現するには大きすぎる xs:unsignedLong 値をシリアライズした場合、文字列としてシリアライズされます。それ以外の場合は、値が数値としてシリアライズされます。つまり、同じ操作でも、入力によって、結果が文字列値になる場合と数値になる場合があります。

例えば、次のコードは、数値であるプロパティ値 1 つと文字列であるプロパティ値 1 つを持つ JSON オブジェクトを生成します。

```
xquery version "1.0-ml";
object-node {
  "notTooBig": 1111111111111,
  "tooBig":11111111111111111
}
```

このコードによって生成されるオブジェクトノードは次のようになります。ここで、「notTooBig」は数値ノード、「tooBig」はテキストノードです。

```
{"notTooBig":1111111111111, "tooBig":"11111111111111111"}
```

シリアライズされた JSON データに、大きい数値が含まれる可能性がある場合、この JSON データを操作するコードは、この可能性を明示する必要があります。

20.9 XQuery における JSON の使用

このセクションでは、XQuery を使用して JSON ドキュメントを操作するためのヒントと例について説明します。次の内容が含まれます。

- JSON ノードの構築
- マップからの JSON オブジェクトの構築
- fn:data との関係
- JSON ドキュメントの操作
- 例：JSON ドキュメントの更新
- JSON ドキュメントの検索

Java、JavaScript、および REST を使用した JSON ドキュメントを操作するためのインターフェイスも用意されています。詳細については、次のガイドを参照してください。

- 『*JavaScript Reference Guide*』
- 『*Node.js Application Developer's Guide*』
- 『*Developing Applications With the Java Client API*』
- 『*REST Application Developer's Guide*』

20.9.1 JSON ノードの構築

JSON オブジェクトおよびリストを構築するために、次の要素コンストラクタをを使用できます。

- `object-node`
- `array-node`
- `number-node`
- `boolean-node`
- `null-node`
- `text`

各コンストラクタは、JSON ノードを作成します。コンストラクタを別のコンストラクタの入れ子にして、任意の複雑な構造を構築できます。JSON プロパティの名前および値は、リテラルまたは XQuery 式です。

次の表は、JSON コンストラクタ式とそれに対応するシリアライズされた JSON の例です。

JSON	コンストラクタ式
`{ "key": "value" }`	`object-node { "key" : "value" }`
`{ "key" : 42 }`	`object-node {"key" : 42 }` `object-node { "key" : number-node { 42 } }`
`{ "key" : true }`	`object-node { "key" : fn:true() }` `object-node { "key" : boolean-node { "true" } }`
`{ "key" : null }`	`object-node { "key" : null-node { } }`

`{ "key" : {`
`"child1" : "one",`
`"child2" : "two"`
`} }`
	`object-node {`
	`"key" : object-node {`
	`"child1" : "one",`
	`"child2" : "two"`
	`}`
	`}`

`{ "key" : [1, 2, 3] }`	`object-node { "key" : array-node { 1, 2, 3 } }`

`{ "date" : "2014/06/24"`	`object-node {`
	`"date" :`
	`fn:format-date(`
	`fn:current-date(),"[M01]/[D01]/[Y01]")`
	`}`

JSON ノードは、`xdmp:unquote` を使用して文字列から作成できます。例えば、次のコードは、{"a": 「b"} が含まれる JSON ドキュメントを作成します。

```
xdmp:document-insert("my.json", xdmp:unquote('{"a": 「b"}'))
```

また、コンストラクタで作成できるすべてのノード型を入力として受け入れる `xdmp:to-json` や、名前 / バリューペアの `map:map` 表現を使用して JSON ドキュメントノードを作成することもできます。詳細については、「マップからの JSON オブジェクトの構築」（364 ページ）および「低レベル JSON XQuery API とプリミティブ型」（377 ページ）を参照してください。

```
xquery version "1.0-ml";
let $object := json:object()
let $array := json:to-array((1, 2, "three"))
let $dummy := (
  map:put($object, "name", "value"),
  map:put($object, "an-array", $array))
return xdmp:to-json($object)
==> {"name":"value", "an-array": [1,2,"three"]}
```

20.9.2 マップからの JSON オブジェクトの構築

map:map を構築してから、そのマップに xdmp:to-json を適用することで、JSON オブジェクトルートノードを持つ JSON ドキュメントを作成できます。このアプローチは、一部のコンテキストでは、JSON ノードコンストラクタを使用するよりも簡単です。

例えば、次のコードは、JSON オブジェクトが含まれるドキュメントノードを作成します。この JSON オブジェクトには、アトミック型のプロパティ（「a」）が 1 つ、配列型のプロパティ（「b」）が 1 つ、object-node 型のプロパティが 1 つあります。

```
xquery version "1.0-ml";
let $map := map:map()
let $_ := $map => map:with('a', 1)
               => map:with('b', (2,3,4))
               => map:with('c', map:map()
                           => map:with('c1', 'one')
                           => map:with('c2', 'two'))
return xdmp:to-json($map)
```

このコードは、次の JSON ドキュメントノードを生成します。

```
{ "a":1,
  "b":[2, 3, 4],
  "c":{"c1":"one", "c2":"two"}
}
```

json:object は、map:map の特別な型であり、JSON オブジェクトを表します。マップ操作と json:* 関数を結合できます。次の例は、json:* 関数（json:object、json:to-array など）と map:map 操作（map:with）の両方を使用します。

```
xquery version "1.0-ml";
let $object := json:object()
let $array := json:to-array((1, 2, "three"))
let $_ := (
  map:put($object, "name", "value"),
  map:put($object, "an-array", $array))
return xdmp:to-json($object)
```

このコードは、次の JSON ドキュメントノードを生成します。

```
{"name":"value", "an-array": [1,2,"three"]}
```

代わりに JSON ノードコンストラクタを使用するには、「JSON ノードの構築」（362 ページ）を参照してください。

20.9.3 fn:data との関係

アトミック型を表現する JSON ノード（number-node、boolean-node、text-node、null-node など）で fn:data を呼び出すと、値が返されます。object-node または array-node で fn:data を呼び出すと、そのノード型の XML 表現（それぞれ <json:object/> または <json:array/> 要素）が返されます。

呼び出しの例	結果

fn:data(
  object-node
  {"a": "b"}
)
	<json:object ...
	  xmlns:json="http://marklogic.com/
	  xdmp/json">
	  <json:entry key="a">
	  <json:value>b</json:value>
	  </json:entry>
	</json:object>

fn:data(
  array-node {(1,2)}
)
	<json:array ...
	  xmlns:json="http://marklogic.com/xdmp
	  /json">
	  <json:value xsi:type="xs:integer">
	  1</json:value>
	  <json:value xsi:type="xs:integer">
	  2</json:value>
	</json:array>

fn:data(number-node { 1 })	1
fn:data(boolean-node { true })	true
fn:data(null-node { })	()

この動作は、Query Console で次のようなクエリを使用して調べることができます。

```
xquery version "1.0-ml";
xdmp:describe(
  fn:data(
    array-node {(1,2)}
))
```

上記の例では、fn:data 呼び出しが xdmp:describe にラップされ、インメモリ型をより正確に表現します。xdmp:describe ラッパーを省略すると、表示するための値のシリアライゼーションによって型が不明瞭になることがあります。例えば、配列の例では、<json:array/> ノードではなく、xdmp:describe ラッパーを削除すると、[1,2] が返されます。

20.9.4 JSON ドキュメントの操作

JSON ドキュメントの作成、読み取り、更新、および削除を行うには、他のドキュメントタイプで使用する場合と同じ関数を使用します。例えば、次のビルトイン関数を使用します。

- xdmp:document-insert
- xdmp:document-load
- xdmp:document-delete
- xdmp:node-replace
- xdmp:node-insert-child
- xdmp:node-insert-before

JSON ノードをプログラムで構築するには、ノードコンストラクタを使用します。詳細については、「JSON ノードの構築」（362 ページ）を参照してください。

注： オブジェクトノードに挿入されるノードには名前が必要です。配列ノードに挿入されるノードには、名前は不要です。

シリアライズされた JSON をデータベースに挿入するためにノードに変換するには、xdmp:unquote を使用します。例：

```
xquery version "1.0-ml";
let $node := xdmp:unquote('{"name" : "value"}')
return xdmp:document-insert("/example/my.json", $node)
```

同様のドキュメント操作を、Java、JavaScript、および REST API を使用して実行できます。JSON ドキュメントをデータベースに読み込むために mlcp コマンドを使用することもできます。

20.9.5 例：JSON ドキュメントの更新

次の表は、xdmp:node-replace、xdmp:node-insert、xdmp:node-insert-before、および xdmp:node-insert-after を使用した JSON ドキュメントの更新の例を示しています。同様の機能を、他の言語インターフェイス（JavaScript、Java、REST など）でも使用できます。

次の表は、JSON ドキュメントの更新の例を示しています。

更新操作	結果	

Replace a string value in a name-value pair.
 	
xdmp:node-replace(
  fn:doc("my.json")/a/b,
  text { "NEW" }
)
	更新前	{"a":{"b":"OLD"}}

	更新後	{"a":{"b":"NEW"}}

Replace a string value in an array.
 	
xdmp:node-replace(
  fn:doc("my.json")/a[2],
  text { "NEW" }
)
	更新前	{"a": ["v1","OLD","v3"] }

	更新後	{"a": ["v1", "NEW", "v3"] }

Insert an object.
 	
xdmp:node-insert-child(
  fn:doc("my.json")/a,
  object-node {"c": "NEW" }/c
)
	更新前	{"a": {"b":"val"} }

	更新後	{ "a": {"b":"val","c":"NEW"} }

Insert an array member.
 	
xdmp:node-insert-child(
  fn:doc("my.json")/array-node("a"),
  text { "NEW" }
)
	更新前	{ "a": ["v1", "v2"] }

	更新後	{ "a": ["v1", "v2", "NEW"] }

更新操作	結果	

Insert an object before another node.
 	
xdmp:node-insert-before(
  fn:doc("my.json")/a/b,
  object-node { "c": "NEW" }/c
)
	更新前	{ "a": {"b":"val"} }

	更新後	{ "a": {"c":"NEW","b":"val"} }

Insert an array member before another member.
 	
xdmp:node-insert-before(
  fn:doc("my.json")/a[2],
  text { "NEW" }
)
	更新前	{ "a": ["v1", "v2"] }

	更新後	{ "a": ["v1", "NEW", "v2"] }

Insert an object after another node.
 	
xdmp:node-insert-after(
  fn:doc("my.json")/a/b,
  object-node { "c": "NEW" }/c
)
	更新前	{ "a": {"b":"val"} }

	更新後	{ "a": {"b":"val","c":"NEW"} }

Insert an array member after another member.
 	
xdmp:node-insert-after(
  fn:doc("my.json")/a[2],
  text { "NEW" })
	更新前	{ "a": ["v1", "v2"] }

	更新後	{ "a": ["v1", "v2", "NEW"] }

オブジェクトを他のオブジェクトに挿入するときは、名前付きオブジェクトノードをノード操作に渡す必要があります。つまり、object-node {"c": "NEW"} という形式のノードを挿入する場合は、その式を、xdmp:node-insert-child のような操作に直接渡すことはできません。代わりに、関連付けられた名前付きノード object-node {"c": 「NEW"}/**c** を渡す必要があります。

例えば、fn:doc("my.json")/a/b の対象がオブジェクトノードである場合、次のコードは XDMP-CHILDUNNAMED エラーを生成します。

```
xdmp:node-insert-after(
  fn:doc("my.json")/a/b,
  object-node { "c": "NEW" }
)
```

20.9.6 JSON ドキュメントの検索

検索は、全般的に JSON と XML の両方のコンテンツで同じ動作になりますが、ここで説明する例外があります。このセクションでは、次の検索関連トピックについて説明します。

- 使用可能な cts クエリ関数
- cts クエリのシリアライゼーション

JSON ドキュメントの検索には、クライアント API による文字列クエリ、構造化クエリ、および QBE を使用することもできます。詳細については、次を参照してください。

- 『*Search Developer's Guide*』
- 『*Node.js Application Developer's Guide*』
- 『*Java Application Developer's Guide*』
- 『*MarkLogic REST API Reference*』

20.9.6.1 使用可能な cts クエリ関数

JSON ドキュメント内の名前 / バリューペアは、プロパティと呼ばれます。次のクエリコンストラクタおよび `cts:search` を使用して、JSON プロパティの CTS クエリを実行できます。

- `cts:json-property-word-query`
- `cts:json-property-value-query`
- `cts:json-property-range-query`
- `cts:json-property-scope-query`
- `cts:json-property-geospatial-query`
- `cts:json-property-child-geospatial-query`
- `cts:json-property-pair-geospatial-query`

次のレキシコン関数も使用できます。

- `cts:json-property-words`
- `cts:json-property-word-match`
- `cts:values`
- `cts:value-match`

JSON インデックスリファレンスのコンストラクタ（`cts:json-property-reference` など）も利用できます。

search API および MarkLogic クライアント API（REST、Java、Node.js）も、文字列クエリ、構造化クエリ、および QBE を使用した JSON ドキュメントに対するクエリをサポートしています。詳細については、次を参照してください。

- 『*Node.js Application Developer's Guide*』の Querying Documents and Metadata
- 『*Java Application Developer's Guide*』の Searching
- 『*Search Developer's Guide*』
- 『*REST Application Developer's Guide*』の Using and Configuring Query Features

JSON ドキュメントでインデックスおよびレキシコンを作成するときは、XML 要素でインデックスおよびレキシコンを作成するためのインターフェイスを使用します。詳細については、「JSON ドキュメントに対するインデックスおよびレキシコンの作成」（357 ページ）を参照してください。

20.9.6.2 cts クエリのシリアライゼーション

CTS クエリは、XML または JSON としてシリアライズできます。適切な形式は、親ノードおよび呼び出し元言語に基づいて選択されます。

親ノードが XML 要素ノードである場合、クエリは XML としてシリアライズされます。親ノードが JSON オブジェクトまたは配列ノードである場合、クエリは JSON としてシリアライズされます。それ以外の場合は、クエリは呼び出し元言語に基づいてシリアライズされます。つまり、JavaScript から呼び出す場合は JSON として、それ以外の場合は XML としてシリアライズされます。

JSON クエリプロパティの値が配列で、その配列が空である場合は、シリアライズされたクエリからプロパティが省略されます。プロパティの値が、1 項目のみを含む配列である場合でも、配列としてシリアライズされます。

20.10 サーバーサイド JavaScript における JSON の使用

サーバーサイド JavaScript から JSON ドキュメントを操作するときは、通常、コンテンツを JavaScript オブジェクトとして操作できます。それでも、「MarkLogic が JSON ドキュメントを表現する方法」（348 ページ）で説明するドキュメントモデルを認識しておく必要があります。

JavaScript からデータベース内の JSON ドキュメントにアクセスすると、イミュータブルドキュメントノードを取得します。これを変更するには、そのノードで `toObject` を呼び出す必要があります。例：

```
declareUpdate();

const theDoc = cts.doc('my.json');
theDoc.a = 1;  // error
```

```
// create a mutable in-memory copy of the document
let mutable = theDoc.toObject();
mutableDoc.a = 1;
xdmp.documentInsert('my.json', mutableDoc);
```

ドキュメントコンテンツをトラバースできることは必要であるが、変更する必要はない場合は、代わりに `root` プロパティを使用できます。この場合は、ドキュメントのコピーが作成されません。例：

```
let myValue = theDoc.root.a;
```

詳細については、『*JavaScript Reference Guide*』を参照してください。

20.11 JSON から XML へ、および XML から JSON への変換

MarkLogic API を使用すると、JSON ドキュメントと XML 間において、セマンティックの意味を失うことなく、シームレスに効率よく変換できます。このセクションでは、これらの変換を実行する方法について説明します。以下のように構成されています。

JSON XQuery ライブラリモジュールは、JSON および XML ドキュメントの相互変換を行います。高速に変換するためには、基盤となる低レベル API を使用します（「低レベル JSON XQuery API とプリミティブ型」（377 ページ）を参照）。この章では、XQuery ライブラリを使用する方法について説明します。以下のように構成されています。

- 変換の方針
- JSON から XML への変換
- カスタム変換のための設定方式について
- 例：basic 方式を使用した変換
- 例：full 方式を使用した変換
- 例：custom 方式を使用した変換

20.11.1 変換の方針

MarkLogic の JSON 変換機能の動作を理解するには、変換の設計時に考慮された次の目標を理解することが役立ちます。

- デフォルトの変換パラメータを使用して、シンプルな変換を簡単かつ高速に実行します。
- カスタム変換を実行できるようにし、カスタム JSON やカスタム XML を、出力または入力として使用できるようにします。
- JSON ドキュメントに対して、高速なキー / 値ルックアップと詳細な検索の両方を実行できるようにします。
- セマンティックがロスレスな変換を実行できるようにします。

これらの目標に合わせて、高速かつ簡単に変換できるようにデフォルト値が設定されています。カスタム変換は可能ですが、若干の作業が必要です。

20.11.2 XML と JSON の間の変換を行う関数

20.11.3 JSON から XML への変換

JSON から XML に変換する主な関数は次のとおりです。

- `json:transform-from-json`

XML から JSON へ変換する主な関数は次のとおりです。

- `json:transform-to-json`

詳細については、以降の各セクションを参照してください。

- 「例：basic 方式を使用した変換」（373 ページ）
- 「例：full 方式を使用した変換」（374 ページ）
- 「例：custom 方式を使用した変換」（375 ページ）

20.11.4 カスタム変換のための設定方式について

JSON 変換では、3 つの方式を使用できます。

- `basic`
- `full`
- `custom`

「方式」とは、変換の動作を JSON 変換ライブラリに指示する設定です。`basic` 変換方式は、最初に JSON を XML に変換し、JSON と XML の間の変換を繰り返した後、JSON に戻す場合に使用します。`full` 方式は、最初に XML を JSON に変換した後、XML に戻す場合に使用します。`custom` 方式では、JSON や XML の出力をカスタマイズできます。

`basic` 方式以外の方式を使用する場合は、次の関数を使用して設定オプションを設定および確認できます。

- `json:config`
- `json:check-config`

`custom` 方式の場合は、要件に合わせて変換を調整できます。変換を制御するために設定できるプロパティの詳細については、『*MarkLogic XQuery and XSLT Function Reference*』の `json:config` を参照してください。

20.11.5 例：basic 方式を使用した変換

次のコードは、basic（デフォルト）方式を使用して、JSON 文字列を XML に変換してから JSON に戻します。JSON オブジェクトまたは配列ノードを渡すこともできます。

```
xquery version '1.0-ml';
import module namespace json =
    "http://marklogic.com/xdmp/json" at "/MarkLogic/json/
      json.xqy";

declare variable $j :=  '{
    "blah":"first value",
    "second Key":["first item","second item",null,
       "third item",false],
    "thirdKey":3,
    "fourthKey":{"subKey":"sub value",
                 "boolKey" : true, "empty" : null }
    ,"fifthKey": null,
    "sixthKey" : []
}'  ;

let $x := json:transform-from-json( $j )
let $jx := json:transform-to-json( $x )
return ($x, $jx)

=>
<json type="object"
     xmlns="http://marklogic.com/xdmp/json/basic">
  <blah type="string">first value</blah>
  <second_20_Key type="array">
    <item type="string">first item</item>
    <item type="string">second item</item>
    <item type="null"/>
    <item type="string">third item</item>
    <item type="boolean">false</item>
  </second_20_Key>
  <thirdKey type="number">3</thirdKey>
  <fourthKey type="object">
    <subKey type="string">sub value</subKey>
    <boolKey type="boolean">true</boolKey>
    <empty type="null"/>
  </fourthKey>
  <fifthKey type="null"/>
  <sixthKey type="array"/>
</json>
```

```
{"blah":"first value",
 "second Key":["first item","second item",null,"third
   item",false],
 "thirdKey":3,
 "fourthKey":{"subKey":"sub value", "boolKey":true,
   "empty":null},
 "fifthKey":null, "sixthKey":[]}
```

20.11.6 例：full 方式を使用した変換

次のコードは、`full` 方式を使用して、XML 要素を JSON 文字列に変換します。full 方式では、一貫性のある方法で命名されたプロパティを持つ JSON 文字列が出力されます。文字列ではなく、JSON オブジェクトノードに XML を変換するには、`json:transform-to-json-object` を使用します。

```
xquery version "1.0-ml";
import module namespace json =
    "http://marklogic.com/xdmp/json" at "/MarkLogic/json/
      json.xqy";

declare variable $doc := document {
<BOOKLIST>
 <BOOKS>
  <ITEM CAT="MMP">
   <TITLE>Pride and Prejudice</TITLE>
   <AUTHOR>Jane Austen</AUTHOR>
   <PUBLISHER>Modern Library</PUBLISHER>
    <PUB-DATE>2002-12-31</PUB-DATE>
    <LANGUAGE>English</LANGUAGE>
    <PRICE>4.95</PRICE>
    <QUANTITY>187</QUANTITY>
    <ISBN>0679601686</ISBN>
    <PAGES>352</PAGES>
    <DIMENSIONS UNIT="in">8.3 5.7 1.1</DIMENSIONS>
    <WEIGHT UNIT="oz">6.1</WEIGHT>
  </ITEM>
 </BOOKS>
</BOOKLIST> } ;

let $c := json:config("full") ,
    $x := map:put($c,"whitespace" , "ignore" ),
    $j := json:transform-to-json(  $doc ,$c ),
    $xj := json:transform-from-json($j,$c)
return ($j, $xj)
```

```
{"BOOKLIST":{"_children":
 [{"BOOKS":{"_children":
  [{"ITEM":{"_attributes":{"CAT":"MMP"},"_children":
   [{"TITLE":{"_children":
    ["Pride and Prejudice"]}},
    {"AUTHOR":{"_children":["Jane Austen"]}},
    {"PUBLISHER":{"_children":["Modern Library"]}},
    {"PUB-DATE":{"_children":["2002-12-31"]}},
    {"LANGUAGE":{"_children":["English"]}},
    {"PRICE":{"_children":["4.95"]}},
    {"QUANTITY":{"_children":["187"]}},
    {"ISBN":{"_children":["0679601686"]}},
    {"PAGES":{"_children":["352"]}},
    {"DIMENSIONS":{"_attributes":{"UNIT":"in"},
      "_children":["8.3 5.7 1.1"]}},
    {"WEIGHT":{"_attributes":{"UNIT":"oz"},
      "_children":["6.1"]}
}]}}]}}]}}
<BOOKLIST>
  <BOOKS>
    <ITEM CAT="MMP">
      <TITLE>Pride and Prejudice</TITLE>
      <AUTHOR>Jane Austen</AUTHOR>
      <PUBLISHER>Modern Library</PUBLISHER>
      <PUB-DATE>2002-12-31</PUB-DATE>
      <LANGUAGE>English</LANGUAGE>
      <PRICE>4.95</PRICE>
      <QUANTITY>187</QUANTITY>
      <ISBN>0679601686</ISBN>
      <PAGES>352</PAGES>
      <DIMENSIONS UNIT="in">8.3 5.7 1.1</DIMENSIONS>
      <WEIGHT UNIT="oz">6.1</WEIGHT>
    </ITEM>
  </BOOKS>
</BOOKLIST>
```

20.11.7 例：custom 方式を使用した変換

次のコードは、custom 方式を使用して、両方向の変換を慎重に制御します。REST クライアント API では、同様の手法を使用して、XML と JSON の間で options ノードを変換します。

```
xquery version "1.0-ml";
import module namespace json = "http://marklogic.com/xdmp
  /json" at "/MarkLogic/json/json.xqy";

declare namespace search=
```

```
    "http://marklogic.com/appservices/search"  ;
declare variable $doc :=
<search:options
xmlns:search="http://marklogic.com/appservices/search">
  <search:constraint name="decade">
    <search:range facet="true" type="xs:gYear">
      <search:bucket ge="1970" lt="1980"
name="1970s">1970s</search:bucket>
      <search:bucket ge="1980" lt="1990"
name="1980s">1980s</search:bucket>
      <search:bucket ge="1990" lt="2000"
name="1990s">1990s</search:bucket>
      <search:bucket ge="2000"
name="2000s">2000s</search:bucket>
      <search:facet-option>limit=10</search:facet-option>
      <search:attribute ns="" name="year"/>
      <search:element ns="http://marklogic.com/wikipedia"
name="nominee"/>
    </search:range>
  </search:constraint>
</search:options>
 ;

let $c := json:config("custom") ,
    $cx := map:put( $c, "whitespace"     , "ignore" ),
    $cx := map:put( $c, "array-element-names" ,
                    xs:QName("search:bucket") ),
    $cx := map:put( $c, "attribute-names",
                    ("facet","type","ge","lt","name","ns" ) ),
    $cx := map:put( $c, "text-value", "label" ),
    $cx := map:put( $c , "camel-case", fn:true() ),
    $j := json:transform-to-json(  $doc ,$c ) ,
    $x := json:transform-from-json($j,$c)
return ($j, $x)

=>
{"options":
 {"constraint":
  {"name":"decade",
   "range":{"facet":true, "type":"xs:gYear",
     "bucket":[{"ge":"1970", "lt":"1980", "name":"1970s",
       "label":"1970s"},
    {"ge":"1980", "lt":"1990", "name":"1980s",
     "label":"1980s"},
    {"ge":"1990", "lt":"2000", "name":"1990s",
     "label":"1990s"},
    {"ge":"2000", "name":"2000s", "label":"2000s"}],
    "facetOption":"limit=10",
```

```
    "attribute":{"ns":"", "name":"year"},
    "element":{"ns":"http:\/\/marklogic.com\/wikipedia",
       "name":"nominee"}
}}}}
<options>
  <constraint name="decade">
    <range facet="true" type="xs:gYear">
      <bucket ge="1970" lt="1980" name="1970s">
       1970s</bucket>
      <bucket ge="1980" lt="1990" name="1980s">
       1980s</bucket>
      <bucket ge="1990" lt="2000" name="1990s">
       1990s</bucket>
      <bucket ge="2000" name="2000s">2000s</bucket>
      <facet-option>limit=10</facet-option>
      <attribute ns="" name="year"/>
      <element ns="http://marklogic.com/wikipedia"
       name="nominee"/>
    </range>
  </constraint>
</options>
```

20.12 低レベル JSON XQuery API とプリミティブ型

MarkLogic サーバーには、いくつかの JSON API が組み込まれています。また、プリミティブ XQuery/XML 型もいくつか組み込まれており、XML と JSON の間での変換に役立ちます。これらの API は、XQuery/XML データモデルと JSON データモデルの間での変換という、負荷の大きい作業を実行します。高レベル JSON ライブラリモジュール関数は、これらの低レベル API を使用します。JSON ライブラリモジュールを使用する場合は、低レベル API を使用する必要はないと考えられます。

このセクションには、次の内容が含まれます。

- 使用可能な関数とプリミティブ型
- 例：JSON ノードへのシリアライゼーション
- 例：JSON ノードの項目リストへのパース

20.12.1 使用可能な関数とプリミティブ型

JSON プロパティのシリアライゼーションに特化した API は 2 つあります。XQuery から JSON にシリアライズするための API と、JSON 文字列を読み取り、その文字列から XQuery データモデルを作成するための API です

- `xdmp:to-json`
- `xdmp:from-json`

これらの API は、データを XQuery でマップとして利用できるようにし、XML データを JSON 文字列としてシリアライズします。ほとんどの XQuery 型は、ラウンドトリップ（JSON にシリアライズされ、JSON から、元の XQuery データモデルの一連の項目にパースされる）できる方法で、損失なしで JSON にシリアライズされますが、損失なしではラウンドトリップされない型もあります。例えば、`xs:dateTime` 値は JSON 文字列にシリアライズされますが、この同じ文字列が元の文字列と同等になるには、XQuery の `xs:dateTime` 値にキャストされて戻される必要があります。高レベル API を使用すると、このような問題の大半に対処できます。

XML データモデルの拡張機能である低レベル API の集まりもあるため、配列やシーケンスのシーケンス（どちらも XML データモデルには存在しない）などのデータ変換を損失なしで実行できます。このようなデータモデル変換をサポートする関数は次のとおりです。

- `json:array`
- `json:array-pop`
- `json:array-push`
- `json:array-resize`
- `json:array-values`
- `json:object`
- `json:object-define`
- `json:set-item-at`
- `json:subarray`
- `json:to-array`

さらに、XQuery/XML データモデルを拡張して、JSON オブジェクト（`json:object`）や JSON 配列（`json:array`）を指定したり、`xdmp:to-json` に渡されるときに `xs:string` を JSON 文字列に簡単にシリアライズできるようにする型（`json:unquotedString`）を指定したりするプリミティブ XQuery 型があります。

JSON との間の変換のパフォーマンスが向上するように、次のビルトインを使用して文字列を XML NCName に変換します。

- `xdmp:decode-from-NCName`
- `xdmp:encode-for-NCName`

低レベル JSON API、サポートする XQuery 関数、およびプリミティブ型は、JSON を消費または生成する効率的で便利なアプリケーションを構築するための構成要素です。これらの API は、JSON と XML の間で変換するために使用されますが、低レベルであるため、あらゆる種類のデータ変換に使用できます。ただし、ほとんどのアプリケーションでは低レベル API を使用する必要はありません。代わりに、XQuery ライブラリ API（お

よびその上位に構築されている REST および Java クライアント API）を使用してください（「JSON から XML へ、および XML から JSON への変換」（371 ページ）を参照）。

各関数のシグネチャおよび説明については、『*MarkLogic XQuery and XSLT Function Reference*』を参照してください。

20.12.2 例：JSON ノードへのシリアライゼーション

次のコードは、マップ、文字列、および整数が含まれる JSON 配列ノードを返します。

```
let $map := map:map()
let $put := map:put($map, "some-prop", 45683)
let $string := "this is a string"
let $int := 123
return
xdmp:to-json(($map, $string, $int))

(:
returns:
[{"some-prop":45683}, "this is a string", 123]
:)
```

マップの詳細については、「map 関数を使用した名前 / バリューマップの作成」（165 ページ）を参照してください。

20.12.3 例：JSON ノードの項目リストへのパース

次のコードを考えてみましょう。これは、前の例の逆です。

```
let $json :=
  xdmp:unquote('[{"some-prop":45683}, "this is a string",
    123]')
return
xdmp:from-json($json)
```

これは、次の項目を返します。

```
json:array(
<json:array xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:json="http://marklogic.com/xdmp/json">
 <json:value>
  <json:object>
   <json:entry key="some-prop">
    <json:value xsi:type="xs:integer">45683
    </json:value>
   </json:entry>
```

```
  </json:object>
 </json:value>
 <json:value xsi:type="xs:string">this is a string
 </json:value>
 <json:value xsi:type="xs:integer">123</json:value>
</json:array>)
```

上記は、json:array XML 要素のシリアライゼーションを示しています。この XML データモデルの一部またはすべての項目を使用することもできます。例えば、次のコードは、他の値に基づいて json:object に追加します（そして、得られた JSON 文字列を出力します）。

```
xquery version "1.0-ml";
let $json :=
  xdmp:unquote('[{"some-prop":45683}, "this is a string",
    123]')
let $items := xdmp:from-json($json)
let $put := map:put($items[1], xs:string($items[3]),
  $items[2])
return
($items[1], xdmp:to-json($items[1]))

(: returns the following json:array and JSON string:
json:object(
<json:object xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:json="http://marklogic.com/xdmp/json">
  <entry key="some-prop">
    <json:value xsi:type="xs:integer">45683</json:value>
  </entry>
  <entry key="123">
    <json:value xsi:type="xs:string">this is a string
      </json:value>
  </entry>
</json:object>)
{"some-prop":45683, "123":"this is a string"}

This query uses the map functions to modify the first
json:object
in the json:array.
:)
```

上記のクエリで、xdmp:from-json 呼び出しから返される最初の項目（$items[1]）は json:array です。これで、map 関数を使用して json:array を変更できます。その後、変更された json:array がクエリから返されます。json:array はマップのように扱えます。主な違いは、json:array は順序付きですが、map:map はそうでは

ありません。マップの詳細については、「map 関数を使用した名前 / バリューマップの作成」（165 ページ）を参照してください。

20.13 JSON ドキュメントの読み込み

このセクションでは、さまざまな MarkLogic ツールおよびインターフェイスを使用した JSON ドキュメントの読み込みの例について説明します。次の内容が含まれます。

- mlcp を使用した JSON ドキュメントの読み込み
- Java クライアント API を使用した JSON ドキュメントの読み込み
- Node.js クライアント API を使用した JSON ドキュメントの読み込み
- REST クライアント API を使用した JSON の読み込み

20.13.1 mlcp を使用した JSON ドキュメントの読み込み

JSON ドキュメントは、XML、バイナリ、およびテキストのドキュメントの場合と同様に、mlcp を使用して読み込むことができます。ファイル拡張子が「.json」である場合、MarkLogic ではコンテンツを自動的に JSON として認識します。

詳細については、『*Loading Content Into MarkLogic Server Guide*』の Loading Content Using MarkLogic Content Pump を参照してください。

20.13.2 Java クライアント API を使用した JSON ドキュメントの読み込み

Java クライアント API を使用すると、Java アプリケーションから MarkLogic サーバーを操作できます。詳細については、Java Application Developer's Guide を参照してください。

Java アプリケーションで JSON ドキュメントを作成するには、クラス `com.marklogic.client.document.DocumentManager` を使用します。ファイル、文字列、Jackson など、Java クライアント API ハンドルインターフェイスでサポートされるあらゆるソースからの入力データを使用できます。詳細については、『*Java Application Developer's Guide*』の Document Creation を参照してください。

Java クライアント API を使用して、POJO ドメインオブジェクトを表現する JSON ドキュメントを作成することもできます。詳細については、『*Java Application Developer's Guide*』の POJO Data Binding Interface を参照してください。

20.13.3 Node.js クライアント API を使用した JSON ドキュメントの読み込み

Node.js クライアント API を使用すると、クライアントサイドコードで JSON データを JavaScript オブジェクトとして処理できます。DatabaseClient.documents インターフェイスを使用して、そのようなオブジェクトからデータベース内に JSON ドキュメントを直接作成できます。

詳細については、『*Node.js Application Developer's Guide*』の Loading Documents into the Database を参照してください。

20.13.4 REST クライアント API を使用した JSON の読み込み

REST クライアント API を使用して、JSON ドキュメントを MarkLogic サーバーに読み込むことができます。次の例は、REST クライアント API を使用して MarkLogic で JSON ドキュメントを読み込む方法を示しています。

次のようなコンテンツを持つ JSON ファイル `test.json` を考えてみます。

```
{
"key1":"value1",
"key2":{
        "a":"value2a",
        "b":"value2b"
       }
}
```

次の `curl` コマンドを実行し、`documents` エンドポイントを使用して JSON ドキュメントを作成します。

```
curl --anyauth --user user:password -T ./test.json -D - \
  -H "Content-type: application/json" \
  http://my-server:5432/v1/documents?uri=/test/keys.json
```

ドキュメントが作成され、エンドポイントによって次の内容が返されます。

```
HTTP/1.1 100 Continue

HTTP/1.1 401 Unauthorized
WWW-Authenticate: Digest realm="public", qop="auth",
  nonce="b4475e81fe81b6c672a5
  d105f4d8662a", opaque="de72dcbdfb532a0e"
Server: MarkLogic
Content-Type: text/xml; charset=UTF-8
Content-Length: 211
Connection: close

HTTP/1.1 100 Continue

HTTP/1.1 201 Document Created
Location: /test/keys.json
Server: MarkLogic
Content-Length: 0
Connection: close
```

これで、次のように REST クライアント API からドキュメントを取得できます。

```
$ curl --anyauth --user admin:password -X GET -D - \
  http://my-server:5432/v1/documents?uri=/test/keys.json
==>
HTTP/1.1 401 Unauthorized
WWW-Authenticate: Digest realm="public", qop="auth",
  nonce="2aaee5a1d206cbb1b894
  e9f9140c11cc", opaque="1dfded750d326fd9"
Server: MarkLogic
Content-Type: text/xml; charset=UTF-8
Content-Length: 211
Connection: close

HTTP/1.1 200 Document Retrieved
vnd.marklogic.document-format: json
Content-type: application/json
Server: MarkLogic
Content-Length: 56
Connection: close

{"key1":"value1", "key2":{"a":"value2a", "b":"value2b"}}
```

REST クライアント API の詳細については、REST Application Developer's Guide を参照してください。

21.0 トリガーを使用したアクションのスポーン

MarkLogic サーバーには、プリコミットトリガーおよびポストコミットトリガーが含まれています。この章では、MarkLogic サーバーにおけるトリガーの仕組みについて説明します。次のセクションから構成されます。

- トリガーの概要
- トリガーと Content Processing Framework
- プリコミットトリガーとポストコミットトリガー
- トリガーイベント
- トリガーのスコープ
- トリガーによって呼び出されるか、スポーンされるモジュール
- triggers.xqy によるトリガーの作成および管理
- シンプルなトリガーの例
- トリガーの無限ループ（トリガーストーム）の回避

21.1 トリガーの概要

概念上、トリガーは、特定のイベント（ドキュメントの作成、削除、更新、データベースのオンライン化など）の発生をリッスンし、そのイベントの発生後に XQuery モジュールを呼び出して実行します。トリガー定義によって、アクションモジュールが実行されるタイミング（トリガーの作動を引き起こすトランザクションのコミットの前または後）が決定されます。

堅牢なトリガーフレームワークの作成は複雑な作業です。特に、トリガーが状態を維持したり、サービス中断からスムーズに復旧したりする必要がある場合は複雑です。カスタムトリガーを作成する前に、Content Processing Framework の使用を検討してください。CPF は、イベント管理の複雑さのほとんどをアプリケーションから取り除く、リッチで信頼性の高いフレームワークを提供します。詳細については、「トリガーと Content Processing Framework」（386 ページ）を参照してください。

21.1.1 トリガーのコンポーネント

トリガー定義は、XML ドキュメントとしてデータベースに格納されます。次の情報が含まれています。

- イベント定義。次の内容を記述します。
  - トリガーが作動する条件
  - 監視対象コンテンツのスコープ

- イベントが発生したときに呼び出されるか、スポーンされる XQuery モジュール

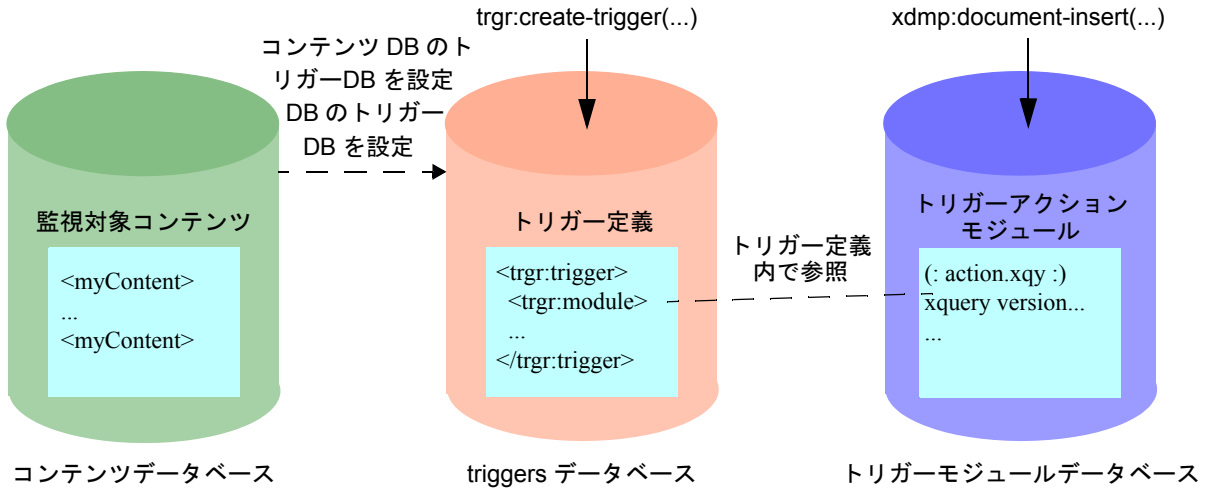
トリガー定義を作成してインストールするには、`trgr:create-trigger` を呼び出します。トリガーイベント定義の詳細については、「トリガーイベント」（388 ページ）を参照してください。

21.1.2　トリガーで使用されるデータベース

完全なトリガーには、監視対象コンテンツ、トリガー定義、およびアクションモジュールが必要です。これらのコンポーネントには、3 つのデータベースが必要です。

- トリガーによって監視されるコンテンツデータベース。
- triggers データベース。`trgr:create-trigger` によってトリガー定義が格納されます。これは、コンテンツデータベース用に設定された triggers データベースである必要があります。
- モジュールデータベース。ここにトリガーアクションモジュールが格納されます。これは、アプリケーションサーバー用に設定されるモジュールデータベースである必要はありません。

次の図は、これらのデータベースとトリガーコンポーネントの関係を示しています。



通常、コンテンツデータベース、triggers データベース、およびモジュールデータベースはそれぞれ、別の物理データベースですが、分離されているという要件はありません。便宜上、Triggers という名前のデータベースが MarkLogic サーバーによってインストールされますが、任意のデータベースをコンテンツデータベース、triggers データベース、またはモジュールデータベースとして使用できます。どのように選択するかは、アプリケーションのニーズによって異なります。

例えば、適用対象のコンテンツとともにトリガーをバックアップする場合は、トリガー定義とそのアクションモジュールをコンテンツデータベースに格納するという方法があります。複数のコンテンツデータベースに適用される複数のトリガーで 1 つのトリガーアクションモジュールを共有する場合は、他のデータベースから分離されているトリガーモジュールデータベースを使用します。

注： ほとんどのトリガー API 関数呼び出しは、triggers データベースのコンテキストで評価される必要があります。

21.2 トリガーと Content Processing Framework

CPF（Content Processing Framework）では、トリガーを使用してイベントをキャプチャした後、コンテンツ処理パイプラインで状態を設定します。このフレームワークがトリガーを作成および管理するため、ユーザーは、パイプラインを設定してアクションモジュールを提供するだけで済みます。

CPF（Content Processing Framework）で使用されるパイプラインでは、1 つの段階（ドキュメントの更新など）が完了するとトリガーが作動し、トリガーで指定されている XQuery モジュールが実行されます。完了すると、パイプラインで次のトリガーが作動し、それが繰り返されます。このようにして、ドキュメントを処理する複雑なパイプラインを作成できます。

ステータス変更パイプラインは、データベースに Content Processing をインストールするときにインストールされます。このパイプラインは、コンテンツ処理アプリケーションに必要なすべてのトリガーを作成して管理します。そのため、コンテンツアプリケーションでトリガーを直接作成したり、管理したりする必要はありません。

独自のトリガーを記述する代わりに Content Processing Framework を使用する場合は、次の点に留意してください。

- アクションは、パイプラインを利用して簡単に連結できます。
- 作成およびインストールが必要なのは、トリガーアクションモジュールのみです。
- 中断からの復旧は、CPF が処理します。
- CPF は自動的に、パイプラインの複数の段階にわたってモジュールが状態を利用できるようにします。

CPF（Content Processing Framework）のステータス変更パイプラインを使用するアプリケーションでは、トリガーを明示的に作成する必要はありません。データベースへの Content Processing のインストールの一部として、パイプラインがトリガーを自動的に作成して管理します。詳細については、マニュアル『*Content Processing Framework Guide*』を参照してください。

21.3 プリコミットトリガーとポストコミットトリガー

トリガーのトランザクショナルセマンティックを設定する方法は 2 つあります。プリコミットとポストコミットです。このセクションでは、それぞれのトリガータイプについて説明します。次の部分から構成されます。

- プリコミットトリガー
- ポストコミットトリガー

21.3.1 プリコミットトリガー

プリコミットトリガーの結果として呼び出されるモジュールは、トリガーを作動させるイベントを生成したトランザクションの一環として評価されます。評価するには、トリガーを起動させるトランザクションが実行しているアプリケーションサーバーでモジュールを呼び出します。ただし、これは、`xdmp:invoke` によるモジュールの呼び出しとは、ある意味で異なります。プリコミットトリガーで呼び出されるモジュールは、トリガーを作動させるドキュメントに対する更新を認識します。

そのため、プリコミットトリガーと、トリガーの呼び出し元であるモジュールは、単一のコンテキストで実行されます。何らかの理由でトリガーが完了しない場合（例外をスローした場合など）、トリガーを作動させるトランザクションを含むトランザクション全体が、トランザクションの評価開始前のポイントにロールバックされます。

このようなトランザクションの整合性は、複数の非同期ステップに分割しても意味のない処理を実行しているときに便利です。例えば、ドキュメントの作成時に作動するトリガーを持つアプリケーションがあり、後続処理がドキュメントの状態を認識できるように初期プロパティがドキュメントに設定されている必要がある場合は、ドキュメントの作成と初期プロパティの設定が単一のトランザクションとして発生することに意味があります。（プリコミットトリガーを使用した）単一のトランザクションでは、プロパティの追加時にエラーが発生した場合に、ドキュメントは作成されませんが、アプリケーションがその障害に対処できます。単一のトランザクションでなかった場合、ドキュメントは作成されるものの、初期プロパティが作成されず、コンテンツ処理アプリケーションにとってその新しいドキュメントの処理方法が不明のままであるという状態になる可能性があります。

21.3.2 ポストコミットトリガー

ポストコミットトリガーの結果としてスポーンされるモジュールは、トリガーを作動させるイベントを生成したモジュールとは別のトランザクションとして評価されます。非同期で実行され、トリガーモジュールで障害が発生した場合に呼び出し元トランザクションをロールバックしません。また、トリガーモジュールが呼び出された場合に、それが完了する保証はありません。

ポストコミットトリガーが XQuery モジュールをスポーンすると、「タスクサーバー」のキューに入れられます。タスクサーバーは、このタスクキューを管理し、タスクを受け取った順で開始します。タスクサーバーには、キューを処理する複数のスレッドがあります。グループごとに 1 つのタスクサーバーがあり、タスクサーバーのパラメータを、管理画面の［Groups］>［*group_name*］>［Task Server］で設定できます。

ポストコミットトリガーは非同期であるため、ポストコミットトリガーを呼び出すコードは、トリガーモジュールに含まれる要素に左右されずにデータの一貫性を維持する必要があります。例えば、CPF（Content Processing Framework）コード内の状態トランザクションがポストコミットトリガーを使用するとします。トリガーを作動させるイベントを開始するコードは、トリガーの呼び出し前にプロパティの状態を更新します。これにより、何らかの理由でトリガーのコードが完了しなかった場合でも、一貫性のある状態が保たれます。各状態は、完了するまでに時間がかかることがあるため、状態処理では非同期処理に多くの利点があります。（ポストコミットトリガーを使用した）非同期

処理を実行すると、パイプラインの処理の途中で何かが発生した場合でも、すでに発生した処理が完全には失われないアプリケーションを構築できます。システムが再度使用可能になると、CPF（Content Processing Framework）は、中断した箇所から処理を続行します。

21.4 トリガーイベント

トリガーイベント定義では、トリガーが作動する条件と、適用先のコンテンツを記述します。トリガーイベントには、データイベントとデータベースイベントの 2 種類があります。トリガーは、次のイベントをリッスンできます。

- ドキュメントの作成
- ドキュメントの更新
- ドキュメントの削除
- 任意のプロパティの変更（`last-modified` や `directory` など、MarkLogic サーバーが制御するプロパティを除く）
- 特定の（名前付き）プロパティの変更
- データベースのオンライン化

21.4.1 データベースイベント

データベースイベントは、データベースのオンライン化イベントのみです。監視対象データベースがオンラインになるとすぐに、データベースのオンライン化イベントのモジュールが実行されます。データベースのオンライン化イベント定義に必要なのは、アクションモジュールを実行するユーザーの名前のみです。

21.4.2 データイベント

データイベントは、ドキュメントとプロパティに対する変更に適用されます。トリガーデータイベントは次の部分で構成されます。

- トリガーのスコープは、イベントの適用対象となるドキュメントの集まりを定義します。この部分を作成するには、`trgr:*-scope` 関数（`trgr:directory-scope` など）を使用します。詳細については、「トリガーのスコープ」（389 ページ）を参照してください。
- コンテンツ条件は、トリガーが作動する操作を定義します（ドキュメントの作成、更新、削除、プロパティの変更など）。この部分を作成するには、`trgr:*-content` 関数（`trgr:document-content` など）を使用します。

  複数の操作を監視するには、複数のトリガーイベントを使用し、複数のトリガーを定義する必要があります。

- タイミングインジケータは、イベント条件にマッチするトランザクションに関連してトリガーアクションが発生するタイミング（プリコミットまたはポストコ

ミット）を定義します。この部分を作成するには、`trgr:*-commit` 関数（`trgr:post-commit` など）を使用します。詳細については、「プリコミットトリガーとポストコミットトリガー」（386 ページ）を参照してください。

イベントが適用されるコンテンツデータベースは、イベントにもトリガー定義にも明示されていません。代わりに、コンテンツデータベース用に設定された triggers データベースを通じて関連付けられます。

トリガーが呼び出すモジュールが、トリガーを作動させるイベントを生成したモジュールよりも前にコミットするか、後にコミットするかは、トリガーがプリコミットトリガーとポストコミットトリガーのどちらであるかによって決まります。MarkLogic サーバーのプリコミットトリガーは、イベントをリッスンし、トランザクションのコミット「前」にトリガーモジュールを呼び出すため、プロセス全体は単一トランザクションとなり、すべてが完了するか、すべてが失敗します（ただし、プリコミットトリガーから呼び出されるモジュールは、トリガーを作動させるイベントからの更新を認識します）。

MarkLogic サーバーのポストコミットトリガーは、イベントのコミット後に開始され、このトリガーがスポーンするモジュールは、ドキュメントを更新したトランザクションとは別のトランザクションで実行されます。例えば、ドキュメントの更新イベントのトリガーは、ドキュメントを更新するトランザクションがデータベースにコミットした「後」に発生します。

ポストコミットトリガーモジュールは、トリガーによるモジュールのスポーンを引き起こしたトランザクション（作成または更新イベントなど）とは別のトランザクションで実行されるため、トリガーモジュールトランザクションは、トランザクションで障害が発生した場合に、ドキュメントの元の状態（つまり、トリガーの作動を引き起こした更新の「前」の状態）に自動的にはロールバックできません。これによって、ドキュメントが一貫性のない状態になる場合は、この状態に対処するためのロジックがアプリケーションに必要です。

プリコミットトリガーおよびポストコミットトリガーの詳細については、「プリコミットトリガーとポストコミットトリガー」（386 ページ）を参照してください。

21.5 トリガーのスコープ

「トリガーのスコープ」は、作成、更新、削除、またはプロパティ変更イベントをリッスンするスコープのことです。スコープは、トリガースコープ値 `document`、`directory`、`collection` のいずれかに対応するデータベースの部分を表します。

`document` トリガースコープは特定のドキュメント URI を指定し、トリガーは、そのドキュメントで指定されたトリガーイベントのみに反応します。

`collection` トリガースコープは特定のコレクション URI を指定し、トリガーは、指定されたコレクション内のドキュメントに指定されたトリガーイベントに反応します。

directory スコープは、指定されたディレクトリ（ディレクトリ自体（深さ 1）、または指定されたディレクトリ自体とその再帰的サブディレクトリ）にあるドキュメントを表します。例えば、ディレクトリスコープの URI が /（フォワードスラッシュ文字）で深さが infinity である場合は、データベース内にある、URI がフォワードスラッシュ文字（/）で始まる任意のドキュメントでは、指定したトリガーイベントの発生時にこのスコープによりトリガーが作動します。このディレクトリ例では、ドキュメント hello.xml はこのトリガースコープに含まれない（/ ディレクトリ内にないため）が、URI が /hello.xml や /mydir/hello.xml であるドキュメントは含まれることに注意してください。

21.6 トリガーによって呼び出されるか、スポーンされるモジュール

トリガー定義は、モジュールの URI を指定します。このモジュールは、トリガーが作動するとき（イベントの完了時）に評価されます。この動作の仕組みは、プリコミットトリガーとポストコミットトリガーで異なります。このセクションでは、トリガーモジュールの呼び出しまたはスポーンで発生する状況について説明します。このセクションは、次の部分から構成されます。

- プリコミットトリガーとポストコミットトリガーによるモジュール動作の相違点
- モジュール外部変数 trgr:uri および trgr:trigger

21.6.1 プリコミットトリガーとポストコミットトリガーによるモジュール動作の相違点

プリコミットトリガーでは、トリガーの作動時（イベントの完了時）にモジュールが呼び出されます。呼び出されたモジュールは、XQuery ステートメントの xdmp:invoke 呼び出しに相当する方法で、呼び出し元 XQuery モジュールと同じアプリケーションサーバーで同期的に評価されます。異なるのは、プリコミットトリガーでは、呼び出されたモジュールが、トリガーを作動させるイベントの結果を認識する点です。例えば、ドキュメントが更新されると作動するように定義されたプリコミットトリガーがあり、ドキュメント内の段落数をモジュールがカウントする場合は、トリガーを作動させた更新の「後」に段落数がカウントされます。さらに、何らかの理由（シンタックスエラーなど）によりトリガーモジュールで障害が発生すると、トリガーを作動させた更新を含むトランザクション全体が、更新前の状態にロールバックされます。

ポストコミットトリガーでは、トリガーが作動すると（イベントの完了時に）、モジュールがタスクサーバー上にスポーンされます。スポーンされたモジュールは、XQuery ステートメントの xdmp:spawn の呼び出しに相当する方法で評価され、タスクサーバー上のモジュールは非同期で評価されます。ポストコミットトリガーモジュールがスポーンされると、評価されるまで、タスクサーバーのキューで待機します。スポーンされたモジュールは、評価されるときに自身のトランザクションとして実行されます。通常は、タスクサーバーのキュー内にあるモジュールは、キューに追加された順序で開始されます。ただし、タスクサーバーのキューは、システムのシャットダウン時に保持されないため、タスクサーバーのキュー内にあるモジュールの実行は保証されません。

21.6.2 モジュール外部変数 trgr:uri および trgr:trigger

モジュールをトリガーするときに使用できる外部変数は 2 つあります。

- `trgr:uri as xs:string`
- `trgr:trigger as node()`

`trgr:uri` 外部変数は、トリガーが作動する原因となったドキュメントの URI です（データイベントによるトリガーでのみ使用でき、データベースオンラインイベントによるトリガーでは使用できない）。`trgr:trigger` 外部変数は、URI `http://marklogic.com/xdmp/triggers/`*trigger_id*（*trigger_id* はトリガーの ID）を使用して triggers データベースに格納されているトリガー XML ノードです。これらの外部変数をトリガーモジュールで使用するには、次のようにプロローグで宣言します。

```
xquery version "1.0-ml";
import module namespace trgr='http://marklogic.com/
       xdmp/triggers' at '/MarkLogic/triggers.xqy';

declare variable $trgr:uri as xs:string external;
declare variable $trgr:trigger as node() external;
```

21.7 triggers.xqy によるトリガーの作成および管理

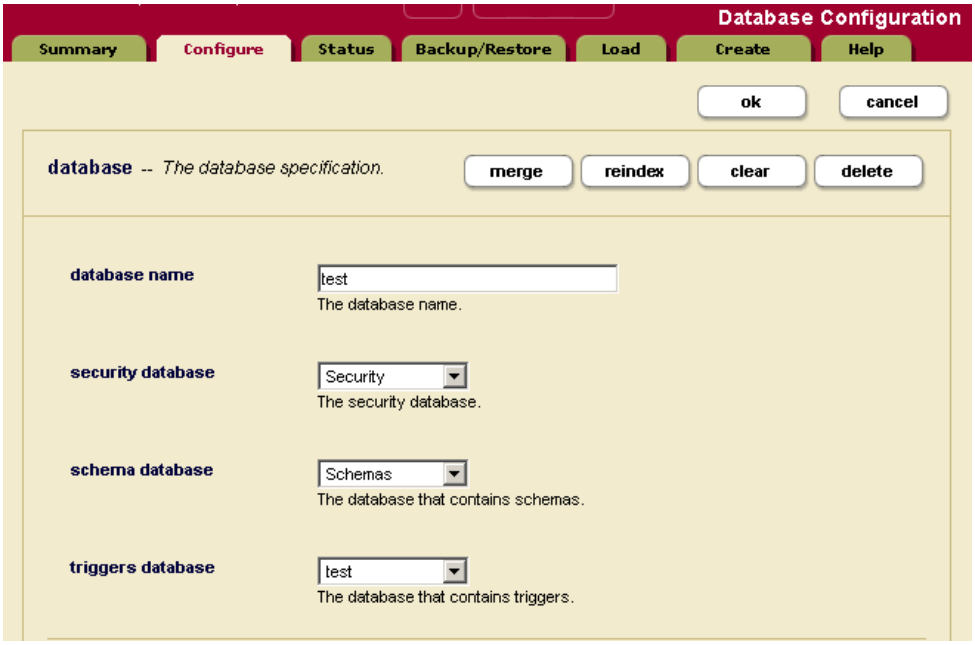
`<install_dir>/Modules/MarkLogic/triggers.xqy` XQuery モジュールファイルには、トリガーを作成、削除、および管理するための関数が含まれています。ステータス変更パイプラインを使用している場合、このパイプラインは、トリガーの詳細をすべて処理します。ユーザーがトリガーを作成したり、管理したりする必要はありません。トリガー関数の詳細については、『*MarkLogic XQuery and XSLT Function Reference*』を参照してください。

トリガーを作成する実際の XQuery コード例については、`<install_dir>/Modules/MarkLogic/cpf/domains.xqy` XQuery モジュールファイルを参照してください。サンプルトリガーの例については、「シンプルなトリガーの例」（392 ページ）を参照してください。このモジュール内にある関数は、管理画面を使用してドメインを作成するときに必要なトリガーを作成するために使用されます。

21.8 シンプルなトリガーの例

次の例は、ドキュメントを作成すると作動するシンプルなトリガーを示しています。

1. 管理画面を使用して、triggers データベースを使用するようにデータベースを設定します。任意のデータベースを triggers データベースとして指定できます。次のスクリーンショットは、test という名前のデータベースが、コンテンツデータベースと triggers データベースの両方で使用される様子を示しています。



2. ディレクトリ `/myDir/` に作成されるドキュメントをリッスンするトリガーを、次の XQuery コードで作成します。このコードは、コンテンツが格納される triggers データベースと照合して評価される必要があることに注意してください。

```
xquery version "1.0-ml";
import module namespace trgr="http://marklogic.com/
       xdmp/triggers" at "/MarkLogic/triggers.xqy";

trgr:create-trigger("myTrigger", "Simple trigger example",
  trgr:trigger-data-event(
      trgr:directory-scope("/myDir/", "1"),
      trgr:document-content("create"),
      trgr:post-commit()),
  trgr:trigger-module(xdmp:database("test"), "/modules/",
"log.xqy"),
  fn:true(), xdmp:default-permissions() )
```

このコードは、トリガーの ID を返します。作成したトリガードキュメントは、URI `http://marklogic.com/xdmp/triggers/`*`trigger_id`* を使用してドキュメントに格納されます。ここで *trigger_id* は、作成したトリガーの ID です。

3. コンテンツがトリガーアクションの XQuery モジュールであるドキュメントを読み込みます。これは、以前に指定したトリガーが作動するとスポーンされるモジュールです。この例では、モジュールの URI がデータベース `test` 内の `/modules/log.xqy`（前出の `trgr:create-trigger` コードの `trgr:trigger-module` 部分）でなければなりません。読み込むドキュメントは XQuery ドキュメントであるため、テキストドキュメントとして読み込まれる必要があり、実行パーミッションを持っていることが必要です。例えば、トリガーアクションが評価されるアプリケーションサーバー用のモジュールデータベースと照合して次の XQuery を評価することで、`test` データベース内にトリガーモジュールを作成します。

```
xquery version '1.0-ml';
(: evaluate this against the database specified
   in the trigger definition (test in this example)
:)
xdmp:document-insert("/modules/log.xqy",
  text{ "
xquery version '1.0-ml';
import module namespace trgr='http://marklogic.com/
     xdmp/triggers' at '/MarkLogic/triggers.xqy';

declare variable $trgr:uri as xs:string external;

xdmp:log(fn:concat('*****Document ', $trgr:uri, ' was
created.*****'))"
}, xdmp:permission('app-user', 'execute'))
```

4. これで、ドキュメントを `/myDir/` ディレクトリのデータベース `test` に作成すると、トリガーが作動します。例えば、次のコードについて考えてみましょう。

```
xdmp:document-insert("/myDir/test.xml", <test/>)
```

このコードは、次のようなメッセージを `ErrorLog.txt` ファイルに書き込みます。

```
2007-03-12 20:14:44.972 Info: TaskServer: *****Document
/myDir/test.xml was created.*****
```

注： この例では、ドキュメントが作成されたときにのみトリガーが作動します。ドキュメントが更新されたときにトリガーを作動させるには、`trgr:document-content` の値を `"modify"` に設定した別のトリガーが必要です。

21.9 トリガーの無限ループ（トリガーストーム）の回避

ドキュメント自体が更新されるトリガーを作成すると、トリガーの作動により無限ループが発生します。これは「トリガーストーム」とも呼ばれます。

プリコミットトリガーが作動すると、そのアクションは、同じトランザクションの一部になります。そのため、トリガーで実行される更新によって、その同じトリガーが再度作動しないようにする必要があります。そうしないと、確実にトリガーストームが発生し、一般には XDMP-MAXTRIGGERDEPTH エラーメッセージが発生します。

次の例では、データベースの /storm/ ディレクトリ内にあるドキュメントが変更されるとモジュールを呼び出すトリガーを作成します。トリガーで呼び出されたモジュールは、新しい子ノードでドキュメントを更新しようとします。これが他のドキュメントの更新をトリガーし、それがさらに他の更新をトリガーし、この動作が無限に繰り返されます。最終的に XDMP-MAXTRIGGERDEPTH エラーメッセージが発生し、ドキュメントは更新されません。

トリガーストームを作成するには、次の手順を実行します。

1. Modules データベースで、トリガーによって呼び出される storm.xqy モジュールを作成します。

```
xquery version "1.0-ml";

import module namespace trgr="http://marklogic.com/
    xdmp/triggers" at "/MarkLogic/triggers.xqy";

if (xdmp:database() eq xdmp:database("Modules"))
   then ()
   else error((), 'NOTMODULESDB', xdmp:database()) ,

xdmp:document-insert( '/triggers/storm.xqy', text {
   <code>
   xquery version "1.0-ml";
   import module namespace trgr='http://marklogic.com/
       xdmp/triggers' at '/MarkLogic/triggers.xqy';

    declare variable $trgr:uri as xs:string external;
    declare variable $trgr:trigger as node() external;

    xdmp:log(text {{
               'storm:',
               $trgr:uri,
               xdmp:describe($trgr:trigger)
               }}) ,
```

```
    let $root := doc($trgr:uri)/*
    return xdmp:node-insert-child(
              $root,
              element storm
              {{ count($root/*) }})
  </code>
} )
```

2. Triggers データベースで、データベースの /storm/ ディレクトリ内にあるドキュメントが変更されるたびに storm.xqy モジュールを呼び出す次のトリガーを作成します。

```
xquery version "1.0-ml";

import module namespace trgr="http://marklogic.com/
    xdmp/triggers" at "/MarkLogic/triggers.xqy";

if (xdmp:database() eq xdmp:database("Triggers"))
   then ()
   else error((), 'NOTTRIGGERSDB', xdmp:database()) ,

trgr:create-trigger(
     "storm",
     "storm",
     trgr:trigger-data-event(trgr:directory-scope("/storm/
", "1"),
     trgr:document-content("modify"),
     trgr:pre-commit()),
     trgr:trigger-module(
         xdmp:database("Modules"),
         "/triggers/",
         "storm.xqy"),
     fn:true(),
     xdmp:default-permissions(),
     fn:true() )
```

3. ここで、Triggers を triggers データベースとして使用する任意のデータベースに、ドキュメントを 2 回挿入します。

```
xquery version "1.0-ml";

xdmp:document-insert('/storm/test', <test/> )
```

4. 2 回目にドキュメントを挿入しようとすると、トリガーが作動します。これにより、XDMP-MAXTRIGGERDEPTH エラーメッセージが生成され、ErrorLog.txt には、次のような繰り返しのメッセージが生成されます。

```
2010-08-12 15:04:42.176 Info: Docs: storm: /storm/test
<trgr:trigger
xmlns:trgr="http://marklogic.com/xdmp/triggers">
  <trgr:trigger-id>13904462711559236l4</trgr:trigger-id>
<trgr:trig...</trgr:trigger>
```

アプリケーションで同様の状態が発生したが、アプリケーションロジックを変更できない場合は、trgr:create-trigger 関数の $recursive パラメータを fn:false() に設定することでトリガーストームを回避できます。その場合、新しいトリガーは次のようになります。

```
trgr:create-trigger(
     "storm",
     "storm",
     trgr:trigger-data-event(trgr:directory-scope("/storm/
", "1"),
     trgr:document-content("modify"),
     trgr:pre-commit()),
     trgr:trigger-module(
         xdmp:database("Modules"),
         "/triggers/",
         "storm.xqy"),
     fn:true(),
     xdmp:default-permissions(),
     fn:false() )
```

この結果、ドキュメントが 1 回更新され、それ以上は更新が繰り返されません。

22.0 ネイティブプラグインの使用

ネイティブプラグインは、C++ で動的に読み込まれるライブラリであり、1 つあるいは複数のプラグイン実装を MarkLogic に提供します。この章では、ネイティブプラグインを作成、インストール、および管理する方法を説明します。

- ネイティブプラグインとは
- MarkLogic サーバーによるネイティブプラグインの管理方法
- ネイティブプラグインライブラリの構築
- ネイティブプラグインのパッケージング
- ネイティブプラグインのインストール
- ネイティブプラグインのアンインストール
- 実行時のネイティブプラグイン登録
- ネイティブプラグインのバージョン管理
- 読込済プラグインのステータスの確認
- プラグインマニフェスト
- ネイティブプラグインのセキュリティ上の考慮事項
- ネイティブプラグインの例

22.1 ネイティブプラグインとは

native plugin とは、1 つあるいは複数の UDF (User Defined Function) 実装を含む、動的にリンクされたライブラリです。想定されている方法でネイティブプラグインをパッケージ化して導入すると、MarkLogic はクラスタ全体にコードを配布し、特定の拡張ポイントで実行できるようにします。

UDF インターフェイスは、ネイティブプラグインを利用できる拡張ポイントを定義します。MarkLogic では、現時点で次の UDF をサポートしています。

- AggregateUDF：MarkLogic の map-reduce 機能を使用して、レキシコンおよびレンジインデックスの値を分析します。詳細については、「集計ユーザー定義関数」（409 ページ）を参照してください。
- LexerUDF：言語のカスタムトークナイザーを定義します。詳細については、『*Search Developer's Guide*』の User-Defined Lexer Plugins を参照してください。
- StemmerUDF：言語のカスタムステミングを定義します。詳細については、『*Search Developer's Guide*』の Using a User-Defined Stemmer Plugin を参照してください。

各 UDF の実装要件はさまざまですが、どれもパッケージング、導入、およびバージョンのネイティブプラグインメカニズムを使用しています。

22.2 MarkLogic サーバーによるネイティブプラグインの管理方法

ネイティブプラグインは、アプリケーションからの参照時に MarkLogic サーバーがオンデマンドで読み込む、動的に読み込まれるライブラリとして導入されます。ネイティブプラグインによって実装されるユーザー定義関数（UDF）は、プライグインへの相対パスと UDF 名で識別されます。サポートされる UDF の種類のリストについては、「ネイティブプラグインとは」（397 ページ）を参照してください。

ネイティブプラグインライブラリをインストールすると、MarkLogic サーバーでは Extensions データベースに格納されます。プラグインのインストール先となる MarkLogic サーバーインスタンスがクラスタの一部である場合、プラグインライブラリは、クラスタ内のすべてのノードに自動的にプロパゲートされます。

プラグインをインストールしてから、新しいバージョンが利用可能になるまでに、わずかな遅延が生じることがあります。MarkLogic サーバーがプラグインの変更を確認するのは、1 秒間に 1 回だけです。変更が検出されると、古いバージョンのインストールされているホストにプラグインがコピーされます。

また、各ホストには、ネイティブライブラリの読み込み元であるローカルキャッシュがありますが、プラグインの使用中はキャッシュを更新できません。プラグインキャッシュがリフレッシュを開始すると、プラグインの使用を試みる操作は、キャッシュの更新が完了するまで再試行されます。

MarkLogic サーバーはオンデマンドでプラグインを読み込みます。ネイティブプラグインライブラリは、プラグインで実装された UDF をアプリケーションが初めて呼び出すまでは、動的に読み込まれません。プラグインの読み込みと読み込み解除は、ホスト上のプラグインがすべて使用中でないときに限られます。

22.3 ネイティブプラグインライブラリの構築

ネイティブプラグインは、MarkLogic サーバーコアと同じプロセスコンテキストで実行されます。そのため、MarkLogic サーバー実行可能ファイルと互換性のある方法でライブラリをコンパイルし、リンクする必要があります。ライブラリをビルドするには、次の基本的な手順に従います。

- C++ コンパイラ、および MarkLogic サーバーと互換性のある標準ライブラリを使用して、ライブラリをコンパイルします。後述の表を参照してください。これが必要なのは、C++ ではバイナリ互換性がコンパイラバージョン間で保証されていないためです。
- 共有オブジェクトを作成するためにプラットフォームで必要なオプションを指定して、C++ コードをコンパイルします。例えば、Linux では、`-fPIC` オプションを指定してコンパイルします。
- 64 ビットライブラリ（Windows では 32 ビット）をビルドします。

marklogic_dir/Samples/NativePlugins にあるサンプルプラグインには、サポートされるすべてのプラットフォームにおいて GNU make で使用可能な makefile が用意されています。この makefile には、必須のコンパイラオプションがすべて含まれているため、独自のプラグインをビルドするための基礎として使用してください。

この makefile は共有ライブラリをビルドし、マニフェストを生成し、ライブラリとマニフェストをインストールパッケージに zip 圧縮します。makefile は、ファイルの先頭にある make 変数をいくつか変更するだけで、独自のプラグイン用に簡単にカスタマイズできます。

```
PLUGIN_NAME = sampleplugin
PLUGIN_VERSION = 0.1
PLUGIN_PROVIDER = MarkLogic
PLUGIN_DESCRIPTION = Example native plugin

PLUGIN_SRCS = \
  SamplePlugin.cpp
```

以下の表は、MarkLogic サーバーをビルドするために使用されるコンパイラと標準ライブラリのバージョンを示しています。ネイティブプラグインは、互換性のあるツールを使用してビルドする必要があります。

プラットフォーム	コンパイラ
Linux	gcc 4.8.3
Windows	Microsoft Visual Studio 9 SP1
MacOS	gcc 4.2.1

22.4 ネイティブプラグインのパッケージング

ネイティブプラグインをインストールするには、zip ファイルにパッケージ化する必要があります。インストール用 zip ファイルには、次の内容を含める必要があります。

- プラグインインターフェイスを実装する C++ 共有ライブラリ（marklogic::AggregateUDF など）、および登録関数 marklogicPlugin。
- プラグインのマニフェストファイル manifest.xml。「プラグインマニフェスト」（406 ページ）を参照してください。
- 必要に応じて、プラグインの実装に必要なその他の共有ライブラリ。

プラグイン zip ファイルに依存ライブラリを含めることで、プラグインで使用するライブラリバージョンを明示的に制御でき、プラグインがインストールされているクラスタ内のすべてのノードが依存ライブラリを利用できるようになります。

次の例では、プラグインの実装 libsampleplugin.so、依存ライブラリ libdep.so、およびプラグインのマニフェストからプラグインパッケージ sampleplugin.zip を作成します。

```
$ zip sampleplugin.zip libsampleplugin.so libdep.so
manifest.xml
```

プラグインのコンテンツがサブディレクトリに編成されている場合は、マニフェスト内のパスにサブディレクトリを含めます。例えば、プラグインコンポーネントが zip ファイル内で次のように編成されているとします。

```
$ unzip -l sampleplugin.zip
Archive:  sampleplugin.zip
  Length     Date   Time    Name
 --------    ----   ----    ----
    28261  06-28-12 12:54   libsampleplugin.so
      334  06-28-12 12:54   manifest.xml
        0  06-28-12 12:54   deps/
    28261  06-28-12 12:54   deps/libdep.so
 --------                   -------
    56856                   4 files
```

この場合、このプラグインの manifest.xml は、その依存ライブラリパスに deps/ を含む必要があります。

```
<?xml version="1.0" encoding="UTF-8"?>
<plugin xmlns="http://marklogic.com/extension/plugin">
  <name>sampleplugin-name</name>
  <id>sampleplugin-id</id>
  ...
  <native>
    <path>libsampleplugin.so</path>
    <dependency>deps/libdep1.so</dependency>
  </native>
</plugin>
```

22.5 ネイティブプラグインのインストール

「ネイティブプラグインのパッケージング」（399 ページ）の説明に従ってネイティブプラグインをパッケージングした後、XQuery 関数 plugin:install-from-zip またはサーバーサイド JavaScript 関数 plugin:installFromZip を使用してプラグインをインストールまたは更新します。

例えば、次のコードは、ファイル /space/plugins/sampleplugin.zip に含まれているネイティブプラグインをインストールします。Extensions ディレクトリ内の相対プラグインパスは「native」です。

言語	例

XQuery	`xquery version "1.0-ml";`
	`import module namespace plugin =`
	`"http://marklogic.com/extension/plugin"`
	`  at "MarkLogic/plugin/plugin.xqy";`
	
	`plugin:install-from-zip("native",`
	
	`xdmp:document-get("/space/plugins/sampleplugin.zip")/`
	`node())`

サーバーサイド JavaScript	`'use strict';`
	`declareUpdate();`
	`const plugin = require('/MarkLogic/plugin/plugin');`
	
	`plugin.installFromZip(`
	`  'native',`
	`  fn.head(`
	`   xdmp.documentGet('/space/plugins/sampleplugin.zip'))`
	`  .root);`

プラグインがすでに MarkLogic サーバーにインストールされている場合は、古いバージョンが新しいバージョンに置換されます。

インストールされたプラグインは、その「パス」で識別されます。パスの形式は *scope*/*plugin-id* です。ここで、*scope* は `plugin:install-from-zip` の 1 番目のパラメータ、*plugin-id* はプラグインマニフェストの `<id/>` 要素内の ID です。例えば、前述のプラグインのマニフェストに `<id>sampleplugin-id</id>` が含まれる場合、パスは `native/sampleplugin-id` です。

プラグインの zip ファイルは、MarkLogic サーバーが読み取ることのできるファイルである限り、インストール時にファイルシステムの任意の場所に配置できます。インストールプロセスでは、プラグインを Extensions データベースに導入し、ローカルのディスク上キャッシュを MarkLogic サーバーディレクトリ内に作成します。

MarkLogic サーバークラスタの任意のホストでネイティブプラグインをインストールまたは更新すると、クラスタ全体でプラグインが更新されます。ただし、新しいプラグインまたは更新されたプラグインはすぐには使用できないことがあります。詳細については、「MarkLogic サーバーによるネイティブプラグインの管理方法」（398 ページ）を参照してください。

22.6 ネイティブプラグインのアンインストール

ネイティブプラグインをアンインストールするには、XQuery 関数 `plugin:uninstall` またはサーバーサイド JavaScript 関数 `plugin.uninstall` を呼び出します。1 番目のパラメータは、プラグインをインストールしたときのスコープを渡します。2 番目のパラメータは、プラグイン ID（マニフェストの `<id/>`）を渡します。例 :

言語	例

XQuery	`xquery version "1.0-ml";`
	`import module namespace plugin =`
	`"http://marklogic.com/extension/plugin"`
	`  at "MarkLogic/plugin/plugin.xqy";`
	
	`plugin:uninstall("native", "sampleplugin-id")`

サーバーサイド JavaScript	`'use strict';`
	`declareUpdate();`
	`const plugin = require('/MarkLogic/plugin/plugin');`
	
	`// Install a plugin package in the Extensions database`
	`   under`
	`// the relative path 'my/native/plugin'.`
	`plugin.uninstall('native', 'sampleplugin-id');`

プラグインは Extensions データベースから削除され、クラスタ内のすべてのノードでメモリから読み込み解除されます。プラグインがすべてのホストでアンインストールされるまでに、わずかな遅延が生じることがあります。詳細については、「MarkLogic サーバーによるネイティブプラグインの管理方法」（398 ページ）を参照してください。わずかな遅延が生じることがあります。

22.7 実行時のネイティブプラグイン登録

ネイティブプラグインをインストールすると、使用できるようになります。プラグインはオンデマンドで読み込まれます。プラグインが読み込まれると、MarkLogic サーバーは登録ハンドシェイクを使用してプラグインの詳細（バージョン、プラグインが実装している UDF など）をキャッシュします。

この読み込み時登録を実行するために、C++ ネイティブプラグインライブラリごとに `extern "C"` 関数 `marklogicPlugin` を実装する必要があります。この関数のインターフェイスは次のとおりです。

```
using namespace marklogic;
extern "C" void marklogicPlugin(Registry& r) {...}
```

MarkLogic サーバーでは、プラグインライブラリを読み込むときに marklogicPlugin を呼び出すため、プラグインは自身を登録できます。登録の正確な要件は、プラグインで実装するインターフェイスによって異なりますが、少なくとも次の点が含まれている必要があります。

- marklogic::Registry::version を呼び出して、プラグインのバージョンを登録すること。
- 適切な marklogic::Registry 登録メソッドを呼び出して、プラグインが実装するインターフェイスを登録すること。例えば、marklogic::AggregateUDF の実装の場合は Registry::registerAggregate。

プラットフォームでの必要に応じて marklogicPlugin を宣言して、ライブラリ外部でアクセスできるようにします。例えば、Microsoft Windows では、宣言に拡張属性 dllexport を含めます。

```
extern "C" __declspec(dllexport) void
marklogicPlugin(Registry& r)...
```

例えば、次のコードは、2 つの AggregateUDF 実装を登録します。完全な例については、marklogic_dir/Samples/NativePlugins を参照してください。

```
#include "MarkLogic.h"
using namespace marklogic;

class Variance : public AggregateUDF {...};
class MedianTest : public AggregateUDF {...};

extern "C" void marklogicPlugin(Registry& r)
{
    r.version();
    r.registerAggregate<Variance>("variance");
    r.registerAggregate<MedianTest>("median-test");
}
```

22.8 ネイティブプラグインのバージョン管理

登録関数 marklogicPlugin の実装には、プラグインバージョンを登録するために marklogic::Registry::version への呼び出しを含める必要があります。MarkLogic サーバーでは、この情報を使用して、クラスタ全体でのプラグインバージョンの一貫性を維持します。

新しいプラグインバージョンを導入するときは、プラグインの古いバージョンと新しいバージョンがクラスタ内で短時間、共存することがあります。プラグインが使用されるときに MarkLogic サーバーがこの状態を検出した場合、MarkLogic サーバーは XDMP-BADPLUGINVERSION を報告し、プラグインバージョンが同期されるまで、その操作を再試行します。

Registry::version を引数なしで呼び出すと、コンパイル日時（__DATE__ および __TIME__）から構築されたデフォルトバージョンを使用します。これにより、プラグインをコンパイルするたびにバージョン番号が確実に変更されます。次の例では、デフォルトのバージョン番号を使用します。

```
extern "C" void marklogicPlugin(Registry& r)
{
    r.version();
    ...
}
```

明示的なバージョンを Registry::version に渡すことで、この動作をオーバーライドできます。バージョンは数値にする必要があります。例：

```
extern "C" void marklogicPlugin(Registry& r)
{
    r.version(1);
    ...
}
```

MarkLogic サーバーネイティブプラグイン API（marklogic_dir/include/MarkLogic.h）もバージョン管理されます。API の 1 つのバージョンに対してプラグインライブラリをコンパイルし、別のバージョンを実行している MarkLogic サーバーインスタンスに導入することはできません。この不一致を MarkLogic サーバーが検出すると、XDMP-BADAPIVERSION エラーが発生します。

22.9 読込済プラグインのステータスの確認

管理画面または xdmp:host-status 関数を使用して、MarkLogic サーバーに読込済のネイティブプラグインライブラリ、およびそのバージョンと UDF 機能を監視できます。

> **注**：ネイティブプラグインライブラリは、アプリケーションがプラグインで実装された UDF のいずれかを使用するときにオンデマンドで読み込まれます。インストールされていても読み込まれていないプラグインは、ホストのステータスに表示されません。

管理画面を使用して、読み込まれているプラグインを監視するには、次の手順に従います。

1. ブラウザで、管理画面（http://*yourhost*:8001）に移動します。
2. ツリーメニューまたは［Summary］ページで、監視するホストの名前をクリックします。［host summary］ページが表示されます。
3. 右上の［Status］タブをクリックします。［host status］ページが表示されます。
4. ［native plugin status］セクションまで下にスクロールします。

読み込まれているものをプログラムで確認するには、Query Console を開き、次のようなクエリを実行します。

言語	例

XQuery	`xquery version "1.0-ml";`
	`(: List native plugins loaded on this host :)`
	`xdmp:host-status(xdmp:host())//*:native-plugins`

サーバーサイド JavaScript	`'use strict';` `fn.head(xdmp.hostStatus(xdmp.host()))['native-plugins']`

読み込まれているプラグインがある場合は、次のような出力が表示されます。XQuery コードは、XML を送出します。JavaScript コードは、JavaScript オブジェクト（Query Console で JSON として整形出力された）を送出します。この出力は、*MARKLOGIC_DIR*/Samples/NativePlugin にあるサンプルプラグインをインストールして読み込んだ結果です。このサンプルプラグインは、複数の aggregate UDF（「max」、「min」など）と、lexer UDF および stemmer UDF を 1 つずつ実装しています。

言語	例

XQuery	`<native-plugins xmlns=`
	`"http://marklogic.com/xdmp/status/host">`
	`<native-plugin>`
	`<path>native/sampleplugin/libsampleplugin.so</path>`
	`<version>356528850</version>`
	`<capabilities>`
	`<aggregate>max</aggregate>`
	`<aggregate>min_point</aggregate>`
	`<aggregate>min</aggregate>`
	`<aggregate>variance</aggregate>`
	`<aggregate>median-test</aggregate>`
	`<aggregate>max_dateTime</aggregate>`
	`<aggregate>max_string</aggregate>`
	`<lexer>sample_lexer</lexer>`
	`<stemmer>sample_stemmer</stemmer>`
	`</capabilities>`
	`</native-plugin>`
	`</native-plugins>`

言語	例
サーバーサイド JavaScript	[{ "path":"native/sampleplugin/libsampleplugin.so", "version":"356528850", "capabilities":["max", "min_point", "min", "variance", "median-test", "max_dateTime", "max_string", "sample_lexer", "sample_stemmer"] }]

22.10 プラグインマニフェスト

ネイティブプラグイン zip ファイルには、マニフェストファイル `manifest.xml` を含める必要があります。マニフェストファイルには、プラグイン名とプラグイン ID、および zip ファイル内のネイティブプラグイン実装ライブラリごとに 1 つの `<native>` 要素を含める必要があります。オプションのメタデータ（プロバイダやプラグインの説明など）を含めることもできます。詳細については、*MARKLOGIC_INSTALL_DIR*`/Config/plugin.xsd` 内のスキーマを参照してください。

プラグインライブラリへのパスと依存ライブラリへのパスは相対パスでなければなりません。

複数のプラットフォームで同一のマニフェストを使用するには、ネイティブプラグインライブラリをファイル拡張子（Unix の場合は `lib` プレフィックス）なしで指定します。この場合、MarkLogic サーバーでは、次に示すようにプラットフォーム固有の形式でライブラリ名を構成します。

- Windows：`.dll` 拡張子を追加します。
- Linux：`lib` プレフィックスと `.so` 拡張子を追加します。
- Mac OS X：`lib` プレフィックスと `.dylib` 拡張子を追加します。

次の例は、共有ライブラリ `libsampleplugin.so` によって実装された、ID が「sampleplugin-id」のネイティブプラグインのマニフェストです。

```
<?xml version="1.0" encoding="UTF-8"?>
<plugin xmlns="http://marklogic.com/extension/plugin">
  <name>sampleplugin-name</name>
  <id>sampleplugin-id</id>
  <version>1.0</version>
  <provider-name>MarkLogic</provider-name>
  <description>Example native plugin</description>
  <native>
    <path>libsampleplugin.so</path>
```

```
  </native>
</plugin>
```

プラグインパッケージに依存ライブラリが含まれる場合は、そのリストを <native> 要素内に含めます。例：

```
<?xml version="1.0" encoding="UTF-8"?>
<plugin xmlns="http://marklogic.com/extension/plugin">
  <name>sampleplugin-name</name>
  ...
  <native>
    <path>libsampleplugin.so</path>
    <dependency>libdep1.so</dependency>
    <dependency>libdep2.so</dependency>
  </native>
</plugin>
```

22.11 ネイティブプラグインのセキュリティ上の考慮事項

ネイティブプラグインの管理（インストール、更新、またはアンインストール）には、以下が必要です。

- http://marklogic.com/xdmp/privileges/plugin-register 権限、または
- application-plugin-registrar ロール

ネイティブプラグインの読み込みと実行は、次の 2 通りの方法で制御できます。

- native-plugin 権限（http://marklogic.com/xdmp/privileges/native-plugin）が付与されると、すべてのネイティブプラグインを使用できるようになります。
- プラグイン固有の権限を形式 http://marklogic.com/xdmp/privileges/native-plugin/*plugin-path* で定義して、ユーザーが特定の権限を使用できるようにします。

plugin-path は、プラグインを呼び出すときに使用するものと同じプラグインライブラリパスです。例えば、次のプラグインをインストールし、そのマニフェストでプラグインパスを「sampleplugin」と指定した場合、プラグイン固有の権限は http://marklogic.com/xdmp/privileges/native-plugin/native/sampleplugin になります。

```
plugin:install-from-zip("native",
  xdmp:document-get("/space/udf/sampleplugin.zip")/node())
```

プラグイン固有の権限は、事前定義されていません。管理者が作成する必要があります。ただし、存在する場合は、それが MarkLogic サーバーで使用されます。

22.12 ネイティブプラグインの例

MARKLOGIC_DIR/Samples/NativePlugins にあるソースコードと makefile を利用して、サンプルのネイティブプラグインを調べることができます。この例は、数種類の UDF を実装しています。

サンプルの makefile では、ネイティブプラグインのコンパイル、リンク、およびパッケージングの例を示しています。README.txt には、プラグインライブラリのインストールおよび実行方法が記載されています。

23.0 集計ユーザー定義関数

この章では、ユーザー定義の集計関数を作成する方法について説明します。この章は、次のセクションで構成されています。

- 集計ユーザー定義関数とは
- インデータベース MapReduce の概念
- 集計ユーザー定義関数の実装

23.1 集計ユーザー定義関数とは

集計関数とは、MarkLogic サーバーの MapReduce 機能を利用して、レキシコンやレンジインデックスの値を分析します。例えば、要素、属性、またはフィールドのレンジインデックスについて合計や度数を算出します。集計関数は、処理対象のレンジインデックス値やドキュメントの個数に比例して結果の量が大きくなるような分析ではなく、少量の結果を生成する分析に最適です。

MarkLogic サーバーには、独自の集計関数を定義するための C++ インターフェイスが用意されています。動的にリンクされたライブラリに集計ユーザー定義関数（UDF）を組み込み、「ネイティブプラグイン」としてパッケージ化し、そのプラグインを MarkLogic サーバーにインストールします。ネイティブプラグインの詳細については、「ネイティブプラグインの使用」（397 ページ）を参照してください。

ネイティブプラグインは、MarkLogic クラスタ全体に動的に配布されます。アプリケーションが集計 UDF を呼び出すと、ライブラリは、分析に参加しているクラスタ内の各ホストで MarkLogic サーバーに動的に読み込まれます。集計関数がクラスタ全体でどのように動作するかを理解するには、「インデータベース MapReduce の動作」（410 ページ）を参照してください。

この章では、集計 UDF を実装する方法について説明します。集計 UDF を使用する方法については、『*Search Developer's Guide*』の Using Aggregate User-Defined Functions を参照してください。

23.2 インデータベース MapReduce の概念

MarkLogic サーバーでは、インデータベース MapReduce を使用して、MarkLogic クラスタのホスト間で分析処理を効率よく並列化し、その処理をデータの近くへ移動します。

このセクションには、次の内容が含まれます。

- MapReduce とは
- インデータベース MapReduce の動作

インデータベース MapReduce の効率性は、ビルトインおよびユーザー定義の集計関数を使用して明示的に高めることができます。詳細については、『*Search Developer's Guide*』の Using Aggregate Functions を参照してください。

23.2.1 MapReduce とは

MapReduce は、分散型の並列プログラミングモデルです。このモデルでは、各データサブセットを並列 map タスクおよび reduce タスクへ渡すことで、大規模なデータセットが、独立処理されるサブセットに分割されます。通常、map タスクおよび reduce タスクは複数のホストに分散されます。

map タスクは、入力データを map 関数に渡して、中間結果を計算します。そして、reduce タスクで中間結果を処理して、最終結果を生成します。

MarkLogic サーバーは、2 種類の MapReduce をサポートします。

- インデータベース MapReduce は、修飾関数（ビルトインまたはユーザー定義の集計関数など）を使用するときに、MarkLogic クラスタ全体に処理を分散します。詳細については、「インデータベース MapReduce の動作」（410 ページ）を参照してください。
- 外部 MapReduce は、Apache Hadoop クラスタ全体に処理を分散し、MarkLogic サーバーをデータソースまたは結果リポジトリとして使用します。詳細については、『*MarkLogic Connector for Hadoop Developer's Guide*』を参照してください。

23.2.2 インデータベース MapReduce の動作

インデータベース MapReduce は、MarkLogic サーバーデータベースの内部構造を利用して、データの近くで分析を実行します。集計ユーザー定義関数を呼び出すと、MarkLogic サーバーはインデータベース MapReduce を使用してその関数を実行します。

MarkLogic サーバーは、フォレストおよびスタンドと呼ばれる構造内にデータを格納します。大規模なデータベースは、通常、複数のフォレストに格納されます。フォレストは、MarkLogic サーバークラスタ内の複数のホスト上に配置できます。フォレスト内のデータは、複数のスタンドに格納できます。MarkLogic サーバーによるコンテンツの編成方法の詳細については、『Administrator's Guide』の Understanding Forests、および『*Scalability, Availability, and Failover Guide*』の Clustering in MarkLogic Server を参照してください。

インデータベース MapReduce 分析は次のように動作します。

1. アプリケーションは、インデータベース MapReduce 関数（`cts:sum-aggregate`、`cts:aggregate` など）を呼び出します。関数が評価される E ノードが、MapReduce「ジョブ」を開始します。

2. options ノードは、ジョブで必要な処理を、対象データベースのローカルフォレストとリモートフォレストに分散します。処理の各単位は、ジョブの「タスク」です。

3. 参加している各ホストは、並列して map タスクを実行し、そのホストでデータを処理します。ジョブに必要なデータを含むフォレストあたり 1 つ以上の map タスクがあります。

4. 参加している各ホストは、reduce タスクを実行して、スタンドごとのローカルの map 結果を集計し、この中間結果を options ノードに返します。

5. options ノードは reduce タスクを実行して、各ホストからの結果を集計します。

6. options ノードは、「完了」操作を実行して、最終結果を生成します。

23.3 集計ユーザー定義関数の実装

集計ユーザー定義関数（UDF）を作成するには、`marklogic::AggregateUDF` C++ 抽象クラスのサブクラスを実装し、ネイティブプラグインとして導入します。

このセクションでは、次の内容を取り上げます。

- 集計 UDF の作成および導入
- AggregateUDF::map の実装
- AggregateUDF::reduce の実装
- AggregateUDF::finish の実装
- 集計 UDF の登録
- 集計 UDF のメモリ管理
- AggregateUDF::encode および AggregateUDF::decode の実装
- 集計 UDF のエラー処理とロギング
- 集計 UDF の引数処理
- 集計 UDF の型変換

23.3.1 集計 UDF の作成および導入

集計ユーザー定義関数（UDF）は、MarkLogic レンジインデックス値またはインデックス値共起の計算を実行する C++ クラスです。`marklogic::AggregateUDF` のサブクラスを実装するには、XQuery、Java、または REST アプリケーションで使用可能な独自のインデータベース map 関数および reduce 関数を記述します。MarkLogic サーバーのインデータベース MapReduce フレームワークは、C++ コードの配布および並列化を処理します（「インデータベース MapReduce の動作」（410 ページ）を参照）。

注： 集計 UDF は、MarkLogic サーバーと同じメモリおよびプロセス空間内で実行されます。そのため、プラグインでエラーが発生すると、MarkLogic サー

バーがクラッシュすることがあります。集計 UDF を導入する前に、「ネイティブプラグインの使用」（397 ページ）を確認して理解してください。

集計 UDF を作成および導入するには、次の手順に従います。

1. C++ クラス `marklogic::AggregateUDF` のサブクラスを実装します。インターフェイスの詳細については、*marklogic_dir*`/include/MarkLogic.h` を参照してください。

2. プラグイン登録を実行する `extern "C"` 関数 `marklogicPlugin` を実装します。「実行時のネイティブプラグイン登録」（402 ページ）を参照してください。

3. 実装をネイティブプラグインにパッケージ化します。「ネイティブプラグインのパッケージング」（399 ページ）を参照してください。

4. XQuery 関数 `plugin:install-from-zip` を呼び出してプラグインをインストールします。「ネイティブプラグインのインストール」（400 ページ）を参照してください。

完全な例は、*marklogic_dir*`/Samples/NativePlugins` にあります。プラグインをビルドするための基礎としてサンプル Makefile を使用してください。詳細については、「ネイティブプラグインライブラリの構築」（398 ページ）を参照してください。

次の表は、実装する必要がある `marklogic::AggregateUDF` の主要なメソッドをまとめたものです。

メソッド名	説明
`start`	ジョブおよびプロセスの引数の状態を初期化します。ジョブごとに 1 回、options ノードで呼び出されます。
`map`	map 計算を実行します。map タスクごとに 1 回（対象コンテンツを含むデータベースのスタンドごとに 1 回以上）呼び出されます。ローカルオブジェクトおよびリモートオブジェクトで呼び出すこともできます。例えば、平均集計では、スタンドごとの合計と件数を計算します。
`reduce`	reduce 計算を実行し、map 結果を集計します。N-1 回（N = map タスクの数）、呼び出されます。例えば、平均集計では、入力データセット全体の合計および件数を計算します。
`finish`	呼び出し元アプリケーションに返される最終結果を生成します。ジョブごとに 1 回、options ノードで呼び出されます。例えば、平均集計では、合計および件数から平均を計算します。

メソッド名	説明
clone	集計 UDF オブジェクトのコピーを作成します。オブジェクトを作成するために map タスクごとに 1 回以上呼び出され、map および reduce メソッドを実行します。
close	クローンされたオブジェクトが不要になったことを、実装に通知します。
encode	集計 UDF オブジェクトをシリアライズして、クラスタ内のリモートホストに転送できるようにします。
decode	集計 UDF オブジェクトがリモートホストとの間で転送された後、デシリアライズします。

23.3.2 AggregateUDF::map の実装

AggregateUDF::map には、次のシグネチャがあります。

```
virtual void map(TupleIterator&, Reporter&);
```

入力レンジインデックス値にアクセスするには、marklogic::TupleIterator を使用します。map が呼び出されるオブジェクトのメンバーとして map 結果を格納します。エラーのレポートおよびロギングには、marklogic::Reporter を使用します。「集計 UDF のエラー処理とロギング」（421 ページ）を参照してください。

このセクションには、次の内容が含まれます。

- TupleIterator を使用したインデックス値に対する反復処理
- Map 入力タプルの順序の制御

23.3.2.1 TupleIterator を使用したインデックス値に対する反復処理

AggregateUDF::map に渡される marklogic::TupleIterator は、1 つの map タスクに割り当てられた入力レンジインデックス値のシーケンスです。TupleIterator を使用すると、次の処理を実行できます。

- TupleIterator::next および TupleIterator::done を使用して、タプルに対して反復処理を実行する。T
- TupleIterator::width を使用して、各タプルの値の個数を決定する。
- TupleIterator::value を使用して、各タプルの値にアクセスする。
- TupleIterator::type を使用して、タプル内の値の型をクエリする。

集計 UDF が単一のレンジインデックスで呼び出される場合、各タプルに含まれる値は 1 つだけです。集計 UDF が N 個のインデックスで呼び出される場合、各タプルは、1 つの N 方向共起を表現し、N 個の値（インデックスごとに 1 つ）を含んでいます。詳細については、『*Search Developer's Guide*』の Value Co-Occurrences Lexicons を参照してください。

タプル内の値の順序は、集計 UDF を呼び出すときのレンジインデックスの順序に対応します。1 番目のインデックスで、各タプルの 1 番目の値が決定されます。空（null）のタプル値を使用できます。

タプルの値を、互換性のない型の C++ 変数として抽出しようとすると、MarkLogic サーバーは例外をスローします。詳細については、「集計 UDF の型変換」（424 ページ）を参照してください。

次の例では、`map` メソッドが、`<name>`（文字列）と `<zipcode>`（整数）の 2 方向共起を操作すると想定されています。各タプルは、`(name, zipcode)` という値ペアです。各タプルで、name は 0 番目の項目、zipcode は 1 番目の項目です。

```
#include "MarkLogic.h"
using namespace marklogic;
...
void myAggregateUDF::map(TupleIterator& values, Reporter& r)
{
    if (values.width() != 2) {
        r.error("Unexpected number of range indexes.");
        // does not return
    }
    for (; !values.done(); values.next()) {
        if (!values.null(0) && !values.null(1)) {
            String name;
            int zipcode;

            values.value(0, name);
            values.value(1, zipcode);
            // work with this tuple...
    }
}
```

23.3.2.2 Map 入力タプルの順序の制御

MarkLogic サーバーは、`marklogic::TupleIterator` を使用して入力データを map 関数に渡します。デフォルトでは、イテレータの対象であるタプルは降順です。`AggregateUDF::getOrder` をオーバーライドすることで、順序を制御できます。

次の例では、入力タプルが昇順で提供されます。

```
#include "MarkLogic.h"
using namespace marklogic;
...
RangeIndex::getOrder myAggregateUDF::getOrder() const
{
    return RangeIndex::ASCENDING;
}
```

23.3.3 AggregateUDF::reduce の実装

`AggregateUDF::reduce` は、作成した 2 つの集計 UDF オブジェクトからの中間結果を組み込みます。`reduce` が呼び出されるオブジェクトは、アキュムレータとして機能します。

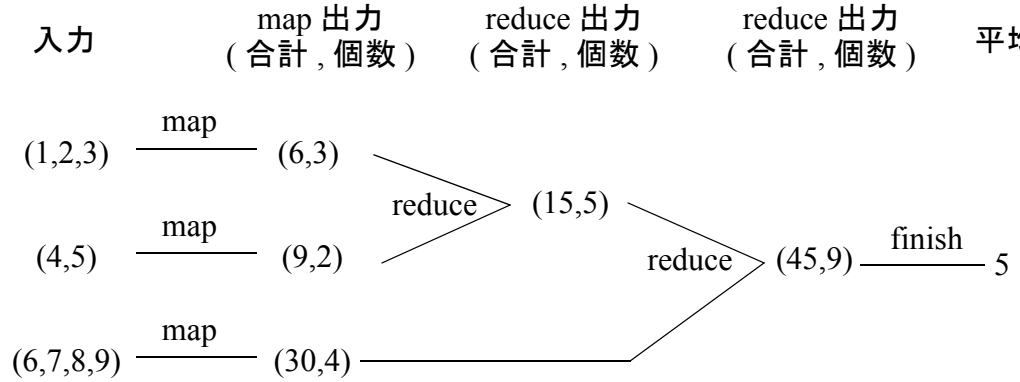
`reduce` 関数には、次のシグネチャがあります。入力 `AggregateUDF` からのデータを、reduce が呼び出されるオブジェクトに組み込みます。エラーのレポートやメッセージのログ記録には `Reporter` を使用します。「集計 UDF のエラー処理とロギング」（421 ページ）を参照してください。

```
virtual void reduce(const AggregateUDF*, Reporter&);
```

すべての map 結果が組み込まれるまで、MarkLogic サーバーは `reduce` を繰り返し呼び出します。その後、`finish` を呼び出して、最終結果を生成します。

例えば、値の集まりの算術平均を計算する集計 UDF を考えてみましょう。計算には、値の合計と値の個数が必要です。map タスクは、データのサブセットに関する中間の合計と個数を蓄積します。すべての reduce タスクが完了すると、E ノードの 1 つのオブジェクトに合計と個数が含まれます。ここで MarkLogic サーバーは、このオブジェクトで `finish` を呼び出して、平均を計算します。

例えば、入力レンジインデックスに値 1 ～ 9 が含まれる場合、その平均は 5（45÷9）です。次の図は、MarkLogic サーバーがインデックス値を、シーケンス (1,2,3)、(4,5)、および (6,7,8,9) として 3 つの map タスクに分散した場合の map-reduce-finish サイクルを示しています。



次のコードスニペットは、レンジインデックスからの値の平均（合計 ÷ 個数）を計算する集計 UDF です。map メソッド（ここには示されていない）は、レンジインデックスの一部分に対して合計と個数を計算し、それらの値を集計 UDF オブジェクトに格納します。reduce メソッドは、集計 UDF オブジェクトのペアからの合計と個数を組み込んで、最終的にインデックスのすべての値に対する合計と個数を取得します。

```
#include "MarkLogic.h"
using namespace marklogic;

class Mean : public AggregateUDF
{
public:
    void reduce(const AggregateUDF* o, Reporter& r)
        sum += o->sum;
        count += o->count;
    }

    // finish computes the mean from sum and count
    ....
protected:
    double sum;
    double count;
};
```

完全な例については、*marklogic_dir*/Samples/NativePlugin を参照してください。

23.3.4 AggregateUDF::finish の実装

AggregateUDF::finish は、最終計算を実行し、呼び出し元アプリケーションに返す出力シーケンスを準備します。シーケンスの各値は、シンプルな値（int、string、DateTime など）またはキー / バリューマップ（XQuery の map:map）です。MarkLogic サーバーは、ジョブごとに 1 回、options ノードで finish を呼び出します。MarkLogic サーバーは、累積した reduce 結果を保持する集計 UDF オブジェクトで finish を呼び出します。

AggregateUDF::finish には、次のシグネチャがあります。最終的な値または map を記録するには、marklogic::OutputSequence を使用します。エラーとログのメッセージを報告するには、marklogic::Reporter を使用します。「集計 UDF のエラー処理とロギング」（421 ページ）を参照してください。

```
virtual void finish(OutputSequence&, Reporter&);
```

出力シーケンスに値を追加するには、OutputSequence::writeValue を使用します。キー / バリューマップである値を追加するには、OutputSequence::startMap and OutputSequence::endMap の間で、OutputSequence::writeMapKey および OutputSequence::writeValue の呼び出しを括弧で囲みます。例 :

```
void MyAggregateUDF::finish(OutputSequence& os, Reporter& r)
{
    // write a single value
    os.writeValue(int(this->sum/this->count));

    // write a map containing 2 key-value pairs
    os.startMap();
    os.writeMapKey("sum");
    os.writeValue(this->sum);
    os.writeMapKey("count");
    os.writeValue(this->count);
    os.endMap();
}
```

MarkLogic サーバーが C++ コードと呼び出し元アプリケーションの間で型を変換する方法については、「集計 UDF の型変換」（424 ページ）を参照してください。

23.3.5 集計 UDF の登録

集計 UDF 実装をアプリケーションで利用できるようにするには、MarkLogic サーバーに登録する必要があります。

実装を登録するには、marklogic::Registry::registerAggregate を marklogicPlugin から呼び出します。marklogicPlugin の詳細については、「実行時のネイティブプラグイン登録」（402 ページ）を参照してください。

Registry::registerAggregate を呼び出すと、UDF クラスのオブジェクトを作成するために使用できる関数へのポインタが MarkLogic サーバーに渡されます。MarkLogic サーバーは、アプリケーションが集計 UDF を呼び出すときは常に、この関数を呼び出します。詳細については、「集計 UDF のメモリ管理」（419 ページ）を参照してください。

MarkLogic サーバーでデフォルトのアロケータおよびコンストラクタを使用するには、marklogic::Registry::registerAggregate のテンプレートバージョンを呼び出します。独自のオブジェクトファクトリを使用するには、仮想バージョンを呼び出します。次のコードスニペットは、2 つの登録インターフェイスを示しています。

```
// From MarkLogic.h
namespace marklogic {
```

```
typedef AggregateUDF* (*AggregateFunction)();
class Registry
{
public:
  // Calls new T() to allocate an object of your UDF class
  template<class T> void registerAggregate(const char* name);

  // Calls your factory func to allocate an object of your
     UDF class
  virtual void registerAggregate(const char* name,
  AggregateFunction);
  ...
};
}
```

Registry::registerAggregate に渡される文字列は、プラグインを呼び出すためにアプリケーションが使用する名前です。例えば、次の XQuery の cts:aggregate の 2 番目のパラメータです。

```
cts:aggregate("pluginPath", "ex1", ...)
```

または、次の REST クライアント API を使用した /values/{name} の aggregate パラメータの値です。

```
GET /v1/values/theLexicon?aggregate=ex1&aggregatePath=
pluginPath
```

次の例は、MyFirstAggregate を名前「ex1」で登録するテンプレート関数、およびオブジェクトファクトリを使用する 2 番目の集計を名前「ex2」で登録する仮想メンバー関数の使用方法を示しています。

```
#include "MarkLogic.h"
using namespace marklogic;
...
AggregateUDF* mySecondAggregateFactory() {...}

extern "C" void marklogicPlugin(Registry& r)
{
    r.version();
    r.registerAggregate<MyFirstAggregate>("ex1");
    r.registerAggregate("ex2", &mySecondAggregateFactory);
}
```

23.3.6 集計 UDF のメモリ管理

このセクションでは、MarkLogic サーバーが集計 UDF クラスのオブジェクトを作成および削除する方法の概要を説明します。

- 集計 UDF オブジェクトの有効期間
- 集計 UDF でのカスタムアロケータの使用

23.3.6.1 集計 UDF オブジェクトの有効期間

集計 UDF クラスのオブジェクトは、次の 2 通りの方法で作成されます。

- プラグインを登録するときに、登録関数は `marklogic::Registry::registerAggregate` を呼び出し、`AggregateUDF` サブクラスのオブジェクトを作成する関数へのポインタを MarkLogic サーバーに渡します。この関数は、アプリケーションがいずれかの集計 UDF を呼び出すときに、`AggregateUDF::start` の呼び出しの前に呼び出されます。
- MarkLogic サーバーは `AggregateUDF::clone` を呼び出して、追加のオブジェクトを作成し、必要に応じて、map タスクや reduce タスクを実行します。

MarkLogic サーバーは `AggregateUDF::clone` を使用して、UDF の呼び出し時に map タスクや reduce タスクでアルゴリズムを実行する一時的なオブジェクトを作成します。MarkLogic サーバーは、集計関数を評価するときに、フォレストごとに 1 つ以上のクローンを作成します。

タスクやジョブの終了時などにクローンが不要になると、MarkLogic サーバーは `AggregateUDF::close` を呼び出してクローンを解放します。

集計 UDF の `clone` および `close` メソッドは、ジョブごとに何回も呼び出されることがあります。

23.3.6.2 集計 UDF でのカスタムアロケータの使用

カスタムアロケータを使用して独自のオブジェクトを管理する場合は、オブジェクトファクトリ関数を実装して `marklogic::Registry::registerAggregate` に渡します（「集計 UDF の登録」（417 ページ）を参照）。

このファクトリ関数は、アプリケーションがプラグインを呼び出すときに常に呼び出されます。つまり、`cts:aggregate`（または等価な関数）の呼び出しごとに 1 回です。map タスクおよび reduce タスクの実行に必要な追加のオブジェクトは、`AggregateUDF::clone` を使用して作成されます。

ファクトリ関数は、次のように `marklogic::AggregateFunction` インターフェイスに準拠している必要があります。

```
// From MarkLogic.h
namespace marklogic {

typedef AggregateUDF* (*AggregateFunction)();
}
```

次の例では、オブジェクトファクトリ関数を Registry::registerAggregate に渡します。

```
#include "MarkLogic.h"
using namespace marklogic;
...
AggregateUDF* myAggregateFactory() { ...}

extern "C" void marklogicPlugin(Registry& r)
{
    r.version();
    r.registerAggregate("ex2", &myAggregateFactory);
}
```

ファクトリ関数および AggregateUDF::clone で作成されるオブジェクトは、MarkLogic サーバーが AggregateUDF::close メソッドを呼び出すまで存続する必要があります。

集計 UDF オブジェクトの割り当ておよび割り当て解除を制御するには、次のエントリーポイントを使用します。

- Registry::registerAggregate に渡す AggregateFunction
- AggregateUDF::clone 実装
- AggregateUDF::close 実装

23.3.7 AggregateUDF::encode および AggregateUDF::decode の実装

MarkLogic サーバーは、Aggregate::encode および Aggregate::decode を使用して、集計分析をクラスタ全体に配布するときに集計オブジェクトをシリアライズおよびデシリアライズします。これらのメソッドのシグネチャは次のとおりです。

```
class AggregateUDF
{
public:
  ...
  virtual void encode(Encoder&, Reporter&) = 0;
  virtual void decode(Decoder&, Reporter&) = 0;
  ...
};
```

次のガイドラインに従った encode および decode の実装を提供する必要があります。

- オブジェクトの実装固有の状態をエンコード / デコードします。
- データメンバーを任意の順序でエンコードできるが、エンコードとデコードが一貫している必要があります。つまり、エンコード時と同じ順序でメンバーをデコードする必要があります。

データメンバーをエンコード / デコードするには、marklogic::Encoder および marklogic::Decoder を使用します。これらのクラスには、基本的な項目型や任意のバイトシーケンスをエンコードおよびデコードするためのヘルパーメソッドが用意されています。詳細については、*marklogic_dir*/include/MarkLogic.h を参照してください。

次の例は、2 つのデータメンバー sum および count を持つ集計 UDF をエンコード / デコードする方法を示しています。データメンバーのエンコードとデコードが、同じ順序で実行されることに注意してください。

```
#include "MarkLogic.h"

using namespace marklogic;

class Mean : public AggregateUDF
{
public:
  ...
  void encode(Encoder& e, Reporter& r)
    {
      e.encode(this->sum);
      e.encode(this->count);
    }
  void decode(Decoder& d, Reporter& r)
    {
      d.decode(this->sum);
      d.decode(this->count);
    }
  ...
protected:
  double sum;
  double count;
};
```

23.3.8 集計 UDF のエラー処理とロギング

メッセージをログに記録して致命的エラーを MarkLogic サーバーに通知するには、marklogic::Reporter を使用します。コードでは、例外のスローによって MarkLogic サーバーにエラーを報告しないでください。

致命的なエラーを報告するには marklogic::Reporter::error を使用します。Reporter::error を呼び出すと、制御がコードに戻りません。報告元のタスクは直ちに停止し、そのホストでは追加の関連タスクが作成されず、ジョブが早期に停止します。MarkLogic サーバーは XDMP-UDFERR をアプリケーションに返します。エラーメッセージが XDMP-UDFERR に含まれています。

> **注：** ジョブは、すぐには停止しません。エラーを報告するタスクは停止しますが、その他の進行中の map タスクと reduce タスクは、完了するまで実行される場合があります。

致命的でないエラーおよびその他のメッセージを報告するには、marklogic::Reporter::log を使用します。このメソッドは、MarkLogic サーバーのエラーログ ErrorLog.txt にメッセージを記録し、制御をコードに戻します。AggregateUDF のほとんどのメソッドには、marklogic::Reporter 入力パラメータがあります。

次の例は、呼び出し元が必須パラメータを提供しない場合は分析をアボートし、呼び出し元が追加パラメータを提供する場合は警告をログに記録します。

```
#include "MarkLogic.h"
using namespace marklogic;
...
void ExampleUDF::start(Sequence& arg, Reporter& r)
{
  if (arg.done()) {
    r.error("Required parameter not found.");
  }
  arg.value(target_);
  arg.next();
  if (!arg.done()) {
    r.log(Reporter::Warning, "Ignoring extra parameters.");
  }
}
```

23.3.9 集計 UDF の引数処理

このセクションには、次の内容が含まれます。

- 引数を集計 UDF に渡す
- AggregateUDF::start での引数処理
- 例：引数を集計 UDF に渡す

23.3.9.1 引数を集計 UDF に渡す

引数は、XQuery からのみ、集計 UDF に渡すことができます。Java および REST クライアント API は、引数の受け渡しをサポートしません。

XQuery から、cts:aggregate の 4 番目のパラメータで引数シーケンスを渡します。次の例では、2 つの引数を「count」集計 UDF に渡します。

```
cts:aggregate(
  "native/samplePlugin",
  "count",
  cts:element-reference(xs:QName("name"),
  (arg1,arg2))
```

引数は、AggregateUDF::start に渡された marklogic::Sequence としてプラグインに届きます。詳細については、「AggregateUDF::start での引数処理」（423 ページ）を参照してください。

完全な例については、「例：引数を集計 UDF に渡す」（423 ページ）を参照してください。

23.3.9.2 AggregateUDF::start での引数処理

MarkLogic サーバーでは、AggregateUDF::start に渡された marklogic::Sequence を使用して、集計固有の引数を利用できるようにします。

```
class AggregateUDF
{
public:
  ...
  virtual void start(Sequence& arg, Reporter&) = 0;
  ...
};
```

Sequence クラスに用意されているメソッドでは、引数値に対して反復処理し（next および done）、現在の引数の型をチェックし（type）、現在の引数値を、複数あるネイティブ型の 1 つとして抽出します（value）。

型変換は、値の抽出時に適用されます。詳細については、「集計 UDF の型変換」（424 ページ）を参照してください。

map メソッドおよび reduce メソッドに引数データをプロパゲートする必要がある場合は、start が呼び出されたオブジェクトのデータメンバーに引数データをコピーします。このデータメンバーを encode メソッドおよび decode メソッドに含めることで、リモートの map タスクおよび reduce タスクでデータを利用できるようにします。

23.3.9.3 例：引数を集計 UDF に渡す

インデックス値の 1 つが呼び出し元の提供する値に一致する 2 方向共起の件数を数える集計 UDF を考えてみましょう。次の例では、呼び出し元が値 95008 を cts:aggregate に渡します。

```
xquery version "1.0-ml";
cts:aggregate("native/sampleplugin", "count",
    (cts:element-reference(xs:QName("zipcode"))
    ,cts:element-reference(xs:QName("name"))
    ),
   95008
 )
```

次に示す start メソッドは、引数値を入力 Sequence から抽出し、データメンバー ExampleUDF::target に格納します。値は、MarkLogic サーバーで start を呼び出すオブジェクトのクローンを作成するときに、ジョブのすべてのタスクに自動的にプロパゲートされます。

```
using namespace marklogic;
...
void ExampleUDF::
start(Sequence& arg, Reporter& r)
{
  if (arg.done()) {
    r.error("Required argument not found.");
  } else {
    arg.value(this->target);
    arg.next();
    if (!arg.done()) {
      r.log(Reporter::Warning, "Ignoring extra
                        arguments.");
    }
  }
}
```

23.3.10 集計 UDF の型変換

MarkLogic のネイティブプラグイン API は、プリミティブ型またはラッパークラスを使用して、XQuery 値を等価な C++ 型としてモデル化します。集計 UDF と呼び出し元アプリケーションの間で渡される値は、このアプリケーションが XQuery で実装されていない場合でも MarkLogic サーバー XQuery エバリュエータコアを使用して渡されるため、等価な型およびそれらの間でサポートされる型変換を理解する必要があります。

- 型変換が適用される場所
- 型変換の例
- C++ と XQuery で等価な型

23.3.10.1型変換が適用される場所

プラグインは、次の場所でネイティブ XQuery 値を操作します。

- 呼び出し元アプリケーションから `marklogic::Sequence` を使用してプラグインに渡される引数。
- `marklogic::TupleIterator` を使用して `AggregateUDF::map` に渡されるレンジインデックス値。
- `AggregateUDF::finish` によって `marklogic::OutputSequence` を使用してアプリケーションに返される結果。

これらのすべてのインターフェイス（`Sequence`、`TupleIterator`、`OutputSequence`）には、値を C++ 型として挿入または抽出するメソッドが用意されています。詳細については、*marklogic_dir*`/include/Marklogic.h` を参照してください。

値の抽出時に C++ と XQuery の型が完全には一致しない場合は、XQuery の型キャストルールが適用されます。2 つの型の間で変換を利用できない場合、MarkLogic サーバーは、`XDMP-UDFBADCAST` などのエラーを報告し、ジョブをアボートします。XQuery の型キャスト処理については、次を参照してください。

http://www.w3.org/TR/xpath-functions/#Casting

23.3.10.2型変換の例

この例では、集計 UDF が整数値を取り、アプリケーションは、XQuery ルールで数値に変換できる文字列を渡します。値は整数値として直接抽出できます。呼び出し元アプリケーションが「12345」を渡すとします。

```
(: The application passes in the arg "12345" :)
cts:aggregate("native/samplePlugin", "count", "12345")
```

この場合、C++ コードでは引数を整数値として安全に直接抽出できます。

```
// Your plugin can safely extract the arg as int
void YourAggregateUDF::start(Sequence& arg, Reporter& r)
{
    int theNumber = 0;
    arg.value(theNumber);
}
```

代わりに、「dog」のような数値以外の文字列をアプリケーションが渡した場合は、`Sequence::value` で例外が発生し、ジョブが停止します。

23.3.10.3C++ と XQuery で等価な型

次の表は、ネイティブプラグイン API でサポートされる C++ と XQuery の型のうち、等価な型をまとめたものです。次に示す C++ クラス型はすべて、*marklogic_dir*/include/MarkLogic.h で宣言されています。

XQuery の型	C++ の型
xs:int	int
xs:unsignedInt	unsigned
xs:long	int64_t
xs:unsignedLong	uint64_t
xs:float	float
xs:double	double
xs:boolean	bool
xs:decimal	marklogic::Decimal
xs:dateTime	marklogic::DateTime
xs:time	marklogic::Time
xs:date	marklogic::Date
xs:gYearMonth	marklogic::GYearMonth
xs:gYear	marklogic::GYear
xs:gMonth	marklogic::GMonth
xs:gDay	marklogic::GDay
xs:yearMonthDuration	marklogic::YearMonthDuration
xs:dayTimeDuration	marklogic::DayTimeDuration
xs:string	marklogic::String
xs:anyURI	marklogic::String
cts:point	marklogic::Point
map:map	marklogic::Map
item()*	marklogic::Sequence

24.0 ドキュメントコンテンツのリダクション

「リダクション」とは、データベースからの読み取り時にドキュメントの一部を除去または不明瞭化するプロセスのことです。例えば、リダクションを使用して、クレジットカード番号、電話番号、メールアドレスなど、機密性の高い個人情報をドキュメントから除去または不明瞭化できます。この章では、ドキュメントをデータベースから読み取るときに使用できるリダクション機能について説明します。

> **注：** リダクションを使用するときは、Advanced Security License オプションが必要です。

この章には、次の内容が含まれます。

- 用語と定義
- リダクションの概要
- 例：リダクションの基本的な使用方法
- セキュリティ上の考慮事項
- リダクションルールの定義
- リダクションルールのインストール
- リダクションルールの適用
- リダクションルールの検証
- ビルトインリダクション関数のリファレンス
- 例：ビルトインリダクション関数の使用
- ユーザー定義のリダクション関数
- 例：カスタムリダクションルールの使用
- 辞書ベースのマスキングの使用
- 例：辞書ベースのマスキング
- サンプルの実行準備

24.1 用語と定義

この章で使用する用語は次のとおりです。

用語	定義
リダクション	機密情報を不明瞭化または隠ぺい化するための、ドキュメントの変更プロセス。XML および JSON ドキュメントをリダクションできます。
リダクションルール	ドキュメントでリダクションの対象となる部分と、変更を加えるために使用する関数に関する仕様。ルールは、XML または JSON で定義できます。詳細については、「リダクションルールの定義」（442 ページ）を参照してください。
ルールドキュメント	リダクションルールが 1 つだけ含まれるドキュメント。ルールドキュメントは、コンテンツのリダクションに使用する前に、スキーマデータベースにインストールし、コレクションに含めておく必要があります。詳細については、「リダクションルールのインストール」（454 ページ）を参照してください。
ルールコレクション	ルールドキュメントのみが含まれるデータベース collection。ルールは、ドキュメントのリダクションに使用する前に、コレクションに含めておく必要があります。
リダクション関数	リダクション時にコンテンツの変更に使用する関数。リダクションルールには、リダクション関数の仕様を含める必要があります。MarkLogic には、ビルトインリダクション関数が複数用意されています。ユーザー定義のリダクション関数を作成することもできます。詳細については、「ビルトインリダクション関数のリファレンス」（461 ページ）および「ユーザー定義のリダクション関数」（498 ページ）を参照してください。
ソースドキュメント	1 つあるいは複数のリダクションルールを適用するデータベースドキュメント。ドキュメントにリダクションを実行すると、インメモリコピーが作成されます。ソースドキュメントは変更されません。
マスキング	リダクションの形式の 1 つ。元の値を新しい値に置換します。新しい値は、決定論的でもランダムでもかまいません。
決定論的マスキング	リダクションの形式の 1 つ。元の値を新しい値に置換しますが、入力が同じである場合は、出力は常に同じになります。例については、「mask-deterministic」（462 ページ）を参照してください。

用語	定義
ランダムマスキング	リダクションの形式の 1 つ。元の値を、新しいランダムな値に置換します。入力が同じであっても、毎回同じ出力になるとは限りません。例については、「mask-random」（466 ページ）を参照してください。
辞書ベースのマスキング	ランダムまたは決定論的マスキングの形式の 1 つ。新しい値が、ユーザー定義の辞書から取り出されます。詳細については、「辞書ベースのマスキングの使用」（517 ページ）を参照してください。
リダクション辞書	辞書ベースマスキングのソースとして使用できる、特殊な形式の値コレクション。リダクション辞書は、Schemas データベースにインストールする必要があります。XML または JSON を使用して辞書を定義できます。詳細については、「リダクション辞書の定義」（517 ページ）を参照してください。
隠ぺい化	リダクションの形式の 1 つ。元の値は完全に隠ぺいされます。通常は、リダクション操作のセマンティックに応じて、リダクションされた値が含まれる XML 要素または JSON プロパティも表示されません。例については、「conceal」（469 ページ）を参照してください。

24.2 リダクションの概要

このセクションでは、リダクション機能の概要について簡単に説明します。次の内容が含まれます。

- リダクションとは
- ルールによるリダクション要件の表現
- 複数のインターフェイスを使用したルール適用
- リダクションロジックの保護

24.2.1 リダクションとは

この章で説明するリダクション機能は、XML および JSON ドキュメントに適用できる読み取り変換です。通常、リダクションされたドキュメントは、データベースからの読み取り時に選択部分が削除、置換、または不明瞭化されています。例えば、MarkLogic からドキュメントをエクスポートするときに、リダクションを使用してメールアドレスを除去したり、クレジットカード番号の末尾 4 桁以外をすべて不明瞭化できます。

注： リダクションを使用するには、Advanced Security License オプションが必要です。

リダクションは、コンテンツをデータベースからエクスポートするときにデータの非表示を細かく設定する場合に最適です。アプリケーション内できめ細かくリアルタイムで情報を非表示にするには、要素レベルのセキュリティを使用します。詳細については、『*Security Guide*』の Element Level Security を参照してください。ドキュメントレベルのアクセス制御には、ドキュメントパーミッションや URI 権限などのセキュリティ機能を使用します。MarkLogic のセキュリティ機能の詳細については、『*Security Guide*』を参照してください。

警告 リダクションでは、データベース内のドキュメントは保護されません。例えば、読み取り時にドキュメントをリダクションした場合でも、ドキュメントパーミッションや要素レベルのセキュリティなどの機能を使用してコンテンツを適切に保護しない限り、各アプリケーションがコンテンツを検索したり、変更することは可能です。

以下の表に、コンテンツのリダクションに使用できる手法をいくつか示します。リダクションの対象および適用する手法の詳細は、アプリケーションの要件によって異なります。詳細については、「リダクション方式の選択」（444 ページ）を参照してください。

リダクションのタイプ	バリエーション	説明
マスキング	完全	元の値は完全に不明瞭化されます。例えば、`123-45-6789` が `###-##-####` になります。
	部分	元の値の一部が維持されます。例えば、`123-45-6789` が `###-##-6789` になります。
	決定論的	入力が同じである場合は、リダクションされた出力が常に同じになります。例えば、リダクション対象として選択されたコンテンツ内では、出現する値「12345」がすべて「11111」になります。
	ランダム	入力はそれぞれ、リダクションによってランダムな値に変わります。例えば、値「12345」がマスキングによって、ある場所では「1a2f578」に、別の場所では「30da61b」になります。
	辞書ベース	ランダムまたは決定論的マスキングの形式の 1 つ。置換値は、ユーザー定義のリダクション辞書から取り出されます。
隠ぺい化		元の値（および、その値を含む XML 要素または JSON プロパティの場合もある）が完全に削除されます。例えば、値 `/a/b` の隠ぺい化を行うと、`<a><b>12345</b></a>` が `</a>` になる場合があります。

MarkLogic では、mlcp コマンドラインツール、および rdt 名前空間内の XQuery ライブラリモジュールを使用したリダクションをサポートしています。ライブラリモジュールは、サーバーサイド JavaScript とともに使用することもできます。

リダクション機能には、社会保障番号や電話番号の不明瞭化などの一般的なリダクションタスク用のビルトインリダクション関数が含まれます。独自のリダクション関数をプラグインにすることもできます。

24.2.2 ルールによるリダクション要件の表現

MarkLogic では、ルールベースのリダクションを使用します。リダクションルールは、リダクションの対象となるドキュメント内のコンテンツを特定する方法、およびその部分を変更する方法を MarkLogic に指示します。ルールは、リダクション対象のドキュメントとは無関係に、ビジネスロジックを表現します。

リダクションルールの重要な構成要素は、リダクション関数仕様です。この関数は、ルールで選択される入力ノードが何によって変更されるかを指定します。MarkLogic には、ルールで使用できるビルトインリダクション関数が複数用意されています。例えば、社会保障番号、電話番号、およびメールアドレスをリダクションするためのビルトインリダクション関数があります。独自のリダクション関数を定義することもできます。

詳細については、「リダクションルールの定義」（442 ページ）を参照してください。

ルールを適用する前に、Schemas データベースをルールコレクションの一部としてインストールする必要があります。詳細については、「リダクションルールのインストール」（454 ページ）を参照してください。

24.2.3 複数のインターフェイスを使用したルール適用

ドキュメントを MarkLogic から読み取るときにリダクションルールを適用するには、次のツールおよびインターフェイスを使用します。

- mlcp コマンドラインツール
- `rdt:redact` XQuery 関数
- `rdt.redact` サーバーサイド JavaScript 関数

`rdt:redact` および `rdt.redact` 関数は、主にリダクションルールのテスト用に用意されています。

詳細については、「リダクションルールの適用」（456 ページ）を参照してください。

24.2.4 リダクションロジックの保護

ルールとコンテンツを適切に保護するセキュリティポリシーを設計し、実装することが重要です。

リダクションワークフローを使用すると、リダクションの対象であるドキュメントとは無関係に、リダクションルールでキャプチャされるビジネスロジックを保護できます。例えば、リダクションされたドキュメントを生成するユーザーには、ルールを変更する権限も、作成する権限も必要ありません。また、ルールを作成および管理するユーザーには、リダクションの対象であるコンテンツを読み取る権限も、変更する権限も必要ありません。

詳細については、「セキュリティ上の考慮事項」（439 ページ）を参照してください。

24.3 例：リダクションの基本的な使用方法

このセクションでは、リダクションルールを定義、インストール、および適用するシンプルな例について説明します。この例では、ビルトインリダクション関数「redact-email」および「redact-us-phone」を使用します。

この例で使用するルールは、Query Console を使用してインストールし、適用します。mlcp を利用する同様の例については、『*mlcp User Guide*』の Example: Using mlcp for Redaction を参照してください。

この例の説明は、次の手順で構成されます。

1. ソースドキュメントのインストール
2. ルールのインストール
3. ルールについて
4. ルールの適用

24.3.1 ソースドキュメントのインストール

XQuery および Query Console を使用してサンプルドキュメントをドキュメントデータベースにインストールするには、このセクションの手順を使用します。この例では XQuery を使用しますが、演習を正常に完了するために、XQuery の詳しい知識は必要はありません。

この手順の完了後には、ドキュメントデータベースに次のドキュメントが含まれています。簡単に参照できるように、コレクション「gs-samples」にもドキュメントが挿入されます。

- `/redact-gs/sample1.xml`
- `/redact-gs/sample2.json`

サンプルドキュメントを挿入するには、次の手順に従います。

1. ブラウザで Query Console に移動します。例えば、`http://localhost:8000/qconsole` に移動します。

2. 次のスクリプトを、Query Console の新しいクエリタブに貼り付けます。

```
xquery version "1.0-ml";
xdmp:document-insert("/redact-gs/sample1.xml",
  <personal>
    <name>Little Bopeep</name>
    <summary>Seeking lost sheep.Please call 123-456-7890.
    </summary>
    <id>12-3456789</id>
  </personal>,
  <options xmlns="xdmp:document-insert">
    <permissions>{xdmp:default-permissions()}</permissions>
    <collections>
      <collection>gs-samples</collection>
    </collections>
  </options>);

xquery version "1.0-ml";
xdmp:document-insert("/redact-gs/sample2.json",
xdmp:unquote('
  {"personal": {
     "name": "Jack Sprat",
     "summary": "Free nutrition advice! Call (234)567-8901
     now!",
     "id": "45-6789123"
  }}
'),
  <options xmlns="xdmp:document-insert">
    <permissions>{xdmp:default-permissions()}</permissions>
    <collections>
      <collection>gs-samples</collection>
    </collections>
  </options>
);
```

3. ［Database］ドロップダウンで［Documents］を選択します。

4. ［Query Type］ドロップダウンで［XQuery］を選択します。

5. ［Run］ボタンをクリックします。サンプルドキュメントがインストールされます。

6. 必要に応じて、［Database］ドロップダウンの隣にある［Explore］（眼鏡）アイコンをクリックしてデータベースを調べ、サンプルドキュメントの挿入を確認します。

24.3.2 ルールのインストール

ルールは、コンテンツデータベースに関連付けられているスキーマデータベースにインストールする必要があります。ルールは、使用する前にコレクションに含めておく必要があります。このセクションでは、ルールを Schemas データベースにインストールします。Schemas データベースは、ドキュメントデータベースに関連付けられたデフォルトのスキーマデータベースです。

どのドキュメント挿入手法でもルールをインストールできます。この例では、XQuery と Query Console を使用します。この演習を完了するために、XQuery の詳しい知識は必要はありません。他のルールインストールオプションについては、「リダクションルールのインストール」（454 ページ）を参照してください。

この演習の完了後には、XML で定義されたルールと、JSON で定義されたルールが 1 つずつスキーマデータベースに含まれています。ルールは、コレクション「gs-rules」に挿入されます。XML ルールは redact-us-phone ビルトインリダクション関数を使用します。JSON ルールは conceal ビルトインリダクション関数を使用します。

ルールをインストールするには、次の手順に従います。ルールの実行内容については、「ルールについて」（436 ページ）を参照してください。

1. ブラウザで Query Console に移動します。例えば、`http://localhost:8000/qconsole` に移動します。

2. 次のスクリプトを、Query Console の新しいクエリタブに貼り付けます。

```
(: Apply redact-us-phone to //summary :)
xquery version "1.0-ml";
xdmp:document-insert("/rules/gs/redact-phone.xml",
  <rule xml:lang="zxx" xmlns="http://marklogic.com/
  xdmp/redaction">
    <description>Obscure phone numbers.</description>
    <path>//summary</path>
    <method>
      <function>redact-us-phone</function>
    </method>
```

```
    <options>
      <level>partial</level>
    </options>
  </rule>,
  <options xmlns="xdmp:document-insert">
    <permissions>{xdmp:default-permissions()}</permissions>
    <collections>
      <collection>gs-rules</collection>
    </collections>
  </options>
);

(: Apply conceal to //id :)
xquery version "1.0-ml";
xdmp:document-insert("/rules/gs/conceal-id.json",
xdmp:unquote('
  { "rule": {
    "description": "Remove customer ids.",
    "path": "//id",
    "method": { "function": "conceal" }
  }}
'),
  <options xmlns="xdmp:document-insert">
    <permissions>{xdmp:default-permissions()}
    </permissions>
    <collections>
      <collection>gs-rules</collection>
    </collections>
  </options>
);
```

3. ［Database］ドロップダウンで［Schemas］を選択します。

4. ［Query Type］ドロップダウンで［XQuery］を選択します。

5. ［Run］ボタンをクリックします。URI「/rules/gs/redact-phone.xml」および「/rules/gs/conceal-id.json」を使用してルールドキュメントがインストールされ、「custom-rules」コレクションに追加されます。

24.3.3 ルールについて

「ルールのインストール」（434 ページ）でインストールされる XML ルールの形式は次のとおりです。

```
<rule xml:lang="zxx" xmlns="http://marklogic.com/
xdmp/redaction">
  <description>Obscure phone numbers.</description>
  <path>//summary</path>
  <method>
    <function>redact-us-phone</function>
  </method>
  <options>
    <level>partial</level>
  </options>
</rule>
```

このルール要素には次の効果があります。

- `description` - （オプション）情報としてのメタデータ。
- `path` - パス式「//summary」で選択されたノードに、ルールで指定されたリダクション関数を適用します。
- `method` - ビルトインリダクション関数 `redact-us-phone` を使用して、`summary` XML 要素または JSON プロパティの値をリダクションします。デフォルトでは、この関数は、電話番号のすべての数字を文字「#」に置換します。`method` メソッドには子 `module` がないため、これがビルトイン関数であることがわかります。
- `options` - `level` パラメータの値「partial」を redact-us-phone に渡します。これで、値の末尾 4 桁が変更されなくなります。

このルールの適用により想定される結果は、「summary」という名前のノードの値のうち、米国の電話番号のパターンにマッチするすべてのテキストが置換されることです。置換値は、「#」番号を使用して、末尾 4 桁以外のすべてを置換します。例えば、123-456-7890 という値は ###-###-7890 に置換されます。詳細については、「redact-us-phone」（476 ページ）を参照してください。

「ルールのインストール」（434 ページ）でインストールされる JSON ルールの形式は次のとおりです。

```
{ "rule": {
    "description": "Remove customer ids.",
    "path": "//id",
    "method": { "function": "conceal" }
}}
```

このルールプロパティには次の効果があります。

- `description` - （オプション）情報としてのメタデータ。
- `path` - パス式「//id」で選択されたノードに、ルールで指定されたリダクション関数を適用します。
- `method` - ビルトインリダクション関数 `conceal` を使用して、`id` XML 要素または JSON プロパティをリダクションします。この関数は、`path` で選択されたノードを非表示にします。`method` メソッドには子 `module` がないため、これがビルトイン関数であることがわかります。

このルールの適用により想定される結果は、ノード `id` が削除されることです。例えば、`//id` が XML 要素または JSON プロパティを選択すると、その要素またはプロパティは、リダクションされた出力に出現しません。`//id` が JSON で配列項目を選択すると、その項目は除去されますが、ドキュメントの構造によっては `id` プロパティが残る可能性があります。詳細については、「conceal」（469 ページ）を参照してください。

24.3.4 ルールの適用

コレクション「gs-rules」内のルールをサンプルドキュメントに適用するには、このセクションのステップに従います。この例は、Query Console を使用してルールを適用します。mlcp コマンドを使用してルールを適用することもできます。詳細については、「リダクションルールの適用」（456 ページ）を参照してください。

ルールを適用するユーザーには、ソースドキュメント、ルールドキュメント、およびルールコレクションに対する読み取りパーミッションが必要です。詳細については、「セキュリティ上の考慮事項」（439 ページ）を参照してください。

1. ブラウザで Query Console に移動します。例えば、`http://localhost:8000/qconsole` に移動します。

2. XQuery を使用してルールを適用するには、次の手順を実行します。

   a. 次のスクリプトを、Query Console の新しいクエリタブに貼り付けます。

```
xquery version "1.0-ml";
import module namespace rdt = "http://marklogic.com/
xdmp/redaction"
    at "/MarkLogic/redaction.xqy";
rdt:redact(fn:collection("gs-samples"), "gs-rules")
```

   b. ［Query Type］ドロップダウンで［XQuery］を選択します。

3. サーバーサイド JavaScript を使用してルールを適用するには、次の手順を実行します。

   a. 次のスクリプトを、Query Console の新しいクエリタブに貼り付けます。

```
const rdt = require('/MarkLogic/redaction');
rdt.redact(fn.collection('gs-samples'), ['gs-rules']);
```

   b. [Query Type] ドロップダウンで [JavaScript] を選択します。

4. [Databases] ドロップダウンで [Documents] を選択します。

5. [Run] ボタンをクリックします。「gs-rules」コレクション内のルールが「gs-samples」コレクション内のドキュメントに適用されます。

次の表は、XML サンプルドキュメントをリダクションした結果を示しています。summary 要素内の電話番号は、redact-us-phone 関数によって部分的にリダクションされています。また、id 要素は、conceal 関数によって完全に隠ぺい化されています。表では、コンテンツで影響を受ける部分が強調表示されています。

ステージ	XML コンテンツ

Original Document	<personal>
	  <name>Little Bopeep</name>
	  <summary>Seeking lost sheep.Please call
	  123-456-7890.</summary>
	  <id>123456</id>
	</personal>

Redacted Result	<personal> <name>Little Bopeep</name> <summary>Seeking lost sheep.Please call **###-###-7890.**</summary> </personal>

次の表は、JSON サンプルドキュメントをリダクションした結果を示しています。summary プロパティ内の電話番号は、redact-us-phone 関数によって部分的にリダクションされています。また、id プロパティは、conceal 関数によって完全に隠ぺい化されています。表では、コンテンツで影響を受ける部分が強調表示されています。

ステージ	JSON コンテンツ
Original Document	{"personal": { "name": "Jack Sprat", "summary": "Free nutrition advice! Call **(234)567-8901** now!", **"id": 234567** }}
Redacted Result	{"personal": { "name": "Jack Sprat", "summary": "Free nutrition advice! Call **(###)###-8901** now!" }}

24.4 セキュリティ上の考慮事項

リダクションは、読み取り変換の一種であり、ドキュメントをデータベースからエクスポートするときに使用するために用意されています。リダクションでは、データベース内のコンテンツは保護されません。例えば、十分なドキュメントパーミッションを持つユーザーは、リダクション後も、リダクションした情報を含むドキュメントの検索、読み取り、更新が可能です。要素レベルのセキュリティ、ドキュメントパーミッション、URI 権限を使用して、リアルタイムのセキュリティを提供してください。詳細については、『*Security Guide*』を参照してください。

ルールドキュメントおよびルールコレクションには、機密情報が含まれる可能性があります。リダクションルールやルールコレクションに適用されるアクセス制御とセキュリティ要件は、慎重に検討してください。

例えば、次のような、公開を制限するセキュリティ制御を実装する必要があります。

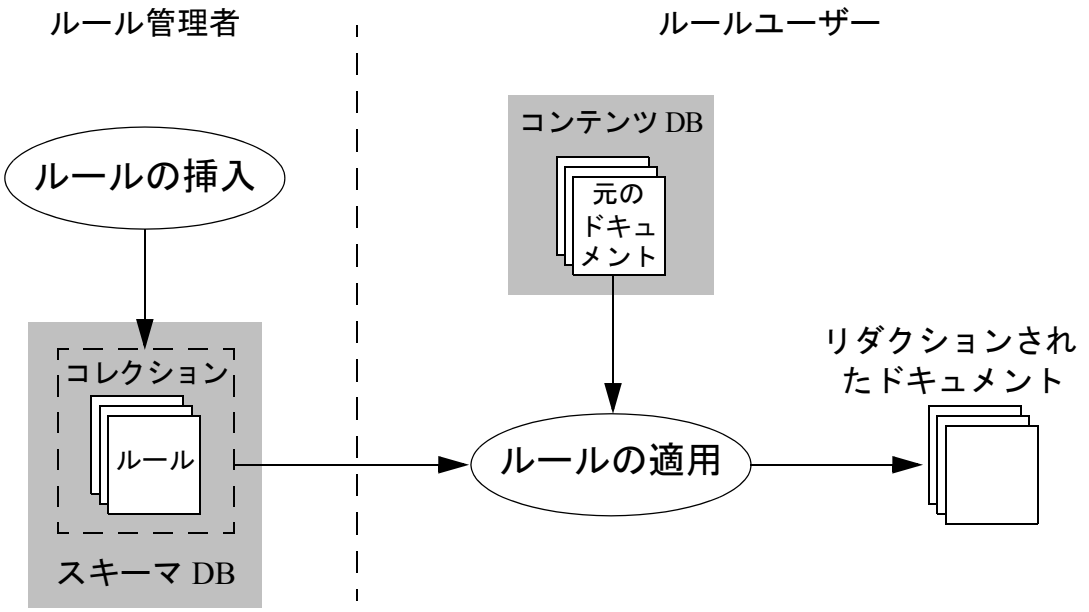
- ルールを読み取ることができる攻撃者は、機密性が高いと考えられるビジネスロジックにアクセスできます。攻撃者がコンテンツに対する読み取りアクセス権を持たない場合でも、ルールロジックに対する読み取りアクセスによって、コンテンツの構造が知られてしまう可能性があります。

- ルールの変更や、ルールコレクションに含まれるルールの変更が可能な攻撃者は、リダクション操作の結果に影響を与えて、リダクションされるはずのデータを公開することができます。

セキュリティアーキテクチャを設計するときは、次の役割を検討してください。

- ルール管理者：ルールの作成、変更、削除や、ルールコレクションの管理、リダクション辞書の作成と変更が可能なユーザーです。このようなユーザーを複数設定し、さまざまなルールコレクションを管理する権限を与えることができます。
- ルールユーザー：ルールを適用できるが、ルールの作成、変更、削除、ルールコレクションの管理は実行できないユーザーです。さまざまなルールユーザーに、それぞれのルールやルールコレクションへのアクセス権を与えることができます。
- 上記以外のユーザー：上記以外のユーザーには、通常、ルールドキュメント、ルールコレクション、リダクション辞書へのアクセスを許可しません。

次の図は、リダクションフローの概要、およびルール管理者とルールユーザーの間の責任の分離を示しています。



リダクションワークフロー

次の表は、リダクションルールの管理および使用に関連した一般的なタスク、そのタスクを通常実行する担当者、および MarkLogic で利用可能な関連セキュリティ機能を示しています。セキュリティ機能の詳細については、表の後で説明します。

タスク	俳優	サポートされるセキュリティ機能
ルールドキュメントを作成または変更する	ルール管理者	ドキュメントパーミッション
ルールコレクションに含めるルールドキュメントを制御する	ルール管理者	protected コレクション
リダクション辞書を作成または変更する	ルール管理者	ドキュメントパーミッション
ルールコレクションを使用してドキュメントをリダクションする	ルールユーザー	ドキュメントパーミッション `redaction-user` セキュリティロール

ドキュメントパーミッションを使用すると、どのユーザーがルールドキュメントとリダクション辞書の読み取り、作成、または更新を実行できるかを制御できます。ルール管理者には、通常、このようなドキュメントに対する読み取りおよび更新パーミッションが必要です。ルールユーザーには、ルールドキュメントやリダクション辞書に対する読み取りパーミッションのみを設定してください。ドキュメントパーミッションの詳細については、『*Security Guide*』の Protecting Documents を参照してください。

protected コレクションにルールドキュメントを配置すると、どのユーザーがコレクションでドキュメントの追加または削除を実行できるかを制御できます。ルール管理者には、通常、保護されているルールコレクションに対する更新パーミッションが必要です。ルールユーザーには、保護されているルールコレクションに対する特別なパーミッションを設定しないでください。protected コレクションは、ドキュメントを追加する前に明示的に作成しておく必要があります。protected コレクションの詳細については、『*Search Developer's Guide*』の Collections and Security を参照してください。

注： protected コレクションを、どのユーザーがコレクション内のドキュメントのコンテンツの読み取りまたは変更を実行できるかを制御するために使用することはできません。これを制御するには、ドキュメントパーミッションを使用する必要があります。また、どのユーザーがコレクション内のドキュメントを表示できるかを制御するために、protected コレクションを使用することもできません。

MarkLogic では、`redaction-user` ロールが事前定義されています。ルールの検証とドキュメントのリダクションには、このロール（または同等の権限）が必要です。つまり、XQuery 関数 `rdt:redact` および `rdt:rule-validate`、JavaScript 関数 `rdt.redact` および `rdt.ruleValidate`、または mlcp の `-redaction` オプションを使用するには、このロールが必要です。

MarkLogic のセキュリティ機能の詳細については、『*Security Guide*』を参照してください。

24.5 リダクションルールの定義

このセクションでは、リダクションルールの作成に関する詳細について説明します。次の内容が含まれます。

- ルール定義の基本
- リダクション方式の選択
- リダクション関数の選択
- XML 名前空間プレフィックスバインドの定義
- リダクションルールにおける XPath 式の制限
- 複数のドキュメント形式で使用可能なルールの定義
- XML ルールシンタックスのリファレンス
- JSON ルールシンタックスのリファレンス

24.5.1 ルール定義の基本

リダクションルールは、XML または JSON で定義できます。ルールの形式（XML または JSON）は、そのルールの適用先となるドキュメントのタイプには影響を及ぼしません。

ルール定義には、少なくとも次の内容を含める必要があります。

- ルールを適用するドキュメントのコンポーネントを定義する XPath 式。一部制限があります。詳細については、「リダクションルールにおける XPath 式の制限」（447 ページ）を参照してください。
- ビルトインまたはユーザー定義のリダクション関数を指定するディスクリプタ。この関数は、パス式で選択されたノードに対してリダクションを実行します。

ルール定義には、説明やオプションなど、追加のデータを含めることができます。詳細については、「XML ルールシンタックスのリファレンス」（450 ページ）または「JSON ルールシンタックスのリファレンス」（452 ページ）を参照してください。

ルールの設計には、次のタスクを含めてください。

- リダクション方式を選択します。例えば、リダクションされる値をマスクするのか、隠ぺい化を行うのかを決定します。詳細については、「リダクション方式の選択」（444 ページ）を参照してください。
- ビルトインとユーザー定義のいずれのリダクション関数を使用するかを決定します。詳細については、「リダクション関数の選択」（445 ページ）を参照してください。

次のルール例では、XPath 式 `//ssn` にマッチするノードに対してビルトインリダクション関数 redact-us-ssn を適用することを指定します。`redact-us-ssn` 関数は、マスクする社会保障番号（SSN）の範囲（全部または一部）を指定する `level` パラメータを使用できます。レベルを指定するには、ルール定義の `options` セクションを使用します。

形式	ルールの例

XML	`<rdt:rule xml:lang="zxx"`
	`  xmlns:rdt="http://marklogic.com/xdmp/`
	`  redaction">`
	`  <rdt:description>Mask SSNs</rdt:description>`
	`  <rdt:path>//ssn</rdt:path>`
	`  <rdt:method>`
	`  <rdt:function>redact-us-ssn</rdt:function>`
	`  </rdt:method>`
	`  <rdt:options>`
	`  <rdt:level>partial</rdt:level>`
	`  </rdt:options>`
	`</rdt:rule>`

JSON	`{"rule": {`
	`  "description": "Mask SSNs",`
	`  "path": "//ssn",`
	`  "method": { "function": "redact-us-ssn" },`
	`  "options": { "level": "partial" }`
	`}}`

これらのルールを「サンプルの実行準備」（528 ページ）のドキュメント例に適用すると、以下のような SSN の XML 要素と JSON プロパティを確認できます。

```
###-##-7890
###-##-9012
###-##-6789
###-##-8901
```

独自の XQuery またはサーバーサイド JavaScript リダクション関数を作成し、その関数を適用するルールを定義することもできます。ユーザー定義関数は、`method` XML 要素または JSON プロパティで、関数名、実装モジュールの URI、およびモジュール名前空間 URI（関数が XQuery で実装されている場合）によって識別されます。詳細については、「ユーザー定義のリダクション関数」（498 ページ）を参照してください。

次の例は、XPath 式 `//name` にマッチするノードに対してユーザー定義リダクション関数「redact-name」を適用することを指定します。詳細と例については、「ユーザー定義のリダクション関数」（498 ページ）を参照してください。

形式	ルールの例

XML	`<rdt:rule xml:lang="zxx"`
	`  xmlns:rdt="http://marklogic.com/xdmp/`
	`  redaction">`
	`  <rdt:description>Mask names</rdt:description>`
	`  <rdt:path>//name</rdt:path>`
	`  <rdt:method>`
	`  <rdt:function>redact</rdt:function>`
	`  <rdt:module>/example/redact-name.xqy`
	`  </rdt:module>`
	`  <rdt:module-namespace>`
	`  http://marklogic.com/example/redaction`
	`  </rdt:module-namespace>`
	`  </rdt:method>`
	`</rdt:rule>`

JSON	`{"rule": {`
	`  "description": "Mask names",`
	`  "path": "//name",`
	`  "method": {`
	`  "function": "redact",`
	`  "module": "/example/redact-name.sjs"`
	`  }`
	`}}`

24.5.2 リダクション方式の選択

通常、リダクションでは、次のいずれかの方法でコンテンツを変更します。

- 部分マスキング：リダクションされる値の一部のみを置換します。例えば、クレジットカード番号の末尾 4 桁以外をすべて文字「#」に置換します。
- 完全マスキング：リダクションされる値全体を新しい値に置換します。例えば、口座番号のすべての文字をランダムな文字列に置換します。

- 隠ぺい化：リダクションされる値またはノードを完全に除去します。

マスキングを使用するときは、次の点も考慮してください。

- 特定の入力に対して置換値を常に同一にするか（決定論的）、またはランダム化するか。

  決定論的マスキングは、値の間の関係性を維持し、検索を容易にすることができます。これは、アプリケーションによってはメリットがある場合と不適切である場合があります。

- 置換値を既知の値リスト（辞書）から取り出すかどうか。

  辞書を使用しない場合、置換値はランダムに生成されるか、文字の集まりの繰り返しになります。これは、ランダムマスキングと決定論的マスキングのどちらを選択するかによって決まります。リダクション辞書を使用すると、事前定義された値の集まりから置換値を取り出すことができます。

- 入力データの「形式」の維持または不明瞭化が重要であるかどうか。

  例えば、「John Smith」をリダクションした結果の値を 2 語と 1 語のどちらにするか。元の入力の語数を維持するか、または「FIRSTNAME LASTNAME」のように正規化するか。

アプリケーションのプライバシー要件が決まったら、適切なビルトインリダクション関数を選択するか、独自に作成することができます。

24.5.3 リダクション関数の選択

リダクション関数は、特定のリダクションルールのロジック（ノードを変更する必要があるかどうかの決定、置換値の生成、値やノードの非表示など）を実装します。ビルトインリダクション関数のいずれかを使用することも、ユーザー定義のリダクション関数を使用することもできます。

次のビルトインリダクション関数が MarkLogic とともにインストールされます。これらの関数は、ほとんどのアプリケーションのニーズに対応しています。これらの関数の詳細については、「ビルトインリダクション関数のリファレンス」（461 ページ）で説明しています。各関数の説明には、例が含まれています。

- `mask-deterministic`
- `mask-random`
- `conceal`
- `redact-number`
- `redact-regex`
- `redact-us-ssn`

- `redact-us-phone`
- `redact-email`
- `redact-ipv4`

ビルトイン関数がアプリケーションのニーズを満たさない場合は、XQuery またはサーバーサイド JavaScript を使用して独自のリダクション関数を作成できます。例えば、ユーザー定義関数で、「顧客が未成年者である場合は名前をリダクションする」という条件付きリダクションを実装することが必要になる場合が考えられます。詳細については、「ユーザー定義のリダクション関数」（498 ページ）を参照してください。

24.5.4 XML 名前空間プレフィックスバインドの定義

`path` XPath 式で名前空間プレフィックスを使用する必要がある場合は、`namespaces` コンポーネントをルールに追加して、名前空間プレフィックスバインドを定義します。例えば、次のルールスニペットは、パス値で「emp」名前空間プレフィックスを使用し、「emp」プレフィックスと名前空間 URI「http://my/employees」の間のバインドを定義します。

ルール形式	ルールスニペット

XML	`<rdt:rule ...>`
	`  <rdt:path>//emp:ssn</rdt:path>`
	`  <rdt:namespaces>`
	`  <rdt:namespace>`
	`  <rdt:prefix>emp</rdt:prefix>`
	`  <rdt:namespace-uri>http://my/employees`
	`  <rdt:namespace-uri>`
	`  <rdt:namespace>`
	`  <rdt:namespace>...</rdt:namespace>`
	`  </rdt:namespaces>`
	`  <rdt:method>...</rdt:method>`
	`</rdt:rule>`

JSON	`{"rule": {`
	`  "path": "//emp:ssn",`
	`  "namespaces": [`
	`  {"namespace": {`
	`  "prefix": "emp"`
	`  "namespace-uri": "http://my/employees"`
	`  }, ...`
	`  ],`
	`  "method": { ...}`
	`}}`

24.5.5 リダクションルールにおける XPath 式の制限

XML ドキュメントに適用されるリダクションルールは、XSLT でサポートされる XPath のサブセットに制限されています。例えば、`parent::*` のような後方軸は使用できません。サポートされるサブセットは、https://www.w3.org/TR/xslt#patterns で定義されています。

JSON ドキュメントに適用されるリダクションルールには、このような制限はありません。ただし、XML ドキュメントと JSON ドキュメントの混合にルールを適用する場合は、ルールを、サポートされる XPath サブセットに制限してください。

ルール検証では、ルールが XML ドキュメントに適用されるかどうかを判断できないため、ルールパスがこの制限に準拠しているかどうかはチェックされません。無効なパスの XML ドキュメントにルールを適用すると、例外 `RDT-INVALIDRULEPATH` が発生します。

24.5.6 複数のドキュメント形式で使用可能なルールの定義

このセクションでは、XML ドキュメントと JSON ドキュメントの両方に適用できるルールを定義するときに重要な考慮事項について説明します。

ルールが XML ドキュメントに適用されるときに、ルールの `path` XML 要素または JSON プロパティにおける XPath 式は、XSLT でサポートされる XPath のサブセットに制限されます。そのため、XML と JSON が混合するコンテキストをリダクションするときは、ルールパスを制限する必要があります。詳細については、「リダクションルールにおける XPath 式の制限」（447 ページ）を参照してください。

リダクションルールでノードを適切に選択するには、XPath とドキュメントモデルのインタラクションを理解する必要があります。XML と JSON のドキュメントモデルには、モデルに慣れていないユーザーにとって、意外な違いが存在します。例えば、「//id」のようなシンプルなパス式は、XML ドキュメントでは、1 つの要素にマッチしますが、JSON では、配列値のすべての項目にマッチします。

ほとんどの場合は、ビルトインリダクション関数が、JSON と XML のドキュメントモデルの相違点を相殺するため、ドキュメントタイプに関係なく、両方の動作の一貫性が保たれます。独自のリダクション関数を記述する場合は、同様の調整が必要になることがあります。

XML ドキュメントと JSON ドキュメントの両方でノードを選択する単一の XPath 式を記述できます。ただし、ドキュメントモデルを十分に理解していないと、想定どおりのノードが選択されない可能性があります。次のヒントに留意してください。

- XML と JSON では、含まれるノード型が異なります。要素ノードと属性ノードが含まれるのは XML ドキュメントのみです。オブジェクト、テキスト、数値、ブール、および null ノードが含まれるのは JSON ノードのみです。つまり、ドキュメントに「color」プロパティが含まれている場合でも、「//@color」のような式が JSON ドキュメント内のノードにマッチすることはありません。

- 「JSON プロパティノード」は存在しません。{"a": 42} のような JSON ドキュメントは、1 つの数値ノード子を持つ名前なしルートオブジェクトノードとしてモデル化されます。数値ノードには「a」という名前が付けられ、その値は 42 です。数値ノードの値は変更できますが、このプロパティは、親オブジェクトノードを操作することでのみ、隠ぺい化することができます。
- JSON 配列の各項目は、同名のノードです。例えば {"a": [1,2]} の場合、パス式「//a」は、これを含んでいる配列ノードではなく、2 つの数値ノードを選択します。配列ノードを選択するには、//array-node('a') のような JSON 固有のパス式が必要です。したがって、配列値のプロパティを隠ぺい化するには、文字列値などのプロパティの隠ぺい化とは異なる方式が必要です。
- 有効な XML 要素ローカル名（スペースが含まれる名前など）ではない名前の JSON プロパティノードは、ノードテスト演算子（node(*name* など）を使用した場合にのみ選択できます。例えば、ドキュメント {"aa bb":「value"} の場合は、パス式 /node('aa bb') を使用して、「aa bb」という名前のプロパティを選択します。
- fn:data() 関数は、XML 要素のテキスト子を集約しますが、JSON プロパティのテキスト子は集約しません。次の表の例を参照してください。

詳細については、「JSON の使用」（347 ページ）を参照してください。

複数のノード型を処理するには、XML と JSON の両方から入力を受け取ることができるリダクション関数を準備する必要があります。例えば、同じ XPath 式によって、XML では要素ノードを選択し、JSON ではオブジェクトノードを選択することができます。

このセクションの残りの部分では、XML と JSON のドキュメントモデルで注意が必要な点をいくつか例示します。JSON における XPath の詳細については、「XPath を使用した JSON ドキュメントのトラバーサル」（349 ページ）を参照してください。

次のドキュメント例をリダクションするとします。

XML	JSON

<person>
  <name>
  <first>John</first>
  <last>Smith</last>
  </name>
  <id>1234</id>
  <alias>Johnboy</alias>
  <alias>Smitty</alias>
</person>
	{ "person": {
	  "name": {
	  "first": "John",
	  "last": "Smith"
	  },
	  "id": 1234,
	  "alias": ["Johnboy",
	"Smitty"],
	  "home phone": "123-4567"
	}}

次の表は、いくつかの XPath 式で選択されるノードをまとめたものです。

XPath	選択される XML ノード	選択される JSON ノード

`//id`	`an element node:`
	`  <id>1234</id>`
		`a number node equivalent to the constructor expression:`
		
		`number-node {"id":1234}`

`//alias`	`two element nodes`
	`  <alias>Johnboy`
	`  </alias>`
	`  <alias>Smitty`
	`  </alias>`
		`two text nodes, equivalent to the constructor expression:`
		
		`text {"Johnboy"}`
		`text {"Smitty"}`

`//node("alias")`	`two element nodes`
	`  <alias>Johnboy`
	`  </alias>`
	`  <alias>Smitty`
	`  </alias>`
		`An array node and two text nodes, equivalent to the constructor expressions:`
		
		`array-node {"Johnboy", "Smitty"}`
		`text {"Johnboy"}`
		`text {"Smitty"}`

`//array-node ("alias")`	`no match`	`An array node, equivalent to the constructor expression:`
		
		`array-node {"Johnboy", "Smitty"}`

`//alias/text()`	`two text nodes`	`no match`

`//name/data()`	`a string:`
	`  "JohnSmith"`
		`an object node:`
		`  {`
		`  "first": "John",`
		`  "last": "Smith"`
		`  }`

`//node("home phone")`	`N/A - invalid XML localname`	`a text node, equivalent to the constructor expression:`
		
		`text {"123-4567"}`

24.5.7 XML ルールシンタックスのリファレンス

XML で表現されるリダクションルールの形式は、次のとおりです。すべてのルール要素は、デフォルトの名前空間 `http://marklogic.com/xdmp/redaction` 内にある必要があります。また、名前空間プレフィックスを使用しないでください。JSON のシンタックスについては、「JSON ルールシンタックスのリファレンス」（452 ページ）を参照してください。

```
<rule xml:lang="zxx"
xmlns="http://marklogic.com/xdmp/redaction">
  <description>any text</description>
  <path>XPath expression</path>
  <namespaces>
    <namespace>
      <prefix>namespace prefix</prefix>
      <namespace-uri>uri</namespace-uri>
    </namespace>
  </namespaces>
  <method>
    <function>redaction function name</function>
    <module>user-defined module URI</module>
    <module-namespace>user-defined module namespace
    </module-namespace>
  </method>
  <options>params as elements</options>
</rule>
```

`rule/@xml:lang` が存在することに注意してください。`@lang` の値「zxx」は、有効な言語ではありません。「zxx」は、この要素をトークン化、ステミング、およびインデックス付けしないように MarkLogic に指示する特殊な値です。この設定をルールに含める必要はありませんが、含めることを強く推奨します。ルールは設定情報であり、検索できるように作成されたものではないためです。

次の表に、rule 子要素の詳細を示します。

要素	説明
description	オプション。このルールの説明です。
path	必須です。リダクション対象コンテンツを識別する XPath 式。式は、XML ノード、JSON ノード、またはその両方（要素、属性、オブジェクト、配列、テキスト、ブール、数値、または null ノードなど）を選択する絶対パス（「/」で始まる）である必要があります。ドキュメントノードが選択されないようにしてください。その他の制限がある場合があります。詳細については、「リダクションルールにおける XPath 式の制限」（447 ページ）を参照してください。
namespaces	オプション。path の XPath 式で名前空間プレフィックスを使用する場合は、プレフィックスと名前空間 URI のバインドをここで定義します。詳細については、「XML 名前空間プレフィックスバインドの定義」（446 ページ）を参照してください。

method	必須です。path にマッチするコンテンツに適用されるリダクション関数の仕様。function 子要素が必要です。module および module-namespace 子要素は、以下のように、ユーザー定義リダクション関数を指定するためにのみ使用されます。
	
	ビルトインリダクション関数を適用するには次の形式を使用します。詳細については、「ビルトインリダクション関数のリファレンス」（461 ページ）を参照してください。
	
	`<method>`
	`  <function>`*builtInFuncName*`</function>`
	`</method>`
	
	JavaScript で実装されたユーザー定義関数を適用するには次の形式を使用します。
	
	`<method>`
	`  <function>`*userDefinedFuncName*`</function>`
	`  <module>`*javascriptModuleURI*`</module>`
	`</method>`

要素	説明

	XQuery で実装されたユーザー定義関数を適用するには次の形式を使用します。
	
	`<method>`
	`  <function>`*`userDefinedFuncLocalName`*`</function>`
	`  <module>`*`xqueryModuleURI`*`</module>`
	`  <module-namespace>`*`moduleNSURI`*`</module-namespace>`
	`</method>`
	
	詳細については、「ユーザー定義のリダクション関数」（498 ページ）を参照してください。

`options`	オプション。リダクション関数に渡すデータを指定します。それぞれの子要素は、リダクション関数に渡される `options` パラメータ内でマップエントリ（XQuery）またはオブジェクトプロパティ（JavaScript）になります。要素名は、マップキーまたはプロパティ名です。

24.5.8 JSON ルールシンタックスのリファレンス

JSON で表現されるリダクションルールの形式は、次のとおりです。XML のシンタックスについては、「XML ルールシンタックスのリファレンス」（450 ページ）を参照してください。

```
{"rule": {
  "description": "any text",
  "path": "XPath expression",
  "method": {
    "function": "redaction function name",
    "module": "user-defined module URI",
    "moduleNamespace": "user-defined module namespace URI",
  },
  "namespaces": [
    {"namespace": {
      "prefix": "namespace prefix",
      "namespaceUri": "uri"
    }, ...
  ],
  "options": {
    "anyPropName": anyValue
  }
} }
```

次の表に、各要素の詳細を示します。

要素	説明
`description`	オプション。このルールの説明です。
`path`	必須です。リダクション対象コンテンツを識別する XPath 式。式は、XML ノード、JSON ノード、またはその両方（要素、属性、オブジェクト、配列、テキスト、ブール、数値、または null ノードなど）を選択する絶対パス（「/」で始まる）である必要があります。パスでドキュメントノードが選択されないようにしてください。その他の制限がある場合があります。詳細については、「リダクションルールにおける XPath 式の制限」（447 ページ）を参照してください。
`namespaces`	オプション。`path` の XPath 式で名前空間プレフィックスを使用する場合は、プレフィックスと名前空間 URI のバインドをここで定義します。詳細については、「XML 名前空間プレフィックスバインドの定義」（446 ページ）を参照してください。

`method`	必須です。`path` にマッチするコンテンツに適用されるリダクション関数の仕様。この要素は、以下に示すいずれかの形式にする必要があります。
	
	ビルトインリダクション関数を適用するには次の形式を使用します。詳細については、「ビルトインリダクション関数のリファレンス」（461 ページ）を参照してください。
	
	`"method": { "function": "builtInFuncName" }`
	
	JavaScript で実装されたユーザー定義関数を適用するには次の形式を使用します。
	
	`"method": {`
	`  "function": "userDefinedFuncName",`
	`  "module": "javascriptModuleURI"`
	`}`

要素	説明

	XQuery で実装されたユーザー定義関数を適用するには次の形式を使用します。
	
	`"method": {`
	`  "function": "userDefinedFuncName",`
	`  "module": "xqueryModuleURI",`
	`  "moduleNamespace": "xqueryModuleNSURI"`
	`}`
	
	詳細については、「ユーザー定義のリダクション関数」（498 ページ）を参照してください。

`options`	オプション。リダクション関数に渡すデータを指定します。これは、リダクション関数の `options` 入力パラメータの値になります。XQuery で実装されるリダクション関数の場合、オプションはプロパティ名をマップキーとして使用して、`map:map` として関数に渡されます。

24.6 リダクションルールのインストール

リダクションルールは、使用する前に、リダクション対象のドキュメントが含まれるデータベースに関連付けられているスキーマデータベース内にドキュメントとしてインストールする必要があります。

ルールドキュメントには 1 つのルールのみを含めることができます。また、ルール以外のデータは含めないでください。ルールコレクションには複数のルールドキュメントを含めることができますが、ルール以外のドキュメントは含めないでください。ルールはコレクションによってリダクション操作に指定されるため、各ルールドキュメントは、少なくとも 1 つの collection に関連付けられている必要があります。

ルールをスキーマデータベースに挿入するには、任意の MarkLogic ドキュメント挿入 API を使用します。例えば、`xdmp:document-insert` XQuery 関数や `xdmp.documentInsert` サーバーサイド JavaScript 関数、または Node.js、Java、または REST クライアント API のドキュメント作成機能を使用します。ルールをコレクションに割り当てるには、挿入時に、または別の操作として実行することができます。

スキーマデータベースをコンテキストデータベースとして使用し、次のいずれかの例を Query Console で実行すると、ルールドキュメントがデータベースに挿入され、2 つのコレクション（「pii-rules」および「security-rules」）に割り当てられます。

言語	例

XQuery	`xquery version "1.0-ml";`
	`xdmp:document-insert("/redactionRules/ssn.xml",`
	`  <rule xml:lang="zxx"`
	`  xmlns="http://marklogic.com/xdmp/redaction">`
	`  <description>hide SSNs</description>`
	`  <path>//ssn</path>`
	`  <method>`
	`  <function>redact-us-ssn</function>`
	`  </method>`
	`  <options>`
	`  <pattern>partial</pattern>`
	`  </options>`
	`  </rule>,`
	`  <options xmlns="xdmp:document-insert">`
	`  <permissions>{xdmp:default-permissions()}`
	`  </permissions>`
	`  <collections>`
	`  <collection>security-rules</collection>`
	`  <collection>pii-rules</collection>`
	`  </collections>`
	`  </options>`
	`)`

サーバーサイド JavaScript	`declareUpdate();`
	
	`xdmp.documentInsert(`
	`  '/redactionRules/ssn.json',`
	`  { rule: {`
	`  description: 'hide SSNs',`
	`  path: '//ssn',`
	`  method: { function: 'redact-us-ssn' },`
	`  options: { pattern: 'partial' }`
	`  }},`
	`  { permissions: xdmp.defaultPermissions(),`
	`  collections: ['security-rules','pii-rules']});`

ルールのアクセスや変更が可能なユーザーを制約するために、ルールドキュメントにパーミッションを設定してください。詳細については、「セキュリティ上の考慮事項」（439 ページ）を参照してください。

24.7 リダクションルールの適用

このセクションでは、ルールコレクションが MarkLogic にインストールされた後のリダクションルールの適用について説明します。次の内容が含まれます。

- 概要
- mlcp を使用したルール適用
- XQuery を使用したルール適用
- JavaScript を使用したルール適用
- ルールの順序は保証されない

推奨されるインターフェイスは mlcp コマンドラインツールです。mlcp を使用すると、データベースからエクスポートしたり、データベース間でコピーしたりするときに、多数のドキュメントにリダクションを効率的に適用できます。mlcp の詳細については、『*mlcp User Guide*』を参照してください。

`rdt:redact` および `rdt.redact` 関数は、リダクションルールのデバッグや、少数のドキュメントの集合に対するリダクションに適しています。

24.7.1 概要

1 つあるいは複数のルールドキュメントを Schemas データベースにインストールし、コレクションに割り当てた後は、次の方法でドキュメントをリダクションできます。

- mlcp コマンドラインツールを使用して、ドキュメントをデータベースからエクスポートする。
- mlcp コマンドラインツールを使用して、ドキュメントをデータベース間でコピーする。
- XQuery 関数 `rdt:redact` を呼び出す。
- サーバーサイド JavaScript 関数 `rdt.redact` を呼び出す。

mlcp コマンドラインツールを使用すると、最高のスループットを得られますが、ルールの開発やデバッグには、`rdt:redact` または `rdt.redact` を利用するほうが便利である可能性があります。

使用するリダクション方式とは関係なく、リダクション対象のドキュメントの集まりと、それらのドキュメントに適用する 1 つあるいは複数のルールコレクションを選択します。

リダクションを使用するときは、次の制限およびガイドラインに留意してください。

- 同じ操作で、XML ドキュメントと JSON ドキュメントの両方をリダクションできます。

- XML で定義されたルールを JSON ドキュメントに適用できます。その逆も可能です。
- リダクションルールは、XML ドキュメントと JSON ドキュメントにのみ適用できます。
- ドキュメントプロパティなどのドキュメントメタデータはリダクションできません。
- ルールは、順序どおりに適用されるとは限りません。詳細については、「ルールの順序は保証されない」（460 ページ）を参照してください。
- リダクション対象のドキュメントとリダクションルールの両方に対する読み取りパーミッションが必要です。
- ユーザー定義リダクション関数を使用するルールを適用する場合は、その実装が含まれるモジュールの実行パーミッションが必要です。詳細については、「セキュリティ上の考慮事項」（439 ページ）を参照してください。

いずれかのルールコレクションに無効なルールが含まれているか、ルールが含まれていない場合、リダクション操作は失敗します。`rdt:rule-validate` XQuery 関数または `rdt.ruleValidate` JavaScript 関数を使用して、ルールコレクションを適用前に検証できます。詳細については、「リダクションルールの検証」（460 ページ）を参照してください。

24.7.2 mlcp を使用したルール適用

mlcp `export` および `copy` コマンドを使用するときに、リダクションルールを適用できます。ソースデータベースからの読み取り時にドキュメントに適用する、1 つあるいは複数のルールコレクションを指定するには、`-redaction` オプションを使用します。リダクションは、ソースホストの MarkLogic によって実行されます。

次のコマンド例は、データベースディレクトリ「/employees/」内にあるドキュメントに、URI が「pii-rules」および「hipaa-rules」であるコレクション内のルールをエクスポート時に適用します。

```
# Windows users, see Modifying the Example Commands for Windows
$ mlcp.sh export -host localhost -port 8000 -username user
\
    -password password -mode local -output_file_path \
    /example/exported/files -directory_filter /employees/ \
   -redaction "pii-rules,hipaa-rules"
```

次の例は、同じルールを、mlcp によるコピー操作時に適用します。

```
$ mlcp.sh copy -mode local -input_host srchost -input_port
8000 \
```

```
    -input_username user1 -input_password password1 \
    -output_host desthost -output_port 8000
-output_username user2 \
    -output_password password2 -directory_filter
/employees/ \
   -redaction "pii-rules,hipaa-rules"
```

詳細については、『*mlcp User Guide*』の Redacting Content During Export or Copy Operations を参照してください。

24.7.3 XQuery を使用したルール適用

MarkLogic サーバーでドキュメントのリダクション済みインメモリコピーを作成するには、`rdt:redact` XQuery ライブラリ関数を使用します。この関数は、ルールのテストおよびデバッグ、または少量のドキュメントのリダクションに最適です。多数のリダクション済みドキュメントの集まりを MarkLogic から抽出するには、代わりに mlcp コマンドラインツールを使用してください。

次の例は、コレクション「personnel」内のドキュメントに、URI が「pii-rules」および「hipaa-rules」であるコレクション内のリダクションルールを適用します。

```
xquery version "1.0-ml";
import module namespace rdt =
"http://marklogic.com/xdmp/redaction"
    at "/MarkLogic/redaction.xqy";
rdt:redact(fn:collection("people"),
("pii-rules","hipaa-rules"))
```

出力は、ドキュメントノードのシーケンスであり、その各ドキュメントは、ルールコレクション内のルールを適用した結果です。結果には、リダクションルールで変更されたドキュメントと、変更されなかったドキュメントの両方が含まれます。変更されなかったのは、どのルールにもマッチしなかったか、リダクション関数で変更されなかったためです。

`rdt:redact` に渡されたルールコレクションのいずれかが空である場合は、`RDT-NORULE` 例外がスローされます。これにより、誤ってルールが適用されないという事態を防ぎます。結果的にリダクションが行われていないコンテンツの発生を防止することが出来ます。

ルールコレクションにルール以外のドキュメントが含まれる場合、いずれかのルールが無効である場合、またはルールのパス式でノード以外が選択される場合にも、例外がスローされます。`rdt:rule-validate` を使用すると、`rdt:redact` の呼び出し前にルールの有効性をテストできます。

24.7.4 JavaScript を使用したルール適用

MarkLogic サーバーでドキュメントのリダクション済みインメモリコピーを作成するには、`rdt.redact` JavaScript 関数を使用します。この関数は、ルールのテストおよびデバッグ、または少量のドキュメントのリダクションに最適です。多数のリダクション済みドキュメントの集まりを MarkLogic から抽出するには、代わりに mlcp コマンドラインツールを使用してください。

リダクション関数をアプリケーションのスコープに入れるには、`require` ステートメントを使用する必要があります。これらの関数は、XQuery ライブラリモジュール `/MarkLogic/redaction.xqy` によって実装されます。例：

```
const rdt = require('/MarkLogic/redaction');
```

次の例は、コレクション「personnel」内のドキュメントに、URI が「pii-rules」および「hipaa-rules」であるコレクション内のリダクションルールを適用します。

```
const rdt = require('/MarkLogic/redaction');
rdt.redact(fn.collection('people'),
['pii-rules','hipaa-rules'])
```

出力は、ドキュメントノードの `Sequence` であり、その各ドキュメントは、ルールコレクション内のルールを適用した結果です。`Sequence` は `Iterable` です。例えば、次のような for-of ループを使用して結果を処理できます。

```
const rdt = require('/MarkLogic/redaction');
const redacted =
  rdt.redact(fn.collection('people'), ['my-rules']);
for (let doc of redacted) {
  // do something with the redacted document
}
```

結果には、リダクションルールで変更されたドキュメントと、変更されなかったドキュメントの両方が含まれます。変更されなかったのは、どのルールにもマッチしなかったか、リダクション関数で変更されなかったためです。

`rdt.redact` に渡されたルールコレクションのいずれかが空である場合は、`RDT-NORULE` 例外がスローされます。これにより、誤ってルールが適用されないという事態を防ぎます。結果的にリダクションが行われていないコンテンツの発生を防止することが出来ます。ルールコレクションにルール以外のドキュメントが含まれる場合、いずれかのルールが無効である場合、またはルールのパス式でノード以外が選択される場合にも、例外がスローされます。

rdt.ruleValidate を使用すると、rdt.redact の呼び出し前にルールの有効性をテストできます。詳細については、「リダクションルールの検証」（460 ページ）を参照してください。

24.7.5 ルールの順序は保証されない

ルールが適用される順序は定義されていません。ルールコレクション内のルールが実行される順序と、同じリダクション操作で使用される複数のルールコレクションでのルール順序は、いずれも固定されていません。

また、リダクションされた最終結果に反映されるルールは、多くとも 1 つのみです。同じノードを選択するルールが複数ある場合は、そのすべてが実行されますが、リダクションによって生成される最終ドキュメントに反映されるのは、それらのルールの中の 1 つの結果だけです。

そのため、同じノードをリダクションまたは調査するルールを同一リダクション操作で複数使用しないでください。

例えば、2 つのルールコレクション、A と B があり、次のような特性があるとします。

```
Collection A contains:
  ruleA1 using path //id
  ruleA2 using path //id
Collection B contains:
  ruleB1 using path //id
```

ドキュメントの集まりに両方のルールコレクションを適用する場合、選択された id ノードに対して ruleA1、ruleA2、および ruleB1 が適用される順序は不明です。また、この順序を決定することもできません。また、ruleA1、ruleA2、ruleB1 のいずれかによって //id に加えられた変更のみが出力に反映されます。

24.8 リダクションルールの検証

rdt:rule-validate XQuery 関数または rdt.ruleValidate サーバーサイド JavaScript 関数を使用して、ルールコレクションの有効性を使用前にテストできます。無効なルールや空のルールコレクションがあるとリダクション操作が失敗するため、導入前にルールを検証してください。

検証によって、ルールおよびルールコレクションが、想定される構造に準拠していること、および存在しないコード（未定義のリダクション関数など）に依存していないことが確認されます。

ただし、ルールが検証によって確認されていても、ランタイムエラーが発生する可能性はあります。例えば、辞書ベースのマスキングがルールで使用される場合でも、ルール検証に辞書の検証は含まれません。同様に、ルール内の XPath 式が「リダクションルールにおける XPath 式の制限」（447 ページ）で説明する制限に準拠していることは検証されません。

入力ルールコレクション内のすべてのルールが有効である場合、検証関数は、すべての検証済みルールの URI を返します。それ以外の場合は、最初の検証エラーが発生した時点で例外がスローされます。

次の例は、URI が「pii-rules」および「hipaa-rules」である 2 つのルールコレクション内にあるルールを検証します。

言語	例

XQuery	`xquery version "1.0-ml";`
	`import module namespace rdt =`
	`  "http://marklogic.com/xdmp/redaction"`
	`  at "/MarkLogic/redaction.xqy";`
	`rdt:rule-validate(("pii-rules", "hipaa-rules"))`

JavaScript	`const rdt = require('/MarkLogic/redaction.xqy');` `rdt.ruleValidate(["pii-rules", "hipaa-rules"])`

24.9 ビルトインリダクション関数のリファレンス

MarkLogic には、リダクションルールで使用されるビルトインリダクション関数が複数用意されています。このような関数のいずれかを使用するには、次の形式の `method` 子 XML 要素または JSON プロパティを持つルールを作成します。

XML	JSON

`<method>`
`  <function>`*`builtInName`*
`  </function>`
`</method>`
	`"method": {`
	`  "function":`
	`"`*`builtInFuncName`*`"`
	`}`

ビルトインで設定パラメータを利用できる場合は、それらのパラメータをルールの `options` 子 XML 要素または JSON プロパティで指定します。シンタックスについては、「リダクションルールの定義」（442 ページ）を参照してください。パラメータの仕様と例については、各ビルトインのリファレンスセクションを参照してください。

次の表は、ビルトインリダクション関数と、想定される入力パラメータをまとめたものです。詳細と例については、各関数に関するセクションを参照してください。

関数名	説明
mask-deterministic	値を決定論的なマスキングテキストに置換します。つまり、特定の入力に対して、適用されるたびに同じマスク値を生成します。生成される値の長さを制御できます。
mask-random	値をランダムなテキストに置換します。マスキング値は、同一の入力値に適用されるたびに異なる可能性があります。生成される値の長さと置換テキストのタイプ（数字または文字）を制御できます。
conceal	マスク対象の値を削除します。
redact-number	値をランダムな数字に置換します。マスキング値のデータ型、レンジ、および形式を制御できます。
redact-us-ssn	米国の社会保障番号（SSN）のパターンにマッチするデータをリダクションします。末尾 4 桁を保持するかどうか、およびマスキング文字として使用する文字を制御できます。
redact-us-phone	米国の電話番号のパターンにマッチするデータをリダクションします。末尾 4 桁を保持するかどうか、およびマスキング文字として使用する文字を制御できます。
redact-email	メールアドレスのパターンにマッチするデータをリダクションします。マスクする対象をアドレス全体、ユーザー名のみ、ドメイン名のみのどれにするかを制御できます。
redact-ipv4	IPv4 アドレスのパターンにマッチするデータをリダクションします。マスキング文字として使用する文字を制御できます。
redact-regex	特定の正規表現にマッチするデータをリダクションします。正規表現とマスキングテキストを指定する必要があります。

すべてのビルトイン関数の詳細な使用例については、「例：ビルトインリダクション関数の使用」（487 ページ）を参照してください。

24.9.1 mask-deterministic

このビルトインは、一定のマスク値で値をマスクするために使用します。つまり、決定論的マスキングでは、特定の入力によって常に同じ出力が生成されます。元の値は、マスク後の値から導き出すことはできません。

決定論的マスキングは、レコード間の関係性を維持する場合に役立ちます。例えば、ソーシャルネットワークで名前がマスクされていても、人々の関係性（X は Y の知り合いで、Z は Y の知り合い）をトレースできます。

この関数の動作を設定するには、次のパラメータを使用します。パラメータは、ルールの `options` セクションで設定します。

- `length`：生成する出力値の長さ（文字数）。オプション。デフォルト：64。このオプションは、`dictionary` オプションと同時には使用できません。
- `character`：マスク後の値を作成するときに使用する文字のクラス。使用可能な値：`any`（デフォルト）、`alphanumeric`、`numeric`、`alphabetic`。このオプションは、`dictionary` オプションと同時には使用できません。
- `dictionary`：リダクション辞書の URI。この辞書は、置換値のソースとして使用します。このオプションは、他のオプションと同時には使用できません。

辞書ベースのマスキングを使用すると、特定の入力が常に同一のリダクション辞書エントリにマッピングされます。辞書を変更すると、辞書のマッピングも変更されます。

次のルール例は、XPath 式「//name」で選択されるノードに決定論的マスキングを適用します。置換値の長さは、`length` オプションに従って 10 文字になります。

XML	JSON

`<rule xml:lang="zxx"`
`  xmlns="http://marklogic.com/xdmp/`
`  redaction">`
`  <path>//name</path>`
`  <method>`
`  <function>`
`  mask-deterministic`
`  </function>`
`  </method>`
`  <options>`
`  <length>10</length>`
`  </options>`
`</rule>`
	`{"rule": {`
	`  "path": "//name",`
	`  "method": {`
	`  "function":`
	`"mask-deterministic",`
	`  },`
	`  "options": {`
	`  "length": 10`
	`  }`
	`} }`

次の表は、さまざまなタイプのノードに mask-deterministic を適用した効果を示しています。完全な例については、「例：ビルトインリダクション関数の使用」（487 ページ）を参照してください。

パス式	形式	元のドキュメント	リダクションの結果
//name シンプルな原子値	XML	<person> <name>Little Bopeep </name> </person>	<person> <name>8d1f713a30 </name> </person>

	JSON	{
		"name": "Georgie Porgie"
		}
			{
			"name": "34fe55c66a"
			}

//alias 複数の項目（JSON 配列）	XML	<person> <alias>Peepers</alias> <alias>Bo</alias> </person>	<person> <alias>7a4fabd518 </alias> <alias>850517542f </alias> </person>
	JSON	{ "alias": ["George", "GP"] }	{ "alias": ["ef36ccc0c8", "fa6f1defad"] }
//address 複素数値	XML	<person> <address> <street> 100 Nursery Lane </street> <city>Hometown</city> <country> Neverland </country> </address> </person>	<person> <address> 8d1f713a30 </address> </person>
	JSON	{"address": { "street": "300 Nursery Lane", "city": "Hometown", "country": "Neverland" }}	{ "address": "fc1f5fcb6d"}

ほとんどの場合、ノードの値全体が、リダクションされた値に置換されます。前述の `//address` の例のように、元のコンテンツが複合的である場合も同様です。

ただし、前述の `//alias` の例では、配列全体ではなく、JSON の例に示す個々の `alias` 配列項目が選択されることに注意してください。配列値全体をリダクションする場合は、JSON 固有のパスセレクタを使用したルールが必要です。例えば、ルールパス `//array-node('alias')` は、JSON ドキュメント内の配列全体を選択します。その結果、「alias」プロパティの値は次のようになります。

```
"alias": "6b162c290e"
```

詳細については、「複数のドキュメント形式で使用可能なルールの定義」（447 ページ）を参照してください。

さまざまな `character` オプション設定の効果を説明するために、`length` オプションが 10 であり、次の入力がリダクション対象であるとします。

```
<pii>
  <priv>redact me</priv>
  <priv>redact me</priv>
  <priv>redact me too</priv>
</pii>
```

次の表は、`character` オプションで使用可能な各値を適用した結果を示しています。

character 設定	リダクションされた値
any（デフォルト）	<pii> <priv>3ba1a188e6</priv> <priv>3ba1a188e6</priv> <priv>a62597fd0c</priv> </pii>
alphanumeric	<pii> <priv>F1Fp64Cnox</priv> <priv>F1Fp64Cnox</priv> <priv>LiN5mrmG0g</priv> </pii>
numeric	<pii> <priv>1838664450</priv> <priv>1838664450</priv> <priv>5771438029</priv> </pii>
alphabetic	<pii> <priv>PQXWBHfASy</priv> <priv>PQXWBHfASy</priv> <priv>ZroFQNkNqi</priv> </pii>

24.9.2 mask-random

このビルトインは、ランダムなマスキング値に値を置換するために使用します。特定の入力により、適用されるたびに異なる出力が生成されます。元の値は、マスク後の値から導き出すことはできません。ランダムマスキングは、レコード間の関係性を不明瞭にするときに役立ちます。

この関数の動作を設定するには、次のパラメータを使用します。パラメータは、ルールの `options` セクションで設定します。

- `length`：生成する出力値の長さ（文字数）。オプション。デフォルト：64。この `option` は、dictionary オプションと同時には使用できません。
- `character`：マスク後の値を作成するときに使用する文字のタイプ。使用可能な値：`any`（デフォルト）、`alphanumeric`、`numeric`、`alphabetic`。このオプションは、`dictionary` オプションと同時には使用できません。
- `dictionary`：リダクション辞書の URI。この辞書は、置換値のソースとして使用します。このオプションは、他のオプションと同時には使用できません。

次のルール例は、XPath 式「//name」で選択されるノードにランダムマスキングを適用します。置換値の長さは、`length` オプションに従って 10 文字になります。

XML	JSON

`<rule xml:lang="zxx"`
`  xmlns="http://marklogic.com/xdmp/`
`  redaction">`
`  <path>//name</path>`
`  <method>`
`  <function>`
`  mask-random`
`  </function>`
`  </method>`
`  <options>`
`  <length>10</length>`
`  </options>`
`</rule>`
	`{"rule": {`
	`  "path": "//name",`
	`  "method": {`
	`  "function":`
	`"mask-random",`
	`  },`
	`  "options": {`
	`  "length": 10`
	`  }`
	`} }`

次の表は、さまざまなタイプのノードに mask-random を適用した効果を示しています。完全な例については、「例：ビルトインリダクション関数の使用」（487 ページ）を参照してください。

パス式	形式	元のドキュメント	リダクションの結果
//name シンプルな原子値	XML	<person> <name>Little Bopeep</name> </person>	<person> <name>8d1f713a30 </name> </person>

	JSON	{
		  "name": "Georgie Porgie"
		}
			{
			  "name":
			  "34fe55c66a"
			}

//alias 複数の項目（JSON 配列）	XML	<person> <alias>Peepers</alias> <alias>Bo</alias> </person>	<person> <alias>7a4fabd518 </alias> <alias>850517542f </alias> </person>

	JSON	{ "alias": ["George", "GP"] }	{ "alias": [
			  "ef36ccc0c8",
			  "fa6f1defad"
			] }

//address
 	
複素数値
	XML	<person>
		  <address>
		  <street>
		  100 Nursery Lane
		  </street>
		  <city>Hometown</city>
		  <country>
		  Neverland
		  </country>
		  </address>
		</person>
			<person>
			  <address>
			  8d1f713a30
			  </address>
			</person>

	JSON	{"address": {
		  "street": "300 Nursery Lane",
		  "city": "Hometown",
		  "country": "Neverland"
		}}
			{ "address":
			"fc1f5fcb6d"}

ほとんどの場合、ノードの値全体が、リダクションされた値に置換されます。前述の //address の例のように、元のコンテンツが複合的である場合も同様です。

ただし、前述の `//alias` の例では、配列全体ではなく、JSON の例に示す個々の `alias` 配列項目が選択されることに注意してください。配列値全体をリダクションする場合は、JSON 固有のパスセレクタを使用したルールが必要です。例えば、ルールパス `//array-node('alias')` は、JSON ドキュメント内の配列全体を選択します。その結果、「alias」プロパティの値は次のようになります。

```
"alias": "6b162c290e"
```

詳細については、「複数のドキュメント形式で使用可能なルールの定義」（447 ページ）を参照してください。

さまざまな `character` オプション設定の効果を説明するために、`length` オプションが 10 であり、次の入力がリダクション対象であるとします。

```
<pii>
  <priv>redact me</priv>
  <priv>redact me</priv>
  <priv>redact me too</priv>
</pii>
```

次の表は、`character` オプションで使用可能な各値を適用した結果を示しています。

character 設定	リダクションされた値
any（デフォルト）	<pii> <priv>2457f4f294</priv> <priv>f18e883ba9</priv> <priv>e5b253aea9</priv> </pii>
alphanumeric	<pii> <priv>qIEsmeJua6</priv> <priv>WfVLAAckzu</priv> <priv>P8BGgCdt5s</priv> </pii>
numeric	<pii> <priv>7902282158</priv> <priv>8313199931</priv> <priv>2026296703</priv> </pii>
alphabetic	<pii> <priv>rZimfgZwSG</priv> <priv>knqbTrKTdl</priv> <priv>wKYeTkVjLC</priv> </pii>

24.9.3 conceal

このビルトインは、選択した値を完全に除去するために使用します。

次のルール例は、パス式 `//name` で選択される値に隠ぺい化を適用します。

XML	JSON

<rule xml:lang="zxx"
xmlns="http://marklogic.com/xdmp/
redaction">
<path>//name</path>
<method>
<function>conceal</function>
</method>
</rule>
	{"rule": {
	"path": "//name",
	"method": {
	"function": "conceal",
	}
	} }

次の表は、さまざまなタイプのノードに `conceal` を適用した効果を示しています。完全な例については、「例：ビルトインリダクション関数の使用」（487 ページ）を参照してください。

パス式	形式	元のドキュメント	リダクションの結果
//name シンプルな原子値	XML	<person> <name> Little Bopeep </name> <id>12-3456789</id> </person>	<person> <id>12-3456789 </id> </person>
	JSON	{ "name":"Jack Sprat", "id": "45-6789123" }	{ "id": "45-6789123" }
//alias 複数の項目（JSON 配列）	XML	<person> <alias>Peepers </alias> <alias>Bo</alias> <id>12-3456789</id> </person>	<person> <id>12-3456789 </id> </person>
	JSON	{ "alias": ["George", "G.P."], "id": "45-6789123" }	{ "alias": [], "id": "45-6789123" }

パス式	形式	元のドキュメント	リダクションの結果

//address
 	
複素数値
	XML	<person>
		  <address>
		  <street>
		  100 Nursery Lane
		  </street>
		
		<city>Hometown</city>
		  <country>
		  Neverland
		  </country>
		  </address>
		  <id>12-3456789</id>
		</person>
			<person>
			  <id>12-3456789
			  </id>
			</person>

	JSON	{"address": {
		  "street": "300
		  Nursery Lane",
		  "city": "Hometown",
		  "country":
		"Neverland"
		  },
		  "id": "45-6789123"
		}
			{ "id":
			"45-6789123" }

ほとんどの場合、選択されたノード全体が隠ぺい化されます。前述の //address の例のように、元のコンテンツが複合的である場合も同様です。

ただし、前述の //alias のようなパスでは、配列全体ではなく、JSON の例に示す個々の配列項目が隠ぺい化されることに注意してください。これは、alias パスステップが個々の配列項目にマッチするためです。詳細については、「複数のドキュメント形式で使用可能なルールの定義」（447 ページ）および「XPath を使用した JSON ドキュメントのトラバーサル」（349 ページ）を参照してください。

配列値全体をリダクションするには、JSON 固有のパスセレクタ（//array-node('alias') など）を使用したルールが必要です。詳細については、「複数のドキュメント形式で使用可能なルールの定義」（447 ページ）を参照してください。

24.9.4 redact-number

このビルトインは、設定可能なレンジと形式に準拠しているランダムな数値で値をマスクするために使用します。

この関数が `mask-random` 関数と異なる点は、マスキング値を細かく制御できることです。また、`mask-random` は常にテキストノードを生成しますが、`redact-number` は、設定に応じて数値ノードまたはテキストノードを生成します。

`redact-number` 関数を使用すると、マスキング値の次の特徴を制御できます。

- 最小値や最大値を指定して、値を一定のレンジに制限する。
- 特定の数値型（integer、decimal、または double）に値を制限する。
- 「ピクチャ文字列」を使用して、値の形式を指定する。例えば、小数部の桁数を制限したり、通貨記号（ドル記号など）を含めたりします。

この関数の動作を設定するには、次のオプションを使用します。

- `min`：利用可能な最小のマスキング値（この値を含む）。この関数は、`min` 値より小さいマスキング値を生成しません。オプション。デフォルト：0。
- `max`：利用可能な最大のマスキング値（この値を含む）。この関数は、`max` 値より大きいマスキング値を生成しません。オプション。デフォルト：18446744073709551615。
- `format`：置換値に適用される特殊な書式。オプション。デフォルト：特殊な書式なし。書式文字列は、XSLT の「ピクチャ文字列」に準拠する必要があります。ピクチャ文字列については、`fn:format-number`（XQuery）または `fn.formatNumber`（JavaScript）の関数リファレンス、および https://www.w3.org/TR/xslt20/#function-format-number で説明されています。形式を指定する場合、置換値は JSON ドキュメントの数値ノードではなく、テキストノードです。注：形式を指定する場合、`min` および `max` で定義されるレンジ内の値は、小数に変換可能である必要があります。
- `type`：置換値のデータ型。オプション。使用可能な値：`integer`、`decimal`、`double`。デフォルト：`integer`。`min` および `max` オプションで指定する値は、ここで指定する型制限の対象です。

次のルール例は、XPath 式 `//balance` で選択される値に `redact-number` を適用します。マッチした値は、0.0 ～ 100000.00 のレンジ内の、小数部が 2 桁の小数値に置換されます。このルールによって、3.55、19.79、82.96 などの置換値が生成されます。

XML	JSON

<rdt:rule xml:lang="zxx"
xmlns:rdt="http://marklogic.com/
xdmp/redaction">
<rdt:path>//balance</rdt:path>
<rdt:method>
<rdt:function>redact-number
</rdt:function>
</rdt:method>
<rdt:options>
<min>1</min>
<max>100000</max>
<format>0.00</format>
<type>decimal</type>
</rdt:options>
</rdt:rule>
	{"rule": {
	"path": "//balance",
	"method": {
	"function":
	"redact-number",
	},
	"options": {
	"min": 1,
	"max": 100000,
	"format": "0.00",
	"type": "decimal
	}
	} }

JSON ドキュメントに適用されると、format オプションを使用したかどうかに応じて、リダクションで置換されたノードがテキストノードまたは数値ノードになります。書式を明示しない場合は、リダクションによって JSON の数値ノードが生成されます。書式を明示した場合は、テキストノードが生成されます。例えば、redact-number は、次のように JSON プロパティ「key」の値に影響を与える可能性があります。

```
no format option
"key": 61.4121623617221

format option value "0.00"
"key": "61.41"
```

redact-number ルールで定義される値レンジは、そのデータ型に有効である必要があります。例えば、次の一連のオプションは無効です。指定したレンジが、値の生成元として意味のある整数レンジを表していません。

```
min: 0.1
max: 0.9
type: integer
```

min および max の値は、指定された型にキャスト可能である必要があります。

次の表は、さまざまなオプションの組み合わせで redact-number を適用した効果を示しています。完全な例については、「例：ビルトインリダクション関数の使用」（487 ページ）を参照してください。

オプション設定	形式	リダクション結果の例
default (no options)	XML	<balance>8137497966986464072</balance> <balance>2363247638359197582</balance>
	JSON	"balance": 8137497966986464072 "balance": 2363247638359197582
min: 100 max: 10000	XML	<balance>3842</balance> <balance>6622</balance>
	JSON	"balance": 3842 "balance": 6622
min: 100 max: 10000 type: decimal	XML	<balance>100.82</balance> <balance>269.419736229</balance>
	JSON	"balance": 100.82 "balance": 269.419736229
min: 100 max: 10000 type: decimal format: 0.00	XML	<balance>102.77</balance> <balance>9596.90</balance>
	JSON	"balance": "102.77" "balance": "9596.90" Note that masking values are text nodes due to the use of the format option.

24.9.5 redact-us-ssn

このビルトインは、次のいずれかのパターンに準拠している値をマスクするために使用します。これらのパターンは、米国の社会保障番号（SSN）で一般的な表現に対応しています。これらのパターンで、*N* はレンジ 0 ～ 9 の 1 桁を表しています。

- *NNN-NN-NNNN*（ダッシュ区切り）
- *NNN.NN.NNNN*（ドット区切り）
- *NNN NN NNNN*（スペース区切り）
- *NNNNNNNNN*

パターンマッチが見つかると、リダクションされる各桁がすべて、同じ文字に置換されます。例えば、「123-45-6789」のような値は、ルール設定に応じて「XXX-XX-XXXX」になる可能性があります。

この関数の動作を設定するには、次のパラメータを使用できます。パラメータは、ルールの `options` セクションで設定します。

- `level` : リダクションの程度。オプション。このオプションに設定できる値は次のとおりです。
  - `full` : デフォルトです。すべての桁を、`character` オプションで指定した文字に置換します。
  - `partial` : 末尾 4 桁は保持します。その他の桁はすべて、`character` オプションで指定した `character` に置換します。
  - `full-random` : すべての桁をランダムな数字に置換します。`character` オプションは無視されます。特定の値をリダクションするたびに、異なる値になります。
- `character` : `level` が `full` または `partial` の場合に、リダクションされる各桁が、この文字に置換されます。オプション。デフォルト : 「#」。

次の例は、パス式 `//id` で選択される SSN をリダクションします。パラメータは、SSN の末尾 4 桁を保持し、残りの桁を文字「X」に置換するように指定します。

XML	JSON

`<rule xml:lang="zxx"`
`  xmlns="http://marklogic.com/xdmp/`
`  redaction">`
`  <path>//id</path>`
`  <method>`
`  <function>redact-us-ssn</function>`
`  </method>`
`  <options>`
`  <level>partial</level>`
`  <character>X</character>`
`  </options>`
`</rule>`
	`{"rule": {`
	`  "path": "//id",`
	`  "method": {`
	`  "function":`
	`"redact-us-ssn",`
	`  },`
	`  "options": {`
	`  "level": "partial",`
	`  "character": "X"`
	`  }`
	`} }`

次の表は、さまざまな入力値および設定パラメータで redact-us-ssn を適用した効果を示しています。完全な例については、「例：ビルトインリダクション関数の使用」（487 ページ）を参照してください。

設定	形式	元のドキュメント	リダクションの結果
Path: //ssn Level: full Char: # （デフォルト）	XML	<pii> <ssn>123-45-6789 </ssn> <ssn>123.45.6789 </ssn> <ssn>123456789 </ssn> </pii>	<pii> <ssn>###-##-#### </ssn> <ssn>###.##.#### </ssn> <ssn>######### </ssn> </pii>
	JSON	{"pii": { ssn: ["123-45-6789", "123.45.6789", "123456789"] } }	{"pii": { ssn: ["###-##-####", "###.##.####", "#########"] } }
Path: //ssn Level: partial	XML	<pii> <ssn>123-45-6789 </ssn> <ssn>123.45.6789 </ssn> <ssn>123456789 </ssn> </pii>	<pii> <ssn>###-##-6789 </ssn> <ssn>###.##.6789 </ssn> <ssn>#####6789 </ssn> </pii>
	JSON	{"pii": { ssn: ["123-45-6789", "123.45.6789", "123456789"] } }	{"pii": { ssn: ["###-##-6789", "###.##.6789", "#####6789"] } }

設定	形式	元のドキュメント	リダクションの結果
Path: //ssn Level: full-random	XML	<pii> <ssn>123-45-6789 </ssn> <ssn>123.45.6789 </ssn> <ssn>123456789 </ssn> </pii>	<pii> <ssn>492-54-3352 </ssn> <ssn>441.65.4885 </ssn> <ssn>501965954 </ssn> </pii>
	JSON	{"pii": { ssn: ["123-45-6789", "123.45.6789", "123456789"] } }	{"pii": { ssn: ["492-54-3352", "441.65.4885", "501965954"] } }
Path: //ssn Level: full Character: X	XML	<pii> <ssn>123-45-6789 </ssn> <ssn>123.45.6789 </ssn> <ssn>123456789 </ssn> </pii>	<pii> <ssn>XXX-XX-XXXX </ssn> <ssn>XXX.XX.XXXX </ssn> <ssn>XXXXXXXXX </ssn> </pii>
	JSON	{"pii": { ssn: ["123-45-6789", "123.45.6789", "123456789"] } }	{"pii": { ssn: ["XXX-XX-XXXX", "XXX.XX.XXXX", "XXX.XX.XXXX"] } }

24.9.6 redact-us-phone

このビルトインは、次のいずれかのパターンに準拠している値をマスクするために使用します。これらのパターンは、米国の電話番号で一般的な表現に対応しています。これらのパターンで、*N* はレンジ 0 ～ 9 の 1 桁を表しています。

- *NNN-NNN-NNNN*（「-」区切り）
- *NNN.NNN.NNNN*（「.」区切り）
- (*NNN*)*NNN-NNNN*（スペース使用不可）
- *NNNNNNNNNN*

パターンマッチが見つかると、リダクションされる各桁がすべて、同じ文字に置換されます。例えば「123-456-7890」のような値は、ルール設定に応じて「XXX-XXX-XXXX」になる可能性があります。

この関数の動作を設定するには、次のパラメータを使用できます。パラメータは、ルールの `options` セクションで設定します。

- `level` : リダクションの程度。オプション。このオプションに設定できる値は次のとおりです。
  - `full` : デフォルトです。すべての桁を、`character` オプションで指定した文字に置換します。
  - `partial` : 末尾 4 桁は保持します。その他の桁はすべて、character オプションで指定した `character` に置換します。
  - `full-random` : すべての桁をランダムな数字に置換します。`character` オプションは無視されます。特定の入力をリダクションするたびに、異なるランダム値になります。
- `character` : `level` が `full` または `partial` の場合に、リダクションされる各桁が、この文字に置換されます。オプション。デフォルト :「#」。

次の例は、パス式 `//id` で選択される電話番号をマスクします。パラメータは、電話番号の末尾 4 桁を保持し、残りの桁を文字「X」に置換するように指定します。

XML	JSON

`<rule xml:lang="zxx"`
`  xmlns="http://marklogic.com/xdmp/`
`  redaction">`
`  <path>//ph</path>`
`  <method>`
`  <function>redact-us-phone`
`  </function>`
`  </method>`
`  <options>`
`  <level>partial</level>`
`  <character>X</character>`
`  </options>`
`</rule>`
	`{"rule": {`
	`  "path": "//ph",`
	`  "method": {`
	`  "function":`
	`"redact-us-phone",`
	`  },`
	`  "options": {`
	`  "level": "partial",`
	`  "character": "X"`
	`  }`
	`} }`

次の表は、さまざまな入力値および設定パラメータで redact-us-phone を適用した効果を示しています。完全な例については、「例：ビルトインリダクション関数の使用」（487 ページ）を参照してください。

設定	形式	元のドキュメント	リダクションの結果
Path: //ph Level: full Char: # （デフォルト）	XML	<pii> <ph>123-456-7890 </ph> <ph>123.456.7890 </ph> <ph>(123)456-7890 </ph> <ph>1234567890</ph> </pii>	<pii> <ph>###-###-#### </ph> <ph>###.###.#### </ph> <ph>(###)###-#### </ph> <ph>##########</ph> </pii>
	JSON	{"pii": { "ph": ["**123-456-789**0", "123.456.7890", "(123)456-7890", "1234567890"] } }	{"pii": { "ph": ["###-###-####", "###.###.####", "(###)###-####", "##########"] } }
Path: //ph Level: partial Char: #	XML	<pii> <ph>123-456-7890 </ph> <ph>123.456.7890 </ph> <ph>(123)456-7890 </ph> <ph>1234567890</ph> </pii>	<pii> <ph>###-###-7890 </ph> <ph>###.###.7890 </ph> <ph>(###)###-7890 </ph> <ph>######7890</ph> </pii>
	JSON	{"pii": { "ph": ["123-456-7890", "123.456.7890", "(123)456-7890", "1234567890"] } }	{"pii": { "ph": ["###-###-7890", "###.###.7890", "(###)###-7890", "######7890"] } }

設定	形式	元のドキュメント	リダクションの結果
Path: //ph Level: full-random Char: #	XML	<pii> <ph>123-456-7890 </ph> <ph>123.456.7890 </ph> <ph>(123)456-7890 </ph> <ph>1234567890</ph> </pii>	<pii> <ph>291-826-5242 </ph> <ph>121.350.3951 </ph> <ph>(804)380-8192 </ph> <ph>9644991161</ph> </pii>
	JSON	{"pii": { "ph": ["123-456-7890", "123.456.7890", "(123)456-7890", "1234567890"] } }	{"pii": { "ph": ["291-826-5242", "121.350.3951", "(804)380-8192", "9644991161"] } }
Path: //ph Level: full Character: X	XML	<pii> <ph>123-456-7890 </ph> <ph>123.456.7890 </ph> <ph>(123)456-7890 </ph> <ph>1234567890</ph> </pii>	<pii> <ph>XXX-XXX-XXXX </ph> <ph>XXX.XXX.XXXX </ph> <ph>(XXX)XXX-XXXX </ph> <ph>XXXXXXXXXX</ph> </pii>
	JSON	{"pii": { "ph": ["**123-456-789**0", "123.456.7890", "(123)456-7890", "1234567890"] } }	{"pii": { "ph": ["XXX-XXX-XXXX", "XXX.XXX.XXXX", "(XXX)XXX-XXXX", "XXXXXXXXXX"] } }

24.9.7 redact-email

このビルトインは、メールアドレスのパターンに準拠している値をマスクするために使用します。この関数は、メールの形式が `name@domain` であることを前提としています。

この関数の動作を設定するには、次のパラメータを使用します。パラメータは、ルールの `options` セクションで設定します。

- `level`：各メールアドレスをリダクションする程度。使用可能な値：`full`、`name`、`domain`。オプション。デフォルト：`full`。

メールアドレスのユーザー名部分をリダクションする場合、ユーザー名を「NAME」に置換します。メールアドレスのドメイン部分をリダクションする場合、ドメイン名を「DOMAIN」に置換します。つまり、メールアドレス「jsmith@example.com」を完全にリダクションすると、置換値「NAME@DOMAIN」が生成されます。

次のルール例は、パス式「//email」で選択されるメールアドレスを完全にリダクションします。

XML	JSON

`<rule xml:lang="zxx"`
`  xmlns="http://marklogic.com/xdmp/`
`  redaction">`
`  <path>//email</path>`
`  <method>`
`  <function>redact-email</function>`
`  </method>`
`  <options>`
`  <level>full</level>`
`  </options>`
`</rule>`
	`{"rule": {`
	`  "path": "//email",`
	`  "method": {`
	`  "function":`
	`"redact-email",`
	`  },`
	`  "options": {`
	`  "level": "full"`
	`  }`
	`} }`

次の表は、さまざまなリダクションレベルで redact-email を適用した効果を示しています。完全な例については、「例：ビルトインリダクション関数の使用」（487 ページ）を参照してください。

設定	形式	元のドキュメント	リダクションの結果
//email level: full（デフォルト）	XML	<person> <email> bopeep@mothergoose.com </email> </person>	<person> <email> NAME@DOMAIN </email> </person>
	JSON	{"email": "gp@mothergoose.com" }	{"email": "NAME@DOMAIN" }
//email level: name	XML	<person> <email> bopeep@mothergoose.com </email> </person>	<person> <email> NAME@mothergoose.com </email> </person>
	JSON	{"email": "gp@mothergoose.com" }	{"email": "NAME@mothergoose.com" }
//email level: domain	XML	<person> <email> bopeep@mothergoose.com </email> </person>	<person> <email> bopeep@DOMAIN </email> </person>
	JSON	{"email": "gp@mothergoose.com" } }	{"email": "gp@DOMAIN"}

24.9.8 redact-ipv4

このビルトインは、アドレスのパターンに準拠している値をマスクするために使用します。この関数は、IPv4 アドレスのみをリダクションします。つまり、次のパターンに準拠している値がリダクションされます（*N* は数字の桁（0 ～ 9）を表す）。

- 1 ～ 3 桁の数字から成る、ピリオド（「.」）で区切られた 4 つのブロック。各ブロックの値は、255 以下である必要があります。例：`123.201.098.112`、`123.45.678.0`。

リダクションされた IP アドレスは、最大桁数分の文字が含まれるように正規化されます。つまり、123.4.56.7 のようなアドレスがマスクされると、「###.###.###.###」になります。

この関数の動作を設定するには、次のオプションを使用します。パラメータは、ルールの `options` セクションで設定します。

- `character`：リダクションされる各桁を置換する文字。オプション。デフォルト：「#」。

次のルール例は、パス式 `//ip` で選択される IP アドレスをリダクションします。`character` パラメータは、リダクションされる IP アドレスの桁を「X」に置換するように指定します。

XML	JSON

`<rule xml:lang="zxx"`
`  xmlns="http://marklogic.com/xdmp/`
`  redaction">`
`  <path>//ip</path>`
`  <method>`
`  <function>redact-ipv4</function>`
`  </method>`
`  <options>`
`  <character>X</character>`
`  </options>`
`</rule>`
	`{"rule": {`
	`  "path": "//ip",`
	`  "method": {`
	`  "function":`
	`"redact-ipv4",`
	`  },`
	`  "options": {`
	`  "character": "X"`
	`  }`
	`} }`

次の表は、さまざまな設定オプションで `redact-ipv4` を適用した効果を示しています。完全な例については、「例：ビルトインリダクション関数の使用」（487 ページ）を参照してください。

設定	形式	元のドキュメント	リダクションの結果

`//ip`
 	
`default`
	XML	`<person>`
		`  <ip>123.45.6.78`
		`  </ip>`
		`</person>`
			`<person>`
			`  <ip>###.###.###.###`
			`  </ip>`
			`</person>`

	JSON	`{"ip": "123.45.6.78"}`	`{"ip":` `"###.###.###.###"}`

`//ip`
 	
`character:`
`X`
	XML	`<person>`
		`  <ip>123.45.6.78`
		`  </ip>`
		`  <ip>123.145.167.189`
		`  </ip>`
		`</person>`
			`<person>`
			`  <ip>`**`XXX.XXX.XXX.XXX`**
			`  </ip>`
			`  <ip>`**`XXX.XXX.XXX.XXX`**
			`  </ip>`
			`</person>`

	JSON	`{"ip": [`
		`  "123.45.6.78",`
		`  "123.145.167.189"`
		`]}`
			`{"ip": [`
			`  "`**`XXX.XXX.XXX.XXX`**`",`
			`  "`**`XXX.XXX.XXX.XXX`**`"`
			`]}`

24.9.9 redact-regex

このビルトインは、正規表現にマッチする値をマスクするために使用します。正規表現と置換テキストを設定できます。

この関数の動作を設定するには、次のオプションを使用します。

- `pattern`：リダクション対象の値を識別する正規表現。必須です。XQuery および XPath 用に定義されている正規表現言語のシンタックスを使用します。詳細については、http://www.w3.org/TR/xpath-functions/%23regex-syntax を参照してください。
- `replacement`：`pattern` にマッチする値を置換するテキスト。

パターンと置換テキストは、`fn:replace` XQuery 関数または `fn.replace` サーバーサイド JavaScript 関数を呼び出したときのように入力値に適用されます。

置換パターンには、マッチしたテキストの一部への後方参照を含めることができます。後方参照を使用すると、マッチしたテキストの一部を「キャプチャ」して、置換値内で再利用できます。このセクションの最後にある例を参照してください。

正規表現パターンには、ルール定義内でエスケープ処理が必要な文字を含めることができます。該当する文字の例を次に示します。該当する文字は、これら以外にもあります。

- XQuery でインストールされる XML ルール内のパターンに含まれる波括弧（「{ }」）は、「{{」および「}}」のようにエスケープ処理される必要があります。これは、XQuery インタプリタによって、コードブロックの区切り文字として扱われないようにするためです。
- XML ルールの左角括弧（「<」）は、実体参照「<」に置換される必要があります。
- JSON ルール定義のバックスラッシュ（「\」）は、「\\」のようにエスケープ処理される必要があります。「\」は、JSON 文字列では特殊文字であるためです。

次の例は、次のいずれかの形式のテキストをリダクションします（*N* はレンジ 0 ～ 9 の 1 桁を表す）。

- *NN-NNNNNNN*（ダッシュ区切り）
- *NN.NNNNNNN*（ドット区切り）
- *NN NNNNNNN*（スペース区切り）
- *NNNNNNN*

次の正規表現は、サポートされる形式にマッチします。

```
\d{2}[-.\s]\d{7}
```

次のルールは、id XML 要素または JSON プロパティでパターンにマッチする値をテキスト「NN-NNNNNNN」に置換することを指定します。パターン内でエスケープ処理された文字に注意してください。

XML	JSON

<rule xml:lang="zxx"
xmlns="http://marklogic.com/xdmp/
redaction">
<path>//id</path>
<method>
<function>redact-regex</function>
</method>
<options>
<pattern>
\d{{2}}[-.\s]\d{{7}}
</pattern>
<replacement>NN-NNNNNNN
</replacement>
</options>
</rule>
	{"rule": {
	"path": "//id",
	"method": {
	"function":
	"redact-regex",
	},
	"options": {
	"pattern":
	"\\d{2}[-.\\s]\\d{7}
	",
	"replacement":
	"NN-NNNNNNN"
	}
	} }

以下の表は、ルールにマッチするドキュメントにルールを適用した結果を示しています。

形式	元のドキュメント	リダクションの結果
XML	<person> <id>**12-3456789**</id> </person>	<person> <id>NN-NNNNNNN</id> </person>
JSON	{"id": "12-3456789"}	{"id": "NN-NNNNNNN" } }

次のルールは、パターンで後方参照を使用して、id の先頭の 2 桁が変更されないようにします。前の例のパターンが変更されて、数字の最初のブロックのサブ式が丸括弧で囲まれています（「(\d{2})」）。この丸括弧は、置換文字列で「$1」として参照される変数に、そのテキストブロックを「キャプチャ」します。

XML	JSON

<rule xml:lang="zxx"
xmlns="http://marklogic.com/xdmp/
redaction">
<path>//id</path>
<method>
<function>redact-regex</function>
</method>
<options>
<pattern>
(\d{{2}})[-.\s]\d{{7}}
</pattern>
<replacement>**$1**-NNNNNNN
</replacement>
</options>
</rule>
	{"rule": {
	"path": "//id",
	"method": {
	"function":
	"redact-regex",
	},
	"options": {
	"pattern":
	"**(\\d{2})**[-.\\s]\\d
	{7}",
	"replacement":
	"**$1**-NNNNNNN"
	}
	} }

このルールを、前と同じドキュメントに適用すると、次のようなリダクション結果になります。

```
12-NNNNNNN
```

詳細については、`fn:replace` XQuery 関数または `fn.replace` サーバーサイド JavaScript 関数を参照してください。

完全な例については、「例：ビルトインリダクション関数の使用」（487 ページ）を参照してください。

24.10 例：ビルトインリダクション関数の使用

この例は、「サンプルの実行準備」（528 ページ）のサンプルドキュメントを使用して、すべてのビルトインリダクション関数を実行します。操作するルールセットとして、XML ルールセットまたは JSON ルートセットを選択できます。ルールは、どちらのルールセットでも同等です。

この例には、次の部分があります。

- ルール例のまとめ
- ルールのインストール。XML または JSON を選択します。両方ではなく、いずれか一方のセットをインストールします。
  - XML ルールのインストール
  - JSON ルールのインストール
- ルールの適用
- 結果の確認

24.10.1 ルール例のまとめ

この例の各ルールは、異なるビルトインリダクション関数を使用しています。ルールが重複しないように、各ルールは、異なるサンプルドキュメントの XML 要素値または JSON プロパティ値を操作します。同一のドキュメントコンポーネントで動作するルールコレクションは適用しないでください。

ルールは、次の形式の URI で挿入されます。ここで *name* は、ルールで選択されるノードの XML 要素ローカル名または JSON プロパティ名です（URI サフィックスは、インストールするルール形式によって異なります）。

```
/rules/redact-name.{xml|json}
```

例えば、`/rules/redact-alias.xml` は、サンプルドキュメントの `alias` XML 要素または JSON プロパティを対象にします。

各ルールは、2 つのコレクションに挿入されます。「all」コレクションと、ルールで使用されるビルトインを識別するコレクションです。例えば、`mask-random` ビルトインを使用する `/rules/redact-alias.json` は、コレクション「all」と「random」に挿入されます。これにより、ルールをまとめて適用することも、選択的に適用することもできます。

以下の表は、この例でインストールされるルールをまとめたものです。

ルール URI ベース名	使用されるビルトイン関数	パスセレクタ	コレクション
redact-name	mask-deterministic	//name	all, deterministic
redact-alias	mask-random	//alias	all, random
redact-address	conceal	//address	all, conceal
redact-balance	redact-number	//balance	all, balance
redact-ssn	redact-us-ssn	//ssn	all, ssn
redact-phone	redact-us-phone	//phone	all, phone
redact-email	redact-email	//email	all, email
redact-ip	redact-ipv4	//ip	all, ip
redact-id	redact-regex	//id	all, regex

24.10.2 XML ルールのインストール

XML ルールをインストールするには、次のスクリプトを Query Console にコピーし、Schemas データベースに対して実行します。Query Console を使用してルールをインストールする詳細な例については、「例：リダクションの基本的な使用方法」（432 ページ）を参照してください。

XQuery を使用した XML 形式のルール例をインストールするには、次の手順に従います。JSON ルールを使用する場合は、「JSON ルールのインストール」（492 ページ）を参照してください。Query Console を使用してルールをインストールする詳細な例については、「例：リダクションの基本的な使用方法」（432 ページ）を参照してください。

1. Query Console に以下のスクリプトをコピーします。
2. ［Query Type］を［XQuery］に設定します。
3. ［Database］を［Schemas］に設定します。

4. ［Run］をクリックします。ルールが Schemas データベースにインストールされます。

5. 必要に応じて、Query Console データベースエクスプローラーを使用してルールを確認します。

ルールをインストールするには次のスクリプトを使用します。これらのルールの機能の概要については、「ルール例のまとめ」（487 ページ）を参照してください。

```
xquery version "1.0-ml";
import module namespace rdt = "http://marklogic.com/
xdmp/redaction"
    at "/MarkLogic/redaction.xqy";

let $rules := (
<rules>
  <rule>
    <name>redact-name</name>
    <collection>deterministic</collection>
    <rdt:rule xml:lang="zxx"
        xmlns:rdt="http://marklogic.com/xdmp/redaction">
      <rdt:path>//name</rdt:path>
      <rdt:method>
        <rdt:function>mask-deterministic</rdt:function>
      </rdt:method>
      <rdt:options>
        <length>10</length>
      </rdt:options>
    </rdt:rule>
  </rule>
  <rule>
    <name>redact-alias</name>
    <collection>random</collection>
    <rdt:rule xml:lang="zxx"
        xmlns:rdt="http://marklogic.com/xdmp/redaction">
      <rdt:path>//alias</rdt:path>
      <rdt:method>
        <rdt:function>mask-random</rdt:function>
      </rdt:method>
      <rdt:options>
        <length>10</length>
      </rdt:options>
    </rdt:rule>
  </rule>
  <rule>
    <name>redact-address</name>
```

```
    <collection>conceal</collection>
    <rdt:rule xml:lang="zxx"
        xmlns:rdt="http://marklogic.com/xdmp/redaction">
      <rdt:path>//address</rdt:path>
      <rdt:method>
        <rdt:function>conceal</rdt:function>
      </rdt:method>
    </rdt:rule>
  </rule>
  <rule>
    <name>redact-balance</name>
    <collection>balance</collection>
    <rdt:rule xml:lang="zxx"
        xmlns:rdt="http://marklogic.com/xdmp/redaction">
      <rdt:path>//balance</rdt:path>
      <rdt:method>
        <rdt:function>redact-number</rdt:function>
      </rdt:method>
      <rdt:options>
        <min>0</min>
        <max>100000</max>
        <format>0.00</format>
        <type>decimal</type>
      </rdt:options>
    </rdt:rule>
  </rule>
  <rule>
    <name>redact-ssn</name>
    <collection>ssn</collection>
    <rdt:rule xml:lang="zxx"
        xmlns:rdt="http://marklogic.com/xdmp/redaction">
      <rdt:path>//ssn</rdt:path>
      <rdt:method>
        <rdt:function>redact-us-ssn</rdt:function>
      </rdt:method>
      <rdt:options>
        <level>partial</level>
      </rdt:options>
    </rdt:rule>
  </rule>
  <rule>
    <name>redact-phone</name>
    <collection>phone</collection>
    <rdt:rule xml:lang="zxx"
        xmlns:rdt="http://marklogic.com/xdmp/redaction">
      <rdt:path>//phone</rdt:path>
```

```
      <rdt:method>
        <rdt:function>redact-us-phone</rdt:function>
      </rdt:method>
      <rdt:options>
        <level>full</level>
      </rdt:options>
    </rdt:rule>
  </rule>
  <rule>
    <name>redact-email</name>
    <collection>email</collection>
    <rdt:rule xml:lang="zxx"
        xmlns:rdt="http://marklogic.com/xdmp/redaction">
      <rdt:path>//email</rdt:path>
      <rdt:method>
        <rdt:function>redact-email</rdt:function>
      </rdt:method>
      <rdt:options>
        <level>name</level>
      </rdt:options>
    </rdt:rule>
  </rule>
  <rule>
    <name>redact-ip</name>
    <collection>ip</collection>
    <rdt:rule xml:lang="zxx"
        xmlns:rdt="http://marklogic.com/xdmp/redaction">
      <rdt:path>//ip</rdt:path>
      <rdt:method>
        <rdt:function>redact-ipv4</rdt:function>
      </rdt:method>
      <rdt:options>
        <character>X</character>
      </rdt:options>
    </rdt:rule>
  </rule>
  <rule>
    <name>redact-id</name>
    <collection>regex</collection>
    <rdt:rule xml:lang="zxx"
        xmlns:rdt="http://marklogic.com/xdmp/redaction">
      <rdt:path>//id</rdt:path>
      <rdt:method>
        <rdt:function>redact-regex</rdt:function>
      </rdt:method>
      <rdt:options>
```

```
        <pattern>\d{{2}}[-.\s]\d{{7}}</pattern>
        <replacement>NN-NNNNNNN</replacement>
      </rdt:options>
    </rdt:rule>
  </rule>
</rules>
)
return
for $r in $rules/rule return
  let $options :=
    <options xmlns="xdmp:document-insert">

<permissions>{xdmp:default-permissions()}</permissions>
      <collections>
        <collection>all</collection>
        <collection>{$r/collection}</collection>
      </collections>
    </options>
  return xdmp:document-insert(
    fn:concat("/rules/", $r/name, ".xml"),
    $r/rdt:rule, $options
  )
```

24.10.3 JSON ルールのインストール

サーバーサイド JavaScript を使用した JSON 形式のルール例をインストールするには、次の手順に従います。XML ルールを使用する場合は、「XML ルールのインストール」（488 ページ）を参照してください。Query Console を使用してルールをインストールする詳細な例については、「例：リダクションの基本的な使用方法」（432 ページ）を参照してください。

1. Query Console に以下のスクリプトをコピーします。
2. ［Query Type］を［JavaScript］に設定します。
3. ［Database］を［Schemas］に設定します。
4. ［Run］をクリックします。ルールが Schemas データベースにインストールされます。
5. 必要に応じて、Query Console データベースエクスプローラーを使用してルールを確認します。

ルールをインストールするには次のスクリプトを使用します。これらのルールの機能の概要については、「ルール例のまとめ」（487 ページ）を参照してください。

```
declareUpdate();
const rules = [
  { name: 'redact-name',
    content:
      {rule: {
       path: '//name',
       method: {function: 'mask-deterministic'},
       options: {length: 10}
      }},
    collection: 'deterministic'
  },
  { name: 'redact-alias',
    content:
      {rule: {
       path: '//alias',
       method: {function: 'mask-random'},
       options: {length: 10}
      }},
    collection: 'random'
  },
  { name: 'redact-address',
    content:
      {rule: {
       path: '//address',
       method: {function: 'conceal'},
      }},
    collection: 'conceal'
  },
  { name: 'redact-balance',
    content:
      {rule: {
       path: '//balance',
       method: {function: 'redact-number'},
       options: {min: 0, max: 100000, type: 'decimal',
format: '0.00'}
      }},
    collection: 'balance'
  },
  { name: 'redact-ssn',
    content:
      {rule: {
       path: '//ssn',
       method: {function: 'redact-us-ssn'},
```

```
        options: {level: 'partial'}
       }},
    collection: 'ssn'
  },
  { name: 'redact-phone',
    content:
      {rule: {
       path: '//phone',
       method: {function: 'redact-us-phone'},
       options: {level: 'full'}
      }},
    collection: 'phone'
  },
  { name: 'redact-email',
    content:
      {rule: {
       path: '//email',
       method: {function: 'redact-email'},
       options: {level: 'name'}
      }},
    collection: 'email'
  },
  { name: 'redact-ip',
    content:
      {rule: {
       path: '//ip',
       method: {function: 'redact-ipv4'},
       options: {character: 'X'}
      }},
    collection: 'ip'
  },
  { name: 'redact-id',
    content:
      {rule: {
       path: '//id',
       method: {function: 'redact-regex'},
       options: {
         pattern: '\\d{2}[-.\\s]\\d{7}',
         replacement: 'NN-NNNNNNN'
       }
      }},
    collection: 'regex'
  }
];
rules.forEach(function (rule, i, a) {
  xdmp.documentInsert(
```

```
    '/rules/' + rule.name + '.json',
    rule.content,
    { permissions: xdmp.defaultPermissions(),
      collections: ['all', rule.collection] }
    );
})
```

24.10.4 ルールの適用

すべてのルール例を適用するには、次の手順に従います。

サンプルドキュメントをまだインストールしていない場合は、「サンプルの実行準備」（528 ページ）からインストールします。この例では、ドキュメントデータベースにインストールされていることを前提としています。

次の方法のいずれかを選択して、ルールを適用します。

- XQuery を使用したリダクション
- JavaScript を使用したリダクション
- mlcp を使用したリダクション

24.10.4.1XQuery を使用したリダクション

XQuery および Query Console を使用してルール例を適用するには、次の手順に従います。すべてのルールがサンプルドキュメントに適用されます。

1. 次のスクリプトを Query Console にコピーします。

```
xquery version "1.0-ml";
import module namespace rdt = "http://marklogic.com/xdmp/
redaction"
    at "/MarkLogic/redaction.xqy";
rdt:redact(fn:collection("people"), "all")
```

2. [Query Type] を [XQuery] に設定します。

3. [Database] を [Documents] に設定します。

4. [Run] をクリックします。

リダクションされたドキュメントが Query Console に表示されます。想定される結果については、「結果の確認」（497 ページ）を参照してください。

24.10.4.2JavaScript を使用したリダクション

サーバーサイド JavaScript および Query Console を使用してルール例を適用するには、次の手順に従います。すべてのルールがサンプルドキュメントに適用されます。

1. 次のスクリプトを Query Console にコピーします。

```
const rdt = require('/MarkLogic/redaction.xqy');
rdt.redact(fn.collection('people'), 'all');
```

2. [Query Type] を [JavaScript] に設定します。

3. [Database] を [Documents] に設定します。

4. [Run] をクリックします。

リダクションされたドキュメントが Query Console に表示されます。想定される結果については、「結果の確認」（497 ページ）を参照してください。

24.10.4.3mlcp を使用したリダクション

リダクションされたドキュメントをドキュメントデータベースからエクスポートするには、次のようなコマンドラインを使用します。すべてのルールがサンプルドキュメントに適用されます。

使用中の環境にマッチするように、コマンドライン例を適宜変更してください。出力ディレクトリ（`./results`）が、まだ存在していないことを確認してください。

```
# Windows users, see Modifying the Example Commands for Windows
$ mlcp.sh export -host localhost -port 8000 -username user \
    -password password -mode local -output_file_path \
    ./results -collection_filter people \
    -redaction "all"
```

リダクションされたドキュメントが `./results` にエクスポートされます。想定される結果については、「結果の確認」（497 ページ）を参照してください。

リダクションで mlcp を使用する方法の詳細については、『*mlcp User Guide*』の Redacting Content During Export or Copy Operations を参照してください。

24.10.5 結果の確認

すべてのルール例を適用すると、サンプルドキュメントのほとんどの XML 要素および JSON プロパティがリダクションされます。次のルールが各要素またはプロパティに適用されることを思い出してください。

パスセレクタ	ビルトイン関数
//name	mask-deterministic
//alias	mask-random
//address	conceal
//ssn	redact-us-ssn
//phone	redact-us-phone
//email	redact-email
//ip	redact-ipv4
//id	redact-regex
//balance	redact-number

次の表は、サンプルドキュメント /redact-ex/person1.xml に対する効果を示しています。ルールが既存の値をマスクするのではなく、値を生成する場合は、表示されるリダクション後の値が、ここに示す値とは異なります。

元のドキュメント	リダクション後のドキュメント
<person> <name>Little Bopeep</name> <alias>Peepers</alias> <alias>Bo</alias> <address> <street>100 Nursery Lane </street> <city>Hometown</city> <country>Neverland</country> </address> <ssn>123-45-6789</ssn> <phone>123-456-7890</phone> <email>bopeep@mothergoose.com </email> <ip>111.222.33.4</ip> <id>12-3456789</id> <birthdate>2015-01-15 </birthdate> <balance>12.34</balance> </person>	<person> <name>63a63aa762</name> <alias>47c1fc8b29</alias> <alias>7a314dcf2d</alias> <ssn>###-##-6789</ssn> <phone>###-###-#### </phone> <email>NAME@mothergoose.com </email> <ip>XXX.XXX.XXX.XXX</ip> <id>NN-NNNNNNN</id> <birthdate>2015-01-15 </birthdate> <balance>0.67</balance> </person>

次の表は、サンプルドキュメント /redact-ex/person3.json に対する効果を示しています。

元のドキュメント	リダクション後のドキュメント

{ "name": "Georgie Porgie",
"alias": ["George", "G.P."],
"address": {
"street": "300 Nursery Lane",
"city": "Hometown",
"country": "Neverland"
},
"ssn": "345678901",
"phone": "(345)678-9012",
"email": "gp@mothergoose.com",
"ip": "33.44.5.66",
"id": "34-5678912",
"birthdate": "2012-07-12",
"balance": "12345.67"
}
	{ "name":"34fe55c66a",
	"alias":["27a76af34e",
	"8b87c3e8c6"],
	"ssn":"#####8901",
	"phone":"(###)###-####",
	"email":"NAME@mothergoose.com",
	"ip":"XXX.XXX.XXX.XXX",
	"id":"NN-NNNNNNN",
	"birthdate":"2012-07-12",
	"balance": "5.28"
	}

/redact-ex/person2.xml および /redact-ex/person3.json には、同様の変化が見られます。

> **注**： Query Console 内の結果は、person1、person2、person3 という順序になるとは限りません。

24.11 ユーザー定義のリダクション関数

ビルトインリダクション関数ではアプリケーションのニーズに対処できない場合は、XQuery またはサーバーサイド JavaScript でユーザー定義のリダクション関数を実装できます。ユーザー定義関数を導入して適用するには、次の手順に従います。

1. 関数を実装します。詳細については、「ユーザー定義のリダクション関数の実装」（499 ページ）を参照してください。

2. 記述した関数を、アプリケーションサーバーに関連付けられているモジュールデータベースにインストールします。詳細については、「ユーザー定義のリダクション関数のインストール」（500 ページ）を参照してください。

3. 関数を指定するルールを定義します。シンタックスについては、「リダクションルールの定義」（442 ページ）を参照してください。

4. ルールをインストールし、適用します。

このセクションには、次の内容が含まれます。

- ユーザー定義のリダクション関数の実装
- ユーザー定義のリダクション関数のインストール

完全な例については、「例：カスタムリダクションルールの使用」（503 ページ）を参照してください。

24.11.1 ユーザー定義のリダクション関数の実装

ユーザー定義関数は、XQuery またはサーバーサイド JavaScript で実装できます。実装は、次のいずれかのインターフェイスに準拠している必要があります。

言語	インターフェイス

XQuery	`declare function yourNS:yourFunc(`
	`  $node as node(),`
	`  $options as map:map`
	`) as node()?`

サーバーサイド JavaScript	`function yourFunc(node, options)`
	`// where:`
	`//  node is a Node`
	`//  options is an Object with paramName:value properties`
	`//  return 1 Node or nothing`

入力の `node` パラメータは、ルール内で関数を使用する XPath 式で選択されるノードです。`options` パラメータは、ユーザ定義データをルールから関数に渡すために使用できます。この関数は、ノード（リダクションされた、またはリダクションされていない）を返すか、または何も返しません。

関数は、XQuery または JavaScript ライブラリモジュールで定義する必要があります。モジュールは、リダクションの適用に使用するアプリケーションサーバーに関連付けられているモジュールデータベースにインストールする必要があります。詳細については、「ユーザー定義のリダクション関数のインストール」（500 ページ）を参照してください。

次の表は、準拠する独自のモジュールの定義に適したモジュールテンプレートを示しています。完全な例については、「例：JavaScript を使用したカスタムリダクション」（504 ページ）または「例：XQuery を使用したカスタムリダクション」（511 ページ）を参照してください。

言語	インターフェイス

XQuery	`xquery version "1.0-ml";`
	`module namespace yourNS = "/your/module/namespace";`
	
	`declare function yourNS:redact(`
	`  $node as node(),`
	`  $options as map:map`
	`) as node()?`
	`{`
	`  (: your implementation here :)`
	`};`

サーバーサイド JavaScript	`function yourFunc(node, options)`
	`{`
	`  // your implementation here`
	`}`
	
	`exports.redact = yourFunc`

24.11.2 ユーザー定義のリダクション関数のインストール

通常のドキュメント挿入方法（`xdmp:document-insert` XQuery 関数、`xdmp.documentInsert` サーバーサイド JavaScript 関数、あるいは Node.js、Java、または REST クライアント API のドキュメント挿入機能など）を使用して、アプリケーションサーバーに関連付けられているモジュールデータベースに実装をインストールします。

詳細については、次のいずれかのトピックを参照してください。

- XQuery を使用したリダクションモジュールのインストール
- JavaScript を使用したリダクションモジュールのインストール
- クライアント API を使用したリダクションモジュールのインストール

24.11.2.1XQuery を使用したリダクションモジュールのインストール

このセクションの手順では、Query Console および XQuery を使用してモジュールデータベースにモジュールをインストールする方法を示します。このタスクには、サーバーサイド JavaScript や、Java、Node.js、および REST クライアント API も使用できます。

ここで説明する手順には、次のような前提があります。環境とアプリケーション要件にマッチするように、手順やコード例を変更する必要があります。

- MarkLogic がローカルホストにインストールされている。
- アプリケーションサーバーに関連付けられているモジュールデータベースが Modules である。
- 実装が、ファイルシステム上のパス `/your/module/path/impl.xqy` のファイルに保存されている。
- デフォルトのドキュメントパーミッションがモジュールパーミッションに適している。

XQuery モジュールを Modules データベースにインストールするには、次のような手順を使用します。

1. ブラウザで Query Console に移動します。例えば、`http://localhost:8000/qconsole` に移動します。

2. 次のスクリプトを Query Console に貼り付けます。環境にマッチするように、モジュールの URI および `xdmp:document-get` 行のパスを変更します。

```
(: MODIFY THE FILE SYSTEM PATH AND URI TO MATCH YOUR ENV :)
xquery version "1.0-ml";
xdmp:document-insert(
  "/your/module/uri",
  xdmp:document-get("/your/module/path/impl.xqy"),
  <options xmlns="xdmp:document-insert">
    <permissions>{xdmp:default-permissions()}
    </permissions>
  </options>
)
```

3. ［Database］ドロップダウンで［Modules］を選択します。

4. ［Query Type］ドロップダウンで［XQuery］を選択します。

5. ［Run］ボタンをクリックします。モジュールが Modules データベースにインストールされます。

Query Console の［Explore］機能を使用して、Modules データベースを閲覧し、インストールを確認できます。

24.11.2.2JavaScript を使用したリダクションモジュールのインストール

このセクションの手順では、Query Console およびサーバーサイド JavaScript を使用してモジュールデータベースにモジュールをインストールする方法を示します。このタスクには、XQuery や、Java、Node.js、および REST クライアント API も使用できます。

ここで説明する手順には、次のような前提があります。環境とアプリケーション要件にマッチするように、手順やコード例を変更する必要があります。

- MarkLogic がローカルホストにインストールされている。
- アプリケーションサーバーに関連付けられているモジュールデータベースが Modules である。
- 実装が、ファイルシステム上のパス `/your/module/path/impl.sjs` のファイルに保存されている。
- デフォルトのドキュメントパーミッションがモジュールパーミッションに適している。

XQuery モジュールを Modules データベースにインストールするには、次のような手順を使用します。

1. ブラウザで Query Console に移動します。例えば、`http://localhost:8000/qconsole` に移動します。

2. 次のスクリプトを Query Console に貼り付けます。環境にマッチするように、モジュールの URI および `xdmp.documentGet` 行のパスを変更します。

```
// MODIFY THE FILE SYSTEM PATH and URI TO MATCH YOUR ENV
declareUpdate();
xdmp.documentInsert(
  '/your/module/uri',
  xdmp.documentGet('/your/module/path/impl.sjs'));
```

3. ［Database］ドロップダウンで［Modules］を選択します。

4. ［Query Type］ドロップダウンで［JavaScript］を選択します。

5. ［Run］ボタンをクリックします。モジュールが Modules データベースにインストールされます。

Query Console の［Explore］機能を使用して、Modules データベースを閲覧し、インストールを確認できます。

24.11.2.3クライアント API を使用したリダクションモジュールのインストール

Java クライアント API、Node.js クライアント API、および Node.js クライアント API には、モジュールデータベースにモジュールをインストールする機能があります。いずれかのクライアント API を使用してモジュールをインストールする方法の詳細については、次のいずれかのトピックを参照してください。

- Java：『*Java Application Developer's Guide*』の「Managing Dependent Libraries and Other Assets」
- Node.js: 『*Node.js Application Developer's Guide*』の「Managing Assets in the Modules Database」
- REST：『*REST Application Developer's Guide*』の「Managing Dependent Libraries and Other Assets」

24.12 例：カスタムリダクションルールの使用

この例では、カスタムリダクション関数をインストールおよび適用する方法について説明します。この例には 2 つのバージョンがあります。1 つは JSON/JavaScript 中心の例、もう 1 つは XML/XQuery 中心の例です。このように人為的に分割することで、例が複雑にならないようにしています。XML と JSON は、XQuery とサーバーサイド JavaScript のどちらとも自由に併用できます。

カスタムリダクションルールの使用方法を確認するには、次のいずれかの例を選択してください。

- 例：JavaScript を使用したカスタムリダクション
- 例：XQuery を使用したカスタムリダクション

24.12.1 例：JavaScript を使用したカスタムリダクション

この例では、個人のプロフィールデータ（名前、住所、誕生日など）が含まれる JSON ドキュメントを操作します。カスタムのサーバーサイド JavaScript リダクション関数を使用して、個人が 18 歳未満である場合に名前をリダクションします。ルール固有のオプション値で置換テキストが制御されます。

シンプルにするために、この例では JavaScript と JSON のみを使用します。カスタム関数を記述して、XML と JSON の両方を処理することもできます。XQuery/XML による同様の例については、「例：JavaScript を使用したカスタムリダクション」（504 ページ）を参照してください。

例を実行する前に、「サンプルの実行準備」（528 ページ）のサンプルドキュメントをインストールします。

この例には、次の部分があります。

- 入力データ
- リダクション関数のインストール
- リダクションルールのインストール
- JavaScript を使用したルール適用
- mlcp を使用したルール適用

24.12.1.1 入力データ

入力ドキュメントの構造は次のとおりです。`birthdate` プロパティは、`name` プロパティをリダクションするかどうかを決定するために使用します。

```
{ "name": "any text",
  ...
  "birthdate": "YYYY-MM-DD"
}
```

サンプルドキュメントをインストールするには、「サンプルの実行準備」（528 ページ）を参照してください。

24.12.1.2 リダクション関数のインストール

URI `/redaction/redact-xml-name.sjs` を使用してカスタム関数を Modules データベースにインストールするには、次の手順を使用します。この手順では、サーバーサイド JavaScript および Query Console を使用しますが、任意のドキュメント挿入インターフェイスを使用できます。手順の後に関数の説明が続きます。

1. 次のカスタムリダクション関数実装を、「redact-json-name.sjs」という名前のファイルに保存します。MarkLogic が認識できる場所を選択してください。

```
function redactName(node, options) {
  const parent = fn.head(node.xpath('./parent::node()'));

  // only redact if containing obj has the expected 'shape'
  if (parent.nodeKind == 'object' &&
      parent.hasOwnProperty('birthdate')) {
    const birthday =
      xdmp.parseDateTime('[Y0001]-[M01]-[D01]',
      parent.birthdate);
    const age = Math.floor(fn.daysFromDuration(
                fn.currentDateTime().subtract
                (birthday)) / 365);
    if (age < 18) {
      // underage, so redact
      const builder = new NodeBuilder();
      builder.addText(options.newName);
      return builder.toNode();
    }
  }
  // not expected input, or not underage - do nothing
  return node;
};

exports.redact = redactName;
```

2. ブラウザで Query Console に移動します。例えば、`http://localhost:8000/qconsole` に移動します。

3. 次のスクリプトを Query Console に貼り付けます。手順 1 のファイルの場所にマッチするように、`xdmp.documentGet` 行のパスを変更します。

```
// MODIFY THE FILE SYSTEM PATH TO MATCH YOUR ENV
declareUpdate();
xdmp.documentLoad(
  '/your/path/redact-json-name.sjs',
  {uri: '/redaction/redact-json-name.sjs'});
```

4. [Database] ドロップダウンで [Modules] を選択します。

5. ［Query Type］ドロップダウンで［JavaScript］を選択します。

6. ［Run］ボタンをクリックします。このモジュールは、URI が「/redaction/redact-xml-name.sjs」の Modules データベースにインストールされます。

Query Console を使用して、Modules データベースを調べ、インストールを確認できます。

このカスタム関数は、リダクションの候補であるノードに対応する JSON ノードを受け取ることを想定しています。このノードは、`birthdate` プロパティも持っているオブジェクトの子である必要があります。このコードスニペットは、次のチェックを実装しています。

```
const parent = fn.head(node.xpath('./parent::node()'));

// only redact if containing obj has the expected 'shape'
if (parent.nodeKind == 'object' &&
  parent.hasOwnProperty('birthdate')) {

  ...
```

理論的には、入力が親オブジェクトであると想定し、`/name/parent::node()` のような XPath 式をリダクションルールで使用する関数を記述できます。ただし、このようなルールパスは、ルールが XML ドキュメントに適用される場合は無効です。そのため、ルール内ではなく、リダクション関数内で親ノードまでトラバースしています。詳細については、「リダクションルールにおける XPath 式の制限」（447 ページ）を参照してください。

このリダクション関数は、`birthdate` 要素を使用して年齢を計算しています。年齢が 18 歳未満である場合、`name` 要素内のテキストがリダクションされます。options オブジェクトの「newName」プロパティの値が置換テキストとして使用されます。

```
const birthday =
  xdmp.parseDateTime('[Y0001]-[M01]-[D01]',
  parent.birthdate);
const age = Math.floor(fn.daysFromDuration(
            fn.currentDateTime().subtract
            (birthday)) / 365);
if (age < 18) {
  // underage, so redact
  const builder = new NodeBuilder();
  builder.addText(options.newName);
  return builder.toNode();
}
```

リダクション関数は、単純値ではなく、ノードを返す必要があります。この場合、元の入力値を置換する JSON テキストノードを返す必要があります。テキストノードはネイティブ JavaScript オブジェクトからは構築できません。そのため、関数は、`NodeBuilder` を使用して、返すノードを構築します。

これらの要件は、ルートオブジェクトノードの操作に特有ではありません。入力としてノードを使用し、ネイティブ JavaScript 型として変更する場合は常に、`toObject` を使用する必要があります。同様に、ネイティブ JavaScript 値ではなく、常にノードを返す必要があります。

24.12.1.3リダクションルールのインストール

コンテンツデータベースに関連付けられているスキーマデータベースにルールをインストールするには、次の手順を使用します。手順の後にルールの説明が続きます。

この手順では、localhost:8000 にプレインストールされたアプリケーションサーバー、および Schemas データベースを使用するように設定されたドキュメントデータベースを使用することを前提にしています。この例では、サーバーサイド JavaScript および Query Console を使用してルールをインストールしますが、任意のドキュメント挿入インターフェイスを使用できます。

1. ブラウザで Query Console に移動します。例えば、`http://localhost:8000/qconsole` に移動します。

2. 次のスクリプトを、Query Console の新しいクエリタブに貼り付けます。

```
declareUpdate();
xdmp.documentInsert('/rules/redact-name.json',
  { rule: {
    path: '/name',
    method: {
      function: 'redact',
      module: '/redaction/redact-json-name.sjs'
    },
    options: { newName: 'Jane Doe' }
  }},
  { permissions: xdmp.defaultPermissions(),
    collections: ['custom-rules'] }
);
```

3. ［Database］ドロップダウンで［Schemas］を選択します。

4. ［Query Type］ドロップダウンで［JavaScript］を選択します。

5. ［Run］ボタンをクリックします。ルールドキュメントが URI 「/rules/redact-name.json」でインストールされ、「custom-rules」コレクションに追加されます。

ルール内のパス式は、リダクション対象として `name` プロパティを選択します。カスタム関数は `name` の `birthdate` 兄弟プロパティを使用してリダクションを制御するため、ある意味では、親オブジェクトにルールを適用する方が、より自然です。しかし、親オブジェクトは匿名であるため、XPath 式では名前を使用して処理できません。

`/name/parent::node()` のような XPath 式は匿名の親オブジェクトを選択しますが、ルールが XML ドキュメントに適用される場合はエラーが発生します。ドキュメントセットでは、XML と JSON が混在しているため、`name` プロパティをリダクション対象として使用するルールとカスタム関数を記述することにします。

ルール内でカスタム関数は、エクスポートされた関数名、およびモジュールデータベースにインストールされた実装の URI によって識別されます。

```
method: {
  function: 'redact',
  module: '/redaction/redact-json-name.sjs'
}
```

`options` プロパティには、子 `newName` が 1 つ含まれています。この値は、リダクションされる `name` 要素の置換値として使用されます。

```
options: { newName: 'Jane Doe' }
```

カスタム関数を使用するルールを定義およびインストールする XQuery/XML の同様の例については、「例：XQuery を使用したカスタムリダクション」（511 ページ）を参照してください。

24.12.1.4 JavaScript を使用したルール適用

Query Console および `rdt.redact` を使用して、カスタムリダクション関数例を適用するには、次の手順に従います。カスタムリダクションモジュール、ルール、およびサンプルドキュメントがすでにインストールされている必要があります。

1. ブラウザで Query Console に移動します。例えば、`http://localhost:8000/qconsole` に移動します。

2. 次のスクリプトを、Query Console の新しいクエリタブに貼り付けます。

```
const jsearch = require('/MarkLogic/jsearch');
const rdt = require('/MarkLogic/redaction');

jsearch.collections('people').documents()
    .map(function (match) {
        match.document = fn.head(
          rdt.redact(fn.root(match.document),
'custom-rules')
        .root;
        return match;
    }).result();
```

3. [Databases] ドロップダウンで [Documents] を選択します。

4. [Query Type] ドロップダウンで [JavaScript] を選択します。

5. [Run] ボタンをクリックします。「custom-rules」コレクション内のルールが「people」コレクション内のドキュメントに適用されます。

「サンプルの実行準備」（528 ページ）のサンプルドキュメントを使用する場合、スクリプトを実行すると、検索結果のマッチに、次のような効果があります。

- /redact-ex/person1.xml：ルールパスにマッチしないため、リダクションされない
- /redact-ex/person2.xml：ルールパスにマッチしないため、リダクションされない
- /redact-ex/person3.json：名前が「Jane Doe」に変更される
- /redact-ex/person4.json：年齢制限より下ではないため、リダクションされない

（XQuery/XML と JavaScript/JSON の両方のカスタムリダクション例をインストールした場合は、`/people/person1.xml` もリダクションされて「John Doe」と表示されます。）

rdt.redact に渡されるノードは、`fn.root` を `match.document` に適用して取得されます。

```
rdt.redact(fn.root(match.document), 'custom-rules')
```

`rdt.redact` 関数は、ドキュメントノードを入力として想定していますが、`match.document` は、ドキュメントノード（JSON オブジェクトノードや XML 要素ノードなど）内のルートノードです。`DocumentsSearch.map` のコンテキストでは、`match.document` 内のノードはインデータベースノードであり、インメモリ構成体ではありません。そのため、前述のように、`fn.root` を使用して上位のドキュメントノードにアクセスできます。

同様の手法を使用して、反対に、リダクション結果を検索結果に保存します。

```
match.document = fn.head(rdt.redact(...)).root;
```

これは、`rdt.redact` 関数がインメモリドキュメントノードの `Sequence` を返すため、必要です。リダクション後のコンテンツを、想定された形式で保存するには、`Sequence` 内の最初のノードに `fn.head` でアクセスし、「.root」プロパティを使用して「参照を解除」します。これで、ドキュメントノード内のルートノードに `match.document` が再び含まれます。

24.12.1.5 mlcp を使用したルール適用

以下のようなコマンドを実行して、カスタムリダクションルール例を適用できます。このコマンドは、リダクションされたドキュメントを `./mlcp-output` にエクスポートします。このディレクトリが、まだ存在していないことを確認してください。

使用中の環境にマッチするように、コマンドラインを適宜修正してください。

```
# Windows users, see Modifying the Example Commands for Windows
$ mlcp.sh export -host localhost -port 8000 -username user \
    -password password -mode local -output_file_path \
    ./mlcp-output -collection_filter people \
    -redaction "custom-rules"
```

詳細については、『*mlcp User Guide*』の Redacting Content During Export or Copy Operations を参照してください。

「サンプルの実行準備」（528 ページ）のサンプルドキュメントを使用する場合は、スクリプトを実行すると、4 つのファイルが `./mlcp-output` ディレクトリに作成されます。

これらのファイルは、入力ドキュメントに関する次の効果を反映します。

- /redact-ex/person1.xml：ルールパスにマッチしないため、リダクションされない
- /redact-ex/person2.xml：ルールパスにマッチしないため、リダクションされない
- /redact-ex/person3.json：名前が「Jane Doe」に変更される
- /redact-ex/person4.json：年齢制限より下ではないため、リダクションされない

（XQuery/XML と JavaScript/JSON の両方のカスタムリダクション例をインストールした場合は、`/people/person1.xml` もリダクションされて「John Doe」と表示されます。）

24.12.2 例：XQuery を使用したカスタムリダクション

この例では、個人のプロフィールデータ（名前、住所、誕生日など）が含まれる XML ドキュメントを操作します。カスタムの XQuery リダクション関数を使用して、個人が 18 歳未満である場合に名前をリダクションします。ルール固有のオプション値で置換テキストが制御されます。

この例では XQuery と XML のみを使用します。カスタム関数を記述して、XML と JSON の両方を処理できますが、XML には XQuery を使用し、JSON にはサーバーサイド JavaScript を使用する方が便利です。JavaScript/JSON による同等の例については、「例：JavaScript を使用したカスタムリダクション」（504 ページ）を参照してください。

この例を実行する前に、「サンプルの実行準備」（528 ページ）のサンプルドキュメントをインストールする必要があります。

この例には、次の部分があります。

- 入力データ
- リダクション関数のインストール
- リダクションルールのインストール
- XQuery を使用したルール適用
- mlcp を使用したルール適用

24.12.2.1 入力データ

入力ドキュメントの構造は次のとおりです。`birthdate` 要素は、`name` 要素をリダクションするかどうかを決定するために使用します。

```
<person>
  <name>any text</name>
  ...
  <birthdate>YYYY-MM-DD</birthdate>
</person>
```

サンプルドキュメントをインストールするには、「サンプルの実行準備」（528 ページ）を参照してください。

24.12.2.2 リダクション関数のインストール

URI `/redaction/redact-xml-name.xqy` を使用してカスタム関数を Modules データベースにインストールするには、次の手順を使用します。この手順では、XQuery および Query Console を使用しますが、任意のドキュメント挿入インターフェイスを使用できます。

1. 次のカスタムリダクション関数実装を、「redact-xml-name.xqy」という名前のファイルに保存します。MarkLogic が認識できる場所を選択してください。

```
xquery version "1.0-ml";
module namespace my =
"http://marklogic.com/example/redaction";


declare function my:redact(
  $node as node(),
  $options as map:map
) as node()?
{
  if (xdmp:node-kind($node) = "element" and
      fn:local-name-from-QName(fn:node-name($node)) =
     "person")
  then
    let $birthdate :=
      xdmp:parse-dateTime('[Y0001]-[M01]-[D01]',
      $node//birthdate)
    let $age := math:floor(fn:days-from-duration(
        fn:current-dateTime() - $birthdate)) div 365
    return
      if ($age < 18)
      then element { fn:node-name($node) } {
        $node/@*,
        for $n in ($node/node()) return
          if (fn:local-name-from-QName(fn:node-name($n)) =
          "name")
          then element {fn:node-name($n)} {
            $n/@*, text {map:get($options, "new-name")}
          }
          else $n
      }
      else $node
  else $node
};
```

2. ブラウザで Query Console に移動します。例えば、`http://localhost:8000/qconsole` に移動します。

3. 次のスクリプトを Query Console に貼り付けます。手順 1 のファイルの場所にマッチするように、`xdmp:document-get` 行のパスを変更します。

```
(: MODIFY THE FILE SYSTEM PATH TO MATCH YOUR ENV :)
xquery version "1.0-ml";
xdmp:document-load(
  "/your/path/redact-xml-name.xqy",
  <options xmlns="xdmp:document-load">
    <uri>/redaction/redact-xml-name.xqy</uri>
  </options>
)
```

4. [Database] ドロップダウンで [Modules] を選択します。

5. [Query Type] ドロップダウンで [XQuery] を選択します。

6. [Run] ボタンをクリックします。このモジュールは、URI が「/redaction/redact-xml-name.xqy」の Modules データベースにインストールされます。

Query Console を使用して、Modules データベースを調べ、インストールを確認できます。

このカスタム関数は、入力としての `<person/>` ノード、および置換名の値を指定する「new-name」キーが含まれるオプションを受け取ることを想定しています。

この関数は、`birthdate` 要素を使用して年齢を計算しています。年齢が 18 歳未満である場合、`name` 要素内のテキストがリダクションされます。

入力の「形式」が想定と異なる場合、または年齢が 18 歳以上である場合は、変更されていない入力ノードが返されます。

JavaScript を利用する同様のソリューションについては、「例：JavaScript を使用したカスタムリダクション」（504 ページ）を参照してください。

24.12.2.3 リダクションルールのインストール

コンテンツデータベースに関連付けられているスキーマデータベースにルールをインストールするには、次の手順を使用します。手順の後にルールの説明が続きます。

この手順では、localhost:8000 にプレインストールされたアプリケーションサーバー、および Schemas データベースを使用するように設定されたドキュメントデータベースを使用することを前提にしています。この例では、XQuery および Query Console を使用してルールをインストールしますが、任意のドキュメント挿入インターフェイスを使用できます。

1. ブラウザで Query Console に移動します。例えば、`http://localhost:8000/qconsole` に移動します。

2. 次のスクリプトを、Query Console の新しいクエリタブに貼り付けます。

```
xquery version "1.0-ml";
xdmp:document-insert("/rules/redact-name.xml",
  <rdt:rule xml:lang="zxx"
      xmlns:rdt="http://marklogic.com/xdmp/redaction">
    <rdt:path>/person</rdt:path>
    <rdt:method>
      <rdt:function>redact</rdt:function>

<rdt:module>/redaction/redact-xml-name.xqy</rdt:module>
      <rdt:module-namespace>http://marklogic.com/example/
      redaction</rdt:module-namespace>
    </rdt:method>
    <rdt:options>
      <new-name>John Doe</new-name>
    </rdt:options>
  </rdt:rule>
, <options xmlns="xdmp:document-insert">
    <permissions>{xdmp:default-permissions()}</permissions>
    <collections>
      <collection>custom-rules</collection>
    </collections>
  </options>)
```

3. [Database] ドロップダウンで [Schemas] を選択します。

4. [Query Type] ドロップダウンで [XQuery] を選択します。

5. [Run] ボタンをクリックします。ルールドキュメントが URI 「/rules/redact-name.xml」でインストールされ、「custom-rules」コレクションに追加されます。

サンプルドキュメントのルートが <person/> であることを思い出してください。つまり、ルールは「/person」をパス値として使用してコンテンツ全体を選択します。これにより、リダクション関数は /person/birthdate を簡単に調べたり、/person/name を変更することができます。

ルール内でカスタム関数は、関数名、モジュール URI、およびモジュール名前空間によって識別されます。

```
<rdt:method>
  <rdt:function>redact</rdt:function>
  <rdt:module>/redaction/redact-xml-name.xqy</rdt:module>
    <rdt:module-namespace>
      http://marklogic.com/example/redaction
    </rdt:module-namespace>
</rdt:method>
```

options 要素には、要素 new-name が 1 つ含まれており、リダクションされる name 要素の置換値として使用されます。

```
<rdt:options>
  <new-name>John Doe</new-name>
</rdt:options>
```

カスタム関数を使用するルールを定義およびインストールする JavaScript/JSON の同様の例については、「例：JavaScript を使用したカスタムリダクション」（504 ページ）を参照してください。

24.12.2.4 XQuery を使用したルール適用

Query Console および rdt:redact を使用してカスタムリダクション関数例を適用するには、次の手順に従います。カスタムリダクションモジュール、ルール、およびサンプルドキュメントがすでにインストールされている必要があります。

1. ブラウザで Query Console に移動します。例えば、http://localhost:8000/qconsole に移動します。

2. 次のスクリプトを、Query Console の新しいクエリタブに貼り付けます。

```
xquery version "1.0-ml";
import module namespace rdt =
"http://marklogic.com/xdmp/redaction"
    at "/MarkLogic/redaction.xqy";
rdt:redact(
  cts:search(fn:doc(), cts:collection-query("people")),
  "custom-rules")
```

3. ［Databases］ドロップダウンで［Documents］を選択します。

4. ［Query Type］ドロップダウンで［XQuery］を選択します。

5. ［Run］ボタンをクリックします。「custom-rules」コレクション内のルールが「people」コレクション内のドキュメントに適用されます。

「サンプルの実行準備」（528 ページ）のサンプルドキュメントを使用する場合は、スクリプトを実行すると、次の結果が返されます。:

- /redact-ex/person1.xml：リダクションにより John Doe に変更された名前
- /redact-ex/person2.xml：年齢が 18 歳より上であるため、リダクションされない
- /redact-ex/person3.json：ルールパスにマッチしないため、リダクションされない
- /redact-ex/person4.json：ルールパスにマッチしないため、リダクションされない

（XQuery/XML と JavaScript/JSON の両方のカスタムリダクション例をインストールした場合は、`/people/person3.json` もリダクションされて「Jane Doe」と表示されます。）

24.12.2.5 mlcp を使用したルール適用

次のようなコマンドを実行して、カスタムリダクションルール例を適用できます。このコマンドは、リダクションされたドキュメントを `./mlcp-output` にエクスポートします。このディレクトリが、まだ存在していないことを確認してください。

使用中の環境にマッチするように、コマンドラインを適宜修正してください。

```
# Windows users, see Modifying the Example Commands for Windows
$ mlcp.sh export -host localhost -port 8000 -username user \
    -password password -mode local -output_file_path \
    ./mlcp-output -collection_filter people \
    -redaction "custom-rules"
```

詳細については、『*mlcp User Guide*』の Redacting Content During Export or Copy Operations を参照してください。

「サンプルの実行準備」（528 ページ）のサンプルドキュメントを使用する場合は、スクリプトを実行すると、4 つのファイルが `./mlcp-output` ディレクトリに作成されます。これらのファイルは、入力ドキュメントに関する次の効果を反映します。

- /redact-ex/person1.xml：リダクションにより John Doe に変更された名前
- /redact-ex/person2.xml：年齢が 18 歳より上であるため、リダクションされない

- /redact-ex/person3.json：ルールパスにマッチしないため、リダクションされない
- /redact-ex/person4.json：ルールパスにマッチしないため、リダクションされない

（XQuery/XML と JavaScript/JSON の両方のカスタムリダクション例をインストールした場合は、`person3.json` もリダクションされて「Jane Doe」と表示されます。）

24.13 辞書ベースのマスキングの使用

コンテンツをマスクする事前定義済みリダクション関数の一部では、マスキング値をリダクション辞書から抽出できます。このセクションでは、マスキングソースとしての辞書の使用方法に関する次のトピックについて説明します。

- リダクション辞書の定義
- リダクション辞書のインストール
- リダクション辞書の使用

24.13.1 リダクション辞書の定義

リダクション辞書とは、以下に指定する形式の XML または JSON ドキュメントです。

形式	シンタックス

XML	`<dictionary xmlns="http://marklogic.com/xdmp/redaction">`
	`  <entry>`*value*`</entry>`
	`  ...`
	`</dictionary>`

JSON	`{ "dictionary": {`
	`  "entry":[`
	`  `*value*`,`
	`  ...`
	`  ]`
	`}}`

以下の要件が適用されます。これらの要件を満たさない場合は、辞書を使用するときに `RDT-INVALIDDICTIONARY` エラーが発生します。

- 辞書には必ず 1 つ以上のエントリが含まれている。
- エントリ内の値は空であることも、null であることもない。

- 値は必ずアトミックである。つまり、次のとおりです。
    - XML では、エントリ値は任意のテキスト（単語、フレーズ、日付、小数など）です。
    - JSON では、値は文字列、数値、またはブール値です。

次の例は、さまざまな型のエントリが 4 つ含まれる簡単な辞書です。完全な例については、「例：辞書ベースのマスキング」（519 ページ）を参照してください。

形式	シンタックス
XML	`<dictionary xmlns="http://marklogic.com/xdmp/redaction"> <entry>a phrase</entry> <entry>a_term</entry> <entry>1234</entry> <entry>true</entry> </dictionary>`
JSON	`{ "dictionary": { "entry":[ "a phrase", "a_term", 1234, true ] }}`

24.13.2 リダクション辞書のインストール

リダクション辞書は、使用する前に、リダクション対象のコンテンツが含まれるデータベースに関連付けられているスキーマデータベース内にインストールする必要があります。これは、リダクションルールをインストールするデータベースと同じであることが必要です。

「リダクションルールのインストール」（454 ページ）で説明しているものと同じ手法を使用してインストールします。

セキュリティ上、ドキュメントパーミッションを使用して、辞書の読み取りまたは変更が可能なユーザーを慎重に制御してください。詳細については、「セキュリティ上の考慮事項」（439 ページ）を参照してください。

24.13.3 リダクション辞書の使用

辞書ベースマスキングをサポートする事前定義済みリダクション関数は、辞書 URI を値として利用できる `dictionary` オプションを使用して辞書ベースマスキングを実行します。

例えば、`mask-deterministic` および `mask-random` ビルトインリダクション関数は `dictionary` オプションをサポートするため、次のようなルールを使用して辞書から値を取り出すことができます。

```
<rule xml:lang="zxx"
xmlns="http://marklogic.com/xdmp/redaction">
  <path>//country</path>
  <method>
  <function>mask-random</function>
  </method>
  <options>
    <dictionary>/rules/dict/countries.xml</dictionary>
  </options>
</rule>
```

詳細については、「ビルトインリダクション関数のリファレンス」（461 ページ）を参照してください。完全な例については、「例：辞書ベースのマスキング」（519 ページ）を参照してください。

24.14 例：辞書ベースのマスキング

このセクションでは、リダクション辞書をインストールしてビルトインリダクション関数で使用する方法を例示します。ルール例は、次のリダクションを実行します。

- `mask-deterministic` 関数および JSON 辞書を、サンプルデータの `country` XML 要素または JSON プロパティに適用します。
- `mask-random` 関数および XML 辞書を、サンプルデータの street XML 要素または JSON プロパティに適用します。

この例を実行する前に、「サンプルの実行準備」（528 ページ）のサンプルドキュメントをインストールする必要があります。

次の手順に従ってこの例を実行します。

- 辞書のインストール
- ルールのインストール
- ルールの適用

24.14.1 辞書のインストール

辞書例をインストールするには、次のいずれかの手順を使用します。この手順では 2 つの辞書をインストールします。XML で定義された国名辞書と、JSON で定義された番地辞書です。

- XQuery を使用した辞書のインストール
- JavaScript を使用した辞書のインストール

24.14.1.1 XQuery を使用した辞書のインストール

次の手順では 2 つの辞書例をインストールします。

1. Query Console で、新しいクエリに以下のスクリプトをコピーします。
2. [Query Type] を [XQuery] に設定します。
3. [Database] を [Schemas] に設定します。
4. [Run] をクリックします。辞書は、URI `/rules/dict/countries.xml` および `/rules/dict/streets.json` で Schemas データベースにインストールされます。
5. 必要に応じて、Query Console データベースエクスプローラーを使用して辞書を確認します。

前述の手順 1 で、次のスクリプトを使用します。

```
(: NOTE: RUN AGAINST YOUR SCHEMAS DB :)

(: Install example XML dictionary :)
xquery version "1.0-ml";
let $dictURI := '/rules/dict/countries.xml'
let $dict :=
  <dictionary xmlns="http://marklogic.com/xdmp/redaction">
    <entry>Brazil</entry>
    <entry>China</entry>
    <entry>France</entry>
    <entry>Germany</entry>
    <entry>United States</entry>
    <entry>United Kingdom</entry>
  </dictionary>
return
xdmp:document-insert($dictURI, $dict,
  <options xmlns="xdmp:document-insert">
```

```
    <permissions>{xdmp:default-permissions()}
    </permissions>
  </options>);

(: Install example JSON dictionary :)
xquery version "1.0-ml";
let $dictURI := '/rules/dict/steets.json'
let $dict := xdmp:unquote(
  '{ "dictionary": {
    "entry": [
      "10 Oak Ln",
      "2451 Elm St",
      "892 Veterans Blvd",
      "P.O.Box 1234",
      "250 Park Ln",
      "16 Highway 82, Suite 301"
    ]
  } }')
return
xdmp:document-insert(
  $dictURI, $dict,
  <options xmlns="xdmp:document-insert">
    <permissions>{xdmp:default-permissions()}
    </permissions>
  </options>);
```

24.14.1.2 JavaScript を使用した辞書のインストール

次の手順では 2 つの辞書例をインストールします。

1. Query Console で、新しいクエリに以下のスクリプトをコピーします。
2. [Query Type] を [JavaScript] に設定します。
3. [Database] を [Schemas] に設定します。
4. [Run] をクリックします。辞書は、URI `/rules/dict/countries.xml` および `/rules/dict/streets.json` で Schemas データベースにインストールされます。
5. 必要に応じて、Query Console データベースエクスプローラーを使用して辞書を確認します。

前述の手順 1 で、次のスクリプトを使用します。

```
// NOTE: RUN AGAINST YOUR SCHEMAS DB
declareUpdate();

// Install example XML dictionary
const countryDict = fn.head(xdmp.unquote(
  '<dictionary
xmlns="http://marklogic.com/xdmp/redaction">' +
    '<entry>Brazil</entry>' +
    '<entry>China</entry>' +
    '<entry>France</entry>' +
    '<entry>Germany</entry>' +
    '<entry>United States</entry>' +
    '<entry>United Kingdom</entry>' +
  '</dictionary>'));
xdmp.documentInsert(
  '/rules/dict/countries.xml', countryDict,
  { permissions: xdmp.defaultPermissions() }
);

// Install example JSON dictionary
const streetDict =
  { dictionary: {
    entry: [
      '10 Oak Ln',
      '2451 Elm St',
      '892 Veterans Blvd',
      'P.O.Box 1234',
      '250 Park Ln',
      '16 Highway 82, Suite 301'
    ]
  } };
xdmp.documentInsert(
  '/rules/dict/streets.json', streetDict,
  { permissions: xdmp.defaultPermissions() }
);
```

24.14.2 ルールのインストール

辞書を使用するルールをインストールするには、次のいずれかの手順を使用します。一方のルールは XML を使用して定義され、もう一方は JSON を使用して定義されています。

- XQuery を使用したルールのインストール
- JavaScript を使用したルールのインストール

24.14.2.1 XQuery を使用したルールのインストール

次の手順では、2 つのルールをインストールします。どちらも、「辞書のインストール」（520 ページ）でインストールした辞書の一方を使用します。

1. Query Console で、新しいクエリに以下のスクリプトをコピーします。
2. [Query Type] を [XQuery] に設定します。
3. [Database] を [Schemas] に設定します。
4. [Run] をクリックします。ルールは、URI `/rules/randomize-country.xml` および `/rules/redact-street.json` として、Schemas データベースにインストールされます。
5. 必要に応じて、Query Console データベースエクスプローラーを使用してルールを確認します。

前述の手順 1 で、次のスクリプトを使用します。

```
(: NOTE: RUN AGAINST YOUR SCHEMAS DB :)

(: Install rule using mask-random with a dictionary :)
xquery version "1.0-ml";
let $ruleURI := '/rules/randomize-country.xml'
let $rule :=
<rule xml:lang="zxx"
  xmlns="http://marklogic.com/xdmp/redaction">
  <path>//country</path>
  <method>
  <function>mask-random</function>
  </method>
  <options>
    <dictionary>/rules/dict/countries.xml</dictionary>
  </options>
</rule>
return xdmp:document-insert(
  $ruleURI, $rule,
  <options xmlns="xdmp:document-insert">
    <permissions>{xdmp:default-permissions()}
    </permissions>
    <collections>
      <collection>dict</collection>
      <collection>dict-random</collection>
    </collection>
  </options>);
```

```
(: Install rule using mask-deterministic with a dictionary
:)
xquery version "1.0-ml";
let $ruleURI := '/rules/redact-street.json'
let $rule := xdmp:unquote(
  '{"rule": {
     "path": "//street",
     "method": {"function": "mask-deterministic"},
     "options": {"dictionary": "/rules/dict/streets.json"}
  }}'
)
return xdmp:document-insert(
  $ruleURI, $rule,
  <options xmlns="xdmp:document-insert">
    <permissions>{xdmp:default-permissions()}
    </permissions>
    <collections>
      <collection>dict</collection>
      <collection>dict-deter</collection>
    </collections>
  </options>
);
```

24.14.2.2 JavaScript を使用したルールのインストール

次の手順では、2 つのルールをインストールします。どちらも、「辞書のインストール」（520 ページ）でインストールした辞書の一方を使用します。

1. Query Console で、新しいクエリに以下のスクリプトをコピーします。
2. ［Query Type］を［JavaScript］に設定します。
3. ［Database］を［Schemas］に設定します。
4. ［Run］をクリックします。ルールは、URI `/rules/randomize-country.xml` および `/rules/redact-street.json` として、Schemas データベースにインストールされます。
5. 必要に応じて、Query Console データベースエクスプローラーを使用してルールを確認します。

前述の手順 1 で、次のスクリプトを使用します。

```
// NOTE: RUN AGAINST YOUR SCHEMAS DB

declareUpdate();

// Install rule using mask-random with dictionary
xdmp.documentInsert(
  '/rules/randomize-country.xml',
  fn.head(xdmp.unquote(
    '<rule xml:lang="zxx"
xmlns="http://marklogic.com/xdmp/redaction">' +
     '<path>//country</path>' +
      '<method>' +
        '<function>mask-random</function>' +
      '</method>' +
      '<options>' +

'<dictionary>/rules/dict/countries.xml</dictionary>' +
      '</options>' +
    '</rule>')),
    { permissions: xdmp.defaultPermissions(),
      collections: ['dict','dict-random'] }
);

// Install rule using mask-deterministic with dictionary
xdmp.documentInsert(
  '/rules/redact-street.json',
  {rule: {
     path: '//street',
     method: {function: 'mask-deterministic'},
     options: {dictionary: '/rules/dict/streets.json'}
  }},
  { permissions: xdmp.defaultPermissions(),
    collections: ['dict','dict-deter'] }
);
```

24.14.3 ルールの適用

次のいずれかの方法を選択して、辞書ベースのマスキングを使用するルールを適用します。

- XQuery を使用したルール適用
- JavaScript を使用したルール適用
- mlcp を使用したルール適用

24.14.3.1 XQuery を使用したルール適用

XQuery および Query Console を使用してルール例を適用するには、次の手順に従います。すべてのルールがサンプルドキュメントに適用されます。

1. 次のスクリプトを Query Console にコピーします。

```
xquery version "1.0-ml";
import module namespace rdt = "http://marklogic.com/xdmp/
redaction"
    at "/MarkLogic/redaction.xqy";
let $results := rdt:redact(fn:collection("people"), "dict")
return (
  "*** REDACTED STREETS ***",
  $results//street/data(),
  "*** REDACTED COUNTRIES ****",
  $results//country/data()
)
```

2. [Query Type] を [XQuery] に設定します。

3. [Database] を [Documents] に設定します。

4. [Run] をクリックします。各ドキュメントでリダクションされた番地と国名が表示されます。

出力は次のようになりますが、値が異なっている可能性があります。

```
*** REDACTED STREETS ***
P.O.Box 1234
2451 Elm St
892 Veterans Blvd
250 Park Ln
*** REDACTED COUNTRIES ****
United States
Brazil
Germany
France
```

番地名の値は `mask-deterministic` を使用してリダクションされるため、スクリプトを再実行しても変わりません。国の値は `mask-random` を使用してリダクションされるため、実行するたびに値は異なります。

24.14.3.2 JavaScript を使用したルール適用

XQuery および Query Console を使用してルール例を適用するには、次の手順に従います。すべてのルールがサンプルドキュメントに適用されます。

1. 次のスクリプトを Query Console にコピーします。

```
const rdt = require('/MarkLogic/redaction.xqy');
const results = rdt.redact(fn.collection('people'),
'dict');

// Extract the redacted streed and country data for display
purposes
const displayAccumulator = ['*** STREETS ***'];
for (let doc of results) {
  displayAccumulator.push(doc.xpath('//street/data()'));
}
displayAccumulator.push('*** COUNTRIES ***');
for (let doc of results) {
  displayAccumulator.push(doc.xpath('//country/data()'));
}

// Dump the redacted street and country values
displayAccumulator
```

2. [Query Type] を [JavaScript] に設定します。

3. [Database] を [Documents] に設定します。

4. [Run] をクリックします。各ドキュメントでリダクションされた番地と国名が表示されます。

出力は次のようになりますが、値が異なっている可能性があります。

```
*** REDACTED STREETS ***
P.O.Box 1234
2451 Elm St
892 Veterans Blvd
250 Park Ln
*** REDACTED COUNTRIES ****
United States
Brazil
Germany
France
```

番地名の値は mask-deterministic を使用してリダクションされるため、スクリプトを再実行しても変わりません。国の値は mask-random を使用してリダクションされるため、実行するたびに値は異なります。

24.14.3.3 mlcp を使用したルール適用

リダクションされたドキュメントをドキュメントデータベースからエクスポートするには、次のようなコマンドラインを使用します。両方の辞書ベースのルールがサンプルドキュメントに適用されます。

使用中の環境にマッチするように、コマンドライン例を適宜変更してください。出力ディレクトリ（./dict-results）が、まだ存在していないことを確認してください。

```
# Windows users, see Modifying the Example Commands for Windows
$ mlcp.sh export -host localhost -port 8000 -username user \
    -password password -mode local -output_file_path \
    ./dict-results -collection_filter people \
    -redaction "dict"
```

リダクションされたドキュメントが ./dict-results にエクスポートされます。//street および //country のに値はそれぞれ、番地辞書と国辞書の値が反映されます。

番地値は mask-deterministic を使用してリダクションされるため、リダクションされた番地値は、ドキュメントをエクスポートするたびに同じになります。国値は mask- ランダムを使用してリダクションされるため、リダクションされた番地値は、ドキュメントをエクスポートするたびに同じになります。

リダクションで mlcp を使用する方法の詳細については、『*mlcp User Guide*』の Redacting Content During Export or Copy Operations を参照してください。

24.15 サンプルの実行準備

特に注記されていない限り、この章の例は、同じソースドキュメントセットに基づいています。ソースドキュメントセットは、構造が似ている 2 つの XML ドキュメントと 2 つの JSON ドキュメントから成ります。また、子 XML 要素または JSON オブジェクト、および JSON 配列などの、複雑な要素やプロパティの値が含まれています。

ドキュメントはコレクションに挿入されるため、リダクション対象として簡単に選択できます。「people」コレクションには、すべてのサンプルが含まれています。「xml-people」コレクションには、XML サンプルのみが含まれています。「json-people」コレクションには、JSON サンプルのみが含まれています。

このセクションの手順を完了すると、ドキュメントデータベースには次のドキュメントが含まれます。次のリストでは、コレクション名を URI の後の丸括弧内に示しています。

- `/redact-ex/person1.xml` (`people`, `xml-people`)
- `/redact-ex/person2.xml` (`people`, `xml-people`)
- `/redact-ex/person3.json` (`people`, `json-people`)
- `/redact-ex/person4.json` (`people`, `json-people`)

サンプルドキュメントをインストールするには、次の手順に従います。

1. ブラウザで Query Console に移動します。例えば、`http://localhost:8000/qconsole` に移動します。

2. 次のスクリプトを、Query Console の新しいクエリタブに貼り付けます。

```
xquery version "1.0-ml";
xdmp:document-insert("/redact-ex/person1.xml",
  <person>
    <name>Little Bopeep</name>
    <alias>Peepers</alias>
    <alias>Bo</alias>
    <address>
      <street>100 Nursery Lane</street>
      <city>Hometown</city>
      <country>Neverland</country>
    </address>
    <ssn>123-45-6789</ssn>
    <phone>123-456-7890</phone>
    <email>bopeep@mothergoose.com</email>
    <ip>111.222.33.4</ip>
    <id>12-3456789</id>
    <birthdate>2015-01-15</birthdate>
  </person>,
  <options xmlns="xdmp:document-insert">
    <permissions>{xdmp:default-permissions()}
    </permissions>
    <collections>
      <collection>people</collection>
      <collection>xml-people</collection>
    </collections>
  </options>
);

xquery version "1.0-ml";
```

```
xdmp:document-insert("/redact-ex/person2.xml",
  <person>
    <name>Humpty Dumpty</name>
    <alias>Dumpy</alias>
    <address>
      <street>200 Nursery Lane</street>
      <city>Hometown</city>
      <country>Neverland</country>
    </address>
    <ssn>234.56.7890</ssn>
    <phone>234.567.8901</phone>
    <email>hdumpty@mothergoose.com</email>
    <ip>222.3.44.5</ip>
    <id>23-4567891</id>
    <birthdate>1965/06/12</birthdate>
  </person>,
  <options xmlns="xdmp:document-insert">
    <permissions>{xdmp:default-permissions()}
    </permissions>
    <collections>
      <collection>people</collection>
      <collection>xml-people</collection>
    </collections>
  </options>
);

xquery version "1.0-ml";
xdmp:document-insert("/redact-ex/person3.json",
xdmp:unquote('
  { "name": "Georgie Porgie",
    "alias": ["George", "G.P."],
    "address": {
      "street": "300 Nursery Lane",
      "city": "Hometown",
      "country": "Neverland"
    },
    "ssn": "345678901",
    "phone": "(345)678-9012",
    "email": "gp@mothergoose.com",
    "ip": "33.44.5.66",
    "id": "34-5678912",
    "birthdate": "2012-07-12"
  }'),
  <options xmlns="xdmp:document-insert">
    <permissions>{xdmp:default-permissions()}
    </permissions>
```

```
    <collections>
      <collection>people</collection>
      <collection>json-people</collection>
    </collections>
  </options>
);

xquery version "1.0-ml";
xdmp:document-insert("/redact-ex/person4.json",
xdmp:unquote('
  { "name": "Jack Sprat",
    "alias": ["Jacko","Beanpole"],
    "address": {
      "street": "400 Nursery Lane",
      "city": "Hometown",
      "country": "Neverland"
    },
    "ssn": "456-78-9012",
    "phone": "4567890123",
    "email": "jack.sprat@mothergoose.com",
    "ip": "4.55.6.77",
    "id": "45-6789123",
    "birthdate": "1995-10-04"
  }'),
  <options xmlns="xdmp:document-insert">
    <permissions>{xdmp:default-permissions()}
    </permissions>
    <collections>
      <collection>people</collection>
      <collection>json-people</collection>
    </collections>
  </options>
);
```

3. ［Database］ドロップダウンで［Documents］を選択します。

4. ［Query Type］ドロップダウンで［XQuery］を選択します。

5. ［Run］ボタンをクリックします。サンプルドキュメントがドキュメントデータベースにインストールされます。

6. 必要に応じて、［Database］ドロップダウンの隣にある［Explore］をクリックしてデータベースを確認し、サンプルドキュメントが挿入されていることを確認します。