HITACHI
Inspire the Next

Cosminexus アプリケーションサーバ V8

Cosminexus XML Processor
ユーザーズガイド

解説・手引書

3020-3-U27-40

■対象製品

●**適用** OS：Windows Server 2003，Windows Server 2003 R2，Windows Server 2003（x64）※1，Windows Server 2003 R2（x64）※1，Windows Server 2008 x86，Windows Server 2008 x64 ※1，Windows Server 2008 R2 ※1

P-2443-7B84 uCosminexus Application Server Standard-R 08-70

P-2443-7D84 uCosminexus Application Server Standard 08-70

P-2443-7K84 uCosminexus Application Server Enterprise 08-70

P-2443-7S84 uCosminexus Service Platform 08-70 ※2

●**適用** OS：Windows Server 2003，Windows Server 2003 R2，Windows Vista，Windows XP，Windows 7（32bit），Windows 7（x64）※1

P-2443-7E84 uCosminexus Developer Standard 08-70

P-2443-7F84 uCosminexus Developer Professional 08-70

P-2443-7T84 uCosminexus Service Architect 08-70 ※2

●**適用** OS：Windows Server 2003，Windows Server 2003 R2，Windows Server 2003（x64）※1，Windows Server 2003 R2（x64）※1，Windows Server 2008 x86，Windows Server 2008 x64 ※1，Windows Server 2008 R2 ※1，Windows Vista，Windows XP，Windows 7（32bit），Windows 7（x64）※1

P-2443-7H84 uCosminexus Client 08-70

●**適用** OS：Windows Server 2003（x64），Windows Server 2003 R2（x64），Windows Server 2008 x64，Windows Server 2008 R2

P-2943-7B84 uCosminexus Application Server Standard-R 08-70

P-2943-7D84 uCosminexus Application Server Standard 08-70

P-2943-7K84 uCosminexus Application Server Enterprise 08-70

P-2943-7S84 uCosminexus Service Platform 08-70 ※2

●**適用** OS：AIX 5L V5.3，AIX V6.1，AIX V7.1

P-1M43-7D81 uCosminexus Application Server Standard 08-70 ※2

P-1M43-7K81 uCosminexus Application Server Enterprise 08-70 ※2

P-1M43-7S81 uCosminexus Service Platform 08-70 ※2

●**適用** OS：HP-UX 11i V2（IPF），HP-UX 11i V3（IPF）

P-1J43-7D81 uCosminexus Application Server Standard 08-70

P-1J43-7K81 uCosminexus Application Server Enterprise 08-70

P-1J43-7S81 uCosminexus Service Platform 08-70 ※2

●**適用** OS：Red Hat Enterprise Linux AS 4（x86），Red Hat Enterprise Linux ES 4（x86），Red Hat Enterprise Linux AS 4（AMD64 & Intel EM64T），Red Hat Enterprise Linux ES 4（AMD64 & Intel EM64T），Red Hat Enterprise Linux 5 Advanced Platform（x86），Red Hat Enterprise Linux 5（x86），Red Hat Enterprise Linux 5 Advanced Platform（AMD/Intel 64），Red Hat Enterprise Linux 5（AMD/Intel 64），Red Hat Enterprise Linux Server 6（32-bit x86），Red Hat Enterprise Linux Server 6（64-bit x86_64）

P-9S43-7B81 uCosminexus Application Server Standard-R 08-70 ※

P-9S43-7D81 uCosminexus Application Server Standard 08-70 ※2

P-9S43-7K81 uCosminexus Application Server Enterprise 08-70 ※2

P-9S43-7S81 uCosminexus Service Platform 08-70 ※2

注※1　WOW64（Windows On Windows 64）環境だけで使用できます。

注※2　この製品については，サポート時期をご確認ください。

これらのプログラムプロダクトのほかにもこのマニュアルをご利用になれる場合があります。詳細は「リリースノート」でご確認ください。
本製品では日立トレース共通ライブラリをインストールします。

■輸出時の注意

本製品を輸出される場合には，外国為替および外国貿易法ならびに米国の輸出管理関連法規などの規制をご確認の上，必要な手続きをお取りください。
なお，ご不明な場合は，弊社担当営業にお問い合わせください。

■商標類

AIX は，米国およびその他の国における International Business Machines Corporation の商標です。
AIX 5L は，米国およびその他の国における International Business Machines Corporation の商標です。
AMD は，Advanced Micro Devices, Inc. の商標です。
HP-UX は，Hewlett-Packard Company のオペレーティングシステムの名称です。
Itanium は，アメリカ合衆国およびその他の国における Intel Corporation の商標です。
J2EE は，Oracle Corporation 及びその子会社，関連会社の米国 及びその他の国における登録商標または商標です。
Java は，Oracle Corporation 及びその子会社，関連会社の米国 及びその他の国における登録商標または商標です。
JSP は，Oracle Corporation 及びその子会社，関連会社の米国 及びその他の国における登録商標または商標です。
Linux は，Linus Torvalds 氏の日本およびその他の国における登録商標または商標です。
Microsoft は，米国 Microsoft Corporation の米国およびその他の国における登録商標または商標です。
Red Hat は，米国およびその他の国で Red Hat, Inc. の登録商標もしくは商標です。
SOAP（Simple Object Access Protocol）は，分散ネットワーク環境において XML ベースの情報を交換するための通信プロトコルの名称です。
Solaris は，Oracle Corporation 及びその子会社，関連会社の米国 及びその他の国における登録商標または商標です。
すべての SPARC 商標は，米国 SPARC International, Inc. のライセンスを受けて使用している同社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。SPARC 商標がついた製品は，米国 Sun Microsystems, Inc. が開発したアーキテクチャに基づくものです。
Sun は，Oracle Corporation 及びその子会社，関連会社の米国 及びその他の国における登録商標または商標です。
Sun Microsystems は，Oracle Corporation 及びその子会社，関連会社の米国 及びその他の国における登録商標または商標です。
UNIX は，The Open Group の米国ならびに他の国における登録商標です。
Windows は，米国 Microsoft Corporation の米国およびその他の国における登録商標または商標です。
Windows Server は，米国 Microsoft Corporation の米国およびその他の国における登録商標または商標です。
Windows Vista は，米国 Microsoft Corporation の米国およびその他の国における登録商標または商標です。
This product includes software developed by the Apache Software Foundation (http://www.apache.org/).

■マイクロソフト製品の表記について

このマニュアルでは，マイクロソフト製品の名称を次のように表記しています。

製品名	表記		
Microsoft(R) Windows(R) 7 Enterprise（32bit）	Windows 7（32bit）		Windows
Microsoft(R) Windows(R) 7 Professional（32bit）			
Microsoft(R) Windows(R) 7 Ultimate（32bit）			
Microsoft(R) Windows(R) 7 Enterprise（x64）	Windows 7（x64）		
Microsoft(R) Windows(R) 7 Professional（x64）			
Microsoft(R) Windows(R) 7 Ultimate（x64）			
Microsoft(R) Windows Server(R) 2003, Enterprise Edition 日本語版	Windows Server 2003 Enterprise Edition	Windows Server 2003	
Microsoft(R) Windows Server(R) 2003, Standard Edition 日本語版	Windows Server 2003 Standard Edition		
Microsoft(R) Windows Server(R) 2003 R2, Enterprise Edition 日本語版	Windows Server 2003 R2 Enterprise Edition	Windows Server 2003 R2	
Microsoft(R) Windows Server(R) 2003 R2, Standard Edition 日本語版	Windows Server 2003 R2 Standard Edition		
Microsoft(R) Windows Server(R) 2003, Enterprise x64 Edition 日本語版	Windows Server 2003 Enterprise x64 Edition	Windows Server 2003（x64）	
Microsoft(R) Windows Server(R) 2003, Standard x64 Edition 日本語版	Windows Server 2003 Standard x64 Edition		
Microsoft(R) Windows Server(R) 2003 R2, Enterprise x64 Edition 日本語版	Windows Server 2003 R2 Enterprise x64 Edition	Windows Server 2003 R2（x64）	
Microsoft(R) Windows Server(R) 2003 R2, Standard x64 Edition 日本語版	Windows Server 2003 R2 Standard x64 Edition		
Microsoft(R) Windows Server(R) 2008 Enterprise 32-bit 日本語版	Windows Server 2008 x86		
Microsoft(R) Windows Server(R) 2008 Standard 32-bit 日本語版			
Microsoft(R) Windows Server(R) 2008 Enterprise 日本語版	Windows Server 2008 x64		
Microsoft(R) Windows Server(R) 2008 Standard 日本語版			

製品名	表記		
Microsoft(R) Windows Server(R) 2008 R2 Enterprise 日本語版	Windows Server 2008 R2		
Microsoft(R) Windows Server(R) 2008 R2 Standard 日本語版			
Microsoft(R) Windows Vista(R) Business	Windows Vista Business	Windows Vista	
Microsoft(R) Windows Vista(R) Enterprise	Windows Vista Enterprise		
Microsoft(R) Windows Vista(R) Ultimate	Windows Vista Ultimate		
Microsoft(R) Windows(R) XP Professional Operating System	Windows XP		

■発行

2011 年 7 月 3020-3-U27-40

■著作権

変更内容

変更内容 (3020-3-U27-40)uCosminexus Application Server Enterprise 08-70，uCosminexus Application Server Standard 08-70，uCosminexus Application Server Standard-R 08-70，uCosminexus Client 08-70，uCosminexus Developer Professional 08-70，uCosminexus Developer Standard 08-70，uCosminexus Service Architect 08-70，uCosminexus Service Platform 08-70

追加・変更内容	変更個所
JAXB2.2 をサポートした。	1.1，1.1.1，1.3.1，2.4，2.5.1，4.7，付録 A.6，付録 B.1
StAX をサポートした。	1.1.1，1.1.2，1.3.1，2.1，2.2，2.2.2，2.2.3，2.2.4，2.2.8，4.2，6.5
JAXP1.4 をサポートした。	1.1.1，1.3.1，6.2
Apache 独自の機能をサポートした。	1.3.7，3.5
csmxjc コマンドのオプションで出力される date 要素の形式を変更した。	2.5.1
csmschemagen コマンドに -encoding オプションを追加した。	2.5.1
JAXB に関する注意事項を変更した。	6.21
対象製品に uCosminexus Application Server Standard-R を追加した。	－
次の製品の適用 OS に AIX V7.1 を追加した。 • uCosminexus Application Server Standard • uCosminexus Application Server Enterprise • uCosminexus Service Platform	－
次の製品の適用 OS に Red Hat Enterprise Linux Server 6（32-bit x86），Red Hat Enterprise Linux Server 6（64-bit x86_64）を追加した。 • uCosminexus Application Server Standard • uCosminexus Application Server Enterprise • uCosminexus Service Platform	－
次の製品の適用 OS から Linux（IPF）を削除した。 • uCosminexus Application Server Standard • uCosminexus Application Server Enterprise • uCosminexus Service Platform	－

単なる誤字・脱字などはお断りなく訂正しました。

はじめに

このマニュアルは，Cosminexus XML Processor が提供する XML パーサ・XSLT トランスフォーマ・JAXB・JAXP の機能，作成方法，および使用方法について説明したものです。Cosminexus XML Processor は，Cosminexus を構成する各製品で提供されています。詳細については，マニュアル「Cosminexus アプリケーションサーバ 概説」の「2.3　構成ソフトウェア」を参照してください。

■対象読者

このマニュアルは，Cosminexus XML Processor を使用して，XML 文書を操作するプログラムを Java 言語で作成されるプログラマの方を対象としています。

次の内容を理解されていることを前提としています。

- XML についての基本的な知識
- JAXB についての基本的な知識
- JAXP についての基本的な知識
- StAX についての基本的な知識
- Java 言語によるオブジェクト指向プログラミングの知識

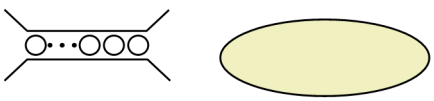
■ Cosminexus XML Processor で出力されるメッセージの説明について

Cosminexus XML Processor で出力されるメッセージの説明については，マニュアル「Cosminexus アプリケーションサーバ メッセージ 3」の「2.　KECX（Cosminexus XML Processor が出力するメッセージ）」に記載しています。

■このマニュアルの図中で使用している記号

このマニュアルの図中で使用している記号を，次のように定義します。

●キュー　　●ネットワーク



■このマニュアルの文法で使用している記号

このマニュアルでは，次に示す記号を使用して，コマンドの形式を記述しています。

記号	意味
[ ]	この記号で囲まれている項目は，省略してもよいことを示します。
< >	この記号で囲まれている項目は，サービスロケーションやファイルなどの可変値を指定することを示します。

目次

1 Cosminexus XML Processor の概要

この章では，Cosminexus XML Processor を使用する前に知っておく必要のある Cosminexus XML Processor の特長，位置づけ，サポート範囲について説明します。
サポート範囲では，使用できる XML パーサ・XSLT トランスフォーマの機能，JAXB データバインディングの機能，および処理できる文字コードについて説明し，Cosminexus XML Processor の拡張機能について紹介します。

1.1 Cosminexus XML Processor の特長

Cosminexus XML Processor は Java 言語用の標準 XML API である JAXP をサポートし，その実装である XML パーサ機能，XSLT トランスフォーマ機能を提供するミドルウェアです。また，JAXB による JAXB データバインディング機能や StAX ストリーミングパーサ機能も提供します。次に Cosminexus XML Processor の特長を説明します。

1.1.1 プラットフォーム間で共通のプログラムを作成できる

Cosminexus XML Processor は JAXP，JAXB，および StAX をサポートします。

- JAXP（Java API for XML Processing）は，米国 Sun Microsystems, Inc. が提供する Java 言語用の標準 XML API です。JAXP には，XML パーサ（DOM パーサ・SAX パーサ），XSLT トランスフォーマなどを生成する API が含まれています。
- JAXB（The Java Architecture for XML Binding）は，Java Community Process の JSR 222 で定義されている標準 XML API です。JAXB には，XML Schema から Java へ変換するスキーマコンパイラ，Java から XML Schema へ変換するスキーマジェネレータ，および XML 文書と Java クラス間のデータを交換する API が含まれています。
- StAX（Streaming API for XML）は，JAXP1.4 からサポートされるようになった，Java Community Process の JSR 173 で定義されている標準 XML API です。StAX には，XML 文書のストリーミングパースを実行するための API が含まれています。StAX は，Java SE 6 上で動作している場合に使用できます。

Cosminexus XML Processor がサポートしている標準 XML API は，XML パーサ，XSLT トランスフォーマ，JAXB データバインディングなどの実装やコードに依存しません。そのため，XML 文書を処理・操作するプログラムを作成する際に，使用する XML パーサ，XSLT トランスフォーマ，JAXB データバインディングなどの実装を意識しなくても，異なるプラットフォーム間で共通のプログラムを作成できます。

注意事項

uCosminexus Operator では，JAXB 機能を使用できません。

1.1.2 Java プログラムで XML 文書を容易に解析できる

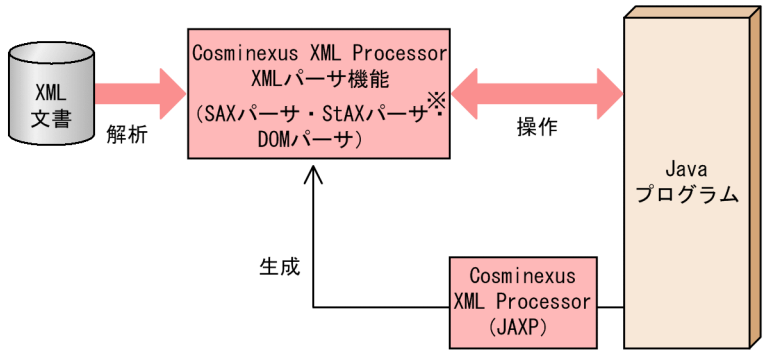
Cosminexus XML Processor は JAXP の実装を提供しているため，Java プログラムで XML パーサを使用できます。

XML パーサは，XML 文書を解析し，解析したデータを扱いやすいデータ構造にして，

ユーザプログラムに提供するライブラリです。XML 文書を解析する際には，XML 文書内のタグや属性などが，XML Schema や DTD に従っているかどうかも検証できます。

JAXP で XML 文書を解析する流れを次の図に示します。

図 1-1　JAXP で XML 文書を解析する流れ



注※ StAXパーサは，Java SE 6上で動作している場合だけ使用できます。

JAXP を使用すると，Java プログラムで XML パーサ（SAX パーサ，StAX パーサ，または DOM パーサ）を生成して，XML 文書を解析できます。

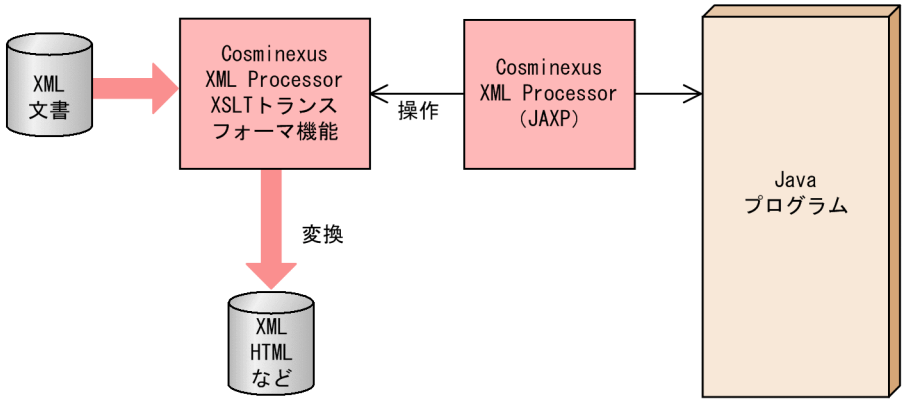
1.1.3　Java プログラムで XML 文書を容易に変換できる

Cosminexus XML Processor は JAXP の実装を提供しているため，Java プログラムで XSLT トランスフォーマを使用できます。

XSLT は XML 文書の構造変換を行うための仕様です。XSLT を実装したものを XSLT トランスフォーマと呼びます。XSLT トランスフォーマは XML 文書を読み込み，加工して出力します。XML 文書だけでなく，HTML，テキストも出力できます。

JAXP で XML 文書を変換する流れを次の図に示します。JAXP を使用すると，Java プログラムで XSLT トランスフォーマの XML 文書の変換処理を操作できます。

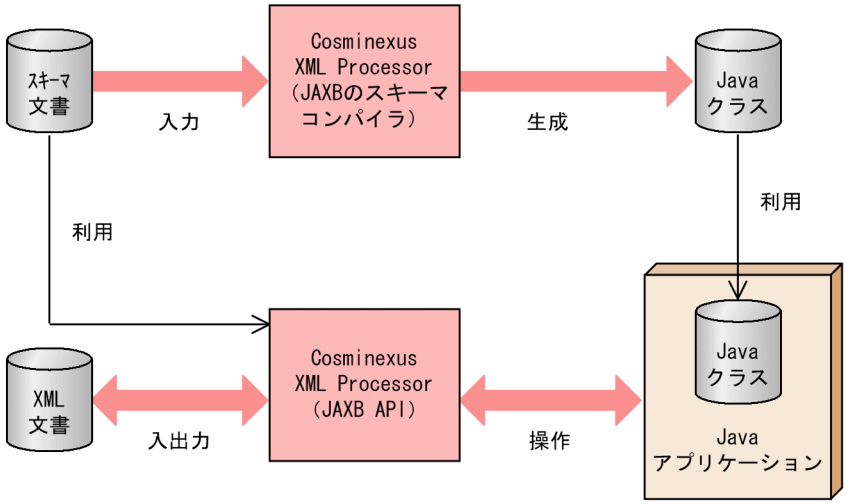
図 1-2　JAXP で XML 文書を変換する流れ



1.1.4　XML 基盤のアプリケーションの開発容易性が向上する

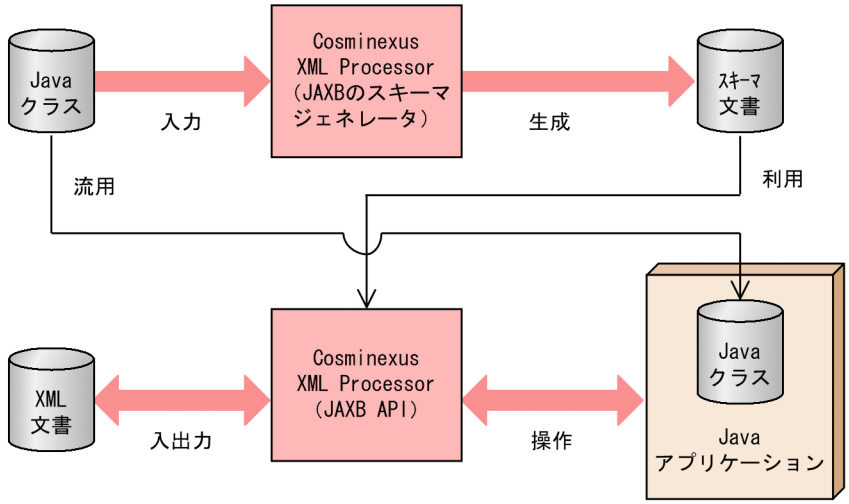
Cosminexus XML Processor は JAXB の実装を提供しています。スキーマ文書をスキーマコンパイラに入力すると，スキーマ文書で定義した XML 文書にアクセスするための Java クラスを生成します。この Java クラスを使用すると，XML 基盤のアプリケーションを容易に開発できます。JAXB を使用して XML を処理する流れを次の図に示します。

図 1-3　JAXB を使用して XML を処理する流れ（新規開発）



スキーマジェネレータに Java クラスを入力すると，Java クラスのデータ構造と等価なスキーマ文書を生成します。これによって，既存資産である Java クラスを流用して，XML 基盤のアプリケーションを容易に構築できます。JAXB を使用して，既存の Java クラスで XML 基盤のアプリケーションを構築する流れを次の図に示します。

図 1-4　JAXB を使用して XML を処理する流れ（既存 Java クラス流用）



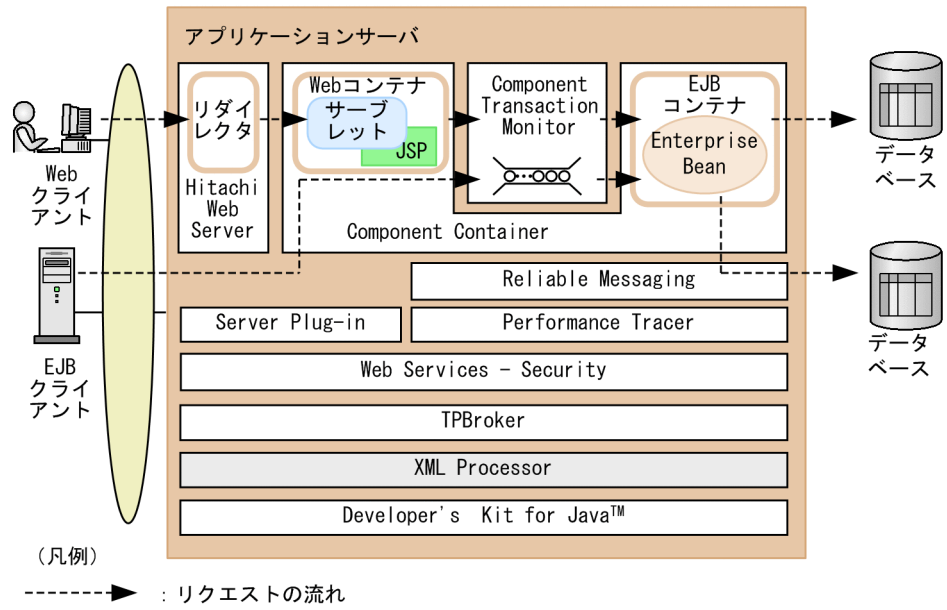
1.2　Cosminexus XML Processor の位置づけ

この節では，Cosminexus Version 8 環境での Cosminexus XML Processor の位置づけを説明します。

Cosminexus XML Processor は，Cosminexus Version 8 で XML 文書の解析，操作，生成と XSLT 変換などを必要とする局面で使用します。また，Cosminexus Version 8 のほかの製品と組み合わせることで，SOAP 通信基盤などの，昨今必要とされている XML 基盤のアプリケーション統合を可能にする環境を構築できます。

Cosminexus の実行環境での Cosminexus XML Processor の位置づけを次の図に示します。

図 1-5　Cosminexus 実行環境での Cosminexus XML Processor の位置づけ



注：　図中では，構成ソフトウェア名の「Cosminexus」は省略しています。

1.3 Cosminexus XML Processor のサポート範囲

この節では，Cosminexus XML Processor のサポート範囲について説明します。

1.3.1 使用できる XML パーサ，XSLT トランスフォーマなどの機能

Cosminexus XML Processor のサポート範囲について説明します。

Cosminexus XML Processor は，JAXP，JAXB，および StAX の仕様をサポートしています。JAXP を介して使用される XML パーサ機能，XSLT トランスフォーマ機能などは，次に示す JAXP で定義された仕様の範囲内で使用できます。

注意事項

JAXP の各バージョン間には，若干の仕様差があります。同様に，JAXB2.2 も，従来の JAXB2.1 に対して若干の仕様差があるので注意してください。

JAXB のサポート範囲の詳細については，「付録 B　JAXB 仕様のサポート範囲」を参照してください。

(1) J2SE 5.0 を使用する場合

JAXP で定義された仕様

- JSR 206 Java API for XML Processing (JAXP) 1.3
- Extensible Markup Language (XML)
  - Extensible Markup Language (XML) 1.1 [http://www.w3.org/TR/xml11]
  - Extensible Markup Language (XML) 1.0 (Second Edition) [http://www.w3.org/TR/2000/REC-xml-20001006]
  - Extensible Markup Language (XML) 1.0 (Second Edition) Errata [http://www.w3.org/XML/xml-V10-2e-errata]
- Namespaces in XML
  - Namespaces in XML 1.1 [http://www.w3.org/TR/xml-names11/]
  - Namespaces in XML 1.0 [http://www.w3.org/TR/REC-xml-names]
  - Namespaces in XML 1.0 Errata [http://www.w3.org/XML/xml-names-19990114-errata]
- XML Schema
  - XML Schema Part 1: Structures [http://www.w3.org/TR/xmlschema-1/]
  - XML Schema Part 1: Structures Errata [http://www.w3.org/2001/05/

xmlschema-errata#Errata1]
  - XML Schema Part 2: Datatypes [http://www.w3.org/TR/xmlschema-2/]
  - XML Schema Part 2: Datatypes Errata [http://www.w3.org/2001/05/xmlschema-errata#Errata2]
- XSL Transformations (XSLT)
  - XSL Transformations (XSLT) Version 1.0 [http://www.w3.org/TR/xslt]
- XML Path Language (XPath)
  - XML Path Language (XPath) Version 1.0 [http://www.w3.org/TR/xpath]
  - XML Path Language (XPath) Version 1.0 Errata [http://www.w3.org/1999/11/REC-xpath-19991116-errata]
- XML Inclusions (XInclude)
  - XML Inclusions (XInclude) Version 1.0 [http://www.w3.org/TR/xinclude/]
- Document Object Model (DOM) Level 3
  - Document Object Model (DOM) Level 3 Core [http://www.w3.org/TR/DOM-Level-3-Core]
  - Document Object Model (DOM) Level 3 Load and Save [http://www.w3.org/TR/DOM-Level-3-LS]
- Simple API for XML (SAX)
  - Simple API for XML (SAX) 2.0.2 (sax2r3) [http://sax.sourceforge.net/]
  - Simple API for XML (SAX) 2.0.2 (sax2r3) Extensions [http://sax.sourceforge.net/sax2-ext.html]

JAXB で定義された仕様

- JSR 222 The Java Architecture for XML Binding 2.2

(2) Java SE 6 を使用する場合

JAXP で定義された仕様

- JSR 206 Java™ API for XML Processing (JAXP) 1.4
- Extensible Markup Language (XML)
  - Extensible Markup Language (XML) 1.1 [http://www.w3.org/TR/xml11]
  - XML 1.1 First Edition Specification Errata [http://www.w3.org/XML/xml-V11-1e-errata]
  - Extensible Markup Language (XML) 1.0 (Third Edition) [http://www.w3.org/TR/2004/REC-xml-20040204/]
  - Extensible Markup Language (XML) 1.0 (Third Edition) Errata [http://www.w3.org/XML/xml-V10-3e-errata]
- Namespaces in XML
  - Namespaces in XML 1.1 [http://www.w3.org/TR/xml-names11/]

  - Namespaces in XML 1.1 Errata [http://www.w3.org/XML/2004/xml-names11-errata]
  - Namespaces in XML 1.0 [http://www.w3.org/TR/REC-xml-names]
  - Namespaces in XML 1.0 Errata [http://www.w3.org/XML/xml-names-19990114-errata]
- XML Schema
  - XML Schema Part 1: Structures Second Edition [http://www.w3.org/TR/xmlschema-1/]
  - XML Schema Part 1: Structures Second Edition Errata [http://www.w3.org/2004/03/xmlschema-errata#Errata1],
  - XML Schema Part 2: Datatypes Second Edition [http://www.w3.org/TR/xmlschema-2/]
  - XML Schema Part 2: Datatypes Second Edition Errata [http://www.w3.org/2004/03/xmlschema-errata#Errata2]
- XSL Transformations (XSLT)
  - XSL Transformations (XSLT) Version 1.0 [http://www.w3.org/TR/xslt]
  - XSL Transformations (XSLT) Version 1.0 Errata [http://www.w3.org/1999/11/REC-xslt-19991116-errata]
- XML Path Language (XPath)
  - XML Path Language (XPath) Version 1.0 [http://www.w3.org/TR/xpath]
  - XML Path Language (XPath) Version 1.0 Errata [http://www.w3.org/1999/11/REC-xpath-19991116-errata]
- XML Inclusions (XInclude)
  - XML Inclusions (XInclude) Version 1.0 [http://www.w3.org/TR/xinclude/]
- Document Object Model (DOM) Level 3
  - Document Object Model (DOM) Level 3 Core [http://www.w3.org/TR/DOM-Level-3-Core]
  - Document Object Model (DOM) Level 3 Load and Save [http://www.w3.org/TR/DOM-Level-3-LS]
- Simple API for XML (SAX)
  - Simple API for XML (SAX) 2.0.2 (sax2r3) [http://sax.sourceforge.net/]
  - Simple API for XML (SAX) 2.0.2 (sax2r3) Extensions [http://sax.sourceforge.net/?selected=ext]
- Streaming API for XML(StAX)
  - Streaming API for XML (StAX) Version1.0 [http://jcp.org/en/jsr/detail?id=173]

JAXB で定義された仕様

- JSR 222 The Java™ Architecture for XML Binding 2.2

StAX で定義された仕様

- JSR 173 Streaming API for XML 1.0

1.3.2 処理できる文字コード

XML 文書の文字コードは，XML 文書に記述する XML 宣言の encoding 属性（encoding="XXX" の XXX の部分）に指定します。このほかにも，XML 文書の入力ソースとなる InputSource オブジェクトのエンコーディングに指定する方法などがあります。

Cosminexus XML Processor では，IANA に登録された文字コードのうち次のものを処理できます。文字コードの定義については，IANA の CHARACTER SETS に関するドキュメントを参照してください。

- UTF-8
- UTF-16
- UTF-16BE
- UTF-16LE
- Shift_JIS ※1
- Windows-31J
- ISO-2022-JP
- EUC-JP
- US-ASCII
- ISO-10646-UCS-2
- ISO-10646-UCS-4 ※2
- ISO-8859-1

注※1

Shift_JIS を使用するときに適用される文字エンコーディング（SJIS または MS932）の切り替えについては，「1.3.3　Shift_JIS 切り替え機能」を参照してください。

注※2

バイトオーダーが big-endian または little-endian であり，かつ，Byte-Order-Mark が付加されていない形式だけをサポートします。

注意事項

上記以外の文字コードを使用した場合の動作は保証しません。

1.3.3 Shift_JIS 切り替え機能

Shift_JIS 切り替え機能は，XML 文書の encoding 属性に Shift_JIS が指定されている場

合に，Cosminexus XML Processor が適用する文字エンコーディングを切り替える機能です。文字エンコーディングは，XML パーサや XSLT トランスフォーマのインスタンスごとに切り替えられます。

XML 文書の encoding に指定されている文字エンコーディングは，Cosminexus XML Processor が提供する Shift_JIS 切り替え機能での指定に従って，SJIS（x-sjis-jdk1.1.7）または MS932（x-sjis-cp932）に切り替えられます。XML 文書の encoding 指定と適用される文字エンコーディングとの対応については，「3.1　Shift_JIS 切り替え機能」の表 3-1 を参照してください。

Shift_JIS 切り替え機能の使用方法については，「3.1　Shift_JIS 切り替え機能」を参照してください。

1.3.4　XSLTC トランスフォーマ機能

XSLTC トランスフォーマは，Cosminexus XML Processor が提供している XSLT トランスフォーマと同じ機能を持ちながら，XML 文書の変換処理の性能を向上させたものです。Cosminexus XML Processor では，XSLT トランスフォーマと XSLTC トランスフォーマの両方を使用したアプリケーションの開発が可能です。

XSLTC トランスフォーマ機能の使用方法については，「3.2.4　XSLTC トランスフォーマの使用方法」を参照してください。

1.3.5　スキーマキャッシュ機能

スキーマキャッシュ機能とは，XML Schema で定義されたスキーマ文書を解析処理した結果（grammar オブジェクト）を事前にメモリ上，またはディスク上にキャッシュする機能です。スキーマ文書を grammar オブジェクトに変換する処理をキャッシュの参照に置き換えるため，妥当性検証の性能を向上させることができます。

キャッシュの対象とするスキーマ文書は，設定ファイル（スキーマ定義ファイル）に記述する必要があります。

スキーマキャッシュ機能の使用方法については，「3.3　スキーマキャッシュ機能」を参照してください。

1.3.6　高速パース機能

高速パース機能とは，事前解析によって解析対象の XML 文書の特徴を学習し，以降の XML 文書解析時に学習結果を利用することで，解析処理の実行速度を向上させる機能です。

高速パース機能の使用方法については，「3.4　高速パース機能」を参照してください。

1.3.7 Apache 独自の機能

Cosminexus XML Processor の 08-70 以降では，JAXP の規格とは別に，Apache 独自のフィーチャー，プロパティ，および名前空間 URI を使用できます。これらの独自機能は，XML パーサや XSLT・XSLTC トランスフォーマで使用できます。ただし，JAXP の規格外の機能であるため，動作は保証されません。そのため，事前に十分な動作確認を実施するなどの対策が必要です。

Apache 独自の機能の詳細については，「3.5　Apache 独自の機能」を参照してください。

2 JAXP および JAXB の機能

この章では，JAXP および JAXB を使用してプログラムを作成する際に知っておく必要のある JAXP および JAXB の概要について説明します。

2.1 JAXP とは

JAXP（Java API for XML Processing）は，米国 Sun Microsystems, Inc. が提供する Java 言語用の標準 XML API です。

Cosminexus XML Processor が提供する JAXP の機能を次の表に示します。

表 2-1 Cosminexus XML Processor が提供する JAXP の機能

機能の名称	機能の概要
DOM	XML 文書を解析して DOM ツリーを生成します。また，生成した DOM ツリーを操作します。
SAX	XML 文書を解析して SAX のイベントを発生させます。また，発生したイベントを処理します。
StAX	XML 文書を解析して StAX のイベントを発生させます。また，発生したイベントを処理します。ハンドラが不要なため，XML 文書を手続き的に処理できます。
XSLT	XML 文書を入力し，スタイルシートに基づいて変換し，ほかの XML 文書，HTML，テキストとして出力します。
XPath	XPath 式を評価します。
Validation	XML 文書をスキーマ文書に基づいて検証します。
Datatype	W3C XML Schema 1.0 で規定された日付／時刻型データを処理します。

JAXP には，DOM パーサ，SAX パーサ，StAX パーサ，XSLT トランスフォーマ，XPath オブジェクト，Validation オブジェクト，および Datatype オブジェクトを生成する API が含まれています。これらの API は JAXP の実装に依存しないので，XML 文書を処理・操作するプログラムを作成する際に，使用する XML プロセッサの実装を意識する必要がありません。そのため，異なるプラットフォーム間で共通のプログラムを作成できます。

2.2 JAXP が規定するパッケージとその機能

JAXP は，表 2-1 に示した機能を利用するためのパッケージを規定しています。JAXP が規定するパッケージに含まれる API の概要を次の表に示します。

表 2-2 JAXP が規定するパッケージに含まれる API の概要

パッケージの名称	パッケージに含まれる API の概要
javax.xml.parsers	DOM パーサおよび SAX パーサでの解析を行うための API。
javax.xml.transform	XSLT トランスフォーマのための API。
javax.xml.transform.dom	XSLT トランスフォーマの入出力に DOM を使用するための API。
javax.xml.transform.sax	XSLT トランスフォーマの入出力に SAX を使用するための API。
javax.xml.transform.stream	XSLT トランスフォーマの入出力にストリームを使用するための API。
javax.xml.validation	XML 文書を検証するための API。
javax.xml.xpath	XPath 式を評価するための API。
javax.xml.namespace	XML 名前空間を処理するための API。
javax.xml.datatype	W3C XML Schema 1.0 で規定された日付／時刻型データを処理するための API。
javax.xml	XML 関連の文字列定数。
org.w3c.dom	DOM ツリーを操作する API。
org.w3c.dom.bootstrap	DOM 実装を取得するための API。
org.w3c.dom.ls	XML 文書をロードおよび保存するための API。
org.w3c.dom.events	DOM のイベント処理のための API。
org.xml.sax	SAX の基本処理のための API。
org.xml.sax.ext	SAX の拡張処理のための API。
org.xml.sax.helpers	SAX の処理のためのヘルパークラス群。
javax.xml.stream	StAX パースのための API。
javax.xml.stream.events	StAX のイベント処理のための API。
javax.xml.stream.util	StAX の処理のためのユーティリティ API。
javax.xml.transform.stax	XSLT トランスフォーマの入出力に StAX を使用するための API。

以降では，それぞれのパッケージの概要について説明します。

2.2.1　javax.xml.parsers パッケージ

javax.xml.parsers パッケージは，DOM パーサ，および SAX パーサでの解析を行うための API を提供するパッケージです。

DOM パーサでの解析を行うアプリケーションの処理の流れを図 2-1 に，SAX パーサでの解析を行うアプリケーションの処理の流れを図 2-2 に示します。

図 2-1　DOM パーサでの解析を行うアプリケーションの処理の流れ

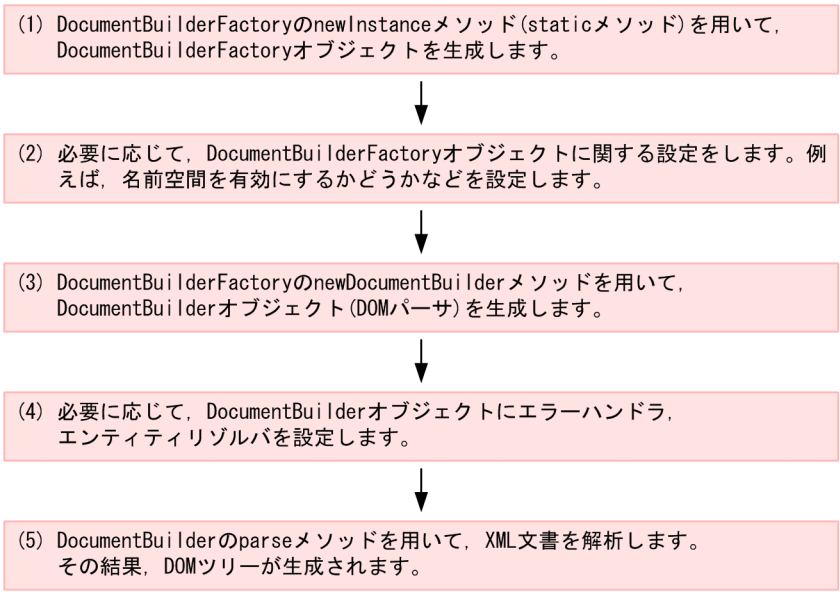
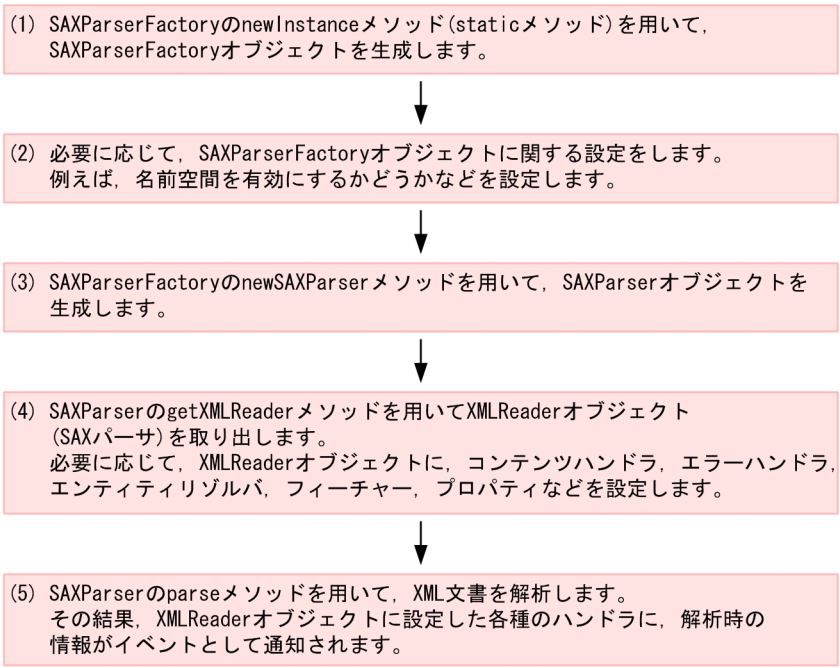


図 2-2　SAX パーサでの解析を行うアプリケーションの処理の流れ



上記のほかに，XMLReaderFactory クラスを用いて SAX パーサを生成する方法なども可能です。

javax.xml.parsers パッケージの詳細については，JSR 206 Java API for XML Processing(JAXP) 1.3 の Chapter 7. Package javax.xml.parsers を参照してください。

DOM パーサでの解析を行うコーディング例については，「5.4　DOM パーサを使用するサンプルプログラム」を，SAX パーサでの解析を行うコーディング例については，「5.5　SAX パーサを使用するサンプルプログラム」を参照してください。

2.2.2　javax.xml.stream パッケージ

javax.xml.stream パッケージは，StAX パースのための API を提供するパッケージです。このパッケージは，Java SE 6 の場合だけ使用できます。

javax.xml.stream パッケージの詳細については，Streaming API for XML Version1.0(JSR-173) の Javadoc を参照してください。

2.2.3　javax.xml.stream.events パッケージ

javax.xml.stream.events パッケージは，StAX のイベント処理の API を提供するパッケージです。このパッケージは，Java SE 6 の場合だけ使用できます。

javax.xml.stream.events パッケージの詳細については，Streaming API for XML Version1.0(JSR-173) の Javadoc を参照してください。

2.2.4 javax.xml.stream.util パッケージ

javax.xml.stream.util パッケージは，StAX の処理のためのユーティリティ API を提供するパッケージです。このパッケージは，Java SE 6 の場合だけ使用できます。

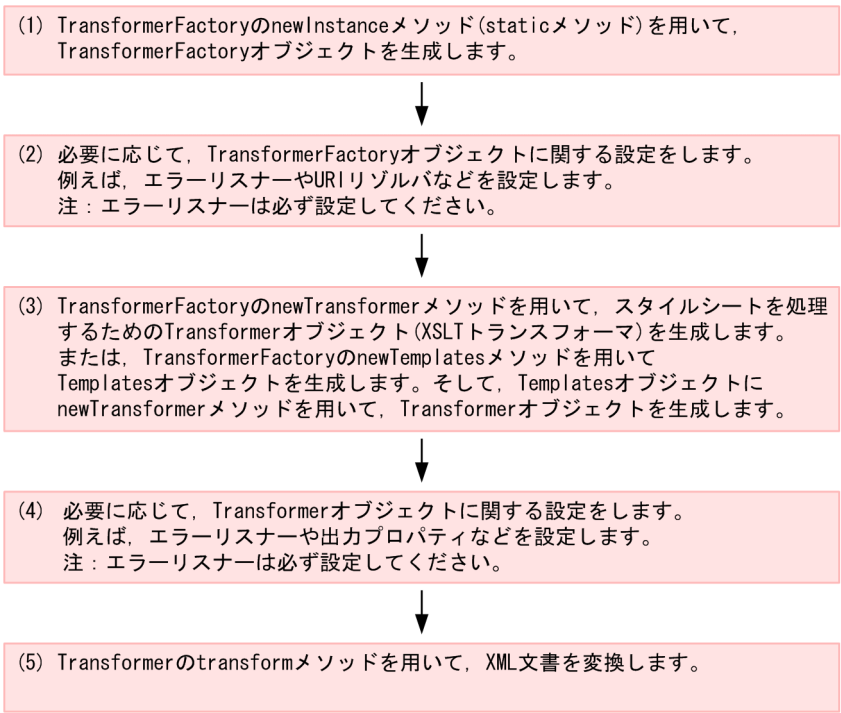
javax.xml.stream.util パッケージの詳細については，Streaming API for XML Version1.0(JSR-173) の Javadoc を参照してください。

2.2.5　javax.xml.transform パッケージ

javax.xml.transform パッケージは，XSLT トランスフォーマのための API を提供するパッケージです。

XSLT による変換を行うアプリケーションの処理の流れを次の図に示します。

図 2-3　XSLT による変換を行うアプリケーションの処理の流れ



javax.xml.transform パッケージの詳細については，JSR 206 Java API for XML Processing(JAXP) 1.3 の Chapter 8. Package javax.xml.transform を参照してください。

XSLT による変換を行うコーディング例については，「5.7　XSLT トランスフォーマを使用するサンプルプログラム」を参照してください。

2.2.6　javax.xml.transform.dom パッケージ

javax.xml.transform.dom パッケージは，XSLT トランスフォーマの入出力に DOM を使用するための API を提供するパッケージです。

javax.xml.transform.dom パッケージの詳細については，JSR 206 Java API for XML Processing(JAXP) 1.3 の Chapter 9. Package javax.xml.transform.dom を参照してください。

2.2.7 javax.xml.transform.sax パッケージ

javax.xml.transform.sax パッケージは，XSLT トランスフォーマの入出力に SAX を使用するための API を提供するパッケージです。

javax.xml.transform.sax パッケージの詳細については，JSR 206 Java API for XML Processing(JAXP) 1.3 の Chapter 10. Package javax.xml.transform.sax を参照してください。

2.2.8 javax.xml.transform.stax パッケージ

javax.xml.transform.stax パッケージは，XSLT トランスフォーマの入出力に StAX を使用するための API を提供するパッケージです。このパッケージは，Java SE 6 の場合だけ使用できます。

javax.xml.transform.stax パッケージの詳細については，Streaming API for XML Version1.0(JSR-173) の Javadoc を参照してください。

2.2.9 javax.xml.transform.stream パッケージ

javax.xml.transform.stream パッケージは，XSLT トランスフォーマの入出力にストリームを使用するための API を提供するパッケージです。

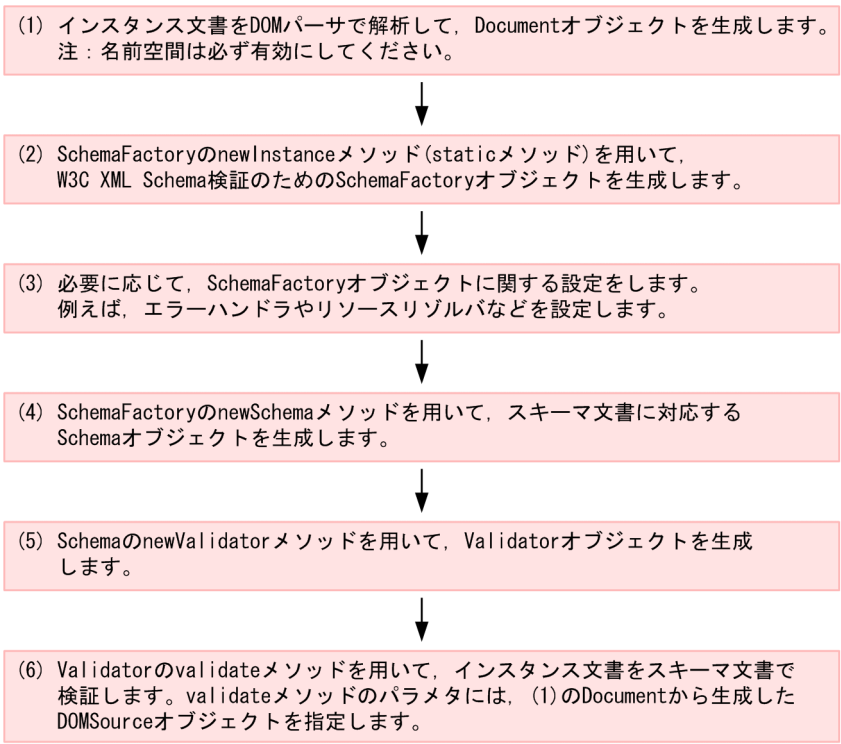
javax.xml.transform.stream パッケージの詳細については，JSR 206 Java API for XML Processing(JAXP) 1.3 の Chapter 11.Package javax.xml.transform.stream を参照してください。

2.2.10 javax.xml.validation パッケージ

javax.xml.validation パッケージは，XML 文書を検証するための API を提供するパッケージです。

javax.xml.validation パッケージに含まれるクラスを用いて XML 文書を検証するアプリケーションの処理の流れを次の図に示します。

図 2-4 javax.xml.validation パッケージに含まれるクラスを用いて XML 文書を検証するアプリケーションの処理の流れ



上記のほかに，DOM パーサでの解析と検証を同時に行うことも可能です。また，SAX パーサでの解析と検証を同時に行うことも可能です。

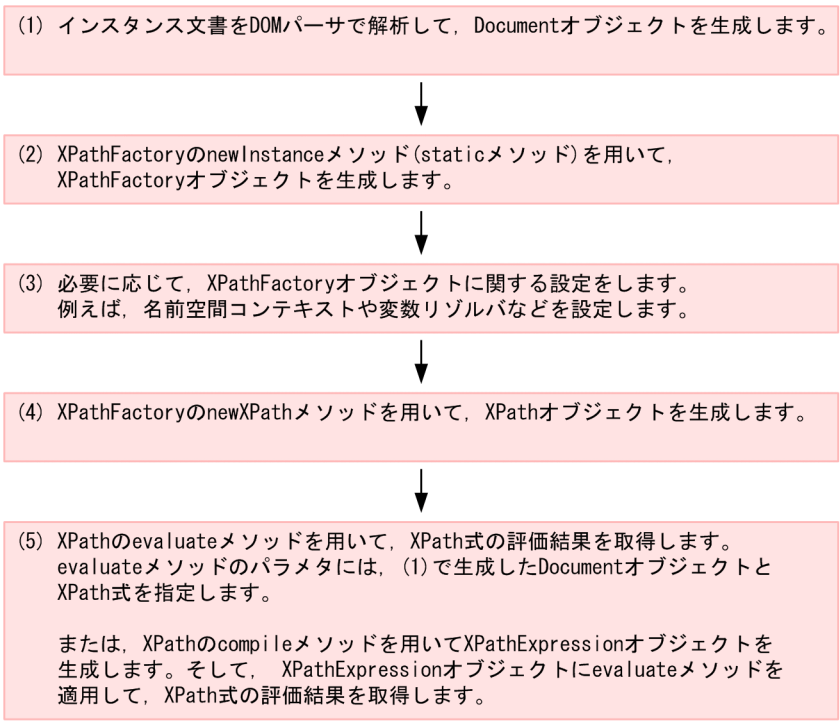
javax.xml.validation パッケージの詳細については，JSR 206 Java API for XML Processing(JAXP) 1.3 の Chapter 14. Package javax.xml.validation を参照してください。

2.2.11　javax.xml.xpath パッケージ

javax.xml.xpath パッケージは，XPath 式を評価するための API を提供するパッケージです。

XPath 式を評価するアプリケーションの処理の流れを次の図に示します。

図 2-5　XPath 式を評価するアプリケーションの処理の流れ



上記のほかに，InputSource オブジェクトを入力として XPath 式を評価できます。

javax.xml.xpath パッケージの詳細については，JSR 206 Java API for XML Processing(JAXP) 1.3 の Chapter 13. Package javax.xml.xpath を参照してください。

2.2.12　javax.xml.namespace パッケージ

javax.xml.namespace パッケージは，XML 名前空間を処理するための API を提供するパッケージです。このパッケージは，javax.xml.xpath パッケージおよび javax.xml.datatype パッケージに含まれる API の引数や戻り値としてだけ使用します。

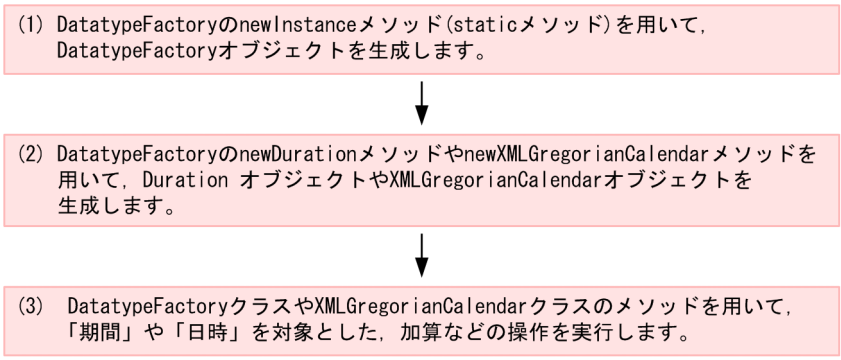
javax.xml.namespace パッケージの詳細については，JSR 206 Java API for XML Processing(JAXP) 1.3 の Chapter 12. Package javax.xml.namespace を参照してください。

2.2.13 javax.xml.datatype パッケージ

javax.xml.datatype パッケージは，W3C XML Schema 1.0 で規定された日付／時刻型データを処理するための API を提供するパッケージです。

日付／時刻型データを処理するアプリケーションの処理の流れを次の図に示します。

図 2-6 日付／時刻型データを処理するアプリケーションの処理の流れ



javax.xml.datatype パッケージの詳細については，JSR 206 Java API for XML Processing(JAXP) 1.3 の Chapter 15. Package javax.xml.datatype を参照してください。

2.2.14 javax.xml パッケージ

javax.xml パッケージは，XML 関連の文字列定数を定義したパッケージです。

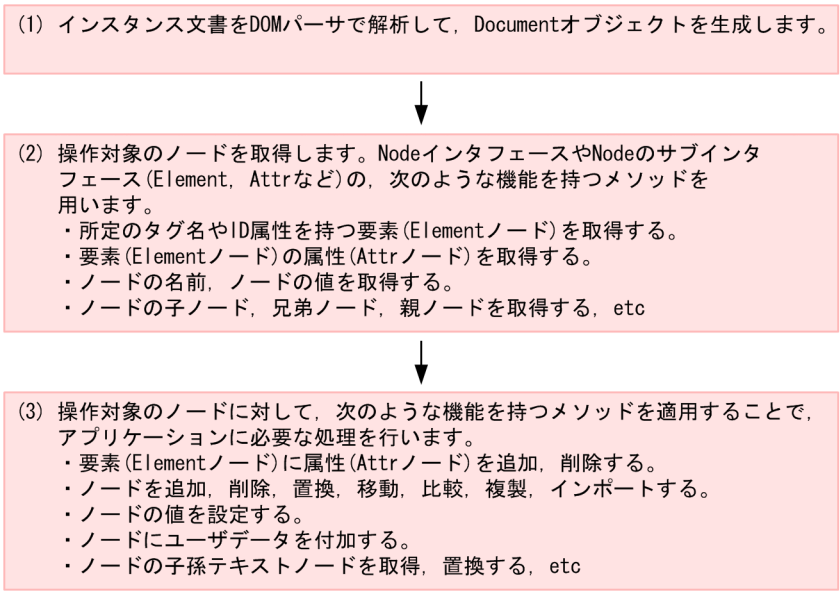
javax.xml パッケージの詳細については，JSR 206 Java API for XML Processing(JAXP) 1.3 の Chapter 16. Package javax.xml を参照してください。

2.2.15　org.w3c.dom パッケージ

org.w3c.dom パッケージは，DOM ツリーを操作する API を提供するパッケージです。

DOM ツリーを操作するアプリケーションの処理の流れを次の図に示します。

図 2-7　DOM ツリーを操作するアプリケーションの処理の流れ



org.w3c.dom パッケージの詳細については，W3C Document Object Model (DOM) Level 3 Core Specification の次に示す個所を参照してください。

- 1.4 Fundamental Interfaces: Core Module
- 1.5 Extended Interfaces: XML Module
- Appendix G: Java Language Binding G.2 Other Core interfaces

DOM ツリーの操作を行うコーディング例は，「5.4　DOM パーサを使用するサンプルプログラム」を参照してください。

2.2.16　org.w3c.dom.bootstrap パッケージ

org.w3c.dom.bootstrap パッケージは，DOM 実装を取得するための API を提供するパッケージです。

org.w3c.dom.bootstrap パッケージの詳細については，W3C Document Object Model (DOM) Level 3 Core Specification の次に示す個所を参照してください。

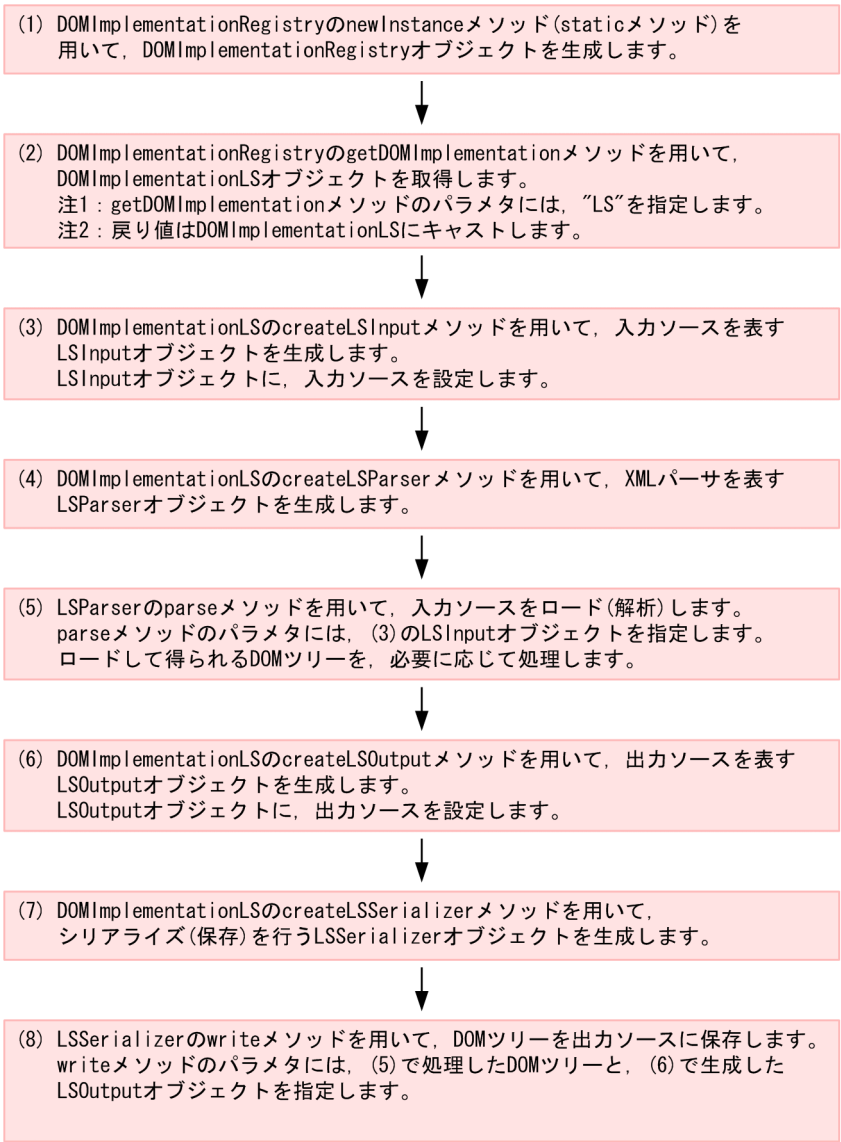
- 1.3.7 Bootstrapping
- Appendix G: Java Language Binding G.1 Java Binding Extension

2.2.17 org.w3c.dom.ls パッケージ

org.w3c.dom.ls パッケージは，XML 文書をロードおよび保存するための API を提供するパッケージです。

XML 文書のロード，および保存を行うアプリケーションの処理の流れを次の図に示します。

図 2-8 XML 文書のロード，および保存を行うアプリケーションの処理の流れ



org.w3c.dom.ls パッケージの詳細については，W3C Document Object Model (DOM) Level 3 Load and Save Specification の次に示す個所を参照してください。

- 1.3 Fundamental Interfaces
- Appendix B: Java Language Binding

2.2.18　org.w3c.dom.events パッケージ

org.w3c.dom.events パッケージは，org.w3c.dom.ls.LSParser インタフェースを実装した非同期 LSParser オブジェクトが使用します。Cosminexus XML Processor は非同期 LSParser オブジェクトをサポートしないので，このパッケージは使用しません。

org.w3c.dom.events パッケージの詳細については，W3C Document Object Model (DOM) Level 3 Events Specification の次に示す個所を参照してください。

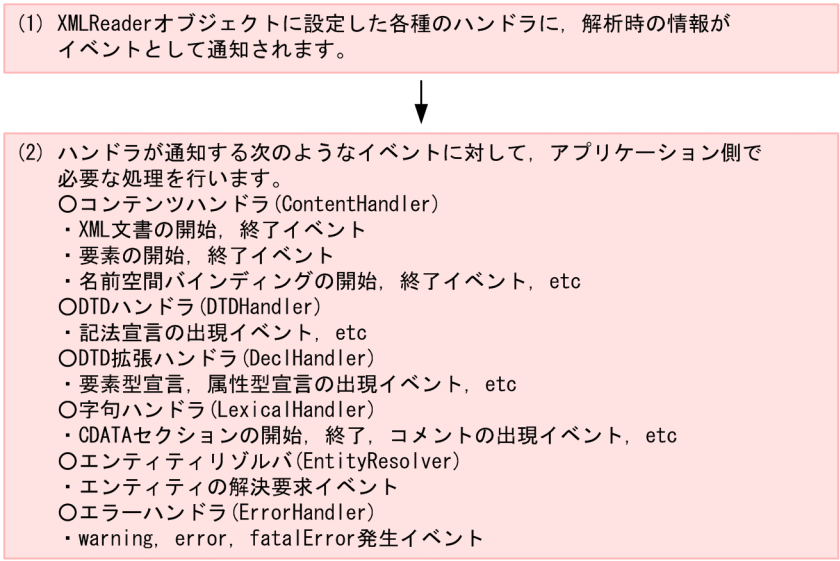
- 1.6 Basic interfaces
- 1.7 Event module definitions
- Appendix D: Java Language Binding

2.2.19　org.xml.sax パッケージ

org.xml.sax パッケージは，SAX の基本処理のための API を提供するパッケージです。

SAX イベントを処理するアプリケーションの処理の流れを次の図に示します。

図 2-9　SAX イベントを処理するアプリケーションの処理の流れ



org.xml.sax パッケージの詳細については，Simple API for XML (SAX) 2.0.2 (sax2r3) の Javadoc を参照してください。

SAX イベントを処理するコーディング例は，「5.5　SAX パーサを使用するサンプルプロ

グラム」を参照してください。

2.2.20 org.xml.sax.ext パッケージ

org.xml.sax.ext パッケージは，SAX の拡張処理のための API を提供するパッケージです。

org.xml.sax.ext パッケージの詳細については，Simple API for XML (SAX) 2.0.2 (sax2r3) Extensions の Javadoc を参照してください。

2.2.21 org.xml.sax.helpers パッケージ

org.xml.sax.helpers パッケージは，SAX の処理のためのヘルパークラス群を提供するパッケージです。

org.xml.sax.ext パッケージの詳細については，Simple API for XML (SAX) 2.0.2 (sax2r3) Extensions の Javadoc を参照してください。

2.3 JAXB とは

JAXB（The Java Architecture for XML Binding）は，Java Community Process の JSR 222 で定義されている Java 言語用の標準 XML API です。

Cosminexus XML Processor が提供する JAXB の機能は，「付録 B　JAXB 仕様のサポート範囲」を参照してください。

JAXB には，Marshaller および Unmarshaller などを生成する API が含まれています。これらの API は JAXB の実装に依存しません。また，JAXB にはスキーマコンパイラおよびスキーマジェネレータが含まれています。これらが生成する Java クラスやスキーマ文書は JAXB の実装に依存しません。このため，XML 文書を処理・操作するプログラムを作成する際に，使用する XML プロセッサの実装を意識する必要がありません。したがって，プラットフォーム間で共通のプログラムを作成できます。

2.4 JAXB が規定するパッケージとその機能

JAXB は，機能を利用するためのパッケージを規定しています。JAXB が規定するパッケージに含まれる API の概要を次の表に示します。

表 2-3 JAXB が規定するパッケージに含まれる API の概要

パッケージの名称	パッケージに含まれる API の概要
javax.xml.bind	マーシャル，アンマーシャルなどの実行時バインディングフレームワークのための API。
javax.xml.bind.annotation	Java クラスからスキーマ文書へのマッピングをカスタマイズするための API。
javax.xml.bind.annotation.adapters	任意の Java クラスを JAXB で使う場合に使用する，アダプタークラスのための API。
javax.xml.bind.attachment	MIME 形式のバイナリデータのマーシャル，アンマーシャルのための API。
javax.xml.bind.util	JAXB のユーティリティ用 API。

パッケージの詳細については，JSR 222 The Java Architecture for XML Binding 2.2 の Javadoc を参照してください。

2.5 JAXB のコマンド

JSR 222 では，スキーマコンパイラおよびスキーマジェネレータの具体的な使用方法を規定していません。Cosminexus XML Processor では，次に示す方法でスキーマコンパイラおよびスキーマジェネレータを使用できます。

2.5.1 コマンド

スキーマコンパイラ，スキーマジェネレータのコマンドラインインタフェースとして Cosminexus XML Processor が提供するコマンドを次の表に示します。

表 2-4 提供コマンド一覧

コマンドのファイル名		機能
Windows	UNIX	
csmxjc.bat	csmxjc	XML Schema から Java へバインディングするスキーマコンパイラです。
csmschemagen.bat	csmschemagen	Java から XML Schema へマッピングするスキーマジェネレータです。

注意事項

uCosminexus Client では，上記のコマンドを使用できません。

(1) csmxjc コマンド（XML Schema から Java へバインディングする）

(a) 形式

```
csmxjc ［オプション ［オプション引数］］ ... 入力スキーマ文書
```

オプションは省略できます。また，オプションは必ずスキーマ文書より前に指定してください。

(b) 機能

XML Schema から Java へバインディングするスキーマコンパイラです。

(c) オプション

-b < 外部バインディングファイル >

外部バインディングファイルを一つ指定します。外部バインディングファイル名の指定方法については，「2.5.2 コマンドに指定するファイル名・ディレクトリ名の規則」を参照してください。

省略した場合は，外部バインディングファイルを使用しないで java クラスを生成します。

-d < **出力先ディレクトリ** >

Java ソースの出力先ディレクトリを指定します。ディレクトリ名の指定方法については，「2.5.2　コマンドに指定するファイル名・ディレクトリ名の規則」を参照してください。

省略した場合は，カレントディレクトリが出力先ディレクトリになります。

-mark-generated

生成した Java ソースに @javax.annotation.Generated アノテーションを追加します。

省略した場合は，Java ソースに @javax.annotation.Generated アノテーションを追加しません。

Java ソースに追加される @javax.annotation.Generated アノテーションの書式を次に示します。斜体の文字には，Java ソースを生成するたびに異なる値が入ります。

```
@Generated(value = "com.cosminexus.jaxb.tools.xjc.Driver", date = "yyyy-MM-ddThh:mm:ssRFC822タイムゾーン")
```

@javax.annotation.Generated アノテーションの要素と XML Processor で設定される値を次の表に示します。

表 2-5　@javax.annotation.Generated アノテーションの要素と XML Processor で設定される値

要素名	説明	値
value	スキーマコンパイラの完全修飾クラス名	com.cosminexus.jaxb.tools.xjc.Driver
date	スキーマから Java ソースを生成した日時	*yyyy-MM-dd*T*hh:mm:ssRFC822 タイムゾーン*※

注※

出力フォーマットの詳細については java.text.SimpleDateFormat クラスの「日付 / 時刻パターン」を参照してください。

(d) 入力スキーマ文書

入力スキーマ文書のファイル名を一つ指定できます。ファイル名に指定できる文字，パス区切り文字，パスの指定方法については「2.5.2　コマンドに指定するファイル名・ディレクトリ名の規則」を参照してください。

(e) 出力 Java ソース

出力 Java ソースのフォーマット

スキーマコンパイラが出力する Java ソースのフォーマットの概要を次の図に示します。

図 2-10 スキーマコンパイラが出力する Java ソースのフォーマットの概要

```
//                                                                                    ┐
// This file was generated by Cosminexus XML Processor 08-50                          ├ 1.
// Any modifications to this file will be lost upon recompilation of the source schema.
//                                                                                    ┘

package xmlprocessor.hitachi.com;                                                     ┐
                                                                                      ├ 2.
import javax.xml.bind.annotation.XmlAccessType;                                       ┤
import javax.xml.bind.annotation.XmlAccessorType;                                     ┤
import javax.xml.bind.annotation.XmlElement;                                          ┤
import javax.xml.bind.annotation.XmlRootElement;                                      ┤
import javax.xml.bind.annotation.XmlType;                                             ┘

/**                                                                                   ┐
 * <p>Java class for anonymous complex type.                                          ├ 3.
 *                                                                                    ┤
 * <p>The following schema fragment specifies the expected content contained within this class.
        :                                                                             ┤
 * <pre>                                                                              ┤
 * &lt;complexType>                                                                   ┤
 *                                                                                    ┤
 */                                                                                   ┘
@XmlAccessorType(XmlAccessType.FIELD)                                                 ┐
@XmlType(name = "", propOrder = {                                                     ├ 4.
    "child"                                                                           ┤
})                                                                                    ┤
@XmlRootElement(name = "root")                                                        ┤
@Generated(value = "com.cosminexus.jaxb.tools.xjc.Driver", date = "2008-08-14T03:48:11+09:00")   ― 5.
public class Root {                                                                   ┤
                                                                                      ┤
    @XmlElement(required = true)                                                      ┤
@Generated(value = "com.cosminexus.jaxb.tools.xjc.Driver", date = "2008-08-14T03:48:11+09:00")   ― 5.
    protected String child;                                                           ┤
        :                                                                             ┤
        :                                                                             ┘
```

1. ヘッダ部
2. package文，import文記述部
3. javadoc部
4. Javaクラス本体
5. -mark-generatedオプション指定時に付加されるコード。生成したクラス，フィールド，JavaBeanプロパティごとに付加される。

出力 Java ソースのフォーマットの詳細を次に示します。

1. ヘッダ部

   Cosminexus XML Processor で生成されたことを示すヘッダです。次のヘッダが挿入されます。

```
//
// This file was generated by Cosminexus XML Processor 08-00
// Any modifications to this file will be lost upon recompilation of the source
schema.
//
```

2. package 文，import 文記述部

   ソースに必要な package 文や import 文が生成されます。生成される package

文，import 文の内容はスキーマコンパイラに入力されたスキーマ文書に依存します。

3. javadoc 部
   入力スキーマ文書から生成したクラスについて javadoc の内容が自動生成されます。javadoc の内容はスキーマコンパイラに入力されたスキーマ文書に依存します。
4. Java クラス本体
   入力スキーマ文書から生成したクラスが生成されます。Java クラス本体の内容はスキーマコンパイラに入力されたスキーマ文書に依存します。

出力 Java ソースのパッケージと出力先

出力 Java ソースのパッケージ名と出力先について次に示します。
出力 Java ソースのパッケージ名は次の優先順位で決定されます。Java ソースの出力先はカレントディレクトリ，または -d オプションで指定されたディレクトリを基準ディレクトリとして，次の規則を順番に適用して決定されます。

1. カスタムバインディングによるパッケージ名の指定がある場合
   カスタムバインディング（jaxb：package 要素）で指定されたパッケージ名が出力 Java ソースのパッケージ名になります。パッケージ名をディレクトリ名に変換し，基準ディレクトリの下に作成します。作成したディレクトリに Java ソースを出力します。
2. 入力スキーマ文書に targetNamespace 属性が指定されている場合
   入力スキーマ文書に記述された対象名前空間から，JAXB 仕様書に基づき出力 Java ソースのパッケージ名を生成します。生成したパッケージ名をディレクトリ名に変換し，基準ディレクトリの下に作成します。作成したディレクトリに Java ソースを出力します。
3. 入力スキーマ文書にカスタムバインディングによるパッケージ名の指定がなく，targetNamespace 属性の指定もない場合
   Java ソースのパッケージ名は "generated" になります。基準ディレクトリの下に "generated" という名称のディレクトリを作成し，作成したディレクトリに Java ソースを出力します。

出力先のディレクトリに同名の Java ソースが存在する場合は上書きされます。

(f) 戻り値

0：
　正常終了しました。

0 以外の値：
　異常終了しました。

(2) csmschemagen コマンド（Java から XML Schema へマッピングする）

(a) 形式

```
csmschemagen [ オプション [ オプション引数 ] ] [package-info.java ファイル名 ] Java ソースファイル名
```

オプションと package-info.java ファイルは省略できます。また，オプションと package-info.java は必ず Java ソースファイル名より前に指定してください。

(b) 機能

Java から XML Schema へマッピングするスキーマジェネレータです。

(c) オプション

-d < 出力先ディレクトリ >

出力スキーマ文書と中間ファイルの出力先ディレクトリを指定します。出力先ディレクトリ名の指定方法については，「2.5.2　コマンドに指定するファイル名・ディレクトリ名の規則」を参照してください。
省略した場合は，カレントディレクトリが出力先ディレクトリになります。

-encoding < 文字エンコーディング >

入力する Java ソースの文字エンコーディングを指定します。
Cosminexus XML Processor で指定できる文字エンコーディングについては，「1.3.2　処理できる文字コード」を参照してください。
省略した場合の動作は，使用している OS によって次のように異なります。

Windows の場合

OS にデフォルトで設定されている文字エンコーディングが適用されます。

UNIX の場合

環境変数 LANG に指定されている文字エンコーディングが適用されます。環境変数 LANG も指定されていない場合は，OS にデフォルトで設定されている文字エンコーディングが適用されます。

(d) 入力 Java ソース

Java ソースファイルを一つ指定できます。ファイル名に指定できる文字，パス区切り文字，パスの指定方法については「2.5.2　コマンドに指定するファイル名・ディレクトリ名の規則」を参照してください。

(e) 出力スキーマ文書

カレントディレクトリまたは -d オプションで指定されたディレクトリに schema*n*.xsd（*n* は数字）というファイル名で出力されます。出力先のディレクトリに同名のファイルが存在する場合は上書きされます。

(f) 中間ファイル

- csmschemagen コマンドは，Java ソースファイルをコンパイルして，中間ファイルである .class ファイルを生成します。
- Java ソースファイルが package 文を含む場合は，出力先ディレクトリの下にサブディレクトリを作成して .class ファイルを出力します。サブディレクトリ名は，package 文に指定したパッケージ名を基に決定します。
- Java ソースファイルが package 文を含まない場合は，出力先ディレクトリの直下に .class ファイルを出力します。
- .class ファイルの出力先ディレクトリは，-d オプションで変更できます。

(g) 戻り値

0：
　正常終了しました。

0 以外の値：
　異常終了しました。

(h) 注意事項

csmschemagen コマンドに入力する Java ソースファイルが構文的に不正でも，スキーマ文書が出力されます。このスキーマ文書は構文的に不正な可能性があるため，使用しないでください。

2.5.2 コマンドに指定するファイル名・ディレクトリ名の規則

コマンドに指定するファイル名・ディレクトリ名の規則について次に示します。

(1) コマンドのファイル名・ディレクトリ名に指定できる文字，パス区切り文字，パスの指定方法

- コマンドのファイル名・ディレクトリ名に指定できる文字は英数字および次に示す記号です。
  @ & = + $, - _ . ! ~ ' ( )
- パス区切り文字として使用できる文字を次に示します。
  ¥（Windows の場合）
  /（UNIX の場合）
- パスの指定方法は，絶対パスと相対パスのどちらも指定できます。

3 Cosminexus XML Processor の拡張機能

この章では，Cosminexus XML Processor の拡張機能について説明します。

3.1　Shift_JIS 切り替え機能

Shift_JIS 切り替え機能は，XML 文書の encoding 属性に Shift_JIS が指定されている場合に，Cosminexus XML Processor が適用する文字エンコーディングを SJIS（x-sjis-jdk1.1.7）または MS932（x-sjis-cp932）に切り替える機能です。XML 文書の encoding 指定と適用される文字エンコーディングとの対応を次の表に示します。

表 3-1　XML 文書の encoding 指定と適用される文字エンコーディングとの対応

XML 文書の encoding 指定	Shift_JIS 切り替え機能の指定	適用される文字エンコーディング
Shift_JIS	指定なし	SJIS（デフォルト）
	SJIS	SJIS
	MS932	MS932
Windows-31J	―	MS932

（凡例）

―：指定できません。

Shift_JIS 切り替え機能の指定は，XML パーサや XSLT トランスフォーマのインスタンスに特定のプロパティを設定することで行います。プロパティの設定方法については「4.5　Shift_JIS 切り替え機能のプロパティの使用方法」を参照してください。

Shift_JIS 切り替え機能によるエンコーディング指定は，次のものに対して適用されます。それ以外には適用されません。

- 入力 XML 文書の encoding 属性
- InputSource オブジェクトに設定されたエンコーディング
- スタイルシート内の xsl:output 要素の encoding 属性に "Shift_JIS" を指定した場合，または，javax.xml.transform.Transformer クラスの setOutputProperty，setOutputProperties メソッドで "Shift_JIS" エンコーディングを指定した場合の，出力文字ストリーム

ただし，次の場合は注意が必要です。

1. DOM パーサ，SAX パーサの入力ソースに InputSource オブジェクトを指定する場合
   InputSource オブジェクトに Shift_JIS 以外のエンコーディングを設定したときには，Shift_JIS 切り替え機能は無効になります。また，InputSource オブジェクトに Reader が設定されているときにも，Shift_JIS 切り替え機能は無効になり，Reader のエンコーディングが適用されます。
2. XSLT または XSLTC トランスフォーマの入力ソースに StreamSource オブジェクトを指定する場合
   StreamSource オブジェクトに Reader が設定されているときには，Shift_JIS 切り替

え機能は無効になり，Reader のエンコーディングが適用されます。

3. XSLT または XSLTC トランスフォーマの入力ソースに DOMSource オブジェクトを指定する場合
   DOMSource が生成された時点で XML 文書は文字コード変換済みなので，Shift_JIS 切り替え機能は無効になります。
   Shift_JIS 切り替え機能を有効にするためには，DOMSource に設定する DOM ノードの生成時に利用するパーサに対して，あらかじめ Shift_JIS 切り替え機能を指定しておく必要があります。
4. XSLT または XSLTC トランスフォーマの入力ソースに SAXSource オブジェクトを指定する場合
   SAXSource オブジェクトに InputSource オブジェクトを設定したときには，1. と同じように注意が必要です。また，SAXSource オブジェクトに XMLReader オブジェクトを設定したときには，XMLReader に指定された Shift_JIS 切り替え機能が有効になります。
5. Shift_JIS 切り替え機能を指定した DOM パーサ，または SAX パーサによって作成された Node，Source，Result などを JAXB の API の引数として渡した場合
   JAXB は，Shift_JIS 切り替え機能をサポートしていません。
6. StAX は Shift_JIS 切り替え機能をサポートしていません。そのため，XSLT または XSLTC トランスフォーマの入出力に，次のどちらかを使用した場合の動作は保証しません。
   - StAXSource
   - StAXResult

3.2 XSLTC トランスフォーマ機能

この節では，XSLTC トランスフォーマ機能について説明します。

3.2.1 XSLTC トランスフォーマの概要

XSLTC トランスフォーマは，Cosminexus XML Processor が提供している XSLT トランスフォーマと同じ機能を持ちながら，XML 文書の変換処理の性能を向上させたものです。

XSLTC トランスフォーマは，XSLT スタイルシートを解析するコンパイラと，変換処理を行う実行時プロセッサで構成されます。コンパイラは，XSLT スタイルシートを解析して，高速に変換するためのトランスレットと呼ばれる Java のバイトコードをメモリ上に生成します。実行時プロセッサは，トランスレットを実行することによって，高速な変換処理を実現します。

XSLTC トランスフォーマと XSLT トランスフォーマは，次に示す点で動作に差異がありますのでご注意ください。

1. XSLTC トランスフォーマは XSLT トランスフォーマよりもエラーチェックが簡略化されています。
2. XSLT1.0 仕様書で規定されていない部分で，XSLT トランスフォーマと XSLTC トランスフォーマの動作に差異がある場合があります。

XSLTC トランスフォーマを使用することで，XSLT トランスフォーマよりも高い変換処理性能を期待できるケースを次に示します。

(1) 同一の XSLT スタイルシートを使って複数の XML 文書を変換する場合

同一の XSLT スタイルシートを使って複数の XML 文書を変換する場合は，XSLT トランスフォーマ，XSLTC トランスフォーマの両方とも，次に示す 1. の方法よりも 2. の方法の方が高い変換性能を期待できます。

1. スタイルシートから Templates オブジェクトや Transformer オブジェクトを毎回生成し，これを用いて変換する。
2. スタイルシートから Templates オブジェクトや Transformer オブジェクトを一度だけ生成し，これを再利用して変換する。

XSLTC トランスフォーマで 2. の方法を用いると，時間の掛かるトランスレット作成処理は一度だけで済みます。その後，トランスレットを用いた高速な変換が繰り返し実行されるため，XSLT トランスフォーマで 2. の方法を用いるよりも高い性能が期待できます。

3.2.2 XSLTC トランスフォーマを使ったシステムの開発・運用

XSLTC トランスフォーマは，XSLT トランスフォーマと比べてエラーチェックが厳密ではありません。XSLTC トランスフォーマを使ったシステムを開発・運用する場合は，次に示すように XSLT トランスフォーマでの動作確認も実施してください。

1. XSLT トランスフォーマを使ってスタイルシートのエラーチェックを必ず実施してください。これによって，XSLTC トランスフォーマではチェックできないエラーをチェックすることができ，他システムとの相互運用などで発生するおそれがあるエラーチェックの相違を事前にチェックできます。
2. XSLTC トランスフォーマによる変換結果が XSLT トランスフォーマのものと同じになることを確認することを推奨します。これによって，トランスフォーマ固有の動作が変換結果に影響を及ぼすかどうかを確認でき，他システムとの相互運用などで発生するおそれがある動作の差異を事前にチェックできます。

XSLT トランスフォーマと XSLTC トランスフォーマに関する次の注意事項も参照してください。

- 「6.7　XSLT/XSLTC 共通の注意事項」
- 「6.8　XSLT に関する注意事項」
- 「6.9　XSLTC に関する注意事項」

3.2.3 XSLTC トランスフォーマで使用するクラス

XSLTC トランスフォーマでは，TransformerFactoryXSLTC クラスを使用します。

XSLTC トランスフォーマで使用するクラスについて説明します。

(1) TransformerFactoryXSLTC クラス

(a) 説明

XSLTC トランスフォーマのための TransformerFactory インスタンスを作成するクラスです。

(b) パッケージ名およびクラス名

com.cosminexus.jaxp.xsltc.TransformerFactoryXSLTC

(c) 形式

```
public class TransformerFactoryXSLTC
```

(d) メソッド一覧

メソッド名	機能
newInstance	TransformerFactory の新しいインスタンスを作成する

(e) メソッド詳細

newInstance メソッド

説明

XSLTC トランスフォーマのための，TransformerFactory の新しいインスタンスを作成します。

形式

```
public static TransformerFactory newInstance()
```

パラメタ

なし

戻り値

TransformerFactory の新しいインスタンスを返却します。

例外

なし

3.2.4 XSLTC トランスフォーマの使用方法

XSLTC トランスフォーマを使用する場合，XSLT トランスフォーマとは TransformerFactory インスタンスの生成方法が異なります。XSLTC トランスフォーマでの TransformerFactory インスタンスの生成方法を次に示します。

```
javax.xml.transform.TransformerFactory factory =
    com.cosminexus.jaxp.xsltc.TransformerFactoryXSLTC.newInstance();
```

その他の使用方法は，XSLT トランスフォーマと同じです。

コード例を次に示します。

```
import javax.xml.transform.*;
import javax.xml.transform.stream.*;
import com.cosminexus.jaxp.xsltc.*;
                        :
// XSLTスタイルシートを設定
StreamSource style = new StreamSource("style.xsl");
// TransformerFactoryインスタンスを生成
TransformerFactory factory = TransformerFactoryXSLTC.newInstance();
// Templatesオブジェクトを生成
Templates templates = factory.newTemplates(style);
// XSLTCトランスフォーマを生成
Transformer transformer = templates.newTransformer();
                        :
// 同一のXSLTスタイルシートを使って複数のXML文書を変換
for (int i = 0; i < 10; i++) {
  // 入力元を設定
  StreamSource source = new StreamSource(args[i]);
  // 出力先を設定
  StreamResult result = new StreamResult("output_"+ args[i]);
  // 変換実行
  transformer.transform(source, result);
}
```

3.3 スキーマキャッシュ機能

この節では，スキーマキャッシュ機能について説明します。

3.3.1 スキーマキャッシュ機能の概要

スキーマキャッシュ機能とは，XML パーサがスキーマ文書を解析して取得できる文法定義をあらかじめオブジェクトの状態でキャッシュし，スキーマ検証時に再利用することで検証パースの実行速度を向上させる機能です。

スキーマキャッシュ機能では，メモリ上およびディスク上にキャッシュを作成できます。また，JAXP1.2 の API を使ったプログラムでも，ソースを改造することなく検証パースの実行速度を向上させることができます。

スキーマキャッシュ機能を使用する場合の前提条件，および適用範囲について説明します。

(1) 前提条件

スキーマキャッシュ機能は，J2EE サーバ上で動作するユーザプログラムが javax.xml.parsers.DocumentBuilder クラス（DOM パーサ）を使って XML 文書をパースする場合だけ有効となります。

スキーマキャッシュ機能を使用する場合の制約を次に示します。

- アプリケーションサーバをサーブレットエンジンモードで使用する場合，スキーマキャッシュ機能の動作は保証しません。
- javax.xml.validation パッケージを使ったスキーマ検証では，スキーマキャッシュ機能が適用されません。
- parse メソッドの引数に File オブジェクトを渡した場合，パーサが内部で保持するスキーマ文書ファイル名の一部が，キャッシュ構築時のファイル名と一致しなくなります。そのため，スキーマキャッシュ機能が使用できません。

(2) 適用範囲

次に示すスキーマ文書がキャッシュの対象となります。

- スキーマ定義ファイルで指定されたスキーマ文書
- スキーマ定義ファイルで指定されたスキーマ文書から，次の要素によって直接または間接的に参照されるスキーマ文書
  - import 要素
  - include 要素
  - redefine 要素

なお，一つのマシン上で複数の J2EE サーバが起動された場合，スキーマキャッシュ機能はサーバごとに独立して動作します。

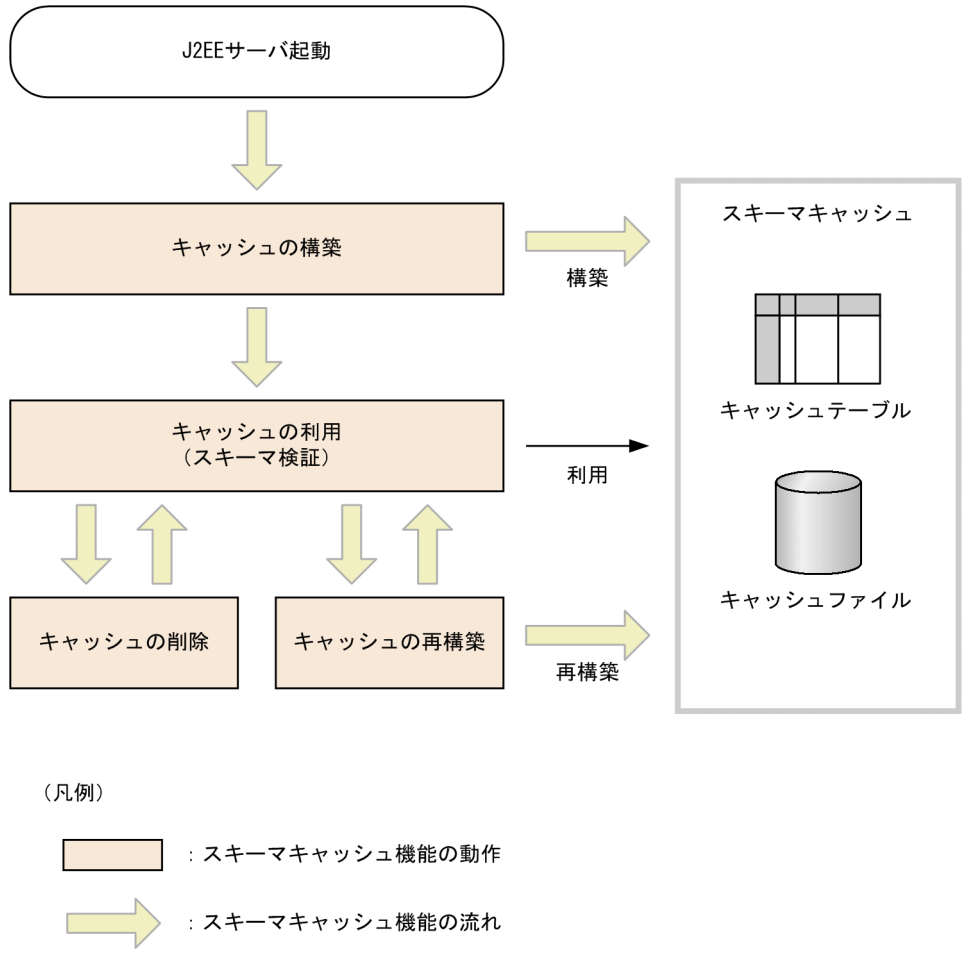
3.3.2　スキーマキャッシュ機能の処理の流れ

スキーマキャッシュ機能には，スキーマキャッシュを操作する次の機能があります。

- スキーマキャッシュの構築
- スキーマキャッシュの利用
- スキーマキャッシュの削除および再構築

スキーマキャッシュ機能の処理の流れを次に示します。

図 3-1　スキーマキャッシュ機能の処理の流れ

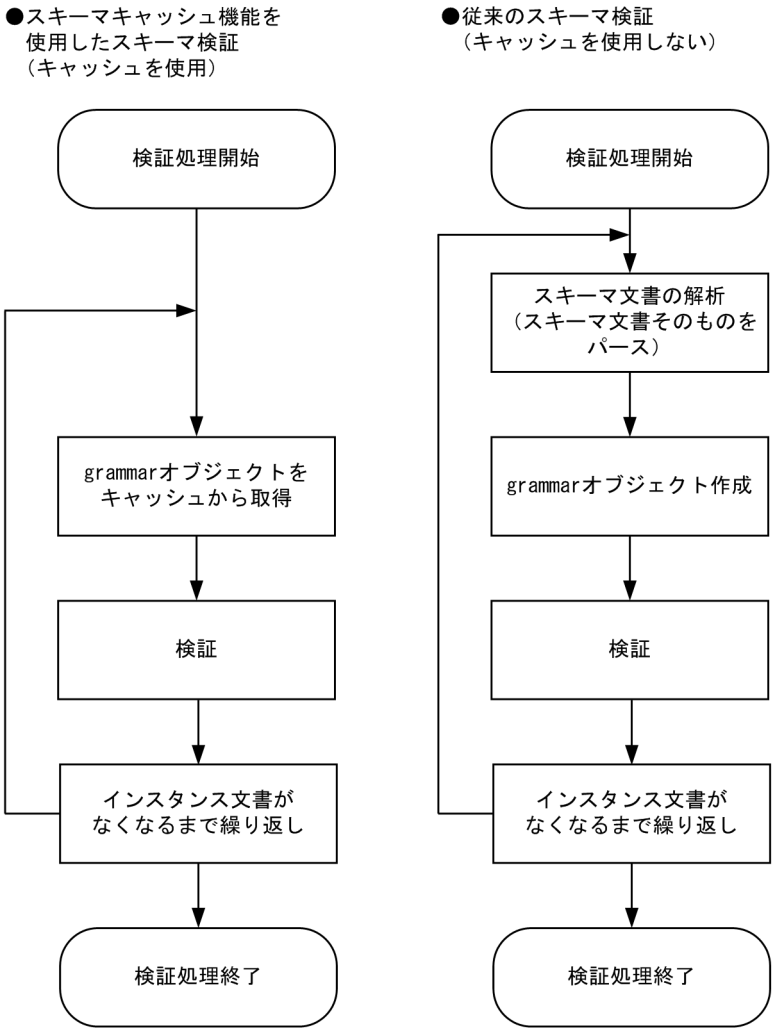


従来のスキーマ検証は，同じスキーマを使う場合でも，検証のたびにスキーマ文書そのものをパースして grammar オブジェクトを作成します。一方，スキーマキャッシュ機能を使用すると，あらかじめ作成された grammar オブジェクトを再利用するため，スキーマ文書解析のオーバーヘッドを軽減できます。

一つのスキーマ文書を使用して複数のインスタンス文書をスキーマ検証する場合の，

キャッシュを利用するスキーマキャッシュ機能を使用したスキーマ検証と，キャッシュを利用しない従来のスキーマ検証の処理の例を次に示します。

図 3-2 スキーマキャッシュ機能を使用したスキーマ検証と従来のスキーマ検証の処理の違い



3.3.3 スキーマキャッシュの構築

スキーマキャッシュ機能を使用するためには，あらかじめキャッシュ対象となるスキーマ文書を決めて，スキーマキャッシュを構築する必要があります。そのため，次のプロパティファイルを適切に設定してください。

- スキーマ定義ファイル
- J2EE サーバのユーザ定義ファイル（usrconf.properties）

上記のファイルは Java のプロパティファイルに相当するため，基本的な構文規則は Java のプロパティファイルの規則に従ってください。なお，プロパティファイルの構文規則の詳細については，java.util.Properties クラスの Javadoc を参照してください。

プロパティファイルの詳細について説明します。

(1) スキーマ定義ファイル

スキーマ定義ファイルには，次の項目を定義できます。

- キャッシュ対象となるスキーマ文書
- キャッシュの形式
- スキーマキャッシュを格納するディスク上の場所

スキーマ定義ファイルのファイル名，および格納場所は任意です。

キャッシュの対象となるのは，次のスキーマ文書です。

- スキーマ定義ファイルに直接指定されたスキーマ文書
- スキーマ定義ファイルで直接指定されたスキーマ文書から，次の要素によって直接または間接的に参照されるスキーマ文書
  - import 要素
  - include 要素
  - redefine 要素

スキーマ定義ファイルに記述できるプロパティを次に示します。次に示す以外のプロパティを指定した場合，無視されます。

表 3-2 スキーマ定義ファイルに記述できるプロパティ

項番	プロパティ名	指定値
1	schema_path	スキーマ文書が存在するディレクトリを示す絶対パスを指定します。
2	diskcache_path	ディスクキャッシュを格納するディレクトリを示す絶対パスを指定します。
3	schema_xxx ("xxx" の部分は 1 文字以上の任意の文字列)	キャッシュ対象となるスキーマ文書のファイルを示す相対パス，およびディスクキャッシュ指定子を指定します。ディスクキャッシュ指定子は省略できます。 ディスクキャッシュ指定子は，スキーマ文書の相対パスの末尾に ",D" と記述してください。ディスクキャッシュ指定子が記述されたスキーマ文書のキャッシュは，ディスク上に作成されます。 ディスクキャッシュ指定子の記述がない場合，キャッシュはメモリ上に作成されます。 ファイル名とディスクキャッシュ指定子の間の連続する空白およびタブは無視されます。

それぞれのプロパティで指定するパスの区切り記号は，Windows の場合も UNIX の場合もスラッシュ（/）です。ディレクトリ名の絶対パスの末尾のスラッシュ（/）は任意で

す。

同じプロパティ名が 2 回以上指定された場合，最後に指定されたプロパティの指定値が有効となります。

スキーマ定義ファイルの記述例を次に示します。

Windows の場合

```
shema_path=C:/usr/app/xmlparser/schemafiles
diskcache_path=C:/usr/app/xmlparser
schema_01=schema/type1.xsd
schema_02=schema/type2.xsd,D
```

UNIX の場合

```
shema_path=/usr/app/xmlparser/schemafiles
diskcache_path=/usr/app/xmlparser
schema_01=schema/type1.xsd
schema_02=schema/type2.xsd,D
```

"schema_01=schema/type1.xsd" と指定した場合，ディスクキャッシュ指定子が記述されていないため，スキーマ文書のキャッシュはメモリ上にキャッシュされます。

"schema_02=schema/type2.xsd,D" と指定した場合，ディスクキャッシュ指定子が記述されているめ，スキーマ文書のキャッシュはディスク上にキャッシュされます。

スキーマ定義ファイル指定時の動作を次に示します。

- schema_path プロパティ，および diskcache_path プロパティは，必ず指定してください。
  これらのプロパティのどちらか一方でも指定していない場合，必要なプロパティ指定が欠けているという警告メッセージ（KECX09503-W）がエラーファイルに出力され，スキーマキャッシュを使用しないでスキーマ検証を実施します。
- 複数の J2EE サーバで diskcache_path プロパティに同じディレクトリが指定されている場合でも，ディスクキャッシュはサーバごとに独立して管理されます。
  diskcache_path プロパティで指定されたディレクトリの下に J2EE サーバ名と同じ名前のディレクトリが作成され，その下にキャッシュファイルが作成されます。ディスクキャッシュが削除される場合は，作成されたサブディレクトリごとにキャッシュファイルが削除されます。
- diskcache_path プロパティの設定値が絶対パスでない場合，エラーファイルに警告メッセージ（KECX09507-W）が出力され，スキーマキャッシュを使用しないでスキーマ検証を実施します。
- diskcache_path プロパティに設定されたディレクトリにアクセスできない場合，ディスクキャッシュ格納用のディレクトリにアクセスできないという警告メッセージ（KECX09505-W）がエラーファイルに出力され，スキーマキャッシュを使用しないでスキーマ検証を実施します。
- schema_path プロパティの設定値が絶対パスでない場合，エラーファイルに警告メッ

セージ（KECX09508-W）が出力され，スキーマキャッシュを使用しないでスキーマ検証を実施します。

- schema_xxx プロパティの設定値が相対パスでない場合，エラーファイルに警告メッセージ（KECX09509-W）が出力され，その schema_xxx プロパティの指定は無効となります。
- ディスクキャッシュ作成時に grammar オブジェクトのファイルへの書き込みが失敗した場合，エラーファイルに警告メッセージ（KECX09511-W）を出力します。パーサは，書き込みに失敗したキャッシュではなく，スキーマ文書を使用してスキーマ検証を実施します。
- 複数の異なる schema_xxx プロパティに同じスキーマ文書が指定された場合，スキーマ文書に対応するキャッシュが一つだけ作成されます。このとき，これらのプロパティのうち，どれかのプロパティにディスクキャッシュ指定子が指定されていれば，キャッシュはディスク上に作成されます。どのプロパティにもディスクキャッシュ指定子が指定されていない場合，キャッシュはメモリ上に作成されます。

スキーマ定義ファイルに関する注意事項については，「6.19.3　スキーマ定義ファイルに関する注意事項」を参照してください。

（2） J2EE サーバのユーザ定義ファイル

J2EE サーバのユーザ定義ファイル（usrconf.properties）は，J2EE サーバ起動時に初期化するクラスや，設定するシステムプロパティを定義するファイルです。

J2EE サーバのユーザ定義ファイルの詳細についてはマニュアル「Cosminexus アプリケーションサーバ リファレンス 定義編（サーバ定義）」の「2.　J2EE サーバで使用するファイル」を参照してください。

スキーマキャッシュ機能を使用する場合に，J2EE サーバのユーザ定義ファイルに記述するプロパティを次に示します。

表 3-3　J2EE サーバのユーザ定義ファイルに記述するプロパティ

項番	プロパティ名	指定値
1	ejbserver.application.InitTermProcessClasses	J2EE 拡張コンテナの初期化・終了クラスを指定します。 スキーマキャッシュ機能では，スキーマキャッシュの構築に必要な "com.cosminexus.jaxp.impl.parsers.util.XMLGrammarCaching" を指定してください。

項番	プロパティ名	指定値
2	com.cosminexus.jaxp.grammar_caching.preload	スキーマキャッシュ機能を使用するかどうかを指定します。"ON" または "OFF" を指定できます。デフォルトは "OFF" です。 "ON" を設定すると，スキーマキャッシュ機能を使用できます。XML パーサはスキーマキャッシュを使用してスキーマ検証を実施します。 "OFF" を設定すると，スキーマキャッシュ機能を使用しません。XML パーサはスキーマキャッシュを使用しないでスキーマ検証を実施します。 なお，このプロパティは，スキーマキャッシュ機能全体を制御するプロパティです。J2EE サーバがスキーマキャッシュ機能を利用する場合，必ず "ON" を指定してください。"OFF" を指定した場合，J2EE サーバ起動後にスキーマキャッシュ機能を有効にすることはできません。
3	com.cosminexus.jaxp.grammar_caching.config	スキーマ定義ファイルの絶対パスを指定します。

スキーマキャッシュ機能は，次のすべての条件に該当する場合だけ，使用できます。

- ejbserver.application.InitTermProcessClasses プロパティを指定。
- com.cosminexus.jaxp.grammar_caching.preload プロパティに "ON" を指定。
- com.cosminexus.jaxp.grammar_caching.config プロパティを指定。

J2EE サーバのユーザ定義ファイルの記述例を次に示します。

```
# Cosminexus起動時に初期化するクラスを指定
ejbserver.application.InitTermProcessClasses=com.cosminexus.jaxp.impl.parsers.
util.XMLGrammarCaching
# キャッシュ機能を使用可能にする
com.cosminexus.jaxp.grammar_caching.preload=ON
# キャッシュに保持するスキーマ文書を定義した「スキーマ定義ファイル」の絶対パス名
com.cosminexus.jaxp.grammar_caching.config=C:/usr/app/xmlparser/
config.properties
```

J2EE サーバのユーザ定義ファイル指定時の動作を次に示します。

- com.cosminexus.jaxp.grammar_caching.preload プロパティの指定値が次の場合，"OFF" が設定されたものとします。
  - "ON" または "OFF" 以外の値が設定されている
  - プロパティが設定されていない
- com.cosminexus.jaxp.grammar_caching.preload プロパティに "ON" が指定されていても，com.cosminexus.jaxp.grammar_caching.config プロパティで指定されたスキーマ定義ファイルにアクセスできない場合，スキーマキャッシュ機能は使用できません（警告メッセージ KECX09504-W）。

3.3.4　スキーマキャッシュの利用

XML パーサを使ってスキーマ検証を実施する場合，スキーマキャッシュ機能を使用するためのユーザ側の操作は不要です。XML パーサはスキーマキャッシュ機能を使用するかどうかを判断し，スキーマキャッシュ機能を使用する場合は，自動的にスキーマキャッシュを使用してスキーマ検証を実施します。スキーマキャッシュを使用しない場合，またはキャッシュが見つからない場合は，従来の XML パーサと同様に grammar オブジェクトを作成してスキーマ検証を実施します。

なお，ディスクキャッシュファイルからの grammar オブジェクトの読み込みに失敗した場合は，警告メッセージ（KECX09512-W）が出力されます。XML パーサは読み込みに失敗したキャッシュは使用しないで，スキーマ文書を使用してスキーマ検証を実施します。

スキーマキャッシュ機能では，キャッシュの制御や状態の確認のために，次のコマンドを使用できます。

表 3-4　スキーマキャッシュ機能のコマンド

項番	コマンドのファイル名		機能
	Windows	UNIX	
1	cacheoff.bat	cacheoff	スキーマキャッシュ機能を無効にして，キャッシュを削除します。
2	cacheon.bat	cacheon	スキーマキャッシュ機能を有効にして，キャッシュを再構築します。
3	cachestate.bat	cachestate	スキーマキャッシュ機能の状態を表示します。

コマンドのファイルは，すべて＜ JAXP_DIR ＞※の下の bin ディレクトリに格納されています。コマンドを使用するためには，＜ JAXP_DIR ＞※の下の bin ディレクトリを PATH 環境変数に設定する必要があります。

注※

Cosminexus をインストールしたディレクトリ下の，jaxp ディレクトリのことです。

コマンドの詳細については，「3.3.5　スキーマキャッシュの削除および再構築」を参照してください。

3.3.5　スキーマキャッシュの削除および再構築

キャッシュ対象のスキーマ文書を削除・変更した場合，コマンドを使用してキャッシュを削除・再構築する必要があります。コマンドを使用することで，J2EE サーバを停止することなくスキーマ文書の変更をキャッシュに反映させることができます。

キャッシュの削除・再構築に関する注意事項については，「6.19.4　キャッシュの再構築・削除に関する注意事項」を参照してください。

キャッシュの削除・再構築時に実行するコマンドについて説明します。

(1) cacheoff コマンド（XML パーサにキャッシュ削除を指示する）

(a) 形式

cacheoff　J2EE サーバ名

(b) cacheoff コマンド実行時の動作

cacheoff コマンドを実行したときの動作を次に示します。なお，スキーマキャッシュ機能が無効の場合，cacheoff コマンドを実行しても，何も動作しません。

- cacheoff コマンドを実行すると，XML パーサはメモリ上のキャッシュをすべて開放し，スキーマ定義ファイルの "diskcache_path" で指定されたディレクトリ以下の，J2EE サーバ名と同じ名前のディレクトリおよびその下に作成されたディスクキャッシュをすべて削除します。
- cacheoff コマンド実行後はスキーマキャッシュ機能が無効となり，XML パーサはキャッシュを使用しないでスキーマ検証を実施します。
- cacheoff コマンドは XML パーサにキャッシュの削除を指示するだけであり，直接キャッシュを削除するわけではありません。cacheoff コマンド実行後，最初に DOM パーサがパースするときにキャッシュが削除されます。
- cacheoff コマンド実行時には「"diskcache_path" で指定されたディレクトリ / サーバ名」のディレクトリを表示して，削除してよいかどうかユーザに確認を求めます。ディスク上にキャッシュが作成されていない場合でも，ディスクキャッシュを格納するためのディレクトリだけは作成されているため，削除してよいかどうかユーザに確認を求めます。
- cacheoff コマンドを実行した場合に，システムの異常などにより内部ディレクトリおよび内部ファイルにアクセスできなかったときは，エラーメッセージ（KECX09506-E）を標準エラー出力に出力し，コマンドの処理を中断します。
- cacheoff コマンドの引数に存在しない J2EE サーバ名が指定された場合，エラーメッセージ（KECX09510-E）が出力されます。
- J2EE サーバのユーザ定義ファイルで，com.cosminexus.jaxp.grammar_caching.preload プロパティに "ON" が指定されていない場合に，その J2EE サーバに対して cacheoff コマンドを実行したときは，エラーメッセージ（KECX09510-E）が出力されます。
- コマンド引数に誤りがあった場合，次のエラーメッセージを出力して処理を中断します。

```
KECX09513-E cacheoff : Invalid parameter.
usage : cacheoff J2EE_server_name
```

(2) cacheon コマンド（XML パーサにキャッシュ再構築を指示する）

(a) 形式

cacheon J2EE サーバ名

(b) cacheon コマンド実行時の動作

cacheon コマンドを実行したときの動作を次に示します。

- cacheon コマンドを実行すると，XML パーサはメモリ上のキャッシュを開放し，スキーマ定義ファイルの "diskcache_path" で指定されたディレクトリ以下の，J2EE サーバ名と同じ名前のディレクトリおよびその下に作成されたディスクキャッシュをすべて削除します。そのあと，スキーマ定義ファイルの内容に従ってキャッシュを再構築します。
- cacheon コマンド実行後はスキーマキャッシュ機能が有効となり，XML パーサはキャッシュを使ってスキーマ検証を実施します。
- cacheon コマンドは XML パーサにキャッシュの再構築を指示するだけであり，直接キャッシュを削除したり再構築したりするわけではありません。cacheon コマンド実行後，最初に XML パーサがパースするときにキャッシュが削除され，再構築されます。
- cacheon コマンドを実行した場合に，システムの異常などにより内部ディレクトリおよび内部ファイルにアクセスできなかったときは，エラーメッセージ（KECX09506-E）を標準エラー出力に出力し，コマンドの処理を中断します。
- cacheon コマンドの引数に存在しない J2EE サーバ名が指定された場合，エラーメッセージ（KECX09510-E）が出力されます。
- J2EE サーバのユーザ定義ファイルで，com.cosminexus.jaxp.grammar_caching.preload プロパティに "ON" が指定されていない場合に，その J2EE サーバに対して cacheon コマンドを実行したときは，エラーメッセージ（KECX09510-E）出力されます。
- コマンド引数に誤りがあった場合，次のエラーメッセージを出力して処理を中断します。

```
KECX09513-E cacheon : Invalid parameter.
usage : cacheon J2EE_server_name
```

(3) cachestate コマンド（キャッシュ機能の状態を表示する）

(a) 形式

cachestate [-d] [J2EE サーバ名]

(b) cachestate コマンド実行時の動作

cachestate コマンドを実行したときの動作を次に示します。

- cachestate コマンドを -d オプションなしで実行すると，スキーマキャッシュ機能の状

態が標準出力に表示されます。表示される内容を次に示します。

表 3-5 表示されるスキーマキャッシュ機能の状態

項番	スキーマキャッシュ機能の状態	表示内容
1	有効	"KECX09501-I J2EE Server 'J2EE サーバ名 ': schema cache: ON"
2	無効	"KECX09501-I J2EE Server 'J2EE サーバ名 ': schema cache: OFF"

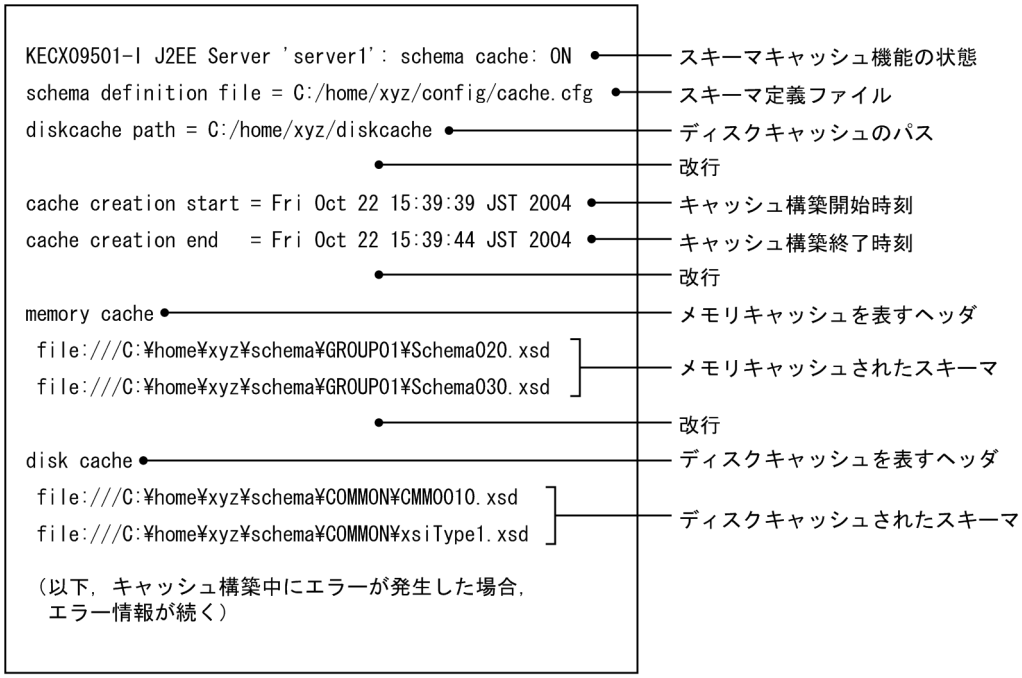
- cachestate コマンドを実行した場合に，システムの異常などにより内部ディレクトリおよび内部ファイルにアクセスできなかったときは，エラーメッセージ（KECX09506-E）を標準エラー出力に出力し，コマンドの処理を中断します。
- cachestate コマンドの引数に存在しない J2EE サーバ名が指定された場合，エラーメッセージ（KECX09510-E）が出力されます。
- cachestate コマンドの引数の J2EE サーバ名が省略された場合，すべての J2EE サーバに関する情報が出力されます。
- J2EE サーバのユーザ定義ファイルで，com.cosminexus.jaxp.grammar_caching.preload プロパティに "ON" が指定されていない場合に，cachestate コマンドに -d オプションを指定して実行しても，何も情報が出力されません。
- コマンド引数に誤りがあった場合，次のエラーメッセージを出力して処理を中断します。

```
KECX09513-E cachestate : Invalid parameter.
usage : cachestate [-d] [J2EE_server_name]
```

- cachestate コマンドに -d オプションを指定して実行すると，次の情報が標準出力に表示されます。
  - スキーマキャッシュ機能の状態（"ON" または "OFF"）
  - 現在有効なスキーマ定義ファイル
  - 現在有効な diskcache_path（ディスクキャッシュが作成されるディレクトリ）
  - キャッシュ構築の開始・終了時刻
  - スキーマ文書のキャッシュ構築状況
  - キャッシュ構築中のエラー情報

  情報の出力フォーマットを次に示します。

図 3-3　cachestate コマンド実行時の出力フォーマット



サーバ名を指定しない場合，サーバの数だけ情報を出力します。途中で KECX09506-E のエラーが発生した場合でも，残りのサーバの情報を出力します。この情報の内容は，キャッシュが再構築されるたびにクリアされ，更新されます。

(4) 各コマンドとパースの同期処理の関係

cacheon コマンド，cacheoff コマンド，および cachestate コマンドとパースの同期処理の関係を次に示します。

表 3-6　各コマンドとパースの同期処理の関係

項番	実行するアクション	システムの状態		
		パース実行中および右記以外	キャッシュ構築中	キャッシュ削除中
1	パース	パースを開始します（すぐに処理を開始します）。	構築が完了したあとパースを開始します。	削除が完了したあとパースを開始します。
2	cacheon コマンド	次の順番で実行します。 1. すぐにコマンドを終了する。ただし，スキーマキャッシュ構築はこの時点では実行されない。 2. 次のパースを開始する。 3. キャッシュを構築する。		

項番	実行するアクション	システムの状態		
		パース実行中および右記以外	キャッシュ構築中	キャッシュ削除中
3	cacheoff コマンド	次の順番で実行します。 1. すぐにコマンドを終了する。ただし，スキーマキャッシュ削除はこの時点では実行されない。 2. 次のパースを開始する。 3. キャッシュを削除する。		
4	cachestate コマンド	次の情報を表示します。 スキーマキャッシュ機能が有効のとき "KECX09501-I J2EE Server 'J2EE サーバ名 ': schema cache: ON" スキーマキャッシュ機能が無効のとき "KECX09501-I J2EE Server 'J2EE サーバ名 ': schema cache: OFF"	"KECX09501-I J2EE Server 'J2EE サーバ名 ': schema cache: OFF" を表示します。	

3.3.6 最適なスキーマキャッシュの使用について

スキーマキャッシュ機能を利用すると，妥当性検証のためのパース時間を短縮することができます。ただし，メモリ使用量が増加したり，ディスクキャッシュ使用時に期待するほど効果が出ないなどの問題が発生するおそれがあります。

ここでは，これらの問題を回避し，最適なスキーマキャッシュを使用するための方法について説明します。

(1) スキーマ文書とインスタンス文書のサイズと検証時間の関係

スキーマ文書は XML で記述されています。妥当性検証に先立ってパースされ，内部的な grammar オブジェクトに変換されます。キャッシュを使用すると，スキーマ文書から grammar オブジェクトを構築する処理を省略できます。妥当性検証のためのパースには，次の時間が掛かります。

インスタンス文書の検証時間＋スキーマ文書から grammar オブジェクトを構築する時間

grammar オブジェクトを構築する時間はキャッシュにより省略できるため，次の条件では，1. に示す条件の方がスキーマキャッシュの効果が大きくなります。

1. スキーマ文書のサイズ＞インスタンス文書のサイズ
2. スキーマ文書のサイズ＜インスタンス文書のサイズ

(2) メモリキャッシュとディスクキャッシュの差

スキーマキャッシュ機能を使用する場合，事前にスキーマ文書から grammar オブジェクトを構築しているため，検証時に grammar オブジェクトを構築する時間を省略できます。

ただし，メモリキャッシュを使用する場合とディスクキャッシュを使用する場合，妥当性検証のためのパースには次に示す時間が掛かります。

- メモリキャッシュ
  妥当性検証のためのパースには，インスタンス文書を検証する時間が掛かります。
- ディスクキャッシュ
  妥当性検証のためのパースには，次の時間が掛かります。
  インスタンス文書の検証時間＋デシリアライズ※時間＋ I/O 時間

  注※
  Java のオブジェクトを丸ごとファイルで保存できるように変換（シリアライズ）したデータを，再び元のオブジェクト形式に復元する処理のことです。

(3) メモリキャッシュ使用時の注意事項

メモリキャッシュを使用する場合，キャッシュ構築後はメモリ上にキャッシュが残ります。そのため，リソースの許す範囲で，メモリ使用量を調整してください。使用頻度の高いスキーマ文書を選択して，優先的にメモリキャッシュを利用すると効果的です。

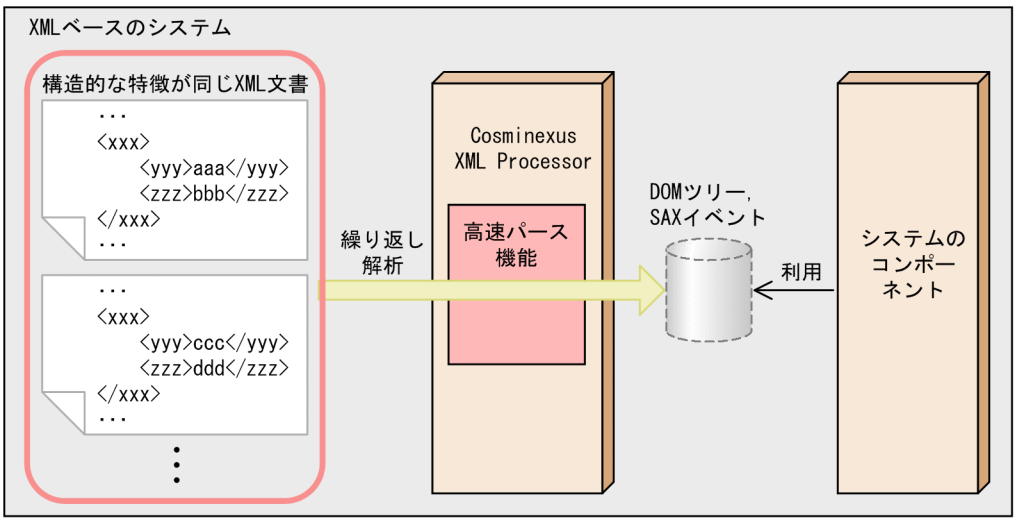
3.4　高速パース機能

この節では，高速パース機能について説明します。

3.4.1　高速パース機能の概要

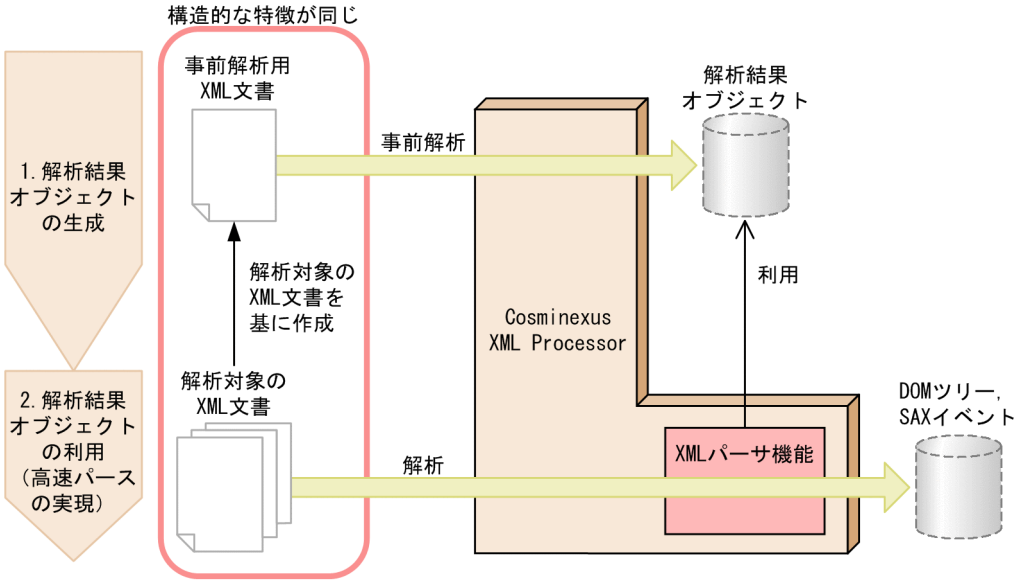
高速パース機能とは，事前解析によって解析対象の XML 文書の特徴を学習し，以降の XML 文書解析時に学習結果を利用することで，解析処理の実行速度を向上させる機能です。この機能は，要素や属性の並び順や入れ子関係などの構造的な特徴が一致している XML 文書を，繰り返し解析するシステムに適しています。高速パース機能を利用したシステムを次の図に示します。

図 3-4　高速パース機能を利用したシステム



事前解析によって学習した XML 文書の特徴は，**解析結果オブジェクト**に記録します。解析結果オブジェクトを生成して，高速パースを実現する流れを次の図に示します。

図 3-5　解析結果オブジェクトを生成して，高速パースを実現する流れ



1. **解析結果オブジェクトの生成**
   解析結果オブジェクトは，Cosminexus XML Processor が，事前解析用 XML 文書から要素の並び順，属性の並び順，要素の繰り返しなどの情報を事前解析して生成します。**事前解析用 XML 文書**とは，解析対象の XML 文書と構造的な特徴が同じ XML 文書です。
   事前解析用 XML 文書は，ユーザが作成する必要があります。XML 文書は一般的に自由な構造を持つため，すべての XML 文書に対応できる解析結果オブジェクトは生成できません。このため，解析対象の XML 文書を考慮して，出現頻度の高い文書構造に的を絞って事前解析用 XML 文書を作成します。

2. **解析結果オブジェクトの利用（高速パースの実現）**
   解析結果オブジェクトを設定した Cosminexus XML Processor の XML パーサ機能で解析を実行すると，解析結果オブジェクトを利用して XML 文書を解析します。解析対象の XML 文書の構造が，解析結果オブジェクトが記録している文書の構造と一致していれば，解析速度が向上します。

図 3-5 に示した流れで高速パース機能を使用する場合，ユーザプログラム中に，解析結果オブジェクトを生成して XML パーサに設定するためのコードを記述する必要があります。ユーザプログラム中にコードを記述して高速パース機能を使用する方法の詳細については，「3.4.2　高速パースのための作業の流れ」を参照してください。

ただし，すべての場合について，通常よりも解析速度を向上できることを保証するものではありません。このため，高速パース機能を使用する場合は，実際に解析対象の XML 文書を使用して性能を評価することを推奨します。

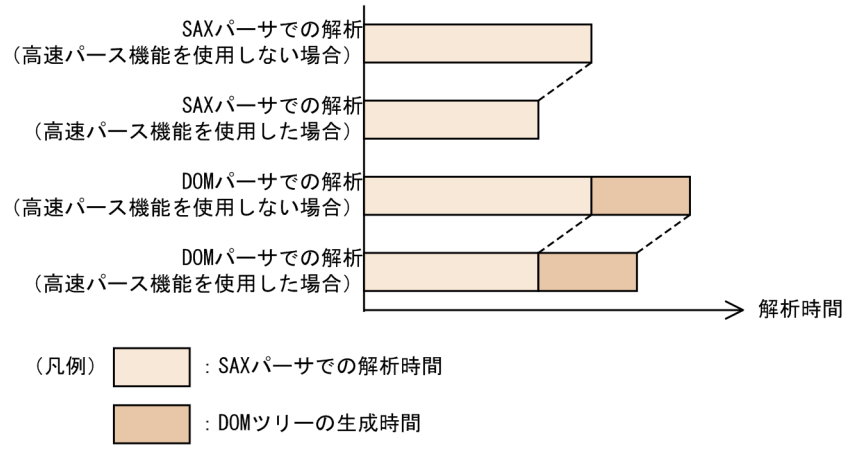
なお，高速パース機能を使用する場合は，「6.20　高速パース機能に関する注意事項」の

内容も確認してください。

参考

高速パース機能を使用することで短縮できる解析時間の割合について

高速パース機能は，SAX パーサでの解析時間を短縮します。また，DOM パーサでの解析は，SAX パーサでの解析と DOM ツリーの生成で実装されているため，DOM パーサでの解析時間も短縮できます。ただし，高速パース機能を使用しても DOM ツリーの生成時間は変わりません。このため，次の図に示すように，SAX パーサでの解析と比べて，解析時間全体に対して短縮される時間の割合は小さくなります。

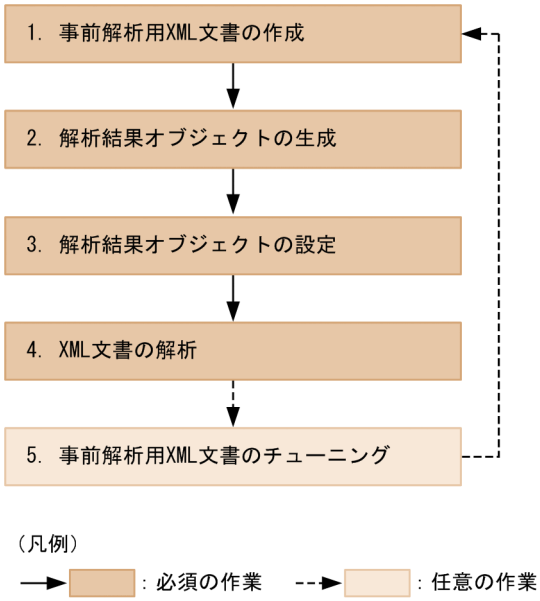


また，妥当性検証を行う場合は，解析時間に占める妥当性検証処理の割合が大きいため，短縮される解析時間の割合は相対的に小さくなります。特に，javax.xml.validation.Schema クラスを使わないで妥当性検証を行う場合は，解析時間はほとんど短縮されません。
小さいサイズの XML 文書を解析する場合も，解析時間に占める XML パーサの初期化時間の割合が大きいため，短縮される解析時間の割合は相対的に小さくなります。

3.4.2 高速パースのための作業の流れ

ユーザプログラム中にコードを記述して高速パース機能を使用する場合の，ユーザが実施する作業の流れを次に示します。

図 3-6　高速パース機能を使用するための作業の流れ



各作業について説明します。

1. **事前解析用 XML 文書の作成**
   解析対象の XML 文書の構造的な特徴を考慮して，事前解析用 XML 文書を作成します。事前解析用 XML 文書を作成するための指針については，「3.4.3　事前解析用 XML 文書の作成」を参照してください。

2. **解析結果オブジェクトの生成**
   事前解析用 XML 文書を解析して，解析結果オブジェクトを生成します。解析結果オブジェクトの生成方法については，「3.4.4　解析結果オブジェクトの生成」を参照してください。

3. **解析結果オブジェクトの設定**
   解析結果オブジェクトを Cosminexus XML Processor の XML パーサに設定します。解析結果オブジェクトの設定方法については，「3.4.5　解析結果オブジェクトの設定」を参照してください。

4. **XML 文書の解析**
   解析結果オブジェクトを設定した XML パーサの parse メソッドで，XML 文書を解析します。高速パース機能の対象となる parse メソッドについては，「3.4.6　XML 文書の解析」を参照してください。

5. **事前解析用 XML 文書のチューニング**
   必要に応じて，事前解析用 XML 文書のチューニングを行います。チューニング情報を調査して事前解析用 XML 文書を最適化することで，解析速度をさらに向上できます。事前解析用 XML 文書のチューニング方法については，「3.4.9　事前解析用 XML

文書のチューニング」を参照してください。

なお，高速パース機能を使用するためにユーザプログラムに記述するコード例については，「3.4.8　高速パース機能を使用するためのコード例」を参照してください。

3.4.3　事前解析用 XML 文書の作成

事前解析用 XML 文書は，解析対象の XML 文書の構造的な特徴を考慮して作成します。事前解析用 XML 文書を作成する際は，次の指針に従ってください。

表 3-7　事前解析用 XML 文書を作成するための指針

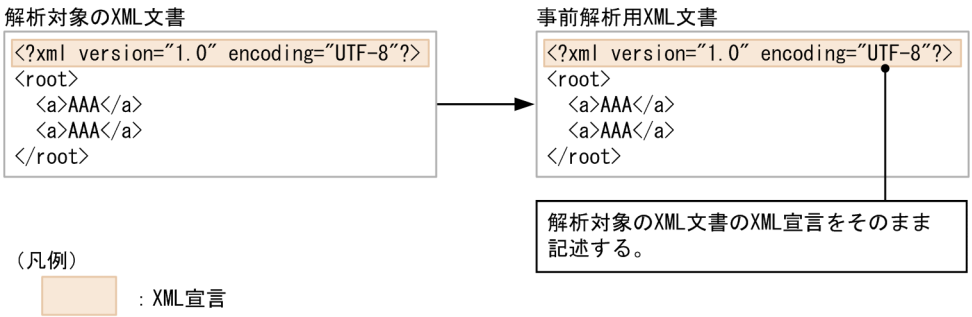
項番	適用個所	指針
1	XML 宣言	解析対象の XML 文書と一致させる。
2	要素と属性	解析対象の XML 文書に記述されている要素と属性を記述する。
3	要素と属性の順序	解析対象の XML 文書と一致させる。
4	要素の入れ子構造	解析対象の XML 文書と一致させる。
5	要素の繰り返し構造	繰り返す可能性がある要素は，繰り返しの形で記述する。
6	テキスト，CDATA セクション，属性値	解析対象の XML 文書と一致させる必要はない。
7	DTD，コメント，処理命令	記述する必要はない。

それぞれの指針について，具体的な記述例を示して説明します。

(1) XML 宣言

解析対象の XML 文書にある XML 宣言を，そのまま事前解析用 XML 文書に記述してください。例を次の図に示します。

図 3-7　XML 宣言を記述する例（事前解析用 XML 文書の作成指針）

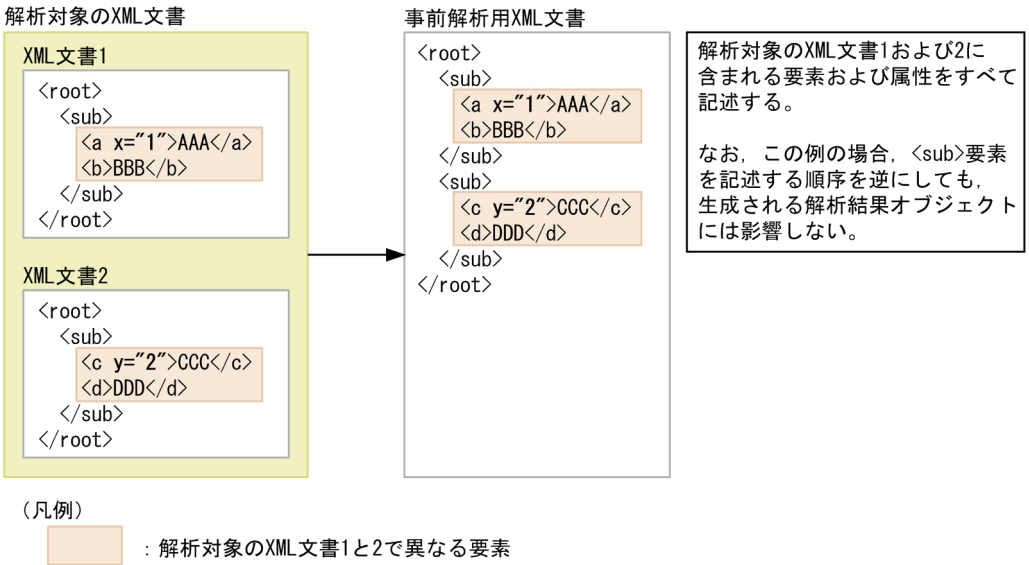


(2) 要素と属性

解析対象の XML 文書にある要素や属性を，事前解析用 XML 文書に記述します。事前解析用 XML 文書に記述された要素や属性の情報は，生成される解析結果オブジェクトに記録されます。

解析対象の XML 文書間で異なる要素や属性がある場合は，すべての要素や属性を事前解析用 XML 文書に記述します。例を次の図に示します。

図 3-8　要素と属性を記述する例（事前解析用 XML 文書の作成指針）



ただし，要素中のテキスト，CDATA セクション，および属性値の内容については，解析対象の XML 文書と一致させる必要はありません。詳細については，「3.4.3(6)　テキスト，CDATA セクション，属性値」を参照してください。

なお，次の記述は XML 規格では区別されませんが，事前解析では異なる要素として区別されます。

- 空要素と空要素タグ（例：<a> と <a/>）
- 改行文字・空白文字・タブの数や位置が異なる要素

解析対象の XML 文書にこれらの要素がある場合は，事前解析用 XML 文書には区別して記述する必要があります。例を次の図に示します。

図 3-9 空要素と空要素タグを記述する例（事前解析用 XML 文書の作成指針）

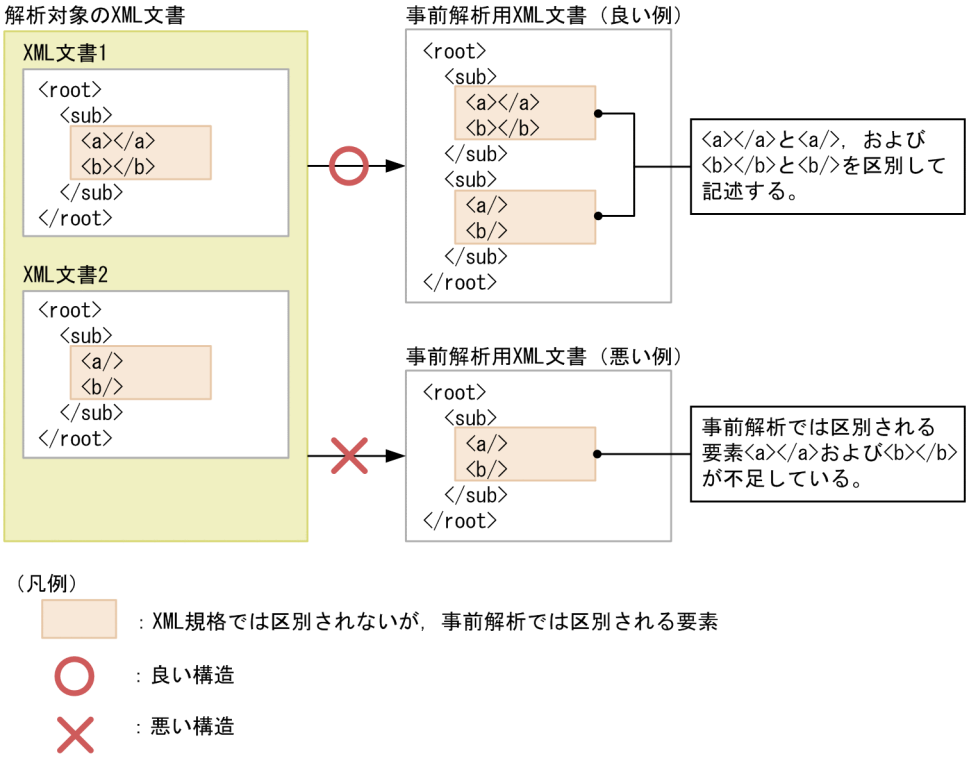
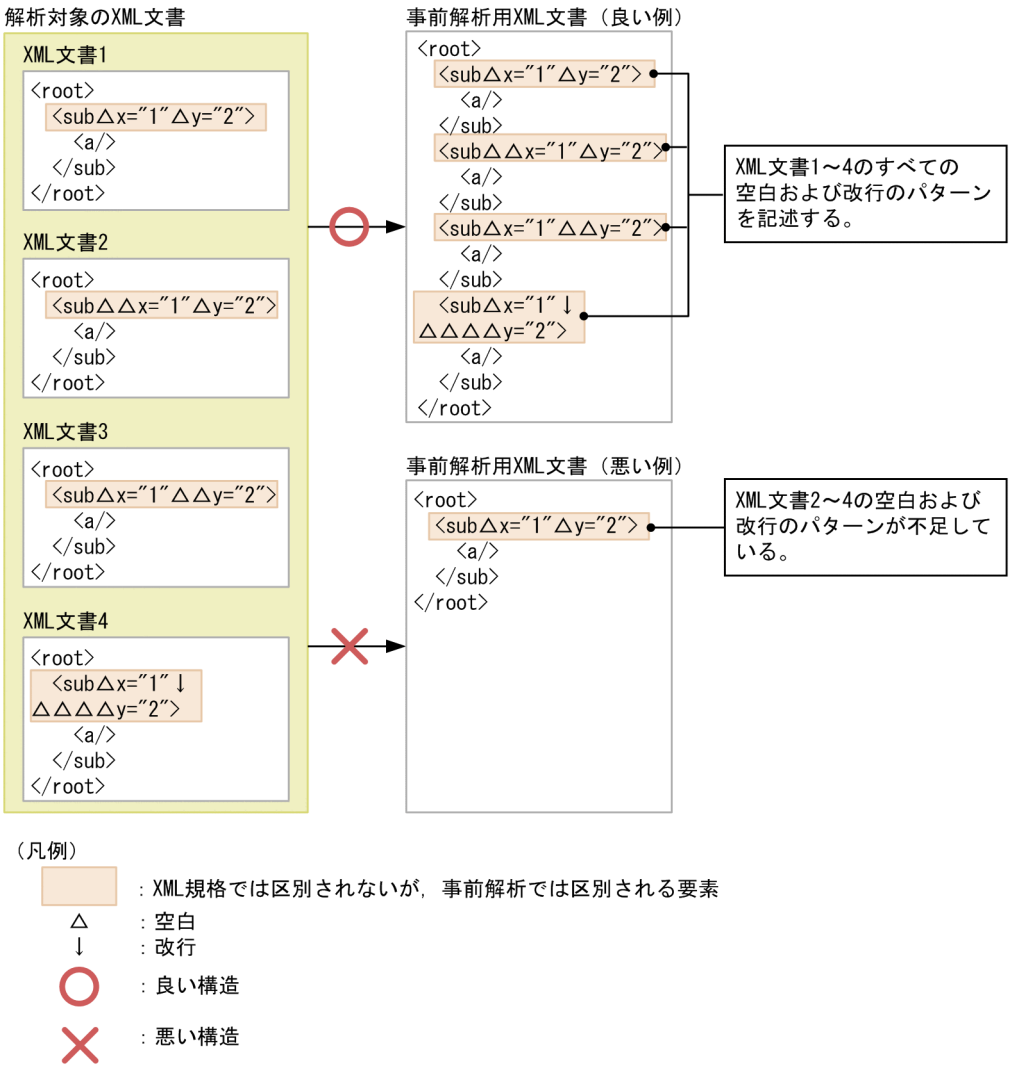


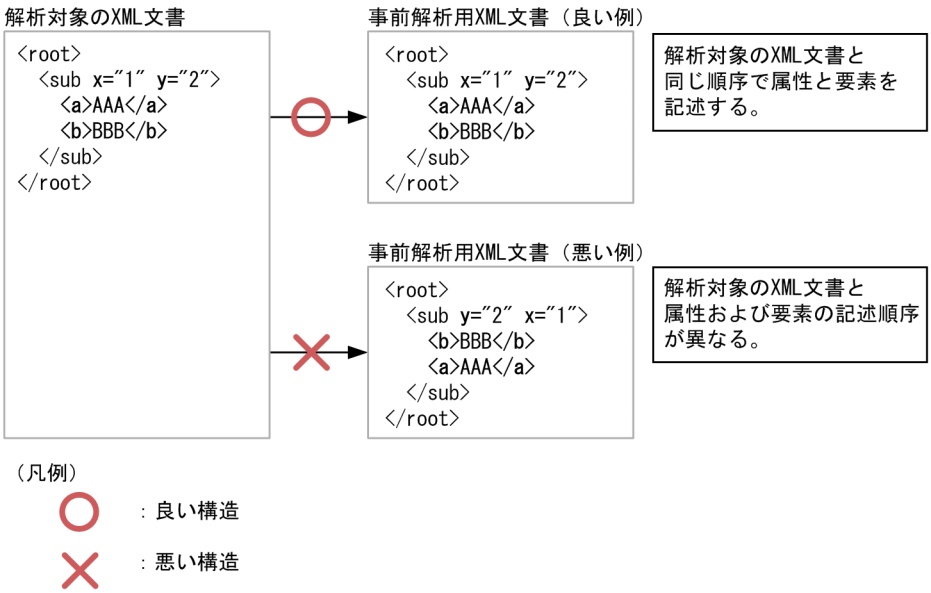
図 3-10　改行文字・空白文字の数や位置が異なる要素を記述する例（事前解析用 XML 文書の作成指針）



(3) 要素と属性の順序

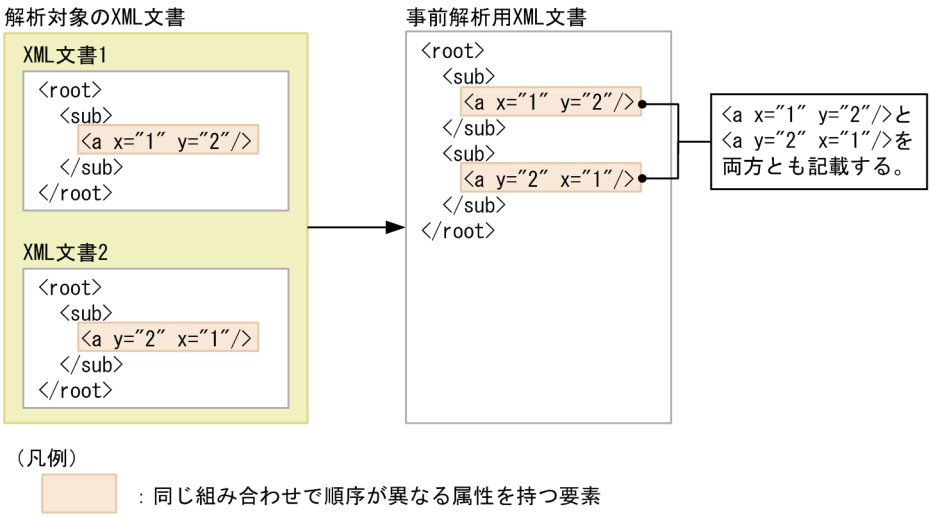
解析対象の XML 文書にある要素や属性は，そのままの順序で事前解析用 XML 文書に記述します。例を次の図に示します。

図 3-11　要素と属性を記述する例（事前解析用 XML 文書の作成指針・要素と属性の順序）



解析対象の XML 文書に，同じ組み合わせで順序が異なる要素や属性が記述されている場合は，それらをすべて事前解析用 XML 文書に記述します。例を次の図に示します。

図 3-12　同じ組み合わせで順序が異なる属性を記述する例（事前解析用 XML 文書の作成指針）

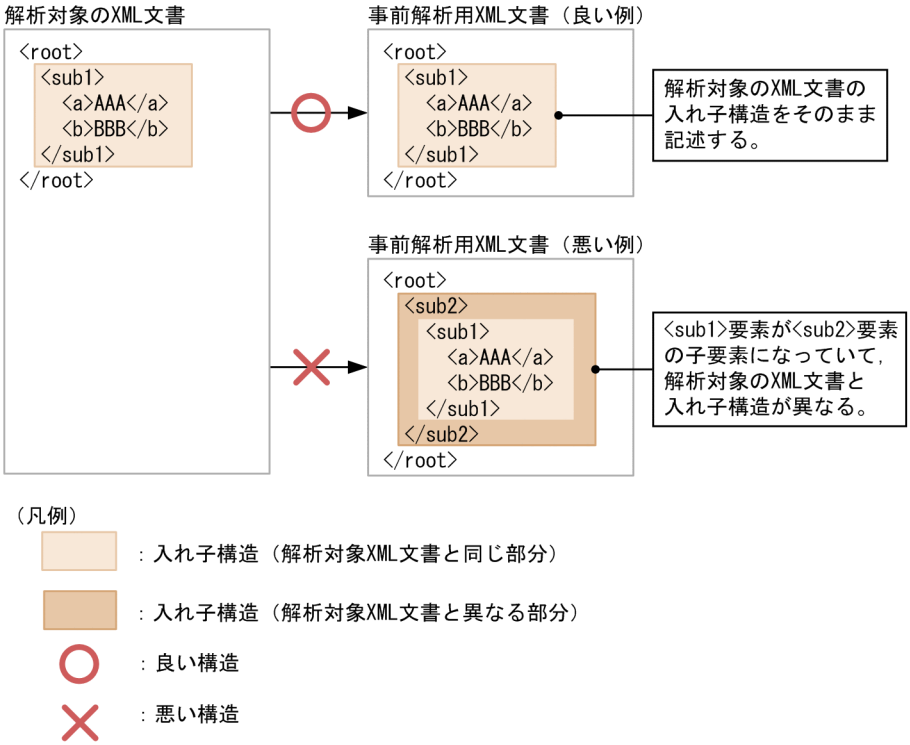


（4）要素の入れ子構造

解析対象の XML 文書にある要素の入れ子構造は，そのままの形で事前解析用 XML 文書

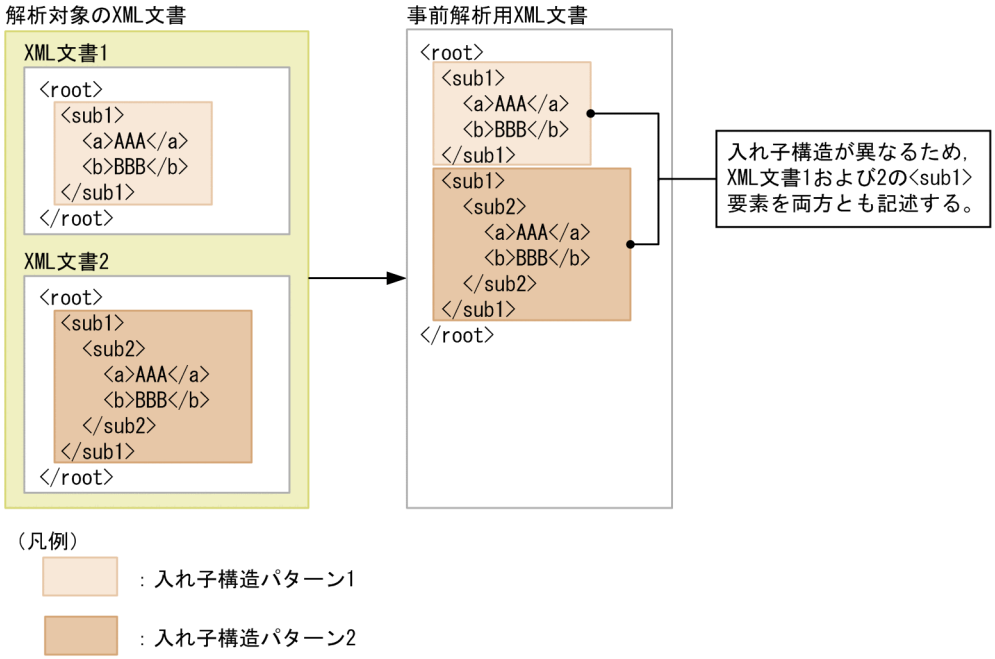
に記述します。例を次の図に示します。

図 3-13　要素の入れ子構造を記述する例（事前解析用 XML 文書の作成指針）



解析対象の XML 文書に異なる入れ子構造が複数ある場合は，すべての入れ子構造を事前解析用 XML 文書に記述します。例を次の図に示します。

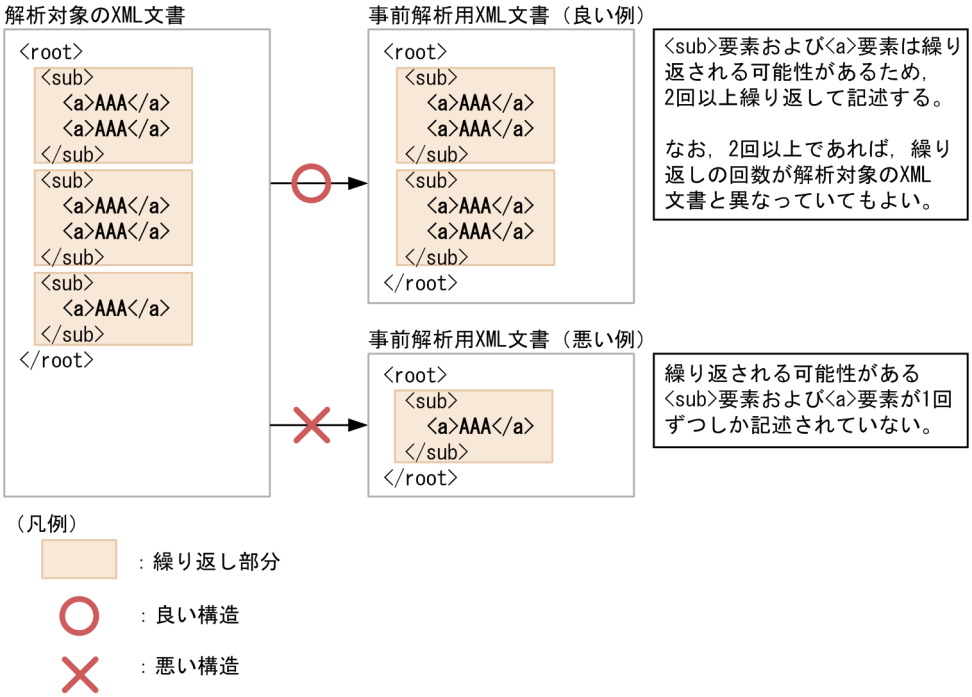
図 3-14　要素の入れ子構造を複数記述する例（事前解析用 XML 文書の作成指針）



(5) 要素の繰り返し構造

解析対象の XML 文書に繰り返す可能性がある要素がある場合は，繰り返す可能性がある要素を，2 回以上連続で事前解析用 XML 文書に記述します。2 回以上連続で記述することで，解析時に，繰り返す可能性がある要素として扱われます。なお，事前解析用 XML 文書に連続して記述する回数が 2 回以上であれば，解析対象の XML 文書と繰り返しの回数が異なっても，高速パース機能の性能に影響はありません。例を次の図に示します。

図 3-15　要素の繰り返し構造を記述する例（事前解析用 XML 文書の作成指針）

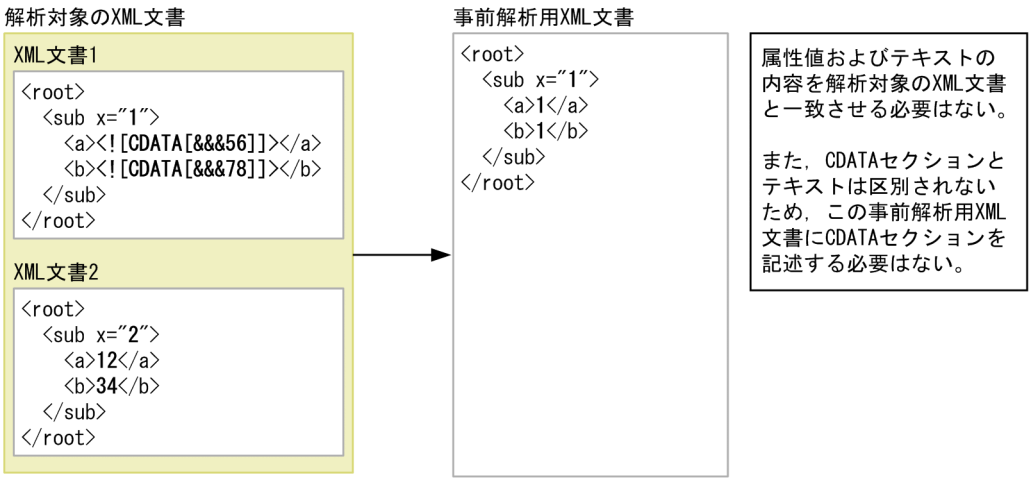


（6）テキスト，CDATA セクション，属性値

事前解析用 XML 文書に記述されたテキスト，CDATA セクション，および属性値の内容は，生成される解析結果オブジェクトに影響を与えません。このため，事前解析用 XML 文書のテキスト，CDATA セクション，および属性値の内容を，解析対象の XML 文書と一致させる必要はありません。

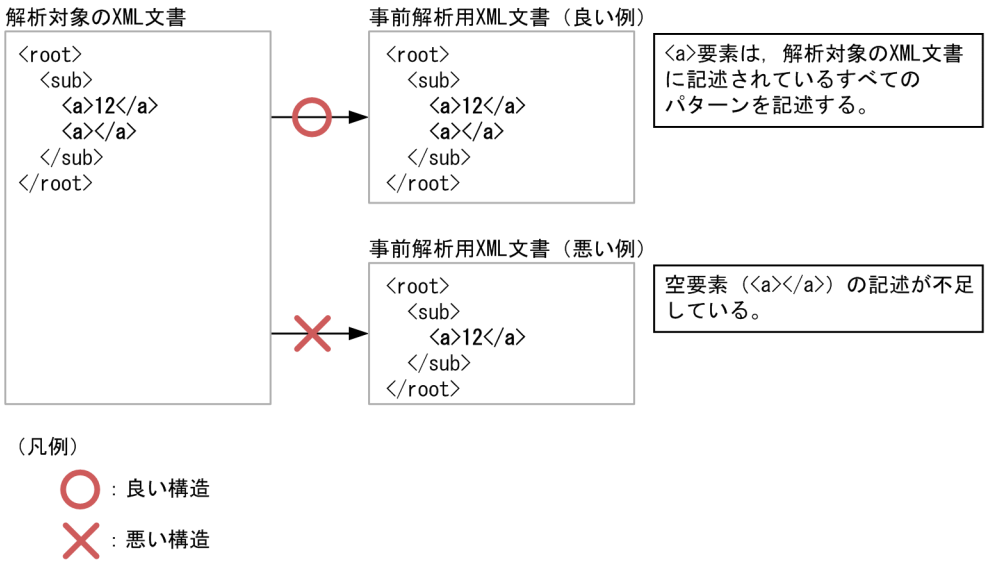
また，テキストと CDATA セクションは区別されません（どちらを記述しても，生成される解析結果オブジェクトは同じです）。例を次の図に示します。

図 3-16　テキスト，CDATA セクション，属性値を記述する例（事前解析用 XML 文書の作成指針）



内容を持つ要素（例：<a>12）と空要素（例：<a>）は区別されます。また，改行，タブ，空白はテキストなので，これらのテキストを持つ要素（例：<a>［改行］）と空要素（例：<a>）は区別されます。このため，解析対象の XML 文書にこのような要素が含まれる場合は，事前解析用 XML 文書にすべての要素を区別して記述する必要があります。例を次の図に示します。

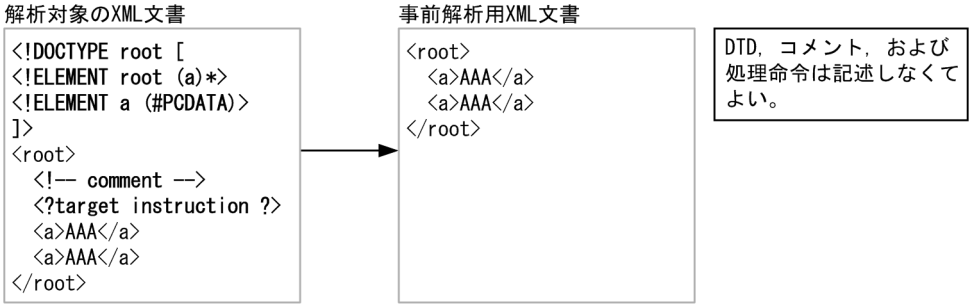
図 3-17　空要素を記述する例（事前解析用 XML 文書の作成指針）



(7) DTD，コメント，処理命令

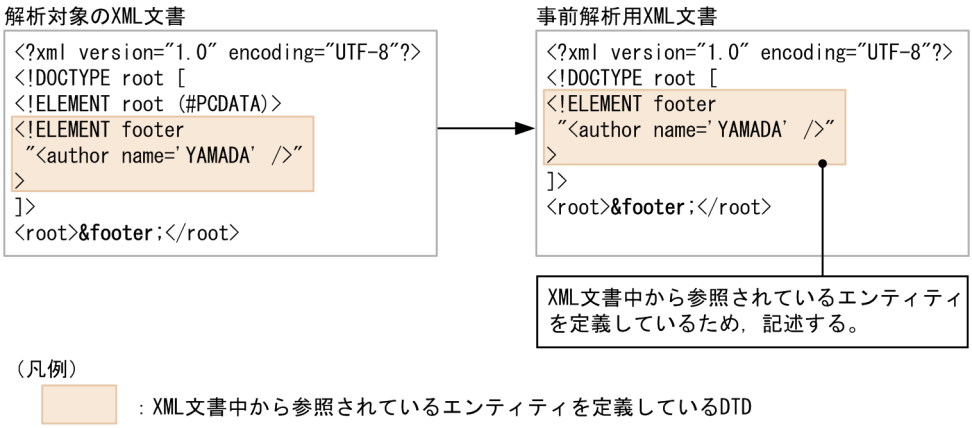
DTD，コメント，および処理命令は，解析結果オブジェクトには含まれません。このため，事前解析用 XML 文書にこれらを記述する必要はありません（記述しても生成される解析結果オブジェクトには影響しません）。例を次の図に示します。

図 3-18　DTD，コメント，処理命令に関する例（事前解析用 XML 文書の作成指針）



ただし，解析対象の XML 文書の中から参照されているエンティティを定義している DTD は，事前解析用 XML 文書に記述する必要があります。例を次の図に示します。

図 3-19　XML 文書中から参照されているエンティティの定義を記述する例（事前解析用 XML 文書の作成指針）



3.4.4　解析結果オブジェクトの生成

事前解析用 XML 文書を解析して，解析結果オブジェクトである PreparsedObject インスタンスを生成します。

解析結果オブジェクト（PreparsedObject インスタンス）を生成するには，次に示すメソッドを使用します。

- com.cosminexus.jaxp.preparsedxml.PreparsedObjectFactory クラスの newInstance

メソッド

- com.cosminexus.jaxp.preparsedxml.PreparsedObject クラスの newPreparsedObject メソッド

メソッドの詳細については，「3.4.7　高速パース機能で使用するクラス」を参照してください。

解析結果オブジェクトを生成するコードの例を次に示します。

```
// 事前解析用XML文書を用意する
File xml = new File("learning1.xml");
// エンティティリゾルバを用意する
MyEntityResolver entityResolver = new MyEntityResolver();
// エラーハンドラを用意する
MyErrorHandler errorHandler = new MyErrorHandler();
// PreparsedObjectFactoryを生成する
PreparsedObjectFactory pof = PreparsedObjectFactory.newInstance();
// 名前空間を有効にする
pof.setNamespaceAware(true);
// エンティティリゾルバを設定する
pof.setEntityResolver(entityResolver);
// エラーハンドラを設定する
pof.setErrorHandler(errorHandler);
// 解析結果オブジェクトを生成する
PreparsedObject pobj = pof.newPreparsedObject(xml);
```

3.4.5　解析結果オブジェクトの設定

解析結果オブジェクトを Cosminexus XML Processor の XML パーサに設定します。解析結果オブジェクトは，次に示す XML パーサに設定できます。

- javax.xml.parsers.DocumentBuilder
- javax.xml.parsers.SAXParser
- org.xml.sax.XMLReader

解析結果オブジェクトを XML パーサに設定するには，http://cosminexus.com/xml/properties/preparsedobject-load プロパティを使用します。プロパティの詳細については「4.6　高速パース機能のプロパティの使用方法」を参照してください。

3.4.6　XML 文書の解析

解析結果オブジェクトを設定した XML パーサの parse メソッドで，XML 文書を解析します。高速パース機能の対象となる parse メソッドを次の表に示します。

注意事項

表に記載のない parse メソッドを使用して高速パース機能を使用した場合の動作は保証しません。

表 3-8 高速パース機能の対象となる parse メソッド

クラス名	メソッド名
javax.xml.parsers.DocumentBuilder	parse(File f)
	parse(InputSource is)
	parse(InputStream is)
	parse(InputStream is, String systemId)
	parse(String uri)
javax.xml.parsers.SAXParser	parse(File f, DefaultHandler dh)
	parse(InputSource is, DefaultHandler dh)
	parse(InputStream is, DefaultHandler dh)
	parse(InputStream is, DefaultHandler dh, String systemId)
	parse(String uri, DefaultHandler dh)
org.xml.sax.XMLReader	parse(InputSource input)
	parse(String systemId)

3.4.7 高速パース機能で使用するクラス

高速パース機能を使用する場合，次に示すクラスを使用します。

- PreparsedObjectFactory
- PreparsedObject

各クラスについて説明します。

(1) PreparsedObjectFactory クラス

(a) 説明

解析結果オブジェクトを生成するためのファクトリクラスです。

PreparsedObjectFactory クラスは，スレッドセーフではありません。このため，複数のスレッドから同時に同じ PreparsedObjectFactory にアクセスすることはできません。PreparsedObjectFactory クラスは，スレッド間の競合を避けるため，次のどちらかの方法で使用してください。

- 各スレッドに一つの PreparsedObjectFactory インスタンスを持つ。
- 各スレッドが排他的に PreparsedObjectFactory インスタンスにアクセスする。

(b) パッケージ名およびクラス名

com.cosminexus.jaxp.preparsedxml.PreparsedObjectFactory

(c) 形式

```
public class PreparsedObjectFactory
```

(d) メソッド一覧

メソッド名	機能
newInstance	解析結果オブジェクトを生成するためのファクトリを生成して返却する。
setNamespaceAware	事前解析用 XML 文書を解析する際の名前空間の有効・無効を設定する。
setErrorHandler	事前解析用 XML 文書の解析で発生するエラーを受け取るエラーハンドラを設定する。
setEntityResolver	事前解析用 XML 文書に含まれるエンティティを解決するためのエンティティリゾルバを設定する。
setTuningInfoFlag	チューニング情報を出力するかどうかを設定する。
newPreparsedObject	事前解析用 XML 文書を解析して，解析結果オブジェクトを生成する。

(e) メソッド詳細

newInstance メソッド

説明

解析結果オブジェクトを生成するためのファクトリを生成して返却します。

形式

```
public static PreparsedObjectFactory newInstance()
```

パラメタ

なし

戻り値

PreparsedObjectFactory の新しいインスタンスを返却します。

例外

なし

setNamespaceAware メソッド

説明

事前解析用 XML 文書を解析する際の名前空間の有効・無効を設定します。

デフォルトでは，`false` が設定されています。

形式

```
public void setNamespaceAware(boolean awareness)
```

パラメタ

- awareness

  名前空間の有効・無効を設定します。

  `true`：名前空間を有効にする

  `false`：名前空間を無効にする

戻り値

なし

例外

なし

setErrorHandler メソッド

説明

newPreparsedObject メソッドを呼び出す際に，事前解析用 XML 文書の解析で発生するエラーを受け取るエラーハンドラを設定します。
デフォルトでは，次のエラーハンドラが設定されているように動作します。

```
class DefaultErrorHandler implements ErrorHandler {
     public void fatalError(SAXParseException e) throws SAXException {
         throw e;
     }
     public void error(SAXParseException e) throws SAXException {
         throw e;
     }
     public void warning(SAXParseException e) throws SAXException {
         // 何もしない
     }
 }
```

形式

```
public void setErrorHandler(org.xml.sax.ErrorHandler
errorHandler)
```

パラメタ

- errorHandler

  エラーハンドラを設定します。null を設定した場合，デフォルトの動作になります。

戻り値

なし

例外

なし

setEntityResolver メソッド

説明

newPreparsedObject メソッドを呼び出す際に，事前解析用 XML 文書に含まれるエンティティを解決するためのエンティティリゾルバを設定します。
デフォルトでは，次のエンティティリゾルバが設定されているように動作します。

```
class DefaultEntityResolver implements EntityResolver {
     InputSource resolveEntity(String publicId, String systemId)
throws SAXException, IOException {
         return null; // 常にnullを返す
     }
}
```

形式

```
public void setEntityResolver(org.xml.sax.EntityResolver
```

```
entityResolver)
```

パラメタ

- entityResolver
  エンティティリゾルバを設定します。null を設定した場合，デフォルトの動作になります。

戻り値

なし

例外

なし

setTuningInfoFlag メソッド

説明

チューニング情報を出力するかどうかを設定します。
デフォルトでは，`false` が設定されています。
チューニング情報を出力する場合は，このメソッドで `true` を設定し，システムプロパティ「com.cosminexus.jaxp.preparsedxml.tuning.level」に `INFO` を設定する必要があります。チューニング情報の出力方法の詳細については，「3.4.9(3)　チューニング情報の出力方法」を参照してください。

形式

```
public void setTuningInfoFlag(boolean state)
```

パラメタ

- state
  チューニング情報を出力するかどうかを設定します。
  `true`：チューニング情報を出力する
  `false`：チューニング情報を出力しない

戻り値

なし

例外

なし

newPreparsedObject メソッド

説明

事前解析用 XML 文書を解析して，解析結果オブジェクトを生成します。

形式

public PreparsedObject newPreparsedObject(java.io.File xml)
throws IllegalArgumentException, SAXException, IOException

パラメタ

- xml
  事前解析用 XML 文書を示す File オブジェクトを指定します。null は指定で

きません。

戻り値

解析結果オブジェクトである PreparsedObject の新しいインスタンスを返却します。

例外

- IllegalArgumentException
  xml パラメタに null が指定されているときに発生します。
- SAXException
  次の場合に発生します。
  ・処理中に SAX エラーが発生したとき
  ・XML1.1 文書を事前解析しようとしたとき（詳細については「6.20.3 XML1.1 に関する注意事項」を参照のこと）
- IOException
  入出力エラーが発生したときに発生します。

(2) PreparsedObject クラス

(a) 説明

解析結果オブジェクトを示すクラスです。

(b) パッケージ名およびクラス名

com.cosminexus.jaxp.preparsedxml.PreparsedObject

(c) 形式

```
public class PreparsedObject
```

(d) メソッド一覧

メソッド名	機能
isNamespaceAware	事前解析用 XML 文書を解析する際に設定された名前空間の有効・無効を返却する。
getErrorHandler	事前解析用 XML 文書を解析する際に設定されたエラーハンドラを返却する。
getEntityResolver	事前解析用 XML 文書を解析する際に設定されたエンティティリゾルバを返却する。

(e) メソッド詳細

isNamespaceAware メソッド

説明

事前解析用 XML 文書を解析する際に設定された名前空間の有効・無効を返却します。

形式

```
public boolean isNamespaceAware()
```

パラメタ

なし

戻り値

`true`：名前空間が有効である

`false`：名前空間が無効である

例外

なし

getErrorHandler **メソッド**

説明

事前解析用 XML 文書を解析する際に設定されたエラーハンドラを返却します。

形式

```
public ErrorHandler getErrorHandler()
```

パラメタ

なし

戻り値

事前解析用 XML 文書を解析する際に設定されたエラーハンドラを返却します。設定されていない場合は，null を返却します。

例外

なし

getEntityResolver **メソッド**

説明

事前解析用 XML 文書を解析する際に設定されたエンティティリゾルバを返却します。

形式

```
public EntityResolver getEntityResolver()
```

パラメタ

なし

戻り値

事前解析用 XML 文書を解析する際に設定されたエンティティリゾルバを返却します。設定されていない場合は，null を返却します。

例外

なし

3.4.8 高速パース機能を使用するためのコード例

高速パース機能を使用するために，ユーザプログラムに記述するコードの例を次に示し

ます。

```
import java.io.File;
import java.io.IOException;
import javax.xml.parsers.SAXParserFactory;
import javax.xml.parsers.SAXParser;
import org.xml.sax.EntityResolver;
import org.xml.sax.ErrorHandler;
import org.xml.sax.SAXException;
import org.xml.sax.helpers.DefaultHandler;
import com.cosminexus.jaxp.preparsedxml.PreparsedObjectFactory;
import com.cosminexus.jaxp.preparsedxml.PreparsedObject;

public class TestSAXParser{
    public static void main(String[] args){
        try{
            // 事前解析用XML文書から解析結果オブジェクトを生成する
            File xml = new File("learning1.xml");
            MyHandler handler = new MyHandler ();
            PreparsedObjectFactory pof = PreparsedObjectFactory.newInstance();
            pof.setNamespaceAware(true);
            pof.setEntityResolver(handler);
            pof.setErrorHandler(handler);
            PreparsedObject pobj = pof.newPreparsedObject(xml);

            // XMLパーサを生成する
            SAXParserFactory spf = SAXParserFactory.newInstance();
            spf.setNamespaceAware(true);
            SAXParser sp = spf.newSAXParser();

            // 解析結果オブジェクトをXMLパーサに設定して，解析を実行する
            sp.setProperty("http://cosminexus.com/xml/properties/
preparsedobject-load", pobj);
            sp.parse("SampleSAX.xml", handler);
        } catch(IllegalArgumentException iae){
            System.out.println("MSG : " + iae.getMessage());
        }catch(SAXException se){
            System.out.println("MSG : " + se.getMessage());
        }catch(IOException ioe){
            System.out.println("MSG : " + ioe.getMessage());
        }catch(Exception e){
            e.printStackTrace();
        }
    }
}

class MyHandler extends DefaultHandler {
  //エラーハンドラとエンティティリゾルバの実装を記述
}
```

3.4.9 事前解析用 XML 文書のチューニング

事前解析用 XML 文書のチューニング情報を参照すると，高速パース機能が効果的に動作しているかどうかを調査できます。調査結果を基に事前解析用 XML 文書を最適化することで，解析速度をさらに向上できます。

事前解析用 XML 文書のチューニング情報の種類，利用方法，および出力方法について説明します。

(1) チューニング情報の種類

チューニング情報として出力できるファイルは，次の 2 種類です。

- チューニングサマリファイル
- チューニング詳細ファイル

これらのファイルには，解析時のマッチング処理についての情報が出力されます。マッチング処理とは，解析時に解析結果オブジェクトの情報を利用できたかどうかを判定する処理のことです。解析結果オブジェクトの情報を利用できた場合を「マッチング処理に成功した」，解析結果オブジェクトの情報を利用できなかった場合を「マッチング処理に失敗した」といいます。

各ファイルについて説明します。

(a) チューニングサマリファイル

高速パース機能を使用したそれぞれの解析について，マッチング処理の成功率，解析結果オブジェクトの生成に使用した事前解析用 XML 文書のファイル名，解析対象の XML 文書のシステム識別子，チューニング詳細ファイル名などが出力されます。高速パース機能を使用した解析処理で，どのくらい効果的に解析結果オブジェクトが利用されているかを調査できます。

チューニングサマリファイルの詳細については，「3.4.10　チューニングサマリファイルの詳細」を参照してください。

(b) チューニング詳細ファイル

高速パース機能を使用した一つの解析について，チューニングサマリファイルの情報に加え，マッチング処理の履歴が出力されます。高速パース機能を使用した解析処理で，マッチングに失敗している個所の詳細を調査できます。

チューニング詳細ファイルの詳細については，「3.4.11　チューニング詳細ファイルの詳細」を参照してください。

(2) チューニング情報の利用方法

チューニング情報の利用方法を次に示します。

1. **チューニング情報を出力する。**
   チューニング情報の出力方法については，「3.4.9(3)　チューニング情報の出力方法」を参照してください。
2. **チューニングサマリファイルを参照して，マッチング処理の成功率が低い解析を抽出する。**
   マッチング処理の成功率が低いということは，解析結果オブジェクトが学習した事前解析用 XML 文書の構造と解析対象の XML 文書の構造の違いが大きく，高速パース機能が効果的に動作していないことを示します。
3. 2. **で抽出した各解析について，チューニング詳細ファイルを参照して，マッチングに失敗した個所の履歴を調査する。**
   マッチングに失敗した個所の履歴を調査して，事前解析用 XML 文書と解析対象の XML 文書の構造が不一致になっている部分を抽出します。

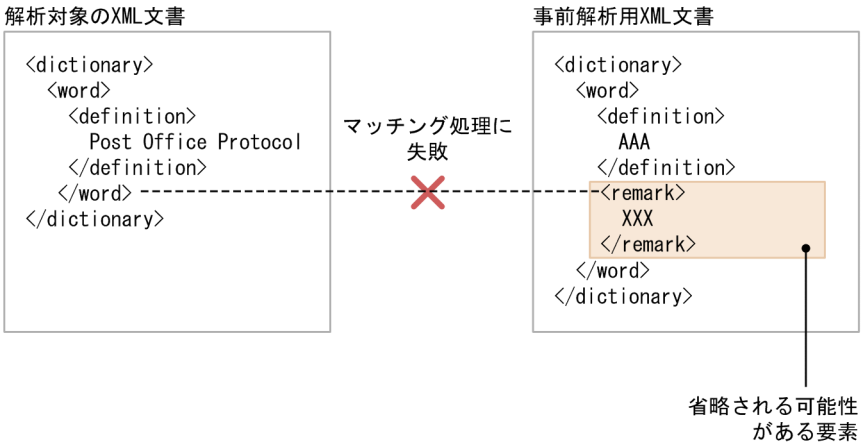
4. **事前解析用 XML 文書を修正する。**

3. の調査結果を基に，事前解析用 XML 文書を修正します。事前解析用 XML 文書を作成するための指針については，「3.4.3　事前解析用 XML 文書の作成」を参照してください。

注意事項

解析対象の XML 文書に記述されているすべての要素や属性が事前解析用 XML 文書に含まれていても，マッチング処理の成功率が低い場合は，高速パース機能の効果は低くなります。

解析対象の XML 文書に記述されているすべての要素や属性が事前解析用 XML 文書に含まれていても，マッチング処理の成功率が低下する場合の例を次に示します。



この例では，解析対象の XML 文書で remark 要素が省略されているため，remark 要素のマッチング処理は失敗し，チューニング詳細ファイルに失敗履歴が記録されます。この結果，マッチング処理の成功率が低下します。

(3) チューニング情報の出力方法

チューニング情報を出力するには，Cosminexus XML Processor が規定するシステムプロパティを設定する必要があります。また，PreparsedObject を使用した高速パースでチューニング情報を出力する場合は，システムプロパティの設定に加え，PreparsedObjectFactory クラスの setTuningInfoFlag メソッドで `true` を設定して，PreparsedObject を生成する必要があります。PreparsedObjectFactory クラスの詳細については，「3.4.7(1)　PreparsedObjectFactory クラス」を参照してください。

チューニング情報を出力するためのシステムプロパティ，チューニング情報の出力先，およびエラー発生時の動作について説明します。

注意事項

チューニング情報を出力した場合は，解析性能が劣化します。このため，チューニング情報は，高速パース機能の性能を評価するときだけ出力してください。

(a) チューニング情報を出力するためのシステムプロパティ

システムプロパティ名

com.cosminexus.jaxp.preparsedxml.tuning.level

値

- OFF またはこのプロパティを設定していない場合：チューニング情報を出力しない
- INFO：チューニング情報を出力する

大文字の INFO 以外が指定された場合は，OFF が指定されたものとして扱われます。

(b) チューニング情報の出力先

チューニング情報は，高速パース機能を利用した解析の終了時に出力されます。チューニング情報の出力先は，次の順序で決定されます。

1. **サーバのログ出力ディレクトリ**※ /jaxp

   注※ 「サーバのログ出力ディレクトリ」は，動作しているサーバによって異なります。

   J2EE サーバの場合

   J2EE サーバのオプション定義ファイル（usrconf.cfg）の ejb.server.log.directory キーに指定されているディレクトリです。デフォルトでは，次のディレクトリとなります。

   <Cosminexus 作業ディレクトリ >/ejb/< サーバ名称 >/logs

   Web コンテナサーバの場合

   Web コンテナサーバのオプション定義ファイル（usrconf.cfg）の web.server.log.directory キーに指定されているディレクトリです。デフォルトでは，次のディレクトリとなります。

   <Cosminexus のインストールディレクトリ >/CC/web/containers/< サーバ名称 >/logs

2. user.dir **システムプロパティで取得できるディレクトリ** /logs/jaxp

なお，同名のファイルは上書きされます。同名のファイルが存在する場合は，必要に応じて退避してください。ファイル名の規則については，「3.4.10(1)　ファイル名（チューニングサマリファイル）」および「3.4.11(1)　ファイル名（チューニング詳細ファイル）」を参照してください。

また，チューニングサマリファイルおよびチューニング詳細ファイルは自動的には削除されません。必要に応じて削除してください。チューニングサマリファイルのファイルサイズの上限値については，「3.4.10(3) ファイルサイズの上限値」を参照してください。チューニング詳細ファイルの合計ファイルサイズの上限値については，「3.4.11(3) 合計ファイルサイズの上限値」を参照してください。

(c) エラー発生時の動作

エラーが発生してチューニング情報を出力できない場合の動作を次に示します。

入出力エラーが発生した場合

チューニングサマリファイル，およびチューニング詳細ファイルを出力する際に入出力エラーが発生した場合は，これらのファイルは出力されません。また，IOException 例外が発生します。

XML 文書の解析中にエラーが発生した場合

チューニングサマリファイルには情報が出力されません。チューニング詳細ファイルには，エラーが発生する前までの情報が出力されます。

3.4.10 チューニングサマリファイルの詳細

チューニングサマリファイルのファイル名，出力形式，およびファイルサイズの上限値を次に示します。

(1) ファイル名（チューニングサマリファイル）

```
csmjaxp_ppxml_summary.csv
```

(2) 出力形式（チューニングサマリファイル）

出力行	出力内容
1 行目	チューニングサマリファイルであることを示すヘッダ `Cosminexus XML Processor  Fast Parse Function  Tuning Summary Information`
2 行目	チューニングサマリファイルが作成された日時
3 行目	4 行目以降の行ヘッダ `matching rate,preparsed XML,parsed XML,tuning detail file,date`
4 行目以降	次の情報がコンマで区切られて出力される。 1. マッチング処理の成功率 成功回数÷マッチング処理の実行回数× 100 2. 事前解析用 XML ファイル名※ 3. 解析対象の XML 文書のシステム識別子※ システム識別子が設定されていない場合は，null が出力されます。 4. チューニング詳細ファイルのファイル名※ 5. 解析が終了した日時

注※

出力文字列にコンマ（,）が含まれている場合も，そのまま出力されます。

出力例

```
1 Cosminexus XML Processor  Fast Parse Function  Tuning Summary Information
2 Create Date: Tue Feb 06 20:57:35 JST 2007
3 matching rate,preparsed XML,parsed XML,tuning detail file,date
4 45.5,D:¥tr1.xml,file:/novelA.xml,csmjaxp_ppxml_tr1_novelistA_00.txt, Tue Feb 06 20:57:35
  JST 2007
5 90.0,D:¥tr1.xml,file:/novelB.xml,...
6 97.5,D:¥tr2.xml,file:/comicA.xml,...
7 85.0,D:¥tr2.xml,file:/comicB.xml,...
```

(3) ファイルサイズの上限値

1MB※

注※

1MB を超えた場合，それ以降のデータは追記しません。

なお，ファイルサイズの上限値は，次のシステムプロパティで変更できます。

システムプロパティ名

com.cosminexus.jaxp.preparsedxml.tuning.summaryfile_maxsize

値

0 以上の整数を指定します。単位は MB です。

このプロパティを設定していない場合，および範囲外の値が指定された場合は，1 が設定されます。

3.4.11 チューニング詳細ファイルの詳細

チューニング詳細ファイルのファイル名，出力形式，およびファイルサイズの上限値を次に示します。

(1) ファイル名（チューニング詳細ファイル）

csmjaxp_ppxml_{事前解析用XML文書を表す文字列}_{解析対象のXML文書を表す文字列}_{連番}.txt

事前解析用 XML 文書を表す文字列

事前解析用 XML 文書の絶対パスの末尾部分から，最後に出現するピリオド（.）以降を除いたものです。

例：事前解析用 XML 文書の絶対パスが「D:¥home¥xml¥training¥tr1.xml」の場合

事前解析用 XML 文書を表す文字列は，「tr1」となります。

解析対象の XML 文書を表す文字列

解析対象となる XML 文書のシステム識別子の末尾部分から，最後に出現するピリオド以降を除いたものです。

例：解析対象の XML 文書のシステム ID が「file:/home/xml/data/

novelA.xml」の場合

解析対象の XML 文書を表す文字列は，「novelA」となります。

連番

00 ～ 99 の連番です。解析ごとに増加します。

なお，J2EE サーバまたは Web コンテナを再起動すると「00」にリセットされます。また，「99」の次は「00」となります。

(2) 出力形式（チューニング詳細ファイル）

出力行	出力内容
1 行目	チューニング詳細ファイルであることを示すヘッダ Cosminexus XML Processor Fast Parse Function Tuning Detail Information
2 行目	チューニング詳細ファイルが作成された日時
3 行目	事前解析用 XML 文書のファイル名
4 行目	解析対象の XML 文書のシステム識別子 システム識別子が設定されていない場合は，null が出力されます。
5 行目	マッチング情報の開始を示すヘッダ Matching Information start
6 行目以降	マッチング情報（マッチング処理の履歴） マッチング処理に成功している場合 o:{ マッチング処理の対象とした要素・属性 } ※ マッチング処理に失敗している場合 x:expected { マッチングを試みたパターン } ※ but { 解析対象 XML 文書中に出現したパターン } ※
最終行から 2 行目	マッチング情報の終了を示すフッタ Matching Information end
最終行	マッチング処理の成功率 成功回数÷マッチング処理の実行回数× 100（%） ただし，マッチング処理が 1 回も実行されなかった場合は，0.0% と出力されます

注※

要素または属性の一部だけが出力されることがあります。また，要素または属性の前後の文字が出力されることがあります。具体例は出力例を参照してください。

出力例

```
 1 Cosminexus XML Processor Fast Parse Function Tuning Detail Information
 2 Create Date: Wed Feb 07 13:12:42 JST 2007
 3 Preparsed XML: d:¥home¥xml¥training¥training1.xml
 4 Parsed XML: word.xml
 5 Matching Information start
 6 o:note>
 7 o:</note>
 8 o:<word>
 9 o:<name is_acronym=
10 o:>
11 o:</name>
12 o:<definition>
13 o:</definition>
14 o:<definition>
15 o:</definition>
16 x:expected <definition> but </word>
17  <w
18 o:</word>
19 o:<word>
20 x:expected <name is_acronym= but <update date="200
21 o:<update date=
22 o:/>
23 o:<name is_acronym=
24 o:>
25 o:</name>
26 o:<definition>
27 o:</definition>
28 x:expected <definition> but </word>
29  <w
30 o:</word>
31 o:<word>
32 x:expected <name is_acronym= but <update date="200
33 o:<update date=
34 x:expected /> but  e
35 o: editor=
36 o:/>
37 o:<name>
38 o:</name>
39 o:<definition>
40 o:</definition>
41 x:expected <definition> but </word>
42 </d
43 o:</word>
44 x:expected <word> but <abcdef
45 x:expected not begin with a but a
46 Matching Information end
47 Matching rate: 78.4%
```

要素<word>のマッチング処理に成功したことを示す。

要素<definition>のマッチング処理に失敗したことを示す。

要素の後ろの文字が出力されることがある。ここでは，</word>の後ろの改行文字と<wが出力されている。

要素の一部だけが出力されることがある。ここでは，閉じタグを示す/>だけが出力されている。

要素の一部だけが出力されることがある。ここでは，<abcdef以降の文字ghi/>が出力されていない。

文字aが出現したためマッチング処理に失敗したことを示す。

(3) 合計ファイルサイズの上限値

32MB ※

注※

出力済みのチューニング詳細ファイルの合計ファイルサイズが 32MB を超えた場合，それ以降はチューニング詳細ファイルが生成されません。

例えば，次の 1 ～ 5 の順にチューニング詳細ファイルが生成される場合，4 のファイルが生成された時点で 1 ～ 4 の合計ファイルサイズが 40MB となり，32MB を超えているため，5 のファイルは生成されません。

1. csmjaxp_ppxml_trl_novelistA_00.txt（10MB）
2. csmjaxp_ppxml_trl_novelistB_00.txt（10MB）
3. csmjaxp_ppxml_trl_novelistC_00.txt（10MB）
4. csmjaxp_ppxml_trl_novelistD_00.txt（10MB）
5. csmjaxp_ppxml_trl_novelistE_00.txt（10MB）

なお，合計ファイルサイズの上限値は，次のシステムプロパティで変更できます。

システムプロパティ名

com.cosminexus.jaxp.preparsedxml.tuning.detailfile_totalmaxsize

値

0 以上の整数を指定します。単位は MB です。
このプロパティを設定していない場合，および範囲外の値が指定された場合は，32 が設定されます。

3.5 Apache 独自の機能

Cosminexus XML Processor の 08-70 以降では，XML パーサや XSLT ／ XSLTC トランスフォーマに対応して Apache 独自のフィーチャーやプロパティを設定できます。また，Apache 独自の名前空間 URI を使用することもできます。

ここでは，使用できる Apache 独自のフィーチャー，プロパティ，および名前空間 URI を紹介します。それぞれの機能の詳細は，Apache の Xerces2 Java および Xalan Java 2 のサイトを参照してください。

注意事項

この節で説明する機能は Apache 独自のものであり，JAXP の規格外となるため，これらの機能を使用した場合の動作は保証しません。これらの機能を使用する場合は，あらかじめ十分な動作確認を実施するようにしてください。

3.5.1 設定できるフィーチャー

● XML パーサに指定できる Apache 独自のフィーチャーを次に示します。

- http://apache.org/xml/features/allow-java-encodings
- http://apache.org/xml/features/continue-after-fatal-error
- http://apache.org/xml/features/disallow-doctype-decl
- http://apache.org/xml/features/dom/create-entity-ref-nodes
- http://apache.org/xml/features/dom/defer-node-expansion
- http://apache.org/xml/features/dom/include-ignorable-whitespace
- http://apache.org/xml/features/generate-synthetic-annotations
- http://apache.org/xml/features/honour-all-schemaLocations
- http://apache.org/xml/features/nonvalidating/load-dtd-grammar
- http://apache.org/xml/features/nonvalidating/load-external-dtd
- http://apache.org/xml/features/scanner/notify-builtin-refs
- http://apache.org/xml/features/scanner/notify-char-refs
- http://apache.org/xml/features/standard-uri-conformant
- http://apache.org/xml/features/validate-annotations
- http://apache.org/xml/features/validation/dynamic
- http://apache.org/xml/features/validation/schema
- http://apache.org/xml/features/validation/schema/augment-psvi
- http://apache.org/xml/features/validation/schema/element-default
- http://apache.org/xml/features/validation/schema/normalized-value
- http://apache.org/xml/features/validation/schema-full-checking
- http://apache.org/xml/features/validation/warn-on-duplicate-attdef

- http://apache.org/xml/features/validation/warn-on-undeclared-elemdef
- http://apache.org/xml/features/warn-on-duplicate-entitydef
- http://apache.org/xml/features/xinclude
- http://apache.org/xml/features/xinclude/fixup-base-uris
- http://apache.org/xml/features/xinclude/fixup-language

● XSLT ／ XSLTC トランスフォーマに指定できる Apache 独自のフィーチャーを次に示します。

- http://xml.apache.org/xalan/features/incremental
- http://xml.apache.org/xalan/features/optimize

3.5.2 設定できるプロパティ

● XML パーサに指定できる Apache 独自のプロパティを次に示します。

- http://apache.org/xml/properties/dom/current-element-node
- http://apache.org/xml/properties/dom/document-class-name
- http://apache.org/xml/properties/input-buffer-size
- http://apache.org/xml/properties/schema/external-noNamespaceSchemaLocation
- http://apache.org/xml/properties/schema/external-schemaLocation
- http://apache.org/xml/properties/security-manager

● XSLT ／ XSLTC トランスフォーマに指定できる Apache 独自のプロパティを次に示します。

- http://xml.apache.org/xalan/properties/source-location

3.5.3 設定できる名前空間 URI

● XSLT ／ XSLTC トランスフォーマで使用できる Apache 独自の名前空間 URI を次に示します。

- http://xml.apache.org/xalan
- http://xml.apache.org/xalan/java
- http://xml.apache.org/xalan/PipeDocument
- http://xml.apache.org/xalan/redirect
- http://xml.apache.org/xalan/sql
- http://xml.apache.org/xalan/xsltc
- http://xml.apache.org/xalan/xsltc/java
- http://xml.apache.org/xslt
- http://xml.apache.org/xslt/java

3.6 XML Schema のエラーメッセージと W3C 仕様との対応

Cosminexus XML Processor が出力するエラーメッセージには，W3C での仕様個所を特定するための識別子が付いています。これは，スキーマ文書での構文エラーおよび XML 文書での検証エラーに対して，一貫したエラー情報を提供するためです。

エラーメッセージに付いている識別子の一覧は，W3C による XML Schema の仕様書，Part1: Structure Appendices C に記載されています。

例えば，スキーマ文書中で要素内のデータ型を整数型（xsd:integer）と規定したにもかかわらず，XML 文書で文字列データを指定した場合は，次のメッセージが出力されます。

```
KECX06036-E cvc-datatype-valid.1.2.1: 'ABC' is not a valid value for 'integer'.
```

このメッセージでは，「cvc-datatype-valid.1.2.1」が識別子となります。これは，「Validation Rule: Datatype Valid」の，1.2.1 に記載された規則に違反していることを示します。

4 フィーチャーおよびプロパティの使用方法

この章では，JAXP および JAXB で規定されているフィーチャーおよびプロパティの使用方法について説明します。また，Cosminexus XML Processor の拡張機能に関連したフィーチャーおよびプロパティの使用方法についても説明します。

4.1 SAX2 のフィーチャーおよびプロパティの使用方法

SAX2 規格では，フィーチャーおよびプロパティという機構によって，SAX パーサのオプション機能を使用する方法を規定しています。

フィーチャー名およびプロパティ名は，SAX パーサのオプション機能に対応した URI です。フィーチャー名，プロパティ名の形式は次のとおりです。

```
フィーチャー名 = http://xml.org/sax/features/ + フィーチャーID
プロパティ名   = http://xml.org/sax/properties/ + プロパティID
```

Cosminexus XML Processor がサポートしている SAX2 のフィーチャーを表 4-1 に，Cosminexus XML Processor がサポートしている SAX2 のプロパティを表 4-2 に示します。

表 4-1 Cosminexus XML Processor がサポートしている SAX2 のフィーチャー一覧

項番	フィーチャー名
1	http://xml.org/sax/features/external-general-entities
2	http://xml.org/sax/features/external-parameter-entities
3	http://xml.org/sax/features/namespaces
4	http://xml.org/sax/features/namespace-prefixes
5	http://xml.org/sax/features/use-attributes2
6	http://xml.org/sax/features/use-locator2
7	http://xml.org/sax/features/use-entity-resolver2
8	http://xml.org/sax/features/validation

表 4-2 Cosminexus XML Processor がサポートしている SAX2 のプロパティ一覧

項番	プロパティ名
1	http://xml.org/sax/properties/declaration-handler
2	http://xml.org/sax/properties/lexical-handler

それぞれのフィーチャー，およびプロパティの意味については，Simple API for XML (SAX) 2.0.2 (sax2r3) の Javadoc の次に示す個所を参照してください。

- org.xml.sax パッケージの "SAX2 Standard Feature Flags"
- org.xml.sax パッケージの "SAX2 Standard Handler and Property IDs"

(1) SAX2 のフィーチャーの使用方法

SAX2 のフィーチャーを設定するには，org.sax.xml.XMLReader クラスの setFeature メソッドを使用します。フィーチャーの設定方法を次に示します。

```
// SAXパーサを生成する
SAXParserFactory factory = SAXParserFactory.newInstance();
SAXParser parser = factory.newSAXParser();

// SAXパーサに含まれるXMLReaderを取得する
XMLReader reader = parser.getXMLReader();

// "validation"フィーチャーを有効に設定する
reader.setFeature("http://xml.org/sax/features/validation", true);
```

(2) SAX2 のプロパティの使用方法

SAX2 のプロパティを設定するには，org.sax.xml.XMLReader クラスの setProperty メソッドを使用します。プロパティの設定方法を次に示します。

```
// SAXパーサを生成する
SAXParserFactory factory = SAXParserFactory.newInstance();
SAXParser parser = factory.newSAXParser();

// SAXパーサに含まれるXMLReaderを取得する
XMLReader reader = parser.getXMLReader();

// プロパティ設定により、DeclHandlerを登録する。
DeclHandler handler = new MyDeclHandler();
reader.setProperty("http://xml.org/sax/properties/declaration-handler", handler);
```

4.2 StAX のプロパティの使用方法

StAX 規格では，XMLInputFactory および XMLOutputFactory にプロパティを設定できます。設定できるプロパティを次に示します。

- javax.xml.stream.isNamespaceAware
- javax.xml.stream.isCoalescing
- javax.xml.stream.isReplacingEntityReferences
- javax.xml.stream.isSupportingExternalEntities
- javax.xml.stream.reporter
- javax.xml.stream.resolver
- javax.xml.stream.allocator
- javax.xml.stream.isRepairingNamespaces

なお，StAX は，Java SE 6 上で動作している場合に使用できます。それぞれのプロパティの意味については，Streaming API For XML Version 1.0(JSR-173) の次に示す個所を参照してください。

- 「Chapter 4.5.1.1. Supported Properties of XMLInputFactory」
- 「Chapter 4.5.2.1 Supported Properties of XMLOutputFactory」
- Javadoc

(1) StAX のプロパティの使用方法

XMLInputFactory に対してプロパティを設定するには，javax.xml.stream.XMLInputFactory クラスの setProperty メソッドを使用します。また，XMLOutputFactory に対してプロパティを設定するには，javax.xml.stream.XMLOutputFactory クラスの setProperty メソッドを使用します。

それぞれのプロパティの設定方法を次に示します。

● XMLInputFactory に対してプロパティを設定する場合

```
XMLInputFactory xif = XMLInputFactory.newInstance();
// 名前空間処理を有効にします。
xif.setProperty("javax.xml.stream.isNamespaceAware", true);
```

● XMLOutputFactory に対してプロパティを設定する場合

```
XMLOutputFactory xof = XMLOutputFactory.newInstance();
//出力側の接頭辞のデフォルトを設定します。
xof.setProperty("javax.xml.stream.isRepairingNamespaces", true);
```

4.3 XSLT のフィーチャーの使用方法

JAXP 規格では，フィーチャーという機構によって，XSLT トランスフォーマのオプション機能を使用する方法を規定しています。

フィーチャー名は，XSLT トランスフォーマのオプション機能に対応した URI です。JAXP で規定されている XSLT フィーチャーを次の表に示します。

表 4-3 JAXP で規定されている XSLT のフィーチャー一覧

項番	フィーチャー名
1	http://javax.xml.transform.stream.StreamSource/feature
2	http://javax.xml.transform.stream.StreamResult/feature
3	http://javax.xml.transform.dom.DOMSource/feature
4	http://javax.xml.transform.dom.DOMResult/feature
5	http://javax.xml.transform.sax.SAXSource/feature
6	http://javax.xml.transform.sax.SAXResult/feature
7	http://javax.xml.transform.sax.SAXTransformerFactory/feature
8	http://javax.xml.transform.sax.SAXTransformerFactory/feature/xmlfilter

それぞれのフィーチャーの意味については，JSR 206 Java API for XML Processing(JAXP) 1.3 の Chapter 8. Package javax.xml.transform，Chapter 9. Package javax.xml.transform.dom，Chapter 10. Package javax.xml.transform.sax の各クラスに記載されている FEATURE フィールドを参照してください。

(1) XSLT のフィーチャーの使用方法

XSLT のフィーチャーは，設定することはできません。XSLT のフィーチャーを参照するには，javax.xml.transform.TransformerFactory クラスの getFeature メソッドを使用します。getFeature の結果はすべて true になります。

4.4 XML Schema のプロパティの使用方法

JAXP で規定されたプロパティを設定すると，スキーマ検証に使用するスキーマ言語，およびスキーマ文書を特定できます。プロパティの詳細については，JSR 206 Java API for XML Processing(JAXP) 1.3 の Chapter 3. Plugability Layer の Properties For Enabling Schema Validation を参照してください。

4.4.1 DOM パーサに対する XML Schema のプロパティの設定方法

DOM パーサに対して XML Schema のプロパティを設定するには，javax.xml.parsers.DocumentBuilderFactory クラスの setAttribute メソッドを使用します。プロパティの設定方法を次に示します。

```
try {
  DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
  dbf.setNamespaceAware(true);
  // 1. 妥当性を検証するように設定します。
  dbf.setValidating(true);
  // 2. スキーマ検証に使用するスキーマ言語を設定します。
  dbf.setAttribute("http://java.sun.com/xml/jaxp/properties/schemaLanguage",
                   "http://www.w3.org/2001/XMLSchema");
  // 3. スキーマ文書を特定します。
  dbf.setAttribute("http://java.sun.com/xml/jaxp/properties/schemaSource",
                   "purchaseOrder.xsd");
  DocumentBuilder db = dbf.newDocumentBuilder();
  Document doc = db.parse("purchaseOrder.xml");
} catch(Exception e) {
  e.printStackTrace();
}
```

コーディング例について次に説明します。各項番はコーディング例のコメントに記述している番号に対応しています。

1. **妥当性を検証するように設定します。**
   javax.xml.parsers.DocumentBuilderFactory クラス，setValidating メソッドの引数に true を設定します。これによって，XML 文書の妥当性検証の機能が有効になります。
2. **スキーマ検証に使用するスキーマ言語を設定します。**
   javax.xml.parsers.DocumentBuilderFactory クラス，setAttribute メソッドで，プロパティ文字列 "http://java.sun.com/xml/jaxp/properties/schemaLanguage" に対して，XML Schema の仕様に従って検証を行うことを示す値，"http://www.w3.org/2001/XMLSchema" を設定します。これによって，XML Schema による妥当性検証が行われます。

3. **スキーマ文書を特定します。**
   javax.xml.parsers.DocumentBuilderFactory クラス，setAttribute メソッドで，プロパティ文字列 "http://java.sun.com/xml/jaxp/properties/schemaSource" に対して，検証に使うスキーマ文書を特定します。XML 文書内でスキーマ文書を特定する場合，この指定は不要です。

4.4.2 SAX パーサに対する XML Schema のプロパティの設定方法

SAX パーサに対して XML Schema のプロパティを設定するには，javax.xml.parsers.SAXParser クラスの setProperty メソッドを使用します。プロパティの設定方法を次に示します。

```
try {
  SAXParserFactory spf = SAXParserFactory.newInstance();
  spf.setNamespaceAware(true);
  // 1. 妥当性を検証するように設定します。
  spf.setValidating(true);
  SAXParser sp = spf.newSAXParser();
  // 2. スキーマ検証に使用するスキーマ言語を設定します。
  sp.setProperty("http://java.sun.com/xml/jaxp/properties/schemaLanguage",
                 "http://www.w3.org/2001/XMLSchema");
  // 3.スキーマ文書を特定します。
  sp.setProperty("http://java.sun.com/xml/jaxp/properties/schemaSource",
                 "purchaseOrder.xsd");
  XMLReader reader = sp.getXMLReader();
  reader.parse("purchaseOrder.xml");
} catch(Exception e) {
  e.printStackTrace();
}
```

コーディング例について次に説明します。各項番はコーディング例のコメントに記述している番号に対応しています。

1. **妥当性を検証するように設定します。**
   javax.xml.parsers.SAXParserFactory クラス，setValidating メソッドの引数に true を設定します。これによって，XML 文書の妥当性検証の機能が有効になります。

2. **XML Schema の仕様に従うように設定します。**
   javax.xml.parsers.SAXParser クラス，setProperty メソッドで，プロパティ文字列 "http://java.sun.com/xml/jaxp/properties/schemaLanguage" に対して，XML Schema の仕様に従って検証を行うことを示す値，"http://www.w3.org/2001/XMLSchema" を設定します。これによって，XML Schema による妥当性検証が行われます。

3. **スキーマ文書を特定します。**
   javax.xml.parsers.SAXParser クラス，setProperty メソッドで，プロパティ文字列 "http://java.sun.com/xml/jaxp/properties/schemaSource" に対して，検証に使うス

キーマ文書を指定します。XML 文書内でスキーマ文書を特定する場合，この指定は不要です。

4.4.3 スキーマ文書を XML 文書内で設定する方法

「4.4.1 DOM パーサに対する XML Schema のプロパティの設定方法」，および「4.4.2 SAX パーサに対する XML Schema のプロパティの設定方法」に示した方法では，プロパティを用いてスキーマ文書を設定しています。これに対して，検証の対象となる XML 文書内で，直接スキーマ文書を特定する方法もあります。なお，XML 文書内でスキーマ文書を特定し，かつ，プロパティを用いてスキーマ文書を特定した場合は，プロパティを使って特定したスキーマ文書が優先されます。

検証の対象となる XML 文書内でスキーマ文書を特定するには，xsi:schemaLocation 属性または xsi:noNamespaceSchemaLocation 属性を使用します。これらの属性は，特定するスキーマ文書内で，名前空間を定義しているかどうかによって異なります。特定するスキーマ文書の種類と使用する属性の関係を次の表に示します。

表 4-4 特定するスキーマ文書の種類と使用する属性の関係

特定するスキーマ文書の種類	使用する属性
名前空間を定義したスキーマ文書	xsi:schemaLocation
名前空間を定義していないスキーマ文書	xsi:noNamespaceSchemaLocation

xsi:schemaLocation 属性を使用する場合，特定するスキーマ文書内で宣言されている要素や属性の名前空間の URI と，特定するスキーマ文書名を空白で区切って指定します。

xsi:schemaLocation 属性，および xsi:noNamespaceSchemaLocation 属性は，どちらも名前空間 "http://www.w3.org/2001/XMLSchema-instance" に属します。したがって，これらの属性を使ってスキーマ文書を特定する場合，名前空間 "http://www.w3.org/2001/XMLSchema-instance" を宣言する必要があります。

xsi:schemaLocation 属性を使用してスキーマ文書を特定する場合の例を次に示します。

```
<? xml version="1.0"?>
<po:purchaseOrder xmlns:po=http://www.myshopping.com/schema/purchaseOrder
xmlns:psd="http://www.myshopping.com/schema/personalData"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"          …1.
xsi:schemaLocation="http://www.myshopping.com/schema/purchaseOrder
purchaseOrder.xsd">                                            …2.
      :
      :
</po:purchaseOrder>
```

1. xsi:schemaLocation 属性の名前空間を宣言します。
   xsi:schemaLocation 属性を使ってスキーマ文書を特定するために，この属性の名前空

間 "http://www.w3.org/2001/XMLSchema-instance" を宣言します。

2. **スキーマ文書を特定します。**
   特定するスキーマ文書内で宣言されている要素や属性の名前空間の URI，およびスキーマ文書名を空白で区切って指定します。
   上記の例では，スキーマ文書として purchaseOrder.xsd を，スキーマ文書 purchaseOrder.xsd 内で宣言されている要素や属性の名前空間として "http://www.myshopping.com/schema/purchaseOrder" を指定しています。

4.5 Shift_JIS 切り替え機能のプロパティの使用方法

Shift_JIS 切り替え機能を使用するためには，Cosminexus XML Processor が独自に規定する次のプロパティを設定する必要があります。

```
プロパティ名 = http://cosminexus.com/xml/properties/shift_jis_map
```

プロパティの値には，文字エンコーディングを設定します。設定するプロパティの値とその意味を次の表に示します。

表 4-5 設定するプロパティの値とその意味（Shift_JIS 切り替え機能）

プロパティに設定する値※1	意味
SJIS	SJIS エンコーディングを適用する。
MS932	MS932 エンコーディングを適用する。
その他の文字列	指定を無視する。※2

注※ 1

指定された文字列は，大文字・小文字を区別しません。

注※ 2

指定が無視された場合は，直前に指定されたエンコーディングが有効となります。

注意事項

Shift_JIS 切り替え機能の指定がない場合，デフォルトでは SJIS エンコーディングとなります。

Shift_JIS 切り替え機能のプロパティは，次に示すクラス，インタフェースのインスタンスごとに設定する必要があります。Shift_JIS 切り替え機能の設定用メソッドを次の表に示します。次の表では，プロパティの設定に使用するメソッドを，クラス，インタフェースごとに示しています。

表 4-6 Shift_JIS 切り替え機能の設定用メソッド

クラス，インタフェース	メソッド
javax.xml.parsers.DocumentBuilderFactory	setAttribute
javax.xml.parsers.SAXParser	setProperty
org.xml.sax.XMLReader	setProperty
javax.xml.parsers.TransformerFactory	setAttribute

DOM パーサ，SAX パーサ，および XSLT トランスフォーマで Shift_JIS 切り替え機能を使用する場合について，例を用いて説明します。

4.5.1 DOM パーサで Shift_JIS 切り替え機能を使用する方法

DOM パーサで Shift_JIS 切り替え機能を使用する基本的なコーディングの流れについて説明します。

```
// 1. DocumentBuilderFactoryを作成します。
javax.xml.parsers.DocumentBuilderFactory factory =
            javax.xml.parsers.DocumentBuilderFactory.newInstance();
// 2. Shift_JISをMS932に切り替えます。
factory.setAttribute("http://cosminexus.com/xml/properties/shift_jis_map","MS932");
// 3. DocumentBuilderを作成します。
javax.xml.parsers.DocumentBuilder db = factory.newDocumentBuilder();
```

コーディング例について次に説明します。各項番はコーディング例のコメントに記述している番号に対応しています。

1. **DocumentBuilderFactory を作成します。**
   javax.xml.parsers.DocumentBuilderFactory の static メソッド newInstance() を使用してファクトリを作成します。
2. **Shift_JIS を MS932 に切り替えます。**
   DOM パーサの設定用メソッド setAttribute の第一引数に設定キー http://cosminexus.com/xml/properties/shift_jis_map を，第二引数に MS932 を指定します。
3. **DocumentBuilder を作成します。**
   newDocumentBuilder() メソッドを使用して，DocumentBuilder を作成します。

4.5.2 SAX パーサで Shift_JIS 切り替え機能を使用する方法

SAX パーサで Shift_JIS 切り替え機能を使用する基本的なコーディングの流れについて説明します。コーディング例を次に示します。

```
// 1. SAXParserFactoryを作成します。
javax.xml.parsers.SAXParserFactory factory =
                    javax.xml.parsers.SAXParserFactory.newInstance();
// 2. SAXParserを作成します。
javax.xml.parsers.SAXParser parser = factory.newSAXParser();
// 3. XMLReaderを取得します。
org.xml.sax.XMLReader reader = parser.getXMLReader();
// 4. Shift_JISをMS932に切り替えます。
reader.setProperty("http://cosminexus.com/xml/properties/shift_jis_map","MS932");
```

コーディング例について次に説明します。各項番はコーディング例のコメントに記述している番号に対応しています。

1. **SAXParserFactory を作成します。**
   javax.xml.parsers.SAXParserFactory の static メソッド newInstance() を使用してファクトリを作成します。
2. **SAXParser を作成します。**
   newSAXParser() を使用して SAX パーサを作成します。
3. **XMLReader を取得します。**
   getXMLReader() を使用して XMLReader を取得します。
4. **Shift_JIS を MS932 に切り替えます。**
   SAX パーサの設定用メソッド setProperty の第一引数に設定キー http://cosminexus.com/xml/properties/shift_jis_map を，第二引数に MS932 を指定します。

4.5.3 XSLT トランスフォーマで Shift_JIS 切り替え機能を使用する方法

XSLT トランスフォーマで Shift_JIS 切り替え機能を使用する基本的なコーディングの流れについて説明します。コーディング例を次に示します。

```
// 1. TransformerFactoryを作成します。
javax.xml.transform.TransformerFactory factory =
              javax.xml.transform.TransformerFactory.newInstance();
// 2. Shift_JISをMS932に切り替えます。
factory.setAttribute("http://cosminexus.com/xml/properties/shift_jis_map","MS932");
// 3. Transformerを作成します。
factory.newTransformer();
```

コーディング例について次に説明します。各項番はコーディング例のコメントに記述している番号に対応しています。

1. **TransformerFactory を作成します。**
   javax.xml.transform.TransformerFactory の static メソッド newInstance() を使用してファクトリを作成します。
2. **Shift_JIS を MS932 に切り替えます。**
   XSLT トランスフォーマファクトリの設定用メソッド setAttribute の第一引数に設定キー http://cosminexus.com/xml/properties/shift_jis_map を，第二引数に MS932 を指定します。
3. **Transformer を作成します。**
   newTransformer() メソッドを使用して，トランスフォーマを作成します。

4.6　高速パース機能のプロパティの使用方法

高速パース機能を使用するためには，Cosminexus XML Processor が独自に規定する次のプロパティを設定する必要があります。

```
プロパティ名 = http://cosminexus.com/xml/properties/preparsedobject-load
```

プロパティの値には，null または PreparsedObject オブジェクトを設定します。設定するプロパティの値とその意味を次の表に示します。

表 4-7　設定するプロパティの値とその意味（高速パース機能）

プロパティに設定する値※	意味
null	解析時に解析結果オブジェクトを使用しない。
PreparsedObject オブジェクト	解析時に，指定した解析結果オブジェクトを使用する。

注※

指定が省略された場合は，null が指定されたものとして扱います。

また，null および PreparsedObject オブジェクト以外の値が設定された場合は，プロパティが設定されている XML パーサによって，次の例外が発生します。

- DocumentBuilderFactory の場合：IllegalArgumentException
- SAXParser または XMLReader の場合：SAXNotSupportedException

高速パース機能のプロパティは，次に示す XML パーサのオブジェクトごとに設定する必要があります。解析結果オブジェクトの設定用メソッドを次の表に示します。次の表では，プロパティの設定に使用するメソッドを，クラス，インタフェースごとに示しています。

表 4-8　解析結果オブジェクトの設定用メソッド

クラス，インタフェース	メソッド
javax.xml.parsers.DocumentBuilderFactory	setAttribute
javax.xml.parsers.SAXParser	setProperty
org.xml.sax.XMLReader	setProperty

DocumentBuilder，SAXParser，および XMLReader で高速パース機能を使用する場合について，例を用いて説明します。

4.6.1 DocumentBuilder に解析結果オブジェクトを設定する方法

DocumentBuilder に解析結果オブジェクトを設定する場合のコーディング例を次に示します。

```
// 1. 解析結果オブジェクトを生成します。
File xml = new File(...);
PreparsedObjectFactory pof = PreparsedObjectFactory.newInstance();
pof.setNamespaceAware(...);
pof.setEntityResolver(...);
pof.setErrorHandler(...);
PreparsedObject pobj = pof.newPreparsedObject(xml);

// 2. DocumentBuilderに解析結果オブジェクトを設定します。
DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
dbf.setAttribute("http://cosminexus.com/xml/properties/preparsedobject-load", pobj);
DocumentBuilder db = dbf.newDocumentBuilder();
```

コーディング例について次に説明します。各項番はコーディング例のコメントに記述している番号に対応しています。

1. **解析結果オブジェクトを生成します。**
   解析結果オブジェクトを生成する方法については，「3.4.4　解析結果オブジェクトの生成」を参照してください。
2. **DocumentBuilder に解析結果オブジェクトを設定します。**
   解析結果オブジェクトの設定用メソッド setAttribute の第一引数に設定キー http://cosminexus.com/xml/properties/preparsedobject-load を，第二引数に PreparsedObject オブジェクトを示す pobj を指定します。

4.6.2 SAXParser に解析結果オブジェクトを設定する方法

SAXParser に解析結果オブジェクトを設定する場合のコーディング例を次に示します。

```
// 1. 解析結果オブジェクトを生成します。
File xml = new File(...);
PreparsedObjectFactory pof = PreparsedObjectFactory.newInstance();
pof.setNamespaceAware(...);
pof.setEntityResolver(...);
pof.setErrorHandler(...);
PreparsedObject pobj = pof.newPreparsedObject(xml);

// 2. SAXParserに解析結果オブジェクトを設定します。
SAXParserFactory spf = SAXParserFactory.newInstance();
SAXParser sp = spf.newSAXParser();
sp.setProperty("http://cosminexus.com/xml/properties/preparsedobject-load", pobj);
```

1. **解析結果オブジェクトを生成します。**
   解析結果オブジェクトを生成する方法については，「3.4.4　解析結果オブジェクトの生成」を参照してください。

2. SAXParser に解析結果オブジェクトを設定します。
   解析結果オブジェクトの設定用メソッド setProperty の第一引数に設定キー http://cosminexus.com/xml/properties/preparsedobject-load を，第二引数に PreparsedObject オブジェクトを示す pobj を指定します。

4.6.3　XMLReader に解析結果オブジェクトを設定する方法

XMLReader に解析結果オブジェクトを設定する場合のコーディング例を次に示します。

```
// 1. 解析結果オブジェクトを生成します。
File xml = new File(...);
PreparsedObjectFactory pof = PreparsedObjectFactory.newInstance();
pof.setNamespaceAware(...);
pof.setEntityResolver(...);
pof.setErrorHandler(...);
PreparsedObject pobj = pof.newPreparsedObject(xml);

// 2. XMLReaderに解析結果オブジェクトを設定します。
SAXParserFactory spf = SAXParserFactory.newInstance();
SAXParser sp = spf.newSAXParser();
XMLReader reader = sp.getXMLReader();
reader.setProperty("http://cosminexus.com/xml/properties/preparsedobject-load", pobj);
```

1. 解析結果オブジェクトを生成します。
   解析結果オブジェクトを生成する方法については，「3.4.4　解析結果オブジェクトの生成」を参照してください。
2. XMLReader に解析結果オブジェクトを設定します。
   解析結果オブジェクトの設定用メソッド setProperty の第一引数に設定キー http://cosminexus.com/xml/properties/preparsedobject-load を，第二引数に PreparsedObject オブジェクトを示す pobj を指定します。

4.7 JAXB のプロパティの使用方法

JAXB では Marshaller にプロパティを設定できます。Cosminexus XML Processor がサポートしている JAXB のプロパティを次に示します。

Cosminexus XML Processor がサポートしている JAXB のプロパティ

- jaxb.encoding
- jaxb.formatted.output
- jaxb.schemaLocation
- jaxb.noNamespaceSchemaLocation
- jaxb.fragment

プロパティの詳細については, JSR 222 The Java Architecture for XML Binding(JAXB) 2.2 の Chapter 4.5.2. Marshalling Properties および javadoc を参照してください。

また，jaxb.encoding に指定できる文字コードについては，「1.3.2　処理できる文字コード」を参照してください。

4.7.1 JAXB のプロパティの使用方法

Marshaller に対してプロパティを設定するには，javax.xml.bind.Marshaller クラスの setProperty メソッドを使用します。プロパティの設定方法を次に示します。

```
// マーシャルするコンテントツリーを生成します。
JAXBContext jc = JAXBContext.newInstance("com.acme.foo");
Unmarshaller u = jc.createUnmarshaller();
Object topelement = u.unmarshal( new File( "foo.xml" ) );
// Marshallerを生成します。
Marshaller m = jc.createMarshaller();
// 出力エンコーディングにShift_JISを指定します。
m.setProperty(Marshaller.JAXB_ENCODING, "Shift_JIS");
// 改行とインデントを使用して結果のXML データを書式設定します。
m.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, Boolean.TRUE);
// 出力するXML データの xsi:schemaLocation 属性を指定します。
m.setProperty(Marshaller.JAXB_SCHEMA_LOCATION, "http://foo/ns fooSchema.xsd");
// コンテントツリーをマーシャルします。
m.marshal(topelement, new FileOutputStream("outputFile.xml"));
```

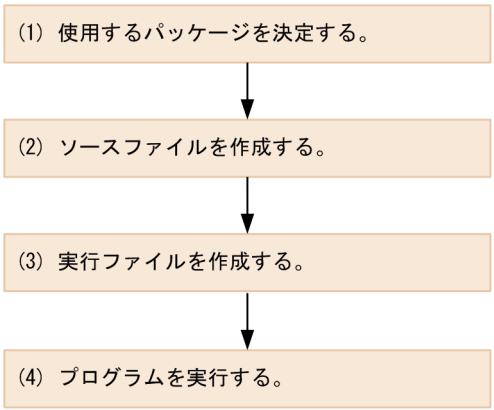
5 プログラムの作成方法

この章では，プログラム作成の流れ，使用するパッケージ，および JAXP の主な機能を用いたサンプルプログラムについて説明します。

5.1　プログラム作成の流れ

Cosminexus XML Processor を使用したプログラムの作成の流れを次の図に示します。

図 5-1　Cosminexus XML Processor を使用したプログラム作成の流れ



(1) 使用するパッケージを決定する

Cosminexus XML Processor は，JAXP 仕様で規定されたパッケージ，XSLTC トランスフォーマ機能のパッケージ，および JAXB 仕様で規定されたパッケージを提供しています。プログラムを作成する際には，そのプログラムの用途に適したパッケージを選択してください。

例えば，読み込んだ XML 文書をメモリ上に保持し，その構造を変更したり，複雑な操作をしたりする必要がある場合は DOM を，対話的に XML 文書を解析しながら処理をする場合は SAX を使用します。また，XML 文書を HTML などに変換する場合は，XSLT を使用します。

JAXP 仕様で規定されたパッケージについては「2.2　JAXP が規定するパッケージとその機能」を，XSLTC トランスフォーマ機能のパッケージについては「3.2.3　XSLTC トランスフォーマで使用するクラス」を，JAXB 仕様で規定されたパッケージについては「2.4　JAXB が規定するパッケージとその機能」を参照してください。

(2) ソースファイルを作成する

使用するパッケージの決定後，それぞれのパッケージに含まれるクラスやインタフェースを使用して，ソースファイルを作成します。

(3) 実行ファイルを作成する

作成したソースファイルから実行ファイルを作成します。

実行ファイルを作成するには，作成したソースファイルを javac コマンドでコンパイル

し，使用するクラスやインタフェースに必要なライブラリをリンクする必要があります。

(4) プログラムを実行する

コンパイルしたプログラムを実行します。

5.2 Cosminexus XML Processor が使用しているパッケージ名

Cosminexus XML Processor は，次の名称で始まるパッケージ名を使用しています。

- com.cosminexus.jaxp.
- com.cosminexus.jaxb.
- com.cosminexus.stax.

注意事項

ユーザプログラムのパッケージ名は，「com.cosminexus.jaxp.」，「com.cosminexus.jaxb.」および「com.cosminexus.stax.」で始まらないようにしてください。

5.3 プログラム実行時のトラブルシュート

Cosminexus XML Processor を使用したシステムを運用する場合，システム管理者への連絡が必要になるような障害や，対処方法が不明な障害が発生した場合の障害調査資料として，アプリケーションログには次の情報を採取しておく必要があります。

- 障害発生時時刻
- 障害発生時の入力となった XML 文書名（スキーマ文書名，スタイルシート名含む）

上記のアプリケーションログと同時に，次に示す障害情報を取得した上，システム管理者に連絡してください。

- アプリケーションログファイル
- 障害発生時の入力となった XML 文書（スキーマ文書，スタイルシート含む）
- 障害発生時に出力された Cosminexus XML Processor のエラーメッセージ
- サーバ，およびクライアントに設定したシステムプロパティとシステムクラスパス
- サーバ，およびクライアントの標準出力と標準エラー出力
- Cosminexus Component Container，および Web サーバのログ
- Cosminexus Component Container で規定する障害時取得情報
  Cosminexus Component Container で規定する障害時取得情報を次に示します。
  - J2EE サーバのユーザ定義ファイル
  - J2EE サーバの保守情報
  - サーバ管理コマンドのユーザ定義ファイル
  - サーバ管理コマンドの保守情報
  - J2EE サーバの標準出力，標準エラー出力
  - J2EE サーバ側の Cosminexus TPBroker のログ
  - EJB クライアントアプリケーションに設定したシステムプロパティ，およびシステムクラスパス
  - EJB クライアントアプリケーションの標準出力，標準エラー出力
  - ネットワークモニタツールで採取したパケットの内容

Cosminexus Component Container で規定する障害時取得情報の取得方法については，マニュアル「Cosminexus アプリケーションサーバ 機能解説 保守／移行／互換編」の「2.3　資料の取得」を参照してください。

5.4　DOM パーサを使用するサンプルプログラム

この節では，DOM パーサを使用するサンプルプログラムについて説明します。

サンプルプログラムのインストール先，ファイル名は次のとおりです。

インストール先

Windows の場合

＜ Cosminexus XML Processor をインストールしたディレクトリ＞¥samples¥DOM

UNIX の場合

＜ Cosminexus XML Processor をインストールしたディレクトリ＞ /samples/DOM

ファイル名

SampleDOM.java

5.4.1　サンプルプログラムの処理

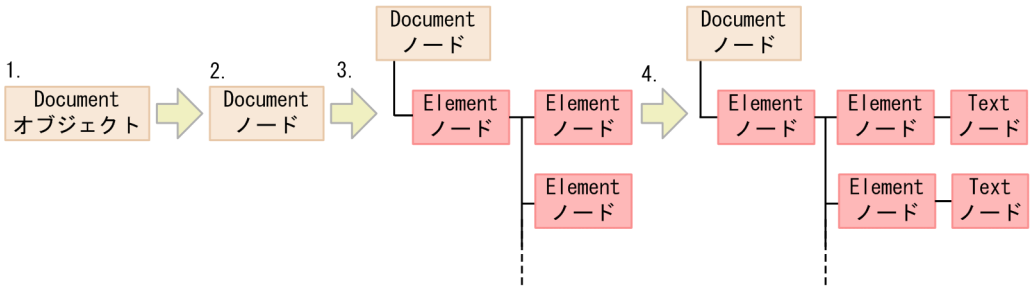
サンプルプログラムは次のような処理を行います。

1. **商品の情報について記述した** DOM **ツリーを新規に作成します。**
2. **作成した** DOM **ツリーを** XML **文書に変換・出力します。**
3. **出力した** XML **文書を読み込みます。**
4. **読み込んだ** XML **文書から，商品の合計金額を画面に出力します。**

5.4.2　サンプルプログラムの作成の流れ

DOM パーサを使用するサンプルプログラムの作成の流れを次の図に示します。

図 5-2　DOM **パーサを使用するサンプルプログラムの作成の流れ**



1. Document **オブジェクトを作成します。**
   Document オブジェクトを新規に作成します。
2. Document **ノードを作成します。**
   1. で作成した Document オブジェクトを使用して DOM ツリーの親となる Document ノードを作成します。
3. Element **ノードを作成します。**
   必要となる要素ノードを Document ノードから作成し，DOM ツリーを作成します。
4. Text **ノードを作成します。**
   要素の内容となる文字列を作成します。

5.4.3 サンプルプログラム（SampleDOM.java）

サンプルプログラムを次に示します。

```
import java.io.*;
import javax.xml.parsers.*;
import javax.xml.transform.*;
import javax.xml.transform.dom.*;
import javax.xml.transform.stream.*;
import org.xml.sax.SAXException;
import org.w3c.dom.*;

public class SampleDOM
{
  public static final void main(String[] args)
  {
    try{
      new SampleDOM().CreateXMLFile();
      new SampleDOM().GetStringFromXML();
    }catch(Exception exception){
      exception.printStackTrace();
    }
  }

  //XMLファイルを作成する
  public final void CreateXMLFile()
  {
    try{
      DocumentBuilderFactory dbf =
          DocumentBuilderFactory.newInstance();
      DocumentBuilder db = dbf.newDocumentBuilder();

      Document doc = db.newDocument();
      Element parent = doc.createElement("伝票");

      Element sub = doc.createElement("商品");
      Element leaf1 = doc.createElement("商品名");
      Element leaf2 = doc.createElement("単価");
      Element leaf3 = doc.createElement("数量");
      leaf1.appendChild(doc.createTextNode("テレビ"));
```

```
        sub.appendChild(leaf1);
        leaf2.appendChild(doc.createTextNode("15000"));
        sub.appendChild(leaf2);
        leaf3.appendChild(doc.createTextNode("14"));
        sub.appendChild(leaf3);
        parent.appendChild(sub);

        sub = doc.createElement("商品");
        leaf1 = doc.createElement("商品名");
        leaf2 = doc.createElement("単価");
        leaf3 = doc.createElement("数量");
        leaf1.appendChild(doc.createTextNode("ラジオ"));
        sub.appendChild(leaf1);
        leaf2.appendChild(doc.createTextNode("3500"));
        sub.appendChild(leaf2);
        leaf3.appendChild(doc.createTextNode("6"));
        sub.appendChild(leaf3);
        parent.appendChild(sub);

        sub = doc.createElement("商品");
        leaf1 = doc.createElement("商品名");
        leaf2 = doc.createElement("単価");
        leaf3 = doc.createElement("数量");
        leaf1.appendChild(doc.createTextNode("ビデオ"));
        sub.appendChild(leaf1);
        leaf2.appendChild(doc.createTextNode("21000"));
        sub.appendChild(leaf2);
        leaf3.appendChild(doc.createTextNode("4"));
        sub.appendChild(leaf3);
        parent.appendChild(sub);

        doc.appendChild(parent);

        OutputStream os =
            new BufferedOutputStream(
            new FileOutputStream("SampleDOM.xml"));
        TransformerFactory tf =
            TransformerFactory.newInstance();
        Transformer tran = tf.newTransformer();
        tran.setOutputProperty("encoding", "Shift_JIS");
        tran.setOutputProperty("indent", "yes");
        tran.transform(new DOMSource(doc),
            new StreamResult(os));

        os.flush();
        os.close();

      }catch(ParserConfigurationException e){
        e.printStackTrace();
      }catch(TransformerConfigurationException e){
        e.printStackTrace();
      }catch(TransformerException e){
        e.printStackTrace();
      }catch(IOException e){
        e.printStackTrace();
      }
    }
```

```
//CreateXMLFileで作成したXMLファイルからデータを取得して計算する
public final void GetStringFromXML()
{
  try{
    int tvPrice = 0;
    int radioPrice = 0;
    int videoPrice = 0;

    DocumentBuilderFactory factory =
        DocumentBuilderFactory.newInstance();
    DocumentBuilder builder = factory.newDocumentBuilder();

    Document doc = builder.parse("SampleDOM.xml");
    Element element1 = doc.getDocumentElement();
    NodeList nodelist1 = element1.getElementsByTagName("商品");

    //商品の数だけループをまわす
    for(int nShouhin = 0 ;
        nShouhin < nodelist1.getLength() ; nShouhin++){
      String str = new java.lang.String();
      int price = 0;
      int number = 0;
      Element element2 = (Element)nodelist1.item(nShouhin);

      //商品名の取得
      NodeList nodelist2 =
          element2.getElementsByTagName("商品名");
      Node node1 = nodelist2.item(0);
      if(node1.getNodeType() == Node.ELEMENT_NODE){
        Element element3 = (Element)node1;
        NodeList nodelist3 = element3.getChildNodes();
        Node node2 = nodelist3.item(0);
        str = node2.getNodeValue();
      }

      //商品名に対する単価の取得
      nodelist2 = element2.getElementsByTagName("単価");
      node1 = nodelist2.item(0);
      if(node1.getNodeType() == Node.ELEMENT_NODE){
        Element element3 = (Element)node1;
        NodeList nodelist3 = element3.getChildNodes();
        Node node2 = nodelist3.item(0);
        price = Integer.parseInt(node2.getNodeValue());
      }

      //商品名に対する数量の取得
      nodelist2 = element2.getElementsByTagName("数量");
      node1 = nodelist2.item(0);
      if(node1.getNodeType() == Node.ELEMENT_NODE){
        Element element3 = (Element)node1;
        NodeList nodelist3 = element3.getChildNodes();
        Node node2 = nodelist3.item(0);
        number = Integer.parseInt(node2.getNodeValue());
      }

      //各商品の小計を計算する
      if(str.equals("テレビ")){
        tvPrice = price * number;
```

```
          }else if(str.equals("ラジオ")){
            radioPrice = price * number;
          }else if(str.equals("ビデオ")){
            videoPrice = price * number;
          }
        }
        int sum = tvPrice + radioPrice + videoPrice;
        System.out.println("合計は" + sum + "円です。");
      }catch(ParserConfigurationException e){
        e.printStackTrace();
      }catch(SAXException e){
        e.printStackTrace();
      }catch(IOException e){
        e.printStackTrace();
      }
    }
}
```

5.4.4 サンプルプログラム（SampleDOM.java）の実行結果

このサンプルプログラムの実行結果は標準出力，および SampleDOM.xml に出力されます。

標準出力の内容

```
$  合計は315000円です。
```

SampleDOM.xml に出力される内容

```
<?xml version="1.0" encoding="Shift_JIS"?>
<伝票>
<商品>
<商品名>テレビ</商品名>
<単価>15000</単価>
<数量>14</数量>
</商品>
<商品>
<商品名>ラジオ</商品名>
<単価>3500</単価>
<数量>6</数量>
</商品>
<商品>
<商品名>ビデオ</商品名>
<単価>21000</単価>
<数量>4</数量>
</商品>
</伝票>
```

5.5 SAX パーサを使用するサンプルプログラム

この節では，SAX パーサを使用するサンプルプログラムについて説明します。

サンプルプログラムのインストール先，ファイル名は次のとおりです。

インストール先

Windows の場合

＜ Cosminexus XML Processor をインストールしたディレクトリ＞¥samples¥SAX

UNIX の場合

＜ Cosminexus XML Processor をインストールしたディレクトリ＞ /samples/SAX

ファイル名

- SampleSAX.xml
- SampleSAX.java

5.5.1 サンプルプログラムの処理

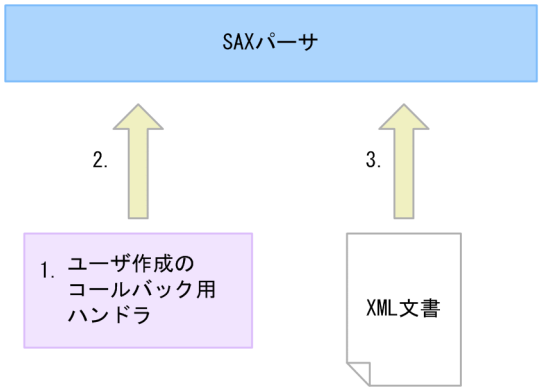
サンプルプログラムは次のような処理を行います。

1. **商品の情報について記述した** XML **文書を読み込みます。**
2. **読み込んだ** XML **文書から，商品の合計金額を画面に出力します。**

5.5.2 サンプルプログラムの作成の流れ

SAX パーサを使用するサンプルプログラムの作成の流れを次の図に示します。

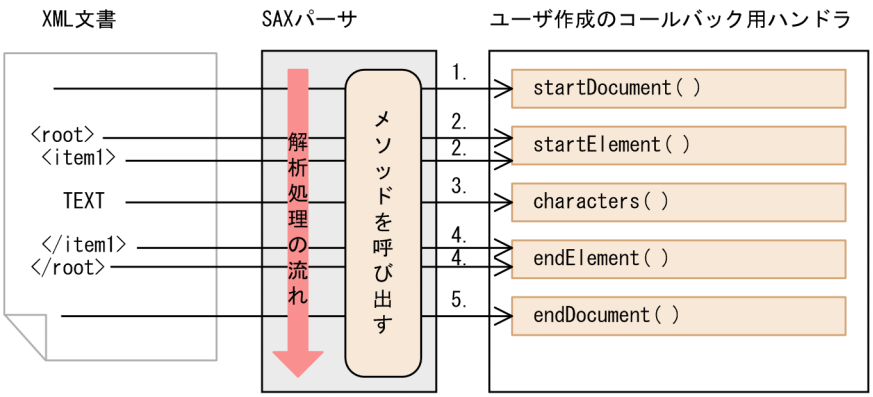
図 5-3 SAX パーサを使用するサンプルプログラムの作成の流れ



1. **コールバック用ハンドラを作成します。**
   SAX パーサを使用して解析を開始すると，SAX パーサから XML 文書内の情報が通知されます。これを受け取るための，コールバック用ハンドラを作成します。
2. **コールバック用ハンドラを SAX パーサに登録します。**
   1. で作成したコールバック用ハンドラを SAX パーサに登録します。
3. **XML 文書を SAX パーサに渡して解析します。**
   XML 文書を解析します。

   SAX パーサが呼び出すメソッド

   SAX パーサが呼び出すメソッドを次の図に示します。



1. startDocumentメソッド
   XML文書の解析開始時に，startDocumentメソッドが呼び出されます。
   このメソッド内に必要な初期化処理を記述してください。

2. startElementメソッド
   解析中に要素の開きタグが見つかるたびに，startElementメソッドが呼び出されます。

3. charactersメソッド
   XML文書内にタグに囲まれたテキストが見つかった場合，charactersメソッドが呼び出され，文字列を取得します。

4. endElementメソッド
   解析中に閉じタグが見つかるたびに，endElementメソッドが呼び出されます。

5. endDocumentメソッド
   解析終了時に，endDocumentメソッドが呼び出されます。

5.5.3　使用する XML 文書（SampleSAX.xml）

使用する XML 文書を次に示します。

```
<?xml version="1.0" encoding="Shift_JIS"?>
<伝票>
  <商品>
    <商品名>テレビ</商品名>
    <単価>15000</単価>
    <数量>14</数量>
  </商品>
  <商品>
    <商品名>ラジオ</商品名>
    <単価>3500</単価>
    <数量>6</数量>
  </商品>
  <商品>
    <商品名>ビデオ</商品名>
    <単価>21000</単価>
    <数量>4</数量>
  </商品>
</伝票>
```

5.5.4 サンプルプログラム（SampleSAX.java）

サンプルプログラムを次に示します。

```
import javax.xml.parsers.*;
import org.xml.sax.*;
import org.xml.sax.helpers.*;
import org.xml.sax.SAXException;
import java.io.*;

public class SampleSAX{

  public static final void main(String[] args){
    try{
      SAXParserFactory factory = SAXParserFactory.newInstance();
      SAXParser parser = factory.newSAXParser();
      String xml = "SampleSAX.xml";
      DefaultHandler handler = new EventHandler();

      parser.parse(xml, handler);
    }catch(ParserConfigurationException e){
      e.printStackTrace();
    }catch(IOException e){
      e.printStackTrace();
    }catch(SAXException e){
      e.printStackTrace();
    }
  }
}

//合計金額を計算して出力する
class EventHandler extends DefaultHandler{

  int     tvPrice;         //テレビ単価
  int     radioPrice;      //ラジオ単価
  int     videoPrice;      //ビデオ単価
  int     tvNum;           //テレビ数量
  int     radioNum;        //ラジオ数量
  int     videoNum;        //ビデオ数量
  boolean tvFlag;          //テレビ用フラグ
  boolean radioFlag;       //ラジオ用フラグ
  boolean videoFlag;       //ビデオ用フラグ
  boolean tankaFlag;       //単価タグ内用フラグ
  boolean volFlag;         //数量タグ内用フラグ
  boolean nameFlag;        //商品名タグ内用フラグ

  String str;              //要素内容の文字列
  StringBuffer buffer;     //要素内容の文字列連結用バッファ

  //XMLの開始通知イベント
  public void startDocument(){
    tvPrice = 0;
    radioPrice = 0;
    videoPrice = 0;
    tvNum = 0;
    radioNum = 0;
```

```
  videoNum = 0;
  tvFlag = false;
  radioFlag = false;
  videoFlag = false;
  tankaFlag = false;
  volFlag = false;
  nameFlag = false;
}

//XMLの終了通知イベント
public void endDocument(){
  int sum = tvNum * tvPrice + radioNum * radioPrice +
      videoNum * videoPrice;
  System.out.println("合計は" + sum + "円です。");
}

//エレメント開始(タグの開始)通知イベント
public void startElement(String nameSpace, String localName,
    String modName, Attributes attr){
  buffer = new StringBuffer();
  if(modName.equals("単価")){
    tankaFlag = true;
  }
  if(modName.equals("数量")){
    volFlag = true;
  }
  if(modName.equals("商品名")){
    nameFlag = true;
  }
}

//エレメント終了(タグの終了)通知イベント
public void endElement(String nameSpace, String localName,
    String modName){
  str = buffer.toString();
  if(nameFlag == true){
    if(str.equals("テレビ")){
      tvFlag = true;
    }
    if(str.equals("ラジオ")){
      radioFlag = true;
    }
    if(str.equals("ビデオ")){
      videoFlag = true;
    }
  }else if(tvFlag == true){
    if(tankaFlag == true){
      //テレビの単価をセット
      tvPrice = Integer.parseInt(str);
    }else if(volFlag == true){
      //テレビの数量をセット
      tvNum = Integer.parseInt(str);
    }
  }else if(radioFlag == true){
    if(tankaFlag == true){
      //ラジオの単価をセット
      radioPrice = Integer.parseInt(str);
```

```
        }else if(volFlag == true){
          //ラジオの数量をセット
          radioNum = Integer.parseInt(str);
        }
      }else if(videoFlag == true){
        if(tankaFlag == true){
          //ビデオの単価をセット
          videoPrice = Integer.parseInt(str);
        }else if(volFlag == true){
          //ビデオの数量をセット
          videoNum = Integer.parseInt(str);
        }
      }
      if(modName.equals("単価")){
        tankaFlag = false;
      }
      if(modName.equals("数量")){
        volFlag = false;
      }
      if(modName.equals("商品名")){
        nameFlag = false;
      }
      if(modName.equals("商品")){
        tvFlag = false;
        radioFlag = false;
        videoFlag = false;
      }
    }

    //文字列通知イベント
    public void characters(char[] ch, int start, int length){
      buffer.append(ch, start, length);
    }
  }
```

5.5.5 サンプルプログラム（SampleSAX.java）の実行結果

このサンプルプログラムの実行結果は標準出力に出力されます。標準出力の内容を次に示します。

```
$ 合計は315000円です。
```

5.6 XML Schema を使用するサンプルプログラム

この節では，XML Schema を使用するサンプルプログラムについて説明します。

サンプルプログラムのインストール先，ファイル名は次のとおりです。

インストール先

Windows の場合

＜ Cosminexus XML Processor をインストールしたディレクトリ＞¥samples¥XMLSchema

UNIX の場合

＜ Cosminexus XML Processor をインストールしたディレクトリ＞ /samples/XMLSchema

ファイル名

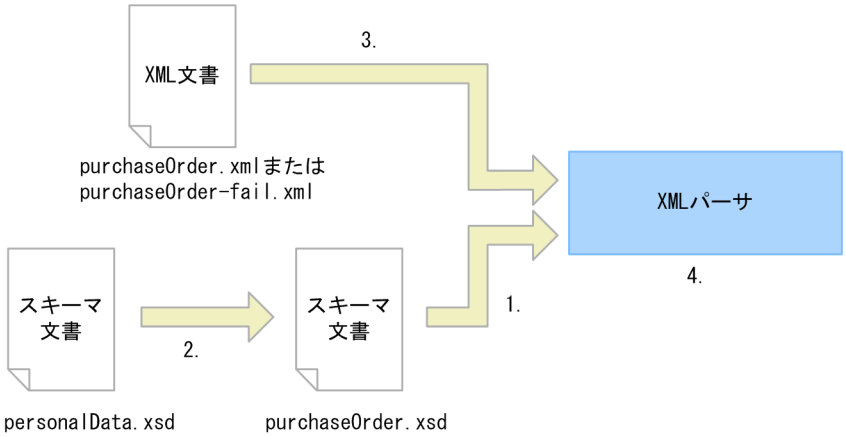
- purchaseOrder.xml
- purchaseOrder-fail.xml
- purchaseOrder.xsd
- personalData.xsd
- SampleValidateDOM.java
- SampleValidateSAX.java

5.6.1 サンプルプログラムの処理

サンプルプログラムでは，XML 文書として作成された注文明細書の妥当性を検証します。注文明細書は顧客の個人データを共有して使用しているため，注文明細書のスキーマ文書は，個人データのスキーマ文書を読み込みます。なお，スキーマ文書は Java プログラム内で特定します。

XML Schema を使用するサンプルプログラムの処理の流れを次の図に示します。

図 5-4　XML Schema を使用するサンプルプログラムの処理の流れ



サンプルプログラムは次のような処理を行います。

1. **スキーマ文書 purchaseOrder.xsd を読み込みます。**
   purchaseOrder.xsd は，商品の注文明細書を規定するスキーマ文書です。
2. **purchaseOrder.xsd は，スキーマ文書 personalData.xsd を読み込みます。**
   personalData.xsd は，注文した顧客の個人データを規定するスキーマ文書で，purchaseOrder.xsd とは異なる対象名前空間を持ちます。
3. **XML 文書 purchaseOrder.xml（または purchaseOrder-fail.xml）を読み込みます。**
   purchaseOrder.xml，および purchaseOrder-fail.xml は，検証される XML 文書となる注文明細書です。
4. **XML 文書の妥当性を検証します。**
   XML 文書がスキーマ文書に対して妥当な場合は「Validation OK」，妥当ではない場合はメッセージとともに「Validation NG」と標準出力に表示します。

5.6.2　使用する XML 文書（purchaseOrder.xml，purchaseOrder-fail.xml）

Cosminexus XML Processor は，XML Schema による検証の対象として，次の 2 種類の XML 文書を提供します。

1. 検証に成功する XML 文書（purchaseOrder.xml）
2. 検証に失敗する XML 文書（purchaseOrder-fail.xml）

（1）XML 文書（purchaseOrder.xml）

検証に成功する XML 文書（purchaseOrder.xml）を次に示します。

```
<?xml version="1.0"?>
<po:purchaseOrder
    xmlns:po="http://www.myshopping.com/schema/purchaseOrder"
    xmlns:psd="http://www.myshopping.com/schema/personalData">
  <po:shipTo age="20">
    <psd:firstName>John</psd:firstName>
    <psd:familyName>Doe</psd:familyName>
    <psd:occupation>accountant</psd:occupation>
    <psd:email>johnD@bpl.com</psd:email>
    <psd:tel>050-1234-1234</psd:tel>
    <psd:address country="US">
      <psd:street>Universal street 100</psd:street>
      <psd:city>Carson</psd:city>
      <psd:state>Nevada</psd:state>
      <psd:zip>10-456</psd:zip>
    </psd:address>
  </po:shipTo>
  <po:billTo age="20">
    <psd:firstName>Jane</psd:firstName>
    <psd:familyName>Doe</psd:familyName>
    <psd:occupation>lawyer</psd:occupation>
    <psd:email>janeD@bpl.com</psd:email>
    <psd:tel>033-1111-2345</psd:tel>
    <psd:address country="US">
      <psd:street>Times Square 555</psd:street>
      <psd:city>New York</psd:city>
      <psd:state>New York</psd:state>
      <psd:zip>155-5600</psd:zip>
    </psd:address>
    <po:credit year="2007" month="10">
      <po:creditHolder>Jane Doe</po:creditHolder>
      <po:creditNumber>1111444422229999</po:creditNumber>
      <po:creditCompany>CardCompanyA</po:creditCompany>
    </po:credit>
  </po:billTo>
  <po:items>
    <po:item>
      <po:productName
          productID="DTPC2000S">DeskTop PC</po:productName>
      <po:quantity>1</po:quantity>
      <po:price>1500</po:price>
      <po:shipDate>2004-05-20</po:shipDate>
    </po:item>
    <po:item>
      <po:productName
          productID="DC500MP">Digital Camera</po:productName>
      <po:quantity>1</po:quantity>
      <po:price>450</po:price>
      <po:shipDate>2004-05-20</po:shipDate>
    </po:item>
    <po:item>
      <po:productName
          productID="LP800S">Printer</po:productName>
      <po:quantity>1</po:quantity>
      <po:price>200</po:price>
      <po:shipDate>2004-05-20</po:shipDate>
    </po:item>
```

```
  </po:items>
</po:purchaseOrder>
```

(2) XML 文書（purchaseOrder-fail.xml）

検証に失敗する XML 文書（purchaseOrder-fail.xml）の一部を次に示します。

```
        :
        :
    <po:credit year="2007" month="10">
      <po:creditHolder>Jane Doe</po:creditHolder>
      <po:creditNumber>111144442222999955</po:creditNumber>
      <po:creditCompany>CardCompanyA</po:creditCompany>
    </po:credit>
        :
        :
```

この XML 文書は，<po:creditNumber> で始まるクレジット番号の定義以外は「5.6.2(1) XML 文書（purchaseOrder.xml）」と同じです。<po:creditNumber> で始まるクレジット番号の値として 18 桁の数字を使用していますが，サンプルのスキーマ文書ではクレジット番号を 16 桁で規定しているため，検証時にエラーとなります。

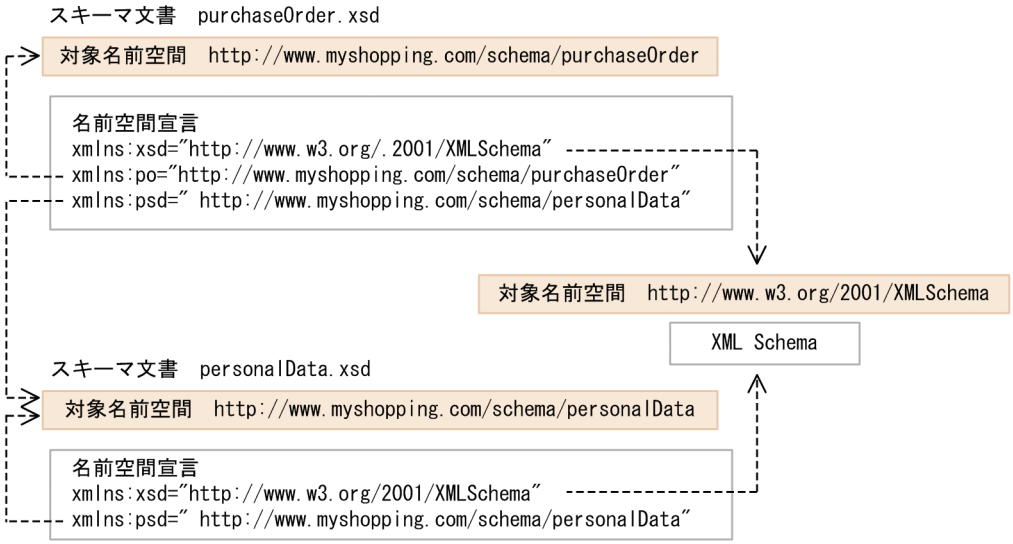
5.6.3 使用するスキーマ文書（purchaseOrder.xsd，personalData.xsd）

このサンプルでは，スキーマ文書として，purchaseOrder.xsd および personalData.xsd を使用します。

(1) スキーマ文書での各名前空間の関係

purchaseOrder.xsd，および personalData.xsd での各名前空間の関係を次の図に示します。それぞれの矢印は，参照する型定義や要素などが定義された対象名前空間を指しています。

図 5-5 スキーマ文書の名前空間の関係



(2) スキーマ文書（purchaseOrder.xsd）

使用するスキーマ文書（purchaseOrder.xsd）を次に示します。

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.myshopping.com/schema/purchaseOrder"
  xmlns:po="http://www.myshopping.com/schema/purchaseOrder"
  xmlns:psd="http://www.myshopping.com/schema/personalData">

<xsd:import
  namespace="http://www.myshopping.com/schema/personalData"
  schemaLocation="personalData.xsd"/>

<xsd:element name="purchaseOrder" type="po:purchaseOrderType"/>
<xsd:element name="shipTo" type="psd:personalDataType"/>
<xsd:element name="billTo" type="po:billToPersonalDataType"/>
<xsd:element name="items" type="po:itemsType">
  <xsd:key name="productID_unique">
    <xsd:selector xpath="po:item/po:productName"/>
    <xsd:field xpath="@productID"/>
  </xsd:key>
</xsd:element>

<xsd:element name="item" type="po:itemType"/>
<xsd:element name="productName" type="po:productNameType"/>
<xsd:element name="quantity">
  <xsd:simpleType>
    <xsd:restriction base="xsd:positiveInteger">
      <xsd:maxExclusive value="100"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```

```
<xsd:element name="price" type="xsd:positiveInteger"/>
<xsd:element name="shipDate" type="xsd:date"/>
<xsd:element name="credit" type="po:creditType"/>
<xsd:element name="creditHolder" type="xsd:string"/>
<xsd:element name="creditNumber">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string">
      <xsd:pattern value="¥d{16}"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>

<xsd:element name="creditCompany">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="CardCompanyA"/>
      <xsd:enumeration value="CardCompanyB"/>
      <xsd:enumeration value="CardCompanyC"/>
      <xsd:enumeration value="CardCompanyD"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>

<xsd:complexType name="purchaseOrderType">
  <xsd:sequence>
    <xsd:element ref="po:shipTo"/>
    <xsd:element ref="po:billTo"/>
    <xsd:element ref="po:items"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="billToPersonalDataType">
  <xsd:sequence>
    <xsd:group ref="psd:personalDataGroup"/>
    <xsd:element ref="po:credit"/>
  </xsd:sequence>
  <xsd:attributeGroup ref="psd:personalAttributeGroup"/>
</xsd:complexType>

<xsd:complexType name="itemsType">
  <xsd:sequence>
    <xsd:element ref="po:item"
        minOccurs="0" maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="itemType">
  <xsd:sequence>
    <xsd:element ref="po:productName"/>
    <xsd:element ref="po:quantity"/>
    <xsd:element ref="po:price"/>
    <xsd:element ref="po:shipDate"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="productNameType">
  <xsd:simpleContent>
```

```
    <xsd:extension base="xsd:string">
      <xsd:attribute name="productID" type="xsd:string"/>
    </xsd:extension>
  </xsd:simpleContent>
</xsd:complexType>

<xsd:complexType name="creditType">
  <xsd:sequence>
    <xsd:element ref="po:creditHolder"/>
    <xsd:element ref="po:creditNumber"/>
    <xsd:element ref="po:creditCompany"/>
  </xsd:sequence>
  <xsd:attribute name="year" type="po:yearType" use="required"/>
  <xsd:attribute name="month" type="po:monthType" use="required"/>
</xsd:complexType>

<xsd:simpleType name="yearType">
  <xsd:annotation>
    <xsd:documentation>
      Year set from 2004 to 2008
    </xsd:documentation>
  </xsd:annotation>
  <xsd:restriction base="xsd:string">
    <xsd:pattern value="200[4-8]"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="monthType">
  <xsd:restriction base="xsd:string">
    <xsd:pattern value="0[1-9]|1[0-2]"/>
  </xsd:restriction>
</xsd:simpleType>

</xsd:schema>
```

（3） スキーマ文書（personalData.xsd）

使用するスキーマ文書（personalData.xsd）を次に示します。

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.myshopping.com/schema/personalData"
  xmlns:psd="http://www.myshopping.com/schema/personalData">

<xsd:element name="personalData" type="psd:personalDataType"/>
<xsd:element name="firstName" type="xsd:string"/>
<xsd:element name="familyName" type="xsd:string"/>
<xsd:element name="occupation" type="xsd:string"/>
<xsd:element name="email" type="xsd:string"/>
<xsd:element name="tel" type="xsd:string"/>
<xsd:element name="address" type="psd:addressType"/>
<xsd:element name="building" type="xsd:string"/>
<xsd:element name="street" type="xsd:string"/>
<xsd:element name="city" type="xsd:string"/>
<xsd:element name="state" type="xsd:string"/>
<xsd:element name="zip" type="xsd:string"/>
```

```
<xsd:complexType name="personalDataType">
  <xsd:group ref="psd:personalDataGroup"/>
  <xsd:attributeGroup ref="psd:personalAttributeGroup"/>
</xsd:complexType>

<xsd:group name="personalDataGroup">
  <xsd:sequence>
    <xsd:element ref="psd:firstName"/>
    <xsd:element ref="psd:familyName"/>
    <xsd:element ref="psd:occupation"/>
    <xsd:element ref="psd:email"/>
    <xsd:element ref="psd:tel"/>
    <xsd:element ref="psd:address"/>
  </xsd:sequence>
</xsd:group>

<xsd:attributeGroup name="personalAttributeGroup">
  <xsd:attribute name="age"
      type="xsd:positiveInteger" use="optional"/>
  <xsd:attribute name="company"
      type="xsd:string" use="optional"/>
  <xsd:attribute name="department"
      type="xsd:string" use="optional"/>
  <xsd:attribute name="title"
      type="xsd:string" use="optional"/>
  <xsd:attribute name="fax"
      type="xsd:string" use="optional"/>
</xsd:attributeGroup>

<xsd:complexType name="addressType">
  <xsd:sequence>
    <xsd:element ref="psd:building" minOccurs="0"/>
    <xsd:element ref="psd:street"/>
    <xsd:element ref="psd:city"/>
    <xsd:element ref="psd:state"/>
    <xsd:element ref="psd:zip"/>
  </xsd:sequence>
  <xsd:attribute name="country" type="xsd:string"
      use="optional" default="US"/>
</xsd:complexType>
</xsd:schema>
```

5.6.4 DOM パーサを使用する場合のサンプルプログラム

DOM パーサを使用して XML 文書を検証する場合のサンプルプログラム，サンプルプログラムの実行方法，および実行結果について説明します。

（1） サンプルプログラム（SampleValidateDOM.java）

DOM パーサを使用する場合のサンプルプログラム（SampleValidateDOM.java）を次に示します。

```
import javax.xml.parsers.*;
import org.w3c.dom.*;
import org.xml.sax.*;
import java.io.*;

public class SampleValidateDOM implements ErrorHandler{
  public static final void main(String[] argv){
    if (argv.length != 2) {
      System.out.println(
        "Usage: java SampleValidateDOM <xml_file> <schema_file>");
      System.exit(1);
    }

    try{
      DocumentBuilderFactory dbf =
        DocumentBuilderFactory.newInstance();
      dbf.setNamespaceAware(true);
      dbf.setValidating(true);
      dbf.setAttribute(
        "http://java.sun.com/xml/jaxp/properties/schemaLanguage",
        "http://www.w3.org/2001/XMLSchema");
      dbf.setAttribute(
        "http://java.sun.com/xml/jaxp/properties/schemaSource",
        argv[1]);

      DocumentBuilder db = dbf.newDocumentBuilder();
      db.setErrorHandler(new SampleValidateDOM());
      Document doc = db.parse(argv[0]);
      System.out.println("Validation OK");

    }catch(ParserConfigurationException e){
      e.printStackTrace();
    }catch(SAXParseException e){
      e.printStackTrace();
    }catch(SAXException e){
      System.out.println("Validation NG: " + e.getMessage());
    }catch(IOException e){
      e.printStackTrace();
    }
  }

  public void warning(SAXParseException exception)
      throws SAXException {
    System.out.println("**Parsing Warning**¥n" +
        "  Line:    " +
        exception.getLineNumber() + "¥n" +
        "  URI:     " +
        exception.getSystemId() + "¥n" +
        "  Message: " +
        exception.getMessage());
  }

  public void error(SAXParseException exception)
      throws SAXException {
    System.out.println("**Parsing Error**¥n" +
        "  Line:    " +
        exception.getLineNumber() + "¥n" +
        "  URI:     " +
```

```
            exception.getSystemId() + "¥n" +
            "  Message: " +
            exception.getMessage());
        throw new SAXException("Error encountered");
    }

    public void fatalError(SAXParseException exception)
        throws SAXException {
        System.out.println("**Parsing Fatal Error**¥n" +
            "  Line:    " +
            exception.getLineNumber() + "¥n" +
            "  URI:     " +
            exception.getSystemId() + "¥n" +
            "  Message: " +
            exception.getMessage());
        throw new SAXException("Fatal Error encountered");
    }
}
```

(2) サンプルプログラムの実行結果

このサンプルプログラムの実行結果は標準出力に出力されます。標準出力の内容を次に示します。

検証対象に purchaseOrder.xml を指定した場合

```
Validation OK
```

検証対象に purchaseOrder-fail.xml を指定した場合

```
**Parsing Error**
  Line:    33
  URI:     file:///＜サンプルが格納されたディレクトリ＞/
purchaseOrder-fail.xml
  Message: KECX06063-E cvc-pattern-valid: Value
'111144442222999955' is not facet-valid with respect to pattern
'¥d{16}' for type '#AnonType_creditNumber'.
Validation NG: Error encountered
```

5.6.5 SAX パーサを使用する場合のサンプルプログラム

SAX パーサを使用して XML 文書を検証する場合のサンプルプログラム，サンプルプログラムの実行方法，および実行結果について説明します。

(1) サンプルプログラム（SampleValidateSAX.java）

SAX パーサを使用する場合のサンプルプログラム（SampleValidateSAX.java）を次に示します。

```
import javax.xml.parsers.*;
```

```
import org.xml.sax.*;
import java.io.*;

public class SampleValidateSAX implements ErrorHandler{
  public static final void main(String[] argv){
    if (argv.length != 2) {
      System.out.println(
        "Usage: java SampleValidateSAX <xml_file> <schema_file>");
      System.exit(1);
    }

    try{
      SAXParserFactory spf = SAXParserFactory.newInstance();
      spf.setNamespaceAware(true);
      spf.setValidating(true);
      SAXParser sp = spf.newSAXParser();
      sp.setProperty(
        "http://java.sun.com/xml/jaxp/properties/schemaLanguage",
        "http://www.w3.org/2001/XMLSchema");
      sp.setProperty(
        "http://java.sun.com/xml/jaxp/properties/schemaSource",
        argv[1]);

      XMLReader reader = sp.getXMLReader();
      reader.setErrorHandler(new SampleValidateSAX());
      reader.parse(argv[0]);
      System.out.println("Validation OK");

    }catch(ParserConfigurationException e){
      e.printStackTrace();
    }catch(SAXParseException e){
      e.printStackTrace();
    }catch(SAXException e){
      System.out.println("Validation NG: " + e.getMessage());
    }catch(IOException e){
      e.printStackTrace();
    }
  }

  public void warning(SAXParseException exception)
      throws SAXException {
    System.out.println("**Parsing Warning**¥n" +
        "  Line:    " +
        exception.getLineNumber() + "¥n" +
        "  URI:     " +
        exception.getSystemId() + "¥n" +
        "  Message: " +
        exception.getMessage());
  }

  public void error(SAXParseException exception)
      throws SAXException{
    System.out.println("**Parsing Error**¥n" +
        "  Line:    " +
        exception.getLineNumber() + "¥n" +
        "  URI:     " +
        exception.getSystemId() + "¥n" +
        "  Message: " +
```

```
            exception.getMessage());
        throw new SAXException("Error encountered");
    }

    public void fatalError(SAXParseException exception)
        throws SAXException {
        System.out.println("**Parsing Fatal Error**¥n" +
            "  Line:    " +
            exception.getLineNumber() + "¥n" +
            "  URI:     " +
            exception.getSystemId() + "¥n" +
            "  Message: " +
            exception.getMessage());
        throw new SAXException("Fatal Error encountered");
    }
}
```

(2) サンプルプログラムの実行結果

このサンプルプログラムの実行結果は標準出力に出力されます。標準出力の内容を次に示します。

検証対象に purchaseOrder.xml を指定した場合

```
Validation OK
```

検証対象に purchaseOrder-fail.xml を指定した場合

```
**Parsing Error**
  Line:    33
  URI:     file:///＜サンプルが格納されたディレクトリ＞/
purchaseOrder-fail.xml
  Message: KECX06063-E cvc-pattern-valid: Value
'111144442222999955' is not facet-valid with respect to pattern
'¥d{16}' for type '#AnonType_creditNumber'.
Validation NG: Error encountered
```

5.7 XSLT トランスフォーマを使用するサンプルプログラム

この節では，XSLT トランスフォーマを使用するサンプルプログラムについて説明します。

サンプルプログラムのインストール先，ファイル名は次のとおりです。

インストール先

Windows の場合

＜ Cosminexus XML Processor をインストールしたディレクトリ＞¥samples¥XSLT

UNIX の場合

＜ Cosminexus XML Processor をインストールしたディレクトリ＞ /samples/XSLT

ファイル名

- SampleXSLT.xml
- SampleXSLT.xsl
- SampleXSLT.java

5.7.1 サンプルプログラムの処理

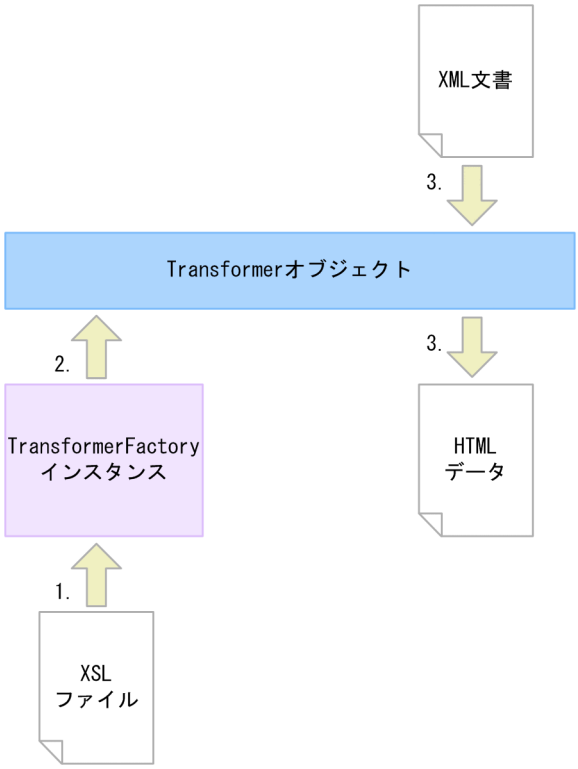
サンプルプログラムは次のような処理を行います。

1. **変換元の** XML **文書を読み込みます。**
2. 1. **で読み込んだ** XML **文書を** HTML **データへ変換します。**

5.7.2 サンプルプログラムの作成の流れ

XSLT トランスフォーマを使用するサンプルプログラムの作成の流れを次の図に示します。

図 5-6 XSLT トランスフォーマを使用するサンプルプログラムの作成の流れ



1. TransformerFactory **インスタンスを取得します。**
   メソッド newInstance で TransformerFactory インスタンスを取得します。
2. Transformer **オブジェクトを作成します。**
3. HTML **データへの変換を行います。**
   Transformer オブジェクトを使用して XML 文書を HTML データに変換します。

5.7.3 使用する XML 文書（SampleXSLT.xml）

サンプルプログラムが読み込む XML 文書を次に示します。

```
<?xml version="1.0" encoding="Shift_JIS"?>
<?xml-stylesheet type="text/xsl" href="SampleXSLT.xsl" ?>
<START>
  <伝票>
    <商品>
      <商品名>テレビ</商品名>
      <単価>15000</単価>
      <数量>14</数量>
    </商品>
    <商品>
```

```
        <商品名>ラジオ</商品名>
        <単価>3500</単価>
        <数量>6</数量>
      </商品>
      <商品>
        <商品名>ビデオ</商品名>
        <単価>21000</単価>
        <数量>4</数量>
      </商品>
    </伝票>
</START>
```

5.7.4 使用する XSL ファイル（SampleXSLT.xsl）

サンプルプログラムが読み込む XSL ファイルを次に示します。

```
<?xml version="1.0" encoding="Shift_JIS"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    version="1.0">
  <xsl:output method="html" indent="yes" encoding="Shift_JIS"/>

  <xsl:template match="/">
    <xsl:apply-templates />
  </xsl:template>

  <!-- STARTをルートとして変換を開始する -->
  <xsl:template match="START">
    <HTML>
      <HEAD>
        <TITLE>XMLをHTMLのテーブルで表示する</TITLE>
      </HEAD>
      <BODY BGCOLOR="#C0C0C0">
      <FONT SIZE="5">発注伝票</FONT>
      <TABLE BORDER="2">
      <TR><TH>商品名</TH><TH>単価</TH>
          <TH>数量</TH><TH>小計</TH></TR>
      <xsl:apply-templates />
      <TR><TH>合計</TH><TD> </TD><TD> </TD>
      <TD ALIGN="RIGHT"><xsl:value-of
          select="(//商品[1]/単価 * //商品[1]/数量 +
                   //商品[2]/単価 * //商品[2]/数量 +
                   //商品[3]/単価 * //商品[3]/数量)" /></TD></TR>
      </TABLE>
      </BODY>
    </HTML>
  </xsl:template>

  <xsl:template match="伝票">
    <xsl:apply-templates />
  </xsl:template>

  <!-- 1商品で1行を作成するし，最後に小計を出力する -->
  <xsl:template match="商品">
    <TR><xsl:apply-templates />
    <TD ALIGN="RIGHT">
```

```
        <xsl:value-of select="number(単価)*number(数量)" />
    </TD></TR>
  </xsl:template>

  <!-- 商品名を出力する -->
  <xsl:template match="商品名">
    <TD><xsl:value-of select="."/></TD>
  </xsl:template>

  <!-- 単価を出力する -->
  <xsl:template match="単価">
    <TD ALIGN="RIGHT"><xsl:value-of select="."/></TD>
  </xsl:template>

  <!-- 数量を出力する -->
  <xsl:template match="数量">
    <TD ALIGN="RIGHT"><xsl:value-of select="."/></TD>
  </xsl:template>
</xsl:stylesheet>
```

5.7.5 サンプルプログラム（SampleXSLT.java）

サンプルプログラムを次に示します。

```
import java.io.*;
import javax.xml.transform.*;
import javax.xml.transform.stream.*;
import javax.xml.parsers.*;

public class SampleXSLT{
  public static void main(String[] args){

    try{
      File file = new File(args[0]);

      //XMLファイルの読み込み
      Source source  = new StreamSource(file);
      Result result = new StreamResult(System.out);
      TransformerFactory factory =
          TransformerFactory.newInstance();

      //HTMLファイルへの変換
      Source style = factory.getAssociatedStylesheet(source,
          null, null, null);

      Transformer transformer = factory.newTransformer(style);
      transformer.transform(source, result);

    }catch(TransformerConfigurationException e){
      e.printStackTrace();
    }catch(TransformerException e){
      e.printStackTrace();
    }
  }
}
```

5.7.6 サンプルプログラム（SampleXSLT.java）の実行結果

(1) サンプルプログラムの実行結果

このサンプルプログラムの実行結果は標準出力に出力されます。標準出力の内容を次に示します。

```
<HTML>
<HEAD>
<META http-equiv="Content-Type" content="text/html;
charset=Shift_JIS">
<TITLE>XMLをHTMLのテーブルで表示する</TITLE>
</HEAD>
<BODY BGCOLOR="#C0C0C0">
<FONT SIZE="5">発注伝票</FONT>
<TABLE BORDER="2">
<TR>
<TH>商品名</TH><TH>単価</TH><TH>数量</TH><TH>小計</TH>
</TR>
<TR>
<TD>テレビ</TD>
      <TD ALIGN="RIGHT">15000</TD>
      <TD ALIGN="RIGHT">14</TD>
    <TD ALIGN="RIGHT">210000</TD>
</TR>
<TR>
<TD>ラジオ</TD>
      <TD ALIGN="RIGHT">3500</TD>
      <TD ALIGN="RIGHT">6</TD>
    <TD ALIGN="RIGHT">21000</TD>
</TR>
<TR>
<TD>ビデオ</TD>
      <TD ALIGN="RIGHT">21000</TD>
      <TD ALIGN="RIGHT">4</TD>
    <TD ALIGN="RIGHT">84000</TD>
</TR>
<TR>
<TH>合計</TH><TD> </TD><TD> </TD><TD
ALIGN="RIGHT">315000</TD>
</TR>
</TABLE>
</BODY>
</HTML>
```

5.8　XSLTC トランスフォーマを使用するサンプルプログラム

XSLTC トランスフォーマを使用するプログラムについて説明します。

ここでは，「5.7　XSLT トランスフォーマを使用するサンプルプログラム」を変更して，XSLTC トランスフォーマ機能を使用する方法を示します。

XSLTC トランスフォーマ機能を使用する場合のプログラム変更を次の図に示します。次の図に示す変更によって，XSLTC トランスフォーマ機能を使用できます。プログラムの実行方法および実行結果は，XSLT トランスフォーマを使用する場合のサンプルプログラムと同じです。詳細は「5.7.6　サンプルプログラム（SampleXSLT.java）の実行結果」を参照してください。

注意事項

プログラム変更後も，ファイル名は変更しないでください。ファイル名を変更すると，コンパイルに失敗します。

図 5-7　XSLTC トランスフォーマ機能を使用する場合のプログラム変更

（変更前）

```
import java.io.*;
import javax.xml.transform.*;
import javax.xml.transform.stream.*;
import javax.xml.parsers.*;


public class SampleXSLT{
  public static void main(String[] args){

    try{
      File file = new File(args[0]);

      //XMLファイルの読み込み
      Source source  = new StreamSource(file);
      Result result = new StreamResult(System.out);
      TransformerFactory factory =
          TransformerFactory.newInstance();

      //HTMLファイルへの変換
      Source style = factory.getAssociatedStylesheet(source,
          null, null, null);

      Transformer transformer = factory.newTransformer(style);
      transformer.transform(source, result);

    }catch(TransformerConfigurationException e){
      e.printStackTrace();
    }catch(TransformerException e){
      e.printStackTrace();
    }
  }
}
```

（変更後）

```
import java.io.*;
import javax.xml.transform.*;
import javax.xml.transform.stream.*;
import javax.xml.parsers.*;
import com.cosminexus.jaxp.xsltc.*;  //…1

public class SampleXSLT{
  public static void main(String[] args){

    try{
      File file = new File(args[0]);

      //XMLファイルの読み込み
      Source source  = new StreamSource(file);
      Result result = new StreamResult(System.out);
      TransformerFactory factory =
          TransformerFactoryXSLTC.newInstance();  //…2

      //HTMLファイルへの変換
      Source style = factory.getAssociatedStylesheet(source,
          null, null, null);

      Transformer transformer = factory.newTransformer(style);
      transformer.transform(source, result);

    }catch(TransformerConfigurationException e){
      e.printStackTrace();
    }catch(TransformerException e){
      e.printStackTrace();
    }
  }
}
```

プログラムの変更個所について次に説明します。各項番はプログラムのコメントに記述している番号に対応しています。

1. XSLTC トランスフォーマ機能を使用するためのパッケージをインポートします。
2. TransformerFactoryXSLTC クラスに newInstance メソッドを適用して，TransformerFactory オブジェクトを生成します。

6 Cosminexus XML Processor 使用時の注意事項

この章では，Cosminexus XML Processor をご使用になる際の注意事項について説明します。

6.1 JAXP1.3 に関する注意事項

JAXP1.3 の共通の注意事項を次の表に示します。

表 6-1 JAXP1.3 の共通の注意事項

項番	注意事項
1	JAXP1.3 仕様書で，引数に null を指定したときの動作が規定されていないメソッドがあります。このようなメソッドの引数に null を指定した場合，NullPointerException 例外が発生するなど，動作結果が未定義ですので注意してください。
2	JAXP1.3 仕様書が規定するメソッドで例外が発生したとき，例外オブジェクトに getMessage メソッドを適用することで，多くの場合は詳細メッセージが得られます。ただし，詳細メッセージが必ず得られることを保証するものではありません。したがって，アプリケーションでは，詳細メッセージの内容に依存した例外処理をしないで，例外の種別に応じて適切な処理をしてください。例えば，DOMException 例外の場合は，code フィールドでエラーの詳細要因を判定してください。
3	DocumentBuilder クラスの parse(String uri) メソッドや SAXParser クラスの parse(String uri, ...) メソッドなどの引数に指定する URI は，正しい形式で指定してください。URI として不正な文字列を指定すると，IllegalArgumentException などの RuntimeException 例外が発生する場合があります。
4	DocumentBuilder クラスの parse(File f) メソッドや SAXParser クラスの parse(File f, ...) メソッドなどの引数に指定する File オブジェクトは，引数に「%」「#」「?」を含まない File コンストラクタで生成してください。引数にこれらの文字が含まれる File コンストラクタで生成された File オブジェクトを指定すると，次に示す現象が発生する場合があります。 • IllegalArgumentException などの RuntimeException 例外が発生する • SAXException 例外が発生する • XML パーサの動作が不正になる
5	XML 文書のエンコーディング，XML 宣言の encoding 擬似属性，および InputSource や Reader のエンコーディングは統一してください。統一されていない場合の動作は保証しません。

6.2 JAXP1.4 に関する注意事項

JAXP1.4 に関する注意事項を説明します。

6.2.1 Java SE 6 を使用する場合のサポート範囲

Java SE 6 を使用する場合は，次の表に示すクラスとメソッドは使用できません。

表 6-2 Java SE 6 でサポートしていない JAXP1.4 のクラスとメソッド

項番	パッケージ名	クラス名	メソッド名
1	javax.xml.datatype	DatatypeFactory	newInstance(String, ClassLoader)
2	javax.xml.parsers	DocumentBuilderFactory	newInstance(String, ClassLoader)
3		SAXParserFactory	newInstance(String, ClassLoader)
4	javax.xml.stream	XMLEventFactory	newInstance(String, ClassLoader)
5		XMLInputFactory	newInstance(String, ClassLoader)
6		XMLOutputFactory	newInstance(String, ClassLoader)
7	javax.xml.transform	TransformerFactory	newInstance(String, ClassLoader)
8	javax.xml.validation	SchemaFactory	newInstance(String, ClassLoader)
9	javax.xml.xpath	XPathFactory	newInstance(String, ClassLoader)

6.2.2 javax.xml.transform.stax.StAXSource クラスおよび javax.xml.transform.stax.StAXResult クラスのサポート範囲

Cosminexus XML Processor の 08-50 以降では，javax.xml.transform.stax.StAXSource クラスおよび javax.xml.transform.stax.StAXResult クラスを使用できます。

JAXP1.4 および JAXB に定義されているメソッドの一部は，
javax.xml.transform.stax.StAXSource クラスおよび
javax.xml.transform.stax.StAXResult クラス，またはこれらのクラスが持つ FEATURE フィールドを引数として利用します。ただし，一部サポートしていないメソッドもあるため，サポートの有無を次の表に示します。

表 6-3 StAXSource クラスまたは StAXResult クラスを引数に使用するメソッドのサポート状況

項番	パッケージ名	クラス名	メソッド名	サポート有無
1	javax.xml.transform	TransformerFactory	getAssociatedStylesheet (Source, String, String, String)	×
2			getFeature(String)	×
3			newTemplates(Source)	○
4			newTransformer(Source)	○
5		Transformer	trnasform(Source, Result)	○
6	javax.xml.transform.sax	SAXTransformerFactory	newTransformerHandler(Source)	×
7			newXMLFilter(Source)	○
8		SAXSource	sourceToInputSource(Source)	×
9		TransformerHandler	setResult(Result)	○
10	javax.xml.validation	SchemaFactory	newSchema(Source)	×
11			newSchema(Source[])	×
12		Validator	validate(Source)	○
13			validate(Source, Result)	○
14	javax.xml.bind	JAXB	marshal(Object, Result)	×
15			unmarshal(Source, Class<T>)	×
16		Marshaller	marshal(Object, Result)	×
17		Unmarshaller	unmarshal(Source)	×
18			unmarshal(Source, Class<T>)	×
19	javax.xml.stream	XMLOutputFactory	createXMLEventWriter(Result)	○
20			createXMLStreamWriter(Result)	○
21		XMLInputFactory	createXMLEventReader(Source)	○
22			createXMLStreamReader(Source)	○

（凡例）
○：サポートしています。
×：サポートしていません。

6.2.3 J2SE 5.0 と Java SE 6 の動作の差異

J2SE 5.0 上で JAXP1.3 を利用する場合と，Java SE 6 上で JAXP1.4 を利用する場合とでは一部の動作が異なります。実行時の動作の差異を次の表に示します。

表 6-4 J2SE 5.0 と Java SE 6 の動作差異

項番	条件	実行結果	
		J2SE 5.0 の場合	Java SE 6 の場合
1	javax.xml.datatype.DatatypeFactory クラスの newDurationYearMonth(long) メソッドが返す Duration オブジェクトの内容	年および月以外のデータも保持されています。	年および月以外のデータは保持されません。
2	javax.xml.datatype.Duration クラスの equals メソッドのパラメタに null を指定した場合	NullPointerException が発生します。	false を返します。NullPointerException は発生しません。
3	U+D800 ～ U+DFFF を含み，属性値またはテキストに指定できない XML 文書をパースした場合（ただし，サロゲートペアは除く）	エラーになります。	正常終了します。ただし，処理結果は不正となるため，注意してください。
4	高速パース機能を使用して，RFC2396 が XML 文書名として許容していない文字（日本語や空白など）を含む File オブジェクトを次のメソッドに指定した場合 • DocumentBuilder クラスの parse(File) メソッド • SAXParser クラスの parse(File, DefaultHandler) メソッド	次のファイル名は，File オブジェクトで指定したとおりになります。 • チューニングサマリファイルの解析対象の XML 文書名 • チューニング詳細ファイルのファイル名 • チューニング詳細ファイルの解析対象の XML 文書名	次のファイル名は，%xx で符号化されます。 • チューニングサマリファイルの解析対象の XML 文書名 • チューニング詳細ファイルのファイル名 • チューニング詳細ファイルの解析対象の XML 文書名
5	javax.xml.stream.XMLStreamException(Throwable) コンストラクタで生成したオブジェクトで getCause メソッドを実行する場合	null を返します。	コンストラクタに指定した Throwable オブジェクトを返します。
6	次のすべての条件に該当する場合 • トランスフォーマのスタイルシートで xsl:sort を使った変換を実施する • トランスフォーマのスタイルシートのソートキーに日本語が指定されている	Java SE 6 の場合とは異なる出力結果になります。	J2SE 5.0 の場合とは異なる出力結果になります。

6.2.4 その他の注意事項

JAXP1.4 について，これまで説明したもの以外の注意事項を次に示します。

参考

関連する注意事項があるため，必要に応じて「6.5 StAX に関する注意事項」も参照してください。

表 6-5 JAXP1.4 使用時の注意事項

項番	注意事項
1	**[条件]** ValidatorHandler に設定した ContentHandler の endElement イベントで javax.xml.validation. TypeInfoProvider クラスの getElementTypeInfo() メソッドを実行する場合です。 **[Cosminexus XML Processor の動作]** IllegalStateException 例外がスローされます。TypeInfo オブジェクトは返されません。
2	**[条件]** 次のすべての条件に該当する場合です。 1. XMLOutputFactory の javax.xml.stream.isRepairingNamespaces プロパティに Boolean.FALSE を設定する。 2. 1. の XMLOutputFactory から，XMLStreamWriter または XMLEventWriter を作成する。 3. 2. の XMLStreamWriter または XMLEventWriter から StAXResult を作成する。 **[Cosminexus XML Processor の動作]** 上記の条件で作成した StAXResult を次に示すメソッドのどちらかの引数に指定して，名前空間宣言を含む XML 文書を処理すると，XMLStreamException 例外が発生します。 • javax.xml.validation.Validator クラスの validate(Source source, Result result) メソッド • javax.xml.transform.Transformer クラスの transform(Source source, Result result) メソッド※ 注※ ただし，このメソッドはスタイルシートを使わない恒等変換の場合だけ該当します。 **[回避策]** XMLOutputFactory の javax.xml.stream.isRepairingNamespaces プロパティには Boolean.TRUE を設定してください。
3	**[条件]** 次のすべての条件に該当する場合です。 1. トランスフォーマとして，XSLTC ではなく XSLT を使っている。 2. XMLInputFactory クラスの createXMLStreamReader メソッドまたは createXMLEventReader メソッドの引数にシステム識別子を指定しないで呼び出して，XMLStreamReader または XMLEventReader を作成する。 3. StAXSource のコンストラクタ引数に 2. で作成した XMLStreamReader または XMLEventReader を指定する。 4. TransformerFactory クラスの newTemplates メソッドまたは newTransformer メソッドの引数に，3. で作成した StAXSource を指定する。 **[Cosminexus XML Processor の動作]** TransformerFactory クラスの newTemplates メソッドまたは newTransformer メソッドを呼び出すときに fatalError イベントが発生します。メソッドの戻り値は null となります。 **[回避策]** 次の回避策の中から選択してください。 • XSLT の代わりに XSLTC を使用する。 • createXMLStreamReader メソッドまたは createXMLEventReader メソッドの引数に，システム識別子を指定する。 • StAXSource 以外の Source を使用する。

項番	注意事項
4	**[条件]** 次の条件に該当する場合です。 1. トランスフォーマの出力に StAXResult を指定する。 2. XSLT スタイルシートまたはトランスフォーマの出力プロパティの設定で，出力メソッドに「xml」を指定している。 3. 1. の StAXResult に XMLStreamWriter または UTF-8 以外の出力エンコーディングを指定した XMLEventWriter を指定する。 **[Cosminexus XML Processor の動作]** • 上記すべての条件に該当する場合は，トランスフォーマの出力結果の XML 宣言に encoding 擬似属性が出力されません。 • 上記のうち，1. および 2. の条件だけが該当する場合は，standalone 擬似属性が出力されません。
5	**[条件]** 次のすべての条件に該当する場合です。 • トランスフォーマの入力 XML 文書を StAXSource で与える。 • 入力 XML 文書のプロローグ部分またはエピローグ部分に処理命令が書かれている。 **[Cosminexus XML Processor の動作]** 入力 XML 文書のプロローグ部分およびエピローグ部分の処理命令は無視します。
6	**[条件]** 次のすべての条件に該当する場合です。 • トランスフォーマの出力に StAXResult を指定する。 • XSLT スタイルシートまたはトランスフォーマの出力プロパティの設定で，出力メソッドに「xml」を指定する。 • XSLT スタイルシートのテンプレートにマッチするノードが，入力 XML 文書内に存在しない。 **[Cosminexus XML Processor の動作]** トランスフォーマの出力結果に XML 宣言を出力しません。
7	**[条件]** 次のすべての条件に該当する場合です。 • トランスフォーマの出力に StAXResult を指定する。 • XSLT スタイルシートまたはトランスフォーマの出力プロパティの設定で，出力メソッドを指定していない。 **[Cosminexus XML Processor の動作]** トランスフォーマの出力結果の内容に関係なく，出力メソッドとして「xml」を仮定します。

項番	注意事項
8	**[条件]** 次のすべての条件に該当する場合です。 • トランスフォーマの出力に StAXResult を指定する。 • XSLT スタイルシートまたはトランスフォーマの出力プロパティに，次に示すプロパティのうち，どれかを指定する。 omit-xml-declaration doctype-system doctype-public cdata-section-elements **[Cosminexus XML Processor の動作]** プロパティは無効になります。
9	**[条件]** 次のすべての条件に該当する場合です。 • トランスフォーマの出力に StAXResult を指定する。 • XSLT スタイルシートまたはトランスフォーマの出力プロパティに，次に示すプロパティのうち，どちらかに不当な値を指定する。 version encoding **[Cosminexus XML Processor の動作]** 指定は無効になります。エラーは通知されません。
10	**[条件]** 次のすべての条件に該当する場合です。 • トランスフォーマの出力に StAXResult を指定する。 • XSLT スタイルシート内の xsl:text 要素の disable-output-escaping 属性に「yes」を指定する。 **[Cosminexus XML Processor の動作]** disable-output-escaping 属性に「yes」を指定した xsl:text 要素の内容はエスケープされません。そのままトランスフォーマの出力結果に出力されます。また，出力する文字列の前後には，次のような処理命令が追加されます。 (例) <?javax.xml.transform.disable-output-escaping ?><<?javax.xml.transform.enable-output-escaping ?>
11	**[条件]** 次のすべての条件に該当する場合です。 • トランスフォーマの出力に StAXResult を指定する。 • XSLT スタイルシートまたはトランスフォーマの出力プロパティの設定で，出力メソッドに「xml」を指定する。 • 出力ツリーのルート要素の前（プロローグ部分）に文字を出力する。 **[Cosminexus XML Processor の動作]** トランスフォーマの出力結果の XML 宣言より前に文字を出力します。
12	StAX は Shift_JIS 切り替え機能をサポートしていません。 Shift_JIS 切り替え機能を指定した DOM パーサ，または SAX パーサによって作成された Node，Source，Result などを StAX の API の引数として渡した場合の動作は保証しません。

項番	注意事項
13	StAX は Shift_JIS 切り替え機能をサポートしていません。 XSLT または XSLTC トランスフォーマの入出力に，StAXSource または StAXResult を使用した場合の動作は保証しません。
14	Validator クラスの validate(Source, Result) メソッドの引数に，StAXSource と StAXResult を指定する場合，StAXSource と StAXResult には同じエンコーディングを指定してください。
15	XSLT に次のフィーチャーが追加されましたが，Cosminexus XML Processor は常に false を返します。 • http://javax.xml.transform.stax.StAXSource/feature • http://javax.xml.transform.stax.StAXResult/feature

6.3 DOM パーサに関する注意事項

DOM パーサに関する注意事項を次の表に示します。

表 6-6 DOM パーサに関する注意事項

項番	注意事項
1	マルチスレッドプログラミングをする場合，DocumentBuilderFactory クラスはスレッドセーフではありません。したがって，複数のスレッドが同時に同一の DocumentBuilderFactory インスタンスにアクセスしてはいけません。スレッド間の競合を避けるため，次のどちらかの方法を使用してください。 • 各スレッドに一つの DocumentBuilderFactory インスタンスを持つ。 • 各スレッドが排他的に DocumentBuilderFactory インスタンスにアクセスする。
2	マルチスレッドプログラミングをする場合，DocumentBuilder クラスはスレッドセーフではありません。したがって，複数のスレッドが同時に同一の DocumentBuilder インスタンスを使用してはいけません。スレッド間の競合を避けるため，次の方法を使用してください。 • 各スレッドごとに一つの DocumentBuilder インスタンスを持つ。
3	マルチスレッドプログラミングをする場合，DOM ツリーはスレッドセーフではありません。したがって，parse メソッドによって生成された同一の DOM ツリーに対して，複数のスレッドが同時にアクセスしてはいけません。更新系メソッドだけではなく，参照系メソッドでも同時にアクセスしてはいけません。スレッド間の競合を避けるため，次の方法を使用してください。 • 各スレッドが排他的に DOM ツリーにアクセスする。
4	マルチスレッドプログラミングをする場合，org.w3c.dom，org.w3c.dom.bootstrap，org.w3c.dom.ls パッケージで規定されたオブジェクトはスレッドセーフではありません。したがって，複数のスレッドが同時にこれらのオブジェクトにアクセスしてはいけません。更新系メソッドだけではなく，参照系メソッドでも同時にアクセスしてはいけません。スレッド間の競合を避けるため，次の方法を使用してください。 • 各スレッドが排他的にこれらのオブジェクトにアクセスする。
5	Document インタフェースの createAttribute メソッドまたは createAttributeNS メソッドを使用して Attr ノードを生成する場合は，Attr インタフェースの setValue メソッドで値を設定してください。値が設定されていない状態で，Node インタフェースの getChildNodes メソッドで NodeList を取得した場合，その NodeList の動作は保証しません。
6	Node インタフェースの insertBefore メソッドや appendChild メソッドを使用して，すでに Element ノードを持つ Document ノードに対して要素を追加する場合，Node インタフェースの replaceChild メソッドを使用した場合と同じ結果になります。
7	DocumentBuilder クラスの「InputStream や InputSource を引数とする parse メソッド」でエラーが発生したとき，エラーハンドラへ渡される SAXParseException に getSystemId メソッドを適用すると null が返される場合があります。エラー発生元のシステム識別子を返すようにする場合は，parse メソッドを次のように使用してください。 • parse(InputStream is, String systemId) メソッドで，システム識別子を引数 systemId に指定する。 • parse(InputSource is) メソッドで，システム識別子を設定した InputSource を引数 is に指定する。
8	BOM(Byte Order Mark) 付きの UTF-16 で保存された XML 文書を parse(InputSource is) メソッドで解析する場合，InputSource に setEncoding メソッドを適用するときは引数に "UTF-16" を指定してください。

項番	注意事項
9	内部エンティティを定義する文字列中に，477 文字以上の要素名が含まれていると，java.lang.IndexOutOfBoundsException 例外が発生する場合があります。

6.4 SAX パーサに関する注意事項

SAX パーサに関する注意事項を次の表に示します。

表 6-7 SAX パーサに関する注意事項

項番	注意事項
1	マルチスレッドプログラミングをする場合，SAXParserFactory クラスはスレッドセーフではありません。したがって，複数のスレッドが同時に同一の SAXParserFactory インスタンスにアクセスしてはいけません。スレッド間の競合を避けるため，次のどちらかの方法を使用してください。 • 各スレッドに一つの SAXParserFactory インスタンスを持つ。 • 各スレッドが排他的に SAXParserFactory インスタンスにアクセスする。
2	マルチスレッドプログラミングをする場合，SAXParser クラスはスレッドセーフではありません。したがって，複数のスレッドが同時に同一の SAXParser インスタンスを使用してはいけません。スレッド間の競合を避けるため，次の方法を使用してください。 • 各スレッドに一つの SAXParser インスタンスを持つ。
3	マルチスレッドプログラミングをする場合，org.xml.sax，org.xml.sax.ext，org.xml.sax.helpers パッケージで規定されたオブジェクトはスレッドセーフではありません。したがって，複数のスレッドが同時にこれらのオブジェクトにアクセスしてはいけません。更新系メソッドだけではなく，参照系メソッドでも同時にアクセスしてはいけません。スレッド間の競合を避けるため，次の方法を使用してください。 • 各スレッドが排他的にこれらのオブジェクトにアクセスする。

4	SAX パーサは，一つの文字列データを複数の文字列（チャンク）に分割し，複数の characters イベントとしてアプリケーションに報告することが許されています。したがって，ContentHandler の実装では，文字列データが分割されることを意識する必要があります。ContentHandler の実装例を次に示します。この例では，characters イベントが発生するたびに文字列データをバッファリングしていき，endElement イベントが報告された時点で文字列データの終端と見なしています。
	
	`class MyHandler implements ContentHandler {`
	`String str = null;`
	`StringBuffer buffer = null;`
	`  :`
	`public void startElement(String uri, String localName, String qName,`
	`Attributes atts)`
	`  throws SAXException {`
	`buffer = new StringBuffer();`
	`}`
	`  :`
	`public void characters(char c[], int start, int length) throws`
	`SAXException {`
	`buffer.append(c, start, length);`
	`}`
	`  :`
	
	`public void endElement(String uri, String name, String qname) throws`
	`SAXException {`
	`str = buffer.toString();`
	`}`
	`  :`

項番	注意事項
5	XMLReaderFactory クラスの createXMLReader(String className) メソッドを使用し，XMLReader の生成を行う場合は，引数 className に "com.cosminexus.jaxp.impl.parsers.parsers.SAXParser" を指定してください。XMLReader の生成例を次に示します。 `XMLReader reader = XMLReaderFactory.createXMLReader("com.cosminexus.jaxp.impl.parsers.parsers.SAXParser");`
6	SAXParser クラスの「InputStream や InputSource を引数とする parse メソッド」でエラーが発生したとき，エラーハンドラへ渡される SAXParseException に getSystemId メソッドを適用すると null が返される場合があります。エラー発生元のシステム識別子を返すようにする場合は，parse メソッドを次のように使用してください。 • parse(InputStream is, …, String systemId) メソッドで，システム識別子を引数 systemId に指定する。 • parse(InputSource is, …) メソッドで，システム識別子を設定した InputSource を引数 is に指定する。
7	BOM(Byte Order Mark) 付きの UTF-16 で保存された XML 文書を parse(InputSource is, …) メソッドで解析する場合，InputSource に setEncoding メソッドを適用するときは引数に "UTF-16" を指定してください。
8	内部エンティティを定義する文字列中に，477 文字以上の要素名が含まれていると，java.lang.IndexOutOfBoundsException 例外が発生する場合があります。

6.5 StAX に関する注意事項

StAX に関する注意事項を次の表に示します。

表 6-8 StAX に関する注意事項

項番	注意事項
1	StAX1.0 仕様書には，動作が明確に定義されていないメソッドがあります。動作が明確に定義されていないメソッドの動作は，実装に依存します。引数を事前に確認するなどして，未規定の動作がないように対処してください。 動作が未規定の場合の動作例を次に示します。 例 1：Javadoc の例外欄に示されていない種別の例外が発生する。 引数に null を指定した場合の動作が規定されていないため，NullPointerException が発生する。 例 2：不当な順番でメソッドを呼び出した場合の動作が規定されていないため，例外が発生する。 XMLStreamWriter インタフェースの writeStartDocument メソッドを呼び出す前に writeEndDocument メソッドを呼び出すと，XMLStreamException が発生する。 例 3：矛盾が生じるような引数でメソッド呼び出しをした場合，その引数が無視される。 XMLEventFactory インタフェースの createStartElement, createEndElement メソッドに引数として指定する要素名が異なると，createEndElement メソッドに指定した引数が無視される。
2	XMLStreamReader インタフェースのメソッドの一部は，Javadoc での説明と各 API の説明との間に差異があります。Cosminexus XML Processor は，API の説明に従って動作します。 説明に差異のあるメソッドと差異の内容を次に示します。 IllegalStateException 例外が発生しない（Javadoc では説明なし） • getCharacterEncodingScheme() • getEncoding () • getVersion() • getNamespaceURI() • getNamespaceURI(String prefix) • getPrefix() • standaloneSet() • isStandalone() Javadoc とは異なる例外が発生する（Javadoc では IllegalStateException 例外が発生） • getElementText() • nextTag() • next()
3	**[条件]** XMLInputFactory の createXMLStreamReader(String systemId, InputStream stream) メソッドの引数 systemId と引数 stream が異なる XML 文書を指す場合です。 **[標準仕様]** 動作の規定なし。 **[Cosminexus XML Processor の動作]** 引数 stream が有効になります。 ただし，引数 stream が null の場合は，引数 systemId が有効になります。

項番	注意事項
4	**[条件]** XMLInputFactory の createXMLStreamReader(String systemId, Reader reader) メソッドの引数 systemId と引数 reader が異なる XML 文書を指す場合です。 **[標準仕様]** 動作の規定なし。 **[Cosminexus XML Processor の動作]** 引数 reader が有効になります。 ただし，引数 reader が null の場合は，引数 systemId が有効になります。
5	**[条件]** XML 文書の属性宣言と名前空間宣言に対して，javax.xml.stream.XMLStreamReader インタフェースの getEventType() メソッドを実行する場合です。 **[標準仕様]** XMLStreamReader の Javadoc には，属性と名前空間のパースに関する説明がありますが，パースによってイベントが発生するかどうかについての明確な規定はありません。 **[Cosminexus XML Processor の動作]** ATTRIBUTE イベント型や NAMESPACE イベント型が戻されることはありません。 START_ELEMENT イベント型が戻されたときに，属性および名前空間を処理してください。
6	**[条件]** XMLStreamReader インタフェースのメソッドうち，次に示すメソッドの引数 index に，属性数または名前空間宣言数を超えた値を指定する場合です。 • getAttributeLocalName(int index) • getAttributeName(int index) • getAttributeNamespace(int index) • getAttributePrefix(int index) • getAttributeType(int index) • getAttributeValue(int index) • getNamespaceURI(int index) • isAttributeSpecified(int index) **[標準仕様]** 動作の規定なし。 **[Cosminexus XML Processor の動作]** isAttributeSpecified(int index) メソッドは，戻り値が false になります。 それ以外のメソッドは，戻り値が null になります。

項番	注意事項
7	**[条件]** COMMENT イベント型に対し XMLStreamReader インタフェースの getTextCharacters(int sourceStart, char[] target, int targetStart, int length) メソッドを実行する場合です。 **[標準仕様]** Javadoc の説明と XMLStreamReader の説明との間で次の差異があります。 • Javadoc CHARACTERS，SPACE，または CDATA に関連づけられたテキストを取得。 • XMLStreamReader getTextCharacters メソッドは COMMENT イベント型に対して有効。 **[Cosminexus XML Processor の動作]** COMMENT イベントに関連づけられたテキストを取得します。
8	**[条件]** 次のすべての条件に該当する場合です。 1. XMLStreamWriter インタフェースの getPrefix(String uri) メソッドを実行する。 2. 1. の引数 uri の名前空間が複数の接頭辞にバインドされている。 **[標準仕様]** 動作の規定なし。 **[Cosminexus XML Processor の動作]** バインドされている接頭辞のうちの一つが戻り値となります。
9	**[条件]** 次のすべての条件に該当する場合です。 1. XMLStreamWriter インタフェースの setPrefix(String prefix, String uri) メソッドを実行する。 2. 1. の引数 prefix の接頭辞がすでに異なる名前空間にバインドされている。 **[標準仕様]** 動作の規定なし。 **[Cosminexus XML Processor の動作]** setPrefix で指定された URI でバインド先の接頭辞を再設定します。
10	**[条件]** 次のすべての条件に該当する場合です。 • XML 文書が UTF-16BE または UTF-16LE でエンコードされている。 • XML 宣言のエンコーディング指定が UTF-16 である。 • XMLStreamReader の getEncoding メソッド，または StartDocument の getCharacterEncodingScheme メソッドを使用して XML 文書のエンコーディングを取得する。 **[標準仕様]** XML 文書自身または XML 宣言のエンコーディングのどちらを返すのかが，規定されていません。 **[Cosminexus XML Processor の動作]** XML 宣言に指定したエンコーディングではなく，XML 文書自身のエンコーディング（UTF-16BE または UTF-16LE）を返します。

項番	注意事項
11	**[条件]** XMLEventFactory クラスの createStartElement および createEndElement, createAttribute メソッドの引数に，名前空間 URI を指定する場合です。 **[標準仕様]** 引数に指定した名前空間 URI が名前空間宣言として出力 XML 文書に書き出されるのかどうか，規定されていません。 **[Cosminexus XML Processor の動作]** 引数に指定した名前空間 URI は，名前空間宣言として出力 XML 文書に書き出しません。createNamespace メソッドを使用して，名前空間宣言を明示的に作成してください。
12	**[条件]** XMLStreamReader インタフェースを使用して，外部エンティティをパースする場合です。 **[標準仕様]** 外部エンティティを展開するかどうかが規定されていません。 **[Cosminexus XML Processor の動作]** EntityReference イベントは生成しません。 エンティティ参照を展開したものに対するイベント（CHARACTERS, START_ELEMENT など）を生成します。
13	XMLEventFactory クラスの createAttribute(String prefix, String namespaceURI, String localName, String value) メソッドの引数 prefix には，"" を指定できません。
14	XMLEventFactory クラスの createIgnorableSpace(String content) メソッドの引数 content には，空白以外の文字列を指定しないでください。
15	XMLInputFactory クラスの javax.xml.stream.isSupportingExternalEntities プロパティのデフォルト値は true です。
16	次に示す XMLInputFactory クラスのプロパティに null を指定するには，setEventAllocator メソッド，setXMLReporter メソッド，または setXMLResolver メソッドを使用します。 setProperty メソッドは使用しないでください。 • javax.xml.stream.allocator • javax.xml.stream.reporter • javax.xml.stream.resolver
17	次に示す XMLInputFactory クラスのプロパティの指定は，javax.xml.stream.supportDTD プロパティの値に false が設定されていても影響されません。 • javax.xml.stream.isReplacingEntityReferences • javax.xml.stream.isSupportingExternalEntities
18	**[条件]** XMLEventWriter クラスの add(XMLEventReader reader) メソッドで，入力 XML に次の要素や属性が含まれる場合，出力 XML と入力 XML とで形式が異なります。 • 空要素（例：<element1/>） • XML 宣言の encoding 属性 • XML 宣言の standalone 属性 **[Cosminexus XML Processor の動作]** • 空要素の場合は開始タグと終了タグだけが出力される（例：<element1></element1>）。 • XML 宣言は <?xml version="1.0" ?> が出力される。

項番	注意事項
19	XMLEventFactory クラスの createStartElement メソッドで生成した StartElement イベントに対する getNamespaceContext() メソッドの戻り値は，null になります。
20	ISO-10646-UCS-4 エンコーディングは使用できません。
21	バージョンが 1.1 の XML 文書は使用できません。
22	StAX では，次の部分に補助文字を指定できません。 • DTD 内の公開識別子 • DTD 内のシステム識別子 • 内部解析対象実体 • 属性値 • コメント • CData
23	Characters イベントは，分割して報告される場合があります。 StAX パーサは，一つの文字列データを複数の文字列（チャンク）に分割して，複数の Characters イベントとしてアプリケーションに報告することもできます。そのため，アプリケーション側では，Characters イベントが連続して発生することを意識する必要があります。
24	**[条件]** START_ELEMENT/END_ELEMENT 以外のイベント型に対して，XMLStreamReader インタフェースの getName() メソッドを実行する場合です。 **[Cosminexus XML Processor の動作]** IllegalArgumentException 例外が発生します。IllegalStateException 例外は発生しません。
25	**[条件]** 接頭辞を持たない要素に対して，XMLStreamReader インタフェースの getPrefix() メソッドまたは getAttributePrefix(int index) メソッドを実行する場合です。 **[Cosminexus XML Processor の動作]** 戻り値は，""(XMLConstants.DEFAULT_NS_PREFIX) となります。
26	**[条件]** 名前空間を持つ属性に対して，XMLStreamReader インタフェースの getAttributeValue(String namespaceURI,String localName) メソッドで引数 namespaceURI に null を，引数 localName に属性のローカル名を指定する場合です。 **[Cosminexus XML Processor の動作]** 戻り値は，null となります。属性値は取得されません。
27	**[条件]** END_DOCUMENT イベント型または ENTITY_REFERENCE イベント型に対して，XMLStreamReader インタフェースの getEncoding() メソッドまたは getVersion() メソッドを実行する場合です。 **[Cosminexus XML Processor の動作]** END_DOCUMENT イベント型の場合，NullPointerException 例外が発生します。 ENTITY_REFERENCE イベント型の場合，戻り値が null となります。

項番	注意事項
28	**[条件]** XMLStreamReader インタフェースの isAttributeSpecified(int index) メソッドを実行する場合です。 **[Cosminexus XML Processor の動作]** 次のように戻り値が異なります。 • XML 文書内に明示的に指定された属性の場合：true • DTD 内で暗黙的に指定されたデフォルト属性の場合：false
29	**[条件]** ENTITY_REFERENCE イベント型に対して，XMLStreamReader インタフェースの hasName() メソッドを実行する場合です。 **[Cosminexus XML Processor の動作]** 戻り値は true となります。
30	**[条件]** DTD イベント型に対して，XMLStreamReader インタフェースの getText() メソッドを実行する場合です。 **[Cosminexus XML Processor の動作]** 「DOCTYPE 宣言の文字列値」を返します。DTD 内部サブセットがある場合も同様です。「DTD 内部サブセットの文字列値」は返されません。
31	**[条件]** 内部エンティティ参照を含む要素のコンテンツに対して，XMLStreamReader インタフェースの getElementText() メソッドを実行する場合です。 **[Cosminexus XML Processor の動作]** 戻り値は，内部エンティティ参照の置換テキストが 2 回連結された文字列となります。
32	**[条件]** 次のどちらかに該当する場合です。 • ENTITY 宣言が存在しない XML 文書を XMLStreamReader インタフェースを使用して読み取り，DTD イベント型で XMLStreamReader インタフェースの getProperty("javax.xml.stream.notations") メソッドを実行する。 • NOTATION 宣言が存在しない XML 文書を XMLStreamReader インタフェースを使用して読み取り，DTD イベント型で XMLStreamReader インタフェースの getProperty("javax.xml.stream.entities") メソッドを実行する。 **[Cosminexus XML Processor の動作]** 戻り値は，要素のない java.util.List となります。
33	**[条件]** 次のすべての条件に該当する場合です。 1. パラメタエンティティ定義の中に DTD 宣言を記述する。 2. DOCTYPE 宣言で 1. のパラメタエンティティを使用する。 3. 1. および 2. の XML 文書を StAX でパースする。 4. XMLStreamReader の getText メソッド，または javax.xml.stream.events.DTD インタフェースを使用して DTD 宣言のテキストを取得する。 **[Cosminexus XML Processor の動作]** 取得した DTD 宣言のテキストが不当になります。

項番	注意事項
34	**[条件]** XML 宣言の擬似属性の属性値に不当な文字列を指定した場合です。 **[Cosminexus XML Processor の動作]** com.cosminexus.stax.xml.stream.internal.xni.XNIException 例外が発生します。 また，メッセージ中の {} 内に詳細文字列が出力されない場合があります。 (例) com.cosminexus.stax.xml.stream.internal.xni.XNIException: The standalone document declaration value must be "yes" or "no", not "{0}".
35	**[条件]** 次に示す，どれかのメソッドを実行した場合です。 • XMLStreamException.getLocation() メソッド • XMLEvent.getLocation() メソッド • XMLStreamReader.getLocation() メソッド **[Cosminexus XML Processor の動作]** 例外が発生した場所を特定するための近似値として，Location 情報を取得します。
36	**[条件]** 次のすべての条件に該当する場合です。 1. XML 文書に「:」で始まる要素名が含まれる。 2. StAX パーサの名前空間が有効になっている。 3. 2. の StAX パーサで 1. の XML 文書をパースする。 **[標準仕様]** パースエラーが発生します。 **[Cosminexus XML Processor の動作]** パースは正常終了します。
37	**[条件]** XMLEventFactory クラスの createStartDocument(String encoding, String version, boolean standalone) メソッドの standalone パラメタに true を設定する場合です。 **[標準仕様]** standalone の状態が true の XML 文書が生成されます。 **[Cosminexus XML Processor の動作]** standalone 宣言が生成されません。
38	**[条件]** 次のすべての条件に該当する場合です。 • version 宣言が存在しない XML 文書を XMLEventReader でパースする。 • StartDocument インタフェースの getVersion メソッドを実行する。 **[標準仕様]** デフォルト値の 1.0 を返します。 **[Cosminexus XML Processor の動作]** null を返します。

項番	注意事項
39	**[条件]** XML 文書内の次の部分に対して，XMLEventReader クラスの nextEvent() メソッド実行をする場合です。 • 空白文字列 • CDATA セクション • 記法宣言 **[Cosminexus XML Processor の動作]** • 空白文字列は，CHARACTERS イベント型となります。SPACE イベント型にはなりません。 • CDATA セクションは，CHARACTERS イベント型となります。CDATA イベント型にはなりません。 • 記法宣言に対する NOTATION_DECLARATION イベントは発生しません。
40	**[条件]** XML 文書内の次の部分に対して，XMLStreamReader インタフェースの getEventType() メソッドを実行する場合です。 • 空白文字列 • CDATA セクション **[Cosminexus XML Processor の動作]** 空白文字列も CDATA セクションも CHARACTERS イベント型となるため，CHARACTERS イベント型として処理してください。
41	**[条件]** 次のすべての条件に該当する場合です。 1. namespaceURI が接頭辞にバインドされていない。 2. XMLStreamWriter インタフェースの次に示すメソッドの引数 namespaceURI に，1. を指定する。 writeAttribute(String prefix, String namespaceURI, String localName, String value) **[Cosminexus XML Processor の動作]** 属性だけが出力されます。名前空間宣言 (xmlns:prefix="namespaceURI") は出力されません。
42	次に示す javax.xml.stream.events パッケージに含まれるインタフェースでは，writeAsEncodedUnicode(Writer writer) メソッドは使用できません。 • Attribute • Characters • Comment • DTD • EndDocument • EndElement • EntityDeclaration • EntityReference • Namespace • NotationDeclaration • ProcessingInstruction • StartDocument • StartElement • XMLEvent

項番	注意事項
43	javax.xml.stream.util パッケージに含まれる XMLEventAllocator インタフェースの allocate(XMLStreamReader reader, XMLEventConsumer consumer) メソッドは，使用できません。
44	マルチスレッドプログラミングの場合，XMLInputFactory クラスはスレッドセーフではありません。そのため，複数のスレッドが同時に同じ XMLInputFactory インスタンスにアクセスしないように注意してください。 スレッド間の競合を避けるため，次のどちらかの方法を採用してください。 • 各スレッドに一つの XMLOutputFactory インスタンスを持つ。 • 各スレッドが排他的に XMLOutputFactory インスタンスにアクセスする。
45	マルチスレッドプログラミングの場合，XMLOutputFactory クラスはスレッドセーフではありません。そのため，複数のスレッドが同時に同じ XMLOutputFactory インスタンスにアクセスしないように注意してください。 スレッド間の競合を避けるため，次のどちらかの方法を採用してください。 • 各スレッドに一つの XMLOutputFactory インスタンスを持つ。 • 各スレッドが排他的に XMLOutputFactory インスタンスにアクセスする。
46	マルチスレッドプログラミングの場合，XMLEventFactory クラスはスレッドセーフではありません。そのため，複数のスレッドが同時に同じ XMLEventFactory インスタンスにアクセスしないように注意してください。 スレッド間の競合を避けるため，次のどちらかの方法を採用してください。 • 各スレッドに一つの XMLEventFactory インスタンスを持つ。 • 各スレッドが排他的に XMLEventFactory インスタンスにアクセスする。
47	マルチスレッドプログラミングの場合，XMLStreamReader インタフェースはスレッドセーフではありません。そのため，複数のスレッドが同時に同じ XMLStreamReader インスタンスを使用しないように注意してください。 スレッド間の競合を避けるため，スレッドごとに一つの XMLStreamReader インスタンスを持つ必要があります。
48	マルチスレッドプログラミングの場合，XMLStreamWriter インタフェースはスレッドセーフではありません。そのため，複数のスレッドが同時に同じ XMLStreamWriter インスタンスを使用しないように注意してください。 スレッド間の競合を避けるため，スレッドごとに一つの XMLStreamWriter インスタンスを持つ必要があります。
49	マルチスレッドプログラミングの場合，XMLEventReader クラスはスレッドセーフではありません。そのため，複数のスレッドが同時に同じ XMLEventReader インスタンスを使用しないように注意してください。 スレッド間の競合を避けるため，スレッドごとに一つの XMLEventReader インスタンスを持つ必要があります。
50	マルチスレッドプログラミングの場合，XMLEventWriter クラスはスレッドセーフではありません。そのため，複数のスレッドが同時に同じ XMLEventWriter インスタンスを使用しないように注意してください。 スレッド間の競合を避けるため，スレッドごとに一つの XMLEventWriter インスタンスを持つ必要があります。
51	javax.xml.stream.XMLResolver インタフェースの resolveEntity メソッドの引数 namespace は，常に null になります。

項番	注意事項
52	javax.xml.stream.XMLResolver インタフェースの resolveEntity メソッドの戻り値のうち，XMLStreamReader 型および XMLEventReader 型のオブジェクトはサポートしていません。 **[標準仕様]** このリソースの戻り値の型は，次のどれかです。 • java.io.InputStream • javax.xml.stream.XMLStreamReader • java.xml.stream.XMLEventReader
53	次に示すイベントに対する getLocation() メソッドの戻り値は null です。 • javax.xml.stream.events.Attribute • javax.xml.stream.events.Namespace • javax.xml.stream.events.NotationDeclaration • javax.xml.stream.events.EntityDeclaration
54	XMLOutputFactory クラスのうち，次に示すメソッドの引数 result にシステム識別子を指定していない StAXResult は，指定できません。 • createXMLEventWriter(Result result) • createXMLStreamWriter(Result result)
55	**[条件]** 次のすべての条件に該当する場合です。 • javax.xml.stream.isRepairingNamespaces プロパティに true を指定する。 • XMLStreamWriter インタフェースのうち，次に示すどれかのメソッドを実行する。 writeAttribute(prefix, namespaceURI, localName, value) writeStartElement(prefix, localName, namespaceURI) writeEmptyElement(prefix, localName, namespaceURI) • 引数 namespaceURI がデフォルトの名前空間 URI と等しい。 • 引数 prefix が「""」でも null でもない。 **[標準仕様]** 接頭辞は書き込まれません。 **[Cosminexus XML Processor の動作]** 接頭辞が書き込まれます。

項番	注意事項
56	**[条件]** 次のすべての条件に該当する場合です。 • javax.xml.stream.isRepairingNamespaces プロパティに true を指定する。 • XMLStreamWriter インタフェースのうち，次に示すどれかのメソッドを実行する。 writeAttribute(prefix, namespaceURI, localName, value) writeStartElement(prefix, localName, namespaceURI) writeEmptyElement(prefix, localName, namespaceURI) • 引数 namespaceURI バインドがあり，異なる接頭辞にバインドしている。 **[標準仕様]** 接頭辞がランダムに生成されます。 (例) xmlns:{generated}="namespaceURI" {generated}:localName="value" <{generated}:localName xmlns:{generated}="namespaceURI"> **[Cosminexus XML Processor の動作]** ランダムな接頭辞は生成されません。 (例) xmlns:prefix="namespaceURI" prefix:localName="value" <prefix:localName xmlns:prefix="namespaceURI">
57	**[条件]** 次のすべての条件に該当する場合です。 • javax.xml.stream.isRepairingNamespaces プロパティを指定しない，または false を指定する。 • XMLStreamWriter インタフェースの writeAttribute(prefix, namespaceURI, localName, value) メソッドを実行する。 • 引数 namespaceURI はバインドがない。 • 引数 prefix は「""」でも null でもない。 **[標準仕様]** 名前空間宣言が生成されます。 **[Cosminexus XML Processor の動作]** 名前空間宣言が生成されません。
58	**[条件]** 次のすべての条件に該当する場合です。 • javax.xml.stream.isRepairingNamespaces プロパティを指定しない，または false を指定する。 • XMLStreamWriter インタフェースのうち，次に示すどれかのメソッドを実行する。 writeAttribute(prefix, namespaceURI, localName, value) writeStartElement(prefix, localName, namespaceURI) writeEmptyElement(prefix, localName, namespaceURI) • 引数 namespaceURI バインドがあり，異なる接頭辞にバインドしている。 **[標準仕様]** XMLStreamException 例外が発生します。 **[Cosminexus XML Processor の動作]** エラーにはなりません。次のように書き込まれます。 • prefix:localName="value" • <prefix:localName>

項番	注意事項
59	**[条件]** 次のすべての条件に該当する場合です。 • javax.xml.stream.isRepairingNamespaces プロパティを指定しない，または false を指定する。 • XMLStreamWriter インタフェースの次のどれかのメソッドを実行する。 writeAttribute(prefix, namespaceURI, localName, value) writeStartElement(prefix, localName, namespaceURI) writeEmptyElement(prefix, localName, namespaceURI) • 引数 namespaceURI がデフォルトの名前空間 URI と等しい。 • 引数 prefix が「""」でも null でもない。 **[標準仕様]** 接頭辞は書き込まれません。 **[Cosminexus XML Processor の動作]** 次に示す接頭辞が書き込まれます。 • prefix:localName="value" • <prefix:localName>
60	**[条件]** 次のすべての条件に該当する場合です。 • javax.xml.stream.isRepairingNamespaces プロパティを指定しない，または false を指定する。 • XMLStreamWriter インタフェースの writeAttribute(prefix, namespaceURI, localName, value) メソッドを実行する。 • 引数 namespaceURI がデフォルトの名前空間 URI と等しい。 • 引数 prefix が「""」または null である。 **[標準仕様]** 接頭辞は書き込まれません。 **[Cosminexus XML Processor の動作]** XMLStreamException 例外が発生します。
61	**[条件]** 次のすべての条件に該当する場合です。 • javax.xml.stream.isRepairingNamespaces プロパティに true を指定する。 • XMLStreamWriter インタフェースの writeEmptyElement(prefix, localName, namespaceURI) メソッドを実行する。 • 引数 prefix が「""」である。 **[標準仕様]** prefix == "" || null の場合，デフォルトの名前空間として処理されます。接頭辞は生成または書き込まれません。 namespaceURI がバインドされていない場合は，xmlns 宣言が生成または書き込まれます。 **[Cosminexus XML Processor の動作]** 接頭辞はランダムに生成されます。デフォルトの名前空間としては処理されません。

項番	注意事項
62	**[条件]** XMLInputFactory クラスのうち，次に示すメソッドの引数 encoding に "ISO-10646-UCS-2" が，引数 stream に encoding 擬似属性 "ISO-10646-UCS-2" が指定された XML 文書を指定する場合です。 • createXMLEventReader(stream, encoding) • createXMLStreamReader(stream, encoding) **[Cosminexus XML Processor の動作]** com.cosminexus.stax.xml.stream.internal.xni.XNIException 例外が発生します。 **[回避策]** encoding 指定のない，createXMLEventReader メソッドまたは createXMLStreamReader メソッドで処理してください。
63	**[条件]** 次のすべての条件に該当する場合です。 1. StartElement イベントの getNamespaceContext() メソッドで NamespaceContext を取得する。 2. 1. で取得した NamespaceContext に対して，接頭辞 xml や接頭辞 xmlns の名前空間 URI を取得する。 **[標準仕様]** • 接頭辞 xml は http://www.w3.org/XML/1998/namespace にバインドされています。 • 接頭辞 xmlns は http://www.w3.org/2000/xmlns/ にバインドされています。 **[Cosminexus XML Processor の動作]** 名前空間コンテキストに接頭辞 xml や接頭辞 xmlns の情報はありません。
64	XMLEventReader で XML 文書を読み込む場合，standalone 属性の有無とは関係なく，StartDocument イベントの standaloneSet() メソッドの戻り値は true になります。 **[標準仕様]** XML 文書のエンコーディング宣言で standalone 属性が設定されていた場合だけ，戻り値が true になります。
65	DTD イベントの getEntities() メソッドおよび getNotations() メソッドは，常に null を返します。 **[回避策]** javax.xml.stream.entities プロパティおよび javax.xml.stream.notations プロパティを使用することで，これらの情報にアクセスできます。詳細は，XMLStreamReader の Javadoc を参照してください。

項番	注意事項
66	**[条件]** XMLInputFactory クラスの createXMLEventReader(Source source) メソッドまたは createXMLStreamReader(Source source) メソッドに対して，システム識別子が指定されていない型を指定する場合です。該当する型を次に示します。 • DOMSource • SAXSource • StAXSource **[Cosminexus XML Processor の動作]** XMLStreamException 例外が発生します。
67	DTD イベントの getProcessedDTD() メソッドの戻り値は，常に null です。
68	Endelement イベントの getNamespaces() メソッドの戻り値は，常に空の Iterator です。 **[Cosminexus XML Processor の動作]** 範囲外になった名前空間の Iterator を返します。
69	**[条件]** 次のすべての条件に該当する場合です。 1. XML 文書中の文字データの中に，CDATA セクション終了区切り子 ("]]>") が単独で含まれる。 （例） <?xml version="1.0" encoding="UTF-8" ?> <root>te]]>xt</root> 2. 1. の文字データを XMLStreamReader または XMLEventReader インタフェースで読み取り，next() メソッドなどで次の構文解析イベントを取得する。 **[標準仕様]** XMLStreamReader インタフェースまたは XMLEventReader インタフェースから XMLStreamException 例外が発生します。 **[Cosminexus XML Processor の動作]** XMLStreamReader インタフェースの場合 XMLStreamReader インタフェースのイベント型が CHARACTERS のときに実行できる getTextXXXX() メソッドの実行結果に「]]>」が含まれます。 getText() メソッドを実行すると「]]>」を含む文字列が返されます。 getTextCharacters() メソッドを実行すると，'] ' '] ' ' > ' を含む配列が返されます。 etTextCharacters(int sourceStart, char[] target, int targetStart, int length) メソッドを実行すると，char 配列 target には '] ' '] ' ' > ' が含まれます。 getTextLength() メソッドを実行すると，「]]>」を含む文字列の長さが返されます。 XMLEventReader インタフェースの場合 nextEvent() メソッドで返される Characters オブジェクトに対して getData() メソッドを実行すると，「]]>」を含む文字列が返されます。
70	XMLOutputFactory クラスの createXMLEventWriter(OutputStream stream) メソッドや createXMLStreamWriter(OutputStream stream) メソッドで出力エンコーディングを指定しない場合，XMLEventWriter や XMLStreamWriter はシステムのデフォルト文字コードで出力されます。
71	XMLEventFactory クラスの createStartDocument() メソッドの実行結果は，createStartDocument(String encoding, ...) メソッドの引数 encoding に UTF-8 を指定した場合と同じです。

項番	注意事項
72	**[条件]** 次のすべての条件に該当する場合です。 • XMLOutputFactory クラスの createXMLEventWriter メソッドや createXMLStreamWriter メソッドで指定する出力エンコーディングが，UTF-8 または UTF-16 以外である。 • 作成された XMLEventWriter や XMLStreamWriter で，2 バイトコードを含む XML 文書を出力する。 • 出力する XML 文書のエンコーディングの指定が次に示すどちらかの条件に該当する。 XMLOutputFactory クラスの createXMLEventWriter メソッドで指定する出力エンコーディングと XMLEventFactory クラスの createStartDocument メソッドで指定する出力 XML 宣言の encoding 擬似属性が一致していない。 XMLStreamWriter インタフェースの writeStartDocument メソッドの引数 encoding を指定していない。 **[Cosminexus XML Processor の動作]** 出力エンコーディングが UTF-8 および UTF-16 以外の，encoding 擬似属性のない不正な XML 文書を出力します。 **[回避策]** XMLOutputFactory クラスの createXMLEventWriter メソッドや createXMLStreamWriter メソッドの encoding 指定には，UTF-8 を指定してください。

6.6 スキーマ検証に関する注意事項

スキーマ検証に関する注意事項を次の表に示します。

表 6-9 スキーマ検証に関する注意事項

項番	注意事項
1	XML パーサにエンティティリゾルバを登録する場合は，エンティティリゾルバの resolveEntity メソッドの戻り値である InputSource に必ずシステム識別子を設定してください。設定しないと，xsd:import や xsd:include で指定したスキーマ文書の場所の解決に失敗してエラーとなる場合があります。

6.7 XSLT/XSLTC 共通の注意事項

XSLT/XSLTC 共通の注意事項を次の表に示します。

表 6-10 XSLT/XSLTC 共通の注意事項

項番	注意事項
1	マルチスレッドプログラミングをする場合，TransformerFactory クラス，および TransformerFactoryXSLTC クラスはスレッドセーフではありません。したがって，複数のスレッドが同時に同一の TransformerFactory インスタンス，および TransformerFactoryXSLTC インスタンスにアクセスしてはいけません。スレッド間の競合を避けるため，次のどちらかの方法を使用してください。 • 各スレッドに TransformerFactory インスタンス，および TransformerFactoryXSLTC インスタンスを持つ。 • 各スレッドが排他的に TransformerFactory インスタンス，および TransformerFactoryXSLTC インスタンスにアクセスする。
2	マルチスレッドプログラミングをする場合，Transformer クラスはスレッドセーフではありません。したがって，複数のスレッドが同時に同一の Transformer インスタンスにアクセスしてはいけません。スレッド間の競合を避けるため，次の方法を使用してください。 • 各スレッドに Transformer インスタンスを持つ。
3	マルチスレッドプログラミングをする場合，SAXTransformerFactory クラス，TemplatesHandler オブジェクト，TransformerHandler オブジェクトはスレッドセーフではありません。したがって，各スレッドは排他的にこれらのオブジェクトにアクセスするようにしてください。
4	XSLT トランスフォーマと XSLTC トランスフォーマは同一の機能を提供しますが，エラー発生時に出力するメッセージは異なる場合があります。
5	xsl:attribute 要素の namespace 属性に "http://www.w3.org/1999/XSL/Transform" を指定した場合，出力される属性の名前空間接頭辞が XSLT と XSLTC とで異なります。ただし，名前空間を有効に設定したパーサにとっては，名前空間接頭辞そのものは意味を持ちません。
6	xsl:attribute 要素および xsl:processing-instruction 要素の name 属性に QName でない名前を指定した場合，XSLT では warning イベントが発生しますが，XSLTC では error イベントが発生します。
7	name 属性以外の属性値が異なる xsl:decimal-format 要素が複数存在する場合に発生するエラーイベントは，XSLT では fatalError イベントが発生し，XSLTC では warning イベントが発生します。
8	xsl:element 要素または xsl:attribute 要素の name 属性には，"xml" で始まる名前空間接頭辞を持つ名前を指定しないでください。
9	xsl:element 要素，xsl:attribute 要素，または xsl:processing-instruction 要素の name 属性が，空文字列になる変数参照である場合，XSLT では warning イベントが発生しますが，XSLTC では error イベントが発生します。
10	xsl:element 要素の name 属性がない場合，XSLT では error イベントが発生しますが，XSLTC では warning イベントが発生します。
11	xsl:namespace-alias 要素を使った場合，出力される属性の名前空間接頭辞が XSLT と XSLTC とで異なります。ただし，名前空間を有効に設定したパーサにとっては，名前空間接頭辞そのものは意味を持ちません。

項番	注意事項
12	xsl:number 要素の format 属性に指定するフォーマットトークンには，"1"，" 1 の前に任意の個数の 0 が続いたもの "，"A"，"a"，"i"，"I" のどれかを指定してください。
13	xsl:number 要素の level 属性が "any" の場合，count 属性に属性ノードを表すパターンを指定すると，XSLT と XSLTC とで動作が異なります。XSLT では属性ノードがカウントされませんが，XSLTC では属性ノードがカウントされます。
14	xsl:output 要素の doctype-system 属性に空文字列を指定しないでください。同様に，Transformer クラスの setOutputProperty メソッドを用いて doctype-system プロパティに空文字列を指定しないでください。
15	xsl:output 要素の method 属性には "xml"，"html"，"text" のうちのどれか一つを指定してください。
16	xsl:processing-instruction 要素の name 属性に，評価結果が空文字列になるような XPath 式を記述すると，XSLT では warning イベントが発生しますが，XSLTC では error イベントが発生します。
17	xsl:value-of 要素の select 属性に変数参照を記述すると，TransformerException が発生します。
18	XSLT と XSLTC では "document(/)" の評価結果が異なります。XSLT ではスタイルシートのルート要素として評価され，XSLTC では空のノード集合として評価されます。
19	document 関数の第一引数に文字列を指定するとき，その文字列に次に該当する XML 文書を指定した場合，XSLTC は XSLT とエラーレベルが異なります。 • 存在しない XML 文書 • 整形式でない XML 文書
20	format-number 関数のフォーマットパターン文字列に通貨記号（#x00A4）を含めてもエラーにならないことがあります。
21	system-property 関数の引数には，"xsl:version"，"xsl:vendor"，"xsl:vendor-url" のうちのどれか一つを指定してください。
22	埋め込みスタイルシートを使った変換で，処理命令 "xml-stylesheet" の擬似属性 "href" で指定した ID に対応するスタイルシート要素が複数存在する場合，XSLT と XSLTC では適用されるテンプレートが異なります。XSLT では最後に現れたテンプレートが適用され，XSLTC では最初に現れたテンプレートが適用されます。
23	トランスフォーマ機能で，出力プロパティに，未サポートのエンコーディングが指定されている場合，アプリケーションでトランスフォーマにエラーリスナーを登録しても，エラーメッセージ（KECX02322-W）は，エラーリスナーに報告されず，標準出力ストリームへ出力されます。なお，XSLTC の場合，上記エラーメッセージが標準出力ストリームへ出力されるのに加えて，エラーメッセージ（KECX07076-W）がエラーリスナーに報告されます。メッセージの詳細については，マニュアル「Cosminexus アプリケーションサーバ メッセージ 3」の「2. KECX（Cosminexus XML Processor が出力するメッセージ）」を参照してください。
24	変換結果を HTML 形式で出力した場合，XSLT と XSLTC とでは要素間の改行出力に差異があります。ただし，HTML 文書では，要素間の改行は意味を持たないため問題ありません。
25	次の条件がすべて重なるとき，DOMResult オブジェクトの setNode（Node node）メソッドを実行すると，IllegalStateException 例外ではなく，IllegalArgumentException 例外が発生します。 • DOMResult オブジェクトの nextSibling が null ではない。 • nextSibling が引数 node の子でない。 この場合は，setNode メソッドの例外を処理する catch 節で，IllegalStateException 例外だけでなく IllegalArgumentException 例外もキャッチしてください。

項番	注意事項
26	スタイルシート内で QName として評価される場所（xsl:element 要素の name 属性，key 関数の第一引数など）に，XML1.0 規格で名前として使用することが許されていない文字（Unicode 補助文字，全角数字など）が使用されても，エラーにならないで処理が続行される場合があります。このような場合，トランスフォーマに出力メソッドとして "xml" を指定し，出力バージョンとして "1.0" を指定しても，出力される文書は XML1.0 規格に準拠しません。
27	内部エンティティを定義する文字列中に，477 文字以上の要素名が含まれていると，java.lang.IndexOutOfBoundsException 例外が発生する場合があります。
28	次に示す DOMResult のコンストラクタ引数に Document オブジェクトを指定する場合は，DocumentBuilder クラスの newDocument メソッドで作成した，新しい Document オブジェクトを指定してください。 • DOMResult(Node node) • DOMResult(Node node, String systemId)
29	値が重複した ID 型属性を含む XML 文書に対して，id 関数を使った XSLT スタイルシートを適用した場合，id 関数の結果が不正になることがあります。
30	次の条件の両方に該当する場合，トランスフォーマの変換処理の実行速度が大幅に低下します。 • 入力 XML 文書が大量の実体参照を含む。 • トランスフォーマの出力先が DOMResult である。

6.8 XSLT に関する注意事項

XSLT に関する注意事項を次に示します。

6.8.1 エラーを通知しないケース

表 6-11 XSLT に関する注意事項（エラーを通知しないケース）

項番	注意事項
1	xsl:copy-of 要素の select 属性に，ノード集合以外のオペランドを "|" 演算子で結合した式が書かれた場合，XSLT トランスフォーマはエラーを通知しません。
2	xsl:for-each 要素の select 属性が XPath 式として許されない文字（"~"，"#" など）を含む場合，XSLT トランスフォーマはエラーを通知しません。
3	xsl:output 要素の cdata-section-elements 属性に QName 規則に合致しない文字列を指定した場合，XSLT トランスフォーマはエラーを通知しません。
4	element-available 関数および function-available 関数に文字列型以外の引数を指定した場合，XSLT トランスフォーマはエラーを通知しません。

6.8.2 その他の注意事項

表 6-12 XSLT に関する注意事項（その他）

項番	注意事項
1	xsl:number 要素の grouping-size 属性に指定できない負の値を指定した場合，正の値に変換してグルーピングを行います。
2	xsl:preserve-space 要素および xsl:strip-space 要素の elements 属性には空文字列以外の文字列を指定してください。
3	XSLT を使って変換を行う場合，xsl:strip-space 要素または xsl:preserve-space 要素の elements 属性に "*" を指定すると，名前空間接頭辞付きの要素が空白の削除対象，または保存対象として選択されない場合があります。名前空間接頭辞付きの要素を空白の削除対象または保存対象とする場合は，elements 属性に名前空間接頭辞付きの要素名を指定してください。
4	format-number 関数の第二引数には，長さが 1 文字以上のフォーマットパターンを指定してください。
5	出力プロパティの method 属性に "html" を指定し，cdata-section-elements 属性に要素名を指定した場合，cdata-section-elements 属性が無視されず，指定した要素の内容が CDATA セクションとして出力されます。

6.9　XSLTC に関する注意事項

XSLTC に関する注意事項を次に示します。

6.9.1　スタイルシートサイズの注意事項

表 6-13　XSLTC に関する注意事項（スタイルシートサイズ）

項番	注意事項
1	XSLTC には，処理可能なスタイルシートサイズの上限があります。18,000 バイトを超えるスタイルシートを処理すると，KECX07058-E のエラーが発生して処理を中止する場合があります。ただし，スタイルシートサイズの上限は，スタイルシートの内容によって変動します。 メッセージの詳細については，マニュアル「Cosminexus アプリケーションサーバ メッセージ 3」の「2.　KECX（Cosminexus XML Processor が出力するメッセージ）」を参照してください。

6.9.2　変換処理性能に関する注意事項

表 6-14　XSLTC に関する注意事項（変換処理性能）

項番	注意事項
1	「3.2.1(1)　同一の XSLT スタイルシートを使って複数の XML 文書を変換する場合」で示したケースに該当するすべての場合について，XSLTC トランスフォーマの変換処理性能が XSLT トランスフォーマより高いことを保証するものではありません。XSLTC トランスフォーマを使用する場合は，実際に使用する XSLT スタイルシートおよび変換対象の XML 文書を使って性能を評価することを推奨します。

6.9.3　XSLT 要素の注意事項

表 6-15　XSLTC に関する注意事項（XSLT 要素）

項番	注意事項
1	xsl:apply-imports 要素が処理されません。
2	テンプレートが xsl:with-param によってパラメタとしてノード集合を受け取った場合，そのパラメタを xsl:apply-templates 要素の select 属性で参照すると，それ以降，同じパラメタを参照してもノード集合を取得できなくなります。
3	xsl:attribute 要素の子要素が xsl:copy の場合，出力される属性値が空になります。
4	xsl:attribute 要素の内容に xsl:value-of 要素を指定する場合は，select 属性に指定する XPath 式を絶対ロケーションパスで記述してください。
5	xsl:attribute-set 要素が use-attribute-sets 属性で自分自身を参照（循環参照）する場合，XSLTC ではスタックオーバーフローが発生します。
6	すべての属性の属性値が等しい xsl:decimal-format 要素を複数個定義した場合，XSLTC はエラーを通知します。

項番	注意事項
7	xsl:element 要素または xsl:attribute 要素の name 属性には属性値テンプレートを指定しないでください。

8	次の条件がすべて重なるとき，要素が属する名前空間が不正になります。
	• xsl:element 要素の子要素に xsl:attribute 要素がある。
	• xsl:element 要素の name 属性が接頭辞付きの名前で，namespace 属性の指定がある。
	• xsl:attribute 要素の name 属性が接頭辞付きの名前で，namespace 属性の指定がある。
	• xsl:element と xsl:attribute の name 属性に指定された接頭辞が同じで，namespace 属性の値が異なる。
	上記現象が発生するスタイルシートの例を次に示します。
	<pre><xsl:template match="/">
	  <xsl:element name="child1" namespace="AAA">
	    <xsl:element name="A:child2" namespace="XXX">
	      <xsl:attribute name="A:attr" namespace="AAA">1</xsl:attribute>
	    </xsl:element>
	  </xsl:element>
	</xsl:template></pre>

9	xsl:for-each 要素の select 属性に述部を持つ XPath 式を指定する場合，述部に「ノードを比較対象とする式」を指定しないでください。この場合，スタイルシートを次のように変更してください。
	1. xsl:for-each 要素の select 属性の XPath 式から述部を削除してください。
	2. xsl:for-each 要素の子に xsl:if 要素を挿入して，test 属性に 1. で削除した述部の内容を指定してください。
	スタイルシートの変更例を次に示します。
	（変更前）
	<pre><xsl:template match="/">
	  <xsl:for-each select="element[sub='x']">
	    <xsl:value-of select="."/>
	  </xsl:for-each>
	</xsl:template></pre>
	（変更後）
	<pre><xsl:template match="/">
	  <xsl:for-each select="element">
	    <xsl:if test="sub='x'">
	      <xsl:value-of select="."/>
	    </xsl:if>
	  </xsl:for-each>
	</xsl:template></pre>

10	xsl:for-each 要素の select 属性に「"descendant::node()" または "descendant-or-self::node()" を含む XPath 式」を指定しないでください。要素ノードおよびテキストノードを選択する目的でこのような XPath 式を使用する場合は，スタイルシートを次のように変更してください。 • "descendant::node()" を "descendant::node()|descendant::text()" に置き換えてください。 • "descendant-or-self::node()" を "descendant-or-self::node()|descendant-or-self::text()" に置き換えてください。

項番	注意事項

11	xsl:if 要素または xsl:when 要素の test 属性に「self 軸と述部を両方持つ XPath 式」を指定しないでください。この場合，スタイルシートを次のように変更してください。
	1. xsl:if 要素または xsl:when 要素の test 属性の XPath 式から，述部を削除してください。
	2. xsl:if 要素または xsl:when 要素の子に xsl:if 要素を挿入して，その test 属性に 1. で削除した述部の内容を指定してください。
	
	スタイルシートの変更例を次に示します。
	（変更前）
	
	`<xsl:template match="/element/sub">`
	`  <xsl:if test="self::sub[attribute::attr='x']">`
	`  <xsl:value-of select="."/>`
	`  </xsl:if >`
	`</xsl:template>`
	
	（変更後）
	
	`<xsl:template match="/element/sub">`
	`  <xsl:if test="self::sub">`
	`  <xsl:if test="attribute::attr='x'">`
	`  <xsl:value-of select="."/>`
	`  </xsl:if>`
	`  </xsl:if >`
	`</xsl:template>`

12	xsl:import 要素の href 属性の値が空文字列または空白だけの場合，error イベントが発生し，エラーメッセージとして "null" が出力されます。
13	xsl:number 要素の count 属性に "node()" を含むパターンを指定しないでください。要素ノードを番号付けする目的で "node()" を使用する場合は，代わりに "*" または " 要素名 " を使用してください。
14	xsl:number 要素または xsl:decimal-format 要素の grouping-separator 属性に複数の空白文字列や 1 文字以上の文字列，または全角文字を指定した場合，エラーにはならず，指定した文字列で区切られた値が出力されます。
15	xsl:number 要素の count 属性に "/" を指定した場合，XSLT では "1" が出力されますが，XSLTC では java.lang.VerifyError が発生します。
16	xsl:number 要素の from 属性に "/" を指定した場合，XSLT では "1" が出力されますが，XSLTC では java.lang.VerifyError が発生します。
17	xsl:output 要素の doctype-public 属性には空文字列以外の文字列を指定してください。同様に，Transformer クラスの setOutputProperty メソッドを用いて doctype-public プロパティを設定する場合も，空文字列以外の文字列を指定してください。
18	xsl:processing-instruction 要素の name 属性に，処理命令のターゲット名として許されない "XML" を指定した場合，"<?XML?>" という不正な処理命令が出力されます。
19	xsl:sort の data-type 属性に不正な値 "NUMBER"（大文字）を指定した場合，デフォルト動作（"text" が指定された場合と同じ順序でソートする）が行われません。
20	xsl:sort の order 属性に不正な値 "DESCENDING"（大文字）を指定した場合，デフォルト動作（"ascending" が指定された場合と同じ順序でソートする）が行われません。

項番	注意事項

21	xsl:template 要素の match 属性に「"node()" を含み，かつ，その後ろにロケーションパスが続く XPath 式」を指定しないでください。要素ノードを選択する目的で "node()" を使用する場合は，代わりに "*" を使用してください。スタイルシートの変更例を次に示します。
	（変更前）
	
	`<xsl:template match="/node()/sub">`
	`<result><xsl:value-of select="."/></result>`
	`</xsl:template>`
	
	（変更後）
	
	`<xsl:template match="/*/sub">`
	`<result><xsl:value-of select="."/></result>`
	`</xsl:template>`

22	xsl:template 要素の子要素としてトップレベル要素を書いた場合，無視されずに有効な子要素として処理されます。
23	xsl:value-of 要素の select 属性に，負のゼロを引数とする string 関数を記述した場合，XSLT では "0" が出力されますが，XSLTC では "-0" が出力されます。
24	xsl:variable 要素の子要素に xsl:call-template 要素を指定し，かつ，xsl:call-template で呼び出されたテンプレート内に名前空間接頭辞を持つリテラル要素がある場合，xsl:copy-of 要素でその変数が示す要素をコピーすると，要素の名前空間宣言が正しくコピーされません。
25	リテラル結果要素に xsl:extension-element-prefixes 属性を指定して拡張要素とした場合，拡張要素の子要素として書かれた xsl:fallback 要素が無視されます。代わりに xsl:stylesheet 要素に extension-element-prefixes 属性を指定してください。
26	スタイルシート内で仕様上，許されない場所に子要素を書いた場合（例：xsl:stylesheet 要素の子要素としてリテラル要素を書いた場合，xsl:namespace-alias 要素の子要素として xsl:text 要素が現れた場合など），エラーにはならないで，不正な子要素が無視されます。

6.9.4 XPath 式の注意事項

表 6-16 XSLTC に関する注意事項（XPath 式）

項番	注意事項
1	document 関数に XPath 式を続けて指定する場合，"//" は使用しないで，代わりに "/descendant-or-self::node()/" を使用してください。
2	最上位要素の中で current 関数を使用しないでください。
3	XPath 式の中でノードテスト "node()" を使用した場合，ノードを正しく取得できません。
4	XPath 式に，"*" を含むノードテストと論理演算子を組み合わせた XPath 式を書いた場合，XPath 式の構文解析中にエラーが発生します。
5	変数およびパラメタの値が，結果ツリーフラグメント（select 属性ではなく，要素内容として書かれたテンプレートによって定義される値）の場合，そのテンプレートの中に変数参照があると，NullPointerException が発生する場合があります。

6.9.5 エラーを通知しないケース

要素や属性の指定内容が不当な場合や，関数の引数が不正な場合など，XSLTC がエラーを通知しないケースがあります。XSLTC がエラーを通知しないケースを次の表に示します。

表 6-17 XSLTC に関する注意事項（エラーを通知しないケース）

項番	注意事項
1	xsl:apply-templates 要素の子要素として xsl:sort 要素と xsl:with-param 要素以外のスタイルシート要素を指定した場合。
2	xsl:apply-templates 要素の select 属性に空文字列を指定した場合。
3	xsl:comment 要素の要素内容にテキスト以外を指定した場合。
4	xsl:decimal-format 要素の次に挙げた属性に 2 文字以上の値を指定した場合。 • decimal-separator • grouping-separator • minus-sign • percent • per-mille • zero-digit • pattern-separator
5	xsl:element 要素または xsl:attribute 要素の name 属性に接頭辞で修飾した要素名を指定し，かつ，namespace 属性に空文字列を指定した場合。
6	xsl:for-each 要素の先頭以外の子要素として xsl:sort を指定した場合。
7	xsl:import 要素に要素内容を指定した場合。
8	xsl:import 要素がトップレベル要素の先頭要素でない場合や，xsl:import 要素や xsl:include 要素がトップレベル要素でない場合。
9	xsl:key 要素の use 属性に key 関数を含む XPath 式を指定した場合。
10	xsl:message 要素を xsl:stylesheet 要素の子要素として指定した場合。
11	xsl:message 要素の terminate 属性に "yes"，"no" 以外の値を指定した場合。
12	xsl:namespace-alias 要素の stylesheet-prefix 属性や result-prefix 属性に指定した接頭辞に対応する名前空間宣言がない場合。
13	xsl:namespace-alias 要素の stylesheet-prefix 属性や result-prefix 属性の指定がない場合。
14	xsl:namespace-alias 要素に要素内容を指定した場合。
15	xsl:number 要素の grouping-size 属性に Number 規則に合致しない文字列を指定した場合。
16	xsl:number 要素の lang 属性および level 属性に不当な値を指定した場合。
17	xsl:number 要素の letter-value 属性に不当な値を指定した場合。
18	xsl:number 要素の grouping-separator 属性に，値が空文字となる変数参照を指定した場合。
19	xsl:output 要素の version 属性に不当な値を指定した場合。
20	xsl:output 要素の indent 属性に不当な値を指定した場合。
21	xsl:output 要素の standalone 属性に不当な値を指定した場合。

項番	注意事項
22	xsl:output 要素の omit-xml-declaration 属性に不当な値を指定した場合。
23	xsl:param 要素の select 属性に空文字列を指定した場合。
24	xsl:sort 要素の data-type 属性，order 属性，または case-order 属性に不正な値を指定した場合。
25	xsl:sort 要素の data-type 属性，order 属性，または case-order 属性が空文字列になる変数参照である場合。
26	xsl:sort 要素の data-type 属性，order 属性，または case-order 属性に大文字の値を指定した場合，小文字で指定されたものとして扱われます。
27	xsl:strip-space 要素，xsl:preserve-space 要素，xsl:apply-imports 要素など，仕様上は要素内容を持てない要素に要素内容を指定した場合。
28	xsl:stylesheet 要素の version 属性に不当な値を指定した場合。
29	xsl:stylesheet 要素の exclude-result-prefixes 属性に指定した接頭辞に対応する名前空間 URI が，この属性を持つ要素で有効でない場合。
30	xsl:stylesheet 要素の中に xsl:stylesheet 要素を指定した場合。
31	xsl:template 要素の match 属性および mode 属性に空文字列を指定した場合。
32	xsl:template 要素に name 属性の指定がなく，match 属性の指定もない場合。
33	xsl:template 要素または xsl:apply-template 要素の mode 属性に空文字列を指定した場合。
34	xsl:template 要素の priority 属性に空文字列を指定した場合。
35	最初の xsl:template 要素より前にリテラル結果要素が現れた場合。
36	xsl:template 要素に mode 属性があり，match 属性がない場合。
37	複数の xsl:with-param 要素が同じ値の name 属性を持っている場合。
38	xsl:text 要素の disable-output-escaping 属性に "yes"，"no" 以外の値を指定した場合。
39	xsl:value-of 要素の disable-output-escaping 属性に "yes" または "no" 以外の値を指定した場合。
40	xsl:variable 要素の select 属性に空文字列を指定した場合。
41	xsl:variable 要素または xsl:param 要素に，select 属性とテキスト内容の両方が存在する場合。
42	xsl:with-param 要素の select 属性に空文字列を指定した場合。
43	次の関数呼び出しの引数の数が不正な場合。 • number 関数 • boolean 関数 • lang 関数 • concat 関数
44	document 関数の第二引数が空文字列の場合。
45	format-number 関数の第三引数に空文字列を指定した場合。
46	key 関数の第一引数に存在しないキー名を指定した場合。
47	unparsed-entity-uri 関数の引数の個数が不正な場合。

6.9.6 その他の注意事項

表 6-18 XSLTC に関する注意事項（その他）

項番	注意事項
1	XSLTC を使った場合，デフォルトのエラーリスナーはエラー時に例外をスローします。
2	ErrorListener の warning メソッドや error メソッドの実装が例外をスローしない場合，スタイルシート解析時に警告やエラーが発生するとトランスレットの作成ができないため，変換ができません。
3	エラーメッセージ中のエラー ID の前に，エラー原因となったファイルのファイル名と行番号が出力される場合があります。
4	SAXTransformerFactory クラスの newTemplatesHandler メソッドで作成した TemplatesHandler オブジェクトを XMLReader クラスの setContentHandler メソッドでパーサに登録して変換処理を行う場合，エラーのあるスタイルシートを処理してもエラーが出力されません。
5	SAXTransformerFactory を使って変換を行う場合，エラーリスナーの設定にかかわらず，スタイルシートのエラーがエラーリスナーに通知されません。
6	XSLTC を使って変換を行う場合，一つの xsl:namespace-alias 要素の stylesheet-prefix 属性と result-prefix 属性の両方に "#default" を指定すると，NullPointerException が発生します。
7	XSLTC を使って変換を行う場合，インポート優先順位が異なる二つのテンプレートが name 属性と match 属性を持ち，name 属性の値が同じであるとき，インポート優先順位が低い方のテンプレートが無視されます。
8	XSLTC を使って変換を行う場合，スタイルシートと JAXP インタフェースの両方で出力プロパティの cdata-section-elements 属性を設定すると，スタイルシート内で定義された値が JAXP インタフェースで設定された値によって上書きされます。
9	XSLTC を使って変換を行う場合，DocumentFragment から作った DOMSource を返すように実装した URIResolver を使用すると，正しい変換処理が行われません。
10	出力メソッドに HTML 出力を指定した場合，xsl:element 要素の name 属性に "{name()}" または "{name(.)}" を指定すると，NullPointerException が発生します。

6.10 javax.xml.datatype パッケージに関する注意事項

javax.xml.datatype パッケージに関する注意事項を次の表に示します。

表 6-19 javax.xml.datatype パッケージに関する注意事項

項番	注意事項
1	マルチスレッドプログラミングをする場合，javax.xml.datatype パッケージで規定されたオブジェクトはスレッドセーフではありません。したがって，各スレッドは排他的にこれらのオブジェクトにアクセスするようにしてください。
2	DatatypeFactory クラスの各メソッドで IllegalArgumentException が発生したとき，getMessage() メソッドでは，各メソッドに指定したパラメタが戻ります。
3	DatatypeFactory クラスの newXMLGregorianCalendar メソッド，newXMLGregorianCalendarDate メソッド，newXMLGregorianCalendarTime メソッドで，月を表すフィールドに 4，6，9，11 のどれかを設定したとき，日を表すフィールドに 31 を設定してもエラーが発生しません。
4	DatatypeFactory クラスの newXMLGregorianCalendar メソッドおよび newXMLGregorianCalendarTime メソッドの引数 millisecond には，0 から 999 までの値を指定してください。
5	月を表すフィールドが 1 の Duration オブジェクトと，日を表すフィールドが 28 の Duration オブジェクトを比較したとき，結果は次のようになります。 • compare メソッドで比較したときは，DatatypeConstants.EQUAL が戻ります。 • equals メソッドで比較したときは，true が戻ります。
6	DatatypeFactory クラスの newDuration(long durationInMilliSeconds) メソッド，newDurationDayTime(long durationInMilliseconds) メソッド，または newDurationYearMonth(long durationInMilliseconds) メソッドで作成した Duration オブジェクトは，DatatypeConstants.DURATION_DAYTIME 型になります。したがって，この Duration オブジェクトの月を表すフィールド，日を表すフィールドの値は 0 になります。
7	XMLGregorianCalendar クラスの setHour メソッド，および setTime メソッドの引数 hour には，0 から 23 までの値を指定してください。また，DatatypeFactory クラスの newXMLGregorianCalendar メソッド，newXMLGregorianCalendarTime メソッドの引数の「時間を表すフィールド」には，0 から 23 までの値を指定してください。
8	JAXP1.3 仕様書では，XMLGregorianCalendar クラスの isValid メソッドは「Validate instance by getXMLSchemaType() constraints.」と定義されています。Cosminexus XML Processor では，次の制約をすべて満たしているときに true を返します。 • 月を表すフィールドが 2 のとき，日を表すフィールドが 28 以内（うるう年以外の場合），または 29 以内（うるう年の場合）である。 • 年を表すフィールドが 0 ではない。
9	XMLGregorianCalendar クラスの equals メソッドの引数が null のとき，NullPointerException 例外は発生せずに false が戻ります。
10	DatatypeConstants.GMONTH 型の XMLGregorianCalendar オブジェクトに toString，toXMLFormat メソッドを適用して得られる字句表現は，W3C XML Schema 1.0 Part 2（second edition）仕様書のセクション 3.2.14 で規定された文字列「--MM」ではなく，second edition より前の仕様書で規定された文字列「--MM--」になります。

6.11 javax.xml.validation パッケージに関する注意事項

javax.xml.validation パッケージに関する注意事項を次の表に示します。

表 6-20 javax.xml.validation パッケージに関する注意事項

項番	注意事項
1	マルチスレッドプログラミングをする場合，SchemaFactory クラス，Validator クラス，ValidatorHandler クラス，TypeInfoProvider クラスはスレッドセーフではありません。したがって，各スレッドは排他的にこれらのオブジェクトにアクセスするようにしてください。
2	SchemaFactory クラスの newInstance(String schemaLanguage) メソッドの引数に指定可能な URI は "http://www.w3.org/2001/XMLSchema" だけです。
3	SchemaFactory クラスの isSchemaLanguageSupported メソッドには，"http://www.w3.org/2001/XMLSchema" を指定してください。その他のスキーマ言語を指定した場合は，false が戻ります。
4	SchemaFactory クラスの newSchema(Source[] schemas) メソッドの引数 schemas の配列要素には，それぞれ異なる targetNamespace を持つスキーマ文書を指定してください。同一の targetNamespace を持つスキーマ文書，または targetNamespace を持たないスキーマ文書を複数個指定すると，インデックスの小さい配列要素のスキーマ文書が優先されます。
5	次の条件で SchemaFactory クラスの newSchema(Source[] schemas) メソッドを実行すると，例外が発生しない場合があります。 • 引数 schemas の配列要素が，正しいスキーマ文書と不正なスキーマ文書の両方を含んでいる。 この問題は，次のようにして回避できます。 1. SchemaFactory オブジェクトのエラーハンドラで，error, fatalError メソッドに渡される例外オブジェクトを保存しておく。 2. newSchema メソッドの呼び出し後に，1. で例外オブジェクトが保存されたかどうかチェックして，保存されていればその例外オブジェクトをスローする。
6	Validator クラスの validate(Source)，validate(Source, Result) の引数には，表 6-21 に示す組み合わせで指定してください。
7	Validator クラスの validate メソッドによって，ID 型の属性の定義を含むスキーマ文書を用いて XML 文書を検証する場合は，validate(DOMSource, DOMResult) ではなく，validate(SAXSource)，validate(DOMSource)，validate(SAXSource, SAXResult) のどれかのメソッドを使用してください。
8	次の条件がすべて重なるとき，Validator クラスの validate(Source source, Result result) メソッドを実行すると，DOMException 例外が発生する場合があります。 1. 引数 result が DOMResult オブジェクトである。 2. DOMResult オブジェクトに設定された「結果ツリーを含む DOM ノード」が子ノードを持つ Document オブジェクトである。 したがって，2. の Document オブジェクトには子ノードを持たせないようにしてください。
9	スキーマ文書の中に xsd:ENTITY 型，および xsd:ENTITIES 型が含まれるとき，ValidatorHandler クラスを用いたスキーマ検証はサポートしていません。このような場合は，例えば Validator クラスなどのほかの方法でスキーマ検証するようにしてください。

項番	注意事項
10	スキーマ文書で定義されていない要素および属性から取得した TypeInfo オブジェクトは，妥当性検証失敗時には，xsd:anyType 型の TypeInfo オブジェクトと同じになります。
11	Validator クラスの validate(Source) および validate(Source, Result) の引数 Source に DOMSource オブジェクトを指定する場合，DOMSource オブジェクトが保持する Node オブジェクトは，次に示すすべての条件に従う必要があります。 1. 名前空間を有効に設定したパーサによって生成されている。 例えば，setNamespaceAware(true) を適用した DocumentBuilderFactory で生成した DocumentBuilder など。 2. 妥当性検証をしないパーサによって生成されている。 例えば，setValidating (false) を適用した DocumentBuilderFactory で生成した DocumentBuilder など。
12	Validator クラスの validate(Source)，validate(Source, Result) の引数 Source に SAXSource オブジェクトを指定する場合に，その SAXSource オブジェクトが XMLReader オブジェクトを使用するとき，XMLReader オブジェクトは次に示すすべての条件に従う必要があります。 1. 名前空間を有効に設定した XMLReader オブジェクトである。 例えば，setNamespaceAware(true) を適用した SAXParserFactory で SAXParser を生成し，その SAXParser から取り出した XMLReader オブジェクトなど。 2. 妥当性検証をしない XMLReader オブジェクトである。 例えば，setValidating (false) を適用した SAXParserFactory で SAXParser を生成し，その SAXParser から取り出した XMLReader オブジェクトなど。
13	次の条件がすべて重なるとき，例外コードが NO_MODIFICATION_ALLOWED_ERR である DOMException 例外が発生する場合があります。 1. SchemaFactory クラスの newSchema(Source) および newSchema(Source[]) の引数に DOMSource オブジェクトを指定して，これらのメソッドを実行する。 2. 1. の DOMSource オブジェクトがラップしている DOM ツリーを更新する。
14	javax.xml.validation.SchemaFactory クラスの newSchema メソッドに，XMLReader を設定した SAXSource を指定する場合，指定する XMLReader は次の条件に従う必要があります。 • SAX2 のフィーチャー "http://xml.org/sax/features/namespace-prefixes" の値が "true" である。

表 6-21 Validator クラスの validate(Source)，validate(Source, Result) の引数の組み合わせ

引数 Result	引数 Source			
	SAXSource	DOMSource	StreamSource	null
SAXResult	○	×	×	×
DOMResult	×	○	×	×
StreamResult	×	×	×	×
null	○	○	×	×

（凡例）

○：組み合わせることができます。

×：組み合わせることはできません。

6.12 javax.xml.xpath パッケージに関する注意事項

javax.xml.xpath パッケージに関する注意事項を次の表に示します。

表 6-22 javax.xml.xpath パッケージに関する注意事項

項番	注意事項
1	マルチスレッドプログラミングをする場合，XPathFactory クラスはスレッドセーフではありません。したがって，複数のスレッドが同時に同一の XPathFactory インスタンスにアクセスしてはいけません。スレッド間の競合を避けるため，次のどちらかの方法を使用してください。 • 各スレッドに一つの XPathFactory インスタンスを持つ。 • 各スレッドが排他的に XPathFactory インスタンスにアクセスする。
2	マルチスレッドプログラミングをする場合，XPathFactory クラスの newInstance メソッドで生成される XPath インスタンス，および XPath インタフェースの compile メソッドで生成される XPathExpression インスタンスはスレッドセーフではありません。したがって，複数のスレッドが同時に同一の XPath インスタンス，および XPathExpression インスタンスにアクセスしてはいけません。スレッド間の競合を避けるため，次の方法を使用してください。 • 各スレッドに一つの XPath インスタンス，および XPathExpression インスタンスを持つ。
3	XPathFactory クラスの newInstance(String uri) メソッドの引数に指定可能な URI は "http://java.sun.com/jaxp/xpath/dom" だけです。
4	次のメソッドを用いて，コンテキストを参照するような XPath 式を評価する場合は，引数 item に null ではないコンテキストを指定してください。 • XPath インタフェースの evaluate(String expression, Object item, QName returnType) メソッド • XPath インタフェースの evaluate(String expression, Object item) メソッド • XPathExpression インタフェースの evaluate(Object item, QName returnType) メソッド • XPathExpression インタフェースの evaluate(Object item) メソッド
5	この表の項番 4 で示したメソッドの引数 item には，Document，DocumentFragment，Element，Text，Attr，ProcessingInstruction，Comment のどれかのオブジェクトを指定してください。
6	XPathFunction インタフェースを実装するクラスの evaluate メソッドの戻り値の型，および，XPathVariableResolver インタフェースを実装するクラスの resolveVariable メソッドの戻り値の型は，java.lang.String，java.lang.Boolean，java.lang.Number，org.w3c.Node，org.w3c.NodeList のどれかにしてください。
7	javax.xml.xpath パッケージのメソッドで例外が発生した場合，発生した例外オブジェクトに getMessage メソッドを適用した戻り値が null になり詳細メッセージを取得できないことがあります。この場合，例外オブジェクトに getCause メソッドを適用して「ラップされた例外オブジェクト」を取得し，これに getMessage メソッドを適用することで，詳細なメッセージを取得できる場合があります。
8	and，or，mod，div 演算子の前後には，空白を入れてください。 （例）"1000 div 10"
9	次のすべての条件が重なる XPath 式はサポートしません。 1. ｜演算子のオペランドのそれぞれにノードセットを返す XPath 関数を指定して，ノードセットの和集合を作成する。 2. 1. で作成したノードセットの和集合に述語を適用する。

項番	注意事項
10	次のすべての条件が重なる XPath 式はサポートしません。 1. local-name，namespace-uri，name のどれかの関数を使用する。 2. 1. で示した関数の引数に次の式を指定する。 self::node()/descendant::prefix1:*（※） この場合，1. で示した関数の引数を次の式に変更することで回避できます。 self::node()/self::node()/descendant::prefix1:*（※） 注※：prefix1 は，任意の名前空間接頭辞を表します。
11	属性ノード，名前空間ノードに following-sibling 軸を適用した XPath 式はサポートしません。
12	| 演算子のオペランドがロケーションセット以外の XPath 式はサポートしません。
13	名前空間接頭辞と名前空間 URI の組が等しい名前空間ノードが複数存在するとき，それらの名前空間ノードに軸を適用する XPath 式はサポートしません。
14	値が重複した ID 型属性を含む XML 文書に対して id 関数を適用した場合，id 関数の結果が不正になることがあります。

6.13 org.w3c.dom パッケージに関する注意事項

org.w3c.dom パッケージに関する注意事項を次の表に示します。

表 6-23 org.w3c.dom パッケージに関する注意事項

項番	注意事項
1	DOMConfiguration オブジェクトに設定可能なパラメタを表 6-24 に示します。設定可能なパラメタは，getParameterNames メソッドの戻り値，および canSetParameter メソッドの戻り値とは一致しません。また，設定不可の値を設定しても，例外が発生しない場合があります。設定不可の値を設定した場合の動作は保証しません。
2	DOMConfiguration オブジェクトのパラメタ名は，大文字と小文字が区別されません。
3	DOM Level 3 では，従来の DOM Level 2 のインタフェースに対して新しいメソッドが追加されています。例えば，Attr インタフェースには isId メソッドが追加されています。通常のアプリケーションプログラムがこれらのインタフェースを implements することはありませんが，もし implements している場合は，DOM Level 3 で追加されたメソッドの実装をアプリケーションに追加する必要があります。
4	XMLSchema で属性のデフォルト値を定義して，その属性に対応する Attr ノードを削除したとき，getSpecified メソッドの戻り値が false である属性ノードは生成されません。
5	Document オブジェクトの normalizeDocument メソッドはサポートしていません。
6	Document オブジェクトの adoptNode メソッドはサポートしていません。代わりに，Document オブジェクトの importNode メソッドと Node オブジェクトの removeChild メソッドを組み合わせて使用してください。
7	Document オブジェクトの getInputEncoding メソッドおよび Entity オブジェクトの getInputEncoding メソッドはサポートしていません。
8	次の条件がすべて重なるとき，Document オブジェクトの renameNode(Node n, String namespaceURI, String qualifiedName) メソッドを実行しても DOMException 例外は発生しません。 • 引数 n が名前空間をサポートしない Attr オブジェクトである。 • 引数 namespaceURI が null である。 • 引数 qualifiedName が不正な修飾名である。 また，引数 n に Element または Attr オブジェクトを指定し，かつ，引数 qualifiedName に null を指定した場合，DOMException 例外ではなく NullPointerException 例外が発生します。
9	Attr オブジェクト，Text オブジェクト，EntityReference オブジェクト，CDATASection オブジェクト，DocumentType オブジェクト，DocumentFragment オブジェクトの getBaseURI メソッドでは URI の取得ができません。
10	Text オブジェクトの replaceWholeText(String) メソッドは EntityReference ノードを含まない DOM ツリーに適用してください。
11	DOMLocator オブジェクトの getByteOffset の戻り値は常に -1 です。
12	DOMErrorHandler オブジェクトの handleError メソッドの引数 DOMError に getRelatedData メソッドを適用したときの戻り値のオブジェクトは，Node オブジェクトとは限りません。エラーメッセージを表す String オブジェクトが戻り値となる場合があります。

項番	注意事項
13	パースして取得した Document オブジェクトに対して，insertBefore メソッドまたは appendChild メソッドで不当なノードを追加した場合でも，DOMException 例外が発生しないことがあります。
14	次のすべての条件に該当する場合，要素ノードに対する getSchemaTypeInfo メソッド呼び出しの戻り値が不正になります。 1. XML スキーマ文書で union による派生型の要素を定義する。 2. 1. のスキーマ文書を使って妥当性検証パースを実施する。 3. 2. によって作成した DOM から，1. に該当する要素ノードを取得し，getSchemaTypeInfo メソッドを呼び出す。

表 6-24 DOMConfiguration オブジェクトに設定可能なパラメタ

パラメタの名前	設定可能な値
canonical-form	false
cdata-sections	true，false
check-character-normalization	false
comments	true，false
datatype-normalization	false
element-content-whitespace	true
entities	true，false
error-handler	DOMErrorHandler オブジェクト
infoset	true，false
namespaces	true
namespace-declarations	true，false
normalize-characters	false
schema-location	設定できません。
schema-type	設定できません。
split-cdata-sections	true，false
validate	false
validate-if-schema	false
well-formed	true
resource-resolver	LSResourceResolver オブジェクト

6.14　org.w3c.dom.bootstrap パッケージに関する注意事項

org.w3c.dom.bootstrap パッケージに関する注意事項を次の表に示します。

表 6-25　org.w3c.dom.bootstrap パッケージに関する注意事項

項番	注意事項
1	DOMImplementationRegistry クラスの getDOMImplementation メソッド，および getDOMImplementationList メソッドの引数とする文字列には空文字列以外を指定してください。

6.15 org.w3c.dom.ls パッケージに関する注意事項

org.w3c.dom.ls パッケージに関する注意事項を次の表に示します。

表 6-26 org.w3c.dom.ls パッケージに関する注意事項

項番	注意事項
1	DOMImplementationLS インタフェースの createLSParser(short mode, String schemaType) メソッドの引数 mode には DOMImplementationLS. MODE_SYNCHRONOUS だけを指定できます。また，引数 schemaType には null，"http://www.w3.org/2001/XMLSchema"，"http://www.w3.org/TR/REC-xml" のどれかを指定できます。
2	LSParser オブジェクトに getDomConfig メソッドを適用して得られる DOMConfiguration オブジェクトに設定可能なパラメタを表 6-27 に示します。これに加えて，表 6-24 に示したパラメタも設定可能です。設定可能なパラメタは，getParameterNames メソッドの戻り値，および canSetParameter メソッドの戻り値とは一致しません。また，設定不可の値を設定しても，例外が発生しない場合があります。設定不可の値を設定した場合の動作は保証しません。
3	LSParser オブジェクトの charset-overrides-xml-encoding パラメタに値を設定しても，無視されます。
4	LSParser インタフェースの parseWithContext メソッドはサポートしていません。
5	LSParser オブジェクトの setNewLine メソッドはサポートしていません。
6	LSParser オブジェクトでは，XML1.1 文書を解析できません。
7	LSParser オブジェクトの startElement メソッドは，ルート要素については呼び出されません。
8	LSParser オブジェクトでの解析時，および LSSerializer オブジェクトでの直列化時に LSException 例外が発生したとき，getMessage() メソッドの戻り値が null になる場合があります。この場合は，LSParserFilter や LSSerializerFilter のユーザ実装クラスで例外が発生している可能性があります。
9	LSSerializer オブジェクトに getDomConfig メソッドを適用して得られる DOMConfiguration オブジェクトに設定可能なパラメタを表 6-28 に示します。これに加えて，表 6-24 に示したパラメタも設定可能です。設定可能なパラメタは，getParameterNames メソッドの戻り値，および canSetParameter メソッドの戻り値とは一致しません。また，設定不可の値を設定しても，例外が発生しない場合があります。設定不可の値を設定した場合の動作は保証しません。
10	LSSerialize オブジェクトの write，writeToString，および writeToURI メソッドの引数 nodeArg に，Document，DocumentFragment，Element 以外のノードを指定した場合，それらのノードは直列化されません。
11	LSSerialize オブジェクトの write，writeToString，および writeToURI メソッドは，名前空間が無効なノードを直列化できません。
12	LSSerializer インタフェースの getNewLine メソッドで得られる行末シーケンス文字のデフォルト値は，"¥n" です。
13	ProcessingInstruction オブジェクトが持つ処理命令の内容に文字列「?>」が含まれるとき，その ProcessingInstruction オブジェクトを直列化してもエラーは通知されません。
14	LSParser でパースした場合，Document オブジェクトから取得した DocumentType オブジェクトの内容を変更しても DOMException 例外が発生しないことがあります。

表 6-27 DOMConfiguration オブジェクトに設定可能なパラメタ（LSParser オブジェクト）

パラメタの名前	設定可能な値
charset-overrides-xml-encoding	true，false
disallow-doctype	false
ignore-unknown-character-denormalizations	true
supported-media-types-only	false

表 6-28 DOMConfiguration オブジェクトに設定可能なパラメタ（LSSerializer オブジェクト）

パラメタの名前	設定可能な値
discard-default-content	true，false
format-pretty-print	false
ignore-unknown-character-denormalizations	true
xml-declaration	true，false

6.16 org.xml.sax.ext パッケージに関する注意事項

org.xml.sax.ext パッケージに関する注意事項を次の表に示します。

表 6-29 org.xml.sax.ext パッケージに関する注意事項

項番	注意事項

1	Attributes2Impl クラスの addAttribute メソッドはサポートしていません。
	代わりに，AttributesImpl クラスの addAttribute メソッドを使用し，アプリケーションを次のように変更してください。
	（変更前）
	`Attributes2Impl myAtt = new Attributes2Impl();`
	`String type = "ID";`
	`myAtt.addAttribute("http://hitachi-xmlprocessor.com/", "attr",`
	`"ht:attr", type, "value");`
	`:`
	（変更後）
	`AttributesImpl myAtt = new AttributesImpl();`
	`String type = "ID";`
	`myAtt.addAttribute("http://hitachi-xmlprocessor.com/", "attr",`
	`"ht:attr", type, "value");`
	`Attributes2Impl myAtt2 = new Attributes2Impl(myAtt);`
	`int position = myAtt2.getLength() -1;`
	`myAtt2.setSpecified(position, true);`
	`myAtt2.setDeclared (position, !"CDATA".equals(type));`
	`:`

2	Locator2Impl クラスの getXMLVersion メソッド，getEncoding メソッドはサポートしていません。

6.17 XInclude に関する注意事項

XInclude に関する注意事項を次の表に示します。

表 6-30 XInclude に関する注意事項

項番	注意事項
1	xi:include 要素の xpointer 属性に指定可能なスキーマは，element() スキーマと簡略ポインタだけです。
2	xi:include 要素の xpointer 属性に簡略ポインタを指定する場合，その簡略ポインタは「DTD で ID 型と定義した属性」の属性値である必要があります。「XML Schema で xsd:ID 型と定義した属性」の属性値は指定できません。

6.18 実装依存仕様に関する注意事項

XML Schema の実装に依存した仕様を次の表に示します。

表 6-31 データ型に関する実装依存仕様

データ型			最大値	最大桁数
duration 型 (形式：P*n*Y*n*M*n*DT*n*H*n*M*n*S)	*n*Y(年)		○	−
	*n*M(月)		○	
	*n*D(日)		○	
	*n*H(時)		○	
	*n*M(分)		○	
	*n*S(秒)	1秒以上	○	
		1秒未満	○	
date 型 (形式：CCYY-MM-DD)	CCYY ("9999" を超える場合，桁数の追加ができる)		○	
gYearMonth 型 (形式：CCYY-MM)				
gYear 型 (形式：CCYY)				
dateTime 型 (形式： CCYY-MM-DDThh:mm:ss)	1秒未満 (ss に続く，小数点以下に1秒未満の値を指定できる)		○	
time 型 (形式：hh:mm:ss)				
base64Binary 型			△	◎
hexBinary 型				◎
decimal 型				◎
integer 型				◎
nonPositiveInteger 型				◎
nonNegativeInteger 型				◎
negativeInteger 型				◎
positiveInteger 型				◎

(凡例)

○：指定できる値は，最大 2147483647 です。

△：指定する値の最大値の規定はありません。

◎：指定できる最大桁数は，メモリに依存します。

−：該当しません。

表 6-32 制約ファセットに関する実装依存仕様

制約ファセット	最大値	最大桁数
length	○	－
minLength	○	
maxLength	○	
totalDigits	○	
fractionDigits	○	
maxInclusive	△	◎
maxExclusive		◎
minInclusive		◎
minExclusive		◎

（凡例）

○：指定できる値は，最大 2147483647 です。

△：指定する値の最大値の規定はありません。

◎：指定できる最大桁数は，メモリに依存します。

－：該当しません。

表 6-33 出現回数の指定に関する実装依存仕様

出現回数の指定	最大値	最大桁数
minOccurs	○	－
maxOccurs	○	

（凡例）

○：指定できる値は，最大 2147483647 です。

－：該当しません。

6.19 スキーマキャッシュ機能に関する注意事項

スキーマキャッシュ機能に関する注意事項を次に示します。

6.19.1 性能に関する注意事項

ディスクキャッシュを使用した場合，キャッシュをファイルから読み込んでオブジェクトに展開する処理の分だけ，オーバーヘッドが大きくなります。そのため，キャッシュを使用しない場合より性能が低下することがあります。

6.19.2 エラー出力に関する注意事項

キャッシュの構築中に発生したエラーは，標準エラー出力ストリームではなく，エラーファイルに出力されます。エラーファイルに出力された内容を確認するには，J2EE サーバの起動後に cachestate コマンドに -d オプションを指定して実行し，エラー情報を出力してください。

6.19.3 スキーマ定義ファイルに関する注意事項

スキーマ定義ファイルに関する注意事項を次の表に示します。

表 6-34 スキーマ定義ファイルに関する注意事項

項番	注意事項
1	schema_xxx プロパティの設定値の末尾に記載された ",D" は，ディスクキャッシュ指定子と見なされるため，",D" で終わるスキーマ文書ファイル名は使用しないでください。
2	xsd:import によって参照されるスキーマ文書は，スキーマ定義ファイルの指定とは関係なく，常にメモリ上にキャッシュされます。xsd:include または xsd:redefine によって参照されるスキーマ文書は，参照元のスキーマ文書の grammar オブジェクトに吸収されるため，キャッシュ場所（メモリまたはディスク）も参照元のスキーマ文書と同じになります。
3	検証で使用するスキーマ文書が，http://java.sun.com/xml/jaxp/properties/schemaSource プロパティで「InputStream」および「オブジェクトの配列」の形で指定された場合，検証時にキャッシュが使用されません。 キャッシュを使用できるのは，スキーマ文書が「String」，「File」，「InputSource」のどれかの形で指定された場合だけです。ただし，InputSource の場合，getSystemId メソッドでシステム識別子が取得できる必要があります。
4	キャッシュの構築中にパースエラーが発生した場合，エラーレベルにかかわらず，エラーの原因となったスキーマ文書およびその文書を参照するスキーマ文書はキャッシュされません。
5	＜ JAXP_DIR ＞※の下の cache_info ディレクトリ（INFO ディレクトリ）以下には，スキーマキャッシュ機能がキャッシュ構築・削除で使用する内部ファイル（INFO ファイル）が格納されます。このため，＜ JAXP_DIR ＞※のディスク使用量が 100% を超えた場合，キャッシュ機能が正常に動作しなくなるおそれがあります。

項番	注意事項
6	diskcache_path プロパティで指定したディレクトリ以下のディレクトリやファイルは，スキーマキャッシュ機能によって変更・削除されます。そのため，diskcache_path プロパティにはシステムが使用するディレクトリを指定しないでください。
7	エンティティリゾルバを使用して，外部参照の解決を行う必要のある XML スキーマ文書をスキーマ定義ファイルに指定した場合，キャッシュ構築時に外部参照の解決が行えません。そのため，スキーマ文書の解析に失敗し，キャッシュが構築されません。

注※

Cosminexus をインストールしたディレクトリ下の，jaxp ディレクトリのことです。

6.19.4 キャッシュの再構築・削除に関する注意事項

キャッシュの再構築・削除に関する注意事項を次の表に示します。

表 6-35 キャッシュの再構築・削除に関する注意事項

項番	注意事項
1	キャッシュの再構築時は，スキーマ定義ファイルに指定されたすべてのスキーマ文書を解析して grammar オブジェクトを構築するため，パース処理に掛かる時間が増加します。 このため，cacheon コマンド実行後の最初のパースは，通常よりも処理に時間が掛かります。
2	キャッシュの再構築時，および J2EE サーバ起動・終了時には，diskcache_path で指定したディレクトリ以下にある，J2EE サーバ名と同じ名前のディレクトリが削除されます。このため，diskcache_path で指定したディレクトリに，J2EE サーバ名と同じ名前のファイルまたはディレクトリを生成しないでください。
3	キャッシュ構築後に，スキーマ定義ファイルの diskcache_path のディレクトリ指定を変更した場合，J2EE サーバの終了時や，cacheoff コマンドおよび cacheon コマンド実行時にディスクキャッシュが削除されないおそれがあります。その場合，残っているファイルを手動で削除してください。

6.20 高速パース機能に関する注意事項

高速パース機能に関する注意事項を次に示します。

6.20.1 解析結果オブジェクトの設定に関する注意事項

表 6-36 高速パース機能に関する注意事項（解析結果オブジェクトの設定）

項番	注意事項
1	事前解析時の名前空間の有効・無効は，解析時の名前空間の有効・無効に合わせる必要があります。これらが異なる場合，解析結果が不正になるおそれがあります。
2	エンティティ参照を解決するとき，事前解析時のエンティティリゾルバで解決した結果と，解析時のエンティティリゾルバで解決した結果は一致している必要があります。これらの結果が異なる場合，高速化の効果が得られないおそれがあります。

6.20.2 解析結果オブジェクトを利用する XML パーサに関する注意事項

表 6-37 高速パース機能に関する注意事項（解析結果オブジェクトを利用する XML パーサ）

項番	注意事項
1	解析結果オブジェクトを利用するパーサの次のフィーチャーを `false` に設定した場合，高速化の効果が得られない場合があります。 1. http://xml.org/sax/features/external-general-entities 2. http://xml.org/sax/features/external-parameter-entities

6.20.3 XML1.1 に関する注意事項

表 6-38 高速パース機能に関する注意事項（XML1.1）

項番	注意事項
1	高速パース機能では，XML1.1 はサポートしていません。このため，XML1.1 文書を事前解析するとエラーが発生します。また，XML1.1 文書を高速パース機能の対象に設定しても，高速パース機能は適用されません。

6.20.4 XInclude に関する注意事項（高速パース機能）

表 6-39 高速パース機能に関する注意事項（XInclude）

項番	注意事項
1	XInclude で取り込むファイルの解析に対しては，高速パース機能は適用されません。

6.20.5 絶対パスの指定方法に関する注意事項

表 6-40 高速パース機能に関する注意事項（絶対パスの指定方法）

項番	注意事項
1	絶対パスを指定する場合は，パス区切り文字に "/" を使用してください。
2	Windows の場合，絶対パスにはドライブ名を含めてください。 Windows の場合と UNIX の場合の，絶対パスの指定例を次に示します。 Windows の場合 D:/home/xml/training/training1.xml UNIX の場合 /home/xml/training/training1.xml

6.20.6 事前解析用 XML 文書に関する注意事項

表 6-41 高速パース機能に関する注意事項（事前解析用 XML 文書）

項番	注意事項
1	事前解析用 XML 文書に記述できる要素名および属性名の長さには上限があります。事前解析用 XML 文書に記述されている最も長い要素名および属性名の文字数を足した長さ※が 21,000 文字を超える場合，KECX09607-E のエラーが発生して処理が中止されることがあります。メッセージの詳細については，マニュアル「Cosminexus アプリケーションサーバ メッセージ 3」の「2. KECX（Cosminexus XML Processor が出力するメッセージ）」を参照してください。

注※

例えば，次の XML 文書では，最も長い要素名（element または content）の文字数は 7 で，最も長い属性名（index）の文字数は 5 なので，これらを足した文字数は 12 になります。

```
<root>
  <data index="1"/>
  <element>
    <content>example</content>
  </element>
</root>
```

6.20.7 チューニング情報に関する注意事項

表 6-42 高速パース機能に関する注意事項（チューニング情報）

項番	注意事項
1	パース対象 XML 文書のシステム識別子のパス区切り文字には "/" を使用してください。
2	コンストラクタ引数に相対パスを指定して作成した File オブジェクトを，PreparsedObjectFactory クラスの newPreparsedObject メソッドに指定すると，チューニングサマリファイルおよび詳細ファイルに出力される事前解析用 XML 文書のファイル名のパス区切り文字が，次に示すように重複して出力される場合があります。 `Preparsed XML: C:¥¥a.xml`

6.20.8 解析性能に関する注意事項

表 6-43 高速パース機能に関する注意事項（解析性能）

項番	注意事項
1	外部エンティティ参照を含む XML 文書に高速パース機能を適用すると，通常の解析と比べ，解析性能が向上しない場合があります。外部エンティティ参照を含む XML 文書に高速パース機能を適用する場合は，事前に性能評価を行ってください。

6.21 JAXB に関する注意事項

JAXB に関する注意事項を次に示します。

6.21.1 共通の注意事項

共通の注意事項を次の表に示します。

表 6-44 JAXB に関する注意事項（共通）

項番	注意事項
1	JAXB 仕様書が規定するメソッドがスローする例外オブジェクトに getMessage メソッドを適用すると戻り値が null になり，詳細メッセージを取得できないことがあります。この場合，例外オブジェクトに getCause メソッドを適用して「ラップされた例外オブジェクト」を取得し，これに getMessage メソッドを適用することで，詳細なメッセージを取得できる場合があります。また，JAXBException 例外およびそのサブクラスの場合は，toString メソッドを適用することで，詳細なメッセージを取得できる場合があります。
2	JAXB 仕様書では，メソッド，スキーマコンパイラ，スキーマジェネレータの動作が明確に定義されていない個所があります。この場合，動作は実装に依存します。また，引数に null を指定した場合，Javadoc では NullpointerException 以外の例外が発生すると規定されていても，この実装では NullPointerException が発生することがあります。引数の事前チェックをするなどの手段によって，動作が規定されていない状態にならないようにしてください。 （例 1）Javadoc の例外欄に示されていない種別の例外が発生する（引数に null を指定した場合，NullPointerException が発生するなど）。 （例 2）スキーマコンパイラの動作が規定されていない状態で生成した Java ソースを用いてマーシャル，アンマーシャルを実行したとき，動作は規定されません。
3	ISO-10646-UCS-4 エンコーディングは使用できません。
4	バージョンが 1.1 の XML 文書は使用できません。
5	Unicode の補助文字は使用できません。
6	Java SE 6 は，標準では JAXB 2.2 をサポートしていません。そのため，日立製 JavaVM 以外で JAXB 2.2 の機能を使用する場合は，Java 推奨標準優先機構で <Cosminexus のインストール先 >/jaxp/lib/csmjaxb-api.jar を指定してください。 設定方法の詳細は，次に示す Oracle のサイトを参照してください。 http://download.oracle.com/javase/6/docs/technotes/guides/standards/

6.21.2 スキーマコンパイラに関する注意事項

スキーマコンパイラに関する注意事項を次の表に示します。

項番 1 ～項番 28 の注意事項に該当するスキーマは，スキーマコンパイラで処理しないでください。項番 29 ～項番 44 については，注意事項に留意して使用してください。

表 6-45 JAXB に関する注意事項（スキーマコンパイラ）

項番	注意事項
1	**[条件]** 繰り返し指定のある内容モデルに次のすべての条件が該当する場合です。 • nillable="true" を指定した要素定義が存在する。 • 内容モデルが mixed ではなく，ワイルドカードを含まない。 • 内容モデルの要素定義が xs:list や xs:IDREF の xml データ型または xs:list や xs:IDREF から派生したデータ型を持っていない。 • 内容モデルのすべての要素定義に対し，型が同じ Java データ型にバインドされる二つの異なった要素定義が存在しない。 （例） <xs:complexType> <xs:sequence maxOccurs="unbounded"> <xs:element name="A" type="xs:string" /> <xs:element name="B" type="xs:int" nillable="true" /> <xs:element name="C" type="xs:float" nillable="true"/> <xs:element name="D" type="xs:double" /> </xs:sequence> </xs:complexType> **[Cosminexus XML Processor の動作]** コレクションの最適化を適用します。@XmlElementRefs/@XmlElementRef ではなく @XmlElements/@XmlElement でアノテートされたコレクションを出力します。この出力 java ソースを使用してアンマーシャル・マーシャルを実行した場合，不正な動作をすることがあります。
2	**[条件]** 次のすべての条件に該当する場合です。 1. xs:schema 要素の targetNamespace 属性に「www.com1.hitachi」など Windows の予約デバイス名を含む。 2. Windows 上でスキーマコンパイラを実行する。 **[Cosminexus XML Processor の動作]** エラーが発生します。 （例） [ERROR] .¥hitachi¥com1¥ObjectFactory.java（指定されたパスが見つかりません。[errno=3, syscall=CreateFileW]）

項番	注意事項
3	**[条件]** xs:schema 要素の targetNamespace 属性に指定した URI を ":" または "/" で分割した文字列の先頭文字が，次のすべての条件に該当する場合です。 • Character.isJavaIdentifierStart の戻り値が false となる。 • Character.isJavaIdentifierPart の戻り値が false となる。 **[Cosminexus XML Processor の動作]** 不正なパッケージ名であるというエラーが発生します。 (例) [ERROR] The package name 'publicid.__-.w3c.dtd_html_4_01.en' used for this schema is not a valid package name.
4	**[条件]** xs:schema 要素の targetNamespace 属性および xmlns 属性に指定する URI が，次の文字だけで構成されている場合です。 • ピリオド (.) • スラッシュ (/) • 文字列 "www" **[Cosminexus XML Processor の動作]** java.lang.ArrayIndexOutOfBoundsException などの，規定されていない例外が発生する場合があります。
5	**[条件]** 次のどちらかの条件に該当する場合です。 • 繰り返し指定のない内容モデルに xs:any 要素が複数存在する。 • 複合型の拡張で，基底型と拡張型の両方が要素内容として xs:any 型を含んでいる。 **[Cosminexus XML Processor の動作]** 一つの一般的内容プロパティではなく，複数のプロパティの生成を試みて，エラーが発生します。一般的内容プロパティの詳細については，JSR 222 The Java Architecture for XML Binding 2.1 の 6.12.3 を参照してください。 (例) [ERROR] Property "Any" is already defined. Use <jaxb:property> to resolve this conflict. なお，jaxb:property カスタムバインディングを指定し，複数の @XmlAnyElement 注釈されたプロパティを出力した java ソースを使用してアンマーシャル・マーシャルを実行した場合，不正な動作をすることがあります。
6	**[条件]** 次のどちらかの条件に該当する場合です。 • 型を派生したとき，派生した型の complexType 要素の mixed 属性の属性値が "true" である。 • 型を派生したとき，基底型の mixed 属性の属性値が "true" である。 **[Cosminexus XML Processor の動作]** 派生した型の子要素を取得するメソッドが生成されません。

項番	注意事項
7	**[条件]** 次のすべての条件に該当する場合です。 1. xs:simpleType 要素に対して，jaxb:typesafeEnumMember を子要素として持つ jaxb:typesafeEnumClass カスタムバインディングが存在する。また，jaxb:typesafeEnumMember には name 属性が指定されている。 2. 1. の xs:simpleType の下位要素である xs:enumuration 要素に対して jaxb:typesafeEnumMember カスタムバインディングが指定されていて，name 属性が指定されている。 3. 1. と 2. の name 属性の属性値が異なる。 **[Cosminexus XML Processor の動作]** 条件 1. の name 属性を基に，Java クラスの列挙定数名を生成します。
8	**[条件]** 次のどれかに示す XML スキーマ要素に，カスタムバインディングを指定する場合です。 • xs:attributeGroup 要素に jaxb:typesafeEnumClass を指定する。 • xs:complexContent 要素の子要素である xs:extension 要素に jaxb:typesafeEnumClass を指定する。 • xs:simpleType 要素の子要素である xs:restriction 要素，またはその子要素の xs:maxExclusive/xs:maxInclusive/xs:minExclusive/xs:maxLength/xs:pattern ファセットに jaxb:typesafeEnumClass/jaxb:typesafeEnumMember/jaxb:property を指定する。 • xs:restriction 要素の子要素の xs:totalDigits/xs:whiteSpace ファセットに jaxb:schemaBindings を指定する。 **[Cosminexus XML Processor の動作]** このようなスキーマ文書を用いた場合の JAXB 実行時の動作は規定されていません。
9	**[条件]** 次のすべての条件に該当する場合です。 • map 属性が "false" であるような jaxb:schemaBindings カスタムバインディングが存在する。 • jaxb:class または jaxb:typesafeEnumClass カスタムバインディングが存在する。 **[Cosminexus XML Processor の動作]** カスタムバインディングが不正であるというエラーが出力されます。 （例） [ERROR] compiler was unable to honor this class customization. It is attached to a wrong place, or its inconsistent with other bindings.
10	**[条件]** 匿名複合型で型定義されている局所要素定義（＝ xs:complexType を子要素として持つ xs:element）に対し jaxb:class カスタムバインディングを指定する場合です。 **[Cosminexus XML Processor の動作]** エラーが出力されます。 （例） Exception in thread "main" java.lang.IllegalArgumentException: Illegal class inheritance loop. Outer class < 要素名 > may not subclass from inner class: < 要素名 >

項番	注意事項
11	**[条件]** 複合型定義の xs:restriction の中で指定された xs:choice, xs:sequence または xs:all に対し，jaxb:property カスタムバインディングを指定する場合です。 **[Cosminexus XML Processor の動作]** エラーが出力されます。 (例) [ERROR] compiler was unable to honor this property customization. It is attached to a wrong place, or its inconsistent with other bindings.
12	**[条件]** 次のすべての条件に該当する場合です。 1. xs:complexType 要素に jaxb:typesafeEnumClass カスタムバインディングを指定する。 2. 1. のカスタムバインディング指定の map 属性に "true" または "1" または "0" を指定する。 **[Cosminexus XML Processor の動作]** KECX06051-E cvc-enumeration-valid: Value 'true' is not facet-valid with respect to enumeration '[false]'. It must be a value from the enumeration. が発生します。 map 属性に指定した "true" は正しい値ですが，xs:complexType 要素に jaxb:typesafeEnumClass カスタムバインディングを指定すること自体が不正です。
13	**[条件]** 次のどちらかの条件に該当する場合です。 • jaxb:property の collectionType 属性に次のどれかを指定する。 ・空文字列 ・java.util.List を実装していないクラス ・java.util.List を実装しているが，完全修飾でないクラス • jaxb:globalBindings の collectionType 属性に次のどちらかを指定する。 ・java.util.List を実装していないクラス ・java.util.List を実装しているが，完全修飾でないクラス **[Cosminexus XML Processor の動作]** エラーは発生しないで，java クラスが生成されます。
14	**[条件]** jaxb:globalBindings の collectionType 属性に空文字列を指定する場合です。 **[Cosminexus XML Processor の動作]** java.lang.StringIndexOutOfBoundsException 例外をスローします。

項番	注意事項
15	**[条件]** 次のどれかの条件に該当する場合です。 • jaxb:class カスタムバインディングで次の両方の属性を指定する。 ・ref 属性 ・排他である name 属性 • jaxb:class カスタムバインディングで次の両方の属性を指定する。 ・ref 属性 ・排他である implClass 属性 • jaxb:typesafeEnumClass カスタムバインディングで次の両方の属性を指定する。 ・ref 属性 ・排他である name 属性 • jaxb:typesafeEnumClass カスタムバインディングで次の両方の属性を指定する。 ・ref 属性 ・排他である map 属性 **[Cosminexus XML Processor の動作]** エラーは発生しません。このような場合の JAXB 実行時の動作は規定されていません。
16	**[条件]** 次のすべての条件に該当する場合です。 • xs:element 大域要素定義，または xs:element 局所要素定義に，jaxb:class カスタムバインディングを指定する。 • jaxb:class カスタムバインディングで implClass 属性を指定する。 **[Cosminexus XML Processor の動作]** エラーは発生しません。このような場合の JAXB 実行時の動作は規定されていません。
17	**[条件]** 次のすべての条件に該当する場合です。 • 列挙ファセットの xs:simpleType 定義に，jaxb:typesafeEnumClass カスタムバインディングを指定する。 • jaxb:typesafeEnumClass の map 属性に "true"，"1"，または "0" を指定する。 **[Cosminexus XML Processor の動作]** エラーが出力されます。map 属性を指定していない場合，デフォルトの動作は，"true" または "1" を指定したときの動作となります。
18	**[条件]** 次のすべての条件に該当する場合です。 • 局所または大域属性宣言に対して，jaxb:property カスタムバインディングを指定する。 • generateElementProperty 属性に "true"，"false"，"1"，または "0" を指定する。 **[Cosminexus XML Processor の動作]** エラーは出力されないで，プロパティを含む java ソースが出力されます。
19	**[条件]** 次のすべての条件に該当する場合です。 • 同じ型で name 属性が異なる複数の要素宣言に対して，jaxb:property カスタムバインディングを指定する。 • generateElementProperty 属性に "false" を指定する。 **[Cosminexus XML Processor の動作]** エラーは出力されないで，JAXBElement を含む java ソースが出力されます。

項番	注意事項
20	**[条件]** 次のどちらかの条件に該当する場合です。 • jaxb:baseType 要素の name 属性に，デフォルト基底型と同じ継承階層にないクラスを指定する。 • jaxb:baseType 要素の name 属性に，完全修飾された Java クラス名ではないクラスを指定する。 **[Cosminexus XML Processor の動作]** エラーは出力されないで，java ソースが出力されます。
21	**[条件]** jaxb:property カスタムバインディングの attachmentRef 属性を使用する場合です。 **[Cosminexus XML Processor の動作]** スキーマコンパイル中に次のエラーが出力されます。 [ERROR] KECX06031-E cvc-complex-type.3.2.2: Attribute 'attachmentRef' is not allowed to appear in element 'jaxb:property'.
22	**[条件]** jaxb:globalBindings カスタムバインディングの underscoreBinding 属性に，"asWordSeparator" または "asCharInWord" 以外の文字列を指定する場合です。 **[Cosminexus XML Processor の動作]** NullPointerException 例外をスローします。
23	**[条件]** jaxb:globalBindings カスタムバインディングの typesafeEnumMemberName 属性に，"skipGeneration" を指定する場合です。 **[Cosminexus XML Processor の動作]** 次のエラーが出力されます。 KECX06051-E cvc-enumeration-valid: Value 'skipGeneration' is not facet-valid with respect to enumeration '[generateError, generateName]'. It must be a value from the enumeration. **[回避策]** typesafeEnumMemberName 属性を指定しない場合，デフォルトの動作は "skipGeneration" の動作となります。
24	**[条件]** 次の jaxb:globalBindings カスタムバインディングのカスタム化指定を，複数指定した場合です。 • name 属性の値が異なる xmlType 属性で，その xmlType 属性の値が同じ jaxb:javaType • uid 属性の値が異なる jaxb:serializable **[Cosminexus XML Processor の動作]** このような場合の JAXB 実行時の動作は規定されていません。

項番	注意事項
25	**[条件]** 次のすべての条件に該当する場合です。 • jaxb:globalBindings カスタムバインディングの optionalProperty 属性に，"primitive" を指定する。 • スキーマに，次のどれかのローカル要素定義がある。 ・minOccurs="0" が指定されているプリミティブ型の要素 ・xs:choice の中で定義されたプリミティブ型の要素 **[Cosminexus XML Processor の動作]** primitive 型ではなく，ラッパークラス型で出力されます。
26	**[条件]** 次のすべての条件に該当する場合です。 • jaxb:globalBindings カスタムバインディングの generateElementProperty 属性に，"false" を指定する。 • スキーマに同一の型で異なる名前の要素が複数存在するなど，JAXBElement 型のプロパティとなる指定が存在する。 **[Cosminexus XML Processor の動作]** エラーは発生しません。このような場合の JAXB 実行時の動作は規定されていません。
27	**[条件]** バイナリ型の単純内容を持った xs:complexType に，jaxb:inlineBinaryData を指定する場合です。 **[Cosminexus XML Processor の動作]** カスタムバインディングが不正であるというエラーが出力されます。 （例） [ERROR] compiler was unable to honor this inlineBinaryData customization. It is attached to a wrong place, or its inconsistent with other bindings.
28	**[条件]** 次のどれかの条件に該当する場合です。 • jaxb:javaType の name 属性に，存在しない java データ型を指定する。 • jaxb:javaType の parseMethod 属性，または printMethod 属性を指定し，name 属性にプリミティブ型を指定する。 • jaxb:javaType の name 属性に，String 型の一つの引数を持つコンストラクタが定義されていない java データ型を指定する。 **[Cosminexus XML Processor の動作]** エラーは発生しません。
29	標準仕様では，カスタムバインディング jaxb:javadoc を用いることで，生成される Java クラスのドキュメンテーションコメントの内容をカスタマイズする方法が規定されています。jaxb:javadoc を指定しない場合でも，デフォルトのドキュメンテーションコメントが生成されますが，その内容は標準仕様では規定されていません。

項番	注意事項
30	**[条件]** xs:totalDigits，xs:maxExclusive，xs:maxInclusive，xs:minExclusive，xs:minInclusive ファセットのどれかで指定された範囲が，int または long の範囲に収まるような単純型定義が存在する場合です。 **[Cosminexus XML Processor の動作]** xs:totalDigits ファセットを用いた場合は，常に java.math.BigInteger が生成されます。 xs:maxExclusive，xs:maxInclusive，xs:minExclusive，xs:minInclusive ファセットを用いた場合は，JSR 222 The Java Architecture for XML Binding 2.1 の 6.2.2 に示されているアルゴリズムに従って Java 型が生成されます。
31	**[条件]** 次のどれかを含むモデルグループ定義が存在する場合です。 • 匿名の複合型定義 • enum type にマップされる xs:enumeration ファセット定義 • 要素クラスにマップされる内容モデル **[Cosminexus XML Processor の動作]** 作成されるクラス名に，モデルグループ名は付けられません。

32	**[条件]**
	xs:element 局所要素定義に jaxb:class カスタムバインディングを指定する場合です。
	
	[Cosminexus XML Processor の動作]
	JAXBElement<T> を継承する，作成されたクラスを返す要素ファクトリメソッドを作成します。
	(例)
	`public class ObjectFactory {`
	`  public Base.Bar createBaseBar(String value) {`
	`  return new Base.Bar(value);`
	`  }`
	`  ・・・・`
	`}`
	注
	Base.Bar は，JAXBElement<T> を継承したクラスです。

33	**[条件]** 次のすべての条件に該当する場合です。 1. xs:element 大域要素定義が存在する。 2. 1. の abstract 属性の値が "true" である。 **[Cosminexus XML Processor の動作]** 要素定義に対応する要素ファクトリメソッドは生成されますが，そのファクトリメソッドを使用してインスタンスを生成できません。
34	**[条件]** スキーマ文書が次のどちらかに該当する場合です。 • 内容モデルに，要素名が同じで名前空間が異なる要素定義が指定されている。 • 内容モデルに，要素名が「class または Class」と「clazz または Clazz」である要素定義が，両方とも含まれる。 **[Cosminexus XML Processor の動作]** 名前衝突エラーにならないで，@XmlElementRef/@XmlElementRefs でアノテートされた一般的内容リストとして処理されます。内容リストの詳細については，JSR 222 The Java Architecture for XML Binding 2.1 の 6.12.3 を参照してください。

項番	注意事項
35	**[条件]** xs:any の processContents 属性に "lax" または "skip" を指定する場合です。 **[Cosminexus XML Processor の動作]** バインドされるプロパティのデータ型は，processContents 属性が "lax" の場合，java.lang.Object 型，"skip" の場合 org.w3c.dom.Element 型となります。
36	**[条件]** xs:element 要素の fixed 属性を指定する場合です。 **[Cosminexus XML Processor の動作]** 指定した fixed 属性の値は，生成される Java ソースの @XmlElement および @XmlElementDecl の defaultValue 要素に反映されません。 **[回避策]** fixed 属性の値をデフォルト値として反映させたい場合は，fixed 属性ではなく default 属性を指定してください。
37	**[条件]** xs:enumeration 要素の value 属性に，生成される java 定数識別子が衝突するような文字列を指定する場合です。 （例） 「name-with-dashes」と「name_with_dashes」 **[Cosminexus XML Processor の動作]** 列挙定義を出力しません。エラーは発生しません。
38	**[条件]** xs:schema 要素の targetNamespace 属性に指定した URI が，次のすべての条件に該当する場合です。 • トップレベルドメイン（com，gov，net，org，edu など）を含まない。 • 国別コードを含まない。 **[Cosminexus XML Processor の動作]** 生成される java ソースのパッケージ名はドメイン名が逆順にされ，"www." は削除されます。
39	**[条件]** 要素名や属性名に次のどれかを指定する場合です。 • java キーワード • java リテラル（"true"，"false"，"null"） • java.lang.Object クラスで定義されているメソッド名 • javax.xml.bind クラスで定義されているメソッド名 **[Cosminexus XML Processor の動作]** 名前衝突エラーにならないで，フィールドやメソッド名が生成されます。java キーワードや java リテラルの場合 "_" を付けたフィールド名が作成されます。
40	**[条件]** 再定義された複合型定義が存在する場合です。 **[Cosminexus XML Processor の動作]** 再定義された複合型定義は，"_" が付加されたクラス名でなく，接頭辞 Original が付加されたクラス名が生成されます。

項番	注意事項
41	**[条件]** 次のすべての条件が該当する場合です。 • モデルグループ定義が存在する。 • jaxb:schemaBindings カスタムバインディングで，jaxb:modelGroupName の suffix 属性，または prefix 属性を指定する。 **[Cosminexus XML Processor の動作]** モデルグループ定義はクラスにマッピングされないため，jaxb:modelGroupName の指定は生成されるソースに反映されません。
42	**[条件]** 次のすべての条件が該当する場合です。 1. jaxb:globalBindings カスタムバインディングの underscoreBinding 属性に，"asCharInWord" を指定する。 2. xs:element の name 属性，または fixed 属性を指定した xs:attribute の name 属性に，アンダーラインが含まれる。 3. 2. が xs:attribute の name 属性の場合に，jaxb:globalBindings の fixedAttributeAsConstantProperty 属性に "true" を指定している。 **[Cosminexus XML Processor の動作]** 仕様書と異なる java 識別子が生成されます。 (例) • xs:element の name 属性の場合 child_element ＝＞ getChild_Element（正しくは getChild_element） • xs:attribute の name 属性の場合 child_attribute ＝＞ CHILD___ATTRIBUTE（正しくは CHILD_ATTRIBUTE）
43	**[条件]** 次のすべての条件が該当する場合です。 • jaxb:globalBindings カスタムバインディングの enableJavaNamingConventions 属性に，"true" を指定または属性を指定しない。 • <jaxb:schemaBindings> の子要素である jaxb:package の name 属性に，java 識別子として不正な値を指定する。 **[Cosminexus XML Processor の動作]** 適切な java 識別子への変換が実施されません。
44	**[条件]** jaxb:globalBindings カスタムバインディングで，uid 属性のない jaxb:serializable を指定する場合です。 **[Cosminexus XML Processor の動作]** エラーは発生しません。このような場合の JAXB 実行時の動作は規定されていません。

6.21.3 スキーマジェネレータに関する注意事項

スキーマジェネレータに関する注意事項を次の表に示します。

項番 1 ～項番 32，56，57，59 の注意事項に該当する Java ソースは，スキーマジェネレータで処理しないでください。項番 33 ～項番 55，項番 58，60，61 については，注意事項に留意して使用してください。

表 6-46 JAXB に関する注意事項（スキーマジェネレータ）

項番	注意事項
1	JAXB マッピングアノテーションは，JAXB 仕様で指定範囲が制約されています。次の例のように，JAXB 仕様の指定範囲外の個所に JAXB マッピングアノテーションを指定した場合，エラーにならないことがあるため注意してください。指定した場合の JAXB 実行時の動作は規定されていません。 （例） • パッケージに指定できない JAXB マッピングアノテーションをパッケージに指定した。 • トップレベルでないクラスに @XmlRootElement，および @XmlType を指定した。 • フィールドに @XmlEnumValue を指定した。 • 列挙型に @XmlRegistry を指定した。 • 同時に指定できると記載されていない JAXB マッピングアノテーションを同時に指定した。
2	**[条件]** 次の組み合わせの JAXB マッピングアノテーションのどれかを同時に指定する場合です。 • @XmlAttribute と @XmlValue • @XmlAttribute と @XmlMimeType • @XmlElement と @XmlValue **[Cosminexus XML Processor の動作]** エラーを出力します。スキーマは生成しません。 （例） @XmlAttribute と @XmlValue の場合 mypackage.ChildType#name has mutually exclusive annotations @javax.xml.bind.annotation.XmlAttribute and @javax.xml.bind.annotation.XmlValue @XmlAttribute と @XmlMimeType の場合 javax.xml.bind.annotation.XmlMimeType annotation cannot be placed here
3	**[条件]** transient フィールドに次のどれかを指定する場合です。 • @XmlAttribute • @XmlElement • @XmlElements • @XmlElementWrapper • @XmlList • @XmlMimeType • @XmlValue • @XmlAnyElement **[Cosminexus XML Processor の動作]** エラーが発生します。 （例） Transient field "child" cannot have any JAXB annotations.

項番	注意事項
4	**[条件]** static フィールドに次のどれかを指定する場合です。 • @XmlElement • @XmlElements • @XmlElementWrapper • @XmlList • @XmlMimeType • @XmlValue • @XmlAnyElement • @XmlID • @XmlIDREF **[Cosminexus XML Processor の動作]** エラーは発生しません。
5	**[条件]** @XmlElements に @XmlElement を一つも指定しない場合です。 **[Cosminexus XML Processor の動作]** 不正なスキーマが出力されます。
6	**[条件]** @XmlJavaTypeAdapter を一つも含まない @XmlJavaTypeAdapters を指定する場合です。 **[Cosminexus XML Processor の動作]** @XmlJavaTypeAdapters が指定されなかったかのようなスキーマを出力します。
7	**[条件]** value 要素が空の配列である @XmlSeeAlso を指定する場合です。 **[Cosminexus XML Processor の動作]** @XmlSeeAlso が指定されなかったかのようなスキーマを出力します。
8	**[条件]** @XmlElementWrapper，@XmlAttribute，@XmlElement，@XmlRootElement， @XmlElementDecl のどれかの name 要素に，空文字列を指定する場合です。 **[Cosminexus XML Processor の動作]** • @XmlElementWrapper，または @XmlElement の場合，name 要素が指定されていないときと同様に，フィールド名やプロパティ名から要素名を生成します。 • @XmlAttribute の場合，<xs:attribute name="" .../> という不正な属性定義を生成します。 • @XmlRootElement，または @XmlElementDecl の場合，<xs:element name="" .../> という不正な要素定義を生成します。

項番	注意事項
9	**[条件]** 次のすべての条件に該当する場合です。 1. パラメタ化されたコレクション型のフィールドまたは JavaBean プロパティに @XmlElement を付ける。または，配列型かパラメタ化されたリスト型の JavaBean プロパティに @XmlElements を付ける。 2. 1. の @XmlElement の type 要素を明示的に指定する。 **[Cosminexus XML Processor の動作]** @XmlElement の type 要素に関する制約は適用されません。このような Java ソースを用いた場合の JAXB 実行時の動作は規定されていません。制約の詳細については，JSR 222 The Java Architecture for XML Binding 2.1 の 8.9.1.2 および Javadoc を参照してください。
10	**[条件]** 次のすべての条件に該当する場合です。 1. パラメタ化されたコレクション型のフィールドまたは JavaBean プロパティに @XmlElements を指定する。 2. 1. の @XmlElements の要素に，複数の @XmlElement を指定する。 **[Cosminexus XML Processor の動作]** エラーは発生しません。しかし，このスキーマおよび java ソースを使用してアンマーシャルを実行しても正しく処理されません。 **[回避策]** パラメタ化されたコレクション型のプロパティまたは JavaBean フィールドに複数の要素定義を指定したい場合は，@XmlElementRefs および @XmlElementRef を使用し JAXBElement のコレクション型として処理してください。
11	**[条件]** @XmlElement の defaultValue 要素に，@XmlElement を指定したフィールドまたは JavaBean プロパティの型に対して不適切な値を指定する場合です。 **[Cosminexus XML Processor の動作]** default 属性に不適切な値が指定された要素定義のスキーマ文書を生成し，エラーは発生しません。しかし，このスキーマを設定した Unmarshaller でアンマーシャルを実行すると，スキーマ不正のエラー（KECX06161-E）になります。生成されるスキーマは不正なものとなります。
12	**[条件]** 次のすべての条件に該当する場合です。 • @XmlElement の nillable 要素に true を指定する。 • @XmlElement の namespace 要素に，フィールドまたは JavaBean プロパティが含まれるクラスの namespace 要素と異なる値を指定する。 **[Cosminexus XML Processor の動作]** 不正なスキーマ文書（nillable 属性のある要素参照）が出力されます。このスキーマを設定した Unmarshaller でアンマーシャルを実行すると，スキーマ不正エラー（KECX06231-E）となります。
13	**[条件]** 次のすべての条件に該当する場合です。 1. @XmlSchema の namespace 要素に空文字列以外の値を指定する。 2. @XmlSchema の xmlns 要素に @XmlNs を指定する。 3. 2. に指定した @XmlNs の prefix 要素の値が "tns" かつ namespace 要素の値が 1. の値と異なる。 **[Cosminexus XML Processor の動作]** IllegalArgumentException 例外が発生します。

項番	注意事項
14	**[条件]** 次のすべての条件に該当する場合です。 1. プリミティブ型のフィールドまたは JavaBean プロパティに対して，@XmlAttribute を付ける。 2. 1. で指定した @XmlAttribute に required 要素を指定しないか，"false" を指定する。 **[Cosminexus XML Processor の動作]** 生成される属性定義の use 属性値は，常に "required" です。
15	**[条件]** 次のすべての条件に該当する場合です。 1. 単純型リストにマップされるクラスが存在する。 2. 1. のクラスを型パラメタとして持つ List インスタンスに，@XmlList を指定する。 **[Cosminexus XML Processor の動作]** エラーは発生しないで，@XmlList を反映したスキーマが生成されます。

16	**[条件]**
	次のすべての条件に該当する場合です。
	1. @XmlRootElement，および name 要素が空文字列の @XmlType をクラスに付ける。
	2. 1. のクラスのフィールドまたは JavaBean プロパティの型が，1. のクラス自身である。
	
	[Cosminexus XML Processor の動作]
	無限再帰を示すエラーが発生します。
	(例)
	エラー :Anonymous types form an infinite cycle: ClassInfo(mypackage.ItemType) -> ClassInfo(mypackage.ItemType)
	
	[回避策]
	2. のフィールドまたは JavaBean の名前を，要素クラス名と同じ（ただし，XML 名→ Java 名の変換規則による変換後の名前）にしてください。
	(例)
	`@XmlRootElement`
	`@XmlType(name="")`
	`public class ItemType {  public ItemType itemType;}`

17	**[条件]** 次のすべての条件に該当する場合です。 1. 複合型にマッピングされるクラスに abstract 修飾子を指定する。 2. 1. のクラスのフィールドまたは JavaBean プロパティに，@XmlValue を付ける。 3. 1. のクラスのフィールドまたは JavaBean プロパティに，@XmlAttribute を付ける。 **[Cosminexus XML Processor の動作]** 生成される xs:complexType 要素には，abstract 属性が付けられません。
18	**[条件]** 列挙型に @XmlEnum を指定します。その value 要素に，単純型にマッピングされないクラス（例：java.lang.Object）を指定する場合です。 **[Cosminexus XML Processor の動作]** エラーは出力されません。このような Java ソースを用いた場合の JAXB 実行時の動作は規定されていません。

項番	注意事項
19	**[条件]** 列挙定数に @XmlEnumValue を指定します。その value 要素に，@XmlEnum.value() 型に対して不当な字句表現を指定する場合です。 **[Cosminexus XML Processor の動作]** エラーは出力されません。このような Java ソースを用いた場合の JAXB 実行時の動作は規定されていません。
20	**[条件]** 次のすべての条件に該当する場合です。 • @XmlElement の defaultValue 要素に，"¥u0000" 以外の値を指定する。 • @XmlElement の namespace 要素に，フィールドまたは JavaBean プロパティが含まれるクラスの namespace 要素と異なる値を指定する。 **[Cosminexus XML Processor の動作]** 不正なスキーマ文書（default 属性のある要素参照）が出力されます。このスキーマを設定した Unmarshaller でアンマーシャルを実行すると，スキーマ不正エラー（KECX06231-E）となります。
21	**[条件]** java.util.HashMap 型のフィールドまたは JavaBean プロパティに，@XmlElement または @XmlElements を指定する場合です。 **[Cosminexus XML Processor の動作]** 不正なスキーマ文書（要素定義の maxOccurs 属性が "unbounded" ではなく "1" となるもの）が出力される。 **[回避策]** HashMap 型のプロパティまたはフィールドに対し @XmlElement を指定する場合に限り，JSR 222 The Java Architecture for XML Binding 2.1 の 8.2.5 が示すように @XmlJavaTypeAdapter を追加指定することで回避できます。

22	**[条件]**
	次のすべての条件に該当する場合です。
	1. @XmlRootElement，および name 要素が空文字列の @XmlType をクラスに付ける。
	2. フィールドまたは JavaBean プロパティの型が 1. のクラスであるようなクラスが存在する。
	3. 2. のフィールドまたは JavaBean プロパティがマッピングする要素名と，1. がマッピングする要素名が異なる。
	(例)
	`public class Root {`
	`  // 条件 2. と 3. のフィールド名が childType であれば，要素参照が生成される。`
	`  protected ChildType child;`
	`}`
	
	`// 条件 1. のクラス`
	`@XmlRootElement`
	`@XmlType(name = "")`
	`public class ChildType {`
	`  public String fname;`
	`}`
	
	[Cosminexus XML Processor の動作]
	要素定義（name 属性）が生成されます。

項番	注意事項

23	**[条件]**
	次のすべての条件に該当する場合です。
	1. @XmlRootElement，および name 要素が空文字列の @XmlType をクラスに付ける。
	2. フィールドまたは JavaBean プロパティの型が 1. のクラスであるようなクラスが存在する。
	3. 2. のフィールドまたは JavaBean プロパティがマッピングする要素名と 1. がマッピングする要素名が，一致する。
	4. 2. のフィールドまたは JavaBean プロパティに指定した @XmlElement の defaultvalue 要素に，"¥u0000" 以外を指定する。
	(例)
	`public class Root {`
	`  @XmlElement(defaultValue="default")  // 条件4.`
	`  protected ChildType childType;  // 条件2., 3.`
	`}`
	
	`// 条件1.のクラス`
	`@XmlRootElement`
	`@XmlType(name = "")`
	`public class ChildType {`
	`  public String fname;`
	`}`
	
	[Cosminexus XML Processor の動作]
	default 属性を持つ要素定義（name 属性）が生成されます。

24	**[条件]** JavaBean プロパティが，setter メソッドまたは getter メソッドの一方が存在しないような，不当なものである場合です。 **[Cosminexus XML Processor の動作]** setter メソッドおよび getter メソッドの両方が存在する正しい JavaBean であるかのように，スキーマが生成されます。エラーは発生しません。 このような Java ソースを用いた場合の JAXB 実行時の動作は規定されていません。

25	**[条件]**
	JAXB マッピングアノテーションの要素に，不当なスキーマ文書を生成するような指定が存在する場合です。
	(例)
	`@XmlRootElementpublic`
	`class ItemType {`
	`  @XmlElement(name="A")`
	`  public String item1;`
	`  @XmlElement(name="A")`
	`  public String item2;`
	`}`
	
	@XmlElement(name="A") では，異なるフィールドに同一の要素名（"A"）を指定しています。
	
	[Cosminexus XML Processor の動作]
	エラーチェックがされないで，不当なスキーマ文書が生成されることがあります。

項番	注意事項
26	**[条件]** JAXB マッピングアノテーションの namespace 要素に XML Schema 仕様固有の名前空間 URI を表す文字列 "http://www.w3.org/2001/XMLSchema" を指定する場合です。 **[Cosminexus XML Processor の動作]** この名前空間 URI を targetNamespace 属性に持つスキーマ文書は生成されません。なお，この名前空間 URI を指定する用途はなく，指定する意味はありません。
27	**[条件]** @XmlElements および @XmlMixed を同時に指定した場合です。 **[Cosminexus XML Processor の動作]** 次のエラーが出力されます。 java.lang.IndexOutOfBoundsException: Index: 1, Size: 1
28	**[条件]** 次のすべての条件に該当する場合です。 • @XmlElementRef を指定したフィールドやプロパティのデータ型が JAXBElement である。 • @XmlElementRef.name() と @XmlElementRef.namespace() が，@XmlRegistry で注釈されたクラス内の @XmlElementDecl 注釈を伴う要素ファクトリメソッドである，次の JAXB マッピングアノテーションとそれぞれ同じである。 ・@XmlElementDecl.name() ・@XmlElementDecl.namespace() • @XmlElementRef.namespace() に "##default" を指定する。 **[Cosminexus XML Processor の動作]** エラーとなりスキーマは生成されません。
29	**[条件]** type 要素のない @XmlSchemaType 注釈をパッケージに指定する場合です。 **[Cosminexus XML Processor の動作]** エラーは出力されません。このような Java ソースを用いた場合の JAXB 実行時の動作は規定されていません。
30	**[条件]** クラス中に複数の @XmlID で注釈している場合です。 **[Cosminexus XML Processor の動作]** エラーにはならないで，スキーマが生成されます。
31	**[条件]** 次のどれかの条件に該当する場合です。 • java.lang.Object 型のコレクション型のプロパティに，@XmlID 注釈の付いたプロパティが含まれない。 • java.lang.Object 型のプロパティに，@XmlID 注釈の付いたプロパティが含まれない。 • java.lang.Object 型のコレクション型のプロパティに，@XmlID 注釈の付いたフィールドが含まれない。 • java.lang.Object 型のプロパティに，@XmlID 注釈の付いたフィールドが含まれない。 **[Cosminexus XML Processor の動作]** エラーにはならないで，スキーマが生成されます。

項番	注意事項
32	**[条件]** 「あるクラスとそのスーパークラスで，@XmlAnyElement で注釈された JavaBean プロパティは一つだけである」という制限使用制約に反した，次のような記述をする場合です。 • クラスに @XmlAnyElement を指定したフィールドやプロパティが複数存在する。 • あるクラスとそのスーパークラスで，それぞれ @XmlAnyElement を指定したフィールドやプロパティが存在する。 **[Cosminexus XML Processor の動作]** エラーは出力されません。このような Java ソースを用いた場合の JAXB 実行時の動作は規定されていません。
33	**[条件]** @XmlAttachmentRef を，フィールドまたは JavaBean プロパティに付ける場合です。 **[Cosminexus XML Processor の動作]** 生成される要素定義の minOccurs 属性値は 0 になります。
34	**[条件]** 次の java 型をスキーマ文書にマッピングする場合です。 char, Boolean, Character, Byte, Short, Integer, Long, Float, Double, byte[] **[Cosminexus XML Processor の動作]** それぞれ，xs:unsignedShort，xs:boolean，xs:unsignedShort，xs:byte，xs:short，xs:int，xs:long，xs:float，xs:double，xs:base64Binary 型にマッピングされます。
35	**[条件]** package 文に type 要素を持たない @XmlJavaTypeAdapter を指定する場合です。 **[Cosminexus XML Processor の動作]** エラーは発生しないで，@XmlJavaTypeAdapter 指定が有効なスキーマが生成されます。
36	**[条件]** static final フィールド，または public static フィールドに @XmlAttribute を付ける場合です。 **[Cosminexus XML Processor の動作]** 生成される xs:attribute 要素に，fixed 属性は付けられません。
37	**[条件]** required 要素の値が false である @XmlElement を，プリミティブ型複数次元配列のフィールドまたは JavaBean プロパティに指定する場合です。 **[Cosminexus XML Processor の動作]** minOccurs 属性の値は 0 になります。
38	**[条件]** @XmlList，@XmlJavaTypeAdapter，@XmlAttachmentRef，@XmlMimeType のどれかを，メソッドのパラメタに付ける場合です。 **[Cosminexus XML Processor の動作]** これらの JAXB マッピングアノテーションの指定は無視されます。
39	**[条件]** パッケージに location 要素を持たない @XmlSchema を指定する場合です。 **[Cosminexus XML Processor の動作]** location 要素のデフォルト値は "##generate" と解釈されます。

項番	注意事項
40	**[条件]** @XmlType を列挙型に付ける場合です。 **[Cosminexus XML Processor の動作]** 生成される xs:simpleType に，final 属性は付けられません。
41	**[条件]** 次のすべての条件に該当する場合です。 1. @XmlSchema の namespace 要素に名前空間 URI を指定する。 2. @XmlSchema の xmlns 要素に @XmlNs を指定し，その namespaceURI 要素に名前空間 URI を指定する。 3. 1. と 2. の名前空間 URI が同じ文字列である。 **[Cosminexus XML Processor の動作]** 条件 1. と条件 2. の名前空間宣言が別々に生成されます。生成されたスキーマを使用しても問題はありません。

項番	注意事項
42	csmschemagen コマンド実行時，および JAXB API 呼び出し時には，apt ツールおよび javac ツールが実行されます。このため，エラー発生時にはエラー個所を示す情報に加えて，これらのツールのスタックトレースが出力されることがあります。 (例) ItemType.java:8: シンボルを見つけられません。 シンボル：クラス dummy@XmlEnum(dummy.class) 注釈の処理中に問題が検出されました。 詳細については，下記のスタックトレースを参照してください。 java.lang.RuntimeException: java.lang.reflect.InvocationTargetException at com.cosminexus.jaxb.tools.jxc.apt.InlineAnnotationReaderImpl.getClassValue(InlineAnnotationReaderImpl.java:150) at com.cosminexus.jaxb.tools.jxc.apt.InlineAnnotationReaderImpl.getClassValue(InlineAnnotationReaderImpl.java:64) at com.cosminexus.jaxb.xml.bind.v2.model.impl.EnumLeafInfoImpl.<init>(EnumLeafInfoImpl.java:110) at com.cosminexus.jaxb.xml.bind.v2.model.impl.ModelBuilder.createEnumLeafInfo(ModelBuilder.java:335) at com.cosminexus.jaxb.xml.bind.v2.model.impl.ModelBuilder.getClassInfo(ModelBuilder.java:224) at com.cosminexus.jaxb.xml.bind.v2.model.impl.ModelBuilder.getClassInfo(ModelBuilder.java:209) at com.cosminexus.jaxb.xml.bind.v2.model.impl.ModelBuilder.getTypeInfo(ModelBuilder.java:315) at com.cosminexus.jaxb.xml.bind.v2.model.impl.ModelBuilder.getTypeInfo(ModelBuilder.java:330) at com.cosminexus.jaxb.tools.xjc.api.impl.j2s.JavaCompilerImpl.bind(JavaCompilerImpl.java:90) at com.cosminexus.jaxb.tools.jxc.apt.SchemaGenerator$1.process(SchemaGenerator.java:115) at com.sun.mirror.apt.AnnotationProcessors$CompositeAnnotationProcessor.process(AnnotationProcessors.java:60) at com.sun.tools.apt.comp.Apt.main(Apt.java:454) (略) ... 27 moreItemType.java:8: シンボルを見つけられません。 シンボル：クラス dummy @XmlEnum(dummy.class) ^ エラー 1 個
43	**[条件]** 連続する大文字で始まるクラス名または列挙型名が存在する場合です。 **[Cosminexus XML Processor の動作]** クラス名または列挙型名の先頭の連続する大文字を小文字に変換したものが，XML 名にマッピングされます。

項番	注意事項
44	**[条件]** @XmlElementWrapper の nillable 要素に true を指定する場合です。 **[Cosminexus XML Processor の動作]** 生成するスキーマの要素定義には minOccurs="0" が指定されます。
45	**[条件]** 次のどれかの条件に該当する場合です。 • @XmlElement をフィールドまたは JavaBean プロパティに付ける。 • @XmlElementWrapper をフィールドまたは JavaBean プロパティに付ける。 • @XmlRootElement をクラスまたは列挙型に付ける。 **[Cosminexus XML Processor の動作]** xs:element 要素に final 属性と block 属性は付けられません。また，xs:schema 要素に finalDefault 属性と block 属性は付けられません。
46	**[条件]** @XmlValue をフィールドまたは JavaBean プロパティに付ける場合です。 **[Cosminexus XML Processor の動作]** 生成される xs:simpleType 要素または xs:complexType 要素に，final 属性は付けられません。
47	**[条件]** 絶対パスが "#" を含むディレクトリで csmschemagen コマンドを実行する場合です。 （例） cd C:¥jaxb¥abc#xyz¥test csmschemagen mypackage¥TEST.java **[Cosminexus XML Processor の動作]** 生成されるスキーマ文書の xs:import 要素の schemaLocation 属性の値が不正になります。 **[回避策]** 絶対パスが "#" を含まないディレクトリで csmschemagen コマンドを実行してください。
48	**[条件]** csmschemagen コマンドの -d オプションの引数に，カレントディレクトリ以外のディレクトリを指定した場合です。 **[Cosminexus XML Processor の動作]** コマンドが出力する情報メッセージに，スキーマ文書の出力先のパスではなく，カレントディレクトリのパスが表示されます。 （例） csmschemagen -d D:¥schema mypackage¥TEST.java 注 :Writing D:¥JAXB¥work¥schema1.xsd
49	**[条件]** csmschemagen コマンドの -d オプションの引数に出力先ディレクトリを指定しない場合です。 （例） csmschemagen -d mypackage¥TEST.java **[Cosminexus XML Processor の動作]** スキーマは出力されません。また，このときエラーが出力されないことがあります。

項番	注意事項
50	**[条件]** プロパティやフィールドで，@XmlSchemaType の name および @XmlElement の type に，異なる型にマッピングする値を指定する場合です。 **[Cosminexus XML Processor の動作]** @XmlSchemaType の name に指定されたデータ型の，要素定義のスキーマ文書が生成されます。
51	**[条件]** 次のすべての条件に該当する場合です。 • フィールドまたは JavaBean プロパティに，それらが含まれるクラスの名前空間とは異なる namesapce 要素の @XmlElement を指定する。 • フィールドまたは JavaBean プロパティに，@XmlID または @XmlIDREF を指定する。 **[Cosminexus XML Processor の動作]** 指定された @XmlID または @XmlIDREF は，生成される大域要素定義に反映されません。
52	**[条件]** @XmlElementDecl の substitutionHeadName 要素に，"" 以外を指定する場合です。 **[Cosminexus XML Processor の動作]** xs:element 要素の substitutionGroup 属性が出力されません。
53	**[条件]** @XmlElementDecl の defaultValue 要素に，"¥u0000" 以外を指定する場合です。 **[Cosminexus XML Processor の動作]** xs:element 要素の default 属性が出力されません。
54	**[条件]** 次のすべての条件に該当する場合です。 • @XmlAccessorOrder(XmlAccessOrder.ALPHABETICAL) をパッケージやクラスに指定する。 • 1 クラス内に，アルファベット順でない @XmlAttribute 注釈された複数のフィールドやプロパティがある。 **[Cosminexus XML Processor の動作]** 属性がアルファベット順にソートされたスキーマが生成されます。
55	**[条件]** トップレベルでない内部クラスに，@XmlAccessorType，および @XmlAccessorOrder を指定する場合です。 **[Cosminexus XML Processor の動作]** エラーにはならないで，指定の注釈が有効となります。

項番	注意事項
56	**[条件]** 次のすべての条件に該当する場合です。 • 次に示す要素に JIS X 0208 (1990) to Unicode 漢字コード表に存在しない文字がある。 @XmlEnumValue の要素 @XmlElement/@XmlElementDecl の defaultValue 要素 JAXB マッピングアノテーションの name 要素 • AIX プラットフォームで LANG 環境変数の値が「Ja_JP」または「Ja_JP.IBM-943」である。 **[Cosminexus XML Processor の動作]** java.lang.StringIndexOutOfBoundsException 例外など，予期しない例外が発生することがあります。
57	**[条件]** システムのエンコーディングと異なるエンコーディングの Java ソースを指定した場合です。 **[Cosminexus XML Processor の動作]** エラーが発生したり，不正なスキーマが生成されたりします。
58	**[条件]** 次のすべての条件に該当する場合です。 1. 複数値のフィールドまたは JavaBean プロパティに @XmlElementRef を付加する。 2. 1. の @XmlElementRef の required 要素が指定されていない，または required 要素に true が指定されている。 **[Cosminexus XML Processor の動作]** minOccurs 属性の値は，0 になります。
59	**[条件]** 次のどちらかに該当する場合です。 • システムのエンコーディングと異なるエンコーディング（package-info.java）を指定した。 • 引数に指定した Java ソースと同じディレクトリに，引数に指定していない package-info.java が存在し，そのエンコーディングがシステムのエンコーディングと異なる。 **[Cosminexus XML Processor の動作]** 不正にエラーが何度も出力されたり，引数に指定していない package-info.java のエラーが出力されたりすることがあります。
60	**[条件]** 次のすべての条件に該当する場合です。 • -encoding オプションに不当なエンコーディングを指定する。 • -encoding オプション以外のオプションを指定しない。 **[Cosminexus XML Processor の動作]** エラーが出力されず，スキーマも生成されません。

項番	注意事項
61	**[条件]** 次のすべての条件に該当する場合です。 • -encoding オプションを指定しない。 • デフォルトエンコーディング以外の Java ソースを指定する。 **[Cosminexus XML Processor の動作]** csmschemagen コマンドが異常終了することがあります。 **[回避策]** -encoding オプションで Java ソースのエンコーディングを指定してください。

6.21.4 実行時の注意事項

実行時の注意事項を次の表に示します。

表 6-47 JAXB に関する注意事項（実行時）

項番	注意事項
1	マルチスレッドプログラミングをする場合，javax.xml.bind で始まる名前のパッケージで規定されたオブジェクトはスレッドセーフではありません（例外として，javax.xml.bind.JAXBContext クラスはスレッドセーフです）。したがって，複数のスレッドが同時にこれらのオブジェクトにアクセスしてはいけません。スレッド間の競合を避けるため，次の方法を使用してください。 • 各スレッドが排他的にこれらのオブジェクトにアクセスする。
2	アンマーシャル，マーシャル時には，製品の内部でスキーマジェネレータと類似の処理が実行される場合があります。したがって，スキーマジェネレータの注意事項も参照してください。
3	xs:ENTITY 型または xs:ENTITIES 型を使用したスキーマ文書を検証モードでアンマーシャルまたはマーシャルを実行しないでください。実行した場合，エンティティが宣言されていないというエラーが発生します。 （例） KECX06252-E UndeclaredEntity: Entity 'id1' is not declared.
4	**[条件]** 次のすべての条件に該当する場合です。 • Unmarshaller の unmarshal(org.w3c.dom.Node node) メソッドを実行する。 • node パラメタに org.w3c.dom.Document または org.w3c.dom.Element 以外の Node オブジェクトを指定する。 **[Cosminexus XML Processor の動作]** IllegalArgumentException 例外が発生します。 （例） java.lang.IllegalArgumentException: Unexpected node type: [#text: String]
5	**[条件]** 次のすべての条件に該当する場合です。 • 例外をスローする整列化イベントコールバックメソッド (beforeMarshal/afterMarshal) をクラスに定義する。 • 上記イベントコールバックメソッドが呼ばれるインスタンス文書を出力するようなマーシャルを実行する。 **[Cosminexus XML Processor の動作]** マーシャルプロセスは終了しますが，IllegalStateException 例外がスローされます。
6	**[条件]** 次のすべての条件に該当する場合です。 1. パッケージレベルで @XmlJavaTypeAdapter を指定する。 2. クラスまたは列挙型に対し @XmlJavaTypeAdapter を指定する。 3. インスタンス文書に 1. および 2. で指定した @XmlJavaTypeAdapter が有効となるデータ型定義がある。 4. 1. ～ 3. の条件のもとでアンマーシャルまたはマーシャルを実行する。 **[Cosminexus XML Processor の動作]** クラス，列挙型に対する @XmlJavaTypeAdapter よりも，そのクラスのパッケージレベルで指定された @XmlJavaTypeAdapter が優先されます。動作が標準仕様と異なるため，このような java ソースは処理対象としないでください。

項番	注意事項
7	**[条件]** handleEvent メソッドの中で Java コンテントツリーを変更した場合です。 **[Cosminexus XML Processor の動作]** MarshalException をスローします。
8	**[条件]** 次のすべての条件に該当する場合です。 1. マーシャルを実行する。 2. marshal メソッドの引数 jaxbElement が JAXBElement オブジェクトである。 3. 2. の JAXBElement コンストラクタの name パラメタに指定した QName オブジェクトの localPart，prefix のどちらかが NCName ではない。 **[Cosminexus XML Processor の動作]** marshal メソッドは正常終了しますが，整形式ではない XML 文書が生成されます。
9	**[条件]** 次のすべての条件に該当する場合です。 1. @XmlEnum の value 要素が javax.xml.namespace.QName.class である。 2. @XmlEnumValue の value 要素が QName を表すリテラル文字列である。 3. JAXBContext.newInstance() メソッドの引数に，1. および 2. を含む Java クラスを指定して実行する。 **[Cosminexus XML Processor の動作]** JAXBException 例外が発生します。このような java ソースは処理対象としないでください。
10	**[条件]** 次のすべての条件に該当する場合です。 • ValidationEventLocator インタフェースの getURL メソッドを実行する。 • アンマーシャル対象のオブジェクトの URL が，RFC2396 に指定できない文字を含んでいる（ディレクトリ名，ファイル名から URL が自動生成される場合も含む）。 **[Cosminexus XML Processor の動作]** getURL メソッドの戻り値は，RFC2396 で指定できない文字をそのまま含みます。
11	**[条件]** 次のすべての条件に該当する場合です。 1. 型がプリミティブ型である static フィールドに，@XmlAttribute を指定する。 2. 1. の Java ソースをマーシャルする。 **[Cosminexus XML Processor の動作]** マーシャル時に不当な例外が発生します。このような java ソースは処理対象としないでください。 (例) Exception in thread "main" java.lang.IncompatibleClassChangeError

項番	注意事項

12	**[条件]**
	Java 値クラスの setter メソッドの引数に null を指定する場合です。
	
	[Cosminexus XML Processor の動作]
	NullPointerException が発生します。
	
	[回避策]
	setter メソッド呼び出し部分のコードを，次のように変更してください。
	(変更前)
	items.setId(null);
	(変更後)
	items.setId(new String[] {});

13	**[条件]** 次のすべての条件に該当する場合です。 1. インスタンス文書の要素に xsi:type 属性の指定がある。 2. 1. に指定された型が JAXB マップ型でない。 3. 1.，2. でアンマーシャルを実行する。 **[Cosminexus XML Processor の動作]** アンマーシャルが正常終了します。 要素は，xsi:type による型の置換をしないでアンマーシャルされます。
14	検証モードのマーシャル実行でエラーが発生した場合，エラーになる前までの内容の不正な XML（非整形な XML）が出力される場合があります。
15	jaxb.formatted.output プロパティを指定してマーシャルを実行したとき，出力される XML 文書の改行コードは 0x0A になります。
16	マーシャルを実行したとき，出力される XML 文書の XML 宣言に standalone="yes" 擬似属性が出力されます。
17	**[条件]** Marshaller インタフェースの marshal(Object jaxbElement，Node node) メソッドの引数 node に Element，Document，DocumentFragment 以外の Node オブジェクトを指定する場合です。 **[Cosminexus XML Processor の動作]** DOMException を例外スローします。 (例) org.w3c.dom.DOMException: HIERARCHY_REQUEST_ERR: KECX01304-E An attempt was made to insert a node where it is not permitted.
18	**[条件]** 次のすべての条件に該当する場合です。 1. スキーマ文書で「デフォルト値が指定されている xs:list 型の要素」を宣言している。 2. 1. をスキーマコンパイラに適用し，生成された Java 値クラスを用いて XML 文書をアンマーシャルする。 3. 2. の Java 値クラスのインスタンスを get メソッドで取得する。 **[Cosminexus XML Processor の動作]** 取得したインスタンスには，スキーマ文書で定義したデフォルト値が含まれません。

項番	注意事項
19	**[条件]** 次のすべての条件に該当する場合です。 1. @XmlAttachmentRef をフィールド，JavaBean プロパティ，パラメタのどれかに付ける。 2. 1. の Java ソースをスキーマジェネレータに入力する。 3. 2. で生成したスキーマ文書で妥当性検証を実行する。 **[Cosminexus XML Processor の動作]** 生成されるスキーマの要素定義，属性定義での，type 属性値は {http://ws-i.org/profiles/basic/1.1/xsd}swaRef になります。 名前空間 {http://ws-i.org/profiles/basic/1.1/xsd} の定義は次に示す xs:import として生成されます。 `<xs:import namespace="http://ws-i.org/profiles/basic/1.1/xsd" schemaLocation="http://ws-i.org/profiles/basic/1.1/swaref.xsd"/>` このため，生成したスキーマ文書で妥当性検証を実行すると，XML パーサが http プロトコルを用いて http://ws-i.org/profiles/basic/1.1/swaref.xsd を参照します。 javax.xml.validation.Validator#setResourceResolver メソッドを使用して適切な org.w3c.dom.ls.LSResourceResolver インスタンスを設定することで，任意の場所に存在する swaref.xsd を参照させることができます。
20	**[条件]** 次のすべての条件に該当する場合です。 1. jaxb:typesafeEnumClass カスタムバインディングを指定する。 2. 1. のスキーマ文書をスキーマコンパイラに入力する。 3. 2. で列挙型の fromValue メソッドが生成され，その引数に列挙定数以外を指定して呼び出す。 **[Cosminexus XML Processor の動作]** IllegalArgumentException 例外が発生します。
21	**[条件]** 次のすべての条件に該当する場合です。 • package-info.java が存在する。 • JAXBContext.newInstance(Class... classesToBeBound) を実行後，スキーマ生成やアンマーシャル，マーシャルのメソッドを実行する。 **[Cosminexus XML Processor の動作]** package-info.java を事前にコンパイルしていない場合，処理対象となりません。
22	**[条件]** 次のすべての条件に該当する場合です。 • @XmlAnyElement(lax = true) を，Object 型のフィールドまたは JavaBean プロパティのどちらかに付加する。 • 要素がスキーマ要素定義に一致しないが，その要素の xsi:type 属性に指定された型名がスキーマで定義されているインスタンス文書をアンマーシャルする。 **[Cosminexus XML Processor の動作]** JAXB オブジェクト（スキーマ要素定義に対応する Java クラスのオブジェクト）が生成されないで，DOM ノードが生成されます。

項番	注意事項
23	**[条件]** 次に示すどれかのクラスで，setEventHandler メソッドの引数に null を指定する場合です。 • Unmarshaller クラス • Marshaller クラス • Binder<Node> クラス **[Cosminexus XML Processor の動作]** getEventHandler メソッドで取得したイベントハンドラは，setEventHandler メソッドを実行しなかった場合のイベントハンドラと異なり，JAXB1.0 のデフォルトイベントハンドラ（DefaultValidationEventHandler）となります。このような指定をした場合の動作は保証しません。
24	**[条件]** 次のすべての条件に該当する場合です。 • Binder<Node> クラスの setProperty メソッドで，jaxb.encoding プロパティにデフォルト以外の値を設定する。 • Binder<Node> クラスの marshal メソッドを実行する。 **[Cosminexus XML Processor の動作]** 設定されたプロパティは有効となりません。
25	**[条件]** 次のすべての条件に該当する場合です。 • Binder<Node> クラスの setProperty メソッドで jaxb.formatted.output プロパティを設定しない，または false の値を設定する。 • Binder<Node> クラスの marshal メソッドを実行する。 **[Cosminexus XML Processor の動作]** プロパティの設定に関係なく，改行およびインデントが処理された XML データが出力されます。
26	**[条件]** javax.xml.bind.Binder<XmlNode> クラスの unmarshal(XmlNode xmlNode, Class<T> declaredType) に，次のように指定する場合です。 • declaredType に指定したクラスに一致しない xmlNode を指定する。 **[Cosminexus XML Processor の動作]** 例外は発生しません。このような場合の JAXB 実行時の動作は規定されていないため，保証しません。
27	**[条件]** javax.xml.bind.Binder<XmlNode> クラスの updateXML(Object jaxbObject, XmlNode xmlNode) に，次のように指定する場合です。 • 引数 jaxbObject に unmarshal および marshal で関連づけられていない，JAXB オブジェクトを指定する。 • 引数 xmlNode に関連づけられたあとに削除された DOM ノードを指定する。 **[Cosminexus XML Processor の動作]** このような場合の JAXB 実行時の動作は規定されていないため，保証しません。

項番	注意事項
28	**[条件]** javax.xml.bind.Binder<XmlNode> クラスの updateXML メソッドを実行する場合です。 **[Cosminexus XML Processor の動作]** JAXB オブジェクトに関連づけられた既存の XML ツリーを更新しないで，新しい XML ツリーが作成されます。そのため，既存の XML ツリーに次の情報が含まれる場合，それらの情報はメソッド実行後保持されません。 • コメント • PI • JAXB にバインドされない XML 要素または属性 また，updateXML(Object jaxbObject, XmlNode xmlNode) メソッドの戻り値は，引数の xmlNode と同じノードにはなりません。そのため，updateXML メソッドを繰り返し実行する場合，引数 xmlNode に同じ値を指定しないでください。

29	**[条件]**
	javax.xml.bind.Binder<XmlNode> クラスの updateXML メソッドを実行する場合です。
	
	[Cosminexus XML Processor の動作]
	javax.xml.bind.Binder<XmlNode> クラスの実装が持つ，JAXB オブジェクトツリーと XML ノードの関連づけが不正になります。
	そのため，メソッド実行後，戻り値を getJAXBNode(XmlNode xmlNode) の引数に指定した実行結果は，null となります。また，updateXML メソッドを繰り返し実行した場合，NullPointerException 例外が発生します。
	
	[回避策]
	次のように変更し，updateXML メソッドの呼び出しに加えて，javax.xml.bind.Binder<XmlNode> クラスの updateJAXB メソッドの呼び出しを実施してください。
	(変更前)
	binder.updateXML(rootbinder);
	
	(変更後)
	Node node = binder.updateXML(rootbinder);
	rootbinder = (Root)binder.updateJAXB(node);

30	**[条件]** 次のどちらかの条件に該当する場合です。 • アンマーシャル時，XML データの文字列をターゲットの Java データ型の値に変換している途中で，問題が発生した。 • マーシャル時，Java コンテンツツリーのデータを Java コンテンツツリーの字句表現に変換している途中で，問題が発生した。 **[Cosminexus XML Processor の動作]** イベント（アンマーシャル時の parseConversionEvent，またはマーシャル時の printConversionEvent）は生成されません。

項番	注意事項

31	**[条件]**
	次のすべての条件に該当する場合です。
	1. XML 文書に xsi:type 属性を指定する。
	2. 1. の属性値に，要素名および属性名で使用していない名前空間接頭辞を指定する。
	3. 1. の XML 文書をアンマーシャルする。
	4. 3. でアンマーシャルしたオブジェクトを検証モードでマーシャルする。
	
	（例）
	`<foo xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">`
	`<a/>`
	`<b xsi:type="xsd:string" xmlns:xsd="http://www.w3.org/2001/XMLSchema">abc</b>`
	`<c xsi:type="xsd:int" xmlns:xsd="http://www.w3.org/2001/XMLSchema">123</c>`
	`</foo>`
	
	[Cosminexus XML Processor の動作]
	KECX06253-E UndeclaredPrefix: Cannot resolve 'xsd:string' as a QName: the prefix 'xsd' is not declared. のエラーが報告されます。
	また，検証モードなしでマーシャルを実施すると，必要な名前空間宣言がない XML 文書が出力されます。

32	**[条件]** 次のすべての条件に該当する場合です。 1. フィールド A が java.util.Calendar クラスとして宣言されている。 2. 1. のフィールドが @XmlSchemaType(name="date") を使ってアノテーションされている。 3. 1. および 2. に該当するフィールドを持つクラスをマーシャルする。 **[Cosminexus XML Processor の動作]** フィールド A に対応する XML 文書中の要素の内容が，xs:date 型ではなく，xs:dateTime 型として出力されます。 そのため，JAXBContext クラスの generateSchema メソッドで生成したスキーマを使用してスキーマ検証すると，検証エラーになります。 **[回避策]** マーシャル時に xs:date 形式で出力したい場合には，java.util.Calendar クラスではなく，javax.xml.datatype.XMLGregorianCalendar クラスを使用してください。
33	**[条件]** ValidationEventLocator インタフェースの getOffset メソッドを実行する場合です。 **[Cosminexus XML Processor の動作]** getOffset メソッドの戻り値は，常に -1 になります。

付録

付録 A　バージョン間の差異

Cosminexus XML Processor の以前のバージョンとの動作の差異について説明します。

付録 A.1　DOM パーサ，および SAX パーサ共通の動作の差異

DOM パーサ，および SAX パーサ共通の動作の差異を次の表に示します。

表 A-1　DOM パーサ，および SAX パーサ共通の動作の差異

項番	条件	Cosminexus XML Processor の動作	
		バージョン 06-00 の場合	バージョン 07-00 以降の場合
1	パーサが InputSource オブジェクトを使用してバイトストリームを読み込む場合	XML 仕様のアルゴリズムを用いて文字エンコーディングを自動検出します。	setEncoding メソッドで指定されたエンコーディングを優先します。
2	「EUC で保存されているが XML 宣言に encoding 指定が存在しない XML 文書」など，エンコーディングが不正な状態で解析した場合	IOException 例外が発生します。	SAXException 例外が発生します。
3	parse メソッドの引数 uri または引数 systemId に空文字列を指定した場合	IOException 例外が発生します。	SAXException 例外が発生します。
4	XML 文書の DOCTYPE 宣言のシステム識別子に空文字列を指定した場合	IOException 例外が発生します。	SAXException 例外が発生します。
5	DTD で EMPTY と宣言した要素の内容がコメントだけの場合	妥当性検証時にエラーになりません。	妥当性検証時にエラーになります。

付録 A.2　DOM パーサの動作の差異

DOM パーサの動作の差異を次の表に示します。

表 A-2　DOM パーサの動作の差異

項番	条件	Cosminexus XML Processor の動作	
		バージョン 06-00 の場合	バージョン 07-00 以降の場合
1	同一名の NOTATION 宣言があり，かつ，妥当性検証が有効な場合	重複した NOTATION 宣言を破棄します。	重複した NOTATION 宣言を破棄し，さらにエラーになります。

項番	条件	Cosminexus XML Processor の動作	
		バージョン 06-00 の場合	バージョン 07-00 以降の場合
2	あるノードに対して，そのノードまたは祖先ノードを引数 newChild として，appendChild メソッドまたは insertBefore メソッドを実行した場合	DOMException 例外のコードは WRONG_DOCUMENT_ERR です。	DOMException 例外のコードは HIERARCHY_REQUEST_ERR です。
3	Document インタフェースの createElementNS(String namespaceURI, String qualifiedName) メソッドの引数 qualifiedName に null を指定した場合	DOMException 例外のコードは INVALID_CHARACTER_ERR です。	DOMException 例外のコードは NAMESPACE_ERR です。

付録 A.3　SAX パーサの動作の差異

SAX パーサの動作の差異を次の表に示します。

表 A-3　SAX パーサの動作の差異

項番	条件	Cosminexus XML Processor の動作	
		バージョン 06-00 の場合	バージョン 07-00 以降の場合
1	XMLReaderFactory クラスの createXMLReader() メソッドを使用する場合	システムプロパティ "org.xml.sax.driver" の値を用いて XMLReader インスタンスの生成を試みます。	次の順序で XMLReader インスタンスの生成を試みます。 1. システムプロパティ "org.xml.sax.driver" が示すクラス名 2. JAR ファイル内の META-INF/services/org.xml.sax.driver が示すクラス名 3. デフォルトの XMLReader クラス名 4. ParserFactory クラスの makeParser() メソッドの戻り値を ParserAdapter でラップしたクラス
2	XMLReader オブジェクトに ContentHandler オブジェクトを設定した場合	エンティティをスキップしても ContentHandler オブジェクトの skippedEntity メソッドは呼び出されません。	エンティティをスキップすると，ContentHandler オブジェクトの skippedEntity メソッドが呼び出されます。

項番	条件	Cosminexus XML Processor の動作	
		バージョン 06-00 の場合	バージョン 07-00 以降の場合
3	DeclHandler クラスの externalEntityDecl(String name, String publicId, String systemId) メソッド，および，DTDHandler インタフェースの notationDecl(String name, String publicId, String systemId) メソッドの引数 systemId が URL の場合	URL が完全に解決されないことがあります。	URL が完全に解決されます。
4	XMLReader クラスの setEntityResolver メソッド，setDTDHandler メソッド，setContentHandler メソッド，setErrorHandler メソッドの引数に null を指定した場合	NullPointerException 例外が発生します。	リゾルバやハンドラがクリアされます。
5	ParserAdapter クラスの setEntityResolver メソッド，setDTDHandler メソッド，setContentHandler メソッド，setErrorHandler メソッドの引数に null を指定した場合	NullPointerException 例外が発生します。	リゾルバやハンドラがクリアされます。
6	XMLFilterImpl クラスの setEntityResolver メソッド，setDTDHandler メソッド，setContentHandler メソッド，setErrorHandler メソッド，setParent メソッドの引数に null を指定した場合	NullPointerException 例外が発生します。	リゾルバ，ハンドラ，および親リーダなどがクリアされます。
7	SAX パーサのフィーチャー "http://xml.org/sax/features/namespaces" を false に設定し，かつ，フィーチャー "http://xml.org/sax/features/namespace-prefixes" を true に設定した場合	接頭辞 "xml","xmlns" に対する startPrefixMapping メソッドおよび endPrefixMapping メソッドが呼び出されます。	接頭辞 "xml","xmlns" に対する startPrefixMapping メソッドおよび endPrefixMapping メソッドは呼び出されません。

付録 A.4　スキーマ検証の動作の差異

スキーマ検証の動作の差異を次の表に示します。

表 A-4　スキーマ検証の動作の差異

項番	条件	Cosminexus XML Processor の動作	
		バージョン 06-00 の場合	バージョン 07-00 以降の場合
1	スキーマ文書に xsd:key 要素が存在し，かつ，インスタンス文書にキーに該当するテキストが存在しない場合	エラーにならないことがあります。	エラーになります。
2	スキーマ文書に xsd:key 要素と xsd:keyref 要素の組み合わせを指定していて，xsd:key 要素と xsd:keyref 要素の双方の field 要素の順序と型が一致していない場合	エラーにならないことがあります。	エラーになります。
3	xsd:field 要素の xpath 属性が child 軸を含む XPath 式の場合	エラーになりません。	child 軸が指定できない旨のエラーになります。
4	xsd:selector 要素の xpath 属性が「| ns1:element」のように不当な | 演算子を含む XPath 式の場合	エラーにならないか，または，XML 文書の妥当性検証のエラーになります。	スキーマ文書のエラーになります。
5	xsd:redefine 要素または xsd:import 要素の schemaLocation 属性に空文字列を指定した場合	エラーになりません。	エラーになります。
6	xsd:schema 要素の targetNamespace 属性に "http://www.w3.org/2001/XMLSchema-instance" を指定している場合	－（該当しない）	• 08-00 では KECX06509-E のエラーを通知します。 • 08-50 以降ではエラーを通知しません。ただし，広域属性が存在するときには，KECX06198-E のエラーを通知します。
7	次のすべての条件に該当する場合 1. XML スキーマ文書で anyURI 型またはその派生型を使用している 2. 1. の定義に対応するデータが RFC2396 および RFC2732 の規定に反する文字を含んでいる	KECX06036-E のエラーが発生します。	データ型宣言に適合した値として処理されます。
8	次のすべての条件に該当する場合 1. XML スキーマ文書で time 型または dateTime 型を使用している 2. 1. の定義に対応するデータに，秒の値として「60」が指定されている	データ型宣言に適合した値として処理されます。	データ型宣言の範囲外の値と見なされて，KECX06036-E エラーが発生します。

項番	条件	Cosminexus XML Processor の動作	
		バージョン 06-00 の場合	バージョン 07-00 以降の場合
9	次のすべての条件に該当する場合 1. XML スキーマ文書で time 型または dateTime 型を使用している 2. 1. の定義に対応するデータに，タイムゾーンオフセットとして -14:01 ～ -14:59 または +14:01 ～ +14:59 の範囲の値が指定されている	データ型宣言に適合した値として処理されます。	データ型宣言の範囲外の値と見なされて，KECX06036-E エラーが発生します。

注

メッセージの詳細については，マニュアル「Cosminexus アプリケーションサーバ メッセージ 3」の「2.　KECX（Cosminexus XML Processor が出力するメッセージ）」を参照してください。

付録 A.5　XSLT の動作の差異

XSLT の動作の差異を次の表に示します。

表 A-5　XSLT の動作の差異（バージョン 06-00 とバージョン 07-00 の比較）

項番	条件	Cosminexus XML Processor の動作	
		バージョン 06-00 の場合	バージョン 07-00 以降の場合
1	Transform クラスおよび TransformerFactory クラスのデフォルトのエラーリスナーを使用する場合	エラー時に例外をスローします。	エラー時に例外をスローしません。
2	TransformerFactory クラスの newTransformer(Source source) メソッド , newTemplates メソッドを実行した場合	バージョン間の差異はありません。 これらのメソッドの入力であるスタイルシートに誤りがあると，ErrorListener の fatalError メソッドにエラーが報告されます。このとき，ErrorListener の fatalError メソッドの実装が TransformerException をスローしない場合は，newTransformer(Source source) メソッド，newTemplates メソッドの戻り値が null になることがありますのでご注意ください。 デフォルトの ErrorListener は，TransformerException をスローしません。したがって，回復できないエラー発生時の正常な動作を確保するため，ErrorListener を登録して fatalError メソッドの中から TransformerException 例外をスローすることを強くお勧めします。	

項番	条件	Cosminexus XML Processor の動作	
		バージョン 06-00 の場合	バージョン 07-00 以降の場合
3	XML をファイルに出力する際，xsl:output 要素に indent="yes" が指定されていないか，または，Transform クラスの setOutputProperty(OutputKeys.INDENT, "yes") メソッドが実行されていない場合	XML 宣言の直後に改行が出力されます。	改行は出力されません。
4	変換結果を HTML 形式で出力した場合	要素間の改行出力に差異がありますが，HTML 文書では，要素間の改行は意味を持たないため，問題ありません。	
5	system-property('xsl:version') 関数を実行した場合	結果は "1" になります。	結果は "1.0" になります。
6	xsl:attribute 要素の内容がタブを含む文字列の場合	変換結果の属性値がタブ文字そのものになります。	変換結果の属性値が文字参照（ ）になります。
7	xsl:number 要素の value 属性に generate-id() 関数を使用した場合	"0" が出力されます。	"NaN" が出力されます。
8	xsl:stylesheet 要素の名前空間宣言で定義した接頭辞が，xsl:element 要素によって作成された要素の中で使われない場合	その名前空間宣言が xsl:element 要素に出力されます。	その名前空間宣言は xsl:element 要素には出力されません。
9	xsl:version 属性に "1.1" を指定した簡略化スタイルシートの中にトップレベル要素を記述した場合	前方互換処理で，トップレベル要素がインスタンス化されて変換が実行されます。	トップレベル要素前方互換処理が無視されます。
10	xsl:attribute-set 要素の use-attribute-sets 属性に「存在しない属性セット名」を指定した場合	エラーになりません。	属性セットが存在しない旨のエラーになります。
11	xsl:template 要素に name 属性，match 属性のどちらも指定されていない場合	エラーになりません。	name 属性か match 属性のどちらかが必要な旨のエラーになります。
12	exclude-result-prefixes 属性で指定された「除外する名前空間の接頭辞」が，xsl:stylesheet 要素の名前空間宣言で定義されていない場合	エラーになりません。	エラーになります。
13	xsl:stylesheet 要素の version 属性に数値以外を指定した場合	エラーになりません。	エラーになります。
14	xsl:output 要素の method 属性に "xml" を指定し，かつ，xsl:output 要素の version 属性に "1.0"，"1.1" 以外を指定した場合	警告は発生しません。	"1.0" を仮定する旨の警告が発生します。
15	テキスト出力で，出力時のエンコーディングでは表現できない文字を出力する場合	文字化けとなります。	標準エラー出力に警告が出力され，該当の文字は文字参照として出力されます。

項番	条件	Cosminexus XML Processor の動作	
		バージョン 06-00 の場合	バージョン 07-00 以降の場合
16	Transform クラスの transform(Source xmlSource, Result outputTarget) メソッドの引数 xmlSource に空の Source を指定した場合	エラーになります。	DocumentBuilder クラスの newDocument メソッドで生成された空のドキュメントが使われます。
17	TransformerFactory クラスの getFeature メソッドの引数に null を指定した場合	false が戻ります。	NullPointerException 例外が発生します。
18	名前空間宣言のない接頭辞がある場合	TransformerException 例外が発生します。	XPathStylesheetDOM3Exception 例外が発生することがあります。この例外は TransformerException 例外のサブクラスなので，TransformerException によってキャッチできます。
19	StreamResult クラスの StreamResult(File f) コンストラクタ，および，setSystemId(File f) メソッドの引数 f に null を指定した場合	NullPointerException 例外が発生します。	RuntimeException 例外が発生します。 ただし，Java SE 6 の場合は NullPointerException 例外が発生します。
20	xsl:call-template 要素の name 属性で指定した名前付きテンプレートが存在しない場合 （メッセージの詳細については，マニュアル「Cosminexus アプリケーションサーバ メッセージ 3」の「2. KECX（Cosminexus XML Processor が出力するメッセージ）」を参照してください）	KECX02019-E のエラーを通知します。	KECX02019-E および KECX02015-E のエラーを通知します。
21	「namespace:: 局所名」を含む XPath 式の評価結果	局所名を接頭辞と見なして名前空間 URI を解決し，その名前空間 URI に一致する文字列値を持つ名前空間ノードが評価結果になります。	指定した局所名を持つ名前空間ノードが評価結果になります。

表 A-6　XSLT の動作の差異（バージョン 08-50 とバージョン 08-70 の比較）

項番	条件	Cosminexus XML Processor の動作	
		バージョン 08-50 の場合	バージョン 08-70 の場合
1	XMLInputFactory.createXMLEventReader() メソッドで取得した XMLEventReader を StAXSource のコンストラクタ引数に指定して，Transformer.transform() メソッドの入力ソースとした場合	入力 XML 文書のルート要素の外側にある処理命令は無視されます。	入力 XML 文書のルート要素の外側にある処理命令も変換対象となります。

項番	条件	Cosminexus XML Processor の動作	
		バージョン 08-50 の場合	バージョン 08-70 の場合
2	次の両方の条件に該当する場合 • トランスフォーマの入力 XML 文書内の要素が XSLT 以外の名前空間に属している • XMLEventWriter から作成した StAXResult を使用して変換している	変換中にエラーが発生するか，変換結果が不正になります。	正常に処理されます。

付録 A.6　JAXB の動作の差異

JAXB の動作の差異を，状況別に次の表に示します。

表 A-7　JAXB の動作の差異（csmxjc コマンド）

項番	条件	Cosminexus XML Processor の動作	
		バージョン 08-50 の場合	バージョン 08-70 の場合
1	enumeration ファセットが存在しない XML スキーマ文書の単純型に typesafeEnumClass カスタムバインディングが存在する場合	csmxjc コマンドが正常終了します。	typesafeEnumClass カスタムバインディングの指定が不正であることを示すエラーが発生します。
2	次のどちらかの場合に該当するとき • globalBindings カスタムバインディングで，generateElementClass="true" または generateElementClass="1" を指定した場合 • XML スキーマ文書の element 要素に class カスタムバインディングを指定した場合	要素クラスの参照プロパティは，要素のデータ型（@XmlElement または @XmlElements）で注釈されます。 なお，その要素に対応するソースにデフォルトコンストラクタは出力されません。	要素クラスの参照プロパティは，要素のクラス型（@XmlElementRef または @XmlElementRefs）で注釈されます。 デフォルトコンストラクタが出力されます。
3	group 要素中の element 要素に class カスタムバインディングを指定して，その group 要素がほかの複数の型から参照されている場合	group 要素中の element 要素のうち，class カスタムバインディング指定されていない要素のオブジェクトを作成するファクトリメソッドは，生成されません。	group 要素中の element 要素のうち，class カスタムバインディング指定されていない要素のオブジェクトを作成するファクトリメソッドが，生成されます。
4	globalBindings または property カスタムバインディングに collectionType="indexed" の指定がある場合	csmxjc コマンドで生成したソースの element 要素に対する型は，java.util.List です。	csmxjc コマンドで生成したソースの element 要素に対する型は，配列です。

項番	条件	Cosminexus XML Processor の動作	
		バージョン 08-50 の場合	バージョン 08-70 の場合
5	型定義や匿名型要素定義に factoryMethod カスタムバインディングの指定がある状態で csmxjc コマンドを実行した場合	エラーになります。	正常終了します。
6	csmxjc コマンドで生成したソースに @XmlElementRef 注釈型※が含まれる場合	required 要素は出力されません。	required 要素が出力されます。
7	@XmlAttribute 注釈型の name 要素が生成するフィールド名と同じ場合	name 要素は出力されません。	name 要素が出力されます。

注※

@XmlElementRef 注釈型の required 要素は，JAXB 2.2 でサポートされました。

表 A-8　JAXB の動作の差異（csmschemagen コマンド）

項番	条件	Cosminexus XML Processor の動作	
		バージョン 08-50 の場合	バージョン 08-70 の場合
1	Java ソースに @XmlAccessorOrder または @XmlAccessorOrder(XmlAccessOrder.UNDEFINED) を指定した場合	csmschemagen コマンドで作成した XML スキーマ文書が sequence 要素になります。	csmschemagen コマンドで作成した XML スキーマ文書が all 要素になります。
2	Java ソースに @XmlElementDecl を指定した場合	csmschemagen コマンドで生成した XML スキーマ文書の大域定義の element 要素に，nillable="true" 属性が出力されます。	csmschemagen コマンドで生成した XML スキーマ文書の大域定義の element 要素に，nillable="true" 属性は出力されません。

表 A-9　JAXB の動作の差異（実行時）

項番	条件	Cosminexus XML Processor の動作	
		バージョン 08-50 の場合	バージョン 08-70 の場合
1	@XmlAttribute 注釈型または @XmlValue 注釈型で注釈された java.lang.Object 型の変数が存在するクラス，またはそのクラスが存在するパッケージ名を，JAXBContext クラスの newInstance メソッドに指定した場合	IllegalAnnotationsException がスローされます。	NullPointerException がスローされます。

項番	条件	Cosminexus XML Processor の動作	
		バージョン 08-50 の場合	バージョン 08-70 の場合
2	JAXBContext クラスの newInstance メソッドに指定したパッケージに package-info クラスが存在し，XmlNs 注釈型で名前空間が指定されている場合	marshal で出力される XML 文書に名前空間宣言が出力されません。	marshal で出力される XML 文書に名前空間宣言が出力されます。
3	@XmlMixed 注釈型で注釈された JavaBean プロパティに空白類だけのテキストが含まれる XML 文書をアンマーシャルした場合	アンマーシャルで生成されたオブジェクトにテキストオブジェクトが生成されません。	空白類だけのテキストオブジェクトが生成されます。
4	スキーマ検証やイベントハンドラで，エラーチェックをしない Unmarshaller で数値のプリミティブ型の JavaBean プロパティに，文字列をアンマーシャルした場合	IllegalAccessError となります。	アンマーシャルが正常終了します。
5	JAXB にマッピングされたクラスに beforeUnmarshal メソッドまたは afterUnmarshal メソッドを実装して，アンマーシャル実行時にそのメソッドで例外がスローされた場合	アンマーシャル処理は中断しません。	UnmarshalException がスローされて，アンマーシャル処理が中断します。
6	JAXBContext クラスの newInstance(Class[], Map<String, ?>) メソッドおよび newInstance(Class[]) メソッドの第 1 パラメタに null を指定した場合	NullPointerException がスローされます。	IllegalArgumentException がスローされます。
7	DatatypeConverter クラスの parseBoolean メソッドに "0","1","false","true" 以外を指定した場合	false が返ります。	IllegalArgumentException がスローされます。
8	DatatypeConverter クラスの parseDecimal メソッドに空白だけを指定した場合	java.lang.NumberFormatException がスローされます。	例外はスローされません。
9	enum クラスにアンマーシャルする値に改行や空白が含まれる場合	処理結果が不正になります。	値が正しくアンマーシャルされます。

付録 B　JAXB 仕様のサポート範囲

JAXB 仕様のサポート範囲について説明します。機能の詳細については，各製品のマニュアルを参照してください。

付録 B.1　JAXB の機能のサポート範囲

JAXB の機能のサポート範囲を次の表に示します。

表 B-1　JAXB の機能のサポート範囲

JAXB の機能		機能の概要	サポートの可否
スキーマコンパイラ	デフォルトバインディング	デフォルトの変換をします。	○
	カスタムバインディング	バインディング宣言によって変換をカスタマイズします。	○
	├ インライン注釈付きスキーマ	スキーマ文書内にバインディング宣言を記述します。	○
	├ 外部バインディング宣言	外部ファイルにバインディング宣言を記述します。	○
	└ バインディング言語の拡張	ベンダ固有の拡張バインディング宣言を定義します。	×
スキーマジェネレータ	デフォルトマッピング	デフォルトの変換をします。	○
	カスタムマッピング	Java のアノテーションによって変換をカスタマイズします。	○

JAXB の機能		機能の概要	サポートの可否
バインディングフレームワーク	マーシャル	Java のオブジェクトを XML 文書として出力します。	○
	├ イベントコールバック	マーシャル時に発生するイベントのコールバックメソッドです。	○
	│ ├ クラス定義されたコールバックメソッド	マーシャルの前後に呼び出される，JAXB によってマッピングされたクラス内部のコールバックメソッドです。	○
	│ └ 外部リスナー	マーシャル時に発生するイベントによって呼び出される，外部のコールバックメソッドです。	○
	├ マーシャリングプロパティ	マーシャルを制御するための各種プロパティです。	○
	└ 妥当性検証	マーシャル時に妥当性を検証します。	○
	アンマーシャル	XML 文書を Java のオブジェクトとして読み込みます。	○
	├ イベントコールバック	アンマーシャル時に発生するイベントのコールバックメソッドです。	○
	│ ├ クラス定義されたコールバックメソッド	アンマーシャルの前後に呼び出される，JAXB によってマッピングされたクラス内部のコールバックメソッドです。	○
	│ └ 外部リスナー	アンマーシャル時に発生するイベントによって呼び出される，外部のコールバックメソッドです。	○
	└ 妥当性検証	アンマーシャル時に妥当性を検証します。	○
	イントロスペクタ	JAXB によってマッピングされた Java オブジェクトのマッピングに関する情報にアクセスするための機能です。	○
	バインダー	JAXB によってマッピングされた Java オブジェクトと XML 情報セットの同期を取るための機能です。	○

（凡例）
○：機能をすべてサポートしています。
×：未サポートです。

付録 B.2　JAXB の文字のサポート範囲

スキーマコンパイラの入力となるスキーマ文書およびスキーマジェネレータの入力となる Java クラスは，任意の文字列を指定できます。JAXB で扱えるスキーマジェネレータ入力 java ソースの文字サポート範囲を表 B-2，スキーマコンパイラ入力スキーマ文書の

文字サポート範囲を表 B-3 に示します。

表 B-2　スキーマジェネレータ入力 java ソースの文字サポート範囲

項番	任意文字列の指定個所	文字サポート範囲
1	クラス名 メソッド名 フィールド名	次のすべての条件に合致するようにしてください。 • 英数字およびアンダーラインから成る。 • XML の NCName の構文規則に合致する。
2	列挙定数	次の条件に合致するようにしてください。 • 英数字およびアンダーラインから成る。
3	@XmlNs の prefix 要素	次のすべての条件に合致するようにしてください。 • 英数字およびアンダーラインから成る。 • XML の NCName の構文規則に合致する。
4	@XmlEnumValue の要素 @XmlElement/@XmlElementDecl の defaultValue 要素	次の条件に合致するようにしてください。 • Unicode の Basic Latin および日本語※ 1 から成る。
5	JAXB マッピングアノテーションの name 要素	次のすべての条件に合致するようにしてください。 • 英数字，アンダーラインと日本語※ 1 から成る。 • XML の NCName の構文規則に合致する。
6	JAXB マッピングアノテーションの namespace 要素	次の条件に合致するようにしてください。 • RFC 2396 で規定された URI 文字列である。※ 2
7	@XmlType の factoryMethod 要素	次のすべての条件に合致するようにしてください。 • 英数字およびアンダーラインから成る。 • Java 識別子の構文規則に合致する。
8	@XmlMimeType の value 要素	次の条件に合致するようにしてください。 • MIME 型テキスト表現である。

注※ 1

Unicode の Hiragana，Katakana，CJK Unified Ideographs のカテゴリに含まれる文字です。

注※ 2

RFC2732 は未サポートです（IPv6 未サポート）。

表 B-3　スキーマコンパイラ入力スキーマ文書の文字サポート範囲

項番	任意文字列の指定個所	文字サポート範囲
1	xs:anyURI 型を指定するスキーマ要素の属性（xs:schema 要素の targetNamespace 属性など）	次の条件に合致するようにしてください。 • RFC 2396 で規定された URI 文字列である。※ 1

項番	任意文字列の指定個所	文字サポート範囲
2	xs:NCName 型を指定するスキーマ要素の属性（xs:element 要素の name 属性など）	カスタムバインディング宣言によって出力名を変更しない場合は，次のすべての条件に合致するようにしてください。 • 英数字およびアンダーラインから成る。 • XML の NCName の構文規則に合致する。
		カスタムバインディング宣言によって出力名を変更する場合は，次のすべての条件に合致するようにしてください。 • 英数字，アンダーラインおよび日本語※2 から成る。 • XML の NCName の構文規則に合致する。
3	xs:QName 型を指定するスキーマ要素の属性（xs:element 要素の type 属性など）	次のすべての条件に合致するようにしてください。 • QName 形式の文字列。 • 局所名は項番 2 の範囲内とする。 • 接頭辞は英数字およびアンダーラインから成る。
4	xs:string 型を指定するスキーマ要素の属性（xs:element 要素の fixed/default 属性 xs:attribute 要素の fixed/default 属性など） jaxb:typesafeEnumMember 要素の value 属性	次の条件に合致するようにしてください。 • Unicode の Basic Latin および日本語※2 から成る。
5	xs:enumeration 要素の value 属性	カスタムバインディングによって列挙定数を変更しない場合は，次のすべての条件に合致するようにしてください。 • 英数字およびアンダーラインから成る。 • Java 列挙定数の構文規則に合致する。
		カスタムバインディングによって列挙定数を変更する場合は，次の条件に合致するようにしてください。 • Unicode の Basic Latin および日本語※2 から成る。
6	JAXB 要素の name 属性 JAXB 要素の suffix 属性 JAXB 要素の prefix 属性 jaxb:javaType 要素の parseMethod/ printMethod 属性	次のすべての条件に合致するようにしてください。 • 英数字およびアンダーラインから成る。 • Java 識別子の構文規則に合致する。
7	jaxb:package 要素の name 属性 JAXB 要素の ref 属性 JAXB 要素の collectionType 属性 jaxb:class 要素の implClass 属性	次のすべての条件に合致するようにしてください。 • 英数字，アンダーラインおよびピリオドから成る。 • Java パッケージ名の構文規則に合致する。
8	jaxb:javadoc 要素の要素内容	次のすべての条件に合致するようにしてください。 • Unicode の Basic Latin および日本語※2 から成る。 • Java コメントの構文規則に合致する。

注※ 1

RFC2732 は未サポートです（IPv6 未サポート）。

注※ 2

Unicode の Hiragana，Katakana，CJK Unified Ideographs のカテゴリに含まれる文字です。

付録 B.3　JAXB 仕様書でベンダ固有とされている機能のサポート範囲

JAXB 仕様書でベンダ固有とされている機能のサポート範囲を次の表に示します。

表 B-4　JAXB 仕様書でベンダ固有とされている機能のサポート

JSR 222 の該当個所	記載ページ	ベンダ固有機能の内容	Cosminexus XML Processor での機能サポート
4.3 General Validation Processing	35	Unmarshal-time validation（アンマーシャル時の検証）	アンマーシャル時に妥当性検証する機能をサポートしています。
		On-demand validation（オンデマンド検証）	オンデマンド検証機能はサポートしていません。
		Fail-first validation（フェイルファースト検証）	フェイルファースト検証機能はサポートしていません。
4.5 Marshalling	41	Potentially properties（マーシャル時の追加のプロパティ）	Cosminexus XML Processor 固有の「マーシャル時のプロパティ」はありません。
5.2 Java Package	50	Property setter/getter（Java 値クラスの追加のプロパティ）	Cosminexus XML Processor 固有の「Java 値クラスのプロパティ」はありません。
5.5.1 Simple Property	57	TypeConstraint validation（型制約検証）	型制約検証機能はサポートしていません。
6.13 Modifying Schema-Derived Code	153	Distinguish between generated and user added code（自動生成 Java ソースの識別）	csmxjc コマンドの -mark-generated によって，自動生成 Java ソースの識別をサポートしています。
7.1.1 Extending the Binding Language	159	Extending the Binding Language（バインディング宣言の拡張）	バインディング宣言の拡張機能はサポートしていません。
7.5<globalBindings> Declaration 7.8<property> Declaration	167 189	enableFailFastCheck（enableFailFastCheck 属性）	enableFailFastCheck 属性はサポートしていません。
Compatibility	294	non-default operating modes for binding schema languages（ほかのスキーマ言語のバインディング）	スキーマコンパイラは，XML Schema 以外のスキーマ言語をサポートしません。

JSR 222 の該当個所	記載ページ	ベンダ固有機能の内容	Cosminexus XML Processor での機能サポート
Compatibility	294	non-default operating modes for mapping Java types to schema languages （ほかのスキーマ言語マッピング）	スキーマジェネレータは，XML Schema 以外のスキーマ言語をサポートしません。
付録 G	355	Validator for JAXB 1.0 schema-derived classes （JAXB1.0 互換仕様）	Cosminexus XML Processor では，JAXB 1.0 で作成されたクラスを扱うことはできません。

付録 C　このマニュアルの参考情報

このマニュアルを読むに当たっての参考情報を示します。

付録 C.1　関連マニュアル

このマニュアルと関連する Cosminexus のマニュアルを次に示します。

- Cosminexus **アプリケーションサーバ** V8 **概説**（3020-3-U01）
  アプリケーションサーバの概要について説明しています。
- Cosminexus **アプリケーションサーバ** V8 **機能解説 保守／移行／互換編**（3020-3-U10）
  アプリケーションサーバで構築したシステムの保守に関する機能，移行情報，および互換用機能について説明しています。
- Cosminexus **アプリケーションサーバ** V8 **リファレンス 定義編（サーバ定義）**（3020-3-U15）
  アプリケーションサーバを構築・運用するとき，またはアプリケーションを開発するときに使用するファイルのうち，J2EE サーバや Management Server などのサーバの定義に使用するファイルの形式について説明しています。
- Cosminexus **アプリケーションサーバ** V8 **メッセージ** 3 KECX-KEDT ／ KEOS02000-29999 ／ KEUC-KFRM **編**（3020-3-U43）
  アプリケーションサーバで出力される KECX から KEDT までのメッセージ，KEOS02000 から KEOS29999 までのメッセージ，および KEUC から KFRM までのメッセージについて説明しています。

なお，このマニュアルでは，次のマニュアルについて，バージョン番号を省略して表記しています。マニュアルの正式名称とこのマニュアルでの表記を次の表に示します。

正式名称	このマニュアルでの表記
Cosminexus アプリケーションサーバ V8 概説	Cosminexus アプリケーションサーバ 概説
Cosminexus アプリケーションサーバ V8 機能解説 保守／移行／互換編	Cosminexus アプリケーションサーバ 機能解説 保守／移行／互換編
Cosminexus アプリケーションサーバ V8 リファレンス 定義編（サーバ定義）	Cosminexus アプリケーションサーバ リファレンス 定義編（サーバ定義）
Cosminexus アプリケーションサーバ V8 メッセージ 3 KECX-KEDT ／ KEOS02000-29999 ／ KEUC-KFRM 編	Cosminexus アプリケーションサーバ メッセージ 3

付録 C.2　このマニュアルでの表記

このマニュアルで使用している表記と，対応する製品名を次に示します。

このマニュアルでの表記			製品名称
UNIX	AIX		AIX 5L V5.3
			AIX V6.1
			AIX V7.1
	HP-UX	HP-UX（IPF）	HP-UX 11i V2（IPF）
			HP-UX 11i V3（IPF）
	Linux	Linux（IPF）	Red Hat Enterprise Linux(R) AS 4（IPF）
			Red Hat Enterprise Linux(R) 5（Intel Itanium）
			Red Hat Enterprise Linux(R) 5 Advanced Platform（Intel Itanium）
		Linux（x86／AMD64 & Intel EM64T）	Red Hat Enterprise Linux(R) AS 4（x86）
			Red Hat Enterprise Linux(R) 5 Advanced Platform（x86）
			Red Hat Enterprise Linux(R) 6 Advanced Platform（x86）
			Red Hat Enterprise Linux(R) ES 4（x86）
			Red Hat Enterprise Linux(R) 5（x86）
			Red Hat Enterprise Linux(R) 6（x86）
			Red Hat Enterprise Linux(R) AS 4（AMD64 & Intel EM64T）
			Red Hat Enterprise Linux(R) 5 Advanced Platform（AMD/Intel 64）
			Red Hat Enterprise Linux(R) 6 Advanced Platform（AMD/Intel 64）
			Red Hat Enterprise Linux(R) ES 4（AMD64 & Intel EM64T）
			Red Hat Enterprise Linux(R) 5（AMD/Intel 64）
			Red Hat Enterprise Linux(R) 6（AMD/Intel 64）
	Solaris		Solaris 10（SPARC）
			Solaris 10（x64）
			Solaris 9（SPARC）

また，Linux に関しては，バージョンごとに次のように表記することがあります。

表記	OS 名
Red Hat Enterprise Linux 4	Red Hat Enterprise Linux(R) AS 4（IPF）
	Red Hat Enterprise Linux(R) AS 4（x86）
	Red Hat Enterprise Linux(R) ES 4（x86）
	Red Hat Enterprise Linux(R) AS 4（AMD64 & Intel EM64T）
	Red Hat Enterprise Linux(R) ES 4（AMD64 & Intel EM64T）
Red Hat Enterprise Linux 5	Red Hat Enterprise Linux(R) 5 Advanced Platform（Intel Itanium）
	Red Hat Enterprise Linux(R) 5（Intel Itanium）
	Red Hat Enterprise Linux(R) 5 Advanced Platform（x86）
	Red Hat Enterprise Linux(R) 5（x86）
	Red Hat Enterprise Linux(R) 5 Advanced Platform（AMD/Intel 64）
	Red Hat Enterprise Linux(R) 5（AMD/Intel 64）
Red Hat Enterprise Linux Server 6	Red Hat Enterprise Linux(R) Server 6（32-bit x86）
	Red Hat Enterprise Linux(R) Server 6（64-bit x86_64）

このマニュアルで使用している表記と，対応する Java 関連用語を次に示します。

このマニュアルでの表記	Java 関連用語
EJB または Enterprise JavaBeans	Enterprise JavaBeans™
J2EE または Java 2 Platform, Enterprise Edition	J2EE™
	Java™ 2 Platform, Enterprise Edition
JAR	Java™ Archive
Java	Java™
Java SE	Java™ Platform, Standard Edition
JAXB または The Java Architecture for XML Binding	The Java™ Architecture for XML Binding
JAXP または Java API for XML Processing	Java™ API for XML Processing
JSP	JavaServer Pages™

付録 C.3　英略語

このマニュアルで使用している英略語を次に示します。

英略語	英字の表記
API	Application Programming Interface
DOM	Document Object Model
DTD	Document Type Definition
IANA	Internet Assigned Numbers Authority
IPF	Itanium(R) Processor Family
SAX	Simple API for XML
SAX1	Simple API for XML 1.0
SAX2	Simple API for XML 2.0
SOAP	Simple Object Access Protocol
TrAX	Transformation API for XML
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
W3C	World Wide Web Consortium
XML	Extensible Markup Language
XSL	Extensible Stylesheet Language
XSLT	XSL Transformations

付録 C.4　KB（キロバイト）などの単位表記について

1KB（キロバイト），1MB（メガバイト），1GB（ギガバイト），1TB（テラバイト）はそれぞれ 1,024 バイト，$1{,}024^2$ バイト，$1{,}024^3$ バイト，$1{,}024^4$ バイトです。

付録 D　用語解説

(英字)

DOM

Document Object Model の略です。DOM パーサによって解析された XML 文書の内容を記述したもので，メモリに常駐するツリー形式のデータ構造となっています。このツリー構造を DOM ツリーと呼びます。

DOM API

DOM パーサによって生成された DOM にアクセスするための API を DOM API と呼びます。XML 文書内のデータ構造に基づいたデータの処理ができます。

DOM ツリー

XML 文書の構造をツリー形式で表したものです。DOM ツリーは DOM パーサを使って解析すると生成されます。

DTD

Document Type Definition の略です。XML のスキーマの一つで，XML 文書で使用するタグの属性や階層構造や出現順序などを定義したものです。XML 文書はこの DTD で定義した文法に従って記述します。

grammar オブジェクト

スキーマ文書を解析したときに取得できる，スキーマ文法定義を表す内部のオブジェクトです。

IANA

Internet Assigned Numbers Authority の略です。インターネットの名前または番号の登録業務を行う組織です。

JAXB マッピングアノテーション

スキーマジェネレータを使用してスキーマ文書を生成するとき，入力となる Java ソースコード中に記述するアノテーションです。デフォルトのマッピングを変更する場合に使用します。

JAXP

Java API for XML Processing の略です。米国 Sun Microsystems, Inc. が提供する Java 言語用の標準 XML API です。DOM，SAX，および XSLT のリファレンス実装そのものを JAXP と呼ぶ場合もあります。

JAXP には，DOM パーサ，SAX パーサを生成する API が含まれています。パーサの実装やコードに依存しないので，プラットフォーム間で共通のプログラムを作成できます。また，JAXP には，XSLT トランスフォーマの API が含まれているので，Java プログラムから XSLT トランスフォーマを使用できます。

Namespaces in XML

XML で定義された複数の言語を一つの文書で扱う場合に，要素や属性の名前が衝突することを避けるために用いる修飾子です。

SAX

Simple API for XML の略です。XML 文書内の構文要素を逐次，通知することで XML 文書の内容にアクセスするためのコールバック API です。SAX の場合，解析中の個所に関する情報だけをユーザプログラムに渡し，DOM のようにツリーを作成しません。このため，DOM に比べてメモリ所要量が少なくなります。

StAX

Streaming API for XML の略です。JAXP の仕様に含まれるイベントベースの API で，Java Community Process の JSR 173 で定義されている Java の標準 API です。StAX の場合，XML 文書をパースするときにハンドラを使う必要がないため，XML 文書を手続き的に処理できます。

TrAX

Transformation API for XML の略です。JAXP の仕様に含まれる API で，XSLT トランスフォーマの標準 API です。
TrAX を使用することで Java プログラムから XML 変換処理を実現できます。単純な XML 文書の変換であれば TrAX だけで実現できます。

URI

Uniform Resource Identifier の略です。URL のように Web 上のデータの位置を指定する方法と，Web 上の特定のデータを固有の名前で指定する方法を持ち合わせた規格です。

URL

Uniform Resource Locator の略です。Web 上にある特定のデータの位置を指定する規格です。

W3C

World Wide Web Consortium の略です。XML や DOM を含む Web 関連技術の標準化を推進する非営利団体です。

XML

Extensible Markup Language の略です。W3C が制定したマークアップ言語で，インターネット上でデータのやり取りを実現します。このマニュアルでは，XML 形式で記述された文書を XML 文書と呼びます。

XML Schema

W3C 勧告で定められた仕様で，XML 文書内の論理的なデータ構造を規定するための仕組みです。

XML 仕様

W3C 勧告の「XML 1.0」および「XML 1.1」で定められた仕様です。Cosminexus XML Processor の XML パーサは，XML 仕様に従った XML 文書を処理できます。

XML パーサ

XML 文書を解析するプログラムです。
パーサには，DOM API を使用する DOM パーサと，SAX を使用する SAX パーサの 2 種類があります。DOM パーサは，XML 文書の解析結果を DOM ツリーとして出力します。SAX パーサは，DOM ツリーを出力しない代わりに，解析結果をイベントとして SAX に通知します。
このマニュアルではこれらのパーサを総称して XML パーサと呼びます。

XML 文書
XML 形式で記述された文書です。

XPath
XML 文書の特定の部分を指し示す構文を規定します。XPath を使用すれば XML 文書中にアンカーを指定しなくても，文書中の任意の位置を指定できます。

XSL
Extensible Stylesheet Language の略です。XML 用のスタイルシートのことで，表示方法を記述するだけでなく，データのコピー，分岐，抽出などのデータ処理を記述できます。

XSLT
XSL Transformations の略です。XML 文書をスタイルシートに従った XML 文書や HTML，テキストに変換するなどの，XML 文書の構造変換を行うための仕様です。

XSLTC トランスフォーマ
XSLTC トランスフォーマは，Cosminexus XML Processor が提供している XSLT トランスフォーマと同じ機能を持ちながら，XML 文書の変換処理時の性能を向上させたものです。XSLT スタイルシートを解析するコンパイラと変換処理を行う実行時プロセッサで構成されます。

XSLT トランスフォーマ
XSLT を実装したプログラムです。XML 文書をスタイルシートに従った XML 文書や HTML，テキストに変換できます。

（ア行）

アンマーシャル
JAXB API を使用して XML 文書を読み込み，Java のオブジェクトに変換することをアンマーシャルといいます。XML 文書をデシリアライズすることです。

インスタンス文書
スキーマ検証の対象となる XML 文書です。

エンティティ
XML 文書中で使用される画像データや PDF ファイルなどのファイルや，置換する文字列などのデータを保持する実体です。

（カ行）

カスタムバインディング宣言
スキーマコンパイラを使用して Java ソースファイルを生成するとき，入力のスキーマ文書中に記述するカスタマイズ指示です。デフォルトの JAXB バインディングを変更したい場合に使用します。

キャッシュテーブル
メモリキャッシュを格納する内部のテーブルです。J2EE サーバを起動したときに作成されます。

キャッシュファイル
ディスクキャッシュを格納するディスク上のファイルです。

（サ行）

スキーマキャッシュ
スキーマ文書を再利用するために保存された grammar オブジェクトです。
スキーマ定義ファイルの指定などによって，メモリ上またはディスク上に保存されます。

スキーマコンパイラ
スキーマ文書を入力し，JAXB マッピングアノテーションを追加した JavaBean 様式の Java ソースコードを生成するプログラムです。スキーマ文書中にカスタムバインディング宣言によるユーザの指示がある場合は，指示に基づいてスキーマ要素を Java ソースコードに結び付けます。

スキーマジェネレータ
Java ソースファイルを入力し，クラスやフィールドなどの宣言部からスキーマ文書を生成するプログラムです。Java クラス中に JAXB マッピングアノテーションによるユーザの指示がある場合は，指示に基づいてスキーマコンポーネントへマッピングします。

スキーマ定義ファイル
キャッシュの対象とするスキーマ文書や，キャッシュの格納場所を指定するファイルです。

スキーマ文書
XML Schema の仕様に従って記述された，XML 文書内の論理的なデータ構造を規定した XML 文書です。

（タ行）

ディスクキャッシュ
ディスク上に保存されたスキーマキャッシュです。

データ要素
XML 文書の構成要素となるデータです。タグに挟まれた文字列やタグ属性の値を指します。

トランスレット
XSLTC トランスフォーマを構成するコンパイラが XSLT スタイルシートを解析してメモリ上に生成する，Java のバイトコードです。XML 文書の変換処理を高速に実行できます。

（ハ行）

バインディング

JAXB によって行われるスキーマ文書で定義された XML 文書と，Java オブジェクトの間のデータ対応付けを指します。

（マ行）

マーシャル

JAXB API を使用して Java のオブジェクトを XML 文書に保存することをマーシャルといいます。Java のオブジェクトを XML 文書にシリアライズすることです。

マッピング

JAXB によって行われるバインディングと同じ意味です。Java クラスからスキーマ文書への対応付けの場合,「バインディング」ではなく「マッピング」と呼ぶ場合があります。

メモリキャッシュ

メモリ上に保存されたスキーマキャッシュです。

索引

C

D

G

I

J

M

N

O

P

S

T

U

W

X

あ

い

え

か

き

こ

さ

し

す

せ

た

ち

て

と

な

は

ひ

ふ

ま

め