System Programming Guide

バージョン 3.1

Printed on 24 June, 2005

著作権情報

法律上の特記事項

このマニュアルの情報は Xpriori, LLC により提供されています。Xpriori, LLC の知る限りにおいて、このマニュアルには情報処理テクノロジの現在の状態に関する情報が収められていますが、このマニュアルおよびその記載情報は現状のまま提供されいかなる保証もされません。Xpriori, LLC は、このマニュアルに記載されている情報および製品について、商品性、特定目的適合性、権利非侵害の保証を含め、かつ、それに限定されない一切の保証の責任を、明示的にも暗示的にも一切負うものではありません。このマニュアルの記載内容ならびに特定の環境に対して提案される応用は、詳細な技術的調査なしに信頼できることを保証するものではありません。このマニュアルの記載情報および製品に関する記述の使用またはそれらへの依存に起因する一切の損害について、Xpriori, LLC はいかなる責任も負いません。製品および技術情報は、予告なく変更されることがあります。

特許

Xpriori, LLC の技術は以下の特許権により保護されています。

米国特許番号 5,742,611（1998 年 4 月 21 日）- Client Server Network and Method of Operation - Christopher Lockton Brandin

米国特許番号 5,942,002（1999 年 8 月 24 日）- Method and Apparatus for Generating a Transform - Christopher Lockton Brandin

米国特許番号 6,157,617（2000 年 12 月 5 日）- Method and System of Network Packet Accounting - Christopher Lockton Brandin

米国特許番号 6,167,400（2000 年 12 月 26 日）- Method of Performing a Sliding Window Search - Christopher Lockton Brandin

その他の米国特許および国際特許が出願中です。

商標

NeoCore™、NeoCore™ XML Information Management System（XMS）、Pattern-Based Associative Processing、および Digital Pattern Processing は Xpriori, LLC の米国およびその他の国における商標または登録商標です。他のすべての商標は、それぞれ各社の商標です。

著作権

目次

Xpriori NeoCore XML Information Management System – 概要

Xpriori NeoCore™ **X**ML Information **M**anagement **S**ystem（この資料では **XMS** と呼びます）は、情報管理のための高速で非常に柔軟なツールです。

XMS は Extensible Markup Language（XML）データとそのコンテキストの格納と検索を行う目的で開発されました。

この『***System Programming Guide***』は、NeoCore XMS の NeoCore XMS でのデータの効率的な扱い方、アーキテクチャ、XML 操作のための XPath の使用、基本ロケーションパス、XMS トランザクションとコマンド、XSLT、および Java、C++、Component Object Model（COM）ラッパー、HTTP 用の API に関する情報を提供します。

新機能

- WebDAV - WebDAV は、World Wide Web を読み取り専用の媒体から読み取り/書き込みが可能な媒体へと変換するための HTTP に対する一連のプロトコル拡張機能です。WebDAV は、コラボレーションの手段として幅広く採用されており、すでに Microsoft、Adobe、Altova、EMC、Oracle、IBM などの製品でサポートされています。XMS 用の Xpriori WebDAV インターフェースは、DAV プロトコルを実装することによって、XMS をバックエンドとして使用しながら、DAV サーバーに格納した XML に対するハイパフォーマンスアクセスを実現しています。
- インポートとエクスポートの拡張 - XMS v3.1 では、インポートの実行時にもデータベースファイルの自動拡張機能が用意されているので、データベースのサイズを事前に構成する必要がありません。エクスポートも拡張され、より大きなファイルを生成できる設計になったので、エクスポートしたファイルの管理がシンプルになっています。
- 名前空間 - XMS では、名前空間を指定したタグと名前空間を指定しないタグを組み合わせたクエリーをサポートするようになりました。
- パフォーマンス - XMS（Windows 版）と XMS（Linux 版）では、ディスクベース（Windowing）モードでの実行時に、メモリー管理の拡張機能により更新処理のパフォーマンスが改善されています。

お客様のコメントをお送りください

このガイドについてのコメントやご提案については、Mitsui NeoCore Center support@neocore.jp 宛メールにてお送りください。

対象読者

このマニュアルの対象読者は、NeoCore XMS を使用して Java または C++のプログラミングを行い、Windows や Solaris のオペレーティングシステムに関する理解を持つ開発者です。

関連資料

NeoCore XMS のインストール、構成、使用、および保守についての情報は、『*NeoCore XMS System Administration Guide*』を参照してください。

表記規則

- このガイドで使用する表記規則は、以下のとおりです。
- 書籍、資料、セクションなどの表題は*斜体*で示します。
- コードサンプルや他のリテラル文字列は `courier` フォントで示します。
- **ファイル名、コマンド、タブ、メニュー項目、ウィンドウラベルなどは太字**で示します。

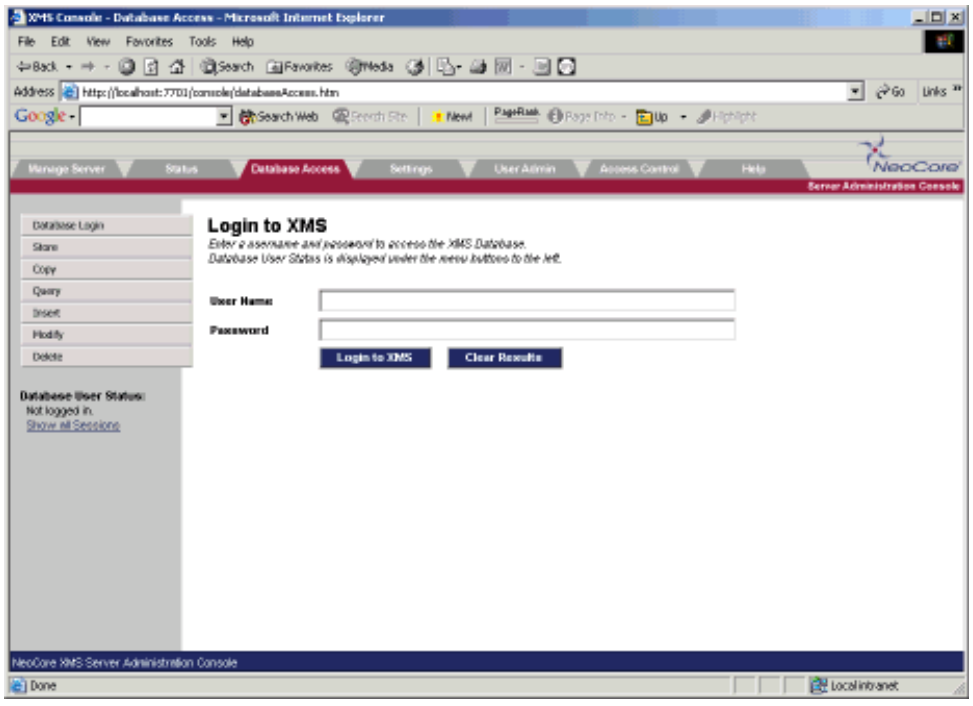
NeoCore XMS へのアクセス

インストールが完了した時点で、ユーザーはコンソール経由、または ***Java*** 、***C*** ++、***COM*** 、***HTTP*** の各 API 経由で NeoCore XMS にアクセスできます。

NeoCore XMS を開始する前に

- ソフトウェアが正しくインストールされている必要があります
- データベースが作成されている必要があります
- データベースの構成ファイルが/NeoCore/neoxml/config に存在する必要があります
- Windows: データベースは、必要なら Windowing を使用して、仮想メモリーに収まっている必要があります。

コンソール経由の XMS アクセス

デスクトップ上の［**Launch Console**］アイコンをダブルクリックするか、［**スタート**］>［**プログラム**］>［**NeoCore XMS**］>［**Launch Console**］を選択します。 NeoServer 管理コンソールにログインするには、インストール時に定義した管理者アカウントでログインする必要があります。	
XMS にログインするには、まず［**Database Access**］タブでログインする必要があります。	

NeoServer の開始と停止

NeoServer の開始 - Windows

注:ログオンする前に、NeoServer を開始している必要があります。

推奨方法:

- C:¥NeoCore¥neoxml¥bin¥NeoServerCmd.exe start
- ［**スタート**］>［**プログラム**］>［**NeoCore XMS**］>［**Start NeoServer**］

別の開始方法:

- ［**スタート**］>［**設定**］>［**コントロールパネル**］>［**管理ツール**］>［**サービス**］を選択します。

NeoServerを選択します。

右クリックして［**開始**］を選択します。

注:起動がタイムアウト時間を超えると Windows ではエラーになりますが、このエラーは無視してかまいません。

注意:コマンドラインコマンドの **net start** や **net stop** は使用しないでください。ロールバックとインデックス再構築が必要になる場合があります。

NeoServer の開始 - Solaris 2.8/2.9

Solaris 2.8/2.9 で NeoServer を開始するための望ましい方法は、/etc/init.d/neocore スクリプトを使用することです。このファイルは、NeoServer を正しく稼働できるようにインストール時に変更されます。

root としてログインし、以下のように入力します。

```
/etc/init.d/neocore start
```

別の開始方法:

```
NeoServer [-d][-r <RootDir>]
例:NeoServer -d -r /usr/local/NeoCore
```

注:NeoServer 実行可能ファイルに rootdir 引数を渡さないと、デフォルトで/opt/NeoCore が使用されます。このデフォルトディレクトリ（またはその log ディレクトリや config ディレクトリ）が存在しない場合は、NeoServer を開始できません。NeoServer は root でも neoadmin でも実行可能です。setuid ビットが設定されるので、常にユーザー「neoadmin」として実行されます。コマンドラインから NeoServer を実行する正しい構文は NeoServer [-d][-r <RootDir>]です。それで、XMS が/usr/local/NeoCore にインストールされている場合は、「`NeoServer -d -r /usr/local/NeoCore`」を実行してください。-d オプションは、デーモンとして（端末セッションと結び付けずに）このコマンドを実行するためのオプションです。

NeoServer の停止 - Windows

推奨方法：

- `C:¥NeoCore¥neoxml¥bin¥NeoServerCmd.exe stop`
- ［**スタート**］>［**プログラム**］>［**NeoCore XMS**］>［**Stop NeoServer**］

あるいは、以下のようにします。

1. ［Server Administration Console］で［Manage Server］タブを選択します。
2. ［**Shut Down**］ボタンを選択します。
3. ［**Shutdown XMS Server**］ボタンを選択します。

注意:コマンドラインコマンドの net start や net stop は使用しないでください。ロールバックとインデックス再構築が必要になる場合があります。

NeoServer の停止 - Solaris 2.8/2.9

Solaris 2.8/2.9 で NeoServer を停止するための望ましい方法は、`/etc/init.d/neocore` スクリプトを使用することです。

root としてログインし、以下のように入力します。

```
/etc/init.d/neocore stop
```

あるいは、以下のようにします。

1. ［Server Administration Console］で［Manage Server］タブを選択します。
2. ［**Shut Down**］ボタンを選択します。
3. ［**Shutdown XMS Server**］ボタンを選択します。

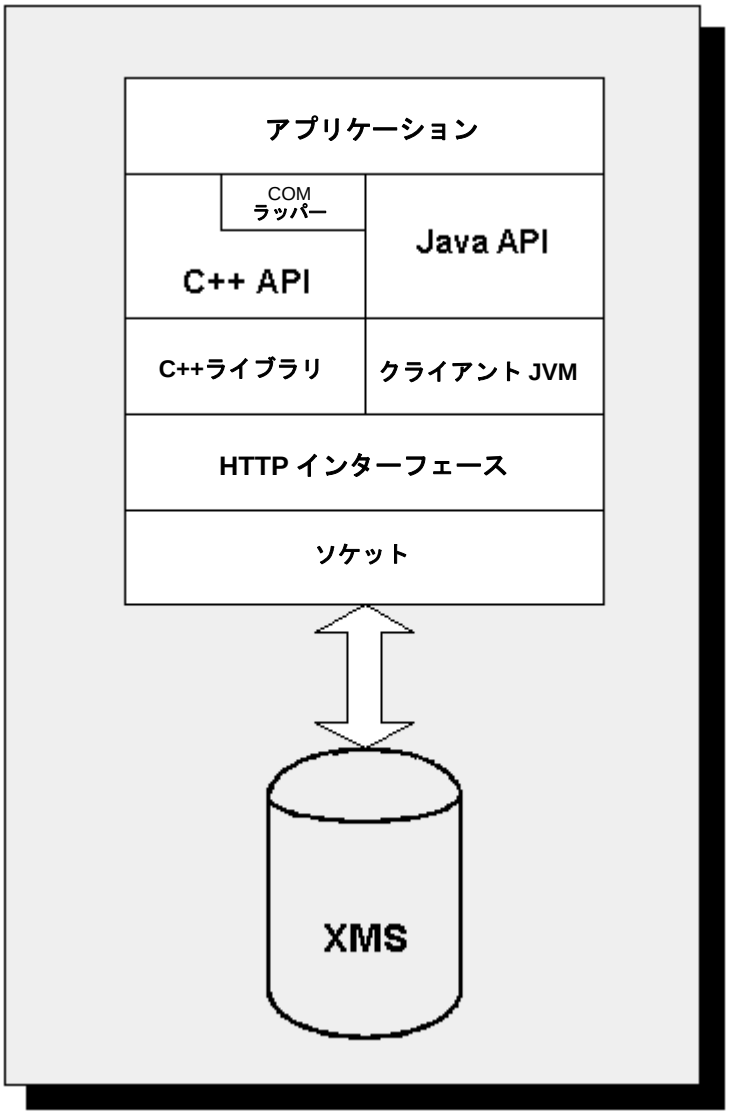
別の方法

```
kill -HUP [pid or %job-id]
```

注:「kill -9 [pid or %job-id]」は使用しないでください。停止時にサーバーがトランザクションを休止していたり、インデックス再構築を行っていたりする場合があります。

アーキテクチャ

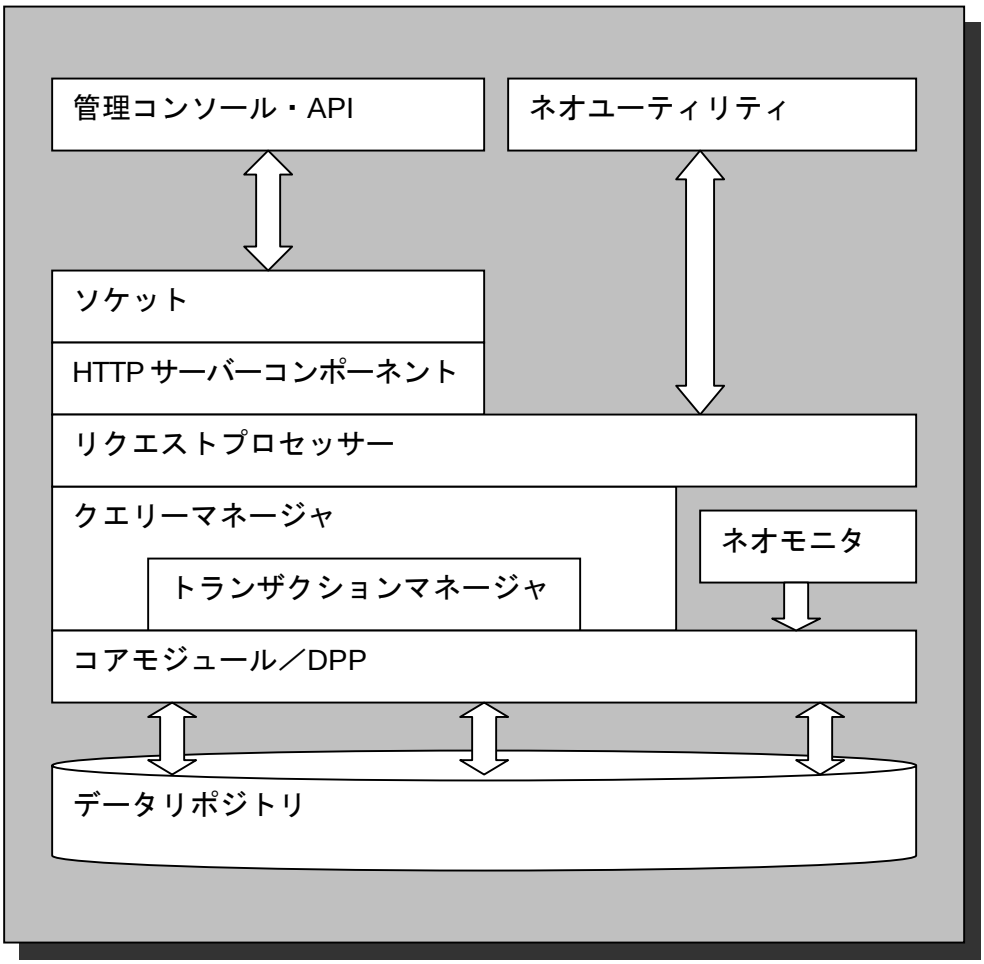
クライアント側の概要



NeoCore XMS のプロトコルレベルインターフェースは TCP 上の HTTP です。アプリケーションは、HTTP を直接生成するか、クライアント側の API によって HTTP を生成するかのいずれかになります。現時点では、Java と C++の API が用意されています。さらに、C++ API には、VB などの Microsoft のプログラミング環境をサポートするための COM ラッパーがあります。

サーバー側の概要

以下の図に、XMS サーバーのアーキテクチャの概要を示します。サーバープロセスは、サーバーの実行を制御し、サーバーコンポーネントマネージャを使用して実行用の正しいコンポーネントをロードします。現時点で、XMS は HTTP コンポーネントだけをサポートしています。HTTP サーバーコンポーネントは、HTTP コマンドを処理し、そのコマンドを適切なサブコンポーネントに送信します。



管理モジュールは、管理コマンド（統計やサーバー管理など）の処理を担当します。これらのコマンドは、データベースのトランザクション処理には影響しません。すべての API からアクセスできますが、管理ポート経由でアクセスする必要があります。

XMS モジュールは HTTP インターフェースからのコマンドを処理します。こうしたコマンドの大部分は、XMS 内で XML のクエリー、変更、格納を行うトランザクションコマンドです。通常のデータベースポート経由でも管理ポート経由でもアクセスできます。

XMS には、サーバー側の拡張処理を行うために、サーバー側の JVM が用意されています。現時点で、XSLT 変換エンジンはこの JVM で実行されます。他のアプリケーションを使用して、この組み込み環境でサーバー機能を拡張することも可能です。ただし、バージョン 2.7 から、この JVM へのインターフェースはエンドユーザーに公開されなくなっており、Xpriori, LLC がサーバー側の追加機能を展開するときに限って使用することになっています。将来のリリースでこのインターフェースが公開される可能性はあります。

クライアント要求を処理する XMS の層を XMS コマンドといいます。これには、通常のデータベースコマンドと管理コマンドが含まれていますが、サーバー側の拡張処理は含まれていません。その拡張処理は JNI 層が取り次ぎます。

サーバーコアは、クライアント要求を実際に実行し、内部データ構造を操作し、アクセス制御を行い、トランザクションの整合性を維持します。

お断り

XMS サーバーは、クライアントとサーバーの間の通信に HTTP を使用する独立したサーバープラットフォームです。このサーバーは、フル機能の HTTP サーバーとしては設計されていないので、サーバーとクライアントの間の通信をサポートする以外の標準的な Web サーバー機能（HTML 文書のサービスなど）は提供しません。将来もこの機能を提供する予定はありません。

エラーメッセージ

以下の表に、クライアント側の例外とそれぞれの例外番号を示します。2 つ目の表では、クライアント側の例外とサーバー側を対応付け、原因と対応策を一覧にしています。

クライアント側の例外	例外番号
NeoInvalidArgumentException	1
NeoAuthorizationFailedException	2
NeoCommunicationException	3
NeoInvalidSessionException	4
NeoMalformedURLException	5
NeoMalformedXMLException	6
NeoMalformedXPathException	7
NeoServerException	8
NeoTransactionDeadlockException	9
NeoTransactionException	10
NeoTransactionRollbackException	11
NeoTransactionSessionException	12
NeoTransactionTimeOutException	13
NeoDestroyedResultChunkException	14
NeoQueryResultExceededMaxException	15
NeoACException	16
NeoACGroupExistsException	17
NeoACRuleExistsException	18
NeoACUserExistsException	19
NeoACUsersExistInGroupException	20
NeoACGroupDoesNotExistException	21
NeoACAssociatedRulesExistException	22
NeoCursorOutOfRangeException	23
NeoManualLoadRequiredException	24
NeoClientException	25

例外番号	例外名	原因	対応策
1	CANNOT_RESOLVE_NS	名前空間が正しく設定されていません。	正しい名前空間を使用します。
1	INVALID_COMMAND	XMS Server に無効なコマンドが送信されました。	コマンドを確認します。
1	MISSING_DATA	コマンドに必要なデータがありません。	必要なデータがすべて含まれたコマンドを再入力します。 modifyXML の変更後データがない等
1	MISSING_PARAM	コマンドにパラメータが不足しています。	必要なパラメータがすべて含まれたコマンドを再入力します。 modifyXML の XPATH 式がない等
1	MODIFY_NODE_NOT_FOUND	変更するノードが存在しません。	ノードへの XPath を確認します。
1	XPATH_EXPR_NO_DOC	文書が存在しません。XPath 式で文書が指定されていません。Copy コマンド実行の際に、文書単位（MetaData 含む）でデータを取得しないクエリ（"/ND/A/B"など）による指定を行った場合、もしくは該当する文書が存在しない場合です。	XPath が正しいことと、文書が格納されていることを確認します。
1	UNK_ISOL_LEVEL	分離レベルが無効です。	正しい分離レベルを再入力します。有効な分離レベルは、Read_uncommitted、Read_committed、および Repeatable_read です。
4	INVALID_SID	現在、データベースにログインしていません。 ログインしていないか、セッションがタイムアウトになりました。セションタイムアウト値は、NeoXDBRuntime.xml の <Sessions><Timeout>で指定します。	前のセッションで NeoServer と切断された可能性があります。 もう一度データベースにログインします。

例外番号	例外名	原因	対応策
6	MALFORMED_XML	整形式の XML ではありません。	XML データを修正します。
7	MALFORMED_XPATH	XPath のシンタックスが正しくありません。	XPath シンタックスを修正します。
8	AC_NOT_ACTIVATED	AC 文書がないか、壊れています。	以前のエクスポートから文書をインポートするか、バックアップから復元します。
8	INTERNAL_ERROR	エラーの詳細については、サーバー ログを確認してください。	詳細をサーバー ログで確認します。
8	CANNOT_ALLOC_BUFFER_SPACE	RAM が不足しており、設定されたバッファを操作できません。クエリ実行時にあらかじめ定義された BufferPool を確保する際に、起動中他のプロセスによるメモリ利用などで、所定の領域が確保できない場合に発生するエラーです。 【補足】NeoServer 起動時には MemoryBufferPool で指定された Buffer をメモリ上に確保するためにメモリの空き領域チェックを行っておりますが、定義されたすべての Buffer を起動直後に展開するわけではありません。	NeoServer.xml（NeoServer.xml の MemoryBufferPool で定義されている Buffer）でバッファの容量を小さくするか、RAM を拡張します。
8	CANNOT_GET_SESSION	NeoServer が動作していません。	（Solaris の場合のみ、中断状態のプロセスが存在します。 NeoServer のプロセスをすべて強制終了してください） NeoServer を起動します。
8	COULD_NOT_GET_BUFFER	十分なバッファが設定されていません。 NeoServer 内部で Buffer 取得に関するファイル IO エラーが生じた場合に出力されます。	NeoServer.xml で容量の大きいバッファを設定します。

例外番号	例外名	原因	対応策
8	UTIL_NEC_COULD_INIT_DB_STARTUP	データベースが壊れている可能性があります。	バックアップからデータベースを復元します。
8	DICT_OR_INX_FULL	Database Dictionary/Index であらかじめ割り当てられた Index は、すべて使用されています。データのストア時に、.Database Dictionary/Index ※ の Status が Full になった状態で発生します。 ※ : NeoDatabase.xml の以下のサイズです。 ・<DataDictionary><Size> ・<DataIndex><Core><Size> ・<TagDictionary><Size> ・<TagIndex><Core><Size> ・<TagPlusDataIndex><Core><Size>	ステータスを確認し、一杯になっているファイルを特定します。NeoDatabase.xml でファイル サイズを拡張し、データベースを再作成します。
8	DB_FULL	ステータスを確認し、一杯になっているファイルを特定します。NeoXMLUtils import コマンド実行時に、データベース領域が足りなくなった場合に出力されます。	NeoDatabase.xml でファイル サイズを拡張し、データベースを再構築します。詳細をサーバー ログで確認します。
8	UTIL_NEC_DB_MAP_FULL	データベースに使用する Database Map File のフットプリントが不足しています。	ステータスを確認し、一杯になっているファイルを特定します。NeoDatabase.xml でファイル サイズを拡張し、データベースを再構築します。
1	UTIL_NEC_CANNOT_INSERT_AT_ND	root レベルに挿入しようとしています。	/ND より下に挿入されるよう、insert 文の XPath を修正します。
8	FULL_INX_MUST_BE_ENABLED	フルインデックスの機能が有効化されていません。	NeoDatabase.xml で Index Mode を変更し、データベースにインデックスを再構築します。

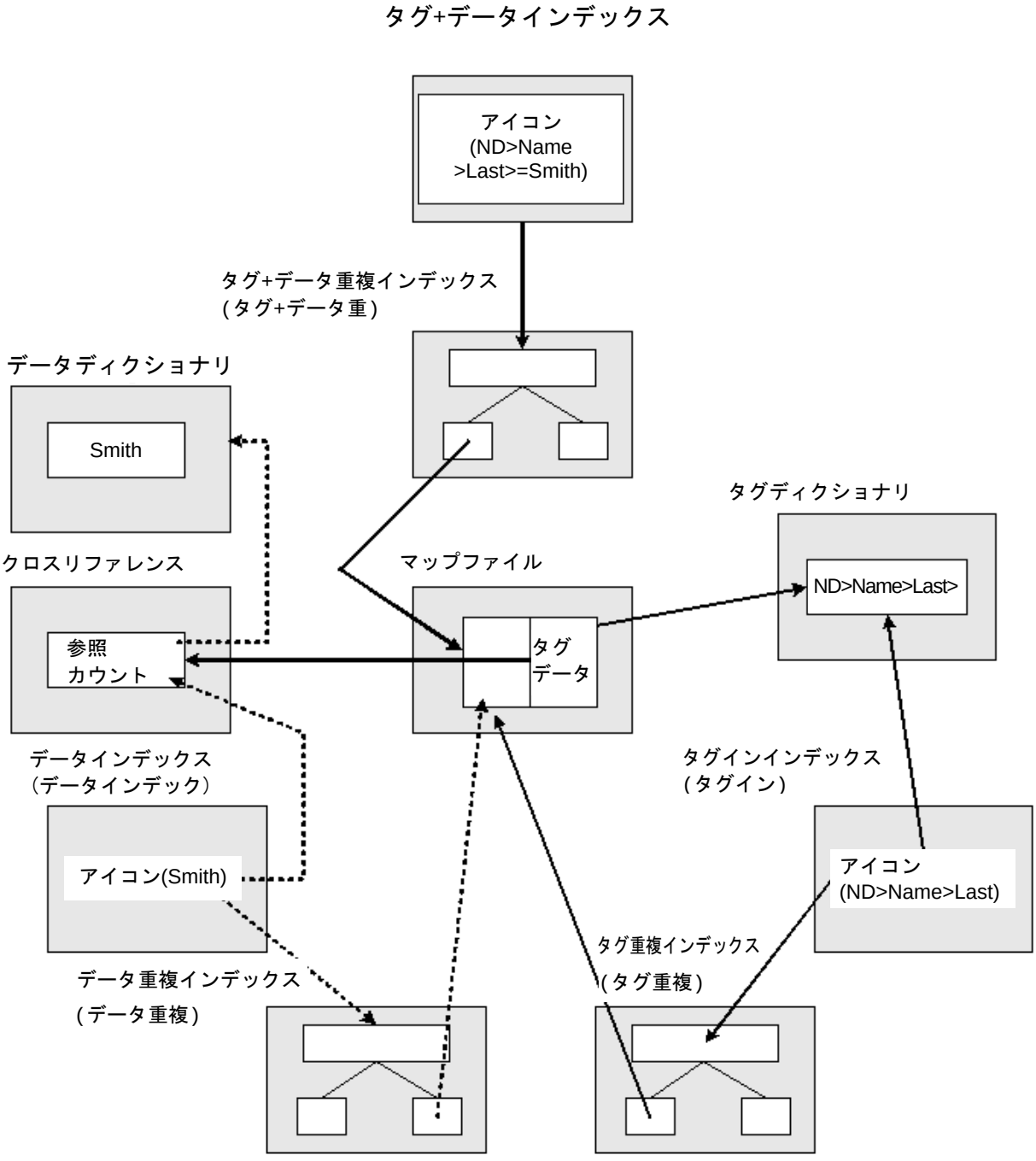
例外番号	例外名	原因	対応策
8	INX_FAILURE	NeoCore XMS Server のインデックスで内部エラーが発生しました。詳細はサーバー ログを参照してください。	インデックスを再構築します（reindex が機能しない場合は、バックアップからデータベースを復元します）。
8	INVALID_ADMIN_COMMAND	無効なコマンドです。	コマンドを修正します。
8	INVALID_MAP_FILE	NeoCore XMS Server の内部エラーが発生しました。 存在しないマップ ファイルへのアクセスがありました。	詳細をサーバー ログで確認します。
8	INVALID_QUERY	無効なクエリーです。	クエリーを修正します。
8	LOCK_EXCP	トランザクションの処理中に XMS Server の内部エラーが発生しました。	例外の種類を調べます。詳細をサーバー ログで確認します。
1	NO_ACTION_TO_TAKE	コマンドが入力されていません。	コマンドを入力します。
8	OP_FAILED	内部操作で、XMS Server の内部エラーが発生しました。	詳細をサーバー ログで確認します。
8	RES_SET_TOO_LARGE	コマンドの戻り値が XmlBufferSize を超えています。	NeoServer.xml で XmlBufferSize を拡張し（サイズの拡張または適切なサイズの BufferPool の作成が必要な場合があります）、NeoServer を再起動します。
8	STORE_FAILED	保存が完了していません。	STORE で領域不足が発生した時に発生。1 件も文書が格納されなかった時等。サーバー ログで詳細を確認し、修正します。
9	TXN_DEADLOCK	同一のデータに複数のトランザクションが試行されました。	トランザクションを終了し、再試行します。

例外番号	例外名	原因	対応策
10	TXN_NOT_ACTIVE	プロセスにトランザクションがない状態で、トランザクションのコミットまたはロールバックが試行されました。	新しいトランザクションを開始し、コマンドを再実行します。
10	TXN_NOT_STARTED	プロセスにトランザクションがない状態で、トランザクションのコミットまたはロールバックが試行されました。	新しいトランザクションを開始し、コマンドを再実行します。
10	TXNS_NOT_ENABLED	NeoCore XMS Server でトランザクションが有効化されていません。	インストールを確認します。
11	TXN_ROLLBACK	Transaction_rollback コマンドまたはタイムアウトにより、トランザクションがロールバックされました。	トランザクションを開始し、コマンドを再実行するか、ロールバック処理を行う場合のロジックをアプリケーションに追加します。
13	TXN_TIMEOUT	トランザクションがタイムアウトになりました。	NeoXDBRuntime.xml で MaxDuration を増やします。
8	XMS_APP_ERROR	XMS Server の内部エラーが発生しました。	詳細をサーバー ログで確認します。
1	QUERY_MATCHED_MULT_DOCS	一度にコピーできる文書は 1 つだけです。	文書を 1 つだけ指定するよう、コピーの設定を変更します。 （/ND[MetaData/DocID=NN] ではなく、/ND[MetaData/DocID=NN and MetaData/CopyNumber=X] とします）
8	DB_NOT_INITIALIZED	データベースを開けません。	サーバー ログで詳細を確認し、必要に応じて修正します。
2	AUTH_FAILED	データベースへのログインに失敗しました。	無効なユーザー名、パスワード、またはアカウントが入力された可能性があります。

例外番号	例外名	原因	対応策
1	INVALID_LOCK_TYPE	無効なロック タイプが指定されました。	次の有効なロック タイプを指定します。 Shared update exclusive
12	SESSION_NOT_ACTIVE	セッションの SID は無効です。	一旦ログアウトし、データベースにログインし直します。
1	NODE_MISMATCH	変更対象の XPath に、変更すべき XML がありません。	変更する XPath または XML を修正します。
11	NODE_MISMATCH_TXN_RB	変更処理でノードの不一致があったため、トランザクションがロールバックされました。	変更する XPath または XML を修正します。
8	AC_PROC_EXCP	アクセス管理ルールの処理中に、NeoCore XMS Server の内部エラーが発生しました。	詳細をサーバー ログで確認します。
1	NEC_ISO_LVL_NOT_SUP	サポートされていないシリアル化可能な分離レベルが使用されました。	有効なレベルは、Read_uncommitted、Read_committed、および Repeatable_read です。
14	RES_CHUNK_NOT_AVAIL	チャンクはすでに転送または破棄されています。	破棄される前にチャンクをフェッチするか、破棄を 2 回行わないようにしてください。 カーソルを使用した時のみ。 カーソルを使用し Destory 済の結果にアクセスした場合。
8	CANNOT_ALLOC_NEORESULT	neoresult を使用できません。	NeoServer がビジー状態でないときに再試行します。 カーソルを使用した時のみ。 カーソルを使用し、複数クライアントからのアクセスでリソースが不足。

例外番号	例外名	原因	対応策
8	SES_REACHED_CAPACITY	設定されている NeoXDBRuntime.xml の最大値を超えました。	config パラメータの値を増やすか、アプリケーションを再設定し、未使用の NeoResult をできる限り早急に解放します。
15	RES_EXCEEDS_MAX	結果セットの要素が多すぎます。	config パラメータの値を増やします。
8	RES_REQ_MORE_CHUNKS	設定されている最大値を超えました。	config パラメータの値を増やします。 カーソルを使用した場合のみ。 <ResultChunking>の<MaxNeoChunksPerNeoResult>の値が不足。
1	NRP_CONTENTS_INVALID	NeoResult Profile に無効なデータが含まれています。	NeoResult Profile コンストラクタの引数を確認します。
1	INVALID_TRACELEVEL	無効なトレース レベルが指定されました。	有効なトレース レベルを指定します。
8	LIC_INVALID	入力された XMS ライセンスは無効です。	有効なライセンスを取得します。
8	LIC_UNINITIALIZED	XMS ライセンスが初期化されていません。	詳細をサーバー ログで確認します。

データ構造



NeoCore XMS は、XML の表現とその関連するインデックスを永続的に格納するために、マップファイル、タグディクショナリ、データディクショナリ、クロスリファレンス、コアインデックス、重複ツリーなどの基本的な構造を使用します。

マップファイル

マップファイルは、マップエントリと呼ばれる固定サイズの構造のディスクベースの配列を反映します。マップエントリには、カプレットの数値表現とディクショナリへのポインタが入っており、このディクショナリにカプレットと関連付けられた実際のタグとデータが入っています。

文書や要素が削除されると、マップエントリが削除済みとして設定されます。その設定が解除されるのは、領域回復かデータベース再構築を実行した場合のみです。

マップファイルの回復時に使用するため、1 つのマップファイルが常に予備のマップファイルとして保持されます。

再構築時や手動による領域拡張時に、ファイルサイズを大きくすることができます。

タグディクショナリ

マップエントリ内のタグディクショナリのオフセットは 40 ビットです。したがって、タグディクショナリは最大 1TB になります。このディクショナリ内の領域は、バイト 1 から順番に使用されます。

タグディクショナリに追加された情報は、その情報を参照する文書がすべて削除された後でもそのまま残ります。その領域を回復するには、データベースを完全に再構築するしかありません。手動による領域拡張時や再構築時に、ファイルのサイズが大きくなることがあります。

タグディクショナリ内のエントリは、タグインデックスに対するクエリーで検索できます。

データディクショナリとクロスリファレンス

マップエントリ内のデータディクショナリのオフセットは、以下の 3 つの方法のいずれかで使用されます。

1. 6 バイトまでの文字列はすべてマップエントリの 6 バイトのデータオフセットに格納されます。
2. 13 文字未満の数値は、BCD 値としてマップエントリに直接格納されます。マッピングでは、数字 0〜9 と特殊文字+、-、,（カンマ）、.（小数点）、$が使用されます。
3. クロスリファレンスファイルは、参照カウントとデータディクショナリ内の各項目へのポインタを保持します。参照カウントがゼロになると、対応するディクショナリエントリは回復の対象になります。

タグディクショナリの場合と同様に、データディクショナリ内のすべての項目はデータインデックス経由で検索され、重複しないエントリだけがディクショナリ内に格納されます。

コアインデックスと重複ツリー

コアインデックスと重複ツリーファイルには、マップエントリとディクショナリ項目への参照が保持されます。コアインデックスと重複ツリーは、クエリー処理時にマップエントリを検索するためのメカニズムです。

- タグ+データインデックスは、ノードの等式演算クエリー（例えば、/ND/Name［Last="Smith"］）で使用され、重複ツリーを参照します。重複ツリーは、タグが/ND/Name/Last でデータが Smith であるすべてのマップエントリを参照します。
- タグインデックスは、ノードの存在確認クエリー（例えば、/ND/Name/Last）で使用され、重複ツリーを参照します。重複ツリーは、タグが/ND/Name/Last であるすべてのマップエントリを参照します。タグインデックスコアには、タグディクショナリのタグ/ND/Name/Last への参照も組み込まれます。

データインデックスは、データオンリーインデックスがオンになっているかどうかに応じて 2 つの役割のいずれかを果たします。データオンリーインデックスを使用すれば、マップエントリと関連付けられたデータのみに基づいたクエリーを実行できます。例えば、Smith を検索すると、「Smith」のすべてのインスタンスを抽出できます。つまり、それぞれのインスタンスを含む XML 構造が ND/Name/Last であれ、ND/Employees/Employee/Job-Title であれ、その他の構造であれ、すべてのインスタンスを抽出できるということです。データオンリーインデックスがオンになっていないと、データインデックスコアにはデータディクショナリのデータ項目への参照（クロスリファレンス経由）だけが含まれ、データインデックス重複は使用されません。一方、データオンリーインデックスがオンになっていると、データインデックスは重複ツリーを参照し、重複ツリーはデータが「Smith」であるマップエントリすべてを参照します。

コアインデックスはカンタムで構成され、サイズは 2 の累乗個のカンタムになります。

タグインデックスのカンタムは 16 バイトです。これには 2 つの関連付けが含まれます。関連付けの 1 つには、タグディクショナリ内の対応するタグのオフセットが入ります。もう 1 つの関連付けには、1 つのマップエントリのオフセットが入る（1 つのマップエントリだけがタグを参照している場合）か、関連付けのリストを含む重複ツリーへのオフセットが入ります。各関連付けは、タグを参照するマップエントリをポイントするか、タグオフセットを保持します（そのタグを参照する文書がすべて削除された場合など、そのタグを参照するマップエントリがない場合）。

データインデックスのカンタムは 12 バイトです。これには 1 つの関連付けが含まれます。この関連付けは、データディクショナリのデータ項目のオフセットを保持する（データオンリーインデックスがオンになっていない場合）か、1 つのマップエントリのオフセットを保持する（このデータを参照するマップエントリが 1 つだけの場合）か、このデータ項目を参照するマップ項目への関連付けのリストを含む重複ツリーをポイントします。

タグ+データインデックスのカンタムは 12 バイトです。これには、1 つの関連付けが含まれます。この関連付けは、1 つのマップエントリのオフセットを保持する（このタグとデータの組み合わせを参照するマップエントリが 1 つだけの場合）か、このタグとデータの項目を参照するマップエントリへの関連付けのリストを含む重複ツリーをポイントします。

インデックス重複は 2 つの構造体として実現されています。

- MTBase（MTMem ファイルに格納）は、第 1 レベルの重複で、関連付けられるマップファイルごとに重複を分割します。
- MTBase には重複ツリー（DTMem に格納）へのポインタが入ります。特定のマップファイルの重複はすべて同じ DTMem に格納されます。

データベース再構築、インデックス再構築、手動による拡張の際に、ファイルサイズを大きくすることができます。

NeoCore XMS でのデータの効率的な扱い方

この章では、情報管理作業に NeoCore XMS を使用するための望ましい方法について全体的な視点から説明します。NeoCore XMS と XPath の組み合わせは、リレーショナルデータベースと SQL の組み合わせと大きく異なっています。例えば、INSERT などの動詞は、XPath と NeoCore XMS の組み合わせにおいて、SQL での場合とは違った振る舞いをします。XML データベースを使用すると、他のソリューションに伴う障害の多くを排除できます。この違いを最大限に活用するには、慣れ親しんだ従来のデータ管理とは違った独特なアプローチが必要です。

データの更新

文書の更新（挿入、変更、削除、コピー）時には、その文書とそれに対応するインデックスチェーンがロックされます。更新中に他人がその文書の更新やクエリーを行うことはできません。

更新時のトランザクションの同時実行を最適化するため、少数の大きな文書ではなく、多数の小さな文書を使用してデータを格納するようにしてください。多数の文書を使用すれば、更新の進行中に別々の文書インスタンス内でさまざまな対象を更新できます。

単一文書の更新

例

```
<CATALOG>
     <CD>
               <TITLE>Empire Burlesque</TITLE>
               <ARTIST>Bob Dylan</ARTIST>
               <COUNTRY>USA</COUNTRY>
               <COMPANY>Columbia</COMPANY>
               <PRICE>10.90</PRICE>
               <YEAR>1985</YEAR>
     </CD>
     <CD>
               <TITLE>Unchain my heart</TITLE>
               <ARTIST>Joe Cocker</ARTIST>
               <COUNTRY>USA</COUNTRY>
               <COMPANY>EMI</COMPANY>
               <PRICE>8.20</PRICE>
               <YEAR>1987</YEAR>
     </CD>

</CATALOG>
```

変更対象を指す XPath:
```
/ND/CATALOG/CD/TITLE[.="Unchain my heart"]
```

XML の変更:
```
<TITLE>Greatest Hits</TITLE>
```

この時点で、別のユーザーが以下のクエリーを使用して、その文書のクエリーを実行すると、そのクエリーは、変更が完了するまでロックされます。
```
/ND/CATALOG/CD/ARTIST
```

複数文書の更新

```
<CATALOG>
  <CD>
   <TITLE>Empire Burlesque</TITLE>
   <ARTIST>Bob Dylan</ARTIST>
   <COUNTRY>USA</COUNTRY>
   <COMPANY>Columbia</COMPANY>
   <PRICE>10.90</PRICE>
   <YEAR>1985</YEAR>
  </CD>

</CATALOG>
```

```
<CATALOG>
  <CD>
   <TITLE>Unchain my heart</TITLE>
   <ARTIST>Joe Cocker</ARTIST>
   <COUNTRY>USA</COUNTRY>
   <COMPANY>EMI</COMPANY>
   <PRICE>8.20</PRICE>
   <YEAR>1987</YEAR>
  </CD>

</CATALOG>



<CATALOG>

  <CD>
    <TITLE>What would you do</TITLE>
    <ARTIST>Joe Cocker</ARTIST>
    <COUNTRY>UK</COUNTRY>
    <COMPANY>BMG</COMPANY>
    <PRICE>12.30</PRICE>
    <YEAR>1986</YEAR>

  </CD>

</CATALOG>
```

変更対象を指す XPath:
```
/ND/CATALOG/CD/TITLE[.="Empire Burlesque"]
```

XML の変更:
```
<TITLE>Greatest Hits</TITLE>
```

別のユーザーは以下のクエリーを使用して、他の文書のクエリーを正常に実行できます。

```
/ND/CATALOG/CD[ARTIST="Joe Cocker"]
```

結果:

```
<Query-Results>
 <CD>
  <TITLE>Unchain my heart</TITLE>
  <ARTIST>Joe Cocker</ARTIST>
  <COUNTRY>USA</COUNTRY>
  <COMPANY>EMI</COMPANY>
  <PRICE>8.20</PRICE>
  <YEAR>1987</YEAR>
 </CD>
</Query-Results>
```

1 つの大きな文書の格納と多数の小さな文書の格納との比較

競合を軽減するようにデータベースをセットアップする方法がいくつかあります。まず大きな影響があるのは、データをどこに格納するかということです。頻繁にアクセスする可能性の高い情報は、複数の文書に分散させておくべきです。そうすれば、1 つの文書に対するクエリーを実行しながら、同時に別の文書を更新することが可能です。1 つの文書として格納した場合は、クエリーが完了するまで更新ができなくなります。

以下のメリットとデメリットがあります。

- トランザクション – このメリットが大きいと言えます。多数の小さな文書に格納すれば、トランザクションロックを文書レベルで使用できます。
- メタデータのサイズ – この違いはほとんど問題になりません。1 つの文書あたり 5 つの XML データ要素のオーバーヘッドが発生するので、1 つの大きな文書に格納すれば、そのオーバーヘッドが少なくなります。ところが、1 つの大きな文書を 1,000 個の小さな文書に分割すれば、5,000 個の XML データ要素というメタデータオーバーヘッドが発生します。
- 格納時間 – 1 つの文書よりも多数の文書のほうが格納に時間がかかります。
- 多数の文書の格納 – マップファイルなどのリソースの取得がより多く必要になります。

データベースに 1 つの大きな文書がある場合、何か要求を行うたびにデータベース全体がロックされます。そのため、Xpriori, LLC では、小さな文書を使用して、トランザクション競合を軽減し、データベースパフォーマンスを向上させることを推奨しています。

複数文書の格納

多数の文書を一括して格納するには、NeoXMLUtils Import を使用します。このユーティリティは NeoCore XMS を初期化する場合に特に便利です。723 個の文書があるディレクトリに対してインポートを行うほうが、723 個の文書を 1 つ 1 つ連続して格納するよりも望ましいのは明らかです。Import はシングルユーザーモードで NeoCore XMS にアクセスします。ロックの必要はなく、ネットワーク転送のオーバーヘッドもありません。

すべての文書を 1 つのディレクトリに格納するかエクスポートしてから、インポートコマンドにそのディレクトリを指定してください。

以前に格納したことのない文書のファイル拡張子は.XML です。格納時に、各文書にはメタデータと<ND>タグが追加されます。通常の格納プロセスで処理された文書とエクスポートされた文書のファイル拡張子は.XPT です。この拡張子の場合、インポートコマンドは、文書を NeoCore XMS にコピーするだけです。メタデータと<ND>タグは最初の格納プロセスで追加されています。

Import の使用に関するマイナス面は、NeoServer を停止しなければならない点です。インポートの実行時には他の処理を一切実行しないでください。

例

NeoCore XMS に格納する注文書が 1 つのディレクトリ C:\temp にある場合、インポートコマンドラインは以下のようになります。

```
C:¥NeoCore¥neoxml¥bin>neoxmlutils Import c:¥NeoCore¥neoxml¥config
C:¥temp
```

1 つのファイル内に存在する複数の文書の格納

1 つのファイル内に存在する複数の文書を格納する場合、別のスレッドを使用して同じ構造のままに文書を格納しようとすると、マップファイルのデッドロックや競合の問題を引き起こしてしまう可能性があります。

これが問題になるのは、格納プロセスでファイル内のすべての文書が格納されるまで、複数のリソースがロックされるからです。ロックされたマップファイルの 1 つを 2 番目のスレッドが要求した場合、1 番目のスレッドがそのリソースを解放するまで、2 番目のスレッドは待機することになります。この待機時間が格納のタイムアウトより長くなれば、いずれかの格納スレッドが失敗します。

クエリー内のノード順序

親ノードに対して文書のクエリーを行った場合、NeoCore XMS はその子ノードすべてを再作成し、概念上の文書順序をクエリー結果に保持します。子レベルでクエリーを構築すると、格納の時間的順序でクエリー結果が返されます。

同様に、複数の文書にまたがるクエリーを行った場合も、文書 ID の割り当て順ではなく、内部格納ファイルに基づいて結果が返されます。

ソート式

ソート式によって、項目の順序を制御できます。ソートキーの直後に「ascending」や「descending」という語を使用してソート順を指定できます。ソート順のデフォルトは昇順（ascending）です。

ソート式の構文は、「sortby（key-list）」です。key-list にはキー式をコンマで区切ったリストを指定します。各キーにはキーワード「ascending」（デフォルト）か「descending」を付けることもできます。ソートキーには相対パスを使用しなければならず、各キーはソート対象の項目ごとに 1 つの値として評価されるものである必要があります。そうでない場合には例外が発生します。

複数のキーを指定した場合、リスト内の最初のキーが基本キーになり、それ以降のキーはその前のキーの値が等しい場合に順番を決めるのに使用されます。

ソートキーの比較は、キー式のデータ型に基づきます。クエリーからデータ型を判別できない場合は、文字列としてキーの比較が行われます。したがって、要素や属性を数値でソートするためには、double() や integer() の呼び出しを使用して明示的に変換を行う必要があります。ソートキーのパスとして常にドットを使用できます。ノードでない場合にはドットが必須です。ドットはソート対象の項目の値で置き換えられます。

例

格納ファイル（.map ファイル拡張子）が 3 つある場合は、最初にファイル 1 に格納されているすべての文書、次にファイル 2 に格納されているすべての文書、最後にファイル 3 に格納されているすべての文書が返されます。

クエリー:

```
/ND/MetaData/DocID
```

結果:

```
<Query-Results>
    <DocID>1</DocID>
    <DocID>5</DocID>
    <DocID>8</DocID>
    <DocID>11</DocID>
    <DocID>14</DocID>
    <DocID>2</DocID>
    <DocID>6</DocID>
    <DocID>9</DocID>
    <DocID>12</DocID>
    <DocID>3</DocID>
    <DocID>4</DocID>
    <DocID>7</DocID>
    <DocID>10</DocID>
    <DocID>13</DocID>
</Query-Results>
```

XMS の構成

挿入は、文書が格納されているのと同じマップファイルに対して行われ、そのマップファイル内に残っている領域によって影響を受ける可能性があります。例えば、他のマップファイルにはたくさんの領域が空いていても、挿入先の文書が入っているマップファイルが満杯であれば、挿入は失敗します。

このことは格納の場合も同様です。文書は複数のマップファイルにまたがることはないので、他のマップファイルに領域があっても、対象のマップファイルで領域がなくなってしまえば、格納は失敗します。

挿入と格納の違い

格納は、SQL 表に対する行の挿入に似ていて、文書の新しいインスタンスを作成します。この点で**格納**は SQL の INSERT に相当します。

XMS **の挿入**は 1 つ以上の文書に構造を追加するために使用します。SQL データベースに列を追加するのと同様です。

兄弟レベルの要素の挿入

文書に新しい要素を挿入すると、NeoCore XMS は、指定の XPath ターゲットの兄弟ノードとしてその新しい要素を扱います。文書を設計するときに、このことを覚えておくのは大切です。一度文書を作成してしまうと、子のない要素に子を挿入することはできません。XML の挿入は、常にターゲットの兄弟としての挿入になります。

後で子を追加できるようにするには、文書の作成時にその要素の「ダミー」の子を設定しておく必要があります。

例

このサンプル XML 文書では、兄弟の後ろに、または子のないノードの後ろに挿入を行う場合をいくつか取り上げます。

```
<Parent>
    <Child1>This is a childless element</Child1>
    <Child2>This is a parent element and also a sibling element of Child1
         <SubChild2-1>Child of Child2</SubChild2-1>
    </Child2>
</Parent>
```

この例で、<Child1>は子のない要素です。新しいノード<Child3>を/ND/Parent/Child1 の後ろに挿入すると、<Child1>の後ろに新しい子のないノードができます。

挿入ターゲットの XPath:

```
/ND/Parent/Child1
```

挿入する XML:

```
<Child3>Newly inserted childless element</Child3>
```

結果:

```
<Parent>
    <Child1>This is a childless element</Child1>
    <Child3>Newly inserted childless element</Child3>
    <Child2>This is a parent element and also a sibling element of Child1
         <SubChild2-1>Child of Child2</SubChild2-1>
    </Child2>
</Parent>
```

ただし、以下のように、整形式の XML フラグメント全体を挿入することは可能です。

挿入ターゲットの XPath:

```
/ND/Parent/Child1
```

挿入する XML:

```
<Child3>Newly inserted parent element
      <SubChild3-1>Child of Child3
      <SubChild3-1a>Child of SubChild3-1</SubChild3-1a>
      </SubChild3-1>
</Child3>
```

結果:

```
<Parent>
   <Child1>This is a childless element</Child1>
   <Child3>Newly inserted parent element
        <SubChild3-1>Child of Child3
        <SubChild3-1a>Child of SubChild3-1</SubChild3-1a>
        </SubChild3-1>
   </Child3>
   <Child2>This is a parent element and also a sibling element of Child1
        <SubChild2-1>Child of Child2</SubChild2-1>
   </Child2>
</Parent>
```

<Child2>の下に別の子要素を挿入できます。<Child2>には既に子要素<SubChild2-1>があるからです。新しいノード<SubChild2-2>を/ND/Parent/Child2/SubChild2-1 の後ろに挿入すると、<SubChild2-2>は<Child2>の子になり、<SubChild2-1>の兄弟になります。

挿入ターゲットの XPath:
/ND/Parent/Child2/SubChild2-1

挿入する XML:
```
<SubChild2-2>Newly inserted child of Child2 and sibling of SubChild2-1
</SubChild2-2>
```

結果:

```
<Parent>
   <Child1>This is a childless element</Child1>
   <Child3>Newly inserted childless element</Child3>
   <Child2>This is a parent element and also a sibling element of Child1
        <SubChild2-1>Child of Child2</SubChild2-1>
        <SubChild2-2>Newly inserted child of Child2 and sibling of
   SubChild2-1</SubChild2-2>
   </Child2>
</Parent>
```

複数の新しいノード（<SubChild2-3>と<SubChild2-4>）を一度に/ND/Parent/Child2/SubChild2-2 の後ろに挿入しようとすると、それぞれの兄弟の親ノードが指定されていないので、挿入処理は拒否されます。NeoCore XMS から以下のエラーメッセージが返されます。
```
Malformed XML.Could not parse XML input.
```

代わりに、以下のように SubChild2-2 の兄弟として 2 つの挿入を行ってください。

挿入ターゲットの XPath:
/ND/Parent/Child2/SubChild2-2

挿入する XML:

```
<SubChild2-3>Sibling 1</SubChild2-3>
```

結果:

```
<Parent>
    <Child1>This is a childless element</Child1>
    <Child3>A newly inserted childless element</Child3>
    <Child2>This is a parent element and also a sibling element of Child1
          <SubChild2-1>Child of Child2</SubChild2-1>
          <SubChild2-2>Newly inserted child of Child2 and sibling of
    SubChild2-1</SubChild2-2>
          <SubChild2-3>Sibling 1</SubChild2-3>
    </Child2>
</Parent>
```

挿入ターゲットの XPath:

```
/ND/Parent/Child2/SubChild2-3
```

挿入する XML:

```
<SubChild2-4>Sibling 2</SubChild2-4>
```

結果:

```
<Parent>
    <Child1>This is a childless element</Child1>
    <Child3>A newly inserted childless element</Child3>
    <Child2>This is a parent element and also a sibling element of Child1
          <SubChild2-1>Child of Child2</SubChild2-1>
          <SubChild2-2>Newly inserted child of Child2 and sibling of
    SubChild2-1</SubChild2-2>
          <SubChild2-3>Sibling 1</SubChild2-3>
          <SubChild2-4>Sibling 2</SubChild2-4>
    </Child2>
</Parent>
```

ノード不一致エラーを回避するための一致順序の変更

構造が異なる複数の文書を変更する場合は、変更処理をノードレベルで分割するのが望ましいと言えます。つまり、2 つの文書の構造が違う場合にノード不一致エラーを回避するには、子ノードをそれぞれ別々に変更します。

例

2 つの CD カタログが格納されています。2 つのカタログの構造の唯一の違いは、そのうちの 1 つからすべての<PRICE>タグが削除されていることです。

```
<CATALOG>
  <CD>
      <TITLE>Empire
  Burlesque</TITLE>
      <ARTIST>Bob Dylan</ARTIST>
      <COUNTRY>USA</COUNTRY>
      <COMPANY>Columbia</COMPANY>
      <PRICE>10.90</PRICE>
      <YEAR>1985</YEAR>
  </CD>
 ...
```

```
<CATALOG>
   <CD>
    <TITLE>Empire
  Burlesque</TITLE>
    <ARTIST>Bob Dylan</ARTIST>
    <COUNTRY>USA</COUNTRY>
    <COMPANY>Columbia</COMPANY>
    <YEAR>1985</YEAR>
   </CD>
  ...
```

変更対象を指す XPath:

```
/ND/CATALOG/CD[ARTIST="Bob Dylan"]
```

XML の変更:

```
<CD>
   <TITLE>Chronicles</TITLE>
   <ARTIST>Eric Clapton</ARTIST>
   <COUNTRY>USA</COUNTRY>
   <COMPANY>CBS</COMPANY>
   <PRICE>10.90</PRICE>
   <YEAR>1999</YEAR>
</CD>
```

`/ND/CATALOG/CD[ARTIST="Bob Dylan"]`という XPath の結果として、NeoCore XMS は 2 つの別々の文書構造を検出するので、この変更はノード不一致エラーを引き起こします。

他の文書の参照

文書間で要素を参照するには、一連のクエリーを使用します。NeoCore XMS の利点の 1 つは XPath の使用です。XPath と **SQL**（Structured Query Language）の比較は、**RISC**（Reduced Instruction Set Computer）アーキテクチャとフル機能の命令セットの比較に似ています。

例

完成した注文書があるとします。その注文に関する請求書を送るために顧客の住所を見つける必要があります。顧客の住所を取得するには、顧客番号を検索します。これには 2 つのクエリーが必要です。顧客番号を見つけるために注文書に対して実行するクエリーと、クエリー条件の一部に顧客番号を組み込んで顧客リストに対して実行するクエリーです。以下のコードサンプルは、両方のクエリーをそれぞれ行います。

```
    try {
      SessionManagedNeoConnection connect =
        new SessionManagedNeoConnection("localhost",7001);
      connect.login("guest","guest");
//First Query
      String results = connect.queryXML(
"/ND/purchase-order[header/po-number=¥"1231¥"]/customer-number");
      String header = "<Query-Results>";
      int end = results.indexOf("</Query-Results>");
      results = results.substring(header.length(),end);
//Second Query
      results = connect.queryXML(
        "/ND/customer[customer-number=¥"" + results +
        "¥"]billing/address");
      System.out.println("The address associated with purchase order 1231 is " +
        results);
    } catch (NeoException e) {
      e.printStackTrace();
    }
```

文字データ（CDATA）の使用

CDATA は XML の一部です。BLOB（Binary Large Object）ではなく、CLOB（Character Large Object）になります。NeoCore XMS の場合、1 つの CDATA セクションのサイズは、最大の空きバッファのサイズに制限されています。

CDATA は、マークアップなど、XML パーサーによって無視される文字データや、インデックスベースの検索を必要としない文字データを埋め込むための手段になります。したがって、インデックス構築のオーバーヘッドを軽減するために使用できます。CDATA の内容はインデックス構築の対象になりませんが、それでもインデックスを使用しない部分文字列検索を行うことは可能です。こうした特性は、PI（処理命令）とコメントにも当てはまります。

CDATA 構文

CDATA の XML 標準表記は次のとおりです。

```
<Element_Node><![CDATA[character data content]]></Element_Node>
```

名前空間

XML の名前空間は、要素群の固有の範囲 ID としての役割を果たす **URI** によって定義します。

NeoCore XMS は、名前空間と、各要素タグに対応するプレフィックスを保持します。ただし、インデックス構築は、タグ自体に基づいています。同じタグを使用した複数の文書からデータを抽出するときには、それらの文書内でタグが別々の意味を持つ場合であっても、名前空間は無視されます。

例

要素タグ名自体は修飾されません。〈para〉

同じ要素名を別々の目的で使用する文書群では、個別にプレフィックスを設定できます。
<me:para>（医療文書用。この場合の<para>タグは「paramedic」を意味します）
<ld:para>（法律文書用。この場合の<para>要素は「paralegal」を意味します）
<pg:para>（段落を設けた文書用。この場合の<para>要素は「paragraph」を意味します）。

完全修飾要素名では、各要素の URI に対する xmlns 参照と、要素タグに対応するプレフィックスを指定します。プレフィックスとタグはコロンで区切ります。

```
<me:para xmlns:me = "http://hospital.com/jobdef">
<ld:para xmlns:ld = "http://lawfirm.com/employeelist">
<pg:para xmlns:pg = "http://indexing.com/literature">
```

/ND//para を対象としたクエリーでは、3 つの文書すべてに含まれているすべての<para>とそれぞれの子要素が返されます。

名前空間の使用

XML 名前空間は URI によって定義します。この URI は、要素群に関する固有の範囲 ID としての役割を果たします。タグとクエリーには名前空間を指定できますが、クエリーでは名前空間を前置きの形で宣言する必要があります。

```
declare namespace "name"="URI";(query)
```

プレフィックスは、「xmlns:」で始まる名前の最上位要素の属性を使用してフルネームにリンクします。プレフィックス自体は何の意味もなく、単にフルネームのための簡便なプレースホルダとしての役割を果たします。そのフルネームが URI です。

以下の XML フラグメントの例を取り上げましょう。

<A> <B>Text</B> </A>	<id:A xmlns:id="http://w3c.com"> <id:B>Text2</id:B> </id:A>

XML フラグメントの比較

/ND/A というクエリーは、上記の両方の XML フラグメントを正確に返します。

/ND/A/Bというクエリーから返される結果は、以下のとおりです。

```
<B>Text</B>
<id:B xmlns:id='http://w3c.com'>Text2</id:B>
```

要素ノードの使用

要素ノードを使用して文書を設計するのは、望ましい方法です。ノードの属性としてではなく、要素ノードとしてデータを格納すれば、1 つのノードを変更するときにすべての属性を指定する必要はありません。また、1 つの属性の変更も可能です。

NeoCore XMS は、通常の要素形式を使用した文書と、属性を使用した文書を、同じ格納効率で格納します。データベースサイズを縮小し、処理効率を高めるために、属性を多用した文書をこれまで使用してきたかもしれませんが、文書を NeoCore XMS に格納する場合は、両方の文書タイプに関するオーバーヘッドとパフォーマンスのレベルはほぼ同じです。

Section 属性の値の変更

例

次のサンプル文書で section 属性の値を 5 から 4 に変更するには、実際に値を変更する属性だけでなく、値を変更しない属性もすべて一緒に変更ステートメントに記述する必要があります。

```
<entry>
    <header>
          <doc
          code="XYZ2"
          type="article"
          section="5"
          date="6/9/2001"
          location="Plunkett Library"
          />
          <version>3</version>
     </header>
     <source
     author="James J. Jameson"
     title="XML Exposed"
     journal="XML Journal"
     />
</entry>
```

変更対象を指す XPath:

```
/ND/entry/header/doc
```

XML の変更:

```
<doc code="XYZ2" type="article" section="4" date="6/9/2001"
location="Plunkett Library"></doc>
```

属性ではなく、要素ノードにデータが入っていると、変更はもっと簡単です。

要素を使用した設計

```
<entry>
    <header>
         <doc>
               <code>XYZ2</code>
               <type>article</type>
               <section>5</section>
               <date>6/9/2001</date>
               <location>Plunkett Library</location>
         </doc>
         <version>3</version>
    </header>
    <source>
         <author>James J. Jameson</author>
         <title>XML Exposed</title>
         <journal>XML Journal</journal>
    </source>
</entry>
```

変更対象を指す XPath:

```
/ND/entry/header/doc/section
```

XML の変更:

```
<section>4</section>
```

属性と要素の使い分け

XML コミュニティではしばしば、属性と要素のどちらがより良い構造になるかについて白熱した議論が交わされます。この議論に決着をつけることはできませんが、NeoCore XMS 内での XML の設計をできるだけ効率的にするための簡単な指針を提供することはできます。

XML 構造の設計では、要素と要素内の属性のどちらでもデータを記述できる状況がよくあります。

基本的に、属性は要素を修飾するものとして使用し、要素は値か他の要素のコンテナとして使用すべきです。要するに、属性は内容のメタデータとして使用し、要素は内容そのものとして使用するのが最善だということです。

この問題の別のアプローチは、データを「記述」と「包含」という観点から検討することです。属性は「記述」関係を定義し、要素は「包含」関係を定義します。

例えば、携帯電話には色と重さがあります（つまり包含します）。重さの測定値はオンスで記述するとします。こうした関係を以下のように表現できます。

```
<cellphone>
      <weight unit="ounces">10</weight>
      <color>blue</color>
</cellphone>
```

携帯電話は色と重さを「包含」するので、こうした特質は cellphone 要素の子要素として表現します。重さはオンスで「記述」するので、weight 要素に units="ounces"属性を置きます。

スタンドアロン要素を使用すると記憶域オーバーヘッドが大きくなるという問題

XML において、カプレットとは属性や要素の中で表現される名前と値のペアのことです。以下の例を考えてみてください。

```
<platform>Solaris</platform>
<server platform="Solaris"/>
```

platform/Solaris のカプレットは、要素として表現したものであれ属性として表現したものであれ、NeoCore XMS ではまったく同じ方法で格納されます。

ただし、ここで覚えておくべきなのは、属性の例ではインデックス構築時に追加のエントリが格納されるということです。属性を持つ<server>要素は、名前だけがあって値のない<server />というカプレットとして格納されます。

したがって、値も子要素もない、属性だけを組み込んだ要素は作成しないようにしてください。そのように属性だけを組み込んだスタンドアロン要素が現に存在する場合は、別の書き方を検討してみるべきです。

以下に、属性カプレットを等価の要素カプレットに変更するべき場合の例を示します。

```
<object>
      <property color="blue"/>
      <property weight="10 ounces"/>
</object>
```

これは以下のようにすべきです

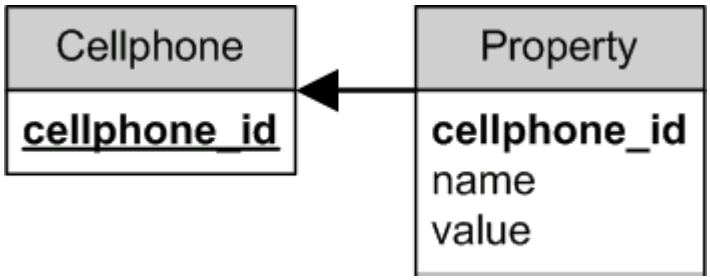
```
<object>
      <color>blue</color>
      <weight>10 ounces</weight>
</object>
```

上の例において、格納されるカプレットの数は 5 個（object/、property/、color/blue、property/、weight/10 ounces）から 3 個（object/、color/blue、weight/10 ounces）になります。この変更のもう 1 つの良い点は、データや構造に対して本質的な意味を持たない要素（property）がなくなっていることです。

属性の誤用

リレーショナルモデルから XML の階層モデルにデータを移行する際に犯しがちな間違いは、名前と値のペア（つまりカプレット）を 2 組の属性にしてしまうことです。

cellphone がいくつかの property を持っていることを示す次のデータベース表を例として取り上げましょう。



携帯電話の特性を示したデータベース表

XML の階層モデルで、上記のデータは次のように表現するのが論理的に見えるかもしれません。

```
<cellphone id="123">
      <property name="color" value="blue" />
      <property name="weight" value="10" />
      <property name="battery_hours" value="2" />
</cellphone>
```

ところが、この XML 構造には、それを使用するアプリケーションに大きな悪影響を与えかねない本質的な問題があります。なぜそう言えるのでしょうか。要素内の属性は互いに兄弟の関係にあり、それゆえに互い同士の直接の関係がないからです。

10 時間持つバッテリのある携帯電話を検索する場合、上記の XML に次の XPATH を使用したくなりますが、これは正しくありません。

```
/ND/cellphone[property/@name="battery_hours" and property/@value="10"]
```

一見、このクエリーは機能しそうに見えます。しかし、「name」属性と「value」属性は互いに直接の関係がないので、このクエリーは期待したとおりに機能しません。この XPATH は、実際には以下のような携帯電話を抽出することになります。

a） name 属性が「battery_hours」である property を持つ

なおかつ

b）value 属性が「10」である property を持つ

ですから、返される結果には上記の cellphone という XML 例が含まれてしまいます。この例には、同じ property 要素ではないものの name="battery_hours"という属性と value="10"という属性があるからです。

```
<Query-Results>
   <cellphone>
         <property name="id" value="123"/>
         <property name="color" value="blue"/>
         <property name="weight" value="10"/>
         <property name="battery_hours" value="2"/>
   </cellphone>
</Query-Results>
```

この問題を回避するには、次のように、述部を移動して、両方の条件にかなう property だけを返すようにする必要があります。

```
/ND/cellphone/property[@name="battery_hours" and @value="10"]
```

しかし、今度は一致した property 要素だけが返されて、その要素がどの cellphone に属するのかが分からなくなってしまいます。

```
<Query-Results>
      <property name="battery_hours" value="10"></property>
</Query-Results>
```

では、この問題はどう解決したらよいのでしょうか。「name」と「value」の間の関係を結合し直す、というのが答えです。この XML は以下のようであるべきです。

```
<cellphone>
   <properties>
         <id>123</id>
         <color>blue</color>
         <weight>10 ounces</weight>
         <battery_hours>2</battery_hours>
   </properties>
</cellphone>
```

10 時間のバッテリ寿命を持つ携帯電話を検索するには、次の XPATH を使用します。

```
/ND/cellphone[properties/battery_hours="10"]
```

このクエリーはより簡単に作成できるだけでなく、前の例よりも速く結果を返します。前のクエリー（/ND/cellphone/property[@name="battery_hours" and @value="10"]）の場合、NeoServer はデータの受け渡しを 2 回行う必要があります。1 回目は name が battery_hours であるものをすべて取り出し、2 回目は value が 10 であるものをすべて取り出します。それから、NeoServer はこの 2 つの結果セットを組み合わせて返す必要があります。

後の簡単なクエリー（/ND/cellphone [properties/battery_hours="10"] ）の場合、

NeoServer は受け渡しを 1 回だけ行い、すぐに結果セットを返します。この 2 つのクエリーの違いはミリ秒単位でしか計測できませんが、そのデータを使用するのが基幹アプリケーションの場合は、そのミリ秒が大きな意味を持つことになります。

tree コマンドの使用

tree コマンドを使用して、アプリケーション内から所定のノードの構造を検出できます。このコマンドは、アプリケーション側が、渡された XML の内容を知らない場合に使用できます。

tree コマンドを使用するには、XPATH クエリーの前に「tree」と 1 個のスペースを置きます。例えば、次のようにします。

```
tree /ND/recipe
```

注:この入力は/ND で始めなければならず、スラッシュで区切った一連の名前だけを含めます。例えば、tree /ND/tag1/tag2 のようにします。正しく使用しないと、NeoMalformedXPathException が発生します。

tree コマンドを使用すると、クエリーの対象であるノードレベルに存在する（または存在していた）すべての要素と属性のリストが表示されます。tree コマンドは、現在のデータベースインスタンスに存在したことのある構造を返します。

このコマンドがどのように機能するのかについて、次の XML を例にして考えてみましょう。

```
<ND>
    <recipe category="breakfast">
          <name>Dutch Baby</name>
          <ingredient>
                <name>Eggs</name>
                <amount>5</amount>
          </ingredient>
          <ingredient>
                <name>Butter</name>
                <unit>Tablespoon</unit>
                <amount>5</amount>
          </ingredient>
    </recipe>
</ND>
```

tree コマンドをこの「recipe」ノードで実行すると、そのノードレベルに含まれるすべての要素名と属性名が表示されます。

例えば、「tree /ND/recipe」というクエリーは次の結果を返します。

```
<Tree-Results>
      name, ingredient, @category
</Tree-Results>
```

この結果は、すべての/ND/recipe ノードに、「name」要素、「ingredient」要素、「category」属性が存在する（または存在した）ことを示しています。

「tree /ND/recipe/ingredient」というクエリーを実行すれば、次の結果が表示されます。

```
<Tree-Results>
      name, unit, amount
</Tree-Results>
```

しかし、ここで注意しなければならないのは、すべてのノードが必ずしも「Tree-Results」で示される構造全体を持っているわけではないということです。「Tree-Results」に表示されるのは、そのノードに存在したことのあるすべての構造であり、現在のいずれかのノードがそれをすべて持っているかどうかは関係ありません。このような結果が表示されるのは、削除時にパフォーマンスを上げるために、ディクショナリやインデックスをクリアしないからです。また、このような結果によって、いったん削除されてもまた作成されると予想される要素に関するアクセス制御の管理を続けることも可能になります。

上記のレシピに別の材料を付け加えることにしましょう。

```
<ingredient>
    <name>Sugar</name>
    <alternate>Brown Sugar</alternate>
    <unit>Cup</unit>
    <amount>1</amount>
</ingredient>
```

「tree /ND/recipe/ingredient」というクエリーをもう一度実行すれば、次の結果が表示されます。

```
<Tree-Results>
   name, alternate, unit, amount
</Tree-Results>
```

注:この砂糖（"Sugar"）という材料を削除してもう一度 tree コマンドを実行しても、同じ「Tree-Results」が返されます。

count コマンドの使用

tree コマンドを使用して XML の構造を調べたい場合に、データベースに今も存在しているノードだけを処理するには、さらに count コマンド（『***System Administration Guide***』を参照）を使用して各ノードのインスタンスがいくつ存在するかを判別できます。count が 0 を戻す場合、その要素や属性は XML 構造内に現在存在しないことが分かります。

以下の XML フラグメントを例にして考えましょう。

削除前:

```
<ND>
    <recipe category="breakfast">
          <name>Dutch Baby</name>
          <ingredient>
                <name>Eggs</name>
                <amount>5</amount>
          </ingredient>
          <ingredient>
                <name>Butter</name>
                <unit>Tablespoon</unit>
                <amount>5</amount>
          </ingredient>
          <ingredient>
                <name>Sugar</name>
                <alternate>Brown Sugar</alternate>
                <unit>Cup</unit>
                <amount>1</amount>
          </ingredient>
    </recipe>
</ND>
```

Sugar という Ingredient の削除後:

```
<ND>
    <recipe category="breakfast">
          <name>Dutch Baby</name>
          <ingredient>
                <name>Eggs</name>
                <amount>5</amount>
          </ingredient>
          <ingredient>
                <name>Butter</name>
                <unit>Tablespoon</unit>
                <amount>5</amount>
          </ingredient>
    </recipe>
</ND>
```

Sugar という Ingredient を削除してから、count コマンドで「/ND/recipe/ingredient」のクエリーを行うと、次の結果になります。

```
<Count-Results>2</Count-Results>
```

ノードの数を数えるもう 1 つの方法は、XQuery の count() 関数を使用することです。

```
"Count(/ND/recipe/ingredient)"
```

上記の結果が返されるのは、このレシピに 2 つの材料（バター"Butter"と卵"Egg"）が残っているからです。

しかし、クエリー「count /ND/recipe/ingredient/alternate」を実行すると、結果は次のようになります。

```
<Count-Results>0</Count-Results>
```

「alternate」という要素は、削除されたノード（sugar ingredient）に含まれていたので、この結果は、「alternate」要素が現在は存在しないという事実を反映しています。

要約すると、あるノードの構造を調べるには tree コマンドを使用しますが、その構造が現在も存在するかどうかを確かめるには count コマンドと組み合わせて使用する必要がある、ということになります。

トランザクションのネストの禁止

NeoCore XMS では、トランザクションのネストを使用できません。つまり、C++のネストされた if ステートメントのようなコーディングはできません。

例（「再試行」の適切な方法）：

次の擬似コード例は正しく機能します。

```
TRANSACTION_START
while !done
{
modify /ND/CATALOG/CD/TITLE
if error
{
TRANSACTION_ROLLBACK
continue;
done = true;
}
}
TRANSACTION_COMMIT
while !done
{
TRANSACTION_START
modify /ND/CATALOG/CD/TITLE
if error
{
TRANSACTION_ROLLBACK
continue;
}
TRANSACTION_COMMIT
done = true;
}
```

例（不適切な方法）：

次のコード例は、トランザクションのネストを含んでおり、機能しません。

```
try {
SessionManagedNeoConnection connect = new
SessionManagedNeoConnection("localhost",8080);
connect.login("guest","guest");
connect.startTransaction();
System.out.println("Started first transaction");
{
      // This will throw a NeoTransactionException
      connect.startTransaction();
      // this will never be printed because of the exception thrown
      System.out.println("Started second transaction.");
      connect.commitTransaction();

}
connect.commitTransaction();
} catch (NeoTransactionException e) {
// this will be executed
e.printStackTrace();
} catch (NeoException e) {
// this should not be executed
e.printStackTrace();
}
```

上記のコードを実行すると、以下の結果が表示されます。

```
Started first transaction
com.neocore.httpclient.NeoTransactionException:<Error>
<Name> Transaction not started </Name>
<Message> There is a transaction currently in progress.</Message>
<Exception-Number> 10 </Exception-Number>
</Error>
at com.neocore.httpclient.XMLUtils.checkAllErrors(XMLUtils.java:479)
at com.neocore.httpclient.XMLUtils.checkNoXMLErrors(XMLUtils.java:568)
at com.neocore.httpclient.NeoConnection.startTransaction
(NeoConnection.java:540)
at com.neocore.httpclient.SessionManagedNeoConnection.startTransaction
(SessionManagedNeoConnection.java:540)
at quickie.main(quickie.java:30)
```

大規模なデータベースの再起動

クラッシュ後には、その時点でインデックスは破損しているだろうと考えられるので、自動的にインデックス再構築が行われます。現在のところ、インデックス再構築にはデータの入力と同じほどの時間がかかる場合があります。時間の制限がある場合は、以下の 2 つの方法について検討してください。

1）コマンドラインから NeoXMLUtils ReIndex コマンドを使用して、手動でインデックス構築を行うことを推奨します。

2）最近のバックアップがある場合、バックアップを復元して、現在のトランザクションログを適用できます。この第 2 の方法のほうが速い場合があります。

ユーティリティクラス分離

次の例のようなネイティブ XML 形式のユーザー独自の XML 結果を解析するとしましょう。

```
<Inserted-Nodes>5</Inserted-Nodes>
```

後になって Xpriori, LLC が XML タグ構造を変更した場合、ユーザーのコードも新しいタグ構造に合わせて変更することが必要になります。こうした状態にならないようにするため、Xpriori, LLC では XMLUtils の使用を推奨しています。XMLUtils は、NeoCore XMS の Java API と C++ API に含まれるクラスです。『*System Programming Guide*』の Java API と C++ API に関する資料を参照してください。

文書のコピーの変更

ここでは、文書のコピーを変更する方法について詳しく説明します。以下の注意点があります。

- コピー操作では、基本的に文書の同一コピーが作成されますが、CopyNumber だけは更新されます。
- 通常の XPath クエリーを使用して、いくつかのコピーのうちの 1 つだけを変更することはできません。
- 特定のコピーを変更するには、文書の MetaData/CopyNumber 要素を使用する必要があります。

例えば、データベース内の次の文書をコピーして変更するとしましょう。

```
<CDOrder>
    <Title>Prototype Title</Title>
    <Artist>Prototype Artist</Artist>
    <Quantity>0</Quantity>
    <Price>$0.00</Price>
</CDOrder>
```

/ND[CDOrder]を使用して、この文書のコピーを作成します。

オリジナルとコピーは同一なので、一方だけを選択するクエリーを作成することはできません。以下に示すように、/ND/CDOrder というクエリーでは、この文書のオリジナルとコピーの両方が返されます。

```
<Query-Results>
    <CDOrder>
       <Title>Prototype Title</Title>
       <Artist>Prototype Artist</Artist>
       <Quantity>0</Quantity>
       <Price>$0.00</Price>
    </CDOrder>
    <CDOrder>
       <Title>Prototype Title</Title>
       <Artist>Prototype Artist</Artist>
       <Quantity>0</Quantity>
       <Price>$0.00</Price>
    </CDOrder>
</Query-Results>
```

MetaData/CopyNumber 要素を使用すれば、オリジナルとコピーのどちらか一方を選択して変更できます。

/ND[MetaData/CopyNumber="2"]/CDOrder という XPath クエリーを使用すると、次の XML（つまりコピー）が返されます。

```
<Query-Results>
   <CDOrder>
         <Title>Prototype Title</Title>
         <Artist>Prototype Artist</Artist>
         <Quantity>0</Quantity>
         <Price>$0.00</Price>
   </CDOrder>
</Query-Results>
```

前のクエリーを使用すれば、以下のクエリー結果のように、コピーのうちの 1 つを変更できます。

```
<Query-Results>
    <CDOrder>
         <Title>Prototype Title</Title>
         <Artist>Prototype Artist</Artist>
         <Quantity>0</Quantity>
         <Price>$0.00</Price>
    </CDOrder>
    <CDOrder>
         <Title>Greatest Hits</Title>
         <Artist>Bob Dylan</Artist>
         <Quantity>1</Quantity>
         <Price>$10.98</Price>
         <Quantity>1</Quantity>
         <Price>$10.98</Price>
    </CDOrder>
</Query-Results>
```

初期データの回復

Xpriori, LLC では、新規の NeoCore XMS データベースを（CreateDB コマンドで）作成した直後に BackupDB を実行することを推奨します。こうすることにより、最初に予定されているバックアップより前のトランザクションを元に戻すことができます。

クエリー最適化のポイント

クエリーを作成するときの指針は、以下のとおりです。

- できるだけ小さいセットを使用するようにします。
- 処理コストが最小になるようにします。
- XMS のインデックス構築のしくみを活用します。
- XQuery や XPath の効率の悪さが最小になるようにします。
- クエリーの最適化は、クエリー、挿入、変更、コピー、削除に適用されます。

クエリー最適化の一般的なルール

- クエリーの「インデックス構築可能」な部分をできる限りはっきりさせます。
  - 「子ステップ」のパスを使用します（子孫ステップやワイルドカードステップではありません）。

    ```
    -/ND/employee
    -/ND/employee/name/@language
    -/ND/employee/name/first
    ```

  - 述部を使用して、修飾インデックスの範囲を狭めます。

    ```
    -/ND/employee[salary/monthly>5000]/name
    -/ND/employee[@id=starts-with("R")][salary/monthly>5000]
    ```

  - タグ+データインデックスを活用します。

    ```
    -ND/employee[name/last="Ricoh"]
    ```

- 基本のフィルタリングまたは検索のメカニズムとして、シーケンス式ではなくセット式を使用します。
- シーケンス式には以下のものがあります。
  - FLWR ステートメント
  - XQuery 関数
  - 算術関数
  - ノード同士の値の比較

- セット式には以下のものがあります。
  - インデックス参照

    -/ND/employee/name/first
  - 述部によるインデックス参照

    -/ND/employee/name[first="Jon"]
  - 定数を使用した比較述部

    -/ND/employee/salary[monthly>5000]
  - セット演算

    -和集合、共通部分

例:

遅い:	for $employee in /ND/employee where $employee/salary/monthly>10000 return $employee/(name/last, salary)
速い:	for $employee in /ND/employee [salary/monthly>10000] return $employee/(name/last, salary) - または - /ND/employee[salary/monthly>10000] /(name/last, salary)

ネストされたセット: 外側を小さく

XQuery の for ステートメントで述部を使用すると、クエリーの残りの部分のセットサイズが小さくなります。

例:

遅い:

```
for $emp in /ND/employee[salary/monthly]
where $emp/name/first="Peter"return $emp
```

速い:

```
for $emp in /ND/employee[name/first="Peter"]
where $emp[salary/monthly]return $emp
```

理由: 外側のセットが小さいほど、クエリーの実行が速くなります。

サンプルデータ:

```
<employee id="Ricoh">
      <name>
            <first>Jon</first>
            <last>Ricoh</last>
      </name>
      <phone>
            <type>Direct</type>
            <number>7000</number>
      </phone>
      <salary>
            <monthly>6500</monthly>
      </salary>
</employee>

<employee id="Lexmark">
      <name>
            <first>Peter</first>
            <last>Lexmark</last>
      </name>
      <phone>
            <type>Direct</type>
            <number>6200</number>
      </phone>
      <salary>
            <monthly>6700</monthly>
      </salary>
</employee>
```

述部と WHERE 句の比較

FLWR ステートメントの WHERE 句は、述部条件と比べて実行速度が遅くなります。

遅い:

```
for $emp in /ND/employee
where $emp/name/first="Peter"
return $emp
```

速い:

```
for $emp in /ND/employee[name/first="Peter"]
return $emp
```

理由: 述部はインデックスを直接検索するので、FLWR ステートメントが評価される前にセットを小さくします。

例:

```
<employee id="Ricoh">
      <name>
            <first>Jon</first>
            <last>Ricoh</last>
      </name>
      <phone>
            <type>Direct</type>
            <number>7000</number>
      </phone>
      <salary>
            <monthly>6500</monthly>
      </salary>
</employee>

<employee id="Lexmark">
      <name>
            <first>Peter</first>
            <last>Lexmark</last>
      </name>
      <phone>
            <type>Direct</type>
            <number>6200</number>
      </phone>
      <salary>
            <monthly>6700</monthly>
      </salary>
</employee>
```

修飾パスと子孫パスの比較

子孫ステップ（//）で、パスの部分的な修飾が可能になります。

「//」を指定すると、データベース構造の「//」のレベルからすべての子のレベルに至るすべてのタグインデックスが検索されます。

遅い:

```
/ND//name
```

速い:

```
/ND/employee/name
```

パスを厳密に修飾したほうが、検索するインデックスが少なくなります。

```
<employee id="Ricoh">
      <name>
            <first>Jon</first>
            <last>Ricoh</last>
      </name>
      <phone>
            <type>Direct</type>
            <number>7000</number>
      </phone>
      <phone>
            <type>Direct</type>
            <number>(719) 226-7000</number>
      </phone>
</employee>

<directory>
      <entry>
            <name>Peter Lexmark</name>
            <phone>226-5000</phone>
            <email>plexmark@direx.com</email>
      <entry>
</directory>
```

修飾パスとワイルドカードパスの比較

ワイルドカード（/*）で、パスに不特定のノード名を使用することが可能になります。

「/*」を使用すると、評価の前に、子ノード名のツリーウォークが行われます。

遅い:

```
/ND/*/name
```

速い:

```
/ND/employee/name
```

ワイルドカードステップはインデックスを使った検索を行いません。

例:

```
<employee id="Ricoh">
      <name>
            <first>Jon</first>
            <last>Ricoh</last>
      </name>
      <phone>
            <type>Direct</type>
            <number>7000</number>
      </phone>
      <phone>
            <type>Direct</type>
            <number>(719) 226-7000</number>
      </phone>
</employee>

<directory>
      <entry>
            <name>Peter Lexmark</name>
            <phone>226-5000</phone>
            <email>plexmark@direx.com</email>
      <entry>
</directory>
```

子孫パスとワイルドカードパスの比較

子孫クエリーは 2 つのインデックス（//の前のインデックスとその後のインデックス）を使用します。「/*」を使用すると、子ノード名のツリーウォークが行われます。インデックスは使用されません。

遅い:

```
/ND/*/name
```

速い:

```
/ND//name
```

ワイルドカードステップはインデックスを使った検索を行いません。

例 :

```
<employee id="Ricoh">
      <name>
            <first>Jon</first>
            <last>Ricoh</last>
      </name>
      <phone>
            <type>Direct</type>
            <number>7000</number>
      </phone>
      <phone>
            <type>Direct</type>
            <number>(719) 226-7000</number>
      </phone>
</employee>

<directory>
      <entry>
            <name>Peter Lexmark</name>
            <phone>226-5000</phone>
            <email>plexmark@direx.com</email>
      <entry>
</directory>
```

修飾文字列とワイルドカード文字列の比較

ワイルドカード文字列（starts-with()、ends-with()、contains()）を使用すると、部分的に修飾された値に照らした評価ができます。ワイルドカードを使用すると、修飾するすべてのタグインデックスのノード値に対して文字列検索が行われます。

遅い:

```
/ND/employee [starts-with(@id, "Ri")]
```

速い:

```
/ND/employee[@id="Ricoh"]
```

等号を使用して完全な文字列を指定すれば、タグ+データインデックスが直接検索されます。

例:

```
<employee id="Ricoh">
      <name>
            <first>Jon</first>
            <last>Ricoh</last>
      </name>
      <phone>
            <type>Direct</type>
            <number>7000</number>
      </phone>
      <salary>
            <monthly>6500</monthly>
      </salary>
</employee>

<employee id="Lexmark">
      <name>
            <first>Peter</first>
            <last>Lexmark</last>
      </name>
      <phone>
            <type>Direct</type>
            <number>6200</number>
      </phone>
      <salary>
            <monthly>6700</monthly>
      </salary>
</employee>
```

数値比較と文字列比較の比較

ノード値に対して数値比較を実行すると、評価前に数値への変換が行われます。

遅い:

```
/ND/employee [phone/number =7000]
```

速い:

```
/ND/employee [phone/number ="7000"]
```

ノード値に対して文字列の等号比較を行えば、対応するタグ+データインデックスが直接検索されます。

例:

```
<employee id="Ricoh">
      <name>
            <first>Jon</first>
            <last>Ricoh</last>
      </name>
      <phone>
            <type>Direct</type>
            <number>7000</number>
      </phone>
      <salary>
            <monthly>6500</monthly>
      </salary>
</employee>

<employee id="Lexmark">
      <name>
            <first>Peter</first>
            <last>Lexmark</last>
      </name>
      <phone>
            <type>Direct</type>
            <number>6200</number>
      </phone>
      <salary>
            <monthly>6700</monthly>
      </salary>
</employee>
```

定数を使用した数値比較

ノード間の数値比較では、ステートメントの評価前に両方の値を数値に変換する必要があります。

遅い:

```
/ND/employee [number(phone/number) > number(salary/monthly)]
```

速い:

```
/ND/employee [number(phone/number) > 6700]
```

2 番目のクエリーはタグ+データインデックスに対するセット演算であり、最初のクエリーはシーケンス演算です。

例:

```
<employee id="Ricoh">
      <name>
            <first>Jon</first>
            <last>Ricoh</last>
      </name>
      <phone>
            <type>Direct</type>
            <number>7000</number>
      </phone>
      <salary>
            <monthly>6500</monthly>
      </salary>
</employee>

<employee id="Lexmark">
      <name>
            <first>Peter</first>
            <last>Lexmark</last>
      </name>
      <phone>
            <type>Direct</type>
            <number>6200</number>
      </phone>
      <salary>
            <monthly>6700</monthly>
      </salary>
</employee>
```

出力サイズの抑制

HTTP のデータ転送は、データ量が少ないほど効率が良くなります。一般ステップやコンストラクタ構文を使用して、必要なノードだけをターゲットとするようにしてください。

遅い:

```
/ND/employee
```

速い:

```
/ND/employee/ (name/last, salary/monthly)
```

結果セットが小さいほど、ネットワーク転送も効率的になります。クライアント側の処理もより軽量になります。

例:

```
<employee id="Ricoh">
      <name>
            <first>Jon</first>
            <last>Ricoh</last>
      </name>
      <phone>
            <type>Direct</type>
            <number>7000</number>
      </phone>
      <salary>
            <monthly>6500</monthly>
      </salary>
</employee>

<employee id="Lexmark">
      <name>
            <first>Peter</first>
            <last>Lexmark</last>
      </name>
      <phone>
            <type>Direct</type>
            <number>6200</number>
      </phone>
      <salary>
            <monthly>6700</monthly>
      </salary>
</employee>
```

基本ロケーションパス

概要

NeoCore XMS のクエリー言語は XQuery のサブセットであり、そのクエリーでは XPath のカスタムサブセットを使用します。Neocore XMS は両方のサブセットをサポートします。XPath は整形式の XML 文書内でデータを検索するために設計された標準規格であり、この標準規格を使用するのが、XML データのクエリーを行うための自然な方法と言えます。ここでは、NeoCore XMS のクエリー、構文、演算子について説明します。例として使用する XML 文書は、NeoCore インストールディレクトリの neoxml¥sample¥xml サブディレクトリに用意されています。

参考資料

「***Java API***」、「***C++ API***」、「***COM API***」、「***HTTP API***」を参照してください。これらのセクションでは、NeoCore XMS のアプリケーションプログラミングインターフェースについて解説しています。

『NeoCore XMS System Administration Guide』では、NeoCore XMS の管理と構成について説明しています。

XPath の公式の資料は、http://www.w3.org/TR/xpath20/にあります。

XQuery の公式の資料は、http://www.w3.org/TR/xquery/にあります。

XMS クエリー

NeoCore XMS クエリーでは、UNIX や MS-DOS でファイルへのナビゲーションパスを指定するのとよく似た方法で、ノードセットへのナビゲーションパスを指定します。特定のフォルダやファイルを参照するファイルパスとは違って、クエリーの場合は同じ名前やタグのノードセットが対象になります。

XPath は文書をノードのツリーとしてモデル化します。この場合のノードは、要素か要素の属性です。クエリーはノードのセットを参照します。クエリーに対する応答として、NeoServer は該当する文書やノードを次のような標準タグペアで囲んで返します。

```
<Query-Results></Query-Results>
```

このセット自体、整形式の XML です。このノードセットは、クエリーの**ターゲット**です。ターゲットを絞り込むには、セットのフィルタとしてクエリーに述部を指定します。

XML では、文書の始まりと終わりを定義するルートレベルのタグを使用します。XPath では必ずルートコンテキストから始まる修飾パス表記を使用します。NeoCore XMS は、格納する各文書を<ND></ND>（neodocument）タグペアに入れるので、下部構造が異なる場合でも、すべての文書は同じルートコンテキストになります。

XMS 規則

ルートコンテキストの表記は常にスラッシュ（/）になります。

NeoCore XMS のパスは、/ND、//、関数呼び出し、コマンドのいずれかで始まり、XPath 式が続きます（ただし、関数呼び出しは文書やデータのクエリーパスのみになります。

- XPath 式の各ノードは、XML 文書内に出現する順番どおりに記述し、後ろにスラッシュを置きます。
- 最後のノードは、後ろのスラッシュなしで指定します。これがターゲットノードになります。
- XPath では大文字小文字の区別があります。
- XPath では先頭と末尾の空白は無視されます。
- 1 つのクエリーで、以下のものを取り出すことができます。
  - 1 つ以上の完全な XML 文書。
  - 1 つ以上の XML 文書からの要素のセット。
  - ターゲットについての情報（**count** など）。
- 格納時に、NeoCore XMS は一般エンティティ用の置き換えを行い、その置き換えた値でインデックスを構築します。
- マークアップ文字のエスケープは、以下のような XML 仕様に基づいています。文書内では事前定義のエンティティを使用してください。

表示される文字	XML **エンティティ**
&	&
<	<
>	>
'	'
"	"

XMS での XPath ターゲットの使用

すべての NeoCore XMS 文書は NeoDocument タグ<ND></ND>の中に入っているので、特定のターゲットに対するほとんどのクエリーは/ND で始まります。

例えば、`/ND/letter/cc-list/name/last-name` という式は、`/ND/letter/cc-list/name` というノードパスを持った親要素の下にあるすべての`<last-name>`要素を参照します。

ただし、ここで注意しなければならないのは、このクエリーはノードの内容データを参照していないという点です。単に、ノードのすべての出現例に対するパスを指定しているだけです。

上記の例の XPath 式では、ノードに対する完全修飾パスを指定しています。XML 文書の構造を家族の系統図と見なせば、完全修飾パスはルートからターゲットに至るまでのそれぞれの子ノードを正しいナビゲーション順で指定したものということになります。

クエリーには、部分修飾パスを含めることもできます。その場合、ターゲット要素の親要素のタグ構造はそれぞれ異なる場合があります。部分修飾パスには、「この位置におけるゼロ個以上の任意のノード」を意味する二重スラッシュ「//」が少なくとも 1 つ含まれます。例えば、`/ND//name/last-name` は`<name>`という親要素を持つ`<last-name>`のインスタンスをすべて返します。

XPath 指定の特徴は、descendant 軸ステップを使用したこの種のパス式にあります（一般に、軸とはパスが 1 つのノードから別のノードにナビゲートするためのさまざまな方法のことをいいます）。

位置クエリー

位置クエリーでは、XPath 関数を使用し、位置に基づいて特定のノードを検索します。文書内の最後のノードを取り出すことも、最後のノードとの相対位置に基づいてノードを取り出すこともできます。

position()	位置条件に合致するノードを返します。
last()	ターゲットセット内の最後のノードを返します。

たくさんの項目がある注文書から 5 番目の項目を取り出す場合は、次の位置クエリーを使用します。

```
（/ND/item）[position()=5]
```

位置クエリーの他の例:

```
/ND/item[last()]
/ND/item[position()=last()-1]
/ND/item[5]
/ND/item[position()>10 and position()<= 20]
```

XPath 述部の使用

述部とは、ターゲットセットのフィルタに使用するブール値ステートメントです。述部を使用することにより、クエリーインターフェースが大幅に強化されます。この種のステートメントはセット内の各要素に適用され、そのセットは、真になる要素と偽になる要素の 2 つのサブセットに分割されます。このような用途のステートメントのことを述部といいます。

位置クエリー用の XPath 関数に加えて、NeoCore XMS は述部内での等価演算子、比較演算子、論理演算子、文字列関数をサポートしています。ワイルドカード文字列引数の使用は、等価演算子を使用する述部に限定されます。以下の表に関数と演算子をまとめます。

=	等しい
!=	等しくない
<	より小さい
<=	以下
>	より大きい
>=	以上
not()	論理否定
and	および
or	または
()	グループ化

*	ゼロ個以上の任意の文字（文字列定数に隣接する場合）
" "または' ' （二重引用符または単一引用符）	文字列定数

注: このような意味でのアスタリスク（*）は、NeoCore XMS の独特の使用法です。

注: XPath 述部は角括弧で囲みます。XMS ではどんなブール式でも使用できます。パスそれ自体は、ノードが存在するかどうかを検査します。演算子と 2 番目の引数がない場合は、要素または属性が存在するときに述部は真になります。述部式では、/ND/person[salary>5.0 * deductions]のような対象の広い式も可能です（各「person」の 2 つの子を比較します）。

式は常に、ブール式、and、or、not、比較、文字列、数値、関数などになります。

述部のパスは常に部分修飾になり、述部の直前の相対パスを拡張する役割を果たします。例えば、`/ND/letter[ID="M12345"]//name` という書き方で先ほどの例を完全 XPath 式に組み込むことができます。

この場合の **ID** は`/ND/letter` パスを拡張する部分修飾パスです。述部の最初の引数は、`/ND/letter/ID` にある要素データを参照します。

述部内の要素や属性の参照は相対パスです。述部は、実際にはパスではない式に適用することもできます。

比較における引数は、数字、リテラル文字列、または**"**Xpriori, LLC**"*** などのワイルドカード文字列のいずれかである文字パターンです。クエリー内のどの部分で使用する場合でも、文字列や数字には引用符を付ける必要があります。二重引用符を含む文字列を使用する場合は、全体を単一引用符で囲みます。

全体が数字（とピリオド）で構成される引数は 10 進数として解釈され、述部の評価の前に浮動小数点表現に変換されます。そうでない場合、2 番目の引数はリテラルかワイルドカード文字列になります。

アスタリスクの前後の文字列はワイルドカードと解釈されます。アスタリスクは`[company="US Steel"*]` のように引用符の外側に付ける必要があります。

アスタリスクは、ワイルドカードとして実際の文字列を閉じ込めてしまうための演算子で、「ゼロ個以上の任意の文字」を意味します。

この演算子は文字列にだけ適用されるので、引数"123"*は「123 で始まるすべての文字列」を意味し、123ABC のようなノード値と一致します。引数*"123"*は「123 を含むすべての文字列」を意味し、876123951 のようなノード値と一致します。アスタリスクを 2 つの文字列の間に使用することはできず、"123"*"456"は無効です。

述部の構文

NeoCore XMS は述部クエリーの構文としていくつかの変化形をサポートしており、このセクションではそのことについて説明します。

- ターゲットと述部パスで同じノードを参照します。
  例:

```
/ND/letter/signature/name/last-name[.="Fictional"]
/ND/letter[@ID]
```

- ターゲットと述部パスで別々のノードを参照し、ターゲットノードを述部の前に指定します。
  例:

```
/ND/letter/signature/name[last-name="Fictional"]
/ND/letter[date/@year="2001"]
/ND/letter[company!="NeoCore Inc."]
```

- ターゲットと述部パスで別々のノードを参照し、ターゲットノードを述部の後に指定します。
  例:

```
/ND/letter/signature/name[last-name="Fictional"]/first-name
/ND/letter[date/@year="2001"]/subject
```

- 上記の書き方を組み合わせて、2 つ以上の述部を指定します。
  例:

```
/ND/letter[return-address/state="WA"]/date[@year="2001"]
/ND/letter[@ID="M"]body/para[.=*"XML"*]
```

これらの例が示しているように、述部は、ターゲットノードのサブセットを選択するために使用します。

例えば、letter.xml 文書の<return-address>要素を抽出するとしましょう。そのためには、属性 ID="M12345"を持った<ND><letter>要素に検索を限定し、参照先の要素の直後に角括弧で囲んだ述部式を使用します。
`/ND/letter[@ID="M12345"]/return-address`

ここまでの述部構文は、属性の内容によってクエリーを限定する例です。要素の内容によってクエリーを限定することもできます。

```
/ND/letter[company="NeoCore Inc."*]
```

この述部の左の角括弧は 2 つの役割を果たします。つまり、述部の始まりを定義すると共に、パスの区切りにもなっています。要するに、述部の完全修飾パスは`/ND/letter/company` なので、この述部はターゲットの子を参照しているということです。XPath 仕様では、これを child 軸と呼んでいます。

今度は、述部で子ではなくターゲットそのものを参照するとしましょう。その場合には、このようにします。

```
/ND/letter/company[.="NeoCore Inc."*]
```

この述部のパスではピリオドを使用しています。このピリオドは述部の直前のターゲットを参照します。XPath 仕様では、これを self 軸と呼んでいます。

述部のタイプ

NeoCore XMS はクエリー内の述部として以下のタイプの述部をサポートしています。

- ノードの存在（バッテリ容量インジケータ付きのすべての携帯電話など）。
- ノード等式（色属性が赤であるすべての車、General Motors 製以外のすべての車など）
- 比較演算子を使用したノード不等式（価格が 25000 ドル未満のすべての車など）
- 複合述部（ホンダまたはトヨタ製のすべての 4 ドアセダン車など）

ノードの存在

空の要素ノードを使用して、ある特性の存在を示すことがあります。

例えば、携帯電話のカタログに、以下のように数多くの機能を列挙したモデルが掲載されているとします。

```
<features>
   <Authentication />
   <RingerAndEarpieceVolumeControl />
   <BatteryStrengthIndicator />
   <AudibleKeypadControls />
   <One-YearPartsAndLaborWarranty />
   <AnyKeyAnswer />
   <OneTouchDialing />
   <CallerID />
   <AudibleElapsedTalkTimer />
   <RapidCharger />
</features>
```

特定の機能セットを持つ携帯電話モデルを絞り込むには、child 軸に対象の要素が存在するかどうかを述部で検査します。その述部では、存在するかどうかを検査するノードのパスだけを引数として指定します。以下の例では、複合述部を使用して、RapidCharger 機能と OneTouchDialing 機能の両方を持つ携帯電話モデルをすべて抽出します。

```
/ND/cellphone-catalog/brand/model[features/RapidCharger and
features/OneTouchDialing]
```

要素と違い、属性には常にデータ値があります。しかし、値を指定しないクエリーによって属性が存在するかどうかを検査できます。

```
/ND/letter[@ID]
```

このクエリーは以下の結果を返します。

```
<Query-Results>
      <letter ID="M12345">
...
      </letter>
</Query-Results>
```

要素または属性の等式

述部の最も一般的な使い方は、ノードの関連データをリテラル値かワイルドカード値と比較することです。等式述部は、1 番目の引数のパス、等式演算子、2 番目の引数という順に書いていきます。演算子が″=″の場合、クエリーは値が対象の文字列と等しいノードを返します。演算子が″!=″の場合は、値が対象の文字列と等しくないノードを返します。

例えば、以下のクエリーを実行するとしましょう。

```
/ND/letter/signature/name[last-name="Fictional"]
```

以下の結果が返されます。

```
<Query-Results>
   <name>
        <first-name>John</first-name>
        <last-name>Fictional</last-name>
   </name>
</Query-Results>
```

また、次のクエリーを実行します。

```
/ND/letter/cc-list/name[last-name!="Trade"]
```

以下の結果が返されます。

```
<Query-Results>
    <name>
         <first-name>Leah</first-name>
         <last-name>Wong</last-name>
    </name>
</Query-Results>
```

要素または属性の不等式

述部を使用して、ノードの関連データがリテラル値より小さいかどうか、または大きいかどうかを判断することもできます。不等式述部は、1 番目の引数のパス、比較演算子、2 番目の引数という順に書いていきます。2 番目の引数は、数字かリテラル文字列である必要があります。ワイルドカードは使用できません。

このタイプのクエリーは、指定の比較が真になるノードを探します。例えば、以下の XPath コマンドを実行するとしましょう。

```
/ND/periodic_table/element[atomic_weight>="200"]/name
```

以下の結果が返されます。

```
<Query-Results>
      <name>actinium</name>
      <name>americium</name>
      <name>astatine</name>
      <name>bohrium</name>
...
</Query-Results>
```

文字列比較と数値比較の比較

[x > "200"]の結果として"99"は該当します。

[x > 200]の結果として"99"は該当しません。

複合述部

複合述部では、論理演算子とグループ化演算子()を使用して 2 つ以上の単純述部を組み合わせます。論理演算子の両側にはスペースを入れて、引数との間を区切ります。式全体を角括弧で囲んでください。

目的の結果を得るには、クエリーに複数の条件を指定しなければならないことがしばしばあるので、そのようなときに複合述部は便利です。例えば、Ameritech か AirTouch がサービスを提供しているニッカド電池("nichkel-cadmium")式の携帯電話モデルを検索するとしましょう。

次のサンプルクエリーでは、目的の結果を得るために 3 つの述部とグループ化演算子を使用します。

```
/ND/cellphone-catalog/brand/model[battery/@type="nickel-cadmium" and
(service-providers/provider="Ameritech" or service-providers
/provider="AirTouch")]
```

評価は左から右に行われるという動作を利用すれば、このサンプルのステートメントを並び替えて括弧を不要にできます。

```
/ND/cellphone-catalog/brand/model[service-providers/
provider="Ameritech" or service-providers/provider="AirTouch" and
battery/@type="nickel-cadmium"]
```

しかし、通常は括弧を使用して、単純述部の場所と評価の優先順位を明示するほうが望ましいと言えます。

```
/ND/cellphone-catalog/brand/model[(service-providers/provider
="Ameritech" or service-providers/provider="AirTouch") and
(battery/@type="nickel-cadmium")]
```

注: and は or よりも優先されます。例えば、x or y and z は、(x or (y and z))とグループ化されます。

parent 軸の使用

NeoCore XMS では、XPath クエリーに parent 軸つまり「..」のメカニズムを使用した場合、その記述が述部の外側か内側かに応じて機能が異なります。

- 述部の外側: この種のクエリーは、現在のノードの親に戻ります。

parent 軸の例 1:

```
/ND/CATALOG/CD/TITLE[.="Stop"]/..
```

（この例 1 では、「TITLE」が現在のターゲットであり、「CD」がターゲットの親です。）

この例は、子要素の 1 つとして「**STOP**」というタイトルを持つ CD を返します。

ただし、例 1 のクエリーの一番きれいで効率的な書き方は次のとおりです。

```
/ND/CATALOG/CD[TITLE="Stop"]
```

- 述部の内側: この種のクエリーは、データ選択やノード検証のためにターゲットを操作しますが、結果を返すレベルは変更しません。

parent 軸の例 2:

```
/ND/CATALOG/CD/TITLE[../ARTIST="Sam Brown"]
```

「..」の複数インスタンス

以下の例を取り上げましょう。

parent 軸の例 3:

```
/ND/CATALOG/CD/TITLE/../../CD/ARTIST
```

これは次のクエリーと同じ結果を返します。

```
/ND/CATALOG[CD/TITLE]/CD/ARTIST
```

注: 上記の例 3 では、クエリー内に「..」のインスタンスが複数あります。「..」のインスタンスは述部の外側でも内側でも、突き合わせ対象のクエリーパスをさかのぼるノードがある分までいくつでも使用できます。例 3 では、「..」の最初のインスタンスは（クエリーの左端から数えて）3 番目の「CD」、2 番目のインスタンスは 2 番目の「CATALOG」を参照します。

注: parent 軸の使用は、クエリーを書き直したくない場合やほかに方法がない場合（つまり、データ構造に依存する方法しかない場合）に限定してください。

descendant 軸の使用

descendant 軸クエリーでは、「//」のインスタンスを使用します。「//」は、子孫と呼ばれる XPath 上のノードと突き合わせるためのショートカットです。

注: このショートカットはパフォーマンスに悪影響を与えます。使用するのは、クエリー対象の文書の構造が良く分からない場合だけにしてください。

descendant 軸の例 1:

```
/ND/letter//last-name
```

descendant 軸の例 2:

次の XPath サンプルを取り上げましょう。

```
<ND>
<A>
      <B>
      <C>"These are the times </C>
      </B>
</A>
<F>
      <C>that try men's souls..."</C>
</F>
</ND>
```

次のクエリーを実行します。

```
/ND//C
```

上記の descendant 軸の例 2 のサンプルからは、「C」のインスタンスが両方返されます。この場合の「F」は「A」の兄弟レベルです。

ちなみに、「A」の下にある「C」からデータが欲しい場合は、次のクエリーを使用できます。

```
/ND//C[.="T"*]
```

このクエリーは、パスによる検索に加えて、「*」文字でデータに対するワイルドカードの突き合わせも行います。

XQuery

概要

Windows フォームや Web フォームを使用するコンシューマにビジネスデータを配信するための望ましい方法として XML Web サービスが定着するにつれ、分析やレポートのための XML 文書の変換処理で重要な役割を果たすようになるのが XQuery です。NeoCore XMS は、クエリー言語として XQuery のサブセットをサポートしています。

XQuery の基本的な目的は、あたかもデータベースにアクセスするかのようにして、実際の XML 文書や仮想の XML 文書のコレクションにアクセスすることです。ただし、XQuery 式は行と列のクエリー結果セットを返すのではなく、クエリー条件に一致するノードを含んだ XML 文書を返します。XQuery には、SQL の集計関数に対応する sum()、avg()、min()、max()の各関数が用意されています。さらに、document()、distinct-nodes()、distinct-values、substring()などの文書関連関数もあります。

パーサーは実行時に数値のノード値を検出します。

XQuery にはいくつかのタイプの式が用意されていますが、データ指向の開発者にとって最も重要なのは FLWR 式です（FLWR はフラワーと読みます）。FLWR は、FOR-LET-WHERE -RETURN という XQuery キーワードの略語です。FLWR 式は、SQL の SELECT columnlist FROM tablename WHERE criteria ステートメントに相当します。XQuery では、ノードとその値の指定に XPath のサブセットを使用します。

ここでは、NecoCore XMS の XQuery 言語サポートと XQuery 関数サポートを取り上げます。

注:XMS の XQuery コマンドはすべて、XMS コンソールと API から実行できます。

XQuery 言語サポート

ここでは、XQuery サポートのために XMS に用意されているすべての構文を取り上げます。それぞれの機能について、以下の文書を使用した例も載せています。この文書は XMS リリースに入っているサンプル文書のサブセットです。

サンプル文書

```
<Query-Results>
      <Item>
            <Value>
                  <COMPANY>Columbia</COMPANY>
            </Value>
            <Count>1</Count>
      </Item>
      <Item>
            <Value>
                  <COMPANY>A and M</COMPANY>
            </Value>
            <Count>1</Count>
      </Item>
      <Item>
            <Value>
                  <COMPANY>Atlantic</COMPANY>
            </Value>
            <Count>1</Count>
      </Item>
</Query-Results>
```

基本式

基本式は、この言語の基本要素です。リテラル、変数、関数呼び出しを含み、演算子の優先順位を制御するために括弧を使用できます。

リテラル

リテラルとは、アトミック値の直接的な構文表現です。XQuery では、文字列リテラルと数値リテラルという 2 種類のリテラルをサポートしています。

リテラル名		
Literal	::=	NumericLiteral | StringLiteral
NumericLiteral	::=	IntegerLiteral | DecimalLiteral | DoubleLiteral
IntegerLiteral	::=	[0-9]+
DoubleLiteral	::=	(("." [0-9]+) | ([0-9]+ ("." [0-9]*)?)) ([e] | [E]) ([+] | [-])? [0-9]+
StringLiteral	::=	(["] ((" ") | [^"])* ["]) | (['] ((' ') | [^'])* ['])

文字列リテラルの値は、プリミティブデータ型 string（文字列）の 1 つの項目を内容とするシングルトンシーケンスです。値の文字列は引用符で囲まれています。

「.」も「e」や「E」という文字も含まない**数値リテラル**の値は、データ型 integer（整数）の 1 つの項目を内容とするシングルトンシーケンスです。値は、データ型 integer の規則に従って数値リテラルを解析した結果として得られます。「.」を含むものの「e」や「E」という文字を含まない数値リテラルと、「e」や「E」の文字を 1 個だけ含む数値リテラルの値は、プリミティブデータ型 double（倍精度）の 1 つの項目を内容とするシングルトンシーケンスです。値は、データ型 double の規則に従って数値リテラルを解析した結果として得られます。

以下にリテラル式の例を示します。

- "12.5"は、文字「1」、「2」、「.」、「5」を内容とする文字列を表します。
- 12 は整数値 12 を表します。
- 12.5 は倍精度値 12.5 を表します。
- 125E2 は倍精度値 12,500 を表します。

データ型変換

式のオペランドのデータ型が正確に一致していなくてよい場合もあります。例えば、関数のパラメータ値と戻り値のデータ型は決まっていますが、基本的な変換処理によって、ノードからのアトミック値の抽出、数値の昇格、データ型のない値の暗黙キャストなどを実行できます。

どんなデータ型でも受け付ける演算子（比較など）のデフォルトは通常、文字列です。

変数

変数は**評価コンテキスト**内で、変数の NCName のバインド先の値として評価されます。

XML 文書では、Qname（qualified name: **修飾名**）として指定できる名前があります。以下にその定義を示します。

NCName の定義

letter = [X_YYYYZZZZ]。x は文字か下線にします。YYYYZZZZ には、下線、ダッシュ、文字、数字など、XML 仕様で文字または数字として定義されているあらゆるものを使用できます。

QName の定義

[6] QName ::= (Prefix ':')?LocalPart

[7] Prefix ::= NCName

[8] LocalPart ::= NCName

Prefix は修飾名の名前空間プレフィックスであり、名前空間宣言の名前空間 URI 参照に関連付ける必要があります（LocalPart は修飾名のローカル部分です）。

注: プレフィックスは、名前空間名のプレースホルダーとしてのみ機能します。スコープが文書の外に広がっている名前をアプリケーションによって組み立てる場合は、プレフィックスではなく名前空間名を使用するべきです。

変数の QName がバインドされていない場合、変数の値はエラー値になります。変数をバインドするには、for **式**や**数量条件設定式**の句を使用するか、関数呼び出しを実行します。関数呼び出しは、関数本体の実行前に関数の仮パラメータに値をバインドします。

名前		
Variable	::=	"$" NCName

括弧の付いた式

複数の演算子がある式では、評価順序を指定するために括弧を使用できます。例えば、(2 + 4) * 5 という式は 30 になります。括弧の付いた式(2 + 4)がまず評価され、その結果に 5 を掛けるからです。括弧がない 2 + 4 * 5 という式は 22 になります。乗算演算子は加算演算子よりも優先順位が高いからです。

関数呼び出し

関数呼び出しの場合は、QName の後にゼロ個以上の式を括弧内に入れて指定します。この QName は、**静的コンテキスト**の**インスコープ関数**の名前と一致している必要があります。括弧内の式では、関数呼び出しの引数を提供します。引数の数は、関数シグネチャ内の仮パラメータの数と一致している必要があります。そうでない場合は、静的エラーが発生します。

名前		
FunctionCall	::=	(QName | "document") "(" (Expr ("," Expr)*)? ")"

関数呼び出し式は以下のように評価されます。

- 各引数式が評価され、引数値が生成されます。
- 以下に挙げる関数の変換ルールに基づいて、各引数値が、対応する関数パラメータに関して宣言されているデータ型に変換されます。
- 関数が組み込み関数の場合は、変換された引数値に基づいて関数が実行されます。結果は、関数の戻り値に関して宣言されているデータ型の値になります。
- 関数がユーザー定義関数の場合は、変換された引数値が関数の仮パラメータにバインドされ、関数本体が実行されます。関数本体によって返された値は、関数の変換ルールに基づいて、関数の戻り値に関して宣言されているデータ型に変換されます。

引数値または戻り値は、**関数の変換ルール**に基づいて、それぞれ要求されているデータ型、つまり関数パラメータまたは戻り値に関して宣言されているデータ型に変換されます。その要求されているデータ型は、SequenceType として指定します。関数の変換ルールは以下のとおりです。

要求されているデータ型が AtomicType の場合:

- 倍精度値を要求する関数で、引数の値が倍精度値でない場合、その引数は暗黙的に倍精度値に変換されます。

要求されているデータ型がシーケンスか省略可能な AtomicType の場合:

- 値の内容がノードの場合、そのノードは自身のデータ型付き値によって置き換えられます。結果のシーケンスのカーディナリティが、要求されているデータ型のカーディナリティと一致しない場合は、データ型例外が発生します。それ以外の場合は、AtomicType の変換ルールに基づいて、各項目が要求されている AtomicType に変換されます。

要求されているデータ型がそれ以外の SequenceType の場合:

- **SequenceType の突き合わせ**によって、値のデータ型が要求されているデータ型と一致するかどうかが検査されます。一致していない場合は、関数呼び出しからエラー値が返されます。

コメント

XQuery コメントを使用して、注釈を付けることができます。コメントは、説明のためだけの構造体であり、式の処理には影響しません。

名前		
ExprComment	::=	{--+--}

コメントは、式内部の主要なトークンの前後に使用できます。

例 : {-- this is a comment --}

パス式

パス式には、ステップ、軸、ノードテスト、修飾語があります。

ステップ

名前		
StepExpr	::=	Predicates
Step	::=	(Axis NodeTest) | AbbreviatedStep

ステップとは、一連のノードを、文書の順に、重複なしで返す式です。通常は、パス式の内部で使用します。コンテキストノードを起点にして、事前定義の軸を経由して到達できるノードにナビゲートし、その到達可能なノードのサブセットを抽出します。ステップには 3 つの部分があります。

1. **軸**。これには、このステップで抽出するノードとコンテキストノードとの関係を指定します。要するに、ステップの「移動方向」と考えることができます。
2. **ノードテスト**。これには、このステップで抽出するノードの種類や名前を指定します。
3. ゼロ個以上の**述部**。このステップで抽出するノードのシーケンスをさらに変更します。

軸 - XPath 軸の明示的な構文（axis::name）

name -

名前を指定して、コンテキストノードを起点とした現在の軸に存在する要素を抽出します。

例:

```
/ND/CATALOG/CD/TITLE
```

結果:

```
<TITLE>Empire Burlesque</TITLE>
<TITLE>Stop</TITLE>
<TITLE>The dock of the bay</TITLE>
```

@name -

attribute::name の省略形です。コンテキストノードの属性を抽出します。

例:

```
/ND/CATALOG/CD/@ID
```

結果:

```
<CD ID="8" />
<CD ID="2" />
<CD ID="5" />
```

child 軸

コンテキストノードの子要素を抽出します。

例:

```
/ND/CATALOG/CD
```

結果:

セット全体

parent 軸

コンテキストノードの親を抽出します。

例:

```
/ND/CATALOG/CD/TITLE/..
```

結果:

セット全体

self 軸

コンテキストノードを抽出します。

例:

```
/ND/CATALOG/CD/ARTIST[.="Bob Dylan"]
```

結果:

```
<ARTIST>Bob Dylan</ARTIST>
```

attribute 軸

コンテキストノードの属性を抽出します。

例:

```
/ND/CATALOG/CD[@ID="2"]
```

結果:

```
<CD ID="2">
      <TITLE>Stop</TITLE>
      <ARTIST>Sam Brown</ARTIST>
      <COUNTRY>UK</COUNTRY>
      <COMPANY>A and M</COMPANY>
      <PRICE>8.90</PRICE>
      <YEAR>1988</YEAR>
</CD>
```

descendant 軸

コンテキストノードの子孫である要素をすべて抽出します。

例:

```
/ND/CATALOG/descendant::TITLE
```

結果:

```
<TITLE>Empire Burlesque</TITLE>
<TITLE>Stop</TITLE>
<TITLE>The dock of the bay</TITLE>
```

descendant-or-self 軸

子孫要素すべてとコンテキストノード自体を抽出します。

例:

```
/ND/CATALOG/CD[descendant-or-self::node()="Stop"]
```

結果:

```
<CD ID="2">
      <TITLE>Stop</TITLE>
      <ARTIST>Sam Brown</ARTIST>
      <COUNTRY>UK</COUNTRY>
      <COMPANY>A and M</COMPANY>
      <PRICE>8.90</PRICE>
      <YEAR>1988</YEAR>
</CD>
```

ノードテスト

//表記（パスのワイルドカード）

/descendant-or-self::node()/の省略形。

例:

```
/ND//CD[TITLE="Stop"]
```

結果:

```
<CD ID="2">
  <TITLE>Stop</TITLE>
  <ARTIST>Sam Brown</ARTIST>
  <COUNTRY>UK</COUNTRY>
  <COMPANY>A and M</COMPANY>
  <PRICE>8.90</PRICE>
  <YEAR>1988</YEAR>
</CD>
```

アスタリスク（*）-

コンテキストノードを起点にした現在の軸に存在するすべての要素を抽出します。

例:

```
/ND/CATALOG/CD/*
```

結果:

```
<TITLE>Empire Burlesque</TITLE>
<ARTIST>Bob Dylan</ARTIST>
<COUNTRY>USA</COUNTRY>
<COMPANY>Columbia</COMPANY>
<PRICE>10.90</PRICE>
<YEAR>1985</YEAR>
<TITLE>Stop</TITLE>
<ARTIST>Sam Brown</ARTIST>
<COUNTRY>UK</COUNTRY>
<COMPANY>A and M</COMPANY>
<PRICE>8.90</PRICE>
<YEAR>1988</YEAR>
<TITLE>The dock of the bay</TITLE>
<ARTIST>Otis Redding</ARTIST>
<COUNTRY>USA</COUNTRY>
<COMPANY>Atlantic</COMPANY>
<PRICE>7.90</PRICE>
<YEAR>1987</YEAR>
```

例:

```
/ND/*/*/TITLE
```

結果:

```
<TITLE>Empire Burlesque</TITLE>
<TITLE>Stop</TITLE>
<TITLE>The dock of the bay</TITLE>
```

例:

```
/ND/CATALOG/CD/TITLE[self::*="Stop"]
```

結果:

```
<TITLE>Stop</TITLE>
```

@* -

attribute::*の省略形。

node() -

コンテキストノードを起点とした現在の軸に存在する要素か属性のいずれかを抽出します。

例:

```
/ND/CATALOG/CD/TITLE/parent::node()
```
または`/ND/node()/CD`

text() -

コンテキストノードのテキスト値を返します。

例:

```
/ND/CATALOG/CD[TITLE="Stop"]/descendant-or-self::text()
```

結果:

```
<Query-Results>Stop Sam Brown UK A and M 8.90 1988</Query-Results>
```

例:

```
/ND/CATALOG/CD[descendant-or-self::text()="Sam Brown"]
```

結果:

```
<CD ID="2">
  <TITLE>Stop</TITLE>
  <ARTIST>Sam Brown</ARTIST>
  <COUNTRY>UK</COUNTRY>
  <COMPANY>A and M</COMPANY>
  <PRICE>8.90</PRICE>
  <YEAR>1988</YEAR>
</CD>
```

ドット（.）

現在のノードを示します。

例:

```
/ND/CATALOG/CD/PRICE[. < 8.00]
```

結果:

```
<PRICE>7.90</PRICE>
```

述部

述部。シーケンス内の各ノードにフィルタを適用して、ノードシーケンスを絞り込みます。

名前		
Qualifiers	::=	("[" Expr "]")*

述部

述部とは、角括弧で囲んだ式であり、位置パスステップ、別の述部、または他の式の後に記述します。ドット（.）以外のパスを述部に指定する場合、述部の前の式はノードセットである必要があります。述部は、入力によって生成されたノードセットに対するフィルタとして機能し、コンテキストノードのサブセットだけを抽出します。コンテキストノードごとに述部式の評価が行われ、必要に応じて結果がブール式に変換され、そのブール値が真である場合にコンテキストノードはフィルタを通過します。

通常、述部には相対パスを記述します。その相対パスは、コンテキストノードごとに評価されます。述部に絶対パスを記述すると、パフォーマンスが大きく低下する場合があります。絶対パスがコンテキストノードごとに評価し直される可能性があるからです。

整数（または整数のシーケンス）と評価される述部式は、[(position)=expr]と同じように機能します（例えば、/no//a［1］と記述すれば、1 を持つ各要素の最初の子「a」が抽出されます）。

述部内の最初のステップの先頭に/を使用することはできません（これは文書のルートになってしまいます）。子の名前で始めてください。次の例は**間違い**です。

`"/ND/CATALOG/CD[/TITLE="Stop"]"`（これはエラーを返します）

例:

```
/ND/CATALOG/CD[TITLE="Stop"]
```

結果:

```
<CD ID="2">
      <TITLE>Stop</TITLE>
      <ARTIST>Sam Brown</ARTIST>
      <COUNTRY>UK</COUNTRY>
      <COMPANY>A and M</COMPANY>
      <PRICE>8.90</PRICE>
      <YEAR>1988</YEAR>
</CD>
```

一般ステップ

式がノードセットとして評価される場合、ステップとして、括弧で囲んだ式を記述できます。例えば、/ND/x/y/(a union b except c)は有効なパスです。一般ステップは、述部の場合と同じように、コンテキストノードごとに評価されます。一般ステップの後には、さらにステップを置くことも、述部を置くこともできます。

例:

```
/ND/CATALOG/CD/(TITLE | ARTIST)
```

結果:

```
<TITLE>Empire Burlesque</TITLE>
<ARTIST>Bob Dylan</ARTIST>
<TITLE>Stop</TITLE>
<ARTIST>Sam Brown</ARTIST>
<TITLE>The dock of the bay</TITLE>
<ARTIST>Otis Redding</ARTIST>
```

シーケンス式

XQuery は、シーケンスを作成して結合するための演算子をサポートしています。**シーケンス**とは、ゼロ個以上の項目を順序に従って並べた集合です。ここで言う**項目**は、アトミック値の場合もあれば、ノードの場合もあります。項目は、その項目を内容とする長さ 1 のシーケンスと同一です。シーケンスのネストはできません。例えば、1 と(2, 3)という値を結合すると、シーケンスは(1, 2, 3)になります。

シーケンスの結合

3 つの演算子（Union、Intersect、Except）はいずれもノードだけに適用できます。

Union 演算子

2 つのシーケンス引数の和集合を返します（重複は返しません）。

例:

```
/ND/CATALOG/CD[PRICE < 10.00] union /ND/CATALOG/CD[PRICE > 8.00]
```

結果:

```
<CD ID="8">
      <TITLE>Empire Burlesque</TITLE>
      <ARTIST>Bob Dylan</ARTIST>
      <COUNTRY>USA</COUNTRY>
      <COMPANY>Columbia</COMPANY>
      <PRICE>10.90</PRICE>
      <YEAR>1985</YEAR>
</CD>
<CD ID="2">
      <TITLE>Stop</TITLE>
      <ARTIST>Sam Brown</ARTIST>
      <COUNTRY>UK</COUNTRY>
      <COMPANY>A and M</COMPANY>
      <PRICE>8.90</PRICE>
      <YEAR>1988</YEAR>
</CD>
<CD ID="5">
      <TITLE>The dock of the bay</TITLE>
      <ARTIST>Otis Redding</ARTIST>
      <COUNTRY>USA</COUNTRY>
      <COMPANY>Atlantic</COMPANY>
      <PRICE>7.90</PRICE>
      <YEAR>1987</YEAR>
</CD>
```

Intersect 演算子

両方のシーケンス引数に存在するノードを返します（つまり、重複だけを返します）。

例:

```
/ND/CATALOG/CD[PRICE < 10.00] intersect /ND/CATALOG/CD[PRICE > 8.00]
```

結果:

```
<CD ID="2">
      <TITLE>Stop</TITLE>
      <ARTIST>Sam Brown</ARTIST>
      <COUNTRY>UK</COUNTRY>
      <COMPANY>A and M</COMPANY>
      <PRICE>8.90</PRICE>
      <YEAR>1988</YEAR>
</CD>
```

Except 演算子

最初のシーケンスに存在して 2 番目のシーケンスに存在しないノードを返します。

例:

```
/ND/CATALOG/CD[PRICE < 10.00] except /ND/CATALOG/CD[ARTIST = "Sam Brown"]
```

結果:

```
<CD ID="5">
      <TITLE>The dock of the bay</TITLE>
      <ARTIST>Otis Redding</ARTIST>
      <COUNTRY>USA</COUNTRY>
      <COMPANY>Atlantic</COMPANY>
      <PRICE>7.90</PRICE>
      <YEAR>1987</YEAR>
</CD>
```

シーケンスの作成

XQuery は、シーケンスを作成して結合するための演算子をサポートしています。**シーケンス**とは、ゼロ個以上の項目を順序に従って並べた集合です。ここで言う**項目**は、アトミック値の場合もあれば、ノードの場合もあります。

シーケンスを作成する方法の 1 つは、コンマ演算子を使用することです。コンマ演算子は、オペランドをそれぞれ評価し、結果の値を順番に連結し、1 つの結果シーケンスを生成します。シーケンスに重複した値やノードを含めることはできますが、シーケンスを他のシーケンスの項目にすることはできません。2 つ以上の入力シーケンスを連結して新しいシーケンスを作成すると、新しいシーケンスには入力シーケンスの全項目が含まれ、長さは入力シーケンスの長さの合計になります。

例:

```
/ND/CATALOG/CD[YEAR="1988" or YEAR="1987"]/(ARTIST, PRICE)
```

結果:

```
<ARTIST>Sam Brown</ARTIST>
<PRICE>8.90</PRICE>
<ARTIST>Otis Redding</ARTIST>
<PRICE>7.90</PRICE>
```

算術式

XQuery には、加算、減算、乗算、除算、剰余の各算術演算子が通常の二項形式と単項形式で用意されています。

名前		
加減式	::=	Expr ("+" | "-") Expr
乗除式	::=	Expr ("*" | "div" | "mod") Expr
単項式	::=	("-" | "+") Expr

二項減算演算子を NCName の後に置く場合は、有効な名前文字であるハイフンと区別するために、直前にスペースを入れる必要があります。例えば、a-b は単一のトークンとして解釈されます。

算術式は、エラーが発生するか値の計算が完了するまで、順番に以下のルールに基づいて評価されます。

各オペランドには**アトミック化**が適用されるので、オペランドごとに 1 つのアトミック値か空シーケンスが作成されます。

- いずれかのオペランドが空シーケンスの場合、演算の結果は空シーケンスになります。
- オペランドが数値型でない場合は、倍精度にキャストされます。キャストが失敗した場合は、エラー値が返されます。
- 2 つのオペランドのデータ型が違っていても、データ型昇格ルールに基づいて両方のデータ型を共通のデータ型に昇格できる場合、両方のオペランドは最小の共通データ型に昇格します。例えば、1 番目のオペランドが倍精度型で、2 番目のオペランドが整数型の場合、両方のオペランドが倍精度型に昇格します。文字列やノードは、もう 1 つのオペランドが整数でない限り、倍精度に昇格します。
- オペランドのデータ型が演算子にとって有効であれば、その演算子がオペランドに適用され、結果はアトミック値かエラーになります（例えば、ゼロで除算を行った場合、結果はエラーです）。オペランドのデータ型が演算子にとって有効でない場合は、データ型例外が発生します。

算術式の例

- 一般に、数値に対して算術演算を行うと、結果は数値になります。($salary + $bonus) div 12 はその例です。
- 減算演算子とハイフンは違います。$unit-price - $unit-discount はその例です。
- 単項演算子は二項演算子より優先順位が高いですが、括弧の使用すれば、もちろん括弧の制約を受けます。-($bellcost + $whistlecost)はその例です。
- -$bellcost + $whistlecost は(-$bellcost) + $whistlecost と解釈されます。

比較式

名前	構文
等しい	=
等しくない	!=
より小さい （記号の後にスペースが必要。ない場合は開始タグと見なされる）	<
以下	<=
より大きい	>
以上	>=

比較にはオペランドのデータ型がかかわっています。数字と文字列を比較する場合、文字列は比較の前に数字に変換されます。

例:

```
/ND/CATALOG/CD[TITLE = "Stop"]
```

結果:

```
<CD ID="2">
      <TITLE>Stop</TITLE>
      <ARTIST>Sam Brown</ARTIST>
      <COUNTRY>UK</COUNTRY>
      <COMPANY>A and M</COMPANY>
      <PRICE>8.90</PRICE>
      <YEAR>1988</YEAR>
</CD>
```

例:

```
/ND/CATALOG/CD[PRICE > 9.00]
```

結果:

```
<CD ID="8">
      <TITLE>Empire Burlesque</TITLE>
      <ARTIST>Bob Dylan</ARTIST>
      <COUNTRY>USA</COUNTRY>
      <COMPANY>Columbia</COMPANY>
      <PRICE>10.90</PRICE>
      <YEAR>1985</YEAR>
</CD>
```

比較の各オペランドがシーケンスとして評価される場合、1 番目のシーケンスのいずれか 1 つの値と 2 番目のシーケンスのいずれか 1 つの値との比較が真であれば、この比較結果は真になります。

論理式

- **and** – 両方のオペランドが真の場合に結果は真になります。
- **or** – いずれかのオペランドが真の場合に結果は真になります。

FLWR 式

FLWR は for-let-where-return の略です。FLWR 式は、1 つ以上の **for/let** 句、1 つの **where** 句（省略可能）、1 つの **return** 句からなります。

let 句には、キーワード「let」の後に、コンマで区切った変数割り当てのリストを記述します。この句によって変数に式を割り当てておけば、FLWR 式の残りの部分でその変数を参照できます。

```
let $x := /ND/player[@first-name="Todd"], $y := $x/batting return ($x/name, $y/hits)
```

for 句には、キーワード「for」の後に、コンマで区切ったループ変数と式のリストを記述します。各変数と式の間にはキーワード「to」を置きます。FLWR 式の残りの部分は式の値ごとに 1 回評価され、その値が変数に割り当てられます。ループ変数と式のペアを複数指定した場合は、内側の各ループのシーケンスが外側のループの値ごとに 1 回評価されます。例えば、`for $p in /ND/player, $b in $p/batting return $b` という式では、1 番目の選手のすべての打撃記録が参照され、それから 2 番目の選手のすべての打撃記録が参照される、といった具合に処理が進んでいきます。

1 つの FLWR 式に、for 句と let 句はいくつでも、どんな順番でも指定できます。

where 句には、外側の for/let 句によって生成される変数値のセットごとに評価されるブール式を指定します。真と評価されれば return が実行され、偽と評価されれば処理がスキップされます。例えば、次のようにします。

```
for $p in /ND/player let $b := $p/batting[1] where $b/hits >= 200 return $b
```

例:

```
for $cd in /ND/CATALOG/CD sortby(number(PRICE) descending, ARTIST)
let $artist:=$cd/ARTIST, $title:=$cd/TITLE, $price:=$cd/PRICE where $artist =
"Otis Redding" or $artist ="Sam Brown" or $artist = "Bob Dylan"
return concat($artist,"-",$title,"-",$price)
```

結果:

```
<Query-Results>Bob Dylan-Empire Burlesque-10.90 Sam Brown-Stop-8.90 Otis
Redding-The dock of the bay-7.90</Query-Results>
```

XML コンストラクタ

基本的にはタグを含めるだけで、式の出力として XML を作成できます。例えば、<hi/>は有効な XQuery 式であり、子を持たない 1 つの要素を生成します。作成するタグをネストすることや、タグに属性を設定することもできます。要素や属性の内容として式を組み込むこともできます。その場合の式は、中括弧で囲む必要があります。例えば、次の式は選手ごとに 1 つの XML 文書シーケンスを作成します。

```
for  $player in /ND/player sortby(@last-name, @first-name)

return <p what="a player" name={string($player/@name)}>

<years-played>This guy played {count($player/batting)} years</years-played>

</p>
```

作成する要素にテキストを組み込むこともできます。そのためには、中括弧で囲んだ式を使用します。作成する属性の値には、引用符で囲んだ文字列か、中括弧で囲んだ 1 つの式を使用します。要素の内容に開始中括弧を置く場合は、2 つ使用してください。

```
<one.curly.brace>{{</one.curly.brace>
```

ソート式

ソート式によって、項目の順序を制御できます。ソートキーの直後に「ascending」や「descending」という語を使用してソート順を指定できます。ソート順のデフォルトは昇順（ascending）です。

例:

```
/ND/CATALOG/CD sortby (COUNTRY ascending)
```

結果:

```
<CD ID="2">
      <TITLE>Stop</TITLE>
      <ARTIST>Sam Brown</ARTIST>
      <COUNTRY>UK</COUNTRY>
      <COMPANY>A and M</COMPANY>
      <PRICE>8.90</PRICE>
      <YEAR>1988</YEAR>
</CD>
<CD ID="8">
      <TITLE>Empire Burlesque</TITLE>
      <ARTIST>Bob Dylan</ARTIST>
      <ARTIST>Bob Dylan</ARTIST>
      <COUNTRY>USA</COUNTRY>
      <COMPANY>Columbia</COMPANY>
      <PRICE>10.90</PRICE>
      <YEAR>1985</YEAR>
      </CD>
      <CD ID="5">
      <TITLE>The dock of the bay</TITLE>
      <ARTIST>Otis Redding</ARTIST>
      <COUNTRY>USA</COUNTRY>
      <COMPANY>Atlantic</COMPANY>
      <PRICE>7.90</PRICE>
      <YEAR>1987</YEAR>
</CD>
```

ソート式の構文は、「sortby（key-list）」です。key-list にはキー式をコンマで区切ったリストを指定します。各キーにはキーワード「ascending」（デフォルト）か「descending」を付けることもできます。ソートキーには相対パスを使用しなければならず、各キーはソート対象の項目ごとに 1 つの値として評価されるものである必要があります。そうでない場合には例外が発生します。

複数のキーを指定した場合、リスト内の最初のキーが基本キーになり、それ以降のキーはその前のキーの値が等しい場合に順番を決めるのに使用されます。

ソートキーの比較は、キー式のデータ型に基づきます。クエリーからデータ型を判別できない場合は、文字列としてキーの比較が行われます。したがって、要素や属性を数値でソートするためには、double() や integer() の呼び出しを使用して明示的に変換を行う必要があります。ソートキーのパスとして常にドットを使用できます。ノードでない場合にはドットが必須です。ドットはソート対象の項目の値で置き換えられます。

ノード（.）の内容のソート

sortby ステートメントでは、「.」という省略記号を相対パスとして使用できます。ソートキーには、ソートキーとして使用する一意の子要素または属性要素の相対パスを指定します。

例:

```
/ND/CATALOG/CD/TITLE sortby (. ascending)
```

結果:

```
<TITLE>Empire Burlesque</TITLE>
<TITLE>Stop</TITLE>
<TITLE>The dock of the bay</TITLE>
```

メタデータによるソート

文書に含まれているメタデータによるソートも可能です。

そのためには、ソートキーを指定する相対パスに parent 軸を使用します。

例:

```
/ND/CATALOG/CD sortby (root(.)/MetaData/ModifyTime descending)
```

結果:

変更日時に基づいてすべての CD がソートされ、変更日時が一番新しいものが最初に、一番古いものが最後になります。

データ型変換や文字列操作をソートキーに組み込む方法

ソートキー式の中でデータ型変換や文字列操作の関数を使用して、データ内容を比較する方法を指定できます。

例:

```
/ND/CATALOG/CD sortby (upper-case(ARTIST), number(PRICE) descending)
```

結果:

CD が ARTIST のアルファベット順に（大文字小文字に関係なく）ソートされ、アーティストが同じ場合は PRICE の数値順にソートされます。

XQuery 関数サポート

ここでは、NeoCore XMS がサポートするすべての XQuery 関数を取り上げます。それぞれの関数について、定義と例と結果を示します。

integer

文字列を解析、解釈して整数値を生成します。引数が数字以外の文字を含む文字列の場合は、エラーが発生します。

例:

```
/ND/CATALOG/CD[integer(YEAR)+ 10 > 1997]
```

結果:

```
<CD ID="2">
      <TITLE>Stop</TITLE>
      <ARTIST>Sam Brown</ARTIST>
      <COUNTRY>UK</COUNTRY>
      <COMPANY>A and M</COMPANY>
      <PRICE>8.90</PRICE>
      <YEAR>1988</YEAR>
</CD>
```

double

文字列を解析、解釈して倍精度値を生成します。整数値から倍精度値への変換も行います。引数が無効な形式の文字列であれば、エラーが発生します。

例:

```
/ND/CATALOG/CD[double(PRICE)+ 2.55 > 11.45]
```

結果:

```
<CD ID="8">
      <TITLE>Empire Burlesque</TITLE>
      <ARTIST>Bob Dylan</ARTIST>
      <COUNTRY>USA</COUNTRY>
      <COMPANY>Columbia</COMPANY>
      <PRICE>10.90</PRICE>
      <YEAR>1985</YEAR>
</CD>
```

floor

引数の値以下の最大の（正の無限大に最も近い）整数値を返します。引数が空シーケンスであれば、空シーケンスを返します。

例:

```
floor(10.5)
```

結果:

```
10
```

例:

```
/ND/CATALOG/CD[floor(PRICE) > 7]
```

結果:

```
<CD ID="8">
      <TITLE>Empire Burlesque</TITLE>
      <ARTIST>Bob Dylan</ARTIST>
      <COUNTRY>USA</COUNTRY>
      <COMPANY>Columbia</COMPANY>
      <PRICE>10.90</PRICE>
      <YEAR>1985</YEAR>
</CD>
<CD ID="2">
      <TITLE>Stop</TITLE>
      <ARTIST>Sam Brown</ARTIST>
      <COUNTRY>UK</COUNTRY>
      <COMPANY>A and M</COMPANY>
      <PRICE>8.90</PRICE>
      <YEAR>1988</YEAR>
</CD>
```

ceiling

引数の値以上の最小の（負の無限大に最も近い）整数値を返します。引数が空シーケンスであれば、空シーケンスを返します。

例:

```
/ND/CATALOG/CD[ceiling(PRICE) = 9]
```

結果:

```
<CD ID="2">
      <TITLE>Stop</TITLE>
      <ARTIST>Sam Brown</ARTIST>
      <COUNTRY>UK</COUNTRY>
      <COMPANY>A and M</COMPANY>
      <PRICE>8.90</PRICE>
      <YEAR>1988</YEAR>
</CD>
```

round

引数に最も近い整数を返します。もっと厳密な言い方をすれば、round(x)は floor(x+0.5)と同じ結果を返す、ということです。1 つの引数を取り、丸めを行って、引数に最も近い整数値を返します。

例:

```
round(3.5)
```

結果:

```
4
```

例:

```
round(3.4999)
```

結果:

```
3
```

string

引数の値の文字列値を返します。入力値が文字列であれば、単に引数文字列を返します。

引数がノードの場合は、ノードの文字列値を返します。この値は、このノードの直近の子のテキストを連結したものであり、それより下位の要素や属性のテキストは含みません。

ノードを文字列値に変換します。これは出力に影響します。また、数字を文字列に変換します。例えば、concat (string(1), string(2))は"12"になります。

例:

```
string('abc')
```

結果:

```
"abc"
```

concat

1 つ以上の文字列を連結します。引数の数に制限はありません。

`concat("x")`は`"x"`、`concat("x","y")`は`"xy"`、といった具合になります。

例:

```
/ND/CATALOG/CD[TITLE = concat('S', 't', 'o', 'p')]
```

結果:

```
<CD ID="2">
      <TITLE>Stop</TITLE>
      <ARTIST>Sam Brown</ARTIST>
      <COUNTRY>UK</COUNTRY>
      <COMPANY>A and M</COMPANY>
      <PRICE>8.90</PRICE>
      <YEAR>1988</YEAR>
</CD>
```

starts-with

1 番目の引数の文字列値が、2 番目の引数の文字列値で始まっているかどうかを示すブール値を返します。

- starts-with (x, "s")は NeoCore 拡張機能の x+"s"*と同じです。

例: `/ND/CATALOG/CD[starts-with(TITLE, "The")]`

結果:

```
<CD ID="5">
      <TITLE>The dock of the bay</TITLE>
      <ARTIST>Otis Redding</ARTIST>
      <COUNTRY>USA</COUNTRY>
      <COMPANY>Atlantic</COMPANY>
      <PRICE>7.90</PRICE>
      <YEAR>1987</YEAR>
</CD>
```

ends-with

1 番目の引数の文字列値が、2 番目の引数の文字列値で終わっているかどうかを示すブール値を返します。

例 : `/ND/CATALOG/CD[ends-with(TITLE, "que")]`

結果 :

```
<CD ID="8">
      <TITLE>Empire Burlesque</TITLE>
      <ARTIST>Bob Dylan</ARTIST>
      <COUNTRY>USA</COUNTRY>
      <COMPANY>Columbia</COMPANY>
      <PRICE>10.90</PRICE>
      <YEAR>1985</YEAR>
</CD>
```

contains

1 番目の引数の文字列値が、2 番目の引数の文字列値を部分文字列として含んでいるかどうかを示すブール値を返します。

例:

```
/ND/CATALOG/CD[contains(TITLE, "e ")]
```

結果:

```
<CD ID="8">
      <TITLE>Empire Burlesque</TITLE>
      <ARTIST>Bob Dylan</ARTIST>
      <COUNTRY>USA</COUNTRY>
      <COMPANY>Columbia</COMPANY>
      <PRICE>10.90</PRICE>
      <YEAR>1985</YEAR>
</CD>
<CD ID="5">
      <TITLE>The dock of the bay</TITLE>
      <ARTIST>Otis Redding</ARTIST>
      <COUNTRY>USA</COUNTRY>
      <COMPANY>Atlantic</COMPANY>
      <PRICE>7.90</PRICE>
      <YEAR>1987</YEAR>
</CD>
```

contains-keyword

1 番目の引数の文字列値が、2 番目の引数に指定されているキーワードをワード全体として含んでいるかどうかを示すブール値を返します（ターゲットパスにキーワードインデックスが存在していることが前提です）。この検索では大文字小文字の区別がありません。

例:

```
/ND/CATALOG/CD[contains-keyword(TITLE, "empire")]
```

結果:

```
<CD ID="8">
      <TITLE>Empire Burlesque</TITLE>
      <ARTIST>Bob Dylan</ARTIST>
      <COUNTRY>USA</COUNTRY>
      <COMPANY>Columbia</COMPANY>
      <PRICE>10.90</PRICE>
      <YEAR>1985</YEAR>
</CD>
```

substring ($sourceString, $startingLoc, $length)

$sourceString の値の一部、つまり、$startingLoc の値で指定する位置を起点にして、$length の値で指定する数だけ文字を返します。もっと具体的に言えば、$sourceString 内の位置$p が次の範囲にある文字を返します。

```
round($startingLoc) <= $p < round($startingLoc + $length)
```

`$length` を指定しない場合、部分文字列は`$sourceString` の最後の文字までになります。

`$length` が`$sourceString` の値の`$startingLoc` の後の部分の文字数より大きい場合、部分文字列は`$sourceString` の最後の文字までになります。

文字列の最初の文字の位置は、0 ではなく 1 です。

例:

```
substring("motor car", 7)
```

結果:

```
"car"
```

例:

```
substring("metadata", 4, 3)
```

結果:

```
"ada"
```

string-length

引数の文字列値の長さを返します。

例:

```
/ND/CATALOG/CD[string-length(TITLE) < 10]
```

結果:

```
<CD ID="2">
      <TITLE>Stop</TITLE>
      <ARTIST>Sam Brown</ARTIST>
      <COUNTRY>UK</COUNTRY>
      <COMPANY>A and M</COMPANY>
      <PRICE>8.90</PRICE>
      <YEAR>1988</YEAR>
</CD>
```

substring-before

1番目の文字列のうち、2番目の文字列の最初の出現位置の前の文字列部分を返します。1番目の文字列に2番目の文字列が含まれない場合は、1番目の文字列全体を返します。

例:

```
substring-before('abcdabcd', 'd')
```

結果:

```
abc
```

例:

```
substring-before(//CD/PRICE,".")
```

結果:

```
<Query-Results>10 8 7</Query-Results>
```

substring-after

1番目の文字列のうち、2番目の文字列の出現位置の後の部分の文字列部分を返します。1番目の文字列に2番目の文字列が含まれない場合は、空文字列を返します。

例:

```
substring-after("abcdabcd", "d")
```

結果:

```
abcd
```

例:

```
substring-after(//CD/PRICE,".")
```

結果:

```
<Query-Results>90 90 90</Query-Results>
```

normalize-space

引数のスペースを正規化した値を返します。先行スペースと後続スペースを取り除き、連続したスペースを１つのスペースに置き換えます。ここで言うスペースには、ブランク、タブ、改行が含まれます。

例:

```
/ND/CATALOG/CD[TITLE=normalize-space("Empire Burlesque")]
```
または`/ND/CATALOG/CD[normalize-space(TITLE)="Empire Burlesque"]`
または `normalize-space (" hello  world") "hello world"`

結果:

```
<CD ID="8">
      <TITLE>Empire Burlesque</TITLE>
      <ARTIST>Bob Dylan</ARTIST>
      <COUNTRY>USA</COUNTRY>
      <COMPANY>Columbia</COMPANY>
      <PRICE>10.90</PRICE>
      <YEAR>1985</YEAR>
</CD>
```

upper-case

引数の文字列値の中の小文字を対応する大文字に置き換えて返します。

例:

```
/ND/CATALOG/CD[upper-case(TITLE) = "STOP"]
```

結果:

```
<CD ID="2">
      <TITLE>Stop</TITLE>
      <ARTIST>Sam Brown</ARTIST>
      <COUNTRY>UK</COUNTRY>
      <COMPANY>A and M</COMPANY>
      <PRICE>8.90</PRICE>
      <YEAR>1988</YEAR>
</CD>
```

lower-case

引数の文字列値の中の大文字を対応する小文字に置き換えて返します。

例:

```
/ND/CATALOG/CD[lower-case(COUNTRY) = 'uk']
```

結果:

```
<CD ID="2">
      <TITLE>Stop</TITLE>
      <ARTIST>Sam Brown</ARTIST>
      <COUNTRY>UK</COUNTRY>
      <COMPANY>A and M</COMPANY>
      <PRICE>8.90</PRICE>
      <YEAR>1988</YEAR>
</CD>
```

replace ($srcval, $mapString, $transString)

1 番目の文字列の中で 2 番目の文字列がそれぞれ独立して出現するすべての箇所を 3 番目の文字列に置き換えます。

例:

```
replace("abracadabra", "ab", "oob")
```

結果:

```
"oobracadoobra"
```

exec(string)

1 つの引数を指定する必要があります。その引数は、1 つのシーケンスではなく 1 つの文字列値に評価されるものでなければなりません（ただし、concat 関数を使用して、文字列値のシーケンスを 1 つの文字列値に結合することは可能です）。

その文字列値は、クエリーとしてコンパイルされ、実行されます。

例:

```
exec ("<greeting>Hello,world</greeting>")
```
作成済みの文書を返します（愚直な方法です）。

```
for $x in (1 to 10)
return exec (concat("/ND/doc" + string($x) + "/title"))
```
クエリー/ND/doc1/title、/ND/doc2/title、/ND/doc3/title、
. . ./ND/doc10/title を実行し、その 10 個のクエリーを結合した結果シーケンスを返します。

not

ブールオペランドの論理否定を返します。

例:

```
/ND/CATALOG/CD[not(TITLE=*"p"*)]
```

結果:

```
<CD ID="5">
      <TITLE>The dock of the bay</TITLE>
      <ARTIST>Otis Redding</ARTIST>
      <COUNTRY>USA</COUNTRY>
      <COMPANY>Atlantic</COMPANY>
      <PRICE>7.90</PRICE>
      <YEAR>1987</YEAR>
</CD>
```

root

ノード引数が属する文書のルートを/ND レベルで返します。

例:

```
root(//CD[double(PRICE)>9.90])
```

結果:

```
<ND>
      <MetaData>
            <ModifyTime>1029954998</ModifyTime>
            <TimeStamp>1029954998</TimeStamp>
            <SourceFile>CD.XML</SourceFile>
            <DocID>7</DocID>
            <CopyNumber>1</CopyNumber>
      </MetaData>
      <CATALOG>
            <CD ID="8">
                  <TITLE>Empire Burlesque</TITLE>
                  <ARTIST>Bob Dylan</ARTIST>
                  <COUNTRY>USA</COUNTRY>
                  <COMPANY>Columbia</COMPANY>
                  <PRICE>10.90</PRICE>
                  <YEAR>1985</YEAR>
            </CD>
            <CD ID="2">
                  <TITLE>Stop</TITLE>
                  <ARTIST>Sam Brown</ARTIST>
                  <COUNTRY>UK</COUNTRY>
                  <COMPANY>A and M</COMPANY>
                  <PRICE>8.90</PRICE>
                  <YEAR>1988</YEAR>
            </CD>
            <CD ID="5">
                  <TITLE>The dock of the bay</TITLE>
                  <ARTIST>Otis Redding</ARTIST>
                  <COUNTRY>USA</COUNTRY>
                  <COMPANY>Atlantic</COMPANY>
                  <PRICE>7.90</PRICE>
                  <YEAR>1987</YEAR>
            </CD>
      </CATALOG>
</ND>
```

local-name

ノード名のローカル部分を文字列として返します。これは空文字列か、NCName の説明的な文字列のいずれかです。

例:

```
local-name(//CD[PRICE= "10.90"]/..)
```

結果:

```
<Query-Results>CATALOG</Query-Results>
```

number

引数の値を倍精度値に変換して返します。引数が文字列の場合、その値が有効な数字の形式になっていなければ、エラーが発生します。

例:

```
/ND/CATALOG/CD[number(PRICE) > 9.65]
```

結果:

```
<CD ID="8">
      <TITLE>Empire Burlesque</TITLE>
      <ARTIST>Bob Dylan</ARTIST>
      <COUNTRY>USA</COUNTRY>
      <COMPANY>Columbia</COMPANY>
      <PRICE>10.90</PRICE>
      <YEAR>1985</YEAR>
</CD>
```

distinct-nodes

ノードが等しいかどうかに基づいて冗長な重複要素を削除したシーケンスを返します。

例:

```
distinct-nodes(/ND/CATALOG/CD/(*,TITLE))
```

結果:

```
<TITLE>Empire Burlesque</TITLE>
<ARTIST>Bob Dylan</ARTIST>
<COUNTRY>USA</COUNTRY>
<COMPANY>Columbia</COMPANY>
<PRICE>10.90</PRICE>
<YEAR>1985</YEAR>
<TITLE>Stop</TITLE>
<ARTIST>Sam Brown</ARTIST>
<COUNTRY>UK</COUNTRY>
<COMPANY>A and M</COMPANY>
<PRICE>8.90</PRICE>
<YEAR>1988</YEAR>
<TITLE>The dock of the bay</TITLE>
<ARTIST>Otis Redding</ARTIST>
<COUNTRY>USA</COUNTRY>
<COMPANY>Atlantic</COMPANY>
<PRICE>7.90</PRICE>
<YEAR>1987</YEAR>
```

distinct-values

値が等しいかどうかに基づいて冗長な重複要素を削除したシーケンスを返します。一群の冗長重複ノードの中でどのノードを取り込むのかは、実装によって異なります。

例:

```
distinct-values(/ND/CATALOG/CD/PRICE)
```

結果:

```
<PRICE>10.90</PRICE>
<PRICE>8.90</PRICE>
<PRICE>7.90</PRICE>
```

histogram

この関数は、1 つのシーケンスステートメントを受け取り、そのシーケンスに含まれている別々の値と、それぞれの値のカウント（何回出現するか）を返します。結果シーケンスは、タグ項目を持った XML 要素と、各項目内の value と count という 2 つの子要素で構成されます。

例:

```
histogram (/ND/CATALOG/CD/COMPANY)
```

次の結果を返します。

```
<Item>
      <Value>
            <Company>Atlantic</Company>
      </Value>
      <Count>/ </Count>
</Item>
```

count

シーケンス内の項目の数を返します。

例:

```
count(/ND/CATALOG/CD)
```

結果:

```
3
```

例:

```
count(/ND/CATALOG/CD/*)
```

結果:

```
18
```

例:

```
count(/ND/CATALOG/CD/@*)
```

結果:

```
3
```

avg

数字のシーケンスの平均を返します。数値でないシーケンス内の値はまず倍精度値に変換されます。

例:

```
avg(/ND/CATALOG/CD/PRICE)
```

結果:

```
9.233
```

例:

```
/ND/CATALOG/CD[double(PRICE) > avg (ND/CATALOG/CD/PRICE)]
```

結果:

```
<CD ID="8">
      <TITLE>Empire Burlesque</TITLE>
      <ARTIST>Bob Dylan</ARTIST>
      <COUNTRY>USA</COUNTRY>
      <COMPANY>Columbia</COMPANY>
      <PRICE>10.90</PRICE>
      <YEAR>1985</YEAR>
</CD>
<CD ID="2">
      <TITLE>Stop</TITLE>
      <ARTIST>Sam Brown</ARTIST>
      <COUNTRY>UK</COUNTRY>
      <COMPANY>A and M</COMPANY>
      <PRICE>8.90</PRICE>
      <YEAR>1988</YEAR>
</CD>
      <CD ID="5">
      <TITLE>The dock of the bay</TITLE>
      <ARTIST>Otis Redding</ARTIST>
      <COUNTRY>USA</COUNTRY>
      <COMPANY>Atlantic</COMPANY>
      <PRICE>7.90</PRICE>
      <YEAR>1987</YEAR>
</CD>
```

max

比較対象オブジェクトのシーケンスから最大の値を持つオブジェクトを返します。オペランドが数字でない場合（つまり、ノードの場合）は、デフォルトで文字列比較を行います。

例:

```
max(integer(/ND/CD/YEAR))
```

結果:

```
<Query-Results>1988</Query-Results>
```

min

比較対象オブジェクトのシーケンスから最小の値を持つオブジェクトを返します。

例:

```
min((1,2,3,0,5)) => 0
```

結果:

```
<CD ID="5">
      <TITLE>The dock of the bay</TITLE>
      <ARTIST>Otis Redding</ARTIST>
      <COUNTRY>USA</COUNTRY>
      <COMPANY>Atlantic</COMPANY>
      <PRICE>7.90</PRICE>
      <YEAR>1987</YEAR>
</CD>
```

例:

```
/ND/CATALOG/CD[PRICE = min(double(/ND/CATALOG/CD/PRICE))]
```

sum

数字のシーケンスの合計を返します。シーケンスの要素は、数値でない場合、倍精度値に変換されます。

例:

```
sum(/ND/CATALOG/CD/PRICE)
```

結果:

```
27.7
```

document

Document("file.xml")は、/ND レベルで/ND/MetaData/SourceFile="file.xml"であるすべての文書を抽出します。

position

現在処理中の項目のシーケンス内でのコンテキスト項目の位置を返します。最初の項目は位置 1 にあるとします。

以下のものの内部での位置を返します。

1. ステップで終了する式のコンテキストノード
2. 括弧で囲んだ式など、他の式の結果シーケンス全体

例:

```
/ND/CATALOG/CD/*[1]= all the titles (one per cd)
```

例:

```
(/ND/CATALOG/CD sortby (TITLE)) [1]= the first CD by Title
```

例:

```
/ND/CATALOG/CD[position()=2]
```
または/ND/CATALOG/CD[2]

結果:

```
<CD ID="2">
      <TITLE>Stop</TITLE>
      <ARTIST>Sam Brown</ARTIST>
      <COUNTRY>UK</COUNTRY>
      <COMPANY>A and M</COMPANY>
      <PRICE>8.90</PRICE>
      <YEAR>1988</YEAR>
</CD>
```

last

現在処理中の項目シーケンス内の項目数を返します。述部内でのみ有効で、position()との比較に使用します。

例:

```
/ND/CATALOG/CD[last()]
```

結果:

```
<CD ID="5">
      <TITLE>The dock of the bay</TITLE>
      <ARTIST>Otis Redding</ARTIST>
      <COUNTRY>USA</COUNTRY>
      <COMPANY>Atlantic</COMPANY>
      <PRICE>7.90</PRICE>
      <YEAR>1987</YEAR>
</CD>
```

例:

```
/ND/CATALOG/CD[position()=last()-2]
```

結果:

```
<CD ID="8">
      <TITLE>Empire Burlesque</TITLE>
      <ARTIST>Bob Dylan</ARTIST>
      <COUNTRY>USA</COUNTRY>
      <COMPANY>Columbia</COMPANY>
      <PRICE>10.90</PRICE>
      <YEAR>1985</YEAR>
</CD>
```

XQuery に関する注意事項

XMS バージョン 3 におけるスペースの取り扱い

構成ファイル NeoXDBRuntime.xml に 1 つの省略可能パラメータが追加されました。このパラメータの作成場所は、<Runtime>ルートタグの下です。このパラメータの関連タグは<StripWhiteSpace>であり、True、False という 2 つの値が可能です。デフォルト値は True であり、タグが存在しない場合はその値が使用されます。値を False に設定した場合は、先頭と末尾のスペースが保持されます。

XQuery キーワードリスト

以下のキーワードは、パス内で要素や属性の名前として使用できません（例えば、/ND/for は、//@in などと同じくエラーになります）。

- in
- to
- for
- let
- where
- return

ただし、上記のキーワードを変数名として使用することは可能です。

```
let $in := (1,2,3), $return := "hi" for $for in $in return $return
```

つまり、上の書き方は有効です。

一方、上記のキーワードをパスの中で使用すると（例えば、/ND/return）、次のエラーメッセージが出ます。

```
<Error>
      <Name> Malformed Xpath </Name>
      <Message> Unexpected symbol in location step (/ND/?return) </Message>
      <Exception-Number> 7 </Exception-Number>
</Error>
```

エラーメッセージ内の疑問符で、エラーを引き起こしている箇所を見つけることができます。

回避策

/ND/child::for のように、ステップに明示的に軸名を指定すれば、予約語を使用できます。

サイズの限界

以下の表に、クエリーごとのサイズの限界を示します。

注:クエリーの実行時に作成される文字列は、最大 64K に制限されます。ただし、データベース内に現在格納されている最大のリテラルのサイズが 64K を超える場合は、そのリテラルのサイズに制限されます。

ツリーノードの解析	1000
解析対象の式の中のリテラル（引用符で囲んだ文字列と数）	256
クエリーあたりの式のリテラルの総文字数	4096
変数（ユーザー変数と内部変数）	100
FLWR ステートメント内でネストしたスコープ（使用する変数ごとに 1 つ）	50
ステートメント（内部生成されたステートメントを含む）	100
クエリーから作成された式ノード	200
アクセス制御ルールから作成された式ノード	200

XMS トランザクションとコマンド

このセクションでは、トランザクションとコマンドの実装の仕組みと、NeoCore XMS がトランザクションとコマンドを使用する仕組みを説明します。

トランザクションの実装には、記録とロックという 2 つの部分があります。その両方を合わせた結果として、ACID（原子性、一貫性、独立性、耐久性）プロパティが得られます。

注:ACID プロパティの概念をよく理解すると、トランザクションを理解しやすくなります。

トランザクション

トランザクションとは、NeoCore XMS 内の 1 つの作業単位のことで、ACID プロパティを備えています。すべてのトランザクションは、トランザクションログに記録されます。このログがあれば、電源障害などの予測できない障害が発生したときでも、NeoCore XMS の復元が可能になります。

デフォルトでは、それぞれの操作（格納、挿入、クエリーなど）が 1 つのトランザクションと見なされます。トランザクションのネストはサポートされていません。

トランザクションの明示的な開始、コミットおよびロールバックのため、また分離レベルの設定のために、以下のコマンドが用意されています。これらのコマンドは、コンソールの［Database Access］〉［Query］パネルで入力するか、API で指定できます。

- TRANSACTION_START は、1 つのトランザクションを開始します。このコマンドの入力後に実行されるコマンドで、コミットまたはロールバックコマンドのサブミット前に実行されるコマンドはすべて、同じトランザクションの一部と見なされます。TRANSACTION_START を指定する場合、TRANSACTION_COMMIT コマンドか TRANSACTION_ROLLBACK コマンドでトランザクションを終了する必要があります。
- TRANSACTION_COMMIT は、対象のトランザクションの実行中に NeoCore XMS に加えられた変更をコミットします。
- TRANSACTION_ROLLBACK は、対象のトランザクションの実行中に加えられた変更を元に戻します。この操作は、エラー後に処理を取り消す場合や、処理中にニーズが変化した場合などに便利です。
- TRANSACTION_ISOLATION {level}は、分離レベルを設定します。このコマンドは、トランザクションの開始前に呼び出します。トランザクションを開始してしまうと、分離レベルを変更できません。使用できる分離レベルは 4 つあり、SQL2 の規格に対応しています。
- READ_UNCOMMITTED
- READ_COMMITTED
- REPEATABLE_READ（**デフォルト**）
- SERIALIZABLE – **この分離レベルは** NeoCore XMS 2.1.1 **以降のバージョンではサポートされません。**

分離レベルの詳細については、この後の「**分離レベル**」を参照してください。

データの格納、クエリー、挿入、削除、変更の実行中にエラーが発生すると、データを消失や破損から守るために、トランザクションが自動的にロールバックされるようになっています。エラーが発生すると、トランザクションは終了します。

トランザクションのシナリオ

並行セッションの実行中に競合を解決しようとすれば、データベースのパフォーマンスが影響を受けます。

このことの意味は小さくありません。同じ文書を変更しようとするセッションが 2 つあれば競合が発生します。競合するセッションは直列的に処理されます。つまり、2 つのセッションが 1 つの文書を同時に変更することはできません。一方のセッションが文書に対するロックを保持し、他方のセッションはロックが付与されるのを待機するからです。

NeoXDBRuntime.xml ファイルとの関連

NeoXDBRuntime.xml のトランザクション関連パラメータのチューニングが必要になる場合があります。トランザクションに関連した以下の構成項目は、NeoXDBRuntime.xml ファイルの終わりの方にあります。

- MaxTransactionLogSize - トランザクションログの最大サイズ（バイト単位）。このサイズに達すると、新しいログが作成されます。
- TransactionLogDirectory - トランザクションログの場所のフルパス。
- TransactionLockHeadPoolSize - 最大ロック数。アプリケーションのメモリー使用率のチューニングに使用します。
- TransactionLockRequestPoolSize - 最大ロック要求数。アプリケーションのメモリー使用率のチューニングに使用します。
- TransactionMaxDuration - トランザクションの合計最大時間（秒単位）。この時間に達すると、強制ロールバックが行われます。
- TransactionInactivityDuration - トランザクションをアイドル状態（アクションが実行されない状態）にしておける最大時間（秒単位）。この時間に達すると、強制ロールバックが行われます。
- LockTimeout - トランザクションがロックを待機できる最大時間（秒単位）。
- MaxLockRequests - トランザクションが保持できるロック要求の数。この数に達すると、グローバルロックが強制されます。

注: パフォーマンスと安全上の理由から、トランザクションログファイルは、NeoCore XMS ファイルとは別にしておくことを強くお勧めします。これにより、並行環境も充実します。さらに、NeoServer 起動時には、コミットされていないトランザクションをロールバックするために、トランザクションログがスキャンされます。

ロック

TransactionLockHeadPoolSize パラメータに関しては、ロックはリソースごとに付与されます。例えば、1 つの文書は 1 つのリソースになります。付与できるロックの数をリソースの数よりも多くするか、グローバルロックを使用することが必要です。

TransactionLockRequestPoolSize パラメータに関しては、リソースごと、トランザクションごとにトランザクションのロック要求が出されますが、そのたびにすべてのロックが付与されるとは限らないので、ロック要求の数には、付与されるロックの数よりも大きな値を設定できます。

注: リリース 2.1 以降は、ロックは文書レベルで行われるようになりました。したがって、ロックの競合を減らすために、データを格納する XML 文書のサイズを小さくすることを推奨します。

グローバルロック

グローバルロック要求とは、データベースに対する排他アクセスを求める要求です。グローバルロックが付与された場合、リソースに対する個々のロックは不要になります。一度に付与できるロックの数は限られているので、付与できるロックの数よりも、ロックを必要とするリソースの数が多い場合は、グローバルロックを使用してください。

トランザクション間の分離レベル

この機能では、トランザクション間に設定できる分離レベルを定義します。セッションのトランザクションは、**TRANSACTION_ISOLATION** コマンド（または該当する API メソッド）によって設定します。

注: 実行中のトランザクションの分離レベルを変更することはできません。

分離は、データベースシステムのデータ整合性を保つための重要な概念です。トランザクションが並行して実行されている場合、そのうちの 1 つのトランザクションを T とすれば、他のトランザクションは T の前か T の後に実行されることになります（前と後の両方ではありません）。

これは非常に厳格な規格ですが、もっと緩やかな分離のためにさまざまな分離レベルが開発されています。

NeoCore XMS では、*SQL2 規格 ANSI/SQL-92* に指定されている 4 つの分離レベルのうち、以下の 3 つをサポートしています。分離レベルが高いほど、並行性が低く、データの整合性が高くなります。

トランザクション分離レベルとトランザクション間で発生し得る現象の対応表

分離レベル	Dirty read	Non-Repeatable read	Phantom read
REPEATABLE_READ	不可	不可	可
READ_COMMITTED	不可	可	可
READ_UNCOMMITTED	可	可	可

トランザクション間で発生し得る現象

・Dirty read

あるトランザクションが書きこんだコミットされていないデータを、別のトランザクションが読み込むこと

・Non-Repeatable read

あるトランザクションが以前に読み込んだデータを再び読み込んだ時、別のトランザクションがそのデータを変更または削除してコミットした結果が反映されること

・Phantom read

あるトランザクションが以前に読み込んだデータを再び読み込んだ時、別のトランザクションが新規データを挿入してコミットした結果が反映されること

トランザクション分離レベル

・REPEATABLE_READ（**XMS のデフォルト**）

このレベルでは、別のトランザクションによるコミット前のデータや、変更・削除されたコミット後のデータは、再読み込み時に反映されませんが、別のトランザクションによるコミット後の新規データは、再読み込み時に反映されます。

・READ_COMMITTED

このレベルでは、別のトランザクションによるコミット前のデータは、再読み込み時に反映されませんが、別のトランザクションによるコミット後の変更・削除および新規データは、再読み込み時に反映されます。

・READ_UNCOMMITTED

このレベルでは、別のトランザクションによるコミット前のデータの再読み込みが可能です。また別のトランザクションによるコミット後の変更・削除および新規データは、再読み込み時に反映されます。

XML の格納

基本的なユーザー格納コマンドは、以下の 2 つです。

storeFileXML – ファイルを格納します。

storeXML – 文字列を格納します。

どちらの方法を使用した場合でも、HTTP の mulit-part mime メッセージが NeoServer に送信されます。NeoServer は、mime メッセージを解析して、その文書を格納します（渡されたスキーマ URL やプレフィックス文書などの情報も一緒に格納します）。正常に格納が行われると、新たに格納された文書の文書 ID（マルチ文書の場合は、最後に使用された文書 ID）と、格納された文書の数が示されます。例えば、次のようにします。

```
<Store-Results>
   <Documents-Processed>1</Documents-Processed>
   <Last-Doc-ID>10</Last-Doc-ID>
</Store-Results>
```

マルチ文書の場合は、以下のような結果が返されます。

```
<Store-Results>
   <Documents-Processed>62</Documents-Processed>
   <Last-Doc-ID>72</Last-Doc-ID>
</Store-Results>
```

XML 文書のための格納パラメータ

注: 格納機能には、［Console］>［Database Access］>［Store］からアクセスできます。

文書を格納するときには、以下の格納パラメータに注意してください。

- **SourceFile** – 必須
- **PrefixFile** – 省略可能 – 文書のメタデータに追加するプリフィックスファイルの位置パラメータ
- **SchemaFile –** 省略可能

同じルートタグを持った文書が複数ある場合は、それらを 1 つのファイルに入れて格納できます。その場合、ファイル内の各文書は別々に格納されます。

XML 格納処理

文書を格納するたびに、NeoCore XMS は以下のことを実行します。

- 文書の前後を`<ND></ND>`タグペアで囲みます。
- デフォルトのメタデータタグを文書の`/ND/MetaData` の下に追加します。
- 格納済みのデータに文書を追加します。

XML 格納要件

- XML は*整形式である*必要があります。つまり、構文的に正しい必要があります。
- NeoCore XMS が稼働中である必要があります。

NeoCore XMS メタデータタグ

格納される文書には MetaData セクションが追加されます。メタデータタグは、以下のとおりです。

- <ModifyTime>
  NeoCore XMS が文書の更新を受け取ったときのシステム時刻（秒単位）。文書を初めて格納したときには、**TimeStamp** タグと **ModifyTime** タグに同じデータが入ります。ModifyTime タグは、文書内で XML の挿入、削除、変更を行うときに変更されます。
- <TimeStamp>
  NeoCore XMS が文書を受け取ったときのシステム時刻（秒単位）。文書をコピーした場合、コピーには新しい **TimeStamp** 値が設定されます。
- <SourceFile>
  データの元となるファイルの名前（**sales.xml** など）。
- <DocID>
  文書 ID。これは、格納順に文書に割り当てられる数値です。1 から始まります。
- <CopyNumber>
  この文書コピーの番号。元の文書には、**CopyNumber** として 1 が割り当てられます。NeoCore XMS で文書をコピーするたびに、そのコピーには元の文書と同じ DocID 値が設定されますが、**CopyNumber** 値はコピーのたびに 1 ずつ増えていきます。
- <SchemaFile>
  この文書と一緒にスキーマを格納した場合に、その文書スキーマが入っているファイルの URI が設定されます。
- <PrefixFile>
  プレフィックスファイルの名前（**prefix.xml** など）。

スキーマの格納

特定の文書で使用する XML スキーマファイルを定義できます。その場合、NeoCore XMS は、そのスキーマの適用対象文書のメタデータとして、スキーマファイルの URI を内容とする `<SchemaFile></SchemaFile>`タグペアをその文書に追加します。

注: NeoCore XMS は、スキーマ情報を参照用として格納しますが、スキーマに基づいて文書を検証することはありません。

コンソールを使用してスキーマを格納するには、以下のようにします。

1. XML **ファイルを格納するための要件**を満たします。スキーマの適用対象文書を格納するときに、そのスキーマを内容とするファイルを同時に指定する必要があります。
2. ［**Database Access**］パネルで［Store］を選択します。
3. ［Schema File］フィールドに、スキーマファイルの URI を入力します。例えば、次のようにします。
   `http://yourcomputer.yourcompany.com/xml/schema/example.xml`
4. ［XML File］フィールドに、格納する XML 文書の URI を入力します。例えば、次のようにします。
   `C:¥Documents¥Purchasing¥Order5489.xml`
5. ［Store XML］をクリックします。文書と、その XML スキーマへの参照が格納されます。

プレフィックスファイルの格納

文書の著者、タイトル、バージョンなど、文書に関する追加のメタデータ（**プレフィックスデータ**）を定義できます。そのデータは XML ファイルで定義し、文書に関連付けます。その場合、NeoCore XMS は、文書のメタデータとして、プレフィックスファイルの URI を内容とする`<PrefixFile></PrefixFile>`タグペアをその文書に追加します。

プレフィックスデータファイルの例:

```
<Company_metadata>
      <department>Software Engineering</department>
      <author>Jane Doe</author>
      <date>09-16-01</date>
      <documentnumber>1234</documentnumber>
</Company_metadata>
```

コンソールを使用してプリフィックスファイルを格納するには、以下のようにします。

1. XML **ファイルを格納するための要件**を満たします。プレフィックスデータの適用対象文書を格納するときに、そのプレフィックスデータを内容とするファイルを同時に指定する必要があります。
2. ［**Database Access**］パネルで［Store］を選択します。
3. ［Prefix File］フィールドに、プレフィックスファイルの URI を入力します。例えば、次のようにします。
   `C:¥Records¥xml¥metadata¥company.xml`
4. ［XML File］フィールドに、格納する文書の URI を入力します。例えば、次のようにします。
   `C:¥Documents¥Specs¥Interface.xml`
5. ［Store］をクリックします。文書と、そのプレフィックスファイルへの参照が格納されます。

名前空間を使用した XML の格納とクエリー

XML 名前空間は URI によって定義します。この URI は、要素群に関する固有の範囲 ID としての役割を果たします。

NeoCore XMS は、名前空間と、各要素タグに対応するプレフィックスを保持します。ただし、インデックス構築は、タグ自体に基づいています。同じタグを使用した複数の文書からデータを抽出するときには、それらの文書内でタグが別々の意味を持つ場合であっても、名前空間は無視されます。

要素タグ名自体は修飾されません。
<para>

同じ要素名を別々の目的で使用する文書群では、個別にプレフィックスを設定できます。

<me:para>（医療文書用。この場合の<para>タグは「paramedic」を意味します）
<ld:para>（法律文書用。この場合の<para>要素は「paralegal」を意味します）
<pg:para>（段落を設けた文書用。この場合の<para>要素は「paragraph」を意味します）

完全修飾要素名では、各要素の URI に対する xmlns 参照と、要素タグに対応するプレフィックスを指定します。プレフィックスとタグはコロンで区切ります。

```
<me:para xmlns:me "http://hospital.com/jobdef">
<ld:para xmlns:ld "http://lawfirm.com/employeelist">
<pg:para xmlns:pg "http://indexing.com/literature">
```

/ND//para を対象としたクエリーでは、3 つの文書すべてに含まれているすべての<para>とそれぞれの子要素が返されます。

名前空間による Query 結果のフィルタリングを行うには以下のように記述します。

```
Declare namespace xmlns:me="http://hospital.com/jobdef";
/ND/me:para
```

上記 Query の場合、医療用文書の 1 文書のみが抽出されます。

また、Namespace 宣言を使用しないでプレフィックスを使用した Query を実行しようとすると例外（CANNOT_RESOLVE_NS）が発生します。

格納例外メッセージ

格納処理で障害条件が発生すると、例外が生成されます。返されるメッセージとその意味は、以下のとおりです。

- <StoreError>Prefix file does not exist.</StoreError> プレフィックスファイルがストアに渡されていて、見つかりませんでした。
- <StoreError>Prefix file too large.</StoreError> プレフィックスファイルは 16K に制限されています。
- <StoreError>Cannot open source file</StoreError> ソース XML ファイルを開けません。
- <StoreError>Cannot find source file</StoreError> ソース XML ファイルが存在しません。
- <StoreError>File level initialization failed</StoreError> アプリケーションはファイルごとに一部の変数を初期設定します。この初期設定が失敗すると、エラーが表示されます。
- <StoreError>Document level initialization failed</StoreError> アプリケーションは文書ごとに一部の変数を初期設定します。この初期設定が失敗すると、エラーが表示されます。
- <StoreError>Parser rejected file- see log for details</StoreError> XML ソースが整形式ではありません。ログエントリには、文書内でのエラー箇所とエラータイプが示されます。
- <StoreError>General application error- check log</StoreError> これは汎用メッセージです。ログに詳細が示されます。このエラーの原因としては、名前空間の非互換が考えられます。大きなバッファの割り振りの失敗が原因になっている場合もあります。
- <StoreError>Empty socket</StoreError> このエラーは、XML データがソケット上に見つからない場合に生成されます。これは、ストアのソケット読み取り機能によってのみ生成されるもので、ファイル読み取り機能によっては生成されません。
- <StoreError>Prefix file must come before source file</StoreError> このエラーは、ストアのソケット読み取り機能によってのみ生成されます。ソケットを読み取る際には、パラメータの順序が重要になります。プレフィックスファイルがあれば、それをソースファイルよりも先に指定する必要があります。
- <StoreError>Schema file name must come before source file</StoreError> 上記を参照してください。スキーマファイル名があれば、それもソースファイルより先に指定する必要があります。

マルチ文書ファイル格納のトランザクション動作

マルチ文書ファイルの格納要求では、そのファイルに含まれるすべての文書が格納されるか、まったく格納されないかのいずれかです。いずれかの文書にエラーがあると、その時点で既に同じファイルから格納されている文書がロールバックされます。

コメントと処理命令は、XML 文書のタグ構造の外側に置くことも、文書自体の前後に置くこともできます。

例えば、次のようにします。

```
<A>
       <B> some data </B>
</A>
<!-- This is a comment outside of a document root  -->
```

この場合は、マルチ文書ファイルを格納するときに、不明瞭な点が出てきます。つまり、このコメントは、マルチ文書ファイルの中で前の文書に属するのか、後の文書に属するのかという点です。同様に、マルチ文書シーケンスの中で、1 つの文書の先頭にコメントを置いた場合も、そのコメントが前の文書に属するのか、後の文書に属するのかがはっきりしません。マルチ文書ファイルの途中でいずれかの状況が発生すると、XMS はこの状況を回避する手段を持たないので、格納は失敗し、ロールバックが行われます。

XML の削除

削除トランザクションは、データストアから XML を削除します。削除対象の指定には、XPath 式を使用します。

XML **を削除するための** XPath **式の例**

削除対象:	入力する式（大文字小文字の区別あり）:
<product name="Heron"> </product>	/ND/sales/region/product [@name="Heron"]
<product name="Heron"> & <product name="Crane">	/ND/sales/region/product [@name="Heron" or @name="Crane"]
<name>Northeast</name>	/ND/sales/region/name[.="Northeast"]
DocID 1、CopyNumber 1 のファイル	/ND[MetaData/DocID=1 and MetaData/CopyNumber=1]

注: デフォルトの<MetaData>タグは削除しないでください。NeoCore XMS では、このタグが必要です。

XML の挿入

挿入トランザクションは、データストア内の既存の XML に XML を追加します。XPath 式を使用して、ターゲットノードを指定します（挿入はそのターゲットの後に行われます）。

XML を挿入するための XPath 式の例

挿入内容:	入力する式（大文字小文字の区別あり）:
「Southeast」という名前の地域ノードの後に XML を挿入。	`/ND/sales/region[name="Southeast"]`
`<name>Southeast</name>`ノードの下に XML を挿入。	`/ND/sales/region/name[.="Southeast"]`
コピー番号 1 の文書の「Southeast」という名前の地域ノードの後に XML を挿入。	`/ND[MetaData/CopyNumber=1]/sales/region [name="Southeast"]`

注:挿入により、文書に複数のルートが存在するようになることがあります（整形式ではありません）。NeoCore XMS は、データベースレベルのルートタグ<ND>で文書を囲みます。これは、データベースに格納されているすべての文書に対して、新しい有効なルートタグとして機能します。したがって、このタグの内部に完全な XML 文書を挿入することによって、2 つの別々の文書から新しい 1 つの文書を作成することができます。この処理は、NeoCore XMS がメタデータを作成するときに NeoCore XMS 自身が行います。例えば、以下の例を取り上げましょう。

ソース文書 1:

```
<A> original root
      <B> child of A </B>
</A>
```

NeoCore XMS では、以下のように格納されます。

```
<ND>
      <MetaData>
            <ModifyTime>10193284</ModifyTime>
            .
            .
            .
      </MetaData>
      <A> original root
            <B> child of A </B>
      </A>
</ND>
```

ソース文書 2:

```
<X> another original root
      <Y> child of X </Y>
</X>
```

文書 1 の後に文書 2 を挿入します（ターゲット:**/ND/A**）。

```
<ND>
      <MetaData>
            <ModifyTime>10195176</ModifyTime>
            .
            .
            .
      </MetaData>
      <A> original root
            <B> child of A </B>
      </A>
      <X>another original root
            <Y> child of X </Y>
      </X>
</ND>
```

複数の挿入

文書に XML を挿入すると、NeoCore XMS は、指定の XPath ターゲットの兄弟ノードとしてその XML を扱います。親ノードのない状態で一度に複数の兄弟ノードを挿入することはできません。

例えば、以下の XML フラグメントを取り上げましょう。

```
<A>
      <B>This is a childless element</B>
      <C>This is a parent element and also a sibling element of B
            <D>Child of C</D>
      </C>
</A>
```

/ND/A/C の後に<F>と<G>という新しいノードを挿入しようとしても、各兄弟ノードの親ノードが指定されていないので、この挿入操作は実行できません。Could not parse XML input というメッセージが返されます。

代わりに、以下の 2 つの挿入を実行します。

挿入ターゲットの XPath:

```
/ND/A/C/D
```

挿入する XML:

```
<E>Sibling 1</E>
```

結果:

```
<A>
      <B>This is a childless element</B>
      <C>This is a parent element and also a sibling element of B
            <D>Child of C</D>
            <E>Sibling 1</E>
      </C>
</A>
```

挿入ターゲットの XPath:

```
/ND/A/C/E
```

挿入する XML:

```
<F>Sibling 2</F>
```

結果:

```
<A>
      <B>This is a childless element</B>
      <B1>A newly inserted childless element</B1>
      <C>This is a parent element and also a sibling element of B
            <D>Child of C</D>
            <E>Sibling 1</F>
            <F>Sibling 2</G>
      </C>
</A>
```

ただし、以下のように、整形式の XML フラグメント全体を挿入することは可能です。

挿入ターゲットの XPath:

```
/ND/A/B
```

挿入する XML:

```
<B1>Newly inserted parent element
      <C1>Child of B1
            <D1>Child of C1</D1>
      </C1>
</B1>
```

XML の変更

変更トランザクションは、既存の XML を変更します。変更対象の指定には、XPath 式を使用します。

XML を変更するための XPath 式の例

変更内容:	入力する式（大文字小文字の区別あり）：
`<name>Southeast</name>`	`/ND/sales/region/name[.="Southeast"]`
コピー番号 1 の `<name>Southeast</name>` (ファイルの複数のコピーが格納されている場合)	`/ND[MetaData/CopyNumber=1]/sales/region/name[.="Southeast"]`

注: デフォルトの`<MetaData>`タグは変更しないでください。NeoCore XMS では、このタグが必要です。

XML 文書のコピー

コピートランザクションは、文書のコピーを NeoCore XMS に格納します。文書をコピーすると、新しいコピーには、前のコピーよりも 1 つ大きい値の MetaData CopyNumber タグが割り当てられます。この番号によって、格納されているコピーを区別できます。文書のコピーを多数格納した場合でも、DocID と SourceFile は同じままです。

XML をコピーするための XPath 式の例

コピー内容:	入力する式（大文字小文字の区別あり）：
DocID が 12 のファイル。	`/ND[MetaData/DocID=12]`
DocID が 17 で、CopyNumber が 3 のファイル（ファイルの複数のコピーが格納されている場合）。	`/ND[MetaData/DocID=17 and MetaData/CopyNumber=3]`

XMS コマンド

NeoCore XMS では、標準のクエリートランザクションのほかに、特殊なクエリーコマンドを使用できます。この種のコマンドを使用するクエリーは、/ND で開始しません。

NeoCore XMS **コマンド**

dataquery	dataquery コマンドは、データオンリークエリーに使用します。クエリーの対象になるのは、コンテキストのデータです。ターゲットデータを含むノードごとに、メタデータとフラット形式の行が返されます。 **注**: dataquery は、2 文字以上のデータに対してのみ有効です。
count	count クエリーは、ターゲットセット内のノードそのものではなく、ノードの数を返します。
flat	flat クエリーは、ターゲットレベル以下のすべてのノードに関するフラット形式の行を返します。
tree	tree クエリーは、ターゲットよりも 1 レベル下のノードの名前を返します。

データオンリークエリー

データオンリークエリーは、データストアを検索して、指定データのすべてのインスタンスを検出します。ユーザーは、文書の構造を知らなくても、この種のクエリーを実行できます。データオンリークエリーは、ターゲットデータを含むノードごとに、メタデータとフラット形式の行を返します。完全なインデックス構築が必要なので、NeoXDBRuntime.xml の IndexMode パラメータを 0 に設定してください。dataquery コマンドの形式は、`dataquery "data"`です。

例えば、データオンリークエリーは、以下のようになります。

```
dataquery "Tina Turner"
```

以下のような結果が返されます。

```
<Query-Results>
    <Result>
       <SourceFile>cd-catalog.xml</SourceFile>
       <DocID>15</DocID>
       <CopyNumber>1</CopyNumber>
       <path>/ND/CATALOG/CD/ARTIST</path>
    </Result>
</Query-Results>
```

データ値全体を指定しない場合のクエリー結果は空になります。例えば、以下のクエリーを実行するとしましょう。

```
dataquery "Tina"
```

以下のような結果が返されます。

```
<Query-Results>
</Query-Results>
```

dataquery コマンドを使用すると、コンテキストのデータを抽出できるだけでなく、XPath のワイルドカード検索の場合よりも詳細な情報を取得できます。

注: dataquery は、2 文字以上のデータに対してのみ有効です。

Count クエリーと Countnotacid クエリー

count クエリーは、データを検索して、ターゲットに一致するノードの数を検出し、ターゲットセット内のノードの数を返します。count クエリーの形式は、`count <XPath expression>`です。

例えば、サンプル XML 文書を格納した状態で、データ内に「Kingfisher」が出現する回数を知りたい場合は、以下のクエリーを使用できます。`count /ND//[.="Kingfisher"]`

以下のような結果が返されます。

```
<Count-Results>
   <Count>5</Count>
</Count-Results>
```

countnotacid クエリーも、ターゲットセット内のノードの数を返します。この countnotacid は、クエリーの中で count コマンドの代わりに使用できます。コマンド名から明らかなとおり、このコマンドには ACID プロパティがありません。NeoCore XMS は、countnotacid クエリーの実行中に、格納されているデータを他のトランザクションに対してロックしません。このクエリーは、他のトランザクションが使用中でないときや、読み取り専用トランザクションだけが使用中のときに使用してください。countnotacid クエリーの形式は、`countnotacid <XPath expression>`です。

Tree クエリー

tree クエリーは、データを検索して、ターゲットよりも 1 レベル下のノードを検出します。tree クエリーの形式は、tree XPath expression です。

例えば、以下のクエリーを実行するとしましょう。

```
tree /ND/sales
```

以下のような結果が返されます。

```
<Tree-Results>
caption,region
</Tree-Results>
```

Flat クエリー

flat クエリーは、データを検索して、ターゲットに一致するノードを検出し、ターゲットノードの各出現部分に関するフラット形式の行を返します。flat クエリーの形式は、flat <XPath expression> です。

例えば、以下のクエリーを実行するとしましょう。

```
flat /ND//*[.="Wong"]
```

以下のような結果が返されます。

```
<Flat-Results>
   <line>ND>letter>cc-list>name>last-name> Wong</line>

</Flat-Results>
```

Java API

API の互換性

必ず、NeoCore XMS に組み込まれている API を使用してください。旧リリースの API は現在の NeoServer と互換性がなく、現在の NeoCore XMS API は旧リリースのサーバーと互換性がありません。

注: 以前は、HTTPBean の開発中に、Java API にいくらかの変更を行うことが必要でした。つまり、NeoConnectionInterface のすべてのメソッドに生成可能な例外を 1 つ追加しなければなりませんでした。現在では、NeoConnectionInterface のすべてのパブリックメソッドは、java.io.IOException を生成します。ただし、HTTPBean は WebLogic 環境でしかテストされていません。

Java API の資料

JavaDoc を読むには、以下のようにします。

1. NeoServer 管理コンソールを開始します。
2. ［**Help**］タブを選択します。
3. ［**Java API Documentation**］リンクを選択します。

NeoServer は、*マルチスレッド*と*トランザクション*に対応しています。それぞれの要求は 1 つのスレッドに対応しており、複数の要求を 1 つのトランザクションにまとめることができます。

Java API メソッド呼び出しの例

ここでは、格納、削除、挿入、変更、クエリーなどの基本的な NeoCore XMS の作業を Java API から実行する方法の例を示します。このバージョンでは、トランザクションは暗黙的です（つまり、トランザクションは、クライアントが明示的に実行するのではなく、むしろ NeoCore XMS によって内部的に実行されます）。ただし、クライアントが明示的にトランザクション処理を行うことも可能です。詳細については、Java API の資料を参照してください。

Java フェーズ 1:Java アプリケーションサンプルソースコード

```
//Title:        Test client demo using NeoCore XMS
//Version:      1.2
//Date:         6/11/01
//Copyright:    Copyright (c) 2001
//Author:       Kevin Huck
//Company:      NeoCore
//Description:  Demonstrates the basic functions of  XIMS
//Updates:      t clark   06/02

/**
This is a very simple Java application that demonstrates the basic functions
including store, delete, insert, modify, and query.  Transactions are implied
in this version, i.e. they are handled internally by XIMS, not by the client.
The client can handle transactions explicity - see users guide for more information.
*/


import com.neocore.httpclient.*;

public class TestClient {
  public static String server = "localhost";    // !!! CHANGE THIS TO THE NAME OR
IP OF YOUR INSTANCE !!!
  public static SessionManagedNeoConnection neosession;

  public TestClient() {
  }

  public static void main(String[] args) {
    TestClient testClient = new TestClient();
    String sid = null;

    //--------------------- get connection and login -------------------
    try{
      neosession = new  SessionManagedNeoConnection(server, 7700);
    // neosession will manage session id for us
    }catch(Exception e){
      System.out.println("Cannot connect to NeoCore server:" + e);
      return;
    }
    try{
      sid = neosession.login("Administrator", "admin");
    }catch(Exception e){
      System.out.println("Cannot login to NeoCore server:" + e);
      return;
    }
    try{
      String s = "xxx";
      //-------------------- first, delete any entry documents ---------
```

```
      System.out.println("deleting documents");
      s = neosession.deleteXML("/ND[Entries]¥n");
      System.out.println("Results from delete: [" + s + "]");

      //-------------------- now store a simple document --------------
      System.out.println("Storing document");
   s =
neosession.storeXML("<Entries><Entry><ID>1</ID><Name>One</Name></Entry><Entry>
<ID>2</ID><Name>Two</Name></Entry></Entries>", null, null);
      System.out.println("Store results: [" + s + "]");

      //-------------------- now insert an entry ---------------------
      System.out.println("Inserting an entry");
      s = neosession.insertXML("/ND/Entries/Entry[ID=¥"2¥"]¥n",
"<Entry><ID>3</ID><Name>Thre</Name></Entry>¥n");
      System.out.println("Results from insert: [" + s + "]");

      s = neosession.queryXML("/ND/Entries¥n");
      System.out.println("Current document: [" + s + "]");

      //-------------------- now fix mispelling of "three" -------------
      System.out.println("Modifying an element");
      s      =      neosession.modifyXML("/ND/Entries/Entry[ID=\"3\"]/Name\n",
"<Name>Three</Name>\n");
      System.out.println("Results from insert: [" + s + "]");

      s = neosession.queryXML("/ND/Entries\n");
      System.out.println("Current document: [" + s + "]");

      //-------------------- now delete entry 2 -------------
      System.out.println("Deleting an entry");
      s = neosession.deleteXML("/ND/Entries/Entry[ID=\"2\"]\n");
      System.out.println("Current document: [" + s + "]");

      s = neosession.queryXML("/ND/Entries\n");
      System.out.println("Current document: [" + s + "]");

      //---------------------- logout ------------------------
      neosession.logout();
    }catch(Exception e){
      System.out.println("NeoCore error: " + e);
    }
  }
}
```

C++ API

API の互換性

必ず、NeoCore XMS に組み込まれている API を使用してください。旧リリースの API は現在の NeoServer と互換性がなく、現在の NeoCore XMS API は旧リリースのサーバーと互換性がありません。

多言語対応の注意点

注: NeoConnection、NeoPool、SessionManagedNeoconnection のいずれかで接続を作成するときに、デフォルトの English/UTF-8 以外の言語と文字セットを使用する場合は、その言語と文字セットを指定する必要があります。

注: NeoConnection をプールから取得すると、この接続は、直前のスレッドの接続と同じ言語と文字セットを使用します。また、同一スレッド内のすべての NeoConnection は、同じ言語と文字セットを使用します。

注: スレッド間で 1 つの NeoConnection を共用することは避けてください。2 つのスレッドが同じ接続を使用してコマンドを送信するのは、安全ではありません。

C++ API の資料

C++ API の資料を読むには、以下のようにします。

1. NeoServer 管理コンソールを開始します。
2. ［Help］タブを選択します。
3. ［C++ Documentation］リンクを選択します。

サンプルについて

Solaris の make や Windows の nmake を使用してコンパイルと実行可能ファイルの構築を行う場合は、サンプルの makefile を使用できます。Winodws の場合は、Visual C++でソースをロードして構築を行うことも可能です。

C++ API コードサンプル

ここでは、クエリー、削除、挿入、変更などの基本的な NeoCore XMS の作業を C++ API から実行する方法の例を示します。このサンプルでは、トランザクションは暗黙的です。つまり、トランザクションは、クライアントが明示的に実行するのではなく、むしろ XMS によって内部的に実行されます。ただし、クライアントが明示的にトランザクション処理を行うことも可能です。この C++ API のコードサンプルには、4 つのフェーズがあります。

1） ソースコード

2） サンプルアプリケーションのコンパイル

3） コンパイル済みの実行可能画面

4） コマンドラインの結果

次の C++サンプルコードは、XMS の情報（つまり XML）に対する操作を実行するときに活用できる構造、ライブラリ呼び出し、ステートメント、引数の例を示しています。Solaris の make や Windows の nmake を使用してコンパイルと実行可能ファイルの構築を行う場合は、サンプルの makefile を使用できます。Winodws の場合は、Visual C++でソースをロードして構築を行うことも可能です。このプログラムでは、ユーザーがコマンドラインからログインデータを入力するためのプロンプトが表示されます。

C++フェーズ 1:C++アプリケーションサンプルソースコード

```
#include "SessionManagedNeoConnection.h"¶
#include "iostream.h"¶
#include "string.h"¶
¶
int main (int argc, char** argv)¶
{¶
     char* menu = (char *) ¶
          "Please make a selection:\n'Q' for query\n'D'
for delete\n'I' for insert\n'M' for modify\n'E' for
exit\n";¶
     char* results ;¶
     cout << "APIC Test start" << endl ;¶
¶
     try¶
     {¶
          SessionManagedNeoConnection
neosession((char*)"localhost", 7700) ;¶
¶
          cout << "Login: ";¶
          char login[30];¶
          char password[30];¶
          cin >> login;¶
          cout << "Password: ";¶
          cin >> password;¶
          cout << "\n";¶
          char* sid = neosession.login((char*)login,
(char*)password) ;¶
¶
          if (sid) cout << "sid: " << sid << endl ;¶
          ¶
          char selection = '?';¶
          while((selection != 'e') && (selection !=
'E')){¶
               cout << menu;¶
               cin >> selection;¶
```

```
            if((selection == 'q') || (selection ==
'Q')) (¶
                  cout << "Enter query:";¶
                  char query[500];¶
                  cin >> query;¶
                  results =
neosession.queryXML((char*)query);¶
                  if (results)¶
                  (¶
                       cout << results << endl ;¶

    neosession.releaseBuffer(results) ;¶
                       results = NULL ;¶
                  ) else (¶
                       cout << "Query failed" << endl;¶
                  )¶
            ) else if((selection == 'd') ||
(selection == 'D')) (¶
                  cout << "Enter delete query:";¶
                  char query[500];¶
                  cin >> query;¶
                  results =
neosession.deleteXML((char*)query);¶
                  if (results)¶
                  (¶
                       cout << results << endl ;¶

    neosession.releaseBuffer(results) ;¶
                       results = NULL ;¶
                  ) else (¶
                       cout << "Delete failed" << endl;¶
                  )¶
            ) else if((selection == 'i') ||
(selection == 'I')) (¶
                  cout << "Enter insert query:";¶
                  char query[500];¶
```

```
                    cin >> query;¶
                    cout << "Enter insert queryXML:";¶
                    char queryXML[500];¶
                    cin >> queryXML;¶
                    results =
neosession.insertXML((char*)query, queryXML);¶
                    if (results)¶
                    {¶
                        cout << results << endl ;¶

    neosession.releaseBuffer(results) ;¶
                        results = NULL ;¶
                    } else {¶
                        cout << "Insert failed" << endl;¶
                    }¶
                } else if((selection == 'm') ||
(selection == 'M')) {¶
                    cout << "Enter modify query:";¶
                    char query[500];¶
                    cin >> query;¶
                    cout << "Enter modify queryXML:";¶
                    char queryXML[500];¶
                    cin >> queryXML;¶
                    results =
neosession.modifyXML((char*)query,queryXML);¶
                    if (results)¶
                    {¶
                        cout << results << endl ;¶

    neosession.releaseBuffer(results) ;¶
                        results = NULL ;¶
                    } else {¶
                        cout << "Modify failed" << endl;¶
                    }¶
                }¶

            }¶
            cout << "Bye";¶
}¶
    catch (NeoException e)¶
    {¶
        const char* msg = e.getMessage() ;¶
        cout << e.getMessage() << endl ;¶
    }¶
    ¶
    return 0 ;¶
}¶
¶
```

C++フェーズ 2:サンプルアプリケーションのコンパイル

Solaris でのコンパイル

Solaris 環境で Solaris CC を使用してアプリケーションを構築するには、コンパイラとリンカーのフラグを正しく組み合わせて指定する必要があります。

32 ビットコンパイルのための CC のフラグは、次のとおりです。

```
-I/opt/NeoCore/API/C/include
```

32 ビットリンクのための CC のフラグは、次のとおりです。

```
-L/opt/NeoCore/API/C/lib
-lxmsclient32 -lsocket –lnsl -mt
```

64 ビットコンパイルのための CC のフラグは、次のとおりです。

```
-I/opt/NeoCore/API/C/include
-xarch=v9
```

64 ビットリンクのための CC のフラグは、次のとおりです。

```
-L/opt/NeoCore/API/C/lib
-lxmsclient64
-xarch=v9
-lsocket -lnsl -mt
```

Solaris アプリケーションの makefile のサンプル

```
all:sample

INCDIRS=-I/opt/NeoCore/API/C/include
CCFLAGS= $(INCDIRS) -xarch=v9
LINK_LIBS= -L/opt/NeoCore/API/C/lib -lxmsclient64
LFLAGS=$(LINK_LIBS) -xarch=v9 -lsocket -lnsl -mt

sample :sample.o

/opt/SUNWspro/bin/CC -o $@ sample.o $(LFLAGS)
sample.o :sample.cpp
/opt/SUNWspro/bin/CC -c sample.cpp $(CCFLAGS)
```

Windows でのコンパイル

Microsoft Windows 環境でアプリケーションを構築するには、コンパイラとリンカーのフラグを正しく組み合わせて指定する必要があります。

コンパイラフラグには、次の指定を入れる必要があります。

```
/I "C:¥NeoCore¥API¥C¥include"
```

リンカーフラグには、次の指定を入れる必要があります。

```
xmsclient.lib
```

Microsoft Windows NMAKE Makefile のサンプル

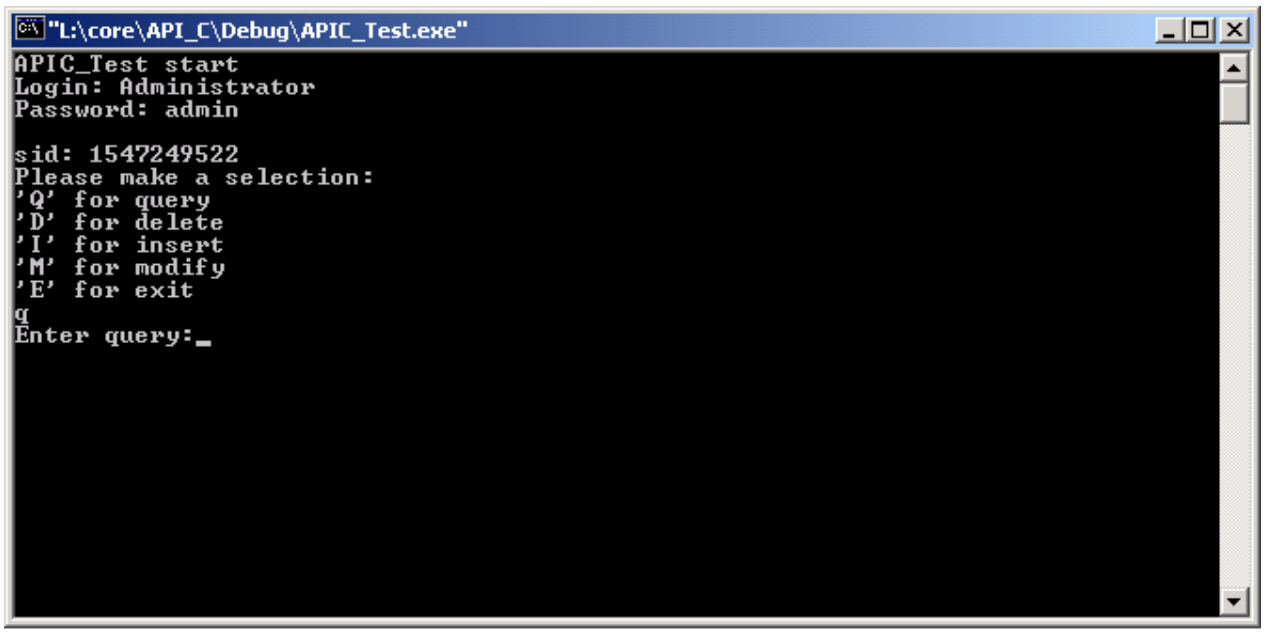
```
# Microsoft Developer Studio Generated NMAKE File, Based on ExampleAPI.dsp
!IF "$(CFG)" == ""
CFG=ExampleAPI - Win32 Debug
!MESSAGE No configuration specified.Defaulting to ExampleAPI - Win32 Debug.
!ENDIF
!IF "$(CFG)" != "ExampleAPI - Win32 Release" && "$(CFG)" != "ExampleAPI - Win32
Debug"
!MESSAGE Invalid configuration "$(CFG)" specified.
!MESSAGE You can specify a configuration when running NMAKE
!MESSAGE by defining the macro CFG on the command line.For example:
!MESSAGE
!MESSAGE NMAKE /f "ExampleAPI.mak" CFG="ExampleAPI - Win32 Debug"
!MESSAGE
!MESSAGE Possible choices for configuration are:
!MESSAGE
!MESSAGE "ExampleAPI - Win32 Release" (based on "Win32 (x86) Console Application")
!MESSAGE "ExampleAPI - Win32 Debug" (based on "Win32 (x86) Console Application")
!MESSAGE
!ERROR An invalid configuration is specified.
!ENDIF
!IF "$(OS)" == "Windows_NT"
NULL=
!ELSE
NULL=nul
!ENDIF
!IF "$(CFG)" == "ExampleAPI - Win32 Release"
OUTDIR=.¥Release
INTDIR=.¥Release
# Begin Custom Macros
OutDir=.¥Release
# End Custom Macros
ALL :"$(OUTDIR)¥ExampleAPI.exe"
CLEAN :
-@erase "$(INTDIR)¥ExampleAPI.obj"
-@erase "$(INTDIR)¥ExampleAPI.pch"
-@erase "$(INTDIR)¥StdAfx.obj"
-@erase "$(INTDIR)¥vc60.idb"
-@erase "$(OUTDIR)¥ExampleAPI.exe"
"$(OUTDIR)" :
if not exist "$(OUTDIR)/$(NULL)" mkdir "$(OUTDIR)"
CPP=cl.exe
CPP_PROJ=/nologo /ML /W3 /GX /O2 /I "C:¥NeoCore¥API¥C¥include" /D "WIN32" /D
"NDEBUG" /D "_CONSOLE" /D "_MBCS" /Fp"$(INTDIR)¥ExampleAPI.pch" /Yu"stdafx.h"
/Fo"$(INTDIR)¥¥" /Fd"$(INTDIR)¥¥" /FD /c
.c{$(INTDIR)}.obj::
$(CPP) @<<
$(CPP_PROJ) $<
<<
.cpp{$(INTDIR)}.obj::
$(CPP) @<<
$(CPP_PROJ) $<
<<
.cxx{$(INTDIR)}.obj::
$(CPP) @<<
$(CPP_PROJ) $<
<<
.c{$(INTDIR)}.sbr::
$(CPP) @<<
$(CPP_PROJ) $<
```

```
<<
.cpp{$(INTDIR)}.sbr::
$(CPP) @<<
$(CPP_PROJ) $<
<<
.cxx{$(INTDIR)}.sbr::
$(CPP) @<<
$(CPP_PROJ) $<
<<
RSC=rc.exe
BSC32=bscmake.exe
BSC32_FLAGS=/nologo /o"$(OUTDIR)¥ExampleAPI.bsc"
BSC32_SBRS= ¥
LINK32=link.exe
LINK32_FLAGS=kernel32.lib user32.lib gdi32.lib winspool.lib comdlg32.lib
advapi32.lib shell32.lib ole32.lib oleaut32.lib uuid.lib odbc32.lib odbccp32.lib
xmsclient.lib /nologo /subsystem:console /incremental:no
/pdb:"$(OUTDIR)¥ExampleAPI.pdb" /machine:I386 /out:"$(OUTDIR)¥ExampleAPI.exe"
/libpath:"C:¥NeoCore¥API¥C¥lib"
LINK32_OBJS= ¥
"$(INTDIR)¥StdAfx.obj" ¥
"$(INTDIR)¥ExampleAPI.obj"
"$(OUTDIR)¥ExampleAPI.exe" :"$(OUTDIR)" $(DEF_FILE) $(LINK32_OBJS)
$(LINK32) @<<
$(LINK32_FLAGS) $(LINK32_OBJS)
<<
!ELSEIF "$(CFG)" == "ExampleAPI - Win32 Debug"
OUTDIR=.¥Debug
INTDIR=.¥Debug
# Begin Custom Macros
OutDir=.¥Debug
# End Custom Macros
ALL :"$(OUTDIR)¥ExampleAPI.exe"
CLEAN :
-@erase "$(INTDIR)¥ExampleAPI.obj"
-@erase "$(INTDIR)¥ExampleAPI.pch"
-@erase "$(INTDIR)¥StdAfx.obj"
-@erase "$(INTDIR)¥vc60.idb"
-@erase "$(INTDIR)¥vc60.pdb"
-@erase "$(OUTDIR)¥ExampleAPI.exe"
-@erase "$(OUTDIR)¥ExampleAPI.ilk"
-@erase "$(OUTDIR)¥ExampleAPI.pdb"
"$(OUTDIR)" :
if not exist "$(OUTDIR)/$(NULL)" mkdir "$(OUTDIR)"
CPP=cl.exe
CPP_PROJ=/nologo /MLd /W3 /Gm /GX /ZI /Od /I "C:¥NeoCore¥API¥C¥include" /D "WIN32"
/D "_DEBUG" /D "_CONSOLE" /D "_MBCS" /Fp"$(INTDIR)¥ExampleAPI.pch" /Yu"stdafx.h"
/Fo"$(INTDIR)¥¥" /Fd"$(INTDIR)¥¥" /FD /GZ /c
.c{$(INTDIR)}.obj::
$(CPP) @<<
$(CPP_PROJ) $<
<<
.cpp{$(INTDIR)}.obj::
```

```
$(CPP) @<<
$(CPP_PROJ) $<
<<
.cxx{$(INTDIR)}.obj::
$(CPP) @<<
$(CPP_PROJ) $<
<<
.c{$(INTDIR)}.sbr::
$(CPP) @<<
$(CPP_PROJ) $<
<<
.cpp{$(INTDIR)}.sbr::
$(CPP) @<<
$(CPP_PROJ) $<
<<
.cxx{$(INTDIR)}.sbr::
$(CPP) @<<
$(CPP_PROJ) $<
<<
RSC=rc.exe
BSC32=bscmake.exe
BSC32_FLAGS=/nologo /o"$(OUTDIR)¥ExampleAPI.bsc"
BSC32_SBRS= ¥
LINK32=link.exe
LINK32_FLAGS=kernel32.lib user32.lib gdi32.lib winspool.lib comdlg32.lib
advapi32.lib shell32.lib ole32.lib oleaut32.lib uuid.lib odbc32.lib odbccp32.lib
xmsclient.lib /nologo /subsystem:console /incremental:yes
/pdb:"$(OUTDIR)¥ExampleAPI.pdb" /debug /machine:I386
/out:"$(OUTDIR)¥ExampleAPI.exe" /pdbtype:sept /libpath:"C:¥NeoCore¥API¥C¥lib"
LINK32_OBJS= ¥
"$(INTDIR)¥StdAfx.obj" ¥
"$(INTDIR)¥ExampleAPI.obj"
"$(OUTDIR)¥ExampleAPI.exe" :"$(OUTDIR)" $(DEF_FILE) $(LINK32_OBJS)
$(LINK32) @<<
$(LINK32_FLAGS) $(LINK32_OBJS)
<<
!ENDIF
!IF "$(NO_EXTERNAL_DEPS)" != "1"
!IF EXISTS("ExampleAPI.dep")
!INCLUDE "ExampleAPI.dep"
!ELSE
!MESSAGE Warning:cannot find "ExampleAPI.dep"
!ENDIF
!ENDIF
!IF "$(CFG)" == "ExampleAPI - Win32 Release" || "$(CFG)" == "ExampleAPI - Win32
Debug"
SOURCE=.¥ExampleAPI.cpp
"$(INTDIR)¥ExampleAPI.obj" :$(SOURCE) "$(INTDIR)" "$(INTDIR)¥ExampleAPI.pch"
SOURCE=.¥StdAfx.cpp
!IF "$(CFG)" == "ExampleAPI - Win32 Release"
CPP_SWITCHES=/nologo /ML /W3 /GX /O2 /I "C:¥NeoCore¥API¥C¥include" /D "WIN32" /D
"NDEBUG" /D "_CONSOLE" /D "_MBCS" /Fp"$(INTDIR)¥ExampleAPI.pch" /Yc"stdafx.h"
/Fo"$(INTDIR)¥¥" /Fd"$(INTDIR)¥¥" /FD /c
"$(INTDIR)\StdAfx.obj""$(INTDIR)¥ExampleAPI.pch" :$(SOURCE) "$(INTDIR)"
$(CPP) @<<
$(CPP_SWITCHES) $(SOURCE)
<<
!ELSEIF "$(CFG)" == "ExampleAPI - Win32 Debug"
```

```
CPP_SWITCHES=/nologo /MLd /W3 /Gm /GX /ZI /Od /I "C:\NeoCore¥API¥C¥include" /D
"WIN32" /D "_DEBUG" /D "_CONSOLE" /D "_MBCS" /Fp"$(INTDIR)¥ExampleAPI.pch"
/Yc"stdafx.h" /Fo"$(INTDIR)¥¥" /Fd"$(INTDIR)¥¥" /FD /GZ /c
"$(INTDIR)¥StdAfx.obj""$(INTDIR)¥ExampleAPI.pch" :$(SOURCE) "$(INTDIR)"
$(CPP) @<<
$(CPP_SWITCHES) $(SOURCE)
<<
!ENDIF
!ENDIF
```

C++フェーズ 3:C++コンパイル済みの実行可能画面



C++フェーズ４： 結果

```
Login:Administrator
Password:pwd

Please make a selection:
'D' for delete
'I' for insert
'M' for modify
'E' for exit
q
Enter query:/ND/TAG_01_00001/TAG_02_00001
<Query-Results>
    <TAG_02_00001>DATA_02_00001</TAG_02_00001>
    <TAG_02_00001>DATA_02_00004</TAG_02_00001>
    <TAG_02_00001>DATA_02_00000</TAG_02_00001>
</Query-Results>
=
Please make a selection:
'Q' for query
'D' for delete
'I' for insert
'M' for modify
'E' for exit
d
Enter delete query:/ND/TAG_01_00001/TAG_02_00001
<Delete-Results>
    <Deleted-Nodes>3</Deleted-Nodes>
</Delete-Results>
-
Please make a selection:
'Q' for query
'D' for delete
'I' for insert
'M' for modify
'E' for exit
i
Enter insert query:/ND/TAG_01_00001[sibling=01]/TAG_02_00001
Enter insert queryXML:<TAG_02_00002>DATA_02_00001</TAG_02_00002>
<Insert-Results>
    <Insert-Nodes>1</Insert-Nodes>
</Insert-Results>
-
Please make a selection:
'Q' for query
'D' for delete
'I' for insert
'M' for modify
'E' for exit
m
01_00001[sibling=01]/TAG_02_00001
<Modify-Results>
    <Modified-Nodes>1</Modified-Nodes>
</Modify-Results>
-
```

COM API

注：COM API はシングルスレッドで動作します。

COM コンポーネントの登録

NeoCore COM コンポーネントの登録は、自動でも手動でも実行できます。

自動登録

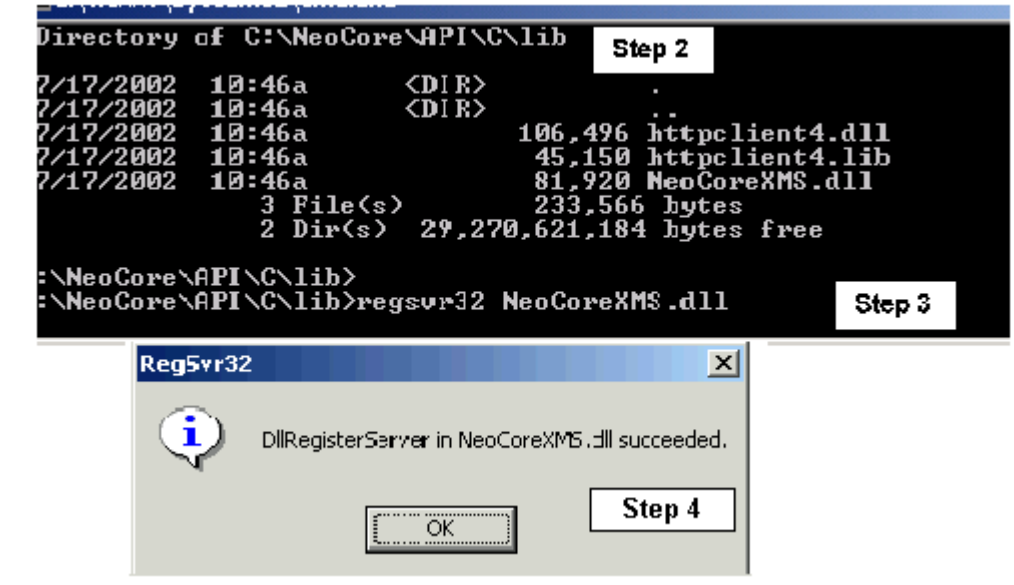
NeoCore XMS ソフトウェアのインストール時に API オプションを選択すれば、NeoCore COM コンポーネントは自動的に登録されます。

手動登録

クライアントマシンに NeoCore API COM オブジェクトを手動で登録することも可能です。例えば、10 台のクライアントマシンに COM API アプリケーションだけを操作する環境を作る必要があるとします。その場合は、各クライアントマシンで以下のようにします。

1. NeoServer の API ディレクトリ（例えば、デフォルトのインストールパス **C:¥NeoCore¥API¥C¥lib**）からクライアントマシンに同じパス構造を使用してすべてのファイルをコピーします。
2. ディレクトリ変更によってクライアントマシンの同じパスに移動します。
3. **regsvr32 NeoCoreXMS.dll** コマンドを実行します。
4. 「RegSvr32」成功メッセージボックスが表示されます。

次の図「NeoCore API COM の手動登録」に、上記のステップ 2、3、4 を示します（ステップ 1 でデフォルトインストールパス **C:¥NeoCore¥API¥C¥lib** を使用した場合を想定しています）。



NeoCore API COM の手動登録

これで、アプリケーションや IDE で COM API を使用するための用意ができました。

NeoServer は、マルチスレッドとトランザクションに対応しています。それぞれの要求は 1 つのスレッドに対応しており、複数の要求を 1 つのトランザクションにまとめることができます。

NeoCore XMS では、標準のクエリートランザクションのほかに、特殊なクエリーコマンド（`dataquery`、`count`、`flat`、`tree`）を使用できます。この種のコマンドを使用するクエリーは、/ND で開始しません。NeoCore XMS 3.0 では、これらのコマンドの使用は望ましくありません（使用自体は可能です）。詳細については、『***System Administration Guide***』を参照してください。

COM に関する付録

API 接続管理

メソッド: connectToNeoCoreXMS

構文:`HRESULT connectToNeoCoreXMS(BSTR host, int port, BSTR locale, BSTR encoding, BSTR resourceRootDir )`

引数［in］:BSTR host

引数［in］:int port

引数［in］:BSTR locale（デフォルトは en）

引数［in］:BSTR encoding（デフォルトは UTF-8）

引数［in］:BSTR resourceRootDir（デフォルトは DEFAULT）

戻り型: HRESULT

説明:ホストとポートを指定して NeoCore XMS に対する接続を確立します。成功すれば、さらに NeoCore API メソッドを使用できます。

注: BSTR locale、BSTR encoding、BSTR resourceRootDir にはいずれもデフォルト値があります。BSTR host と int port の値だけを指定した場合は、コンパイル時に、BSTR locale、BSTR encoding、BSTR resourceRootDir のデフォルト値が使用されます。デフォルト値以外のオプションを使用したい場合は、以下の値の中から選択できます。

- BSTR encoding のオプション:
  - UTF-8
  - Shift_JIS
  - EUC-JP
  - ISO-2022-JP
- BSTR locale のオプション:
  - en
  - ja

注: NeoCore API オブジェクトの作成後に BSTR locale と BSTR encoding の値を設定するには、以下の setCharset メソッドと setLocale メソッドを使用します。

ホストの例: 172.16.3.123 または localhost

ポートの例: 8080

メソッド: setCharset

構文:`HRESULT setCharset([in] BSTR charset`

引数［in］:BSTR charset

戻り型: HRESULT

説明:現在の文字セットを設定します。有効な値は、Shift_JIS、EUC-JP、UTF-8 などの IANA フォーマットです。注:この文字セットは、バイナリバージョンのメソッドだけに適用されます。

メソッド: getCharset

構文: `HRESULT getCharset([out, retval] );`

引数［in］: BSTR* result

戻り型: HRESULT

説明:現在の文字セットを取得します。有効な値は、Shift_JIS、EUC-JP、UTF-8 などの IANA フォーマットです。**注:**この文字セットは、バイナリバージョンのメソッドだけに適用されます。

メソッド: getLocale

構文:`HRESULT getLocale([out, retval] BSTR* result);`

引数［in］:BSTR* result

戻り型: HRESULT

説明:現在のロケールの設定を取得します。有効な値は、en などの http Accept-Language フォーマットです。

メソッド: setLocale

構文: `HRESULT setLocale([in] BSTR locale);`

引数［in］: BSTR locale

戻り型: HRESULT

説明:現在のロケールの設定を設定します。有効な値は、en などの http Accept-Language フォーマットです。

メソッド: isConnected

構文:`HRESULT isConnected(VARIANT_BOOL* connected)`

引数［out, retval］:VARIANT_BOOL* connected

*connected = (m_SMNeoConnection != NULL) ?(TRUE) :(FALSE);

戻り型: HRESULT

説明:現時点で NeoCore XMS に接続している場合は connected に TRUE を返し（connectToNeoCoreXMS() を参照）、そうでない場合は FALSE を返します。

メソッド: **closeConnection**

構文:`HRESULT closeConnection()`

戻り型: HRESULT

説明:NeoCore XMS への接続を閉じます。

セッション管理

メソッド: **login**

構文:`HRESULT login(BSTR user, BSTR password,BSTR* result)`

引数［in］:BSTR user

引数［in］:BSTR password

引数［out, retval］:BSTR* result

戻り型: HRESULT

説明:NeoCore XMS にログインして、セッション識別番号（645929907 など）を返します。

結果の例: 645929907

メソッド: isLoggedIn

構文:`HRESULT isLoggedIn(VARIANT_BOOL* loggedIn)`

引数［out, retval］:VARIANT_BOOL* loggedIn

戻り型: HRESULT

説明:現時点で NeoCore XMS にログインしている場合は loggedIn に TRUE を返し（login()を参照）、そうでない場合は FALSE を返します。

メソッド: logout

構文:`HRESULT logout(BSTR* result)`

引数［out, retval］:BSTR* result

戻り型: HRESULT

説明:NeoCore XMS からログアウトして、NeoServer の応答を返します。

結果の例: `<success/>`

データベースコマンド

メソッド: queryXML

構文: `HRESULT queryXML(BSTR query, BSTR* result)`

引数［in］:BSTR query

引数［out, retval］:BSTR* result

戻り型: HRESULT

説明: クエリーに指定した XQuery 式を使用して NeoCore XMS から XML を検索し、NeoServer の応答を返します。

クエリーの例: /ND/Example

結果の例:

```
<Query-Results>
    <Example> examplequeryXML</Example1>
</Query-Results>
```

メソッド: queryXMLBinary

構文: `HRESULT queryXMLBinary(SAFEARRAY** query, SAFEARRAY** result)`

引数［in］: SAFEARRAY** query

引数［out, retval］: SAFEARRAY** result

戻り型: HRESULT

説明: クエリーに指定した XPath 式を使用して NeoCore XMS から XML を検索します。非 Unicode 文字エンコードを保持するために、クエリーも抽出結果の XML も、Unicode 文字ではなくバイト配列による記述になります。

メソッド: queryXMLUpdateIntent

構文:`HRESULT queryXMLUpdateIntent(BSTR query,BSTR* result)`

引数［in］:BSTR query

引数［out, retval］:BSTR* result

戻り型: HRESULT

説明:更新目的のクエリーに指定した XPath 式を使用して NeoCore XMS から XML を検索し、NeoServer の応答を返します。このメソッドは、クエリーが完了するまで（暗黙的トランザクション）、または commitTransaction()か rollbackTransaction()が実行されるまで（明示トランザクション）、クエリー結果 XML に対する書き込みロックを保持します。通常、このメソッドを使用するのは、クエリーによって抽出した XML を更新する意図があるときです。

クエリーの例: `/ND/Example)`

結果の例:

```
<Query-Results>
    <Example>examplequeryXMLUpdateIntent</Example>
</Query-Results>
```

メソッド: **queryXMLUpdateIntentBinary**

構文:`HRESULT queryXMLUpdateIntentBinary(SAFEARRAY** query, SAFEARRAY** result)`

引数［in］: SAFEARRAY** query

引数［out, retval］: SAFEARRAY** result

戻り型: HRESULT

説明:更新目的のクエリーに指定した XPath 式を使用して NeoCore XMS から XML を検索します。非 Unicode 文字エンコードを保持するために、Unicode 文字ではなくバイト配列を返します。

メソッド: queryXMLProfiled

構文:`HRESULT queryXMLProfiled([in] BSTR query, [in] IResultProfile* resultProfile, [out, retval] IQueryResult** queryResult`

引数［in］:BSTR query、［in］IResultProfile（結果が空でない場合は NeoQueryResult、結果が空の場合は NULL を返します）

引数［out, retval］:QueryResult

戻り型: HRESULT

説明:カーソル機能とチャンク機能をサポートした queryXML です。

結果の例: 結果が空でない場合は NeoQueryResult、結果が空の場合は NULL を返します。

メソッド: **queryXMLProfiledBinary**

構文:`HRESULT queryXMLProfiledBinary([in] SAFEARRAY** query, [in] IResultProfile* resultProfile, [out, retval] IQueryResult** queryResult`

引数［in］:SAFEARRAY**query

引数［in］: IResultProfile（結果が空でない場合は NeoQueryResult、結果が空の場合は NULL を返します）

引数［out, retval］:QueryResult

戻り型: HRESULT

説明:結果が空でない場合は NeoQueryResult、結果が空の場合は NULL を返します。非 Unicode 文字エンコードを組み込むために、入力クエリーにはバイト配列を使用します。

メソッド: queryXMLUpdateIntentProfiled

構文:`HRESULT queryXMLUpdateIntentProfiled([in] BSTR query,[in] IResultProfile* resultProfile, [out, retval] IQueryResult** queryResult`

引数［in］:BSTR query、［in］IResultProfile* resultProfile（query は XPath クエリー、resultProfile は NeoQueryResult のプロファイル）

引数［out, retval］:QueryResult

戻り型: HRESULT

説明:カーソル機能とチャンク機能をサポートした queryXMLUpdateIntent です。更新目的のクエリーコマンドで XMS を呼び出します。つまり、クエリーが完了するまで（暗黙的トランザクション）、またはコミットかロールバックが実行されるまで、クエリーによって抽出した XML をロックするということです。

結果の例: 結果が空でない場合は NeoQueryResult、結果が空の場合は NULL を返します。

メソッド: **queryXMLUpdateIntentProfiledBinary**

構文:`HRESULT queryXMLUpdateIntentProfiledBinary([in] SAFEARRAY** query,[in] IResultProfile* resultProfile, [out, retval] IQueryResult** queryResult`

引数［in］: SAFEARRAY**query

引数［in］: IResultProfile* resultProfile（query は XPath クエリー、resultProfile は NeoQueryResult のプロファイル）

引数［out, retval］:QueryResult

戻り型: HRESULT

説明:カーソル機能とチャンク機能をサポートした queryXMLUpdateIntent です。更新目的のクエリーコマンドで XMS を呼び出します。つまり、クエリーが完了するまで（暗黙的トランザクション）、またはコミットかロールバックが実行されるまで、クエリーによって抽出した XML をロックするということです。非 Unicode 文字エンコードを組み込むために、入力クエリーにはバイト配列を使用します。

メソッド: queryFlatXML

構文:`HRESULT queryFlatXML(BSTR query, BSTR* result)`

引数［in］:BSTR query

引数［out, retval］:BSTR* result

戻り型: HRESULT

説明:NeoCore XMS に対してクエリーを実行し、フラット形式の XML 結果（つまり、クエリー結果をフラット形式の行として組み込んだ XML）を返します。

結果の例:

```
<Flat-Results>
    <line>ND>MetaData>ModifyTime> 1023463375</line>
    <line>ND>MetaData>TimeStamp> 1023463375</line>
</Flat-Results>
```

メソッド: **queryFlatXMLBinary**

構文:`HRESULT queryFlatXMLBinary(SAFEARRAY**ppQuery, SAFEARRAY** ppResult)`

引数［in］: SAFEARRAY** query

引数［out, retval］: SAFEARRAY** result

戻り型: HRESULT

説明:NeoCore XMS に対してクエリーを実行し、フラット形式の XML 結果を返します。非 Unicode 文字エンコードを保持するために、Unicode 文字ではなくバイト配列を返します。

メソッド: queryTreeXML

構文:`HRESULT queryTreeXML(BSTR query, BSTR* result)`

引数［in］:BSTR query

引数［out, retval］:BSTR* result

戻り型: HRESULT

説明:ターゲットクエリーよりも 1 つ下のレベルのノードの名前を検索し、ターゲットクエリーよりも 1 つ下のレベルのノードの名前を含んだ XML を返します。

結果の例:

```
<Tree-Results>
    NeoCore,a
</Tree-Results>
```

メソッド: **queryTreeXMLBinary**

構文:`HRESULT queryTreeXMLBinary(SAFEARRAY** ppQuery, SAFEARRAY** ppResult)`

引数［in］: SAFEARRAY** query

引数［out, retval］: SAFEARRAY** result

戻り型: HRESULT

説明:ターゲットクエリーよりも 1 つ下のレベルのノードの名前を検索し、ターゲットクエリーよりも 1 つ下のレベルのノードの名前を含んだ XML を返します。非 Unicode 文字エンコードを保持するために、Unicode 文字ではなくバイト配列を返します。

メソッド: queryDataContextXML

構文:`HRESULT queryDataContextXML (BSTR query, BSTR* result)`

引数［in］:BSTR query

引数［out, retval］:BSTR* result

戻り型: HRESULT

説明:コンテキストのデータを検索し、ターゲットクエリーデータを含んだ各ノードのメタデータをフラット形式の行として組み込んだ XML を返します。

注:このコマンドをサポートするには、データベースの完全インデックス構築が必要です。

結果の例:

```
<Query-Results>
  <Result>
        <SourceFile>1023465935</SourceFile>
        <path>/ND/a</path>
  </Result>
</Query-Results>
```

メソッド: **queryDataContextXMLBinary**

構文:`HRESULT queryDataContextXMLBinary(SAFEARRAY** ppQuery, SAFEARRAY** ppResult)`

引数［in］: SAFEARRAY** query

引数［out, retval］: SAFEARRAY** result

戻り型: HRESULT

説明:コンテキストのデータを検索し、ターゲットクエリーデータを含んだ各ノードのメタデータをフラット形式の行として組み込んだ XML を返します。非 Unicode 文字エンコードを保持するために、Unicode 文字ではなくバイト配列を返します。

メソッド: queryCountXML

構文:`HRESULT queryCountXML(BSTR query, VARIANT_BOOL acid, BSTR* result)`

引数［in］:BSTR query

引数［in］:VARIANT_BOOL acid

引数［out, retval］:BSTR* result

戻り型: HRESULT

説明:ターゲットクエリーに含まれるノードの数をカウントし、そのカウントを含んだ XML を返します。

結果の例:

```
<Count-Results>
    <Count>10</Count>
</Count-Results>
```

メソッド: **queryCountXMLBinary**

構文:`HRESULT queryCountXMLBinary(SAFEARRAY** ppQuery, VARIANT_BOOL acid, SAFEARRAY** ppResult)`

引数［in］: SAFEARRAY** query

引数［in］: VARIANT_BOOL acid

引数［out, retval］: SAFEARRAY** result

戻り型: HRESULT

説明: ターゲットクエリーに含まれるノードの数をカウントし、そのカウントを含んだ XML を返します。非 Unicode 文字エンコードを保持するために、Unicode 文字ではなくバイト配列を返します。

メソッド: storeFileXML

構文:
```
HRESULT storeFileXMLBinary(BSTR sourceXMLFile,BSTR schemaFile,
        BSTR prefixFile, BSTR* result)
```

引数［in］:BSTR sourceXMLFile

省略可能引数［in］:BSTR schemaFile

省略可能引数［in］:BSTR prefixFile

引数［out, retval］:BSTR* result

戻り型: HRESULT

説明:sourceXMLFile ファイルに含まれている XML を格納し、成功した場合は、正常に格納できた文書の数と最後に格納した文書の ID を示した格納成功メッセージを返します。省略可能なパラメータである schemaFile と prefixFile には、NeoCore XMS 内で sourceXMLFile に関連付けられている XML を含んだファイルを指定します。これらの省略可能な XML ファイルを指定しない場合は、それぞれの引数を単に""と設定します。

パラメータ値の例:

sourceXMLFile :`"filePath¥sourceFileContainingXML.xml"`

schemaFile :`"filePath¥schemaFileContainingXML.xml" or ""`

prefixFile :`"filePath¥prefixFileContainingXML.xml" or ""`

結果の例:

```
<Store-Results>

    <Documents-Processed> 1 </Documents-Processed>

    <Last-Doc-ID> 21 </Last-Doc-ID>

</Store-Results>
```

メソッド: storeXML

構文:HRESULT storeXML(BSTR sourceXML, BSTR schemaFile,
BSTR prefixFile, BSTR* result)

引数［in］:BSTR sourceXML

省略可能引数［in］:BSTR schemaXML

省略可能引数［in］:BSTR prefixXML

引数［out, retval］:BSTR* result

戻り型: HRESULT

説明:sourceXML 文字列に含まれている XML を格納し、成功した場合は、正常に格納できた文書の数と最後に格納した文書の ID を示した格納成功メッセージを返します。省略可能なパラメータである schemaXML と prefixXML には、NeoCore XMS 内で sourceXML に関連付けられている XML を含んだファイルを指定します。これらの省略可能な XML 文字列を指定しない場合は、それぞれの引数を単に""と設定します。

パラメータ値の例:

sourceXML : "<Example1>sourceXML</Example1>"

schemaXML : "<Example2>exampleschemaXML</Example2>" or ""

prefixXML : "<Example3>examplePrefixXML</Example3>" or ""

結果の例:

```
<Store-Results>
    <Documents-Processed> 1 </Documents-Processed>
    <Last-Doc-ID> 22 </Last-Doc-ID>
</Store-Results>
```

メソッド: **storeXMLBinary**

構文:`HRESULT storeXMLBinary(SAFEARRAY** ppSourceXML, SAFEARRAY** ppSchemaFile, SAFEARRAY** ppPrefixFile, SAFEARRAY** ppRresult)`

引数［in］: SAFEARRAY** sourceXML

引数［in］: SAFEARRAY** schemaFile

引数［in］: SAFEARRAY** prefixFile

引数［out, retval］: SAFEARRAY** result

戻り型: HRESULT

説明:sourceXML 文字列に含まれている XML を格納し、成功した場合は、正常に格納できた文書の数と最後に格納した文書の ID を示した格納成功メッセージを返します。非 Unicode 文字エンコードを保持するために、入力文書は、Unicode 文字ではなくバイト配列で記述します。

メソッド: insertXML

構文:`HRESULT insertXML(BSTR query, BSTR newXML, BSTR* result)`

引数［in］:BSTR query

引数［in］:BSTR newXML

引数［out, retval］:BSTR* result

戻り型: HRESULT

説明:クエリーに指定した XPath 式に基づいて NeoCore XMS 内に XML を挿入し、挿入したノードの総数を返します。

クエリーの例:`/ND/Example`

結果の例:

```
<Insert-Results>
    <Insert-Nodes>2</Insert-Nodes>
</Insert-Results>
```

メソッド: **insertXMLBinary**

構文:`HRESULT insertXMLBinary(SAFEARRAY** ppQuery, SAFEARRAY** ppNewXML, SAFEARRAY** ppResult)`

引数［in］: SAFEARRAY** query

引数［in］: SAFEARRAY** newXML

引数［out, retval］: SAFEARRAY** result

戻り型: HRESULT

説明:クエリーに指定した XPath 式に基づいて NeoCore XMS 内に XML を挿入し、挿入したノードの総数を返します。非 Unicode 文字エンコードを保持するために、クエリーも挿入する XML も、Unicode 文字ではなくバイト配列で記述します。

メソッド: modifyXML

構文:`HRESULT modifyXMLBinary(BSTR query, BSTR newXML, BSTR* result)`

引数［in］:BSTR query

引数［in］:BSTR newXML

引数［out, retval］:BSTR* result

戻り型: HRESULT

説明:クエリーに指定した XPath 式で NeoCore XMS 内の XML を識別し、その XML を newXML に指定した XML によって変更し、変更したノードの数と XPath 式に一致したノードの数を返します。

クエリーの例:`/ND/Example`

newXML の例:`<Example>modifiedXML</Example>`

結果の例:

```
<Modify-Results>
    <Modified-Nodes>1/Modified-Nodes>
    <Matching-Nodes>2/Matching-Nodes>
</Modify-Results>
```

メソッド: **modifyXMLBinary**

構文:`HRESULT modifyXMLBinary(SAFEARRAY** ppQuery, SAFEARRAY** ppNewXML, SAFEARRAY** ppResult`

引数［in］: SAFEARRAY** query

引数［in］: SAFEARRAY** newXML

引数［out, retval］: SAFEARRAY** result

戻り型: HRESULT

説明:クエリーに指定した XPath 式で NeoCore XMS 内の XML を識別し、その XML を newXML に指定した XML によって変更し、変更したノードの数と XPath 式に一致したノードの数を返します。非 Unicode 文字エンコードを保持するために、Unicode 文字ではなくバイト配列を返します。

メソッド: **copyXML**

構文:`HRESULT copyXML(BSTR query, BSTR* result)`

引数［in］:BSTR query

引数［out, retval］:BSTR* result

戻り型: HRESULT

説明:クエリーに指定した XPath 式で NeoCore XMS 内の XML を識別し、その XML をコピーします。XPath 式は、NeoCore XMS 内の 1 つの文書を一意に識別するものである必要があります。成功した場合は、コピーされたノードの数を返します。

クエリーの例:`/ND/[ExampleDocument]`

結果の例:

```
<Copy-Results>
    <CopyNumber>2</CopyNumber>
</Copy-Results>
```

メソッド: **copyXMLBinary**

構文:`HRESULT copyXMLBinary(SAFEARRAY** ppQuery, SAFEARRAY** ppResult)`

引数［in］: SAFEARRAY** query

引数［out, retval］: SAFEARRAY** result

戻り型: HRESULT

説明:クエリーに指定した XPath 式で NeoCore XMS 内の XML を識別し、その XML をコピーします。XPath 式は、NeoCore XMS 内の 1 つの文書を一意に識別するものである必要があります。成功した場合は、コピーされたノードの数を返します。非 Unicode 文字エンコードを保持するために、Unicode 文字ではなくバイト配列を返します。

メソッド: deleteXML

構文:`HRESULT deleteXML(BSTR query, BSTR* result)`

引数［in］:BSTR xpath

引数［out, retval］:BSTR* result

戻り型: HRESULT

説明:クエリーに指定した XPath 式で NeoCore XMS 内の XML を識別し、その XML を削除します。削除したノードの数を返します。

クエリーの例:`/ND/Example`

結果の例:

```
<Delete-Results>
    <Deleted-Nodes>2</Deleted-Nodes>
</Delete-Results>
```

メソッド: **deleteXMLBinary**

構文: `HRESULT deleteXMLBinary(SAFEARRAY** ppQuery, SAFEARRAY** ppResult`

引数［in］: SAFEARRAY** xpath

引数［out, retval］: SAFEARRAY** result

戻り型: HRESULT

説明: クエリーに指定した XPath 式で NeoCore XMS 内の XML を識別し、その XML を削除します。削除したノードの数を返します。成功した場合は、削除されたノードの数を返します。非 Unicode 文字エンコードを保持するために、Unicode 文字ではなくバイト配列を返します。

トランザクションコマンド

メソッド: startTransaction

構文:`HRESULT startTransaction(BSTR* result)`

引数［out, retval］:BSTR* result

戻り型: HRESULT

説明:NeoServer に startTransaction コマンドを送信します。コミット時にトランザクションをフラッシュするかどうかは、NeoXDBRuntime.xml の FlushFrequency によって決まります。開始したトランザクションの ID と設定値を返します。

結果の例:

```
<Transaction-Results>
    <Transaction-ID>127</Transaction-ID>
    <MaxDuration>300</MaxDuration>
    <InactivityDuration>30</InactivityDuration>
    <NoFlushOnCommit/>
    <IsolationLevel>REPEATABLE_READ</IsolationLevel>
</Transaction-Results>
```

メソッド: startTransactionControlFlush

構文:HRESULT startTransactionControlFlushVARIANT_BOOL flushOnCommit, BSTR* result)

引数［in］:VARIANT_BOOL flushOnCommit

引数［out, retval］:BSTR* result

戻り型: HRESULT

説明:NeoServer に startTransaction コマンドを送信します。コミット時にトランザクションをフラッシュするかどうかは、flushOnCommit 引数の値によって決まります。開始したトランザクションの ID と設定値を返します。

flushOnCommit の例: 1

結果の例:

```
<Transaction-Results>
    <Transaction-ID>148</Transaction-ID>
    <MaxDuration>300</MaxDuration>
    <InactivityDuration>30</InactivityDuration>
    <FlushOnCommit/>
    <IsolationLevel>REPEATABLE_READ</IsolationLevel>
</Transaction-Results>
```

メソッド: startTransactionControlAll

構文:`HRESULT startTransactionControlAll(int maxTransactionDuration, int inactivityDuration, VARIANT_BOOL flushOnCommit, BSTR* result)`

引数［in］:int maxTransactionDuration（秒単位）

引数［in］:int inactivityDuration（秒単位）

引数［in］:VARIANT_BOOL flushOnCommit

引数［out, retval］:BSTR* result

戻り型: HRESULT

説明:NeoServer に startTransaction コマンドを送信します。開始するトランザクションの特性は、maxTransactionDuration、inactivityDuration、flushOnCommit の各引数の値によって決まります。開始したトランザクションの ID と設定値を返します。

maxTransactionDuration:この startTransaction から後続の commitTransaction までの許容時間を秒単位で指定します。MaxTransactionDuration に指定した時間を超えると、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。

inactivityDuration:このトランザクションが非活動状態でいられる時間を秒単位で指定します。指定した時間を超えると、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。

flushOnCommit:コミット時にトランザクションをフラッシュするかどうかを指定します。

maxTransactionDuration の例: 60

inactivityDuration の例: 120

flushOnCommit の例:1（True/False）

結果の例:

```
<Transaction-Results>
    <Transaction-ID>150</Transaction-ID>
    <MaxDuration>60 </MaxDuration>
    <InactivityDuration>120</InactivityDuration>
    <FlushOnCommit/>
    <IsolationLevel>REPEATABLE_READ</IsolationLevel>
</Transaction-Results>
```

メソッド: commitTransaction

構文:`HRESULT commitTransaction(BSTR* result)`

引数［out, retval］:BSTR* result

戻り型: HRESULT

説明:NeoCore XMS にトランザクションコミットコマンドを送信します。コミットしたトランザクションの ID を返します。

結果の例:

```
<Transaction-Results>
    <Transaction-ID>154</Transaction-ID>
</Transaction-Results>
```

メソッド: rollbackTransaction

構文:`HRESULT rollbackTransaction(BSTR* result)`

引数［out, retval］:BSTR* result

戻り型: HRESULT

説明:NeoCore XMS にトランザクションロールバックコマンドを送信します。ロールバックしたトランザクションの ID を返します。

結果の例:

```
<Transaction-Results>
    <Transaction-ID>162</Transaction-ID>
</Transaction-Results>
```

メソッド: globalLock

構文:`HRESULT globalLock(BSTR lockType, BSTR* result)`

引数［in］:BSTR lockType

引数［out, retval］:BSTR* result

戻り型: HRESULT

説明:NeoCore XMS データベース全体をロックします。データベースの大半をロックする場合は、このメソッドを使用してグローバルロックを取得するのが望ましいと言えます。グローバルロックは、トランザクションのコミットまたはロールバックが実行されるまで保持されます。lockType 引数では、ロックのタイプを指定します。LockType に指定できるのは、SHARED、UPDATE、EXCLUSIVE のいずれかです。globalLock を呼び出すには、事前にトランザクションを開始しておく必要があります。付与したグローバルロックのタイプとトランザクションの ID が返されます。

lockType の例:EXCLUSIVE

結果の例:

```
<Global-Lock>
    <Level>EXCLUSIVE</Level>
    <Transaction-ID>208</Transaction-ID>
</Global-Lock>
```

XMS 管理コマンド

メソッド: setPassword

構文:`HRESULT setPassword(BSTR user, BSTR password, BSTR *result)`

引数［in］:BSTR user

引数［in］:BSTR password

引数［out, retval］:BSTR* result

戻り型: HRESULT

説明:NeoCore XMS のユーザーのパスワードを設定します。

ユーザーの例:Administrator

パスワードの例:mypassword

結果の例: `<success/>`

メソッド: getServerStatistics

構文:`HRESULT getServerStatistics(NeoStatisticType, BSTR* result)`

引数［in］:int statisticType

NeoStatisticType.ncStatAll = 0 =（ALL）

NeoStatisticType.ncStatAdmin = 1 =（ADMIN）

NeoStatisticType.ncStatStorage = 2 =（STORAGE）

NeoStatisticType.ncStatAccess = 3 =（ACCESS）

NeoStatisticType.ncStatBuffer = 4 =（BUFFER）

NeoStatisticType.ncStatTransaction = 5 =（TRANSACTION）

NeoStatisticType.ncStatWindow = 6 =（WINDOW）

引数［out, retval］:BSTR* result

戻り型: HRESULT

説明:NeoServer の現在の統計を取得し、NeoServer の現在の統計を含んだ XML を返します。

結果の例:

```
<Stats-Results>
    <Server-Info>
        NeoCore XMS Version xxxx for Windows
    </Server-Info>
    <Storage-Stats>
        ...
    </Storage-Stats>
    <Access-Stats>
        ...
    </Access-Stats>
    <Admin-Stats>
        ...
    </Admin-Stats>
    <Buffer-Stats>
        ...
    </Buffer-Stats>
    <Transaction-Stats>
        ...
    </Transaction-Stats>
</Stats-Results>
```

メソッド: clearServerStatistics

構文:`HRESULT clearServerStatistics(BSTR* result)`

引数［out, retval］:BSTR* result

戻り型: HRESULT

説明:NeoServer の統計をクリアし、クリア後の統計（つまり、クリア後の NeoServer の現在の統計を含んだ XML）を返します。

結果の例:

```
<Stats-Results>
    <Server-Info>
        NeoCore XMS version xxxx for Windows
    </Server-Info>
    <Storage-Stats>
        ...
    </Storage-Stats>
    <Access-Stats>
        ...
    </Access-Stats>
    <Admin-Stats>
        ...
    </Admin-Stats>
    <Buffer-Stats>
        ...
    </Buffer-Stats>
    <Transaction-Stats>
        ...
    </Transaction-Stats>
</Stats-Results>
```

メソッド: getServerVersion

構文:`HRESULT getServerVersion(BSTR* result)`

引数［out, retval］:BSTR* result

戻り型: HRESULT

説明:NeoCore XMS のバージョンを取得し、NeoCore XMS の現在のバージョンを含んだ XML を返します。

結果の例:

```
<Version-Results>
<Version>x.x.xx</Version>
<TimeStamp>May 27 2002 11:55:48</TimeStamp>
</Version-Results>
```

メソッド: setTraceLevel

構文:`HRESULT setTraceLevel(BSTR level, BSTR* result)`

引数［in］:BSTR level

引数［out, retval］:BSTR* result

戻り型: HRESULT

説明:NeoCore XMS のエラーログシステムのトレースレベルを設定します。

レベルの例:ERR:LOG_all または INFO:LOG_Performance

結果の例:

```
<success>Trace Levels updated</success>
```

メソッド: getTraceLevels

構文:`HRESULT getTraceLevels(BSTR* result)`

引数［out, retval］:BSTR* result

戻り型: HRESULT

説明:NeoCore XMS のエラーログが使用している現在のトレースレベルを示す文字列を返します。

結果の例:

```
<TraceLevel>FATAL:LOG_all;WARNING:LOG_all;ERR:LOG_all</TraceLevel>
```

メソッド: activateAccessControl

構文:`HRESULT activateAccessControl(BSTR* result)`

引数［out, retval］:BSTR* result

戻り型: HRESULT

説明:NeoCore XMS のアクセス制御を有効化します。アクセス制御を適用するには、事前に NeoCore XMS にアクセス制御文書を格納しておく必要があります。デフォルトのアクセス制御文書の位置は、NeoCore/neoxml/config ディレクトリです。このメソッドを呼び出す前に、NeoCore/neoxml/config ディレクトリに DefaultAC.xml 文書を格納してください。

結果の例:

```
<success>Access Control is activated</success>
```

アクセス制御

COM オブジェクトの ACAdmin インターフェースでは、データベース内に存在するアクセス制御用のグループ、ユーザー、ルールを管理します。

メソッド: initialize

構文:`HRESULT initialize([in] INeoCoreAPI* apiInterfaceObj)`

引数:［in］INeoCoreAPI* apiInterfaceObj（接続してログインしている NeoCoreAPI オブジェクトのインスタンス）

戻り型: HRESULT

説明:ACAdmin オブジェクトを初期化します。他のあらゆる ACAdmin メソッドよりも前に呼び出す必要があります。

メソッド: createGroup

構文:`HRESULT createGroup([in] BSTR groupname, [in] BSTR description, [out, retval] BSTR* result)`

引数:［in］BSTR groupname（作成するグループの名前）

引数:［in］BSTR description（このグループの説明）

引数:［out, retval］BSTR* result

戻り型: HRESULT

説明:グループを作成します。

結果の例: 成功した場合は<success/>を返し、そうでない場合は問題を記述したエラーを生成します。

メソッド: createGroupRule

構文:`HRESULT createGroupRule([in] BSTR groupname, [in] NeoOperationType operation, [in] BSTR target, [in] BSTR condition, [out, retval] BSTR* result)`

引数:［in］BSTR groupname（このルールを適用するグループの名前）

引数:［in］NeoOperationType operation（このルールによって制限する操作のタイプ）

引数:［in］BSTR target（制限対象の XML ノードを指定した XPath）

引数:［in］BSTR condition（制限対象のターゲットノードに関して存在している必要がある条件があれば、その条件を指定します）

引数:［out, retval］BSTR* result

戻り型: HRESULT

説明:特定のグループに適用するルールを作成します。

結果の例: 成功した場合は<success/>を返し、そうでない場合は問題を記述したエラーを生成します。

メソッド: createUser

構文:`HRESULT createUser([in] BSTR username, [in] BSTR groupname, [in] BSTR firstName, [in] BSTR lastName, [in] BSTR password, [out, retval] BSTR* result)`

引数:［in］BSTR username（作成するユーザーの名前）

引数:［in］BSTR groupname（この新しいユーザーを割り当てるグループ）

引数:［in］BSTR firstName（このユーザーのファーストネーム）

引数:［in］BSTR lastName（このユーザーのラストネーム）

引数:［in］BSTR password（このユーザーのパスワード）

引数:［out, retval］BSTR* result

戻り型: HRESULT

説明:ユーザーを作成します。

結果の例: 成功した場合は<success/>を返し、そうでない場合は問題を記述したエラーを生成します。

メソッド: createUserRule

構文:`HRESULT createUserRule([in] BSTR username, [in] NeoOperationType operation, [in] BSTR target, [in] BSTR condition, [out, retval] BSTR* result)`

引数:［in］BSTR username（このルールを適用するユーザーの名前）

引数:［in］NeoOperationType operation（このルールによって制限する操作のタイプ）

引数:［in］BSTR target（制限対象の XML ノードを指定した XPath）

引数:［in］BSTR condition（制限対象のターゲットノードに関して存在している必要がある条件があれば、その条件を指定します）

引数:［out, retval］BSTR* result

戻り型: HRESULT

説明:特定のユーザーに適用するルールを作成します。

結果の例: 成功した場合は<success/>を返し、そうでない場合は問題を記述したエラーを生成します。

メソッド: **modifyGroupDescription**

構文:`HRESULT modifyGroupDescription([in] BSTR groupname, [in] BSTR description, [out, retval] BSTR* result)`

引数:［in］BSTR groupname（変更するグループの名前）

引数:［in］BSTR description（このグループの新しい説明）

引数:［out, retval］BSTR* result

戻り型: HRESULT

説明:グループの説明を変更します。

結果の例: 成功した場合は<success/>を返し、そうでない場合は問題を記述したエラーを生成します。

メソッド: modifyUserGroupname

構文:`HRESULT modifyUserGroupname([in] BSTR username, [in] BSTR groupname, [out, retval] BSTR* result)`

引数:［in］BSTR username（変更するユーザーのユーザー名）

引数:［in］BSTR groupname（このユーザーを割り当てるグループ）

引数:［out, retval］BSTR* result

戻り型: HRESULT

説明:このユーザーを別のグループに割り当てます。

結果の例: 成功した場合は<success/>を返し、そうでない場合は問題を記述したエラーを生成します。

メソッド: modifyUserFirstname

構文:`HRESULT modifyUserFirstname([in] BSTR username, [in] BSTR firstName, [out, retval] BSTR* result)`

引数:［in］BSTR username（変更するユーザーのユーザー名）

引数:［in］BSTR firstName（このユーザーの新しいファーストネーム）

引数:［out, retval］BSTR* result

戻り型: HRESULT

説明:ユーザーのファーストネームを変更します。

結果の例: 成功した場合は<success/>を返し、そうでない場合は問題を記述したエラーを生成します。

メソッド: modifyUserLastname

構文:`HRESULT modifyUserLastname([in] BSTR username, [in] BSTR lastName, [out, retval] BSTR* result)`

引数:［in］BSTR username（変更するユーザーのユーザー名）

引数:［in］BSTR lastName（このユーザーの新しいラストネーム）

引数:［out, retval］BSTR* result

戻り型: HRESULT

説明:ユーザーのラストネームを変更します。

結果の例: 成功した場合は<success/>を返し、そうでない場合は問題を記述したエラーを生成します。

メソッド: modifyUserPassword

構文:`HRESULT modifyUserPassword([in] BSTR username, [in] BSTR password, [out, retval] BSTR* result)`

引数:［in］BSTR username（変更するユーザーのユーザー名）

引数:［in］BSTR password（このユーザーの新しいパスワード）

引数:［out, retval］BSTR* result

戻り型: HRESULT

説明:指定のユーザーのパスワードを変更します。

結果の例: 成功した場合は<success/>を返し、そうでない場合は問題を記述したエラーを生成します。

メソッド: deleteGroup

構文: `HRESULT deleteGroup([in] BSTR groupname, [out, retval] BSTR* result)`

引数:［in］BSTR groupname（削除するグループの名前）

引数:［out, retval］BSTR* result

戻り型: HRESULT

説明:グループを削除します。

結果の例: 成功した場合は<success/>を返し、そうでない場合は問題を記述したエラーを生成します。

メソッド: deleteGroupRule

構文: `HRESULT deleteGroupRule([in] BSTR groupname, [in] NeoOperationType operation, [in] BSTR target, [in] BSTR condition, [out, retval] BSTR* result)`

引数:［in］BSTR groupname（このルールの適用先のグループの名前）

引数:［in］NeoOperationType operation（このルールによって制限されている操作のタイプ）

引数:［in］BSTR target（制限されている XML ノードを指定した XPath）

引数:［in］BSTR condition（制限対象のターゲットノードに関して存在している必要がある条件があれば、その条件を指定します）

戻り型: HRESULT

説明:グループに適用するルールを削除します。

結果の例: 成功した場合は<success/>を返し、そうでない場合は問題を記述したエラーを生成します。

メソッド: deleteUser

構文: `HRESULT deleteUser([in] BSTR username, [out, retval] BSTR* result)`

引数:［in］BSTR username（削除するユーザーのユーザー名）

引数:［out, retval］BSTR* result

戻り型: HRESULT

説明:ユーザーを削除します。

結果の例: 成功した場合は<success/>を返し、そうでない場合は問題を記述したエラーを生成します。

メソッド: deleteUserRule

構文: `deleteUserRule([in] BSTR username, [in] NeoOperationType operation, [in] BSTR target, [in] BSTR condition, [out, retval] BSTR* result)`

引数:［in］BSTR username（このルールの適用先のユーザーの名前）

引数:［in］NeoOperationType operation（このルールによって制限されている操作のタイプ）

引数:［in］BSTR target（制限されている XML ノードを指定した XPath）

引数:［in］BSTR condition（制限対象のターゲットノードに関して存在している必要がある条件があれば、その条件を指定します）

戻り型: HRESULT

説明:ユーザーに適用するルールを削除します。

結果の例: 成功した場合は<success/>を返し、そうでない場合は問題を記述したエラーを生成します。

COM オブジェクトによる NeoCore XMS データベースの操作

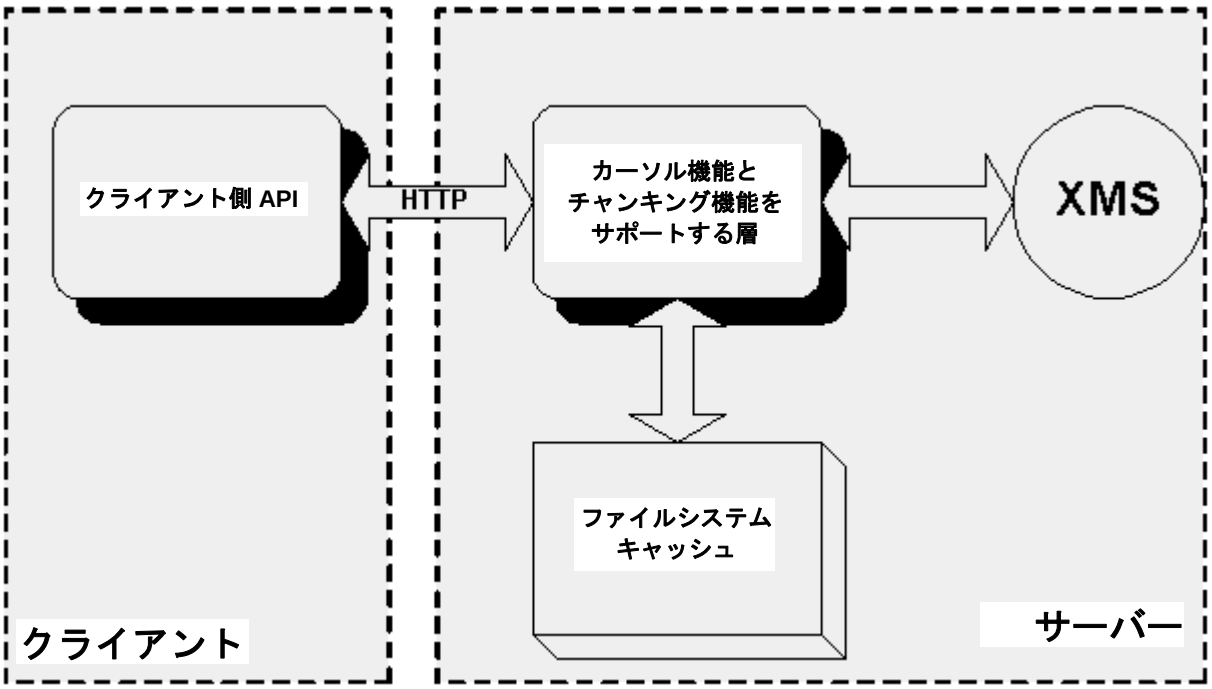
COM オブジェクトによって NeoCore XMS データベースを操作するには、以下のようにします。

1. *connectToNeoCoreXMS* メソッドを使用して COM オブジェクトを初期化します（前述の NeoCore API メソッド一覧表で *connectToNeoCoreXMS メソッド*の項を参照してください）。
2. COM オブジェクトを初期化したら、login メソッドを使用して NeoCore XMS データベースにログインすることもできます（前述の NeoCore API メソッド一覧表で login メソッドの項を参照してください）。
3. それぞれのニーズに適したメソッドを使用して NeoCore XMS データベースを操作します。
4. 最後に、*closeConnection* メソッドを呼び出します（前述の NeoCore API メソッド一覧表で *closeConnection メソッド*の項を参照してください）。

XMS のカーソル機能とチャンク機能

カーソル機能とチャンク機能によって、大きな結果セットをサポートできます。具体的には、非常に大きなクエリーから結果を返すときの結果処理の方法を設定できるということです。さらに、一定数の結果だけを返す（つまり、非常に大きな結果セットの一部を切り捨てる）ための機能もあります。

- NeoServer 側のサポートによって、結果はキャッシュに入れられるので、そのセッションが終了するまで、またはクライアントがキャッシュを解放するまで、クライアントはその結果にアクセスできます。結果は切断されています。
- クライアント側のサポートによって、結果を返す方法を設定し、結果データのカーソル処理や、NeoServer からの追加チャンクのロードを必要に応じて実行できます。



IQueryResult

メソッド: destroyNeoResultChunk

構文: `HRESULT destroyNeoResultChunk( [in] int resultChunkID, [out, retval] VARIANT_BOOL* wasDestroyed)`

引数［in］: int resultChunkID（破棄する NeoResultChunk の ID）

引数［out, retval］:VARIANT_BOOL* wasDestroyed

戻り型: HRESULT

説明:NeoResultChunk を破棄します。基本的には、NeoResultProfile に指定されている NeoResultChunk のロードと破棄をクライアントが手動で管理したい場合に使用します。resultChunkID で指定した NeoResultChunk がクライアントにまったく転送されていない場合（つまり、NEVER_LOADED 状態の場合）、そのチャンクは NeoServer 上で破棄されます。いったん破棄した NeoResultChunk の内容を再び取得することはできません。

wasDestroyed の例:NeoResultChunk が LOADED または NEVER_LOADED の場合は True、そうでない場合は False になります（NeoResultChunk の状態は DESTROYED）。

メソッド: isTruncated

構文: `HRESULT isTruncated([out, retval] VARIANT_BOOL* wasTruncated`

引数:

引数［out, retval］:VARIANT_BOOL* wasTruncated

戻り型: HRESULT

説明: NeoQueryResult の一部が NeoServer によって切り捨てられたかどうかを示します。

wasTruncated の例:この NeoQueryResult の一部が切り捨てられた場合は True、そうでない場合は False になります。

メソッド: isValidNeoResultChunk

構文: `HRESULT isValidNeoResultChunk([in] int resultChunkID, [out, retval] VARIANT_BOOL* isValid)`

引数［in］:int resultChunkID（NeoResultChunk の ID）

引数［out, retval］： VARIANT_BOOL* isValid

戻り型: HRESULT

説明: この NeoQueryResult に関してこの resultChunkID が存在するかどうかを示します。有効な ID とは、単にその ID が存在するという意味であり、NeoResultChunk の状態とは無関係です。

isValid の例:resultChunkID が存在する場合は True、そうでない場合は False になります。

メソッド: isUsableNeoCursorPosition

構文: `HRESULT isUsableNeoCursorPosition([in] CURRENCY cursorPosition, [out, retval] VARIANT_BOOL* isUsable`

引数［in］:CURRENCY cursorPosition

注:CURRENCY を使用するのは、64 ビット数が必要だったからです。

引数［out, retval］： VARIANT_BOOL* isUsable

戻り型: HRESULT

説明: cursorPosition が使用可能かどうかを示します。ここで言う「使用可能」とは、cursorPosition が存在しており、その cursorPositon を保持している NeoResultChunk が DESTROYED の状態になっていない（つまり、LOADED か NEVER_LOADED の状態になっている）という意味です。

isUsable の例: cursorPosition が使用可能の場合は True、そうでない場合は False になります。

メソッド: get_NeoResultChunkState

構文: `HRESULT get_NeoResultChunkState([in] int resultChunkID, [out, retval] NeoResultChunkState* state`

引数［in］: int resultChunkID（NeoResultChunk の ID）

引数［out, retval］: NeoResultChunkState* state

戻り型: HRESULT

説明:resultChunkID で指定した NeoResultChunk の現在の状態を取得します。

state の例 **:** NeoResultChunk の状態です。以下の状態があります。

NeoResultChunkState.ncDestroyed= -1 =（DESTROYED）以前にロードされ、削除されました（この軸要素のチャンクに関する XML は、もはやクライアント上にも NeoServer 上にも存在しません）。

NeoResultChunkState.ncNeverLoaded = 0 =（NEVER_LOADED）NeoServer から XML と共にロードされていません（XML は NeoServer 上に存在しますが、クライアントに転送されていません）。

NeoResultChunkState.ncLoaded = 1 =（LOADED）NeoServer から XML と共にロードされています（この NeoResultChunk に関する XML は、もはや NeoServer 上に存在しません）。

メソッド: loadNeoResultChunk

構文: `HRESULT loadNeoResultChunk([in] int resultChunkID, [out, retval] VARIANT_BOOL* wasLoaded`

引数［in］:int resultChunkID（ロードする NeoResultChunk の ID）

引数［out, retval］:VARIANT_BOOL* wasLoaded

戻り型: HRESULT

説明:resultChunkID で指定した NeoResultChunk をロードします。基本的には、NeoResultProfile 内の NeoResultChunk を手動で管理したいときに使用します。

wasLoaded の例:NeoResultChunk がロードされた場合は True、そうでない場合は False になります（NeoResultChunk はすでに LOADED の状態になっています）。

メソッド: get_StatisticsXML

構文: `HRESULT get_StatisticsXML([out, retval] BSTR* result`

引数［in］:

引数［out, retval］:BSTR* result

戻り型: HRESULT

説明: この NeoQueryResult に関する統計を XML 形式で取得します。返された XML(char*)の確認が済んだら、releaseXML(char*)を呼び出して XML を解放します。

例:

```
<Query-Results>
      <ID>0</ID>
      <Truncated>1</Truncated>
      <SessionID>12345678</SessionID>
      <TotalNeoResultChunks>3</TotalNeoResultChunks>
      <TotalAxisElements>2500</TotalAxisElements>
      <LoadedNeoResultChunks>1</LoadedNeoResultChunks>
      <LoadedAxisElements>1000</LoadedAxisElements>
```

```
      <NeoResultChunk>
           <ID>0</ID>
           <Offset>0</Offset>
           <AxisElements>1000</AxisElements>
           <State>LOADED</State>
           <NeoCursorLoc>999</NeoCursorLoc>
      </NeoResultChunk>
      <NeoResultChunk>
           <ID>1</ID>
           <Offset>1000</Offset>
           <AxisElements>1000</AxisElements>
           <State>DESTROYED</State>
      </NeoResultChunk>
      <NeoResultChunk>
           <ID>2</ID>
           <Offset>2000</Offset>
           <AxisElements>500</AxisElements>
           <State>NEVER_LOADED</State>
      </NeoResultChunk>
</Query-Results>
```

メソッド: get_NeoCursorsNeoResultChunk

構文:`HRESULT get_NeoCursorsNeoResultChunk([out, retval] int* resultChunkID`

引数［in］:

引数［out, retval］:int* resultChunkID

戻り型: HRESULT

説明: NeoCursor が現在置かれている NeoResultChunk の ID を取得します。

resultChunkID の例:NeoCursor が現在置かれている NeoResultChunk の ID です。

メソッド: get_NeoResultChunkXML

構文: `HRESULT get_NeoResultChunkXML([in] int resultChunkID, [out, retval] BSTR* result`

引数［in］: int resultChunkID

引数［out, retval］:BSTR* result

戻り型: HRESULT

説明:resultChunkID で指定した NeoResultChunk に含まれている XML を取得します。また、NeoResultProfile の resultChunkTransferMode が Auto（つまり 1）に設定されており、要求対象の XML が存在する NeoResultChunk の状態が NEVER_LOADED の場合、このメソッドは NeoResultChunk を自動的にロードします。

注:このメソッドによって NeoResultChunk が破棄されることはありません。NeoResultProfile の resultChunkDiscardMode が AUTO に設定されていても、その設定は無視されます。

例:

```
<Query-Results>
    <YourData>1</YourData>
    <YourData>1</YourData>
</Query-Results>
```

メソッド: get_NeoResultChunkXMLBinary

構文: `HRESULT get_NeoResultChunkXMLBinary([in] int resultChunkID, [out, retval] SAFEARRAY** result`

引数［in］: int resultChunkID

引数［out, retval］:SAFEARRAY** result

戻り型: HRESULT

説明:非 Unicode 文字エンコードを保持するために、Unicode 文字ではなくバイト配列を取得します。

メソッド: get_NeoCursorXML

構文:`HRESULT get_NeoCursorXML([in] int additionalPositions, [out, retval] BSTR* result`

引数［in］:int additionalPositions（additionalPositions が正の値の場合は、現在のカーソルの位置とそれより後の N 個のカーソルの位置の XML が返され、負の値の場合は、現在のカーソルの位置とそれより前の N 個のカーソルの位置の XML が返され、ゼロの場合は、現在のカーソルの位置の XML が返されます）

引数［out, retval］:BSTR* result

戻り型: HRESULT

説明:現在のカーソルの位置とそれより前か後のカーソルの位置の XML を取得します。NeoCursor は元の位置のままです。また、NeoResultProfile の resultChunkTransferMode が Auto（つまり 1）に設定されており、要求対象の XML が入り込む別の NeoResultChunk の状態が NEVER_LOADED の場合、このメソッドは NeoResultChunk を自動的にロードします。

注:このメソッドによって NeoResultChunk が破棄されることはありません。NeoResultProfile の resultChunkDiscardMode が AUTO に設定されていても、その設定は無視されます。

例:

```
</Query-Results>

     <YourData>1</YourData>

     <YourData>1</YourData>

</Query-Results>
```

メソッド: get_NeoCursorXMLBinary

構文:`HRESULT get_NeoCursorXML([in] int additionalPositions, [out, retval] SAFEARRAY** result`

引数［in］:int additionalPositions（additionalPositions が正の値の場合は、現在のカーソルの位置とそれより後の N 個のカーソルの位置の XML が返され、負の値の場合は、現在のカーソルの位置とそれより前の N 個のカーソルの位置の XML が返され、ゼロの場合は、現在のカーソルの位置の XML が返されます）

引数［out, retval］:SAFEARRAY** result

戻り型: HRESULT

説明:非 Unicode 文字エンコードを保持するために、Unicode 文字ではなくバイト配列を取得します。

メソッド: get_NeoCursorPosition

構文:

引数［in］:

引数［out, retval］:CURRENCY* position

戻り型: HRESULT

説明:この NeoQueryResult 内の NeoCursor の現在の位置を取得します。

例:この NeoQueryResult 内の NeoCursor の現在の位置を返します。

メソッド: get_TotalNeoResultChunks

構文:`HRESULT get_TotalNeoResultChunks([out, retval] int* total`

引数［in］:

引数［out, retval］:int* total

戻り型: HRESULT

説明:この NeoQueryResult に組み込める NeoResultChunk の最大数を取得します。

例:軸要素の数を返します。

メソッド: get_TotalAxisElementsNeoResultChunk

構文:`HRESULT get_TotalAxisElementsNeoResultChunk([in] int resultChunkID, [out,retval] int* total`

引数［in］:int resultChunkID（NeoResultChunk の ID）

引数［out, retval］:int* total

戻り型: HRESULT

説明:この NeoResultChunk に組み込める軸要素の最大数を取得します。

例:resultChunkID に組み込める軸要素の数を返します。

メソッド: get_LoadedNeoResultChunks

構文:`HRESULT get_LoadedNeoResultChunks([out, retval] int* numLoaded`

引数［in］:

引数［out, retval］:int* numLoaded

戻り型: HRESULT

説明:この NeoQueryResult がロードしている NeoResultChunk（つまり、クライアント上に現在ロードされている軸要素）の現在の数を取得します。

例:軸要素の現在の数を返します。

メソッド: get_TotalAxisElements

構文:`HRESULT get_TotalAxisElements([out,retval] CURRENCY* total`

引数［in］:

引数［out, retval］:CURRENCY* total

戻り型: HRESULT

説明:この NeoQueryResult に組み込める軸要素の最大数を取得します。

例:軸要素の現在の数を返します。

メソッド: get_LoadedAxisElements

構文:`HRESULT get_LoadedAxisElements([out, retval] CURRENCY* numLoaded`

引数［in］:CURRENCY* numLoaded

引数［out, retval］:

戻り型: HRESULT

説明:この NeoQueryResult がロードしている軸要素（つまり、クライアント上に現在ロードされている軸要素）の現在の数を取得します。

例:軸要素の現在の数を返します。

メソッド: moveNeoCursor

構文:`HRESULT moveNeoCursor([in] CURRENCY movement, [out, retval] CURRENCY* cursorPosition`

引数［in］:movement（この NeoQueryResult 内で順方向に移動する場合は正の数を指定し、逆方向に移動する場合は負の数を指定します）

引数［out, retval］:CURRENCY* cursorPosition

戻り型: HRESULT

説明:この NeoQueryResult 内で NeoCursor を順方向または逆方向に移動します。また、NeoResultProfile の resultChunkDisacardMode が Auto（つまり 1）に設定されており、NeoCursor が NeoResultChunk から出る場合、その NeoResultChunk は自動的に破棄されます。さらに、NeoResultProfile の resultChunkTransferMode が Auto（つまり 1）に設定されており、NeoCursor が入る NeoResultChunk の状態が NEVER_LOADED の場合、このメソッドは NeoResultChunk を自動的にロードします。

メソッド: repositionNeoCursor

構文:`HRESULT repositionNeoCursor([in] CURRENCY absoluteCursorLocation,[out, retval] CURRENCY* cursorPosition`

引数［in］:CURRENCY absoluteCursorLocation（この NeoQueryResult 内に NeoCursor を配置するための絶対位置。NeoQueryResult の下限値は常にゼロです）

引数［out, retval］:CURRENCY* cursorPosition

戻り型: HRESULT

説明:この NeoQueryResult 内で NeoCursor の位置を変更します。また、NeoResultProfile の resultChunkDisacardMode が Auto（つまり 1）に設定されており、NeoCursor が NeoResultChunk から出る場合、その NeoResultChunk は自動的に破棄されます。さらに、NeoResultProfile の resultChunkTransferMode が Auto（つまり 1）に設定されており、NeoCursor が入る NeoResultChunk の状態が NEVER_LOADED の場合、このメソッドは NeoResultChunk を自動的にロードします。

メソッド: isEmptyResult

構文:`HRESULT isEmptyResult([out, retval] VARIANT_BOOL* isEmpty`

引数［in］:

引数［out, retval］:VARIANT_BOOL* isEmpty

戻り型: HRESULT

説明:NeoQueryResult が空であるかどうかを示します。

例:isEmpty は、NeoQueryResult が空の場合は True、そうでない場合は False になります。

IResultProfile

メソッド: initializeProfile:

クエリー結果に対するカーソル機能とチャンク機能の構成情報を指定します。

構文:`HRESULT initializeProfile([in] int AxisElementsPerChunk, [in] CURRENCY maxResultAxisElements, [in] CURRENCY initialCursorPosition, [in] NeoChunkCaptureMode resultChunkCaptureMode, [in] NeoExceededResultsMode ExceededResultsMode, [in] NeoMode ResultChunkTransferMode, [in] NeoMode ResultChunkDiscardMode)`

引数［in］:int AxisElementsPerChunk:結果チャンク 1 個あたりの軸要素（ノード）の数です。

引数［in］:CURRENCY maxResultAxisElements:返す軸要素の最大数です。

引数［in］:CURRENCY initialCursorPosition:返す初期カーソル位置です。デフォルトは 0 です。

引数［in］:NeoChunkCaptureMode resultChunkCaptureMode:resultChunkCaptureMode です。

NeoChunkCaptureMode.ncChunkSnapshot = 0 =（SNAPSHOT）クエリー実行時の静的な結果（つまり、その時点でのデータのスナップショット）が返されます。

NeoChunkCaptureMode.ncChunkConnected = 1 =（CONNECTED）これは実装されていません。

引数［in］:NeoExceededResultsMode ExceededResultsMode:許容構成値を超える大きな結果の処理方法を指定します。

NeoExceededResultsMode.ncReject = 0 =（REJECT）結果が大きすぎる場合は、エラーが返されます。

NeoExceededResultsMode.ncTruncate = 1 =（TRUNCATE）指定のクエリー結果構成値に収まるように結果セットの一部が切り捨てられます。

引数［in］:NeoMode ResultChunkTransferMode:ロードが行われていない位置にカーソルが移動した場合に、クライアント側で結果チャンクを自動的にロードするかどうかを指定します。

NeoMode.ncManual = 0 =（MANUAL）ロードが行われていないチャンクにカーソルが移動した場合は、例外が生成されます。

NeoMode.ncAuto = 1 =（AUTO）クライアント上にロードされていない結果チャンクにカーソルが移動した場合は、その結果チャンクが自動的にロードされます。

引数［in］:NeoMode ResultChunkDiscardMode:NeoServer 上で結果チャンクリソースを自動的にクリーンアップするかどうかを指定します。

NeoMode.ncManual = 0 =（MANUAL）IQueryResult.destroyNeoResultChunk を呼び出して、結果チャンクのクリーンアップを実行できます。

NeoMode.ncAuto = 1 =（AUTO）チャンクがクライアント上にロードされた時点で、NeoServer 上の NeoResultChunk がクリーンアップされます。

引数［in］:

メソッド: isInitialized :

クエリー結果オブジェクトが初期化されている場合は True を返します。

構文:`HRESULT isInitialized([out, retval] VARIANT_BOOL* wasInitialized);`

引数［out, retval］:VARIANT_BOOL* wasInitialized

メソッド: get_XML:

この NeoResultProfile に相当する XML を返します。

構文:`HRESULT get_XML([out, retval] BSTR* result)`

引数［out, retval］:BSTR* result:

XML の例:

```
<NeoResultProfile>
    <AxisElementsPerChunk>1000</AxisElementsPerChunk>
    <MaxResultAxisElements>100000</MaxResultAxisElements>
    <InitialCursorPosition>0</InitialCursorPosition>
    <ResultChunkCaptureMode>1</ResultChunkCaptureMode>
    <ExceededResultsMode>1</ExceededResultsMode>
    <ResultChunkTransferMode>1</ResultChunkTransferMode>
    <ResultChunkDiscardMode>1</ResultChunkDiscardMode>
</NeoResultProfile>
```

COM の使用法に関する Visual Basic （VB）サンプルコード

注:以下の内容は、Windows クライアントシステムだけに該当します。

インストール時に API オプションを選択すると、以下のソースコードを含んだ VB プロジェクトが以下のパス（デフォルト）にインストールされます。

```
C:\NeoCore\API\Samples\COM_VB
```

```
'-------------------------------------------------------------------------------

'  Example Visual Basic Code using the NEOCOREXMSLib.NeoCoreAPI COM Connector

'  Copyright 2002, Xpriori, LLC

'  September 30, 2002  Version 2.7

'-------------------------------------------------------------------------------

Option Explicit


'-- NeoXMS Required Connection information


'-- Host IP of your NeoCore XMS Server, IP Address (e.g 172.16.3.124) or localhost

Private Const gstrHost As String = "localhost"


'-- Port you wish to connect to the NeoCore XMS Server on

Private Const gstrPort As String = "7700"


'-- User Name you wish to connect to the NeoCore XMS Server as

Private Const gstrUser As String = "Administrator"


'-- Password for the User
```

```
Private Const gstrPassword As String = "admin"



Private Sub cmdRun_Click()

   results.Text = "Example Test Client" & vbCrLf & vbCrLf & vbCrLf

   '--Prevent the user from clicking twice--
   cmdRun.Enabled = False

   '-- Examples using each of the exposed COM Methods
   exampleUsingNeoCoreXMSviaCOM

   '--Enable the run button again--
   cmdRun.Enabled = True

   results.Text = results.Text & vbCrLf & vbCrLf & "Finished"

Errorhandler:
   If Err.Number = 0 Then
   Else
      '--Display a message to the user--
      MsgBox Err.Number & " - " & Err.Description
      '--Clear the error--
      Err.Clear
```

```
    End If
End Sub



Private Sub exampleUsingNeoCoreXMSviaCOM()

    Dim objNeoAPI As NEOCOREXMSLib.NeoCoreAPI
    Dim objResultProfile As NEOCOREXMSLib.resultProfile
    Dim objQueryResult As NEOCOREXMSLib.queryResult

    Dim strResult As String
    Dim strQuery As String

    '-- Create an instance of the NeoCoreAPI COM object
    Set objNeoAPI = New NEOCOREXMSLib.NeoCoreAPI
    Set objResultProfile = New NEOCOREXMSLib.resultProfile


    '--Initialize (Establish a connection to NeoCore XMS Server
    '-- Must create an instance of the NeoCoreAPI
    objNeoAPI.connectToNeoCoreXMS gstrHost, gstrPort

    '--Login--
    printHeader "Logging in..."
    strResult = objNeoAPI.login(gstrUser, gstrPassword)
```

```
printResults "Login result, SessionID:" & vbCrLf & strResult

'-- isLoggedIn call after Login --
printHeader "Checking if logged in..."
strResult = objNeoAPI.isLoggedIn()
printResults "isLoggedIn result after Login:" & vbCrLf & strResult

'--Store some XML--
printHeader "Store some XML..."
strResult = objNeoAPI.storeXML("<Test1>storedXML</Test1>", "", "")
printResults "storeXML result:" & vbCrLf & strResult

'-- Example of using queryXML Command --
printHeader "Query our XML..."
strQuery = "/ND/Test1"
strResult = objNeoAPI.queryXML(strQuery)
printResults "queryXML result:" & vbCrLf & strResult

'-- Example of using queryXMLProfiled Command --
printHeader "Query our XML with Cursoring and Chunking Support..."
printHeader "  Creating a Result Profile"
Call objResultProfile.initializeProfile(0, 100, 0, 0, 1, 1, 1)
printHeader "  Getting ResultProfile XML"
strResult = objResultProfile.get_XML
printResults "ResultProfile XML result:" & vbCrLf & strResult
```

```
strQuery = "for $i in 0 to 100 return concat(" & Chr$(34) & _

        "Axis Element is =" & Chr$(34) & ", $i)"

Set objQueryResult = objNeoAPI.queryXMLProfiled(strQuery, objResultProfile)

printResults "isEmptyResult result:" & vbCrLf & objQueryResult.isEmptyResult

printResults "isTruncated result:" & vbCrLf & objQueryResult.isTruncated

printResults "get_LoadedAxisElements result:" & vbCrLf & objQueryResult.get_LoadedAxisElements

printResults "get_LoadedNeoResultChunks result:" & vbCrLf &
objQueryResult.get_LoadedNeoResultChunks


printResults "get_TotalAxisElements result:" & vbCrLf & objQueryResult.get_TotalAxisElements()


printResults "get_TotalNeoResultChunks result:" & vbCrLf & objQueryResult.get_TotalNeoResultChunks

printResults "get_TotalAxisElementsNeoResultChunk(0) result:" & vbCrLf &
objQueryResult.get_TotalAxisElementsNeoResultChunk(0)

printResults "get_NeoResultChunkState(0) result:" & vbCrLf &
objQueryResult.get_NeoResultChunkState(0)


printResults "get_StatisticsXML result:" & vbCrLf & objQueryResult.get_StatisticsXML


printResults "get_NeoCursorXML 0 - 20 result:" & vbCrLf & objQueryResult.get_NeoCursorXML(20)

printResults "repositionNeoCursor to 99 result:" & vbCrLf & objQueryResult.repositionNeoCursor(99)

printResults "get_NeoCursorXML current cursor location result:" & vbCrLf &
objQueryResult.get_NeoCursorXML(0)
```

```
    printResults "moveNeoCursor to -10 result:" & vbCrLf & objQueryResult.moveNeoCursor(-10)

    printResults "get_NeoCursorXML current cursor location result:" & vbCrLf &
    objQueryResult.get_NeoCursorXML(0)

    printResults "get_NeoResultChunkXML(0) result:" & vbCrLf &
    objQueryResult.get_NeoResultChunkXML(0)

    Set objQueryResult = Nothing

    Set objResultProfile = Nothing



    '--Logout, should logout to free up the session --

    printHeader "Log out..."

    strResult = objNeoAPI.logout()

    printResults "Logout result:" & vbCrLf & strResult


    '--Clean up, should always cleanup--


    '-- Close the connection --

    objNeoAPI.closeConnection

    '-- Force VB to discard the object when finished

    Set objNeoAPI = Nothing


    strResult = vbNullString


End Sub
```

```
Private Function printResults(resultStr As String)
   Dim modified As String

   'Replacing the carriage returns in the command's result string merely for display purposes
   modified = Replace$(resultStr, Chr$(10), vbCrLf)
   results.Text = results.Text & modified & vbCrLf & vbCrLf

End Function


   Private Function printHeader(header As String)
   Dim strHeader As String
   strHeader = strHeader & "*************************************************************************" & vbCrLf
   strHeader = strHeader & header & vbCrLf
   strHeader = strHeader & "*************************************************************************" & vbCrLf
   results.Text = results.Text & strHeader
   strHeader = ""
End Function
```

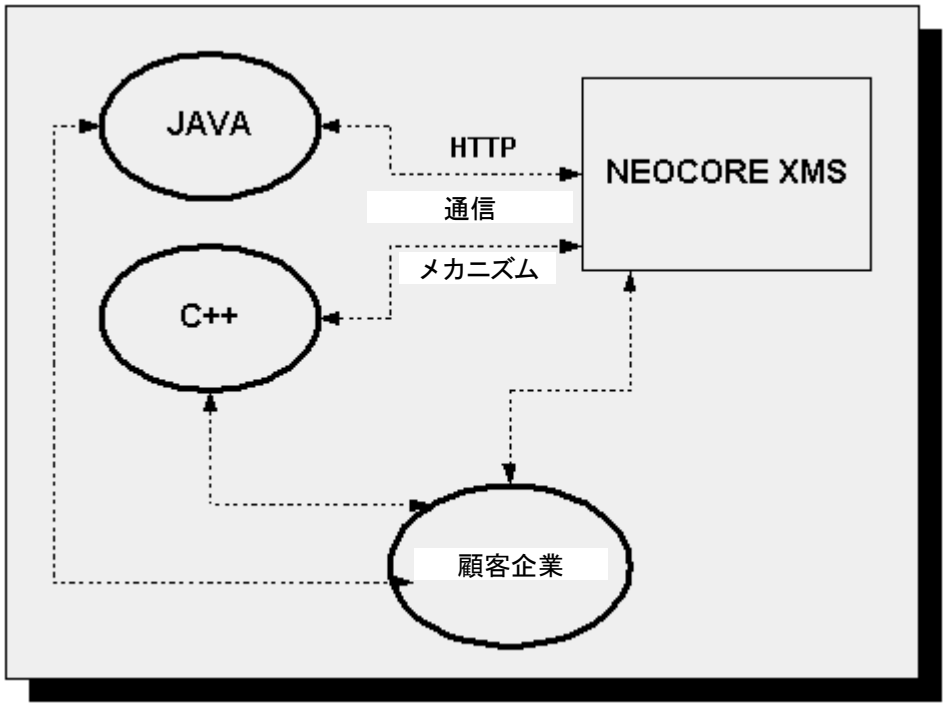
HTTP API

NeoServer は、マルチスレッドとトランザクションに対応しています。それぞれの要求は 1 つのスレッドに対応しており、複数の要求を 1 つのトランザクションにまとめることができます。

詳細については、『***System Administration Guide***』を参照してください。

HTTP API

NeoCore XMS と顧客企業との間の中心的な通信メカニズムは、HTTP です。Java と C++の API は､HTTP プロトコルをカプセル化しています。



HTTP **通信メカニズム**

注:Query、Insert、Modify、Delete、login、setPassword の各メソッドを実行するために NeoServer にアクセスするときには、GET または POST のいずれかを使用できます。GET は、処理が高速ですが、情報量が約 2K に制限されています。POST は、処理が遅いとはいえ、2K を超える情報を挿入するときには POST をお勧めします。Store メソッドは、常に POST を使用します。この章の末尾にある POST のサンプルを参照してください。

以下に示すのは、タスクごとにまとめた実際の HTTP API 情報です。まずは Web ブラウザの URL 行にそのまま入力してから、追加のアクション（「...を追加します」など）を実行してください。

メソッド: **login**

GET/POST

URL 構文:`http://servername:port/neoadmin?`

`cmd=GETSESSION&=&user= username&passwd=password`

追加のアクション:なし

説明:

servername：XMS のマシン名または IP アドレス

port：XMS が通信をリッスンしているポート

username：ユーザーのユーザー名

password：ユーザーのパスワード

戻り値: セッション ID（SID）またはエラーメッセージを含んだ XML

メソッド: **logout**

GET

URL 構文:`http://servername:port/neoadmin?`

`cmd=ENDSESSION`

追加のアクション:ユーザーの SID を使用して HTTP 要求に HTTP ヘッダー（sid）を追加します。例えば、httpconnection.setRequestProperty（"sid", userSid）;のようにします。

説明:

servername：XMS のマシン名または IP アドレス

port：XMS が通信をリッスンしているポート

戻り値: 成功メッセージまたはエラーメッセージの XML

メソッド：setTraceLevels

GET

URL 構文:`http://servername:port/neoadmin?`

`cmd=TRCLVL&RHTRCLVL=tracelvl`

追加のアクション:ユーザーの SID を使用して HTTP 要求に HTTP ヘッダー（sid）を追加します。例えば、httpconnection.setRequestProperty（"sid", userSid）;のようにします。

説明:

servername：XMS のマシン名または IP アドレス

port：XMS が通信をリッスンしているポート

tracelvl：新しいトレースレベル

戻り値: 成功メッセージまたはエラーメッセージの XML

メソッド: **getTraceLevels**

GET

URL 構文: `http://servername:port/neoadmin?`

`cmd=GETTRCLVL`

追加のアクション:ユーザーの SID を使用して HTTP 要求に HTTP ヘッダー（sid）を追加します。例えば、httpconnection.setRequestProperty（"sid", userSid）;のようにします。

説明:

servername：XMS のマシン名または IP アドレス

port：XMS が通信をリッスンしているポート

戻り値: 成功メッセージまたはエラーメッセージの XML

メソッド: activateAccessControl

GET

URL 構文:`http://servername:port/neoadmin?`

`cmd=ACTIVATEAC`

追加のアクション:ユーザーの SID を使用して HTTP 要求に HTTP ヘッダー（sid）を追加します。例えば、httpconnection.setRequestProperty（"sid", userSid）;のようにします。

説明:

servername：XMS のマシン名または IP アドレス

port：XMS が通信をリッスンしているポート

戻り値: 成功メッセージまたはエラーメッセージの XML

メソッド: **setPassword**

GET/POST - GET がサポートされるのは、NeoXDBRuntime.xml で AllowClearTextPasswords が TRUE に設定されている場合に限られます。それ以外の場合は、POST だけがサポートされます。

POST 形式のメッセージ

本体（HTTP Post）

ヘッダー

```
/n

/n

cmd=SETPASSWD=&user=username&passwd=password

passwd=password
```

注:上記のメッセージは、multi-part 形式ではありません。

URL 構文:`http://servername:port/neoadmin?`

```
cmd=SETPASSWD=&user=username&passwd=password
```

追加のアクション:ユーザーの SID を使用して HTTP 要求に HTTP ヘッダー（sid）を追加します。例えば、httpconnection.setRequestProperty("sid", userSid);のようにします。

説明:

servername：XMS のマシン名または IP アドレス

port：XMS が通信をリッスンしているポート

username：ユーザーのユーザー名

password：ユーザーの新しいパスワード

戻り値: 成功メッセージまたはエラーメッセージの XML

メソッド：setIsolationLevel

GET

構文:`http://servername:port/neoquery?`

`cmd= TRANSACTION_ISOLATION&input=level`

追加のアクション:ユーザーの SID を使用して HTTP 要求に HTTP ヘッダー（sid）を追加します。例えば、httpconnection.setRequestProperty（"sid", userSid）;のようにします。

説明:

servername：XMS のマシン名または IP アドレス

port：XMS が通信をリッスンしているポート

level：以下のいずれか
READ_COMMITTED
READ_UNCOMMITTED
REPEATABLE_READ

戻り値: 結果またはエラーメッセージの XML

メソッド: **startTransaction**

GET

URL 構文:`http://servername:port/neoquery?`

`cmd= TRANSACTION_START`

追加のアクション:ユーザーの SID を使用して HTTP 要求に HTTP ヘッダー（sid）を追加します。例えば、httpconnection.setRequestProperty（"sid", userSid）;のようにします。

説明:

servername：XMS のマシン名または IP アドレス

port：XMS が通信をリッスンしているポート

戻り値: 結果またはエラーメッセージの XML

メソッド: **startTransaction**

GET

URL 構文:`http://servername:port/neoquery?`

`cmd= TRANSACTION_START&input=flush`

追加のアクション:ユーザーの SID を使用して HTTP 要求に HTTP ヘッダー（sid）を追加します。例えば、httpconnection.setRequestProperty("sid", userSid);のようにします。

説明:

servername：XMS のマシン名または IP アドレス

port：XMS が通信をリッスンしているポート

flush：以下のいずれか

FLUSH：トランザクションはコミットの時点でディスクにフラッシュされます。

NOFLUSH：トランザクションはバックグラウンドプロセスとして 1 秒ごとにディスクにフラッシュされます。

戻り値: 結果またはエラーメッセージの XML

メソッド: **startTransaction**

GET

URL 構文:`http://servername:port/neoquery?`

```
cmd= TRANSACTION_START&input=FLUSH or NOFLUSH MAXDURATION=<integer>
INACTIVITYDURATION=<integer>
```

追加のアクション:ユーザーの SID を使用して HTTP 要求に HTTP ヘッダー（sid）を追加します。例えば、httpconnection.setRequestProperty（"sid", userSid）;のようにします。

説明:

servername：XMS のマシン名または IP アドレス

port：XMS が通信をリッスンしているポート

FLUSH：トランザクションはコミットの時点でディスクにフラッシュされます。

NOFLUSH：トランザクションはバックグラウンドプロセスとして 1 秒ごとにディスクにフラッシュされます。

MAXDURATION：この startTransaction から後続の commitTransaction までの許容時間を秒単位で指定します。MAXDURATION に指定した時間を超えると、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。無効な（0 以下の）値を指定した場合は、デフォルト値が使用されます。

INACTIVITYDURATUION:このトランザクションが非活動状態でいられる時間を秒単位で指定します。指定した時間を超えると、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。

戻り値: 結果またはエラーメッセージの XML

メソッド: **commitTransaction**

GET

URL 構文:`http://servername:port/neoquery?`

`cmd=TRANSACTION_COMMIT`

追加のアクション:ユーザーの SID を使用して HTTP 要求に HTTP ヘッダー（sid）を追加します。例えば、httpconnection.setRequestProperty("sid", userSid);のようにします。

説明:

servername：XMS のマシン名または IP アドレス

port：XMS が通信をリッスンしているポート

戻り値: 結果またはエラーメッセージの XML

メソッド: **rollbackTransaction**

GET

URL 構文:`http://servername:port/neoquery?`

`cmd=TRANSACTION_ROLLBACK`

追加のアクション:ユーザーの SID を使用して HTTP 要求に HTTP ヘッダー（sid）を追加します。例えば、httpconnection.setRequestProperty("sid", userSid);のようにします。

説明:

servername：XMS のマシン名または IP アドレス

port：XMS が通信をリッスンしているポート

戻り値: 結果またはエラーメッセージの XML

メソッド: **queryXML**

GET/POST

URL 構文: `http://servername:port/neoquery?`

`cmd=QUERY&input=xpathQuery`

追加のアクション:ユーザーの SID を使用して HTTP 要求に HTTP ヘッダー（sid）を追加します。例えば、httpconnection.setRequestProperty（"sid", userSid）;のようにします。

説明:

servername：XMS のマシン名または IP アドレス

port：XMS が通信をリッスンしているポート

xpathQuery：クエリー文字列（例えば、/ND/tag1/tag2）

戻り値: 結果またはエラーメッセージの XML

メソッド: **deleteXML**

GET/POST

URL 構文:`http://servername:port/neoquery?`

`cmd=DELETE&input=xpathQuery`

追加のアクション:ユーザーの SID を使用して HTTP 要求に HTTP ヘッダー（sid）を追加します。例えば、httpconnection.setRequestProperty（"sid", userSid）;のようにします。

説明:

servername：XMS のマシン名または IP アドレス

port：XMS が通信をリッスンしているポート

xpathQuery：クエリー文字列（例えば、/ND/tag1/tag2）

戻り値: 結果またはエラーメッセージの XML

メソッド: **insertXML**

GET/POST

URL 構文:`http://servername:port/neoquery?`

`cmd= INSERT&input=xpathQuery&data=newXML`

追加のアクション:ユーザーの SID を使用して HTTP 要求に HTTP ヘッダー（sid）を追加します。例えば、httpconnection.setRequestProperty（"sid", userSid）;のようにします。

説明:

servername：XMS のマシン名または IP アドレス

port：XMS が通信をリッスンしているポート

xpathQuery：クエリー文字列（例えば、/ND/tag1/tag2）

newXML：挿入する XML

戻り値: 結果またはエラーメッセージの XML

メソッド: **modifyXML**

GET/POST

URL 構文:`http://servername:port/neoquery?`

`cmd= MODIFY&input=xpathQuery&data=newXML`

追加のアクション:ユーザーの SID を使用して HTTP 要求に HTTP ヘッダー（sid）を追加します。例えば、httpconnection.setRequestProperty（"sid", userSid）;のようにします。

説明:

servername：XMS のマシン名または IP アドレス

port：XMS が通信をリッスンしているポート

xpathQuery：変更する XML を指定したクエリー文字列（例えば、/ND/tag1/tag2）

newXML：Xpath クエリーで指定した XML の代わりに使用する新しい XML

戻り値: 結果またはエラーメッセージの XML

メソッド: **copyXML**

GET/POST

URL 構文:`http://servername:port/neoquery?`

`cmd= COPY&input=xpathQuery`

追加のアクション:ユーザーの SID を使用して HTTP 要求に HTTP ヘッダー（sid）を追加します。例えば、httpconnection.setRequestProperty（"sid", userSid）;のようにします。

説明:

servername：XMS のマシン名または IP アドレス

port：XMS が通信をリッスンしているポート

xpathQuery：クエリー文字列（例えば、Xpath）

戻り値: 結果またはエラーメッセージの XML

メソッド: **queryFlatXML**

GET/POST

URL 構文:`http://servername:port/neoquery?`

`cmd= FLAT&input=xpathQuery`

任意選択のアクション:ユーザーの SID を使用して HTTP 要求に HTTP ヘッダー（sid）を追加します。例えば、httpconnection.setRequestProperty（"sid", userSid）;のようにします。

説明:

servername：XMS のマシン名または IP アドレス

port：XMS が通信をリッスンしているポート

xpathQuery：クエリー文字列（例えば、/ND/tag1/tag2）

戻り値: クエリー結果をフラット形式の行として組み込んだ XML

メソッド: **queryXMLUpdateIntent**

GET/POST

URL構文:`http://servername:port/neoquery?`

`cmd=QUERYUPDATE&input=xpathQuery`

任意選択のアクション:ユーザーの SID を使用して HTTP 要求に HTTP ヘッダー（sid）を追加します。例えば、httpconnection.setRequestProperty("sid", userSid);のようにします。

説明:

servername：XMS のマシン名または IP アドレス

port：XMS が通信をリッスンしているポート

xpathQuery：クエリー文字列（例えば、/ND/tag1/tag2）

戻り値: クエリー結果をフラット形式の行として組み込んだ XML

メソッド: **queryCountXML**

GET/POST

URL 構文:`http://servername:port/neoquery?`

`cmd=COUNT &input=xpathQuery`

任意選択のアクション:ユーザーの SID を使用して HTTP 要求に HTTP ヘッダー（sid）を追加します。例えば、httpconnection.setRequestProperty("sid", userSid);のようにします。

説明:

servername：XMS のマシン名または IP アドレス

port：XMS が通信をリッスンしているポート

xpathQuery：クエリー文字列（例えば、/ND/tag1/tag2）

戻り値: カウントを含んだ XML

メソッド: **queryTreeXML**

GET/POST

URL 構文:`http://servername:port/neoquery?`

`cmd= Tree&input=xpathQuery`

任意選択のアクション:ユーザーの SID を使用して HTTP 要求に HTTP ヘッダー（sid）を追加します。例えば、httpconnection.setRequestProperty("sid", userSid);のようにします。

注:この入力は/ND で始めなければならず、スラッシュで区切った一連の名前だけを含めます。例えば、次のようにします。

Tree /ND/tag1/tag2（正しく使用しないと、NeoMalformedXPathException が生成されます）

説明:

servername：XMS のマシン名または IP アドレス

port：XMS が通信をリッスンしているポート

xpathQuery：クエリー文字列（例えば、/ND/tag1/tag2）

戻り値: ターゲットクエリーよりも 1 レベル下のノードの名前を含んだ XML

メソッド: **queryDataContextXML**

GET/POST

URL 構文:`http://servername:port/neoquery?`

`cmd= Dataquery&input=DATA`

任意選択のアクション:ユーザーの SID を使用して HTTP 要求に HTTP ヘッダー（sid）を追加します。例えば、httpconnection.setRequestProperty("sid", userSid);のようにします。

注:このコマンドをサポートするには、データベースの完全インデックス構築が必要です。

説明:

servername：XMS のマシン名または IP アドレス

port：XMS が通信をリッスンしているポート

DATA: どのデータのコンテキストを抽出するかを指定します。二重引用符で囲む必要があります（例えば、"DataToLookFor"）。

戻り値: ターゲットクエリーデータを含むノードごとに、メタデータをフラット形式の行として組み込んだ XML を返します。

メソッド: **getServerStatistics**

GET

URL 構文:`http://servername:port/neoquery?`

`cmd= GETSTATS`

追加のアクション:ユーザーの SID を使用して HTTP 要求に HTTP ヘッダー（sid）を追加します。例えば、httpconnection.setRequestProperty("sid", userSid);のようにします。

説明:

servername：XMS のマシン名または IP アドレス

port：XMS が通信をリッスンしているポート

STAT_TYPE: GETSTATS | GETSTATS_ADMIN | GETSTATS_STORAGE | GETSTATS_ACCESS | GETSTATS_BUFFER | GETSTATS_TRANSACTION | GETSTATS_WINDOW

戻り値: 現在要求されている NeoCore XMS 統計タイプを含んだ XML

メソッド: **clearServerStatistics**

GET

URL 構文:`http://servername:port/neoquery?`

`cmd= CLEARSTATS`

追加のアクション:ユーザーの SID を使用して HTTP 要求に HTTP ヘッダー（sid）を追加します。例えば、httpconnection.setRequestProperty（"sid", userSid）;のようにします。

説明:

servername：XMS のマシン名または IP アドレス

port：XMS が通信をリッスンしているポート

戻り値: クリア後の現在の NeoCore XMS 統計を含んだ XML

メソッド: **getServerVersion**

GET

URL 構文:`http://servername:port/neoquery?`

`cmd= VERSION`

追加のアクション:ユーザーの SID を使用して HTTP 要求に HTTP ヘッダー（sid）を追加します。例えば、httpconnection.setRequestProperty（"sid", userSid）;のようにします。

説明:

servername：XMS のマシン名または IP アドレス

port：XMS が通信をリッスンしているポート

戻り値: 現在の NeoCore XMS バージョンを含んだ XML

メソッド: **storeXML**

POST

URL 構文： `http://servername:port/neoquery?`

`cmd=STORE`

追加のアクション:ユーザーの SID を使用して HTTP 要求に HTTP post ヘッダー（sid）を追加します。例えば、httpconnection.setRequestProperty（"sid", userSid);のようにします。

注: 以下のパラメータは、書いてあるとおりの順番（つまり、schemafile、prefixfile、xmlsourcefile という順番）で指定する必要があります。そうでない場合、格納は中断します。

注意: XML の格納が失敗する原因として 1 つ考えられるのは、NeoServer への格納時にヘッダーの形式が間違っていることです。その場合、XML が実際に送信される代わりに例外が生成される可能性があります。例えば、エンコードを UTF-8 に設定しているのに、XML が UTF-8 でない場合などがあります。

MIME multipart の数は最大 3 つです。

1. schemafile: スキーマファイルへの参照（省略可能）
2. prefixfile：プレフィックスファイルの内容（省略可能）
3. xmlsourcefile：格納する XML（必須）

Store **コマンドの** Post **サンプル**:

```
POST /neocore/neoquery?reqType=STORE HTTP/1.1

Content-Type:multipart/form-data;
boundary=---------------------------e23ec1105c

Host:lwong:8080

sid: 12345698

Connection:Keep-Alive

Content-Length: 974

¥r¥n¥r¥n
```

```
------------------------------e23ec1105c

Content-Disposition:form-data;
name="schemafile";filename="http://mycompany.com/phoneschema.xml"



------------------------------e23ec1105c

Content-Disposition:form-data; name="prefixfile"; filename=""

Content-Type:text/xml



------------------------------e23ec1105c

Content-Disposition:form-data; name="xmlsourcefile";
filename="d:¥xmlfiles¥phonebook.xml"

Content-Type:text/xml


<PhoneListings>

    <Listing>

        <Name>

            <Last>Aab</Last>

            <First>Howard</First>

        </Name>

        <Address>

            <Number>818</Number>

            <Street>Hoorne Ave</Street>

            <City>Colorado Springs</City>

            <State>CO</State>

            <ZIPcode>80907</ZIPcode>

        </Address>

        <PhoneNumber>

            <AreaCode>719</AreaCode>

            <Number>599-3403</Number>

        </PhoneNumber>
```

```
    </Listing>
</PhoneListings>
----------------------------e23ec1105c--
```

Store 以外のすべてのコマンドの Post サンプル（HTTP コマンド全体の長さが 2K を超える場合に有効化される）：

```
POST http://localhost:7700/neoquery?cmd=INSERT
Content-Type:multipart/form-data; boundary=--------------------------3d52e126
Sid: 1869905129
¥r¥n¥r¥n
----------------------------3d52e126
Content-Disposition:form-data; name="input"
/ND/CATALOG/CD/PRICE
----------------------------3d52e126
Content-Disposition:form-data; name="data"
<CATALOG>
   <CD>
       <TITLE>Empire Burlesque</TITLE>
       <ARTIST>Bob Dylan</ARTIST>
       <COUNTRY>USA</COUNTRY>
       <COMPANY>Columbia</COMPANY>
       <PRICE>10.90</PRICE>
       <YEAR>1985</YEAR>
```

```
</CD>
      .
      .
      .
<CD>
    <TITLE>Red</TITLE>
    <ARTIST>The Communards</ARTIST>
    <COUNTRY>UK</COUNTRY>
    <COMPANY>London</COMPANY>
    <PRICE>7.80</PRICE>
    <YEAR>1987</YEAR>
</CD>
<CD>
    <TITLE>Unchain my heart</TITLE>
    <ARTIST>Joe Cocker</ARTIST>
    <COUNTRY>USA</COUNTRY>
    <COMPANY>EMI</COMPANY>
    <PRICE>8.20</PRICE>
    <YEAR>1987</YEAR>
</CD>
</CATALOG>
------------------------------3d52e126--
```

NeoServer 側の XSLT 拡張機能

説明

XSLT（Extensible Stylesheet Language for Transformations）は、XML（Extensible Markup Language）文書の構造を別の形式（テキスト、HTML、XML など）に変換（変更）する方法を記述するための標準的な手段です。

XML 文書のソースツリーまたはデータ構造を新しい XML 文書の結果ツリーに変換する方法を記述した文書のことを XSLT 変換文書といいます。この変換によって、まったく異なる構造を生成できます。

Xpriori, LLC では、XMS 内に格納されている文書に対して XSLT を実行するための NeoServer 側の拡張機能を用意しています。NeoServer は、このサービスに対する拡張 HTTP 要求の受け渡しを行います。NeoServer 側の拡張機能には、HTTP 要求オブジェクトに由来するパラメータのほかに、クエリー用と XSL 用のパラメータがあります。この拡張機能は、コンテキストから SID（セッション ID）を取得します。

このサービスは、XQuery を使用して XMS から文書を取得します。さらに、Xalan を使用して XSLT を実行します。

各 API には、NeoServer 側の XSLT 拡張機能を起動する方法を記述した新しいメソッド呼び出しがあります。そのようにしてスタイルシート言語による変換を利用すれば、NeoServer から変換済みの文書を取り出すことができます。

XMS 内部での処理の流れ

1. 再利用と大規模展開が可能な NeoServer 側の拡張機能を作成します。

2. NeoServer が XSLT を実行します。

3. 拡張機能が、XMS から抽出する XML（つまり変換対象の XML）を指定した XQuery を受け入れます。

4. 拡張機能が、以下のいずれかを記述した文字列を受け入れます。

- スタイルシートを格納した.xsl ファイル

または

- スタイルシートの URL

または

- 直接入力したスタイルシートコマンド

または

- XMS に格納されているスタイルシートを指定した XQuery

5. API が更新され、XSLT サービスへのアクセスが組み込まれます。

6.拡張機能が Xalan（フリーウェア製品）と連携して XSLT を実行します。

メソッド: NeoTransform.transformXML

URL:http://servername:port/sse/xslt?cmd=xquery&input=xsl

追加のアクション:ユーザーの SID を使用して HTTP 要求に HTTP ヘッダー（sid）を追加します。例えば、httpconnection.setRequestProperty（"sid", userSid）;のようにします。

説明:

xquery には、XMS から取り出す XML 文書（つまり変換対象の文書）を指定します。

xsl には、変換に使用する XSLT 文書を指定します。xsl の文字列に付ける接頭辞によって、その xsl がファイルなのか、URL なのか、XMS データベース内のスタイルシートなのか、直接入力したスタイルシートコマンドなのかを示します。xsl の形式は、以下のいずれかになります。

file:filename.xsl - NeoServer 上の XMS インストールディレクトリの下の xms/sse/xsl ディレクトリにあるファイルの名前

- スタイルシートの格納先の URL

/ND/... - XMS 内の XSL 文書を一意に識別するための XQuery

<stylesheet ... - 直接入力したスタイルシートコマンド

戻り値: XML または例外

例

例1:

```
http://servername:port/sse/xslt?cmd=/ND/DocumentA&input=file:formatAtoB.xsl
```

例 1 は、/ND/DocumentA という XQuery で指定されている XML を抽出し、NeoServer 上の xms/sse/xsl という NeoCore ディレクトリにある formatAtoB.xsl ファイルに格納されているスタイルシートを使用してその XML を変換します。その結果は、ユーザーに返されます。xms/sse/xsl ディレクトリ内のスタイルシートは、管理者がインストールしておく必要があります。

例 2:

```
http://servername:port/sse/xslt?cmd=/ND/DocumentA&input=/ND/stylesheet
```

例 2 は、/ND/DocumentA という XQuery で指定されている XML を抽出し、XMS 内に XML 文書として格納されているスタイルシートを使用してその XML を変換します。その結果は、ユーザーに返されます。XMS 内に複数のスタイルシートが格納されている場合は、XQuery で対象のスタイルシート文書を正確に指定する必要があります。

例えば、以下のように指定します。

例3: NeoCoreのMetaDataを使用して、スタイルシート文書を一意に指定します。

```
http://servername:port/sse/xslt?cmd=/ND/DocumentA&input=/ND[MetaData/SourceFil
e="myStylesheet.xsl"]/stylesheet
```

または

```
http://servername:port/sse/xslt?cmd=/ND/DocumentA&input=/ND[MetaData/DocID="20
"]/stylesheet
```

エンコードの動作

ロケール設定

ロケール設定によって、XMS で生成されるメッセージのテキストに使用する言語を指定します。どんな要求に関しても有効なロケールが 2 つあります。

1. クライアントのロケール。無効な要求（クエリーの構文エラーや整形式でない XML 文書など）から生成されるエラーメッセージが対象になります。
2. NeoServer のデフォルトロケール（NeoServer.xml の DefaultLocale）。NeoServer ログメッセージが対象になります。また、ロケールを明示的に設定していないクライアントのロケールにもなります。

ロケールは、2 文字の ISO 言語省略表記で指定します。例えば、英語は「en」、日本語は「ja」です。また、より具体的な指定をするために 2 文字の国別コードを含めることもできます。例えば、アメリカ英語の場合は「en_US」と指定します。

このエンコードは、クライアントについても NeoServer についても設定できます。クライアントがエンコードを要求しない場合は、NeoServer のデフォルト（NeoServer.xml の DefaultEncoding）が使用されます。このエンコードによって、クライアントに対する NeoServer 応答（クエリーで抽出する XML 文書など）やクライアントに返されるエラーメッセージの文字セットと形式が決まります。

有効なエンコード

有効なエンコードとしては、Unicode 文字セットの「UTF-8」や「UTF-16」、日本語文字セットの「Shift_JIS」や「EUC-JP」や「JIS」や「ISO-2022-JP」などがあります。このうち、UTF-16 を除くすべてのエンコードには、標準の 8 ビット ASCII 文字がサブセットとして含まれています。

ロケールまたはエンコードの設定

C++または Java の API を使用するクライアントは、以下のいずれかの方法でロケールまたはエンコードを設定できます。

1. NeoConnection クラスまたは SessionManagedNeoConnection クラスのコンストラクタの 4 引数形式を使用する方法。
2. 既存の接続オブジェクトに対して setLocale メソッドまたは setEncoding メソッドを使用する方法。Java の場合は、java.lang.Locale オブジェクトを作成するコンストラクタの引数としてロケール設定（「en」など）を指定します。この引数は接続クラスのコンストラクタに渡されます。

優先順位

NeoServer に対する 1 つの要求で使用されるロケールとエンコードには、以下の優先順位があります。

1. 要求が格納の場合、格納対象の文書の XML 宣言にはエンコードの指定が含まれており、その文書ではそのエンコードが使用されます。例えば、格納対象の文書の第 1 行が<?xml version="1.0" encoding="UTF-8" ?>となっていれば、エンコードは UTF-8 であると解釈されます。この XML 宣言はロケール設定には影響しませんし、クライアントに対する NeoServer 応答のエンコードにも影響しません。
2. クライアントが特定のロケールやエンコードを要求した場合（例えば、C++や Java の接続オブジェクトでロケールやエンコードを指定した場合）、応答ではその設定が使用されます。
3. それ以外の場合、応答では言語とエンコードに関する NeoServer のデフォルト設定が使用されます。

注: 特殊な状況が 1 つあります。つまり、クライアントがロケールやエンコードを指定せず、サーバーが使用不能になっているという状況です。この場合、API 応答は UTF-8 エンコードを使用した英語で表示されます。

用語集

ACID プロパティ:原子性、一貫性、独立性、耐久性。

API: アプリケーションプログラミングインターフェース。アプリケーションが NeoServer にアクセスするために使用します。

関連付け: 関連付けは、コアインデックスの検索によって返される値です。NeoServer では、関連付けの値は、マップオフセットの一覧へのポインタとして解釈されます。またはそれ自体がマップオフセットとなることも可能です。

領域自動拡張: データベースの稼働中に、要求を処理しながら、NeoServer ファイルのサイズを拡張する機能。

ブラウザ cookie: Netscape と Internet Explorer の 1 機能。細かな情報を記述した小さなファイルです。JavaScript を使用してデータを長期間格納するための安全簡便な方法であり、おそらく唯一の現実的な方法とも言えます。

バッファ: データベースの初期化時に、メモリーの一部が選択され、さまざまな長さのチャンク群に分割されます。メモリーのこの部分のことをバッファプールといいます。各プールは同じサイズのチャンクに対応します。例えば、サイズが 1K のチャンクが集まったプールが 1 つ、2K のチャンクが集まったプールが 1 つ、といった具合になります。1 つのチャンクがバッファであり、処理の必要に応じてスレッドを実行することによって、バッファを割り当てます。バッファが自身のプールに戻されることを確認するのはスレッドの役目です。

CDATA: マークアップなど、XML パーサーによって無視される文字データや、インデックスベースの検索を必要としない文字データを埋め込む方法。

チェックポイント: データベース内の復元ポイントを取り込むログエントリ。チェックポイントでは、すべての未解決トランザクションのトランザクション ID がトランザクションログに記録されます。データファイルはすべてディスクにフラッシュされます。起動時には、最後のチェックポイントに含まれる情報に基づいてログが適用されます。システムは、実行時構成ファイルに定義されている間隔でチェックポイントを設定します。

ConfigMigration ツール: 旧バージョンから構成ファイルを移行します。

コアインデックス: コアインデックスは、NeoData の構造に基づいています。各コアインデックスは、固有の文字列を表します。XMS のコアインデックスには、タグとフラット形式のタグ、データのみ、タグ+データの組み合わせという 3 種類があります。

count クエリー: データを検索して、ターゲットに一致するノードの数を検出し、ターゲットセット内のノードの数を返します。

countnotacid クエリー: ターゲットセット内のノードの数を返します。countnotacid は、クエリー内で count コマンドの代わりに使用できます。

カプレット階層ベクトル: カプレット階層ベクトル（CHV）とは、特に挿入操作後の収束操作で階層を適切に処理するための構造体です。より正式な言い方をすれば、ノードの祖先チェーン内の（そのノードまでの）オフセットの順序付きセットということになります。ただし、CHV では、親同士の間で文書の順序を保持しますが、同じ親の内部では必ずしも文書の順序を保持するわけではありません。

カプレット: 要素のフラット形式のタグとその要素のデータ値の組み合わせです。

クロスリファレンスファイル: クロスリファレンスファイルは、データディクショナリに対するポインタのファイルベースの配列です。マップエントリがデータポインタを含む場合、そのポインタはクロスリファレンスファイルのオフセットをポイントし、そのオフセットはデータディクショナリをポイントします。2.8 よりも前のリリースでは、マップエントリはデータディクショナリを直接ポイントしていました。クロスリファレンスファイルは、データディクショナリの回復を目的として導入されました。クロスリファレンスを使用すると、データディクショナリで変更しなければならないポインタは 1 つだけで済みます。

データディクショナリ: データモデル内のデータオブジェクトまたは項目の記述の集まり。これらを参照するプログラマや他のユーザーにとって便利な機能です。

dataquery コマンド: コンテキストのデータを抽出できるだけでなく、XPath のワイルドカード検索の場合よりも詳細な情報を取得できます。

デッドロック: 同時に実行している複数のトランザクションがリソースを保持して、どのトランザクションからもコミットができない状態。例えば、トランザクション A がリソース 1 を保持し、トランザクション B がリソース 2 を保持している状態で、トランザクション A が処理の完了のためにリソース 2 を必要とし、トランザクション B が処理の完了のためにリソース 1 を必要としている場合、この 2 つのトランザクションはデッドロックの状態になっています。この場合は、終了するトランザクションを 1 つ選んでいったんロールバックし、そのリソースを解放して、他のトランザクションがコミットできるようにします。

削除: 削除操作では、データベース内の要素または属性に対する論理削除を実行します。削除の結果として、余分のスペースが使用可能になるわけではありません。挿入と同様に、削除は一連のターゲットノード上で操作されます。そのため、ターゲットセットのカーディナリティに基づいて、同じ削除がデータベースで複数回実行されることもあります。

ディクショナリファイル: 格納されている XML 文書に現れる固有の文字列を内容とするファイル。XMS には、固有のフラット形式のタグを含むディクショナリファイルと、固有のデータ項目を含むディクショナリファイルがあります。

DPP: Digital Pattern Processing（エンジン）。

DTArray: DupTree Array（重複ツリー配列）。SubTree は、DTArray で構成されます。DTArray は固定サイズで、SubTree 構築時に割り当てられる最小メモリーブロックを表します。DTArray の内容は、他の DTArray への関連付けまたは参照インデックスのいずれかになる要素です。DTArray 内の要素の数は、16、32、64、128、256 のいずれかです。要素の数は、データベース構成変数です。各種インデックス（タグ、データ、タグ+データ）では、さまざまなサイズの DTArray を使用できます。1 つのインデックスタイプ内の DTArray はすべて同じサイズになります。

DTMem: Duplicate Tree Memory（重複ツリーメモリー）。これはサブツリー配列の割り当て元になるメモリー領域です。

重複インデックス: 重複インデックスは、固有の文字列が出現するたびにエントリを格納します。エントリは、その文字列を含むマップオフセットまたはその文字列を参照するマップオフセットに相当します。

flat クエリー: データを検索して、ターゲットに一致するノードを検出し、ターゲットノードの各出現部分に関するフラット形式の行を返します。

フラット形式の行: 行の構造情報を保持する関連数値情報を含んだカプレット。フラット形式の行から文書を再作成できます。フラット形式の行は、要素または属性にデータ値がある場合や、要素または属性が空のリーフノードである場合にのみ生成されます。

フラット形式のタグ: 要素のフラット形式のタグとは、その要素の祖先のタグをすべて含んだ文字列です。例えば、<A> <B> <C> foo </C> </B> </A>という文書があるとすれば、要素 C のフラット形式のタグは、A>B>C>という文字列になります。

FLWR: FOR-LET-WHERE-RETURN XQuery キーワードの略語。FLWR 式は、SQL の SELECT columnlist FROM tablename WHERE criteria ステートメントに相当します。

関数呼び出し: QName の後にゼロ個以上の式を括弧に入れたリストを記述したもの。

関数変換ルール: 引数値または戻り値を必要なデータ型（つまり、関数のパラメータ値または戻り値のために宣言したデータ型）に変換するためのルール。

HTTP: Hypertext Transfer Protocol。

アイコン: コア DPP 処理の基本単位。任意のデータ項目をマッピングできる固定フィールド数量です。アイコンジェネレータ（基本的には、特定の関連する多項式を持つ線形シフトレジスタ）によって生成されます。

挿入: データベースからターゲット要素のセットを選択し、指定の XML フラグメントをセット内の各ノードの後に挿入します。挿入は兄弟レベルでのみ行われます。つまり、挿入される各フラグメントはターゲットセット内のノードの兄弟になります。ターゲットセットのカーディナリティに基づいて、1 度の挿入操作で、同じフラグメントをデータベースに複数回挿入できます。

Internet Explorer: Microsoft のインターネットブラウザ。

IP アドレス: インターネット上でパケット単位でやり取りする情報の送信側と受信側を識別する 32 ビットの数値。

JavaScript: Netscape のインタープリタ型のプログラミング言語またはスクリプト言語。

JNI: Java Native Interface。JNI を使用すれば、Java 仮想マシン（VM）内で実行する Java コードを、他の言語（C、C++、アセンブリなど）で記述したアプリケーションやライブラリで操作できます。さらに、呼び出し API を使用すれば、Java 仮想マシンをネイティブアプリケーションの中に埋め込むこともできます。

JRE: Java Runtime Environment。Java Development Kit（JDK）に組み込まれているファイルのサブセットです。コンパイラやデバッガや関連ツールは含まれません。むしろ、Java テクノロジに対応したアプリケーションのための最小限のランタイム（つまり、JVM、JFC、JIT、サポートファイル）を提供しています。Java Development Kit とは異なり、ライセンス契約の条件に基づいて無償で配布されることもあります。

JVM: Java 仮想マシンとは、コンパイル済みの Java コードを実行するための「CPU」のソフトウェア実装です。その種のコードとしては、スタンドアロンの Java アプリケーションや、NetScape Navigator などの Web ブラウザでダウンロードして実行する「アプレット」があります。

リテラル: アトミック値の直接的な構文表現。XQuery では、文字列リテラルと数値リテラルという 2 種類のリテラルをサポートしています。

ロック: ロックとは、リソースを予約することです。ロックの種類には、共有ロック、更新ロック、書き込みロックがあります。共有ロックは、読み取りロックともいい、同じリソースに対して別の共有ロックを設定できます。更新ロックは更新目的での読み取りを意味します。更新ロックを保持するリソースに対して、共有ロックを設定できますが、更新ロックや書き込みロックは設定できません。書き込みロックは、排他ロックともいい、その同じリソースに対してその他のロックを許可しません。

マップエントリ: マップエントリとは、マップファイルにあるフラット形式の行の格納済みのイメージです。マップエントリは固定長です。マップエントリには、行のタグ部分を指すポインタ、文書全体のコンテキストにおける行の構造を説明する一連の数値、および、次のいずれかが含まれます。行のデータ部分のクロスリファレンスエントリを指すポインタ、または、行のデータ部分の文字列値（長さが 6 バイト以下の場合）、または、行の数値の BCD 表現（長さが 12 バイト以下の数値の場合）。

マップファイル: 固定長のエントリを含むファイル。各エントリは、XML のフラット形式の行を表します。

メタデータ: 別のファイルまたはデータベースの属性、構造、処理、変更に関する情報を格納するファイルまたはデータベースのことをいいます。データの内容や品質など、データのあらゆる特性を記述できます。

変更: 変更操作では、要素や属性に関連付けられたデータ値を変更します。矛盾が生じないように変更するには、要素にデータ値が関連付けられている必要があります。テキストデータを持たないノードを変更すると、動作しない場合があります。挿入と同様に、変更は一連のターゲットノード上で操作されます。そのため、ターゲットセットのカーディナリティに基づいて、同じ変更がデータベースで複数回実行されることもあります。

MTArray: Multi-Tree Array（マルチツリー配列）。MTBase オブジェクトは、MTArray で構成されます。MTArray は固定サイズで、MTBase 構築時に割り当てられる最小メモリーブロックを表します。MTArray は、4 つまたは 8 つの MapBlock で構成されます。この MapBlock 数は、データベース構成変数です。インデックスタイプ（タグ、データ、タグ+データ）ごとに、異なったサイズ（MapBlock が 4 つまたは 8 つ）の MTArray を使用できます。1 つのインデックスタイプの中では、すべての MTArray が同じサイズになります。

MTBase: Multi-Tree Base（マルチツリーベース）オブジェクト。ベースツリーオブジェクトは、コアインデックスから生成されます。MTBase オブジェクトは、MTArray で構成され、レベル 1 またはレベル 2 のツリーとして存在します。各リーフ MTArray は、MapBlock で構成されます。各 MapBlock は、SubTree への関連付けまたは参照のいずれかを格納し、特定の MapFile を参照します。

MTMem: Multi-Tree Memory（マルチツリーメモリー）。これは、マルチツリーベース配列の割り当て元になるメモリー領域です。

相互排除: 厳格な用法ではありませんが、データ構造や構造体の一部に対する非トランザクションロックのことをいいます。このロックは、その構造体の更新が完了するとすぐに解除されます。また、重要なセクションに対するロックを指す場合もあります。

Netscape 7.0: 最も普及している 2 つの Web ブラウザのうちの 1 つ。Netscape は、現在は America Online（AOL）の一部になっています。

NIC: ネットワークインターフェースカード。

オフセット: オフセットという用語は、クロスリファレンスファイル内の固定長エントリの配列インデックスを指すこともありますが、さらに一般的なのは、固定長のマップエントリの配列インデックスを指す用法です。

path 式: ステップ、軸、ノードテスト、修飾子を含んだ式。

述部: ターゲットセットを限定する（絞り込む）ためのブール値のステートメント。クエリーインターフェースの強力な機能です。

プレフィックスファイル: プレフィックスファイルは、XML 文書と共に格納するように指定できます。プレフィックスファイルは、XML 文書に追加されます。

プライマリ式: 言語の基本プリミティブ。リテラル、変数、関数呼び出しを含み、演算子の優先順位を制御するために括弧を使用できます。

カンタム: コアインデックスの基本単位。一連の確認ビット、一連のフラグ、1 つの関連付けを含む、固定長フィールドです。

RDBMS: リレーショナルデータベースの作成、更新、管理を行うためのプログラム。RDBMS は、ユーザーが入力した（またはアプリケーションプログラムに含まれている）Structured Query Language（SQL）ステートメントを取り込み、データベースの作成や更新を実行し、データベースへのアクセスを提供します。

シーケンス型マッチング: 特定の値のデータ型が必要なデータ型と一致しているかどうかを判別する機能。一致していない場合は、関数呼び出しからエラー値が返されます。

セッション: NeoServer の起動時にプール内で作成されるオブジェクト。ユーザーがログインするか、anonymous 要求を行うと、セッションがプールから割り当てられます。前者の場合、ユーザーがログアウトするか、タイムアウトになるまでセッションは継続します。後者の場合、セッションは、要求の存続期間が終わるか、セッションやトランザクションがタイムアウトになるまで継続します。

ソート式: シーケンス内の項目の順序を制御するための手段。

SQL: データベースから情報を抽出したりデータベースを更新したりするための標準的な対話式プログラミング言語。

ステップ: ノードのシーケンスを文書の順序で（重複なしで）返す式。

格納: NeoCore XMS に新しい XML 文書を生成します。文書を格納すると、格納タイムスタンプ、固有の文書 ID、変更タイムスタンプ（格納タイムスタンプと同じ内容）などのメタデータがその文書に関連付けられます。また任意で、ソースファイル名、スキーマ参照、プレフィックスファイル名もメタデータに格納できます。

TCP: Transmission Control Protocol（伝送制御プロトコル）。インターネット上のコンピュータ間でメッセージ単位の形式でデータをやり取りするために、インターネットプロトコル（IP）と一緒に使用する一連のルール（プロトコル）。IP はデータの実際の送信を処理するのに対し、TCP は個々のデータ単位（パケット）を追跡管理します（メッセージをパケット単位に分割するのは、インターネット上でのルーティングを効率的に処理するためです）。

スレッドコンテキスト: 実行スレッドが処理を実行するために取得しなければならないシステム内のオブジェクト（かなり大きなオブジェクト）。システム内のスレッドコンテキストの数を制限することによって、システム内の同時実行コマンドの数を制限でき、結果としてシステム上のロードを調節できます。スレッドコンテキストには、そのスレッドで使用可能なファイルウィンドウへの参照や他の重要な実行データも含まれています。

トランザクション: データベースの作業単位。1 つのトランザクションに含まれるすべてのデータベース要求は、すべて成功するか、すべて失敗するかのどちらかになります。XMS には、明示的トランザクションと暗黙のトランザクションの 2 種類があります。明示的トランザクションは、トランザクションの開始と終了をユーザーが明示的に行うものです。暗黙のトランザクションは、明示的な開始に関連付けられていないコマンドに割り当てられます。したがって、暗黙のトランザクションでは、1 つのデータベースコマンドが 1 つのトランザクションに対応しています。

Tree コマンド: アプリケーション内から所定のノードの構造を検出できます。このコマンドは、アプリケーション側が、渡された XML の内容を知らない場合に使用できます。

Tree クエリー: データを検索して、ターゲットよりも 1 レベル下のノードを検出します。

チューニング: パフォーマンスを改善するために構成設定を最適化することです。

変数: 変数は、評価コンテキストで NCName のバインド先の値に評価されます。

ウィンドウ: メモリーマップファイル上のビュー。ウィンドウでは、ファイル全体をメモリーにマッピングできます。ファイルが大きすぎて入りきらない場合は、ファイルの一部をメモリーにマッピングできます。後者の場合、現在マッピングされているファイルを超えた部分にアクセスすると、参照先の新しい領域を含むファイルの部分をマッピングするために、ウィンドウが「スライド」されます。

XML: Extensible Markup Language。

XML 名前空間: 要素群の固有の範囲 ID としての役割を果たす URI によって定義します。

XMS: NeoCore™ XML Information Management System。

XMS 挿入: 1 つ以上の文書に構造体を追加するために使用します。SQL データベースに列を追加するのと同様です。

Xpath: XML 文書内の項目を検出して処理するための方法を記述する言語。Xpath では、文書の論理構造や階層構造のパスに基づくアドレス指定構文を使用します。

XSLT: Extensible Stylesheet Language for Transformations。

索引

R

S

T

U

X

あ

い

え

お

か

き

く

こ

さ

し

れ

ろ