MarkLogic サーバー

JavaScript Reference Guide

MarkLogic 9
2017 年 5 月

最終更新：9.0-1、2017 年 5 月

目次

JavaScript Reference Guide

1.0 MarkLogic のサーバーサイド JavaScript

MarkLogic 9 には、第一級のサーバーサイドプログラミング言語である JavaScript が統合されています。アプリケーションサーバーから JavaScript プログラムを呼び出すと、そのプログラムは MarkLogic のビルトイン関数に対してサーバーサイドのアクセス権を持つことになります。この章では、MarkLogic での JavaScript の実装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。

- Google V8 JavaScript エンジン
- JavaScript 開発者の習熟度
- サーバーサイドの MarkLogic によるデータサービス
- サーバーサイド JavaScript 内の日付
- JavaScript での数値データ型マッピング
- Query Console での JavaScript
- サーバーサイド JavaScript でのプログラミング
- スクリプトでの xdmp.invoke または xdmp.invokeFunction の使用
- 例外処理

1.1 Google V8 JavaScript エンジン

MarkLogic サーバーは、JavaScript の高パフォーマンスなオープンソース C++ 実装である Google V8 JavaScript エンジン（https://code.google.com/p/v8/）を統合しています。

MarkLogic には、バージョン 5.3 の Google V8 JavaScript エンジンが組み込まれています。

このバージョンの V8 は、より新しい EcmaScript 2015（以前は EcmaScript 6 として知られていました）のいくつかの機能を提供しています。次に、EcmaScript 15 の機能の一部を示します。

- アロー関数
- スプレッド演算子および残余引数
- マップおよびセット
- クラス
- 定数およびブロックスコープ変数
- テンプレート文字列
- シンボル

EcmaScript 2015 ジェネレータは `function*` シンタックスを使用します。EcmaScript 6 ジェネレータの説明については、https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/function* や http://wiki.ecmascript.org/doku.php?id=harmony:generators など、ジェネレータの実装に関するドキュメントを参照してください。ジェネレータについては、MarkLogic は、`Generator.prototype.next()` メソッド（`for ... of` ループで使用）のみをサポートしており、`Generator.prototype.return()` および `Generator.prototype.throw()` メソッドはサポートしていません。

次に、MarkLogic で実行する単純な JavaScript ジェネレータの例を示します。

```
function* gen(limit){
  for (let i = 0; i < limit; i++)
    yield xdmp.eval('xs.dateTime(new Date())');}
const result=[];
for (const i of gen(10)){
  result.push(i);}
result;
/* returns ten different dateTime values (because they are
each run
   in a separate eval) */
```

1.2 JavaScript 開発者の習熟度

JavaScript はプログラミング言語として広く普及しており、多くの開発者によく知られています。ここ数年間で、JavaScript の使用範囲は、ブラウザから Node.js などのプログラミング環境へと拡大しています。MarkLogic のサーバーサイド JavaScript により、1 つ深いデータベースサーバーレベルでも、この使い慣れた言語を使用できます。これにより、データベースレベルのプログラミングにおいて使い慣れた JavaScript を使用できます。

1.3 サーバーサイドの MarkLogic によるデータサービス

JavaScript を MarkLogic 内で実行すると、サーバーレベルでデータを直接処理でき、ミドルティアにデータを転送する必要がありません。これまでも MarkLogic では、XQuery でこれを実現していました。ほとんどの組織には、多数の経験豊富な JavaScript のプログラマーがいるはずであり、MarkLogic 8 では JavaScript を使用して同じ処理ができます。

1.4 サーバーサイド JavaScript 内の日付

MarkLogic には、W3C 標準の XML の日付と期間を使用して日付の値を返す多数の XQuery 関数があります。このような関数は、すべてサーバーサイド JavaScript でも使用可能であり、値は XML 型で返されます。

このような任意の関数からの戻り値に対して `toObject()` を呼び出すことで、日付値を JavaScript の UTC 日付に変換できます。この方法により、必要に応じて強力な XML の日付と期間の関数を使用することができ、さらに希望する任意の JavaScript による日付処理（またはすでに記述済みの JavaScript コード）と組み合わせることができます。JavaScript の日付関数の詳細については、JavaScript リファレンス（Mozilla など）を参照してください。MarkLogic のサーバーサイド JavaScript の日付関数の詳細については、http://docs.marklogic.com/js/fn/dates を参照してください。

次の例について考えてみましょう。

```
const results = new Array();
const cdt = fn.currentDateTime();
results.push(cdt);
const utc = cdt.toObject();
results.push(utc);
results;

=>
["2015-01-05T15:36:17.804712-08:00",
"2015-01-05T23:36:17.804"]
```

上の例では、`cdt` 変数（`results` 配列の最初の項目）からの出力が XML dateTime 値に固有のタイムゾーン情報を保持しています。`utc` 変数（`results` 配列の 2 番目の項目）からの出力は、現在は UTC 値であるためタイムゾーンシフトを含んでいません。

同様に、オブジェクトに変換される MarkLogic ベースの日付に対して任意の UTC メソッド使用できます。例えば、次のコードは UTC の月を返します。

```
fn.currentDateTime().toObject()
  .getMonth();  // note that the month is 0-based, so
January is 0
```

次のコードは、1970 年 1 月 1 日からの時間数をミリ秒単位で返します。

```
const utc = fn.currentDateTime().toObject();
Date.parse(utc);
// => 1420502304000 (will be different for different times)
```

必要に応じて JavaScript の日付関数を使用したり、XML/XQuery の日付関数を使用したりできるので、サーバーサイド JavaScript では日付を柔軟に使用できます。

1.5 JavaScript での数値データ型マッピング

サーバーサイド JavaScript では、数値データ型を含め、MarkLogic の豊富なデータ型すべてに完全にアクセスできます。一般に、サーバーサイド JavaScript は、MarkLogic の数値データ型を JavaScript の `Number` にマップします。ただし、MarkLogic は、JavaScript の `Number` を返さないで数値を MarkLogic の数値型にラップする場合がいくつかあります。次の場合がこれに該当します。

- 値が JavaScript の `Number` でオーバーフローする場合や、精度を失う可能性がある場合、MarkLogic は、その値を数値データ型（例えば `xs.decimal`）にラップします。
- 返される値に頻度情報（例えば、検索から返される数値）が含まれている場合、その値は `xs.integer`、`xs:double`、`xs:float` など、数値型にラップされます。
- 返される値が JavaScript の future である場合、MarkLogic はその値を数値型にラップします。

1.6 Query Console での JavaScript

Query Console は、MarkLogic をインストールするとデフォルトではポート 8000 に設定されます。Query Console を使用すると、サーバーサイド JavaScript を使用して JavaScript を評価できるので、サンプルを非常に簡単に試すことができます。例えば、以下の各例は Query Console で実行できる「hello world」の例です（Query Type として JavaScript を選択します）。

```
"hello world"

fn.concat("hello ", "world")
```

どちらも `hello world` という文字列を返します。Query Console の詳細については、『*Query Console User Guide*』を参照してください。

1.7 サーバーサイド JavaScript でのプログラミング

sjs ファイル拡張子の JavaScript モジュールをアプリケーションサーバーのルート下に配置している場合は、アプリケーションサーバーから HTTP でそのモジュールを評価できます。例えば、ルートが /space/appserver で、ポートが 1234 に設定されている HTTP アプリケーション サーバーを使用している場合は、次のファイルを /space/appserver/my-js.sjs として保存できます。

```
xdmp.setResponseContentType("text/plain");
"hello"
```

ブラウザで http://localhost:1234/my-js.sjs（ブラウザと異なるコンピュータ上にある場合は localhost 部分を置き換えます）をポイントし、このモジュールを評価すると、hello という文字列が返されます。

.js ファイル拡張子（MIME タイプ application/javascript）を持つサーバーサイド JavaScript モジュールは、アプリケーションサーバーから直接は提供できません。直接提供されるモジュールには、.sjs の拡張子（MIME タイプ application/vnd.marklogic-javascript）が必要です。ただし、「JavaScript モジュールを JavaScript プログラムにインポートする」（55 ページ）で説明しているように、どちらの拡張子の JavaScript ライブラリもインポートできます。

1.8 スクリプトでの xdmp.invoke または xdmp.invokeFunction の使用

1 つのプログラムで複数のタスクのスクリプトを実行していて、あるタスクが前のタスクからの更新に依存する場合、xdmp.invoke または xdmp.invokeFunction を使用する JavaScript モジュールを作成できます。この場合、xdmp.invoke または xdmp.invokeFunction の呼び出しでは、個別のトランザクションでコンテンツを評価できます。

1.9 例外処理

XQuery の操作に慣れている場合や、XQuery とサーバーサイド JavaScript の両方で開発している場合は、2 つの言語で例外処理のセマンティックが同じではないことを認識する必要があります。

MarkLogic は、JavaScript 用の標準の例外処理セマンティックを実装しています。try ブロック内の JavaScript ステートメントは、キャッチされる例外が発生する前に完了した場合、ロールバックされません。XQuery では、try ブロックで評価されるすべての式は、例外がキャッチされた場合でもロールバックされます。

例えば、次のコードでは、xdmp.documentSetMetadata データへの呼び出しは、同じトランザクションでドキュメントメタデータを 2 回更新しようとすることになるので、XDMP-CONFLICTINGUPDATES 例外をスローします。例外は try-catch でトラップされます。最初のドキュメント挿入は、例外が発生する前に評価されるので成功します。

```
'use strict';
declareUpdate();

try{
  xdmp.documentInsert("doc.json",
                      {content :"value"},
                      {metadata:{a:1, b:2}})
  xdmp.documentSetMetadata("doc.json", {c:3})
} catch(err) {
  err.toString();
}
```

等価な XQuery コードの場合、「doc.json」は挿入されません。詳細については、『*XQuery and XSLT Reference Guide*』の「try/catch Expression」を参照してください。

2.0 MarkLogic JavaScript のオブジェクト API

この章では、MarkLogic のサーバーサイド JavaScript に含まれるオブジェクト 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。

- ノードとドキュメント API
- XML DOM API
- Value オブジェクト
- JSON ノードへのアクセス
- Sequence
- ValueIterator
- JavaScript instanceof 演算子
- JavaScript エラー API
- JavaScript の console オブジェクト
- JavaScript の期間と日付の算術メソッドと比較メソッド
- MarkLogic の JavaScript 関数

2.1 ノードとドキュメント API

多くの場合、MarkLogic API は、ノードやドキュメントを対象に想定し、それらを返します。このような API を JavaScript で簡単に使用するため、MarkLogic にはビルトインの `Node` オブジェクトと `Document` オブジェクトが追加されています。これらはオブジェクトですが、グローバルオブジェクトではありません。このセクションでは、これらのオブジェクトのインターフェイスについて説明します。以下のように構成されています。

- Node オブジェクト
- Document オブジェクト

2.1.1 Node オブジェクト

`Node` は、要素ノード、ドキュメントノード、テキストノードなどの任意のノードとして使用できます。関数がサーバーサイド JavaScript 内でノードを返す場合、次のプロパティを使用して `Node` オブジェクトを調べることができます。

プロパティ	説明
`baseURI`	ノードのベース URI を表す文字列。
`valueOf()`	ノードのアトミック値。

プロパティ	説明	
nodeType	Node オブジェクトの型を表す数値。nodeType で使用可能な各値の意味は以下のとおりです。	
	ELEMENT_NODE	1
	ATTRIBUTE_NODE	2
	TEXT_NODE	3
	PROCESSING_INSTRUCTION_NODE	7
	COMMENT_NODE	8
	DOCUMENT_NODE	9
	BINARY_NODE	13
	NULL_NODE	14
	BOOLEAN_NODE	15
	NUMBER_NODE	16
	ARRAY_NODE	17
	OBJECT_NODE	18
toObject()	ノードの型が Array、Boolean、Number、Object、または Text の場合は JavaScript オブジェクトであり、それ以外の場合は、Undefined です。	
xpath(String XPathExpression, Object NamespaceBindings)	XPath 式を評価します。最初の引数は XPath 式を表す文字列で、2 番目の引数はオブジェクトです。各キーは名前空間のプレフィックスであり、最初の引数で使用されます。各値は、プレフィックスがバインドされる名前空間です。XPath 式では、現在のノードに対して相対的に式を評価する必要がある場合は、「./my-node」のようにパスの先頭にドット（.）を付けます。	

XML ノードで使用可能な追加の DOM プロパティ（document、element、attribute、processing instruction、comment など）については、「XML ノードの Node オブジェクト」（13 ページ）を参照してください。

次に、xpath 関数を Node オブジェクトに対して使用する例を示します。cts.doc 関数は Node、特に Document ノードを返します。Document ノードは Node から xpath メソッドを継承します。Node.xpath の 2 番目のパラメータは、XML 名前空間プレフィックス「bar」を XML 名前空間 URI「bar」にバインドします。

```
// assume a document created as follows:
declareUpdate();
xdmp.documentInsert("/my/doc.xml", fn.head(xdmp.unquote(
  '<bar:foo xmlns:bar="bar"><bar:hello><bar:goodbye \n\
     attr="attr value">bye</bar:goodbye>\n\
  </bar:hello>\n\
</bar:foo>')));

// Use the Node.xpath method on the document node:
const node = cts.doc("/my/doc.xml");
node.xpath("//bar:goodbye/@attr", {"bar":"bar"});

// Running in Query Console displays the following value
(as an
// attribute node): "attr value"
```

2.1.2 Document オブジェクト

Document オブジェクトは、上記の Node オブジェクトからすべてのプロパティを継承し、さらに次の追加のプロパティを持ちます。

プロパティ	説明	
documentFormat	ドキュメントノードの形式を表す文字列。documentFormat で使用可能な各値の意味は以下のとおりです。	
	BINARY	"BINARY"
	JSON	"JSON"
	TEXT	"TEXT"
	XML	"XML"
root	ドキュメントのルートノード。	

2.2 XML DOM API

MarkLogic には、XML ノードに読み取り専用でアクセスできるようにする XML DOM API が実装されています。このセクションでは、これらの API について説明します。以下のように構成されています。

- XML ノードの Node オブジェクト
- ドキュメントノードの Document オブジェクト
- NodeBuilder API
- 要素
- Attr
- CharacterData および部分型
- TypeInfo
- NamedNodeMap
- NodeList

2.2.1 XML ノードの Node オブジェクト

「Node オブジェクト」（10 ページ）で説明されている `Node` の各プロパティに加え、すべての XML ノード型は、次のように `Node` の W3C DOM API のサブセットを持ちます。

プロパティ	説明
`childNodes`	このノードのすべての子を含むイテレータ。
`firstChild`	このノードの最初の子。そのようなノードがない場合は、null を返します。最初の子ノードは、要素の子に限らず任意の種類のものを返します。そのため、最初の子が空白スペースが含まれるテキストノードである場合は、そのノードが返されます。
`lastChild`	このノードの最後の子。そのようなノードがない場合は、null を返します。最後の子ノードは、要素の子に限らず任意の種類のものを返します。そのため、最後の子が空白のスペースが含まれるテキストノードである場合は、そのノードが返されます。

プロパティ	説明
localname	このノードの QName（Qualified name）のローカル名部分を返します。ELEMENT_NODE または ATTRIBUTE_NODE 以外のすべての型のノードについては、常に null を返します。
namespaceURI	このノードの名前空間 URI。指定されていない場合は null です。ELEMENT_NODE または ATTRIBUTE_NODE 以外のすべての型のノードについては、常に null を返します。
nextSibling	このノードのすぐ後に続くノード。そのようなノードがない場合は、null を返します。
nodeName	このノードの名前。型によって異なります。
nodeValue	このノードの値。型によって異なります。
ownerDocument	ノードが属するドキュメント。
parentNode	ノードの親であるノード。
prefixSibling	ツリー内の前のノードを表すノード。そのようなノードが存在しない場合は null です。
hasChildNodes()	ノードに子ノードがあるかどうかを示すブール値。
hasAttributes()	ノードに属性があるかどうかを示すブール値。
attributes	すべての属性の NamedNodeMap（存在する場合）。ELEMENT_NODE 以外の型のノードの場合、このマップは空になります。
baseURI	このノードのベース URI（存在する場合）。
textContent	ノードの fn.string と似ていますが、ドキュメントノードが null である点が異なります。
isSameNode(Node other)	2 つのノードが同じである場合は true を返します（ノードに対する XQuery 演算子 = と同様です）。
isEqualNode(Node other)	2 つのノードが等しい場合は true を返します（XQuery の fn:deep-equals と似ていますが、すべてのものを型指定なしとして扱う点が異なります）。
insertBefore(Node newChild, Node refChild)	NO_MODIFICATION_ALLOWED エラーを生成します。

プロパティ	説明
`replaceChild( Node newChild, Node oldChild)`	NO_MODIFICATION_ALLOWED エラーを生成します。
`removeChild( Node oldChild)`	NO_MODIFICATION_ALLOWED エラーを生成します。
`appendChildNodes( Node newChild)`	NO_MODIFICATION_ALLOWED エラーを生成します。
`normalize()`	何も実行しません（MarkLogic ドキュメントはすでに正規化されています）。

DOM API は、XML ノードに対する読み取り専用アクセスを提供します。DOM API がノードを変更しようとすると、DOM エラー `NO_MODIFICATION_ALLOWED_ERR` が生成されます。

2.2.2 ドキュメントノードの Document オブジェクト

`Document` オブジェクトは、Node オブジェクトからのプロパティに加え、上記の XML ノードの Node オブジェクトからすべてのプロパティを継承し、さらに次の追加のプロパティを持ちます。

プロパティ	説明
`documentElement`	ドキュメントの直接の子である要素。
`documentURI`	ドキュメントの URI。
`getElementsByTagName( String tagname)`	指定したタグ名を持つドキュメント内の要素の NodeList（ドキュメント順）。タグ名は文字列です。コロンが含まれている場合は、正確なプレフィックスを含む文字列とマッチされます。名前空間が指定された要素の場合は、`getElementsByTagNameNS` 関数が優先されます。

プロパティ	説明
`getElementsByTagNameNS( String namespaceURI, localname)`	指定した名前空間 URI とローカル名を持つドキュメント内の要素の NodeList（ドキュメント順）。名前空間の URI が null 値の場合、名前空間がないことを意味します。
`getElementById( String elementId)`	指定した ID を持つ要素（存在する場合）。
`importNode( Node importedNode, Boolean deep)`	NO_MODIFICATION_ALLOWED エラーを生成します。
`normalizeDocument()`	何も実行しません（MarkLogic ドキュメントはすでに正規化されています）。

2.2.3 NodeBuilder API

NodeBuilder API を使用すると、XML ノードや JSON ノードなどのノードを JavaScript で簡単に作成できます。また、次の関数とプロパティを使用できます。

関数 / プロパティ	説明
`addAttribute(String name, String value, [String URI])`	現在作成中の要素に新しい属性を追加します。重複する属性を作成することはできません。同じ名前の属性が要素内にすでに存在している場合は、XDMP-DUPATTR がスローされます。
`addComment(String text)`	現在作成中の親ノードにコメントノードを追加します。
`addDocument(String text, [String URI])`	指定した URI およびテキストコンテンツを持つドキュメントを追加します。これにより、テキスト形式のドキュメント（つまりテキストノードのルートを持つドキュメントノード）になります。

関数 / プロパティ	説明

addDocument(
  Function content,
  [String URI])
	指定した URI を持つドキュメントを追加します。この関数が引数としてビルダーに渡され、評価されてコンテンツが生成されます。以下に例を示します。
	
	`const x = new NodeBuilder();`
	`const b = x.addDocument(`
	`  function(builder){`
	`  builder.addElement("foo",`
	`  "some stuff")});`
	`b.toNode().root;`
	`=>`
	`<foo>some stuff</foo>`

addElement(String name,
  String text,
  [String URI])
	指定した名前、テキストコンテンツ、名前空間 URI の要素を現在の親ノードに追加します。この関数が引数としてビルダーに渡され、評価されてコンテンツが生成されます。要素の作成は addElement の呼び出しの後に完了します。したがって、後続の addAttribute の呼び出しはこの要素には適用されません。

addElement(String name,
  Function content,
  [String URI])
	指定した名前および名前空間 URI の要素を現在の親ノードに追加します。要素の作成は addElement の呼び出しの後に完了します。したがって、後続の addAttribute の呼び出しはこの要素には適用されません。2 番目の引数にある関数が引数としてビルダーに渡され、評価されてコンテンツが生成されます。以下に例を示します。
	
	`const x = new NodeBuilder();`
	`const b = x.addElement("foo",`
	`  function(builder){`
	`  builder.addText("some`
	`stuff")});`
	`b.toNode();`
	`=>`
	`<foo>some stuff</foo>`

addNode(Node node)	指定したノードのコピーを現在の親ノードに追加します。

関数 / プロパティ	説明
`addProcessingInstruction( String target, String text)`	指定したターゲットとテキストの処理命令ノードを現在の親ノードに追加します。
`addText(String value)`	現在作成中の親ノードにテキストノードを追加します。
`addBinary(String hex)`	現在作成中の親ノードにバイナリノードを追加します。引数は 16 進数でエンコードされた文字列です。
`addNumber(Number val)`	現在作成中の親ノードに数値ノードを追加します。
`addBoolean(Boolean val)`	現在作成中の親ノードにブール型ノードを追加します。
`addNull()`	現在作成中の親ノードに null ノードを追加します。
`endDocument()`	ドキュメントの作成を完了します。
`endElement()`	要素の作成を完了します。
`startDocument([String URI])`	指定した URI を持つドキュメントの作成を開始します。
`startElement(String name, [String URI])`	指定した名前と名前空間 URI（オプション）を使用して、現在のドキュメントまたは要素の子として要素の作成を開始します。
`toNode()`	作成されたノードを返します。

NodeBuilder を使用してノードとして作成されたノードを使用するには、最初に `toNode()` を呼び出す必要があります。

XML ドキュメントノードの作成例は、以下のとおりです。

```
const x = new NodeBuilder();
x.startDocument();
x.startElement("foo", "bar");
x.addText("text in bar");
x.endElement();
x.endDocument();
const newNode = x.toNode();
newNode;
```

```
// returns a document node with the following
serialization:
// <?xml version="1.0" encoding="UTF-8"?>
// <foo xmlns="bar">text in bar</foo>
```

2.2.4 要素

要素ノードでは次のプロパティと関数を使用できます。

プロパティ / 関数	説明
`tagName`	要素の修飾名。
`getAttribute(String name)`	名前別の属性値を返します。
`getAttributeNode(String name)`	名前別の属性ノード（`Attr`）を返します。
`getAttributeNS( String namespace, String name)`	指定した名前空間と名前を持つ属性の値を現在のノードから返します。
`getAttributeNode(String name)`	この要素の指定した属性（存在する場合）をノードとして返します。
`getAttributeNodeNS( String namespaceURI, String localname)`	一致する名前空間 URI とローカル名を持つこの要素の属性を返します。
`getElementsByTagName( String tagname)`	指定したタグ名を持つこの要素の子孫の `NodeList`（ドキュメント順）。タグ名は文字列です。コロンが含まれている場合は、正確なプレフィックスを含む文字列と一致します。名前空間が指定された要素の場合は、`getElementsByTagNameNS` が優先されます。
`getElementsByTagNameNS( String namespaceURI, String localname)`	指定した名前空間 URI とローカル名を持つ要素の子孫の `NodeList`（ドキュメント順）。名前空間の URI が null 値の場合、名前空間がないことを意味します。
`hasAttribute(String name)`	要素が指定した属性を持っている場合は true を返します。

プロパティ / 関数	説明
`hasAttributeNS( String namespaceURI, String localname)`	要素が指定した名前空間 URI とローカル名の属性を持っている場合は true を返します。
`schemaTypeInfo`	要素の `TypeInfo`（型情報）。
`setAttribute(String name, String value)`	NO_MODIFICATION_ALLOWED エラーを生成します。
`removeAttribute(String name)`	NO_MODIFICATION_ALLOWED エラーを生成します。
`setAttributeNode(Attr newAttr)`	NO_MODIFICATION_ALLOWED エラーを生成します。
`removeAttributeNode( Attr newAttr)`	NO_MODIFICATION_ALLOWED エラーを生成します。
`setAttributeNS( String namespaceURI, String localname)`	NO_MODIFICATION_ALLOWED エラーを生成します。
`removeAttributeNS( String namespaceURI, String localname)`	NO_MODIFICATION_ALLOWED エラーを生成します。
`setIdAttribute( String name, Boolean isId)`	NO_MODIFICATION_ALLOWED エラーを生成します。
`setIdAttributeNS( String namespaceURI, String localname, Boolean isId)`	NO_MODIFICATION_ALLOWED エラーを生成します。
`setIdAttributeNode( Attr idAttr, Boolean isId)`	NO_MODIFICATION_ALLOWED エラーを生成します。

2.2.5 Attr

属性（Attr）ノードでは、ノードが継承する XMLNode プロパティに加えて、次のプロパティを使用できます。

プロパティ	説明
name	この属性の修飾名。
specified	属性が明示的か（true）スキーマのデフォルトとして指定されたか（false）を示すブール値。
value	文字列としてのこの属性の値。
ownerElement	この属性を持っている要素。
isId	これが ID 属性かどうかを示すブール値（型は xs:ID です）。
schemaTypeInfo	要素の TypeInfo（型情報）。

DOM4 では Attr オブジェクトの使用は推奨されません。

2.2.6 CharacterData および部分型

CharacterData は、XMLNode からすべての API を継承し、さらに次の追加のプロパティとメソッドを使用できます。また、Text ノード、Comment ノード、および Processing Instruction ノードから継承した部分型を持ちます。これらについても表に記載しています。

関数 / プロパティ	説明
data	ノードのテキストコンテンツ（fn:data と同じです）。
length	ノードのテキストコンテンツ内の文字数。
substringData(Number offset, Number count)	テキストコンテンツのサブ文字列。指定した文字数のオフセット位置から始まり、指定した文字数だけ続きます。

関数 / プロパティ	説明
`isElementContentWhitespace`	テキストノードが無視できるスペースである場合は true になります。MarkLogic では無視できるスペースが読み込み時に削除されるため、MarkLogic のコンテキストではこれは通常は false になります。ただし、データのスキーマが読み込まれる前にデータが読み込まれた場合は true になる可能性があります。（テキストノードのみ）
`wholeText`	論理的に隣接するテキストノードと連結されたこのノードの値を返します。MarkLogic では、論理的に隣接するテキストノードとすでに連結されているので、これは単にノード自体の値になります。（テキストノードのみ）
`target`	処理命令の対象。例えば、PI が `<?example something?>` である場合、`example` がターゲットで、`something` がデータになります。
`appendData(String arg)`	NO_MODIFICATION_ALLOWED エラーを生成します。
`insertData(Number offset, Number count)`	NO_MODIFICATION_ALLOWED エラーを生成します。
`deleteData(Number offset, Number count)`	NO_MODIFICATION_ALLOWED エラーを生成します。
`replaceData(Number offset, Number count, String arg)`	NO_MODIFICATION_ALLOWED エラーを生成します。
`replaceWholeText( String content)`	NO_MODIFICATION_ALLOWED エラーを生成します。（テキストノードのみ）
`splitText(Number offset)`	NO_MODIFICATION_ALLOWED エラーを生成します。（テキストノードのみ）

2.2.7 TypeInfo

TypeInfo の関数とプロパティは、以下に示すとおりです。さらに、スキーマコンポーネントとメソッドがバインドされています。

関数 / プロパティ	説明
`typeName`	型のローカル名。
`typeNamespace`	型の名前空間 URI。
`isDerivedFrom( String typeNamespace, String typeName, unsigned long derivationMethod)`	この型が引数で指定されている型から派生している場合は true を返します。派生メソッドの引数は、許容可能な派生メソッドを示すフラグです（0 はすべてのメソッドが許容可能なことを意味します）。組み合わせ可能なフラグの値は次のとおりです。 DERIVATION_RESTRICTION (0x1) DERIVATION_EXTENSION (0x2) DERIVATION_UNION (0x4) DERIVATION_LIST (0x8)

2.2.8 NamedNodeMap

NamedNodeMap の関数とプロパティは、以下に示すとおりです。

関数 / プロパティ	説明
`length`	マップ内のノードの個数。
`getNamedItem(name)`	指定した名前を持つマップ内のノード（存在する場合）を返します。名前空間が指定されたノードの場合は、`getNamedItemNS` が優先されます。
`getNamedItemNS( String namespaceURI, String localName)`	指定した名前空間 URI とローカル名を持つマップ内のノードを返します。
`item(Number index)`	インデックスの位置（1 番目、2 番目など）でノードを取得します。

関数 / プロパティ	説明
`setNamedItem(Node arg)`	NO_MODIFICATION_ALLOWED エラーを生成します。
`removeNamedItem(` `  String name)`	NO_MODIFICATION_ALLOWED エラーを生成します。
`setNamedItemNS(Node arg)`	NO_MODIFICATION_ALLOWED エラーを生成します。

`removeNamedItem(`
`  String namespaceURI,`
`  String localname)`
	NO_MODIFICATION_ALLOWED エラーを生成します。

2.2.9 NodeList

`NodeList` は、次の追加のプロパティを持つイテレータです。

プロパティ	説明
`length`	リスト内のノードの個数。
`item(Number index)`	インデックスの位置（1 番目、2 番目など）で項目を取得します。

2.3 Value オブジェクト

`Value` オブジェクトは、MarkLogic XQuery の型をラップするラッパーオブジェクトです。これを使用することで、MarkLogic XQuery の型用に設計された関数に、このようなオブジェクトを渡すことができます。わかりやすい例として DateTime 関数が挙げられます。例えば、次のコードについて考えてみましょう。

```
fn.currentDate() instanceof Value
```

これは、XML 型 `xs:date` と同様なので `true` を返します（JavaScript では `xs.date` として出力されます）。MarkLogic の多くのビルトイン JavaScript 関数は、このように Value オブジェクトを返します。このオブジェクトで使用できるメソッドについては、「Value オブジェクト」を参照してください。日付の詳細については、サーバーサイド JavaScript 内の日付を参照してください。

`Value` オブジェクトでは、オブジェクトを使用して、JavaScript の XQuery 関数で使用される型を活用できます。さらに、`Value` オブジェクトで `toObject()` を呼び出して、最も近い JavaScript ネイティブの型に変換することもできます。

2.4 JSON ノードへのアクセス

JSON を MarkLogic データベースに格納すると、JSON ノードの子を持つドキュメントノードとして格納されます。データベース内に格納されている JSON ドキュメントには、`fn.doc` またはドキュメントを返す他の任意の関数を使用してアクセスできます。また JSON ノードには、ネイティブな JavaScript プロパティを使用して読み取り専用で直接アクセスできます。例えば、名前付きプロパティの取得、名前付きプロパティの存在をチェックするクエリ、すべての使用可能な名前付きプロパティの列挙などを実行できます。

JavaScript オブジェクトを JSON ノードに変換する場合は、JavaScript オブジェクトに対して `xdmp.toJson` を呼び出します。これにより、JSON ノードが返されます。

JSON のノードとドキュメントの詳細については、『*Application Developer's Guide*』の「Working With JSON」を参照してください。

2.5 Sequence

`Sequence` は、一連の値を表現する、JavaScript の `Iterable` オブジェクトです。複数の値を返す多くの MarkLogic 関数は、サーバーサイド JavaScript では `Sequence` を返すことで複数の値を返します。反復可能オブジェクトは、`@@iterator` メソッドからのイテレータオブジェクトを返す JavaScript オブジェクトです。

`for..of loop` を使用して、`Sequence` 内の値にわたって反復できます。以下に例を示します。

```
for (const doc of fn.collection('/my/coll')) {
  // do something with doc
}
```

反復を使用せずに `Sequence` 内の最初の（または唯一の）項目だけを抽出する場合は、`fn.head` を使用してください。例えば、`xdmp.unquote` 関数は `Sequence` を返しますが、多くの場合、その `Sequence` は項目を 1 つしか含まない `Sequence` です。したがって、次のようなコードでその単一の結果を抽出できます。

```
const node = fn.head(xdmp.unquote('<data>some xml</data>'))
```

配列、配列類、または別の反復可能オブジェクトから `Sequence.from` を使用して、独自の `Sequence` オブジェクトを作成できます。例えば、次のコードの一部分は配列から `Sequence` を作成します。

```
const mySeq = Sequence.from([1,2,3,4]);
```

`Sequence` 内の項目の数を数えるには、`fn.count` を使用します。以下に例を示します。

```
const mySeq = Sequence.from([1,2,3,4]);
fn.count(mySeq);        // returns 4
```

詳細については、『*MarkLogic Server-Side JavaScript Function Reference*』の「*Sequence Object*」を参照してください。

2.6 ValueIterator

注： このインターフェイスは推奨されていません。MarkLogic 9 現在、`ValueIterator` を返したり、`ValueIterator` を入力として利用したりする MarkLogic 関数はありません。このセクションにあるガイドラインを使用して、コードを Sequence に移行してください。

次の表で説明するように、`ValueIterator` の結果を操作するコードは、戻り値の型を `Sequence` にしても透過的に動作します。

ガイドライン	推奨	非推奨

MarkLogic で返された `ValueIterator` 内の値にわたって反復するには、`for...of` ループのみを使用します。基になる Iterator インターフェイスにそのままプログラムしないでください。	`const uris = cts.uris('/');`
	`for (const u of uris) {`
	`  // do something with u`
	`}`
		`const uris = cts.uris('/');`
		`uris.next.value();`
		`uris.next.value();`
		`...`

パターン *results*`.next().value` を使用するのではなく、`fn.head` を使用して、`ValueIterator` 内の最初の項目にアクセスします。	`fn.head(`
	`  cts.uris('/')`
	`);`
		`cts.uris('/').next().value`

`count` プロパティではなく、`fn.count` を使用して、`ValueIterator` オブジェクト内の項目の数を数えます。	`fn.count(`
	`  cts.uris('/')`
	`);`
		`cts.uris('/').count`

ValueIterator のプロパティと、メソッド next、count、および clone に依存するコードは、Sequence 値とともには使用できません。

ValueIterator を操作するか、Sequence を操作するかで異なるように動作するコードを作成する場合は、instanceof 演算子を使用できます。MarkLogic 8 の関数で Sequence を返すものはないので、そのコードが MarkLogic 8 では実行されないことがはっきりとします。

例えば、次のコードは、MarkLogic 8 の場合は ValueIterator.clone を使用して反復にわたって一連の値を保持していますが、結果の型が Sequence になった場合は不要なクローニングをスキップしています。

```
const results = cts.uris('/', ['limit=10']);
const clone = {};
if (uris instanceof ValueIterator) {
  // iterator destructive, so clone to preserve orig
  clone = results.clone();
} else if (results instanceof Sequence) {
  // iteration is not destructive, no clone needed
  clone = results;
}
for (const val of clone) {
  // do something with val
}
```

2.7 JavaScript instanceof 演算子

JavaScript の instanceof 演算子は、（JavaScript の型に加え）MarkLogic の型のテストに利用できます。例えば、次のプログラムは true を返します。

```
const a = Sequence.from(["saab", "alfa romeo", "tesla"]);
a instanceof Sequence;
// returns true
```

同様に、MarkLogic オブジェクトの型や JavaScript オブジェクトの型とともに instaceof を使用したその他の例は以下のとおりです。

xs.date オブジェクトの型：

```
const a = fn.currentDate();
a instanceof xs.date;
// returns true
```

xs.date オブジェクトではない型：

```
const a = fn.currentDate().toObject();
a instanceof xs.date;
// returns false
```

JavaScript の Date オブジェクトの型：

```
const a = fn.currentDate().toObject();
a instanceof Date;
// returns true
```

instanceof を使用して、MarkLogic オブジェクトの次の型のいずれについてもテストできます。

- Value（MarkLogic オブジェクトの型はすべて、Value の部分型です）
- xs.anyAtomicType
- cts.query（およびその部分型すべて — 部分型は cts.query のインスタンスでもあります）
- ArrayNode
- BinaryNode
- BooleanNode
- ObjectNode
- XMLNode
- Document
- Node
- NodeBuilder
- Attr
- CharacterData
- Comment
- Sequence
- Text
- Element
- ProcessingInstruction
- XMLDocument
- ValueIterator

XML ドキュメントを使用した例は、次のとおりです。

```
// assume "/one.xml" has content <name>value</name>
const a = fn.head(fn.doc("/one.xml")).root;
const b = fn.head(a.xpath("./text()"));
b instanceof Text;
// returns true
```

JSON ドキュメントを使用した例は、次のとおりです。

```
// Assume "/car.json" has the content:
// {"car":"The fast electric car drove down the highway."}
const res = new Array();
const a = fn.head(fn.doc("/car.json"));
res.push(a instanceof Document);
const b = a.root;
res.push(b instanceof ObjectNode);
res;
// returns [true, true]
```

同様に、いずれの XML 型についてもテストできます。JavaScript での XML 型では、名前空間と型名の間にコロン（:）ではなくドット（.）が使用されます。例えば、`xs.integer`、`xs.string` などです。

2.8 JavaScript エラー API

サーバーサイド JavaScript プログラムでエラーや例外がスローされると、スタックがスローされます。これは、標準的な JavaScript の `try/catch` ブロックを使用してキャッチできます。個別のエラーメッセージの詳細については、『*Messages and Codes Reference Guide*』を参照してください。このセクションは、以下のように構成されています。

- JavaScript エラーのプロパティと関数
- JavaScript stackFrame のプロパティ
- JavaScript の try/catch の例

2.8.1 JavaScript エラーのプロパティと関数

JavaScript の例外で使用できる API は以下のとおりです。

プロパティ / 関数	説明
`code`	コード番号を表す文字列。DOM エラーでのみ使用することができ、数値が DOM エラーコードになります。
`data`	エラーでスローされるデータが含まれる文字列の配列。
`message`	エラーメッセージの文字列。
`name`	エラーコードの文字列。
`retryable`	エラーが再試行可能かどうかを示すブール値。
`stack`	JavaScript スタック。XQuery からエラーがスローされた場合、スタックには XQuery と JavaScript の両方からスローされた連結されたスタックが格納されることになります。
`stackFrame`	スタックフレームの配列。スタックフレームについて詳しくは、以下の stackFrame の表を参照してください。詳細については、「JavaScript stackFrame のプロパティ」（30 ページ）を参照してください。
`toString()`	データと一緒に作成された、書式設定されたエラーメッセージ。

2.8.2 JavaScript stackFrame のプロパティ

次に、各 `stackFrame` で使用できる API を示します。

プロパティ	説明
`line`	現在のフレームの行番号。
`column`	現在のフレームから始まるカラム番号。
`operation`	現在のフレームの関数名または演算。

プロパティ	説明
uri	このフレームの関数または演算のスクリプトが含まれるリソースの名前。またはスクリプト名が未定義で、ソースの末尾が //# sourceURL=… 文字列または非推奨の //@ sourceURL=… 文字列になります。
language	現在のフレームのクエリ言語。
isEval	関連付けられている関数が eval の呼び出しからコンパイルされたかどうか（JavaScript のみ）。
variables	フレーム内の変数のバインドを含む（名前、値）オブジェクトの配列。使用可能な変数のバインドがない場合は未定義です（XQuery のみ）。
contextItem	フレーム内のコンテキスト項目。使用可能なコンテキスト項目がない場合は未定義です（XQuery のみ）。
contextPosition	フレーム内のコンテキストの位置。使用可能なコンテキスト項目がない場合は未定義です（XQuery のみ）。

2.8.3 JavaScript の try/catch の例

次に、単純な JavaScript の try/catch の例を示します。

```
try{ xdmp.documentInsert("/foo.json", {"foo": "bar"} ); }
catch (err) { err.toString(); }

=> catches the following error
( because it is missing the declareUpdate() call )
XDMP-UPDATEFUNCTIONFROMQUERY: xdmp:eval("// query&#10;
  try{ xdmp.documentInsert(&quot;/foo.json&quot;, {&q...",
())
  -- Cannot apply an update function from a query
```

2.9 JavaScript の console オブジェクト

MarkLogic には `console` オブジェクトが実装されており、ここには、MarkLogic データディレクトリ内の ErrorLog.txt に出力をログ記録するためのさまざまなことを行う関数が含まれます。`console` の関数は次のとおりです。

- `console.assert`
- `console.debug`
- `console.dir`
- `console.error`
- `console.log`
- `console.trace`
- `console.warn`

2.10 JavaScript の期間と日付の算術メソッドと比較メソッド

XQuery には、期間型のデータに対して日付計算を実行できる演算子があり、期間を減算して dateTime 値を返すなど、さまざまなことを行うことができます。サーバーサイド JavaScript では、さまざまな dateTime 期間型で返されたデータを取得し、期間メソッドを使用してこれらの期間を加算、減算、乗算、除算、および比較できます。このセクションでは、これらの期間の算術および比較メソッドについて説明し、次のパートを含みます。

- 期間に関する算術メソッド
- 期間、日付、および時刻に関する算術メソッド
- 期間、日付、時刻の値に関する比較メソッド

2.10.1 期間に関する算術メソッド

次の期間オブジェクトで、算術メソッドを使用できます。

- xs.yearMonthDuration のメソッド
- xs.dayTimeDuration のメソッド

2.10.1.1 xs.yearMonthDuration のメソッド

xs.yearMonthDuration のインスタンスである JavaScript オブジェクトは、http://www.w3.org/TR/xpath-functions/#dt-yearMonthDuration で説明されているように、XQuery の xs:yearMonthDuration 型に相当し、これと同じレキシコン表現を持ちます。xs.yearMonthDuration オブジェクトでは、次のメソッドを使用できます。

メソッド	説明
add(xs.yearMonthDuration)	2 つの xs.yearMonthDuration 値を加算します。xs.yearMonthDuration を返します。
subtract(xs.yearMonthDuration)	xs.yearMonthDuration 値を別の値から減算します。xs.yearMonthDuration を返します。
multiply(Number)	xs.yearMonthDuration 値に Number を乗算します。xs.yearMonthDuration を返します。
divide(Number)	xs.yearMonthDuration を Number で除算します。xs.yearMonthDuration を返します。
divide(xs.yearMonthDuration)	xs.yearMonthDuration を xs.yearMonthDuration で除算します。Number を返します。

次に、これらのメソッドを使用した簡単な例をいくつか示します。

```
const v1 = xs.yearMonthDuration("P3Y7M");
const v2 = xs.yearMonthDuration("P1Y4M");

const r = {
"v1 + v2" : v1.add(v2),
"v1 - v2" : v1.subtract(v2),
"v1 * 2" : v1.multiply(2),
"v1 / 2" : v1.divide(2),
"v1 / v2" : v1.divide(v2)
};
r;

/*
```

```
returns:
{"v1 + v2":"P4Y11M",
 "v1 - v2":"P2Y3M",
 "v1 * 2":"P7Y2M",
 "v1 / 2":"P1Y10M",
 "v1 / v2":2.6875}
*/
```

2.10.1.2 xs.dayTimeDuration のメソッド

xs.dayTimeDuration のインスタンスである JavaScript オブジェクトは、http://www.w3.org/TR/xpath-functions/#dt-dayTimeDuration で説明されているように、XQuery xs:dayTimeDuration 型に相当し、これと同じレキシコン表現を持ちます。xs.dayTimeDuration オブジェクトでは、次のメソッドを使用できます。

メソッド	説明
add(xs.dayTimeDuration)	2 つの xs.dayTimeDuration 値を加算します。xs.dayTimeDuration を返します。
subtract(xs.dayTimeDuration)	xs.dayTimeDuration 値を別の値から減算します。xs.dayTimeDuration を返します。
multiply(Number)	xs.dayTimeDuration 値に Number を乗算します。xs.dayTimeDuration を返します。
divide(Number)	xs.dayTimeDuration を Number で除算します。xs.dayTimeDuration を返します。
divide(xs.dayTimeDuration)	xs.dayTimeDuration を xs.dayTimeDuration で除算します。Number を返します。

次に、これらのメソッドを使用した簡単な例をいくつか示します。

```
const v1 = xs.dayTimeDuration("P5DT4H");
const v2 = xs.dayTimeDuration("P1DT1H");

const r = {
"v1 + v2" : v1.add(v2),
"v1 - v2" : v1.subtract(v2),
"v1 * 2" : v1.multiply(2),
"v1 / 2" : v1.divide(2),
"v1 / v2" : v1.divide(v2)
};
r;

/*
returns:
{"v1 + v2":"P6DT5H",
 "v1 - v2":"P4DT3H",
 "v1 * 2":"P10DT8H",
 "v1 / 2":"P2DT14H",
 "v1 / v2":4.96}
*/
```

2.10.2 期間、日付、および時刻に関する算術メソッド

次の期間、日付、および dateTime オブジェクトに対して使用できるメソッドがあります。

- xs.dateTime のメソッド
- xs.date のメソッド
- xs.time のメソッド

2.10.2.1 xs.dateTime のメソッド

xs.dateTime オブジェクトでは、次のメソッドを使用できます。

メソッド	説明
add(xs.dayTimeDuration)	期間の始まりとなる xs.dateTime 値に xs.dayTimeDuration を加算することにより、期間の終わりを表す xs.dateTime 値を返します。xs.dateTime を返します。
add(xs.yearMonthDuration)	期間の始まりとなる xs.dateTime 値に xs.yearMonthDuration を加算することにより、期間の終わりを表す xs.dateTime 値を返します。xs.dateTime を返します。
subtract(xs.dateTime)	2 つの xs.dateTime 値の差を xs.dayTimeDuration として返します。
subtract(xs.dayTimeDuration)	期間の終わりとなる xs.dateTime 値から xs.yearMonthDuration を減算することにより、期間の始まりを表す xs.dateTime 値を返します。xs.dateTime を返します。
subtract(xs.dayTimeDuration)	期間の終わりとなる xs.dateTime 値から xs.dayTimeDuration を減算することにより、期間の始まりを表す xs.dateTime 値を返します。xs.dateTime を返します。

次に、これらのメソッドを使用した簡単な例をいくつか示します。

```
const v1 = xs.dateTime(xs.date('2013-08-15'),
xs.time('12:30:45-05:00'))
const v2 = xs.dateTime(xs.date('2012-04-01'),
xs.time('01:10:25-02:00'))
const v3 = xs.yearMonthDuration("P3Y3M")
const v4 = xs.dayTimeDuration("PT1H")

const r = {
"v1 + v3" : v1.add(v3),
"v1 + v4" : v1.add(v4),
"v1 - v2" : v1.subtract(v2),
"v1 - v3" : v1.subtract(v3),
"v1 - v4" : v1.subtract(v4)
```

```
};
r;

/*
returns:
{"v1 + v3":"2016-11-15T12:30:45-05:00",
 "v1 + v4":"2013-08-15T13:30:45-05:00",
 "v1 - v2":"P501DT14H20M20S",
 "v1 - v3":"2010-05-15T12:30:45-05:00",
 "v1 - v4":"2013-08-15T11:30:45-05:00"}
*/
```

2.10.2.2 xs.date のメソッド

xs.date オブジェクトでは、次のメソッドを使用できます。

メソッド	説明
add(xs.dayTimeDuration)	期間の始まりとなる xs.date 値に xs.dayTimeDuration を加算することにより、期間の終わりを表す xs.date 値を返します。xs.date を返します。
add(xs.yearMonthDuration)	期間の始まりとなる xs.date 値に xs.yearMonthDuration を加算することにより、期間の終わりを表す xs.date 値を返します。xs.date を返します。
subtract(xs.date)	2 つの xs.date 値の差を xs.dayTimeDuration として返します。
subtract(xs.dayTimeDuration)	期間の終わりとなる xs.date 値から xs.yearMonthDuration を減算することにより、期間の始まりを表す xs.date 値を返します。xs.date を返します。
subtract(xs.dayTimeDuration)	期間の終わりとなる xs.date 値から xs.dayTimeDuration を減算することにより、期間の始まりを表す xs.date 値を返します。xs.date を返します。

次に、これらのメソッドを使用した簡単な例をいくつか示します。

```
const v1 = xs.date('2013-08-15')
const v2 = xs.date('2012-04-01')
const v3 = xs.yearMonthDuration("P3Y3M")
const v4 = xs.dayTimeDuration("P1DT3H")

const r = {
"v1 + v3" : v1.add(v3),
"v1 + v4" : v1.add(v4),
"v1 - v2" : v1.subtract(v2),
"v1 - v3" : v1.subtract(v3),
"v1 - v4" : v1.subtract(v4)
};
r;

/*
returns:
{"v1 + v3":"2016-11-15",
 "v1 + v4":"2013-08-16",
 "v1 - v2":"P501D",
 "v1 - v3":"2010-05-15",
 "v1 - v4":"2013-08-13"}
*/
```

2.10.2.3 xs.time のメソッド

xs.time オブジェクトでは、次のメソッドを使用できます。

メソッド	説明
add(xs.dayTimeDuration)	xs.dayTimeDuration の時間、分、および秒成分を xs.time 値に加算します。xs.time を返します。
subtract(xs.time)	2 つの xs.time 値の差を xs.dayTimeDuration として返します。
subtract(xs.dayTimeDuration)	xs.dayTimeDuration の時間、分、および秒成分の値を xs.time 値から減算します。xs.time を返します。

次に、これらのメソッドを使用した簡単な例をいくつか示します。

```
const v1 = xs.time('12:30:45-05:00')
const v2 = xs.time('01:10:25-02:00')
const v3 = xs.dayTimeDuration("PT1H")

const r = {
"v1 + v3" : v1.add(v3),
"v1 - v2" : v1.subtract(v2),
"v1 - v3" : v1.subtract(v3)
};
r;

/*
returns:
{"v1 + v3":"13:30:45-05:00",
 "v1 - v2":"PT14H20M20S",
 "v1 - v3":"11:30:45-05:00"}
*/
```

2.10.3 期間、日付、時刻の値に関する比較メソッド

比較メソッドは、次の期間、日付、および dateTime オブジェクトの値を比較する（小なり、大なりなど）ために使用できます。

- xs.yearMonthDuration の比較メソッド
- xs.dayTimeDuration の比較メソッド
- xs.dateTime の比較メソッド
- xs.date の比較メソッド
- xs.time の比較メソッド
- xs.gYearMonth の比較メソッド
- xs.gYear の比較メソッド
- xs.gMonthDay の比較メソッド
- xs.gMonth の比較メソッド
- xs.gDay の比較メソッド

2.10.3.1 xs.yearMonthDuration の比較メソッド

xs.yearMonthDuration オブジェクトでは、次の比較メソッドを使用できます。

メソッド	説明
lt(xs.yearMonthDuration)	xs.yearMonthDuration 値に対する小なり比較。Boolean を返します。
le(xs.yearMonthDuration)	xs.yearMonthDuration 値に対する小なりイコール比較。Boolean を返します。
gt(xs.yearMonthDuration)	xs.yearMonthDuration 値に対する大なり比較。Boolean を返します。
ge(xs.yearMonthDuration)	xs.yearMonthDuration 値に対する大なりイコール比較。Boolean を返します。
eq(xs.yearMonthDuration)	xs.yearMonthDuration 値に対する等値比較。Boolean を返します。
ne(xs.yearMonthDuration)	xs.yearMonthDuration 値に対する不等比較。Boolean を返します。

次に、これらのメソッドを使用した簡単な例をいくつか示します。

```
const v1 = xs.yearMonthDuration("P3Y7M");
const v2 = xs.yearMonthDuration("P1Y4M");

const r = {
"v1 lt v2" : v1.lt(v2),
"v1 le v2" : v1.le(v2),
"v1 gt v2" : v1.gt(v2),
"v1 ge v2" : v1.ge(v2),
"v1 eq v2" : v1.eq(v2),
"v1 ne v2" : v1.ne(v2)
};
r;

/*
returns:
{"v1 lt v2":false,
 "v1 le v2":false,
 "v1 gt v2":true,
 "v1 ge v2":true,
```

```
 "v1 eq v2":false,
 "v1 ne v2":true}
*/
```

2.10.3.2 xs.dayTimeDuration の比較メソッド

xs.dayTimeDuration オブジェクトでは、次の比較メソッドを使用できます。

メソッド	説明
lt(xs.dayTimeDuration)	xs.dayTimeDuration 値に対する小なり比較。Boolean を返します。
le(xs.dayTimeDuration)	xs.dayTimeDuration 値に対する小なりイコール比較。Boolean を返します。
gt(xs.dayTimeDuration)	xs.dayTimeDuration 値に対する大なり比較。Boolean を返します。
ge(xs.dayTimeDuration)	xs.dayTimeDuration 値に対する大なりイコール比較。Boolean を返します。
eq(xs.dayTimeDuration)	xs.dayTimeDuration 値に対する等値比較。Boolean を返します。
ne(xs.dayTimeDuration)	xs.dayTimeDuration 値に対する不等比較。Boolean を返します。

次に、これらのメソッドを使用した簡単な例をいくつか示します。

```
const v1 = xs.dayTimeDuration("P5DT4H");
const v2 = xs.dayTimeDuration("P1DT1H");

const r = {
"v1 lt v2" : v1.lt(v2),
"v1 le v2" : v1.le(v2),
"v1 gt v2" : v1.gt(v2),
"v1 ge v2" : v1.ge(v2),
"v1 eq v2" : v1.eq(v2),
"v1 ne v2" : v1.ne(v2)
};
r;

/*
```

```
returns:
{"v1 lt v2":false,
 "v1 le v2":false,
 "v1 gt v2":true,
 "v1 ge v2":true,
 "v1 eq v2":false,
 "v1 ne v2":true}
*/
```

2.10.3.3 xs.dateTime の比較メソッド

xs.dateTime オブジェクトでは、次の比較メソッドを使用できます。

メソッド	説明
lt(xs.dateTime)	xs.dateTime 値に対する小なり比較。Boolean を返します。
le(xs.dateTime)	xs.dateTime 値に対する小なりイコール比較。Boolean を返します。
gt(xs.dateTime)	xs.dateTime 値に対する大なり比較。Boolean を返します。
ge(xs.dateTime)	xs.dateTime 値に対する大なりイコール比較。Boolean を返します。
eq(xs.dateTime)	xs.dateTime 値に対する等値比較。Boolean を返します。
ne(xs.dateTime)	xs.dateTime 値に対する不等比較。Boolean を返します。

次に、これらのメソッドを使用した簡単な例をいくつか示します。

```
const v1 = xs.dateTime(xs.date('2013-08-15'),
xs.time('12:30:45-05:00'))
const v2 = xs.dateTime(xs.date('2012-04-01'),
xs.time('01:10:25-02:00'))

const r = {
"v1 lt v2" : v1.lt(v2),
"v1 le v2" : v1.le(v2),
"v1 gt v2" : v1.gt(v2),
"v1 ge v2" : v1.ge(v2),
"v1 eq v2" : v1.eq(v2),
"v1 ne v2" : v1.ne(v2)
};
r;

/*
returns:
{"v1 lt v2":false,
 "v1 le v2":false,
 "v1 gt v2":true,
 "v1 ge v2":true,
 "v1 eq v2":false,
 "v1 ne v2":true}
*/
```

2.10.3.4 xs.date の比較メソッド

`xs.date` オブジェクトでは、次の比較メソッドを使用できます。

メソッド	説明
`lt(xs.date)`	`xs.date` 値に対する小なり比較。`Boolean` を返します。
`le(xs.date)`	`xs.date` 値に対する小なりイコール比較。`Boolean` を返します。
`gt(xs.date)`	`xs.date` 値に対する大なり比較。`Boolean` を返します。
`ge(xs.date)`	`xs.date` 値に対する大なりイコール比較。`Boolean` を返します。

メソッド	説明
eq(xs.date)	xs.date 値に対する等値比較。Boolean を返します。
ne(xs.date)	xs.date 値に対する不等比較。Boolean を返します。

次に、これらのメソッドを使用した簡単な例をいくつか示します。

```
const v1 = xs.date('2013-08-15');
const v2 = xs.date('2012-04-01');

const r = {
"v1 lt v2" : v1.lt(v2),
"v1 le v2" : v1.le(v2),
"v1 gt v2" : v1.gt(v2),
"v1 ge v2" : v1.ge(v2),
"v1 eq v2" : v1.eq(v2),
"v1 ne v2" : v1.ne(v2)
};
r;

/*
returns:
{"v1 lt v2":false,
 "v1 le v2":false,
 "v1 gt v2":true,
 "v1 ge v2":true,
 "v1 eq v2":false,
 "v1 ne v2":true}
*/
```

2.10.3.5 xs.time の比較メソッド

xs.time オブジェクトでは、次の比較メソッドを使用できます。

メソッド	説明
lt(xs.time)	xs.time 値に対する小なり比較。Boolean を返します。
le(xs.time)	xs.time 値に対する小なりイコール比較。Boolean を返します。

メソッド	説明
gt(xs.time)	xs.time 値に対する大なり比較。Boolean を返します。
ge(xs.time)	xs.time 値に対する大なりイコール比較。Boolean を返します。
eq(xs.time)	xs.time 値に対する等値比較。Boolean を返します。
ne(xs.time)	xs.time 値に対する不等比較。Boolean を返します。

次に、これらのメソッドを使用した簡単な例をいくつか示します。

```
const v1 = xs.time('12:30:45-05:00');
const v2 = xs.time('01:10:25-02:00');

const r = {
"v1 lt v2" : v1.lt(v2),
"v1 le v2" : v1.le(v2),
"v1 gt v2" : v1.gt(v2),
"v1 ge v2" : v1.ge(v2),
"v1 eq v2" : v1.eq(v2),
"v1 ne v2" : v1.ne(v2)
};
r;

/*
returns:
{"v1 lt v2":false,
 "v1 le v2":false,
 "v1 gt v2":true,
 "v1 ge v2":true,
 "v1 eq v2":false,
 "v1 ne v2":true}
*/
```

2.10.3.6 xs.gYearMonth の比較メソッド

xs.gYearMonth オブジェクトでは、次の比較メソッドを使用できます。

メソッド	説明
eq(xs.gYearMonth)	xs.gYearMonth 値に対する等値比較。Boolean を返します。
ne(xs.gYearMonth)	xs.gYearMonth 値に対する不等比較。Boolean を返します。

次に、これらのメソッドを使用した簡単な例をいくつか示します。

```
const v1 = xs.gYearMonth('2013-08');
const v2 = xs.gYearMonth('2012-04');

const r = {
"v1 eq v2" : v1.eq(v2),
"v1 ne v2" : v1.ne(v2)
};
r;

/*
returns:
{"v1 eq v2":false,
 "v1 ne v2":true}
*/
```

2.10.3.7 xs.gYear の比較メソッド

xs.gYear オブジェクトでは、次の比較メソッドを使用できます。

メソッド	説明
eq(xs.gYear)	xs.gYear 値に対する等値比較。Boolean を返します。
ne(xs.gYear)	xs.gYear 値に対する不等比較。Boolean を返します。

次に、これらのメソッドを使用した簡単な例をいくつか示します。

```
const v1 = xs.gYear('2013');
const v2 = xs.gYear('2012');

const r = {
"v1 eq v2" : v1.eq(v2),
"v1 ne v2" : v1.ne(v2)
};
r;

/*
returns:
{"v1 eq v2":false,
 "v1 ne v2":true}
*/
```

2.10.3.8 xs.gMonthDay の比較メソッド

`xs.gMonthDay` オブジェクトでは、次の比較メソッドを使用できます。

メソッド	説明
`eq(xs.xs.gMonthDay)`	`xs.xs.gMonthDay` 値に対する等値比較。`Boolean` を返します。
`ne(xs.xs.gMonthDay)`	`xs.xs.gMonthDay` 値に対する不等比較。`Boolean` を返します。

次に、これらのメソッドを使用した簡単な例をいくつか示します。

```
const v1 = xs.gMonthDay('--08-20');
const v2 = xs.gMonthDay('--04-14');

const r = {
"v1 eq v2" : v1.eq(v2),
"v1 ne v2" : v1.ne(v2)
};
r;

/*
returns:
{"v1 eq v2":false,
 "v1 ne v2":true}
*/
```

2.10.3.9 xs.gMonth の比較メソッド

xs.gMonth オブジェクトでは、次の比較メソッドを使用できます。

メソッド	説明
eq(xs.gMonth)	xs.gMonth 値に対する等値比較。Boolean を返します。
ne(xs.gMonth)	xs.gMonth 値に対する不等比較。Boolean を返します。

次に、これらのメソッドを使用した簡単な例をいくつか示します。

```
const v1 = xs.gMonth('--08');
const v2 = xs.gMonth('--04');

const r = {
"v1 eq v2" : v1.eq(v2),
"v1 ne v2" : v1.ne(v2)
};
r;

/*
returns:
{"v1 eq v2":false,
 "v1 ne v2":true}
*/
```

2.10.3.10xs.gDay の比較メソッド

xs.gDay オブジェクトでは、次の比較メソッドを使用できます。

メソッド	説明
eq(xs.gDay)	xs.gDay 値に対する等値比較。Boolean を返します。
ne(xs.gDay)	xs.gDay 値に対する不等比較。Boolean を返します。

次に、これらのメソッドを使用した簡単な例をいくつか示します。

```
const v1 = xs.gDay('---08');
const v2 = xs.gDay('---04');

const r = {
"v1 eq v2" : v1.eq(v2),
"v1 ne v2" : v1.ne(v2)
};
r;

/*
returns:
{"v1 eq v2":false,
 "v1 ne v2":true}
*/
```

2.11 MarkLogic の JavaScript 関数

JavaScript では多数の MarkLogic ビルトイン関数を使用できます。一般的に、XQuery で使用可能なほとんどの関数には、それに対応する JavaScript が用意されています。JavaScript で使用可能な MarkLogic 関数の詳細については、「JavaScript 関数とコンストラクタ」（50 ページ）を参照してください。

3.0 JavaScript 関数とコンストラクタ

この章では、MarkLogic のビルトイン関数の使用方法、および JavaScript プログラム内に XQuery ライブラリをインポートして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。

3.1 ビルトイン JavaScript 関数

MarkLogic には多くのビルトイン関数が用意されており、プログラムによって MarkLogic の機能にすばやく簡単にアクセスできます。ビルトイン関数は JavaScript 関数としてそのまま使用でき、インポートしたりライブラリを用意したりする必要はありません（「ビルトイン」と呼ばれるのはそのためです）。関数の詳細については、サーバーサイド JavaScript API のドキュメントを参照してください。

関数は次のグローバルオブジェクトを介して使用できます。

- `cts.`
- `fn.`
- `math.`
- `rdf.`
- `sc.`
- `sem.`
- `spell.`
- `sql.`
- `xdmp.`

例えば、現在の時刻を取得するには、次の関数を呼び出します。

```
fn.currentDateTime();
```

3.2 グローバルオブジェクトの一部である関数

（名前空間プレフィックスなしの）グローバルな JavaScript オブジェクトの一部である MarkLogic 固有の関数があります。このセクションでは、次のグローバル関数を取り上げます。

- declareUpdate 関数
- require 関数

3.2.1 declareUpdate 関数

ドキュメントの更新を実行するには、トランザクションを更新として宣言する必要があります。ステートメントの先頭で `declareUpdate` が呼び出されていない場合、そのステートメントはクエリとして実行されます。`declareUpdate` 関数のシンタックスは、次のとおりです（`Global-Object.declareUpdate` を参照）。

```
declareUpdate(Object options)
```

`options` は、次のようなオプションの引数です。

```
{explicitCommit: true/false}
```

`options` 引数を省略した場合、または `explicitCommit` プロパティが `false` に設定されている場合、トランザクションは自動的にコミットされます。`explicitCommit` プロパティが `true` に設定されている場合はマルチステートメントトランザクションが開始され、トランザクションを完了するには `xdmp.commit` または `xdmp.rollback` が必要です。

トランザクションの詳細については、『*Application Developer's Guide*』の「Understanding Transactions in MarkLogic Server」を参照してください。

次に、JavaScript での更新トランザクションの例を示します。

```
declareUpdate();
const myDoc = {"foo":"bar"};
xdmp.documentInsert("/myDoc.json", myDoc);
// creates the /myDoc.json document
```

次の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として実行されます（ただし、このトランザクションにはステートメントが 1 つしかありませんが）。

```
declareUpdate({explicitCommit: true});
const myDoc = {"foo":"bar"};
xdmp.documentInsert("/myDoc.json", myDoc);
xdmp.commit();
// creates the /myDoc.json document
```

3.2.2 require 関数

`require` 関数（`Global-Object.require` を参照）は、グローバルオブジェクトで使用でき、JavaScript プログラムにライブラリをインポートできます。詳細については、「require 関数」（52 ページ）を参照してください。

3.3 JavaScript で XQuery の関数および変数を使用する

XQuery ライブラリモジュールをサーバーサイド JavaScript プログラムにインポートすると、それらの関数または変数を JavaScript から呼び出せます。XQuery ライブラリのインポートは、既存の XQuery コードを JavaScript プログラムで使いたい場合や、XQuery に適したタスクを JavaScript プログラムから実行する必要がある場合に役立ちます。このセクションでは、XQuery モジュールを JavaScript プログラムで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。

- require 関数
- XQuery モジュールを JavaScript プログラムにインポートする

3.3.1 require 関数

XQuery ライブラリまたは JavaScript ライブラリは、次の JavaScript 関数を使用してインポートできます。

```
require(String location)
```

`location` は、JavaScript ファイルまたは XQuery ファイルへのパスです。パスの拡張子は、簡素化するために省略できます。パスは、『*Application Developer's Guide*』の「Rules for Resolving Import, Invoke, and Spawn Paths」で定義されている XQuery のルールと同じルールに従っています。

一般的に、`require` 関数は JavaScript プログラムの最初の行に配置されます。また、プログラムでは 0 個以上の `require` 関数を使用できます。XQuery ライブラリをインポートするときには、一般的な慣習として、名前空間のプレフィックスの場合と同様に JavaScript 変数に名前を付けます。例えば、search API ライブラリをインポートする場合、require ステートメントは次のようになります。

```
const search =
require("/MarkLogic/appservices/search/search.xqy");
search.search("hello");
// returns a search response for documents matching "hello"
```

3.3.2 XQuery モジュールを JavaScript プログラムにインポートする

MarkLogic には、XQuery ライブラリモジュールの豊富なセットが用意されているため、さまざまな処理を実行するプログラムを簡単に作成できます。例えば、検索やアラートのアプリケーションを作成したり、スペリング修正機能を追加できます。また、豊富な XQuery ライブラリのセットを独自に作成している場合があることでしょう。機能によっては、XQuery では簡単でも、JavaScript では簡単に記述できない場合があります（XPath ステートメントなど）。

`require` 関数を使用すると、このような XQuery ライブラリを MarkLogic のサーバーサイド JavaScript プログラムで利用できます。このセクションでは、XQuery 環境と JavaScript 環境間の名前と型のマッピングについて説明します。以下のように構成されています。

- XQuery の関数および変数名を JavaScript にマッピングする
- XQuery と JavaScript の間で型をマッピングする

3.3.2.1 XQuery の関数および変数名を JavaScript にマッピングする

XQuery では一般的に、関数名や変数名にハイフン（-）を含めますが、JavaScript ではハイフン（-）は減算演算子なので、このような名前に互換性はありません。JavaScript では一般的に、関数や変数に名前を付けるときにキャメルケースを使用します。言語間のこのような違いに対処するため、`require` 関数で JavaScript プログラムにインポートした XQuery 関数や変数にアクセスする場合は、次のルールに従います。

- XQuery では名前空間のプレフィックスの後にコロン（:）を記述し、その後に関数のローカル名を続けますが、JavaScript ではオブジェクトと同じように、名前空間のプレフィックスの後にピリオド（.）を付けます。
- ハイフン（-）が含まれる関数名や変数名は、キャメルケースの名前に変換されます。例えば、`my-function` という名前の XQuery の関数は、JavaScript では `myFunction` という名前で使用できます。
- 上記のルールでは曖昧になるようなケースでは（ほとんどありませんが）、XQuery 関数または変数のリテラル名を使用して、角括弧表記で関数にアクセスすることもできます。例えば、XQuery 関数名 `hello:my-world`（`hello` プレフィックスとローカル名 `my-world` にバインドされている関数）は、次の JavaScript 表記でアクセスできます。`hello["my-world"]()`。

これらのルールにより、公開されている任意の XQuery 関数または変数に JavaScript プログラムからアクセスできます。

3.3.2.2 XQuery と JavaScript の間で型をマッピングする

JavaScript の型指定ルールは XQuery よりも緩やかで、型の数は XQuery よりも少ないです。MarkLogic では、XQuery から JavaScript に自動的に型がマッピングされます。次の表は、XQuery の型が JavaScript の型にどのようにマッピングされるのかを示したものです。

XQuery の型	JavaScript の型	注
`xs:boolean`	`Boolean`	
`xs:integer`	`Integer`	
`xs:double`	`Number`	
`xs:float`	`Number`	
`xs:decimal`	`Number`	値が 9007199254740992 より大きい場合、またはスケールが 0 より小さい場合、値は String になります。
`json:array`	`Array`	
`json:object`	`Object`	
`map:map`	`Object`	
`xs:date`	`Date`	より細かい精度が保持されます。
`xs:dateTime`	`Date`	より細かい精度が保持されます。
`xs:time`	`String`	
`empty-sequence()`	`null`	
`item()`	`String`	
`xs:anyURI`	`String`	
`node()`	`Node`	
`node()*`	`ValueIterator`	

3.4 JavaScript モジュールを JavaScript プログラムにインポートする

`require` 関数を使用すると、サーバーサイド JavaScript ライブラリをサーバーサイド JavaScript プログラムにインポートできます。`require` 関数を使用して JavaScript ライブラリをインポートすると、`require` 関数から返される `exports` オブジェクトを使用して、JavaScript ライブラリ内のすべての関数とグローバル変数を使用できます。以下に例を示します。

```
const circle = require("circle.js");
circle.area(4);
// evaluates the area function from circle.js,
// passing 4 as its parameter
```

`.js` または `.sjs` のいずれかの拡張子を持つ JavaScript ライブラリ（対応する MIME タイプ `application/javascript` および `application/vnd.marklogic-javascript` を持つもの）をインポートできます。ただし、`.js` ファイル拡張子を持つサーバーサイド JavaScript モジュールはアプリケーションサーバーから直接は提供できません。直接提供されるモジュールには、`.sjs` 拡張子が必要です。require 関数の詳細については、「require 関数」（52 ページ）を参照してください。

3.5 JavaScript で使用できるその他の MarkLogic オブジェクト

JavaScript では多数の MarkLogic オブジェクトを使用できます。そのため、ノードやドキュメントを簡単に操作できます。このようなオブジェクトの詳細については、「MarkLogic JavaScript のオブジェクト API」（10 ページ）を参照してください。

3.6 amp（強化）および module.amp 関数

JavaScript では、強化された（「amped」）関数を作成できます。強化された関数は、amp（強化）対象のロールに基づいて強化された権限によって評価します。amp では、modules データベースまたは *<marklogic-dir>/Modules* ディレクトリに含まれる関数が必要であり、さらに security データベースでの設定（amp）が必要です。amp の詳細については、『*Security Guide*』の「Temporarily Increasing Privileges with Amps」を参照してください。このセクションでは、JavaScript の amp について説明します。以下のように構成されています。

- module.amp 関数
- 単純な JavaScript の amp の例

3.6.1 module.amp 関数

`module.amp` 関数は、次のシグネチャを持ちます。

```
module.amp(Function namedFunction)
```

この関数は、modules データベースまたは *<marklogic-dir>/Modules* ディレクトリにある JavaScript モジュール内の exports ステートメントで使用する必要があります。exports ステートメントの例は以下のとおりです。

```
exports.ampedFunctionName = module.amp(ampedFunctionName);
```

`ampedFunctionName` は、強化する対象となるライブラリ内の関数の名前です。

3.6.2 単純な JavaScript の amp の例

次のステートメントでは、amp を目的とした JavaScript モジュールを作成し、モジュールを参照する amp を作成してから、別のモジュールの関数を呼び出します。結果として、この関数には権限が必要なのにもかかわらず、権限のないユーザーでも実行できてしまいます。

1. modules データベースまたは *<marklogic-dir>/Modules* ディレクトリ内にファイルとして amp モジュールを作成します。例えば、UNIX システムでは、`/opt/MarkLogic/test-amp.sjs` として次のファイルを作成します（そのファイルが MarkLogic で読み取り可能である必要があります）。

```
// This is a simple amp module
// It requires creating an amp to the URI of this sjs file
with the
// function name.

function ampedInsert() {
xdmp.documentInsert("/amped.json", {prop:"this was produced
by an \n\
  amped function"}, [xdmp.permission("qconsole-user",
"read"),
                     xdmp.permission("qconsole-user",
"update")]);
};

exports.ampedInsert = module.amp(ampedInsert);
```

2. この関数をポイントする amp を作成します。例えば、Admin 画面で、[Security] > [Amps] を選択し、[Create] タブを選択します。次に [localname] で amp 関数の名前（`ampedInsert`）を入力し、名前空間は空白のままにしておきます。さらに JavaScript モジュールへのパス（例えば、`/test-amp.sjs`）を入力し、データベースのファイルシステムを選択して、最後に関数を強化する対象となるロールを割り当てます。この例では、`admin` ロールを選択します。

3. 次に、アプリケーションサーバーのルートから、次のコンテンツを持つ JavaScript モジュールを作成します。

```
declareUpdate();
const mod = require("/test-amp.sjs");
mod.ampedInsert();
```

4. 権限のないユーザーとして、上記の手順で作成したプログラムを実行します。例えば、プログラムが /space/appserver/test.sjs として保存されていて、ポート 8005 上のアプリケーションサーバールートが /space/appserver である場合は、http://localhost:8005/test.sjs にアクセスします。

   Admin 画面でロールを何も付与せずにユーザーを作成することで、権限のないユーザーを作成できます。

ここで Query Console に移動すると、/amped.json というドキュメントが作成されていることを確認できます。

この例は、次の 2 つの点で現実の例よりも単純化されています。1 つ目は、強化されたモジュールを Modules ディレクトリの下に配置している点です。ベストプラクティスとしては、modules データベースに強化された関数を格納します。modules データベースを使用する際は、ドキュメントに対して必要なパーミッションを持つモジュールをデータベースに挿入する必要があることに注意してください。2 つ目は、この例では admin ロールに対して強化を実行している点です。実際のベストプラクティスとしては、ユーザーが関数の実行に最小限必要な権限のロールを作成します。

3.7 JavaScript の型コンストラクタ

JavaScript 環境には、MarkLogic 固有のコンストラクタが追加されています。これを使用して、JavaScript で XQuery の型を作成できます。このコンストラクタの名前は対応する XQuery のコンストラクタと同じですが、コロン（:）の代わりにドット（.）で名前空間とコンストラクタ名が区切られています。マイナス記号（-）が含まれるコンストラクタの場合は、ローカル名を角括弧で囲んで呼び出す必要があります（例えば、cts['complex-polygon']）。

各コンストラクタを使用するには、作成するオブジェクトをそのコンストラクタに渡します。次の例は、xs.QName コンストラクタの使用方法を示したものです。

```
fn.namespaceUriFromQName(xs.QName("xdmp:foo"))
=> http://marklogic.com/xdmp
   (because the xdmp namespace is always in scope)
```

JavaScript から呼び出すことができる MarkLogic コンストラクタは、次のリストに示すとおりです。

```
xs.simpleDerivationSet
xs.gYear
```

```
xs.public
xs.language
xs.short
xs.decimal
xs.reducedDerivationControl
xs.gYearMonth
xs.date
xs.double
xs.nonPositiveInteger
xs.positiveInteger
xs.blockSet
xs.normalizedString
xs.namespaceList
xs.gMonth
xs.integer
xs.int
xs.anyAtomicType
xs.gMonthDay
xs.NCName
xs.unsignedShort
xs.derivationControl
xs.IDREFS
xs.derivationSet
xs.token
xs.ID
xs.nonNegativeInteger
xs.anyURI
xs.NMTOKEN
xs.allNNI
xs.QName
xs.base64Binary
xs.boolean
xs.long
xs.Name
xs.yearMonthDuration
xs.duration
xs.NMTOKENS
xs.dayTimeDuration
xs.negativeInteger
xs.NOTATION
xs.unsignedInt
xs.unsignedLong
xs.untypedAtomic
xs.formChoice
xs.dateTime
xs.float
```

```
xs.ENTITY
xs.byte
xs.time
xs.unsignedByte
xs.ENTITIES
xs.string
xs.IDREF
xs.hexBinary
xs.gDay
cts.andNotQuery
cts.andQuery
cts.boostQuery
cts.box
cts.circle
cts.collectionQuery
cts.collectionReference
cts.complexPolygon
cts.confidenceOrder
cts.directoryQuery
cts.documentFragmentQuery
cts.documentOrder
cts.documentQuery
cts.elementAttributePairGeospatialQuery
cts.elementAttributeRangeQuery
cts.elementAttributeReference
cts.elementAttributeValueQuery
cts.elementAttributeWordQuery
cts.elementChildGeospatialQuery
cts.elementGeospatialQuery
cts.elementPairGeospatialQuery
cts.elementQuery
cts.elementRangeQuery
cts.elementReference
cts.elementValueQuery
cts.elementWordQuery
cts.falseQuery
cts.fieldRangeQuery
cts.fieldReference
cts.fieldValueQuery
cts.fieldWordQuery
cts.fitnessOrder
cts.geospatialElementAttributePairReference
cts.geospatialElementChildReference
cts.geospatialElementPairReference
cts.geospatialElementReference
cts.geospatialJsonPropertyChildReference
```

```
cts.geospatialJsonPropertyPairReference
cts.geospatialJsonPropertyReference
cts.geospatialPathReference
cts.indexOrder
cts.jsonPropertyChildGeospatialQuery
cts.jsonPropertyGeospatialQuery
cts.jsonPropertyPairGeospatialQuery
cts.jsonPropertyRangeQuery
cts.jsonPropertyReference
cts.jsonPropertyScopeQuery
cts.jsonPropertyValueQuery
cts.jsonPropertyWordQuery
cts.linestring
cts.locksFragmentQuery
cts.longLatPoint
cts.lsqtQuery
cts.nearQuery
cts.notInQuery
cts.notQuery
cts.order
cts.orQuery
cts.pathGeospatialQuery
cts.pathRangeQuery
cts.pathReference
cts.period
cts.periodCompareQuery
cts.periodRangeQuery
cts.point
cts.polygon
cts.propertiesFragmentQuery
cts.punctuation
cts.qualityOrder
cts.query
cts.reference
cts.region
cts.registeredQuery
cts.reverseQuery
cts.scoreOrder
cts.searchOption
cts.similarQuery
cts.space
cts.special
cts.termQuery
cts.token
cts.tripleRangeQuery
cts.trueQuery
```

```
cts.unordered
cts.uriReference
cts.word
cts.wordQuery
dir.type
math.coefficients
sem.iri
sem.variable
sem.blank
```