第7回　衛星データ処理勉強会
XMLの基礎知識と適用事例の紹介

株式会社セック
開発本部　第四開発部
小林佑介・豊田光弘

本日のアジェンダ

- XMLとは？
- XMLの文法
- 名前空間
- XMLのスキーマ定義
  - DTD
  - XML Schema
  - RELAX NG
- XMLの関連技術
  - DOM/SAX
  - XSLT
  - 関連技術の応用
- XMLの適用事例
  - Tsunagi
  - UMS
- まとめ

XMLとは？

XMLとは？

- XML = Extensible Markup Language
  （拡張可能なマーク付け言語）
- 汎用的なデータ記述言語。
- 分野や用途に拠らず、幅広い種類のデータを記述可能。

XMLの利用のされ方

- データ交換や表現のための標準データフォーマットとして。
- システムやアプリケーション同士でデータ交換を行うための通信プロトコルを記述するためのフォーマットとして。
- システムやアプリケーションの振る舞いを定めるための設定情報を記述するファイルとして。

XMLの記述例

```
<?xml version="1.0" encoding="EUC-JP"?>
<図書情報>
  <ISBN>4-9999-9999-9</ISBN>
  <タイトル>XMLの基礎と適用事例</タイトル>
  <著者>abc</著者>
  <監修>def</監修>
  <発行情報>
    <発行 版数="初版" 刷数="第1刷"
          発行日="2006/01/20"/>
    <発行 版数="初版" 刷数="第2刷"
          発行日="2006/10/31"/>
  </発行情報>
  <発行者>ghi</発行者>
  <発行所>株式会社ABC出版</発行所>
  <印刷所>XYZ印刷株式会社</印刷所>
  <定価>2,200円</定価>
</図書情報>
```

図書情報のXML表現例

XMLの基礎と適用事例

発行日	2006年1月20日	第1版	第1刷
	2006年10月31日	第1版	第2刷
著者	abc		
監修	def		
発行者	ghi		
発行所	株式会社ABC出版		
印刷所	XYZ印刷株式会社		

ISBN4-9999-9999-9

定価 （本体2,200円＋税）

XMLの特徴

- テキスト形式。
- 独自のマーク付けが可能。
- 構造化されたデータ記述が可能。
- スキーマによる標準化が可能。

XMLの特徴（1）
テキスト形式

- 一般のテキストエディタで編集が可能。
- 異なるプラットフォームのコンピュータが混在するインターネット上でのデータ交換性に優れる。

XMLの特徴（2）
独自のマーク付け（マークアップ）

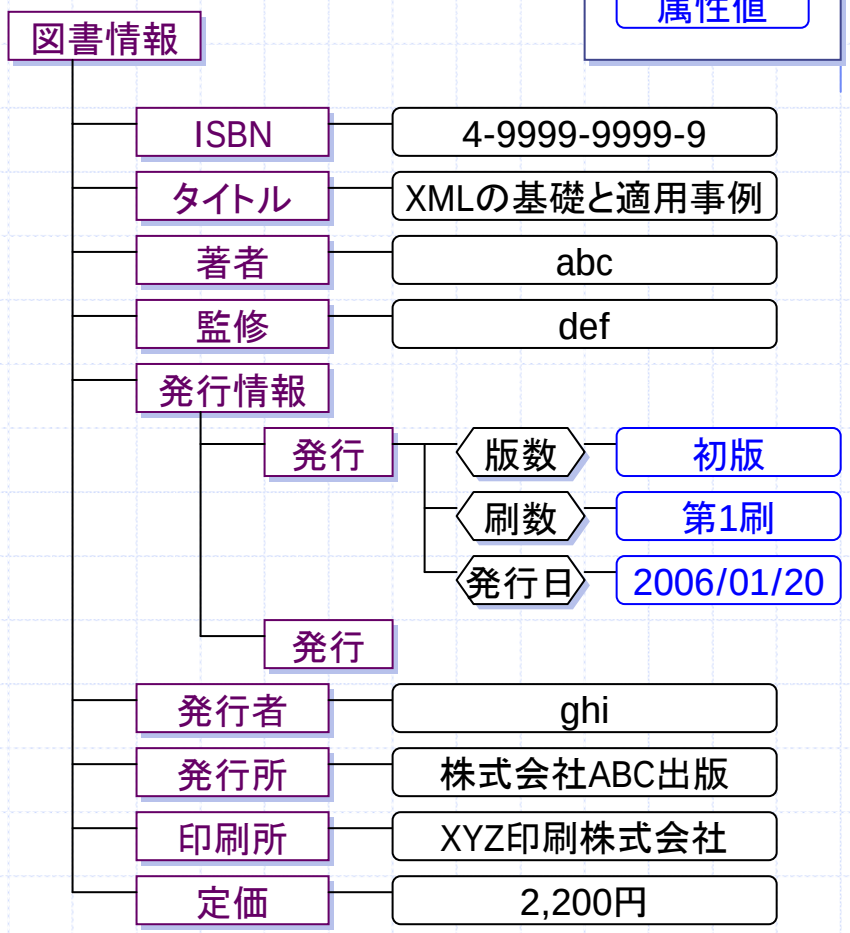
- 「タグ」と呼ばれるマークでデータを区切って記述。
- 「タグ」は、そのデータが何であるかを表し、またそのデータに関連する情報を含む。
- XMLデータは、データでありながら、データ自身を説明する情報を含んでいる。
  →「自己説明型のデータ」（self-describing data）

XMLの特徴（３）
構造化されたデータ記述

- 「タグ」を入れ子に記述することで、ある情報と、それに従属する情報とを階層的に表現することができる。
  →木構造のデータモデルを表す。
- 階層構造や繰り返し構造が出現するような、複雑なデータの記述が可能。

XMLデータの構造

```
<?xml version="1.0" encoding="EUC-JP"?>
<図書情報>
  <ISBN>4-9999-9999-9</ISBN>
  <タイトル>XMLの基礎と適用事例</タイトル>
  <著者>abc</著者>
  <監修>def</監修>
  <発行情報>
    <発行 版数="初版" 刷数="第1刷"
          発行日="2006/01/20"/>
    <発行 版数="初版" 刷数="第2刷"
          発行日="2006/10/31"/>
  </発行情報>
  <発行者>ghi</発行者>
  <発行所>株式会社ABC出版</発行所>
  <印刷所>XYZ印刷株式会社</印刷所>
  <定価>2,200円</定価>
</図書情報>
```



XMLの特徴（4）
スキーマによる標準化

- 「スキーマ」により、XMLデータで使用する「タグ」のセット（「タグ」の書き方の決まり、ルール）を定義することが可能。
- 特定分野において共通の「スキーマ」を利用することでデータの標準化が進む。
- XMLは、「スキーマ」の定義により、新たなデータ記述言語を定義することができる。
  すなわち「言語を定義する言語」である。
  →「メタ言語」（meta-language）

XMLによる標準化動向の例

eコマース	ebXML, BizTalk, cXML
金融	IFX, XBRL
マルチメディア・放送	SMIL, DIG, SVG, WAP/WML, BML
出版・メディア	EBook, JepaX, DocBook, NewsML
音声処理	VoiceXML
旅行	OTA, TravelXML
科学	MathML, MatML, CML, GEML, GML
天文	ADQL, VOTable

XMLの歴史

1986年	10月	SGMLがISO規格に
1996年	7月	W3C作業部会（WG）活動開始
	11月	XML草案（第1版）公開
1998年	2月	XML 1.0、W3C勧告に
	5月	XML 1.0、JIS（日本工業規格）に
1999年	1月	名前空間、W3C勧告に
2000年	10月	XML 1.0 第2版、W3C勧告に

XMLの関連規格

名前空間	ー	名前空間の指定
スキーマ定義	XML Schema	タグやデータ型等の定義
	RELAX NG	タグやデータ型等の定義
位置指定	XPath	XMLデータ中の特定位置指定
構造変換	XSLT	XMLデータを別の構造に変換
プログラミング	DOM	オブジェクトツリー型のプログラムI/F
	SAX	イベント駆動型のプログラムI/F
照会	XQuery	XMLデータへの問い合わせ
リンク	XPointer	リンク対象の位置指定
	XLink	リンク対象の結び付け

XMLは万能？

- XMLのメリット
  - データ自身に「タグ」としてデータを説明する情報が付加されており、一見してそれが何を表しているかがわかる。
  - 階層構造を持つような複雑なデータを表現できる。
  - データの追加／変更を比較的容易に行うことができる。
- XMLのデメリット
  - 「タグ」を付与する分、全体として冗長になり、データ量が大きくなる。
  - データが複雑になると、特に慣れない人にとっては視認性が悪くなり、編集も困難になる。（テキストとは言え「人に易しい」とは言えない）

他のデータ形式との比較

CSV

```
00001,節句太郎,1234
```

- シンプルな表構造で、昔から使われているフォーマットのため馴染み易い。
- 表形式では表せないような複雑なデータを表現することは難しい。
- データ自身を見ただけではそれが何を表すかがわからない。
- 先頭からの順番でデータ項目にアクセスするため、列の変更が困難。

XML

```
<社員>
  <社員番号>00001</社員番号>
  <名前>節句太郎</名前>
  <内線>1234</内線>
</社員>
```

- データ自身に「タグ」としてデータを説明する情報が付加されており、それが何を表しているかがわかる。
- 「タグ」の分データ量が大きくなる。
- 階層構造を持つような複雑なデータを表現できる。
- データの追加／変更が容易。
- データが複雑になると視認性が悪くなり、編集も困難になる。

XMLの文法

XMLの基本構成

```
<?xml version="1.0" encoding="EUC-JP"?>
<satelliteList>
  <satellite name="AKARI" code="ASTRO-F">
    <description>
      赤外線観測衛星
    </description>
    <instrument name="FIS"/>
    <instrument name="IRC"/>
  </satellite>
  <satellite name="HINODE" code="SOLAR-B">
    <description>
      太陽観測衛星
    </description>
    <instrument name="SOT"/>
    <instrument name="XRT"/>
    <instrument name="EIS"/>
  </satellite>
</satelliteList>
```

衛星情報のXML表現例

ASTRO-F

名前	AKARI
詳細情報	赤外線観測衛星
観測装置	FIS, IRC

SOLAR-B

名前	HINODE
詳細情報	太陽観測衛星
観測装置	SOT, XRT, EIS

XMLの基本構成

```
<?xml version="1.0" encoding="EUC-JP"?>
<satelliteList>
  <satellite name="AKARI" code="ASTRO-F">
    <description>
      赤外線観測衛星
    </description>
    <instrument name="FIS"/>
    <instrument name="IRC"/>
  </satellite>
  <satellite name="HINODE" code="SOLAR-B">
    <description>
      太陽観測衛星
    </description>
    <instrument name="SOT"/>
    <instrument name="XRT"/>
    <instrument name="EIS"/>
  </satellite>
</satelliteList>
```

XML宣言

XMLインスタンス
（XMLデータ本体）

XML宣言

```
<?xml version="1.0" encoding="EUC-JP"?>
<satelliteList>
  <satellite name="AKARI" code="ASTRO-F">
    <description>
      赤外線観測衛星
    </description>
    <instrument name="FIS"/>
    <instrument name="IRC"/>
  </satellite>
  <satellite name="HINODE" code="SOLAR-B">
    <description>
      太陽観測衛星
    </description>
    <instrument name="SOT"/>
    <instrument name="XRT"/>
    <instrument name="EIS"/>
  </satellite>
</satelliteList>
```

- XML文書であることを明示する。
- バージョンの宣言、使用している文字コードの宣言を行う。

XML宣言の記述方法

<?xml version=“バージョン番号” encoding=“文字コード名” ?>

- バージョンの宣言
  - バージョン番号は、通常は1.0を指定すれば良い。
- 文字コードの宣言
  - 符号化宣言（encoding declaration）と呼ぶ。
  - XMLデータの記述に使用している文字コード名を指定する。
    （省略した場合はUTF-8またはUTF-16とみなされる）

XML宣言の記述方法

<?xml version=“バージョン番号” encoding=“文字コード名” ?>

※ 指定可能な文字コードの例

- UTF-8、UTF-16　：ISO / SEC 10646関連文字コード
- ISO-8859-n　：ISO 8859関連文字コード
- Shift_JIS、EUC-JP　：日本語関連文字コード

※ 指定された文字コードと、実際に使用されている文字コードとが一致しない場合、多くのツールでXMLデータ処理時にエラーと判断されてしまう。

XMLインスタンス

```
<?xml version="1.0" encoding="EUC-JP"?>
<satelliteList>
  <satellite name="AKARI" code="ASTRO-F">
    <description>
      赤外線観測衛星
    </description>
    <instrument name="FIS"/>
    <instrument name="IRC"/>
  </satellite>
  <satellite name="HINODE" code="SOLAR-B">
    <description>
      太陽観測衛星
    </description>
    <instrument name="SOT"/>
    <instrument name="XRT"/>
    <instrument name="EIS"/>
  </satellite>
</satelliteList>
```

- XMLデータの本体。
- 実際のXMLデータが記述される。
- タグを用いてデータが階層的に表現される。

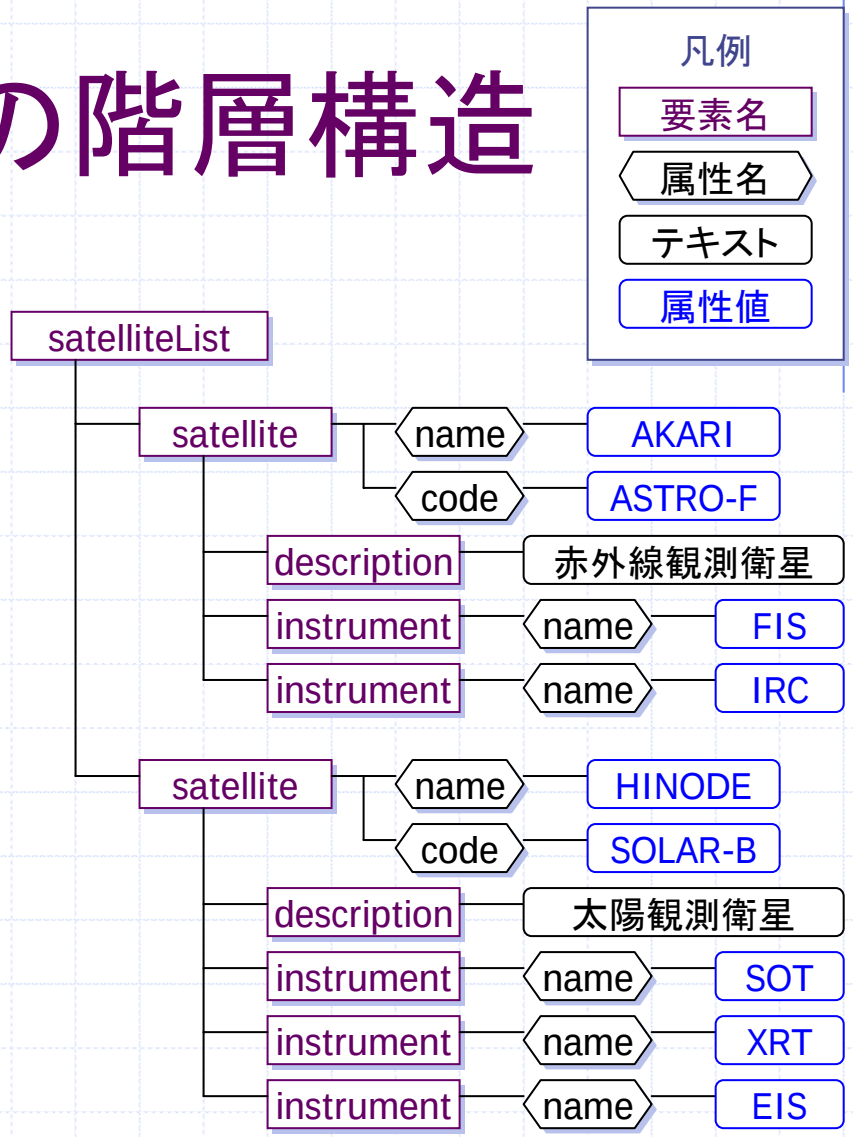
XMLインスタンス

```
<?xml version="1.0" encoding="EUC-JP"?>
<satelliteList>
  <satellite name="AKARI" code="ASTRO-F">
    <description>
      赤外線観測衛星
    </description>
    <instrument name="FIS"/>
    <instrument name="IRC"/>
  </satellite>
  <satellite name="HINODE" code="SOLAR-B">
    <description>
      太陽観測衛星
    </description>
    <instrument name="SOT"/>
    <instrument name="XRT"/>
    <instrument name="EIS"/>
  </satellite>
</satelliteList>
```

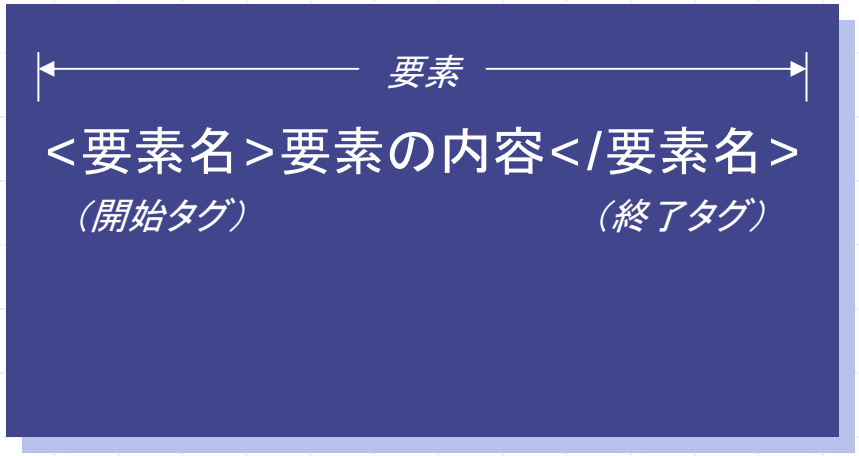
- XMLインスタンスの主な構成要素
  - 要素　satelliteList
satellite
description
instrument
  - 属性　name
code
  - コメント

XMLインスタンスの階層構造

```
<?xml version="1.0" encoding="EUC-JP"?>
<satelliteList>
  <satellite name="AKARI" code="ASTRO-F">
    <description>
      赤外線観測衛星
    </description>
    <instrument name="FIS"/>
    <instrument name="IRC"/>
  </satellite>
  <satellite name="HINODE" code="SOLAR-B">
    <description>
      太陽観測衛星
    </description>
    <instrument name="SOT"/>
    <instrument name="XRT"/>
    <instrument name="EIS"/>
  </satellite>
</satelliteList>
```

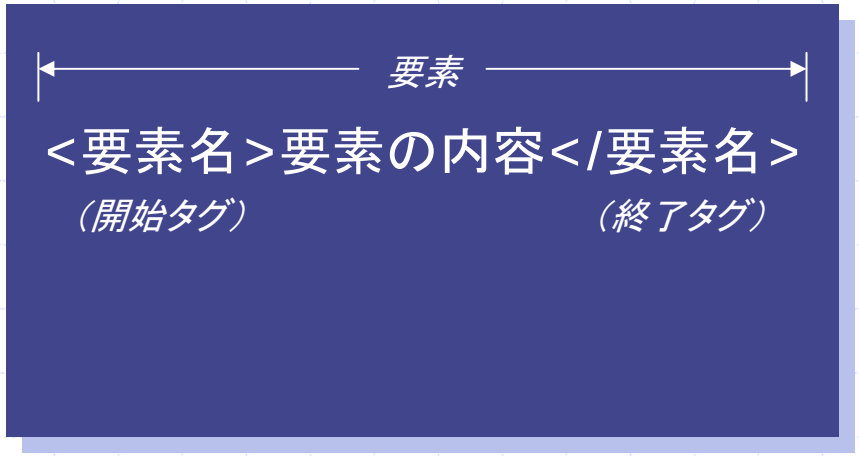


要素の記述方法



- 要素（element）は、XML上で表現したいデータの単位を表す。
- 要素の開始は開始タグ（start tag）で表す。
- 要素の終了は終了タグ（end tag）で表す。

要素の記述方法

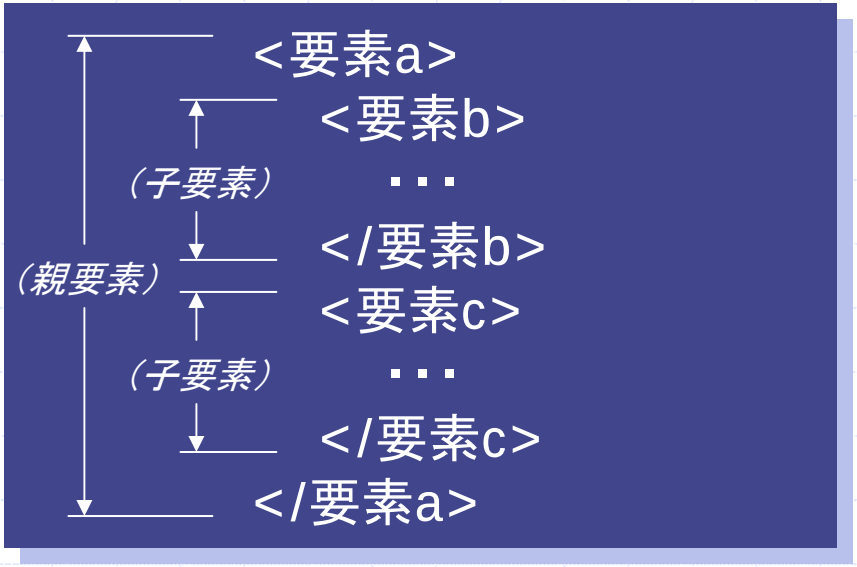


- 開始／終了タグで挟まれた部分を要素の内容（content）と言う。

※ 要素の内容に含められるもの

- テキストデータ
- 別の要素（子要素）
- テキストデータと別の要素（両方）

要素の階層構造の記述方法



- 別の要素を含める場合、要素を入れ子にして記述する。
- ある要素の中に含まれる要素を、それを含む要素の「子要素」と言う。子要素を含む要素を、その子要素の「親要素」と言う。

誤った階層構造の記述

誤りの例

```
<p>XMLは<b>汎用的なデータ記述言語</p></b>
```

正しい例

```
<p>XMLは<b>汎用的なデータ記述言語</b></p>
```

- 要素は正しく入れ子関係にして記述する。

ルート要素

```
<?xml version="1.0" encoding="EUC-JP"?>
<satelliteList>
  <satellite name="AKARI" code="ASTRO-F">
    <description>
      赤外線観測衛星
    </description>
    <instrument name="FIS"/>
    <instrument name="IRC"/>
  </satellite>
  <satellite name="HINODE" code="SOLAR-B">
    <description>
      太陽観測衛星
    </description>
    <instrument name="SOT"/>
    <instrument name="XRT"/>
    <instrument name="EIS"/>
  </satellite>
</satelliteList>
```

- XMLインスタンス中の最上位の要素をルート要素と言う。
- ルート要素はXMLインスタンス中でひとつだけ登場する。
  →複数登場することは許されない。

属性の記述方法

```
<要素名
   属性名1=“属性値1”
   属性名2=“属性値2” … >
  …
</要素名>
```

- 属性（attribute）は、要素に何らかの付属情報を付け加えるもの。
- 属性は開始タグ中に指定する。

属性の記述方法

```
<要素名
   属性名1=“属性値1”
   属性名2=“属性値2” … >
  …
</要素名>
```

- ひとつの開始タグ中には、複数の属性を指定することができる。
- ひとつの開始タグ中に同名の属性を複数定義することは許されない。

属性の記述方法

```
<doc desc="XMLは'汎用的なデータ記述言語'">
  …
</doc>
```

```
<doc desc='XMLは"汎用的なデータ記述言語"'>
  …
</doc>
```

- 属性値は、""（二重引用符）あるいは ''（アポストロフィ）で囲む。
- 属性値自身に "" を含める場合は、'' で囲む。逆に、属性値に '' を含める場合は、"" を使用する。

空要素の記述方法

<要素名 属性名=“属性値” … />

（※ 属性の指定は不要な場合は省略できる）

- 要素が内容を持たない場合は、簡略化して記述できる。

要素名や属性名の制約

- 要素名や属性名の先頭に指定できる文字
  - アルファベット
  - ひらがな
  - カタカナ
  - 漢字
  - アンダースコア (_)
  - コロン (:)

※「xml」と言う文字列は（大文字小文字のどの組合せでも）先頭に使用できない。

※ コロン (:) は、名前空間を示すときに使用する。

- 要素名や属性名の2文字目以降に指定できる文字
  - アルファベット
  - ひらがな、カタカナ
  - 漢字
  - 数字
  - アクセント記号等
  - 「々」「ー」等
  - アンダースコア (_)
  - コロン (:)
  - ピリオド (.)
  - ハイフン (-)

コメントの記述方法

```
<!-- コメント -->
<!--
<a><b>...</b></a>
-->
```

- コメントはXMLインスタンス中の任意の場所に記述可能。
- タグ付けされたデータをコメント中に記述することもできる。

整形式XML文書

- XMLとしての条件を正しく満たすXMLデータを、整形式XML文書（well-formed XML document）と言う。
  - 開始タグと終了タグとが対応していること。
  - 要素の入れ子関係が正しく対応していること。
  - ルート要素（最上位の要素）が複数存在しないこと。
  - etc.
- すなわち、「XMLとして正しいこと」を表す。

整形式XML文書の確認

- WebブラウザでXMLデータが保存されたファイルを開く。
  - 整形式（well-formed）であれば、XMLデータの内容がブラウザ上に表示される。
  - 整形式でなければエラーメッセージが表示され、XMLデータの内容は表示されない。
- 既存の専用エディタを使用して整形式であるかどうかを確かめる。
  - XMLSPY
  - Emacs + nxml-mode　　etc.

名前空間

名前空間とは？

- XMLでは、自由にタグのセット（タグの書き方の決まり、ルール）を定義することができる。
- 名前空間は、XMLデータ中に現れる要素や属性が、どのタグのセットに属するかを識別するための仕組み、考え方である。

名前の衝突

```
<?xml version="1.0" encoding="EUC-JP"?>
<launchInfo date="2006/02/xx" >
  <rocket>
    <name>M-V</name>
  </rocket>
  <satellite>
    <code>ASTRO-F</code>
    <name>AKARI</name>
    <instrument name="FIS"/>
    <instrument name="IRC"/>
  </satellite>
</launchInfo>
```

- 異なるタグのセット同士で同じ名前のタグを使用していた場合、名前の衝突が起こる。
  →そのタグが、どのタグのセットに属するのかを識別することができない。

名前空間による衝突の回避

```
<?xml version="1.0" encoding="EUC-JP"?>
<li:launchInfo date="2006/02/xx"
   xmlns:li="http://plain.isas.jaxa.jp/launchInfo">
  <rkt:rocket
     xmlns:rkt="http://plain.isas.jaxa.jp/rocket">
    <rkt:name>M-V</rkt:name>
  </rkt:rocket>
  <sat:satellite
     xmlns:sat="http://plain.isas.jaxa.jp/satellite">
    <sat:code>ASTRO-F</sat:code>
    <sat:name>AKARI</sat:name>
    <sat:instrument name="FIS"/>
    <sat:instrument name="IRC"/>
  </sat:satellite>
</li:launchInfo>
```

- 要素や属性に、それがどのタグのセットに属するかを表す文字列を付与する。
→衝突を回避する。

名前空間の記述方法

xmlns:名前空間接頭辞="URIで表された名前空間名"

- 名前空間宣言は開始タグ中に記述する。
- 名前空間はURIで表される。
  →世界で唯一に決まるようにするため。
- URIは存在している必要はない。
  その名前空間を識別さえできれば良い。

名前空間の記述方法

xmlns:名前空間接頭辞="URIで表された名前空間名"

- 名前空間を表すURIの代わりに別の文字列(＝名前空間接頭辞)を割り当てる。
- この文字列を要素名や属性名に指定することで、その要素や属性が名前空間に属することを表す。

名前空間の有効範囲

```
<?xml version="1.0" encoding="EUC-JP"?>
<li:launchInfo date="2006/02/xx"
   xmlns:li="http://plain.isas.jaxa.jp/launchInfo">
  <rkt:rocket
     xmlns:rkt="http://plain.isas.jaxa.jp/rocket">
    <rkt:name>M-V</rkt:name>
  </rkt:rocket>
  <sat:satellite
     xmlns:sat="http://plain.isas.jaxa.jp/satellite">
    <sat:code>ASTRO-F</sat:code>
    <sat:name>AKARI</sat:name>
    <sat:instrument name="FIS"/>
    <sat:instrument name="IRC"/>
  </sat:satellite>
</li:launchInfo>
```

- 名前空間宣言は、接頭辞が付与されている要素あるいはそれより上位の要素で指定する。

デフォルトの名前空間

xmlns="URIで表された名前空間名"

- 指定された要素を含め、その下位の要素で、名前空間接頭辞で修飾されていない要素は、このURIに属するものとみなされる。

デフォルトの名前空間

xmlns="URIで表された名前空間名"

```
<satellite
    xmlns="http://plain.isas.jaxa.jp/satellite">
  <code>ASTRO-F</code>
  <name>AKARI</name>
  <instrument name="FIS"/>
  <instrument name="IRC"/>
</satellite>
```

- 名前空間接頭辞を省略した記法。
- 指定された要素を含め、その下位の要素で、名前空間接頭辞で修飾されていない要素は、このURIに属するものとみなされる。

グローバル属性

```
<sat:satellite
    xmlns:sat="http://plain.isas.jaxa.jp/satellite">
    xmlns:html="http://www.w3c.org/1999/xhtml1">
  <sat:code html:class="red">ASTRO-F</sat:code>
  <sat:name html:class="blue">AKARI</sat:name>
  <sat:instrument name="FIS"/>
  <sat:instrument name="IRC"/>
</sat:satellite>
```

- 通常、属性は特定の要素に属する。
- どの要素にも指定できる属性のことを「グローバル属性」と言う。
- 「グローバル属性」には、それが属する名前空間を表す名前空間接頭辞を付与する。

XMLのスキーマ定義

スキーマとは？

- XMLデータで使用する
  - 要素名
  - 属性名
  - 要素の階層構造
  - 要素や属性のデータ型

  等を定義したもの。

スキーマのメリット

- XMLは作成者が独自の構造を定義することができる。
  →皆が自分勝手に構造を決めていてはデータの交換は進まない。
- 同一目的の対象に対してスキーマを決めてやることで、データ構造が標準化される。
  データ交換を行うことが容易になる。
- 同じスキーマを解釈するアプリケーションやシステム同士で、データ交換を行うことが容易になる。

スキーマの種類

- DTD
- XMLスキーマ
- RELAX NG

DTD

- DTD = Document Type Declaration
  （文書型宣言）
- XML 1.0により規定。
- XMLではない独自の記法に従って記述する。

DTDの記述例

```
<!ELEMENT satelliteList (satellite+) >
<!ELEMENT satellite (description?, instrument*) >
<!ELEMENT description (#PCDATA) >
<!ELEMENT instrument EMPTY >
<!ATTLIST satellite
                code CDATA #REQUIRED
                name CDATA #IMPLIED>
<!ATTLIST instrument
                name CDATA #REQUIRED>
```

```
<?xml version="1.0" encoding="EUC-JP"?>
<satelliteList>
  <satellite name="AKARI" code="ASTRO-F">
    <description>
      赤外線観測衛星
    </description>
    <instrument name="FIS"/>
    <instrument name="IRC"/>
  </satellite>
  <satellite name="HINODE" code="SOLAR-B">
    <description>
      太陽観測衛星
    </description>
    <instrument name="SOT"/>
    <instrument name="XRT"/>
    <instrument name="EIS"/>
  </satellite>
</satelliteList>
```

↑ *XMLデータ*
← *XMLデータの構造を表すDTD*

要素型宣言

```
<!ELEMENT satelliteList (satellite+) >
<!ELEMENT satellite (description?, instrument*) >
<!ELEMENT description (#PCDATA) >
<!ELEMENT instrument EMPTY >
<!ATTLIST satellite
                code CDATA #REQUIRED
                name CDATA #IMPLIED>
<!ATTLIST instrument
                name CDATA #REQUIRED>
```

- <!ELEMENT ... >で、要素とその子要素を定義する。

属性リスト宣言

```
<!ELEMENT satelliteList (satellite+) >
<!ELEMENT satellite (description?, instrument*) >
<!ELEMENT description (#PCDATA) >
<!ELEMENT instrument EMPTY >
<!ATTLIST satellite
                code CDATA #REQUIRED
                name CDATA #IMPLIED>
<!ATTLIST instrument
                name CDATA #REQUIRED>
```

- <!ATTLIST … >で、要素に付属する属性を定義する。

XML Schema

- 現在主流のスキーマ言語。
- XMLの関連規格として規定されている。
- データ型や名前空間等、DTDでサポートされない機能を備えている。
- スキーマ自身をXML形式で記述する。
- DTDや他のスキーマ記述言語に比べてやや複雑。

XML Schemaの記述例

```
<?xml version=“1.0” encoding=“EUC-JP” ?>
<xs:schema
   xmlns:xs=“http://www.w3c.org/2001/XMLSchema>
  <xs:element name=“satelliteList”
     type=“satelliteListType”/>
  <xs:complexType name=“satelliteListType”>
    <xs:sequence>
      <xs:element ref=“satellite”
         minOccrus=“1” maxOccurs=“unbounded”/>
    </xs:sequence>
  </complexType>
  <xs:element name=“satellite” type=“satelliteType”/>
  <xs:complexType type=“satelliteType”>
    <xs:sequence>
      <xs:element ref=“description”
         minOccrus=“0” maxOccurs=“1”/>
      <xs:element ref=“instrument”
         minOccrus=“0” maxOccurs=“unbounded”/>
    </xs:sequence>
    <xs:attribute name=“name” type=“xs:string”/>
    <xs:attribute name=“code” type=“xs:string”/>
  </complexType>
  <xs:element name=“description” type=“xs:string”/>
     :    :    :
```

```
<?xml version="1.0" encoding="EUC-JP"?>
<satelliteList>
  <satellite name="AKARI" code="ASTRO-F">
    <description>
      赤外線観測衛星
    </description>
    <instrument name="FIS"/>
    <instrument name="IRC"/>
  </satellite>
  <satellite name="HINODE" code="SOLAR-B">
    <description>
      太陽観測衛星
    </description>
    <instrument name="SOT"/>
    <instrument name="XRT"/>
    <instrument name="EIS"/>
  </satellite>
</satelliteList>
```

↑ *XMLデータ*

← *XMLデータの構造を表すXML Schema*

要素宣言

```
<?xml version=“1.0” encoding=“EUC-JP” ?>
<xs:schema
   xmlns:xs=“http://www.w3c.org/2001/XMLSchema>
  <xs:element name=“satelliteList”
     type=“satelliteListType”/>
  <xs:complexType name=“satelliteListType”>
    <xs:sequence>
      <xs:element ref=“satellite”
         minOccrus=“1” maxOccurs=“unbounded”/>
    </xs:sequence>
  </complexType>
  <xs:element name=“satellite” type=“satelliteType”/>
  <xs:complexType type=“satelliteType”>
    <xs:sequence>
      <xs:element ref=“description”
         minOccrus=“0” maxOccurs=“1”/>
      <xs:element ref=“instrument”
         minOccrus=“0” maxOccurs=“unbounded”/>
    </xs:sequence>
    <xs:attribute name=“name” type=“xs:string”/>
    <xs:attribute name=“code” type=“xs:string”/>
  </complexType>
  <xs:element name=“description” type=“xs:string”/>
      :    :    :
```

- <xs:element>要素にて、要素とその子要素や属性を定義する。

属性宣言

```
<?xml version=“1.0” encoding=“EUC-JP” ?>
<xs:schema
   xmlns:xs=“http://www.w3c.org/2001/XMLSchema>
  <xs:element name=“satelliteList”
     type=“satelliteListType”/>
  <xs:complexType name=“satelliteListType”>
    <xs:sequence>
      <xs:element ref=“satellite”
         minOccrus=“1” maxOccurs=“unbounded”/>
    </xs:sequence>
  </complexType>
  <xs:element name=“satellite” type=“satelliteType”/>
  <xs:complexType type=“satelliteType”>
    <xs:sequence>
      <xs:element ref=“description”
         minOccrus=“0” maxOccurs=“1”/>
      <xs:element ref=“instrument”
         minOccrus=“0” maxOccurs=“unbounded”/>
    </xs:sequence>
    <xs:attribute name=“name” type=“xs:string”/>
    <xs:attribute name=“code” type=“xs:string”/>
  </complexType>
  <xs:element name=“description” type=“xs:string”/>
     :    :    :
```

- <xs:attribute>要素にて、要素に付属する属性を定義する。

型定義

```
<?xml version=“1.0” encoding=“EUC-JP” ?>
<xs:schema
   xmlns:xs=“http://www.w3c.org/2001/XMLSchema>
  <xs:element name=“satelliteList”
     type=“satelliteListType”/>
  <xs:complexType name=“satelliteListType”>
    <xs:sequence>
      <xs:element ref=“satellite”
         minOccrus=“1” maxOccurs=“unbounded”/>
    </xs:sequence>
  </complexType>
  <xs:element name=“satellite” type=“satelliteType”/>
  <xs:complexType type=“satelliteType”>
    <xs:sequence>
      <xs:element ref=“description”
         minOccrus=“0” maxOccurs=“1”/>
      <xs:element ref=“instrument”
         minOccrus=“0” maxOccurs=“unbounded”/>
    </xs:sequence>
    <xs:attribute name=“name” type=“xs:string”/>
    <xs:attribute name=“code” type=“xs:string”/>
  </complexType>
  <xs:element name=“description” type=“xs:string”/>
     :    :    :
```

- <xs:complexType>要素にて、要素の型（パターン）を定義。
- XMLデータ内で同じようなパターンが出現する場合に、定義された型を参照して要素の宣言を行うことが可能。（型の共通化、再利用）

RELAX NG

- RELAX =
  REgular LAnguage description for XML
- 日本発のスキーマ記述言語。
- XML Schemaの表現力を維持しつつ、複雑性を排除し、簡単にスキーマを記述できることを目指している。
- XML Schema同様、スキーマ自身をXML形式で記述する。

RELAX NGのメリット

- XMLインスタンスに近い形でスキーマを定義することができるため、直感的に理解し易い。
- Emacs + nxml-mode
  を使用し、リアルタイムにRELAX NGスキーマとの検証（※）を行いながら編集することができ、誤りの早期発見に役立つ。

（※）正確には、Emacs + nxml-mode では、RELAX NGの簡略化表記である
「RELAX NG Compact Syntax」との検証を行う。
「RELAX NG Compact Syntax」は、ツールによりRELAX NGを変換して得ることができる。

RELAX NGの記述例

```
<?xml version=“1.0” encoding=“EUC-JP” ?>
<grammar xmlns=“http://relaxng.org/ns/structure/1.0”
 datatypeLibrary="http://www.w3.org/...">
 <start>
  <element name=“satelliteList”>
   <element name=“satellite”>
    <optional>
     <attribute name=“name”><data type=“string”/>
     </attribute>
    </optional>
    <attribute name=“code”><data type=“string”/>
    </attribute>
    <optional>
     <element name=“description”><text/></element>
    </optional>
    <zeroOrMore>
     <element name=“instrument”>
      <attribute name=“name”><data type=“string”/>
      </attribute>
      <empty/>
     </element>
    </zeroOrMore>
   </element>
  </element>
 </start>
</grammar>
```

```
<?xml version="1.0" encoding="EUC-JP"?>
<satelliteList>
  <satellite name="AKARI" code="ASTRO-F">
    <description>
      赤外線観測衛星
    </description>
    <instrument name="FIS"/>
    <instrument name="IRC"/>
  </satellite>
  <satellite name="HINODE" code="SOLAR-B">
    <description>
      太陽観測衛星
    </description>
    <instrument name="SOT"/>
    <instrument name="XRT"/>
    <instrument name="EIS"/>
  </satellite>
</satelliteList>
```

↑ *XMLデータ*

← *XMLデータの構造を表すRELAX NG*

要素の定義

RELAX NG

```
<element name=“要素名1”>
  <element name=“要素名2”>
    ・・・
  </element>
</element>
```

XMLインスタンス

```
<要素名1>
  <要素名2>
    ・・・
  </要素名2>
</要素名1>
```

- <element>要素で、要素を定義する。name属性に要素名を記述する。
- 子要素、子孫要素を定義する場合は<element>要素を入れ子にする。

属性の定義

RELAX NG

```
<element name="要素名">
<attribute name="属性名1">…</attribute>
<attribute name="属性名2">…</attribute>
  ・・・
</element>
```

XMLインスタンス

```
<要素名 属性名1="・・・" 属性名2="・・・">
  ・・・
</要素名>
```

- <attribute>要素で、要素の属性を定義する。name属性に属性名を記述する。
- 要素の属性は、その要素を表す<element>要素の子要素として定義する。

テキストの定義

RELAX NG

```
<element name=“要素名”>
  <attribute name=“属性名”>
    <text/>
  </attribute>
  <text/>
</element>
```

XMLインスタンス

```
<要素名 属性名=“テキスト”>
  テキスト
</要素名>
```

- 要素や属性の中身をテキストとする場合は<text>要素を用いる。

データ型の定義

RELAX NG

```
<element name=“要素名”>
  <attribute name=“属性名”>
    <value>abc</value>
  </attribute>
  <data type=“integer”/>
</element>
```

XMLインスタンス

```
<要素名 属性名=“abc”>123</要素名>
```

- 要素や属性の中身に何らかのデータ型を定義する場合は、<data>要素を用いる。type属性にデータ型を記述する。
- 固定の値を指定する場合は<value>要素を用いる。

要素や属性の登場回数の指定

RELAX NG

```
<element name="要素名1">
  <optional>
    <element name="要素名2">
      ・・・
    </element>
  </optional>
</element>
```

XMLインスタンス

```
<要素名1>
  <!-- 要素名2 は登場しない/1回登場 -->
  <要素名2>・・・</要素名2>
</要素名1>
```

- 0回もしくは1回登場する要素または属性を<optional>要素で囲んで示す。

要素や属性の登場回数の指定

RELAX NG

```
<element name="要素名1">
  <oneOrMore>
    <element name="要素名2">
      ・・・
    </element>
  </oneOrMore>
</element>
```

XMLインスタンス

```
<要素名1>
  <!-- 要素名2 は1回以上登場 -->
  <要素名2>・・・</要素名2>
  <要素名2>・・・</要素名2>
</要素名1>
```

- 1回以上登場する要素または属性は<oneOrMore>要素で囲んで示す。

要素や属性の登場回数の指定

RELAX NG

```
<element name="要素名1">
  <zeroOrMore>
    <element name="要素名2">
      ・・・
    </element>
  </zeroOrMore>
</element>
```

XMLインスタンス

```
<要素名1>
  <!-- 要素名2 は0回以上登場 -->
  <要素名2>・・・</要素名2>
  <要素名2>・・・</要素名2>
</要素名1>
```

- 0回以上登場する要素または属性は<zeroOrMore>要素で囲んで示す。

要素や属性の選択の指定

RELAX NG

```
<element name="要素名1">
  <choice>
    <element name="要素名A"/>
    <element name="要素名B"/>
    <element name="要素名C"/>
  </choice>
</element>
```

XMLインスタンス

```
<要素名1>
  <!-- 要素名A、B、C のいずれかが登場 -->
  <要素名A>・・・</要素名A>
  <!-- <要素名B>・・・</要素名B> -->
  <!-- <要素名C>・・・</要素名C> -->
</要素名1>
```

- いくつかの選択肢のうち、いずれかひとつに適合すれば良い場合は、選択的に登場する要素や属性の定義を<choice>要素で囲んで示す。

順不同要素や属性の指定

RELAX NG

```
<element name="要素名1">
  <interleave>
    <element name="要素名A"/>
    <element name="要素名B"/>
    <element name="要素名C"/>
  </interleave>
</element>
```

XMLインスタンス

```
<要素名1>
  <!-- 要素名A、B、C が順不同で登場 -->
  <要素名B>・・・</要素名B>
  <要素名A>・・・</要素名A>
  <要素名C>・・・</要素名C>
</要素名1>
```

- 順番を問わず現れる要素や属性については、<interleave>要素で囲んで示す。

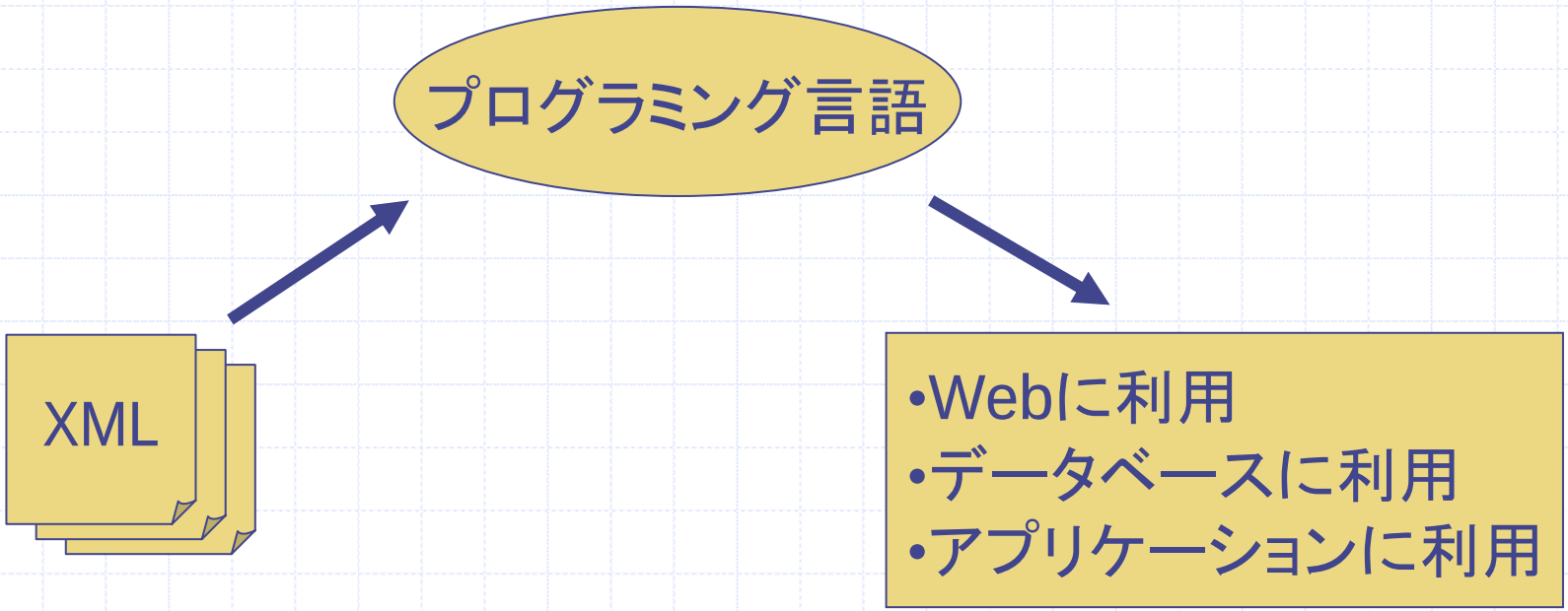
XMLの関連技術

XMLを使用したデータ処理

- XMLのデータ利用
  - XMLとWebの連携
  - XMLとデータベースの連携
  - アプリケーションの設定情報
- XMLの変換
  - XMLからXMLへの変換
  - XMLからHTMLへの変換
  - XMLからテキストへの変換

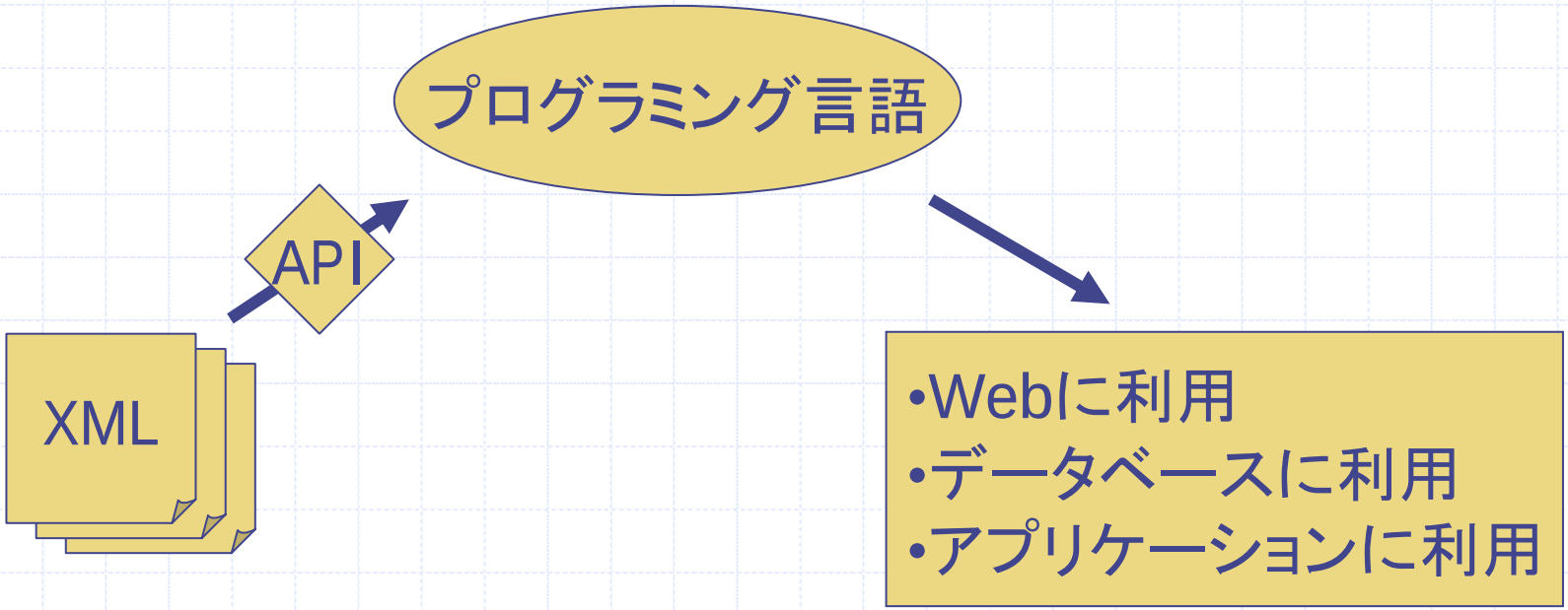
XMLのデータ利用

- プログラミング言語からXMLを利用する。
- XMLに含まれる情報を取り出す。



XMLのデータ利用

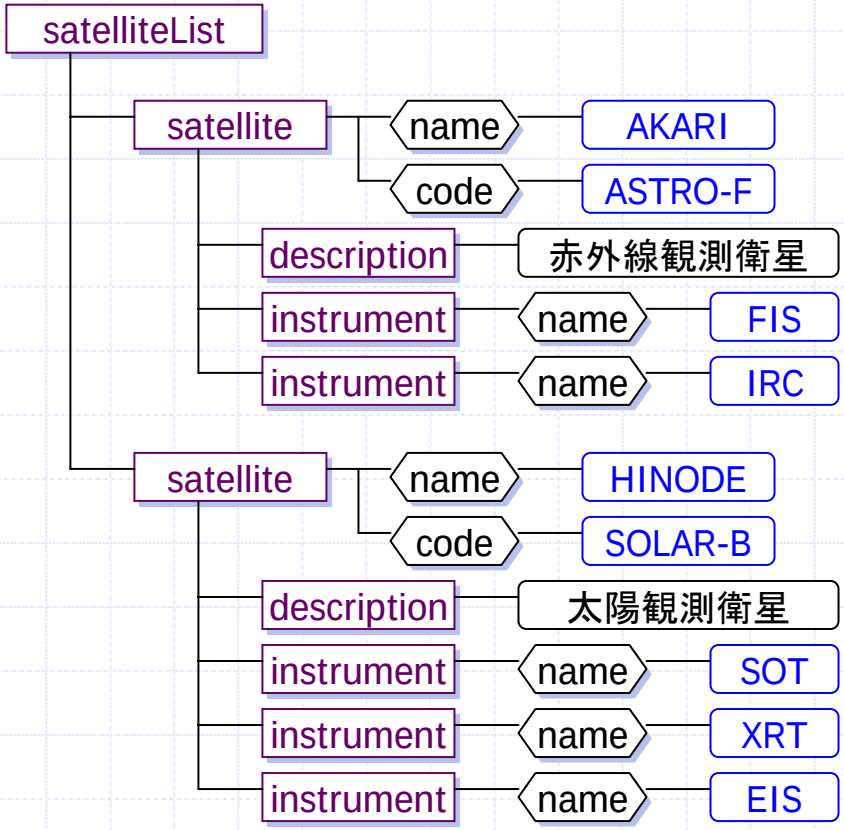
- プログラミング言語からXMLを扱うための API が用意されている。
- API = Application Programming Interface



XMLのデータ利用

- XMLを扱う代表的なAPI
  - DOM
  - SAX

DOMとは？



- DOM = Document Object Model
- XML全体を読み込み、ツリーを形成する。
- ノードの情報を自由に扱うことができる。

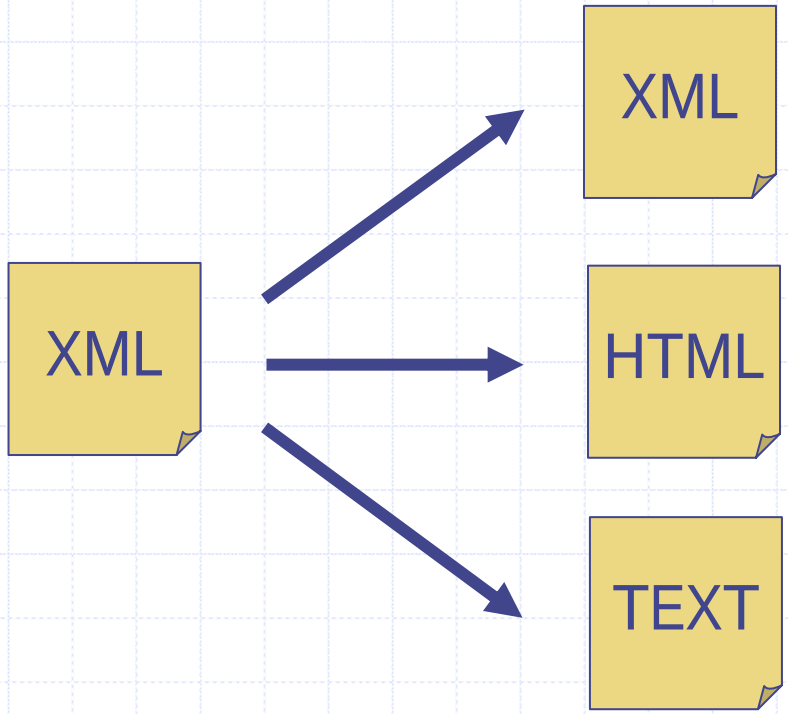
SAXとは？

```
<?xml version="1.0" encoding="EUC-JP"?>
<satelliteList>
  <satellite name="AKARI" code="ASTRO-F">
    <description>
      赤外線観測衛星
    </description>
    <instrument name="FIS"/>
    <instrument name="IRC"/>
  </satellite>
  <satellite name="HINODE" code="SOLAR-B">
    <description>
      太陽観測衛星
    </description>
    <instrument name="SOT"/>
    <instrument name="XRT"/>
    <instrument name="EIS"/>
  </satellite>
</satelliteList>
```

- SAX = Simple API for XML
- 先頭の要素から順にXMLを辿る。
- イベント駆動で操作を行う。

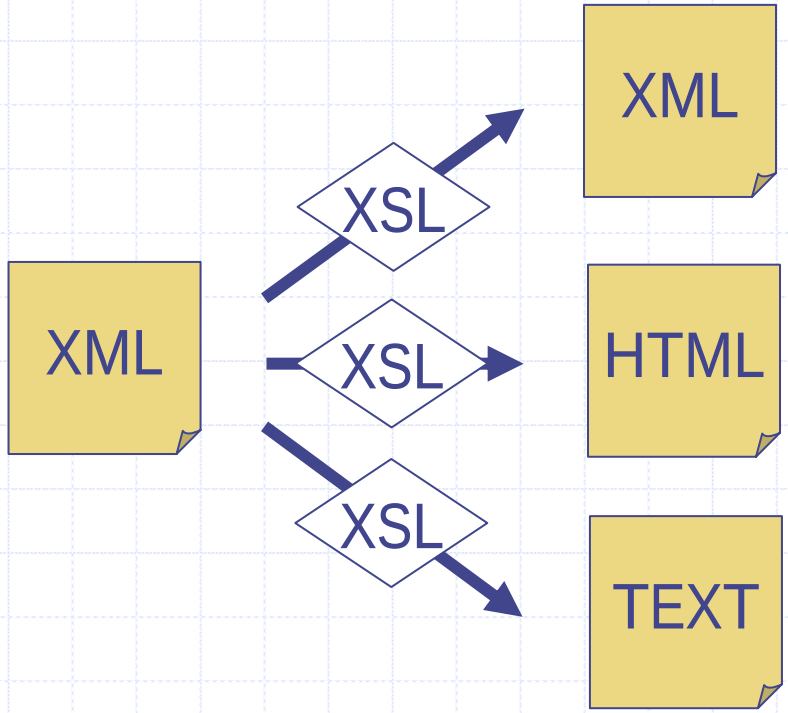
XMLの変換

- XMLを変換する。
  - XMLへ変換
  - HTMLへ変換
  - テキストへ変換



XMLの変換

- XMLの変換を行う言語
  - XSL



XSLとは？

- XSL = Extensible Stylesheet Language
- XMLに対するスタイル指定言語。
- XSLは以下で構成。
  - 構造変換指定：XSLT
  - フォーマッティング指定： XSL-FO

XSLTとは？

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

  <xsl:output method="xml" encoding="euc-jp"/>

  <xsl:template match="satelliteList">
    <xsl:element name="satelliteList">
      <xsl:for-each select="satellite">
        <xsl:element name="satellite">
          <xsl:attribute name="name">
          <xsl:value-of select="@name"/></xsl:attribute>
          <xsl:attribute name="description">
          <xsl:value-of select="description"/></xsl:attribute>
        </xsl:element>
      </xsl:for-each>
    </xsl:element>
  </xsl:template>
</xsl:stylesheet>
```

- XSLT = XSL Transformations
- XMLの構造変換を行う言語。
- XML形式で記述。

XSLTによる変換の実演

- XMLからXMLへ変換
- XMLからHTMLへ変換
- XMLからテキストへ変換

関連技術の応用

- ソフトウェアの試験フレームワーク

- 想定する試験
  - ある観測装置のソフトウェアの試験。
  - 観測装置にコマンドを送信し、動作を確認する。
  - コマンド送信用のスクリプトを作成し、スクリプトに記述された処理を順に実行する。

現状の試験作業の問題点

- 試験仕様書の体裁を整えるのに時間がかかる。
- 試験仕様書から試験スクリプトを作成するのに手間がかかり、ミスをすることもある。
- 試験仕様書の改版時に、試験スクリプトを修正する必要がある。

試験フレームワーク

- 試験項目、試験手順、確認項目を記述した設定ファイルをXMLで記述する。
- 以下のファイルを自動生成する。
  - 試験仕様書
  - 試験項目リスト
  - 試験用スクリプト

試験フレームワークの実演

- 試験仕様書の自動生成
- 試験項目リストの自動生成
- 試験用スクリプト自動生成

試験フレームワークの効果

- 試験仕様の源泉のみを作成すればよいため、試験準備の時間を削減することが可能。
- 試験仕様書の表示フォーマットを柔軟に変更することができる。
- 試験仕様の源泉を変更しても、試験スクリプトを手作業で変更する必要がない。
- 試験仕様の入力ミスを検出することが可能。

XMLの適用プロジェクト

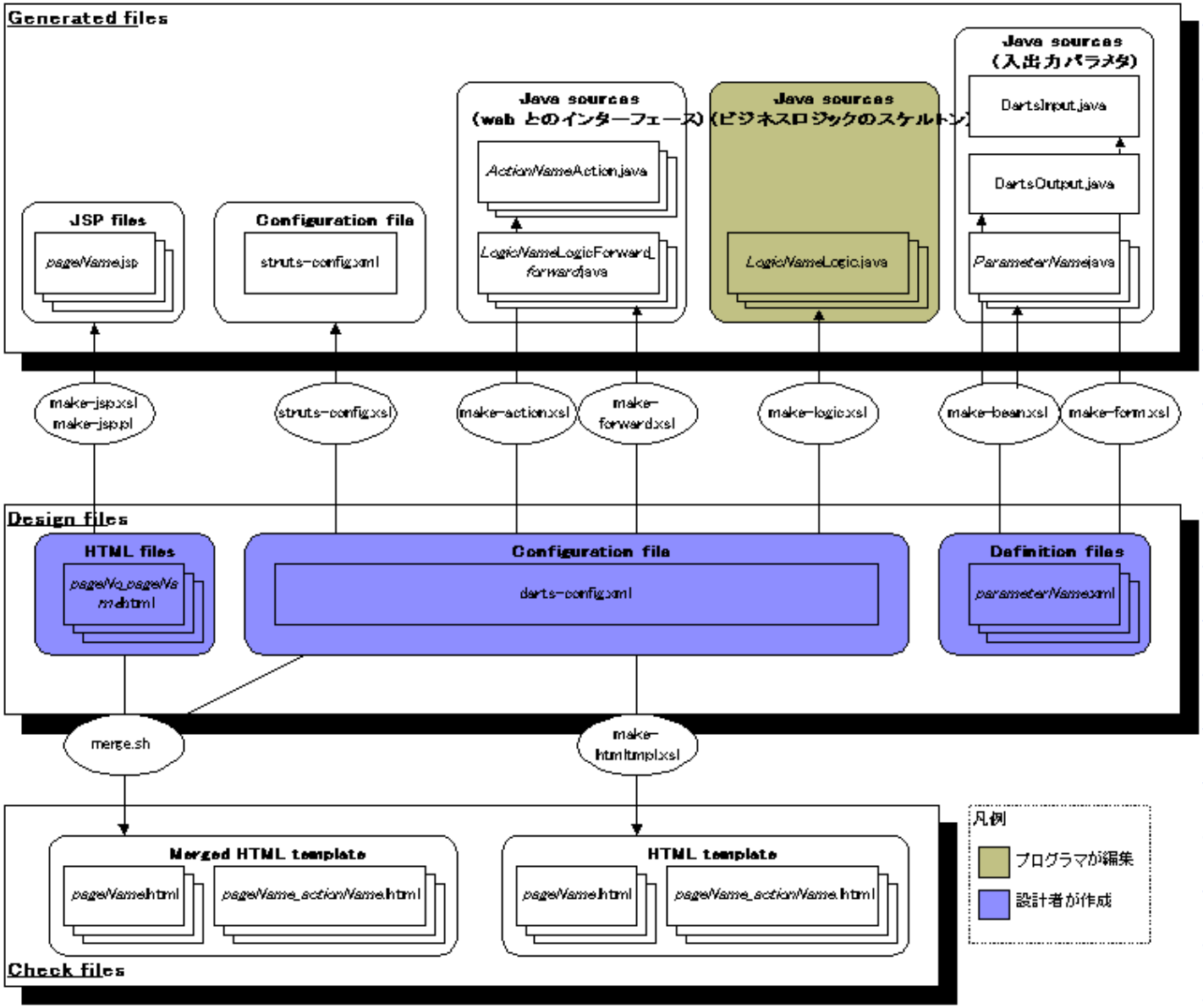
XMLの適用プロジェクト

- Tsunagi
- UMS

Tsunagiとは？

- Web アプリケーション作成フレームワーク
- 画面遷移時に行う処理のソースコードを自動生成する。
- 画面仕様のHTMLから、JSPを自動生成する。

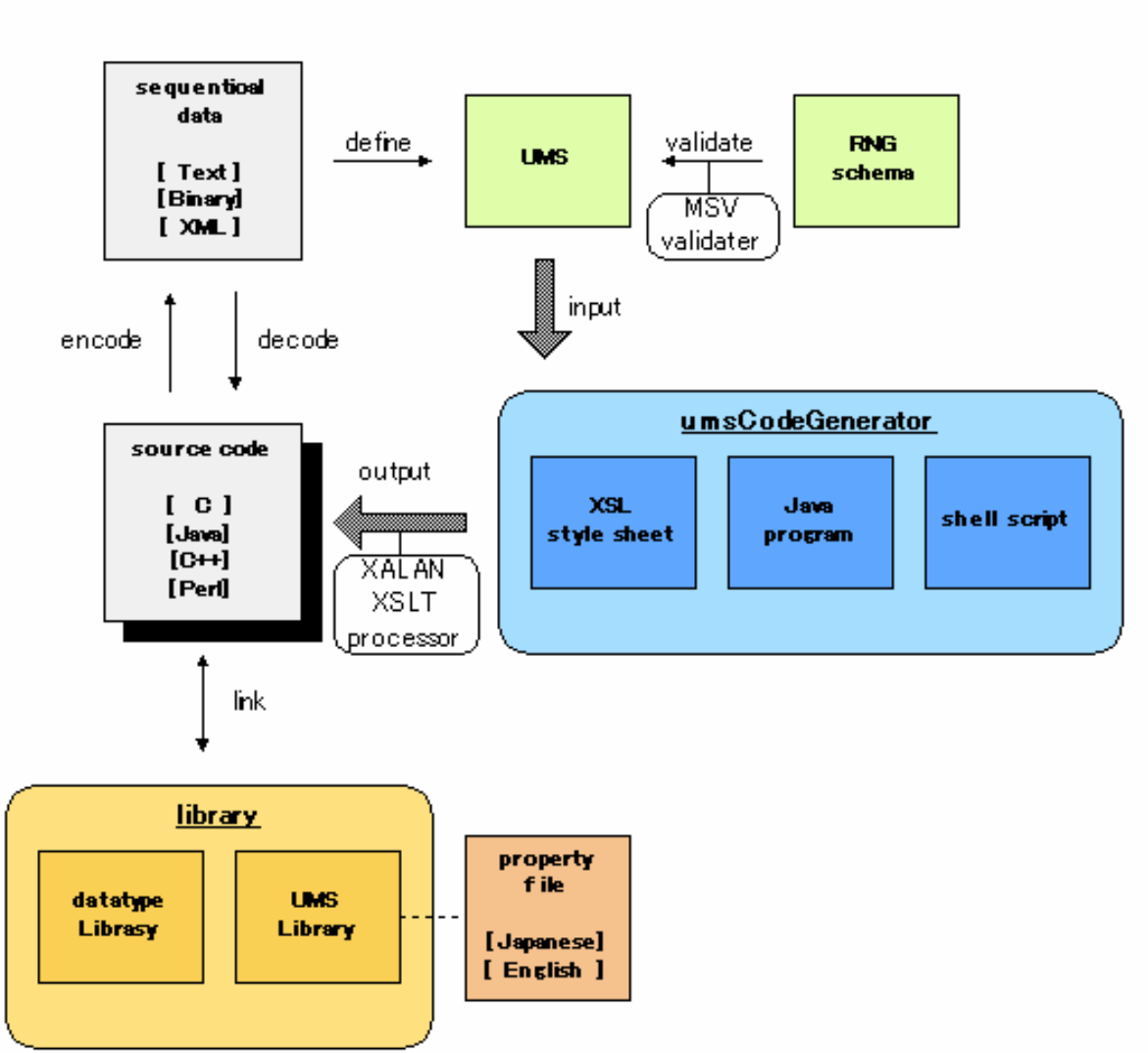
Tsunagiでできること



UMSとは？

- UMS(Universal Mapping Schema)
  - データ構造とプログラミング言語のマッピングを定義するスキーマ。
- umsCodeGenerator
  - UMSの処理系。
  - データのデコード/エンコードを行うソースコードを自動生成する。

umsCodeGeneratorでできること



まとめ

まとめ

- XMLは、汎用的なデータ記述言語である。
  階層構造を持ち、複雑なデータ形式の表現に優れる。
- XMLは、データでありながらデータ自身を説明する情報を含む「自己説明型のデータ」である。
- XMLは、それを用いて異なる言語を定義することができる「メタ言語」である。
  多くの分野でデータ交換/表現のための標準フォーマットとして広く利用されている。

まとめ

- XMLは、XMLそのものだけではなく関連する規格やツールが数多く公開されている。
  これらを利用することで開発の効率や利便性を向上させることができる。
- DOM/SAXやXSL等の関連技術を単独あるいは組み合わせて利用することで、XMLにより記述（モデル化）された設計情報を、下流の工程に（文書や口頭による伝達ではなく）そのまま利用することが可能になる。

参考：XMLエディタ

- XMLSpy
  （http://www.xmlspy.jp/01products/xmlspy.html）
  - Window環境で使用する商用アプリケーション。
  - XMLやXSLTの編集、整形、デバッグ等が可能。
- Emacs + nxml-mode
  （http://www.thaiopensource.com/nxml-mode/）
  - Emacs用のXMLモード。
  - XML、XSLTのタグの保管や、スキーマによるリアルタイムな検証が可能。

参考：XMLの分析ツール

- xmllint （http://xmlsoft.org/xmllint.html）
  - コマンドラインで実行するXML分析ツール。
  - XMLの検証や、整形を行うことが可能。

参考：XMLの検証ツール

- MSV（Sun Multi-Schema XML Validator）（https://msv.dev.java.net/）
  - コマンドラインで実行する検証ツール（Javaが必要）。
  - DTD、RELAX NG、W3C XML Schema等によるXMLデータの検証が可能。
- Jing（http://thaiopensource.com/relaxng/jing.html）
  - コマンドラインで実行する検証ツール（Javaが必要）。
  - DTD、RELAX NG、W3C XML Schema等によるXMLデータの検証が可能。

参考:スキーマ変換ツール

- Trang
  （http://thaiopensource.com/relaxng/trang.html）
  - コマンドラインで実行するスキーマ変換ツール（Javaが必要）。
  - 以下のようなスキーマ同士の変換が可能。
    - RELAX NG → W3C XML Schema
    - DTD → RELAX NG

    etc.

参考：XSLTプロセッサ

- Xalan-Java　（http://xml.apache.org/xalan-j/）
  - XMLを別形式に変換するためのXSLTプロセッサ。
  - コマンドラインから利用可（Javaが必要）。Javaプログラムから呼び出して利用することも可能。
  - C++から利用するためのXalan-C++も公開。
- Saxon　（http://saxon.sourceforge.net/）
  - XMLを別形式に変換するためのXSLTプロセッサ。
  - コマンドラインから利用可（Javaが必要）。Javaプログラムから呼び出して利用することも可能。
  - XSLTの最新仕様XSLT2.0をサポートした版も公開。