MarkLogic サーバー

JavaScript リファレンスガイド

MarkLogic 8
2015 年 2 月

最終更新：8.0-1、2015 年 2 月

目次

JavaScript リファレンスガイド

1.0 MarkLogic のサーバーサイド JavaScript

MarkLogic 8 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。

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 が埋め込んでいる V8 のバージョンは 3.24 で、次のフラグが有効です。

- `--harmony-iteration`

このバージョンの V8 では、新しい EcmaScript 6 の機能が複数提供されており、その 1 つは `for ... of` ループを実装するために使用されています。このループを `ValueIterator` を返す API で使用すると、関数で大きな結果セットを返し、結果を効率的にストリーム処理できます。詳細については、「ValueIterator」（23 ページ）を参照してください。

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。

```
var results = new Array();
var cdt = fn.currentDateTime();
results.push(cdt);
var 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。

```
var 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 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。

1.6 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 の HTTP アプリケーションサーバーでは、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。

1.7 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。

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
- ValueIterator
- JavaScript エラー API
- 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」（11 ページ）を参照してください。

次に、firstChild 関数と 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。

```
// assume a document created as follows:
declareUpdate();
xdmp.documentInsert("/my/doc.xml", xdmp.unquote(
```

```
  '<bar:foo xmlns:bar="bar"><bar:hello><bar:goodbye \n\
     attr="attr value">bye</bar:goodbye>\n\
  </bar:hello>\n\
</bar:foo>').next().value);

// Given the above document:
var node = cts.doc("/my/doc.xml").root.firstChild;
node.xpath("./bar:goodbye/@attr", {"bar":"bar"});
=>
"attr value" (as an attribute node)
```

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 およびサブタイプ

- 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」（8 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。
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。

プロパティ	説明
`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。
`isSameNodes( 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）。
`isEqualNodes( 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。
`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、XML ノードの Node オブジェクトからのプロパティに加え、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
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。
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。

`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。
	`var x = new NodeBuilder();`
	`var 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, String URI, Function content)`	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。

関数 / プロパティ	説明
`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。
`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。

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

`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）。

プロパティ / 関数	説明
`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 およびサブタイプ

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サブタイプを持ちます。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。
`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。

関数 / プロパティ	説明
`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 amespaceURI,`
`  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。
`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。これにより、XQuery 関数で使用される型として使用し、それらに対して `toObject()` を呼び出すことで、最も近い JavaScript のネイティブの型に変換することもできます。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。

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 ValueIterator

反復処理される結果セットを返すビルトイン JavaScript 関数は、ValueIterator として返されます。ValueIterator は、JavaScript の Iterator の拡張機能であり、Iterator のすべてのプロパティに加え、追加のプロパティを持ちます。次の表は、ValueIterator API をまとめたものです。これらのプロパティおよび関数は、ValueIterator として型指定されたオブジェクトに対して呼び出すことができます。

プロパティ / 関数	説明
count	イテレータ内の残りの値の数を返します。
clone()	イテレータのクローンを返します。これは、データベース内のドキュメントへの参照を保持し続けることがないようにノードのコピーを作成する必要がある場合に役に立ちます。
next()	イテレータ内の次の項目を返します。この関数には 2 つのプロパティがあります。value と done です。また、next() を対象とする discard という関数もあります。この関数では、基になる参照先の項目が展開ツリーから削除されます。
toArray()	イテレータ内に残っている値の値項目の配列を返します。それ以上値がない場合は、空の配列を返します。

次の例に示すように、ValueIterator として返される値を反復処理する while ループを作成できます。この例では、URI を条件とした検索で最初の 2 つのドキュメントが返されます。

```
// gather up results into an array
var results = new Array();
// first two search results
// empty cts.andQuery matches every document
var res = fn.subsequence(cts.search(cts.andQuery([])), 1, 2);
// loop thru ValueIterator and push result and URI into results array
while (true) {
  var doc = res.next();
  if (doc.done == true ) { break; }
  else { results.push(doc.value);
         results.push(xdmp.nodeUri(doc.value));
       }
};
// print out the results
results;
```

2.6 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。

2.6.1 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」（25 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。

2.6.2 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。
`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）。

2.6.3 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
```

2.7 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」（27 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 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以下のとおりです。

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

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。

```
var 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`	

XQuery の型	JavaScript の型	注
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。

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

require 関数の詳細については、「require 関数」（29 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」（8 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、『*Understanding and Using 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();
var 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.scalar-type
cts.option
cts.special
cts.localname
cts.word
cts.key
cts.coordinate-system
cts.qname
cts.search-option
cts.path-expression
cts.polygon
cts.operator
cts.collation
cts.box
cts.distance
cts.complex-polygon
cts.point
cts.weight
```

```
cts.linestring
cts.text
cts.token
cts.attribute-localname
cts.field-name
cts.semantic-operator
cts.json-property-name
cts.namespace-uri
cts.nodes
cts.iri
cts.uri
cts.id
cts.field
cts.space
cts.circle
cts.long-lat-point
cts.depth
cts.punctuation
cts.region
sec.privilege-name
sec.compartment
sec.ldap-password
sec.uri
sec.kind
sec.password
sec.external-name
sec.id
sec.timeout
sec.ldap-server-uri
sec.aws-access-key
sec.authentication
sec.aws-secret-key
sec.role-name
sec.description
sec.ldap-default-user
sec.realm
sec.external-security-name
sec.capability
sec.authorization
sec.local-name
sec.namespace
sec.ldap-base
sec.user-name
sec.action
sec.document-uri
sec.ldap-attribute
error.xquery-version
dir.type
dav.responsedescription
dav.status
dav.getcontenttype
dav.getcontentlanguage
dav.timeout
dav.getcontentlength
```

```
dav.depth
dav.getetag
dav.creationdate
dav.displayname
dav.getlastmodified
lock.timestamp
lock.lock-scope
lock.lock-token
lock.timeout
lock.lock-type
lock.depth
prop.last-modified
prop.directory
math.coefficients
sem.iri
sem.variable
sem.blank
spell.word-type
spell.distance-threshold
spell.maximum
```