XML の基礎と実践

Ⅰ． XML の基本と構造

１．XML とは

XML(eXtensible Markup Language)とは、「情報記述言語」である。
HTML(Hyper Text Markup Language)はプレーンテキスト（平文）にタグと呼ばれる命令を埋め込む（Markup）ことで文書を修飾したり、構造を示したりする Web 用のマークアップ言語である。
たとえば、

<B>WORD</B>は今一番普及しているワープロソフトです。

と記述すると、<B>タグで囲まれた文字列は太字で表示してくれます。
しかし、太字で表された「WORD」という言葉は、人間にとって太字であるという表面的な事実にとどまらず、「重要な単語」であることを推測させる材料になっています。分脈によっては、それが「テーマ」、「タイトル」であることまで判断できるかもしれない。
しかし、コンピュータでは、依然<B>タグで囲まれた「WORD」は、太字で表されるべき文字列にすぎません。コンピュータにとって、それは「テーマ」でもなければ、「タイトル」でもない。
ところが、XML を使用すると、これが次のように、明確にその文字列単位のデータとしての意味合いを定義することができる。

<keyword>WORD</keyword>は今一番普及しているワープロソフトです。

このデータは人間にとってもコンピュータにとっても、非常に可読性に優れた言語であることがわかります。
また、このことは文書データをシステム的に検索する場合に大きな意味を持ちます。たとえば、HTML文書では、「WORD」というキーワードを含んだ文書を検索するのに、「全文くまなく」舐めなければならなかった。結果、キーワードでもない「WORD」であったり、「PassWORD」といった同じ「WORD」でも「WORD」違いであったり、また、「SWORD」などといった、まったく関係ない言葉の一部だったりします。
これは、HTML では、情報の属性情報をもたないための「欠陥」です。
これにたいして、XML文書では、キーワードがキーワードとしてコンピュータに認識されるように記述されます。これによって、コンピュータはキーワードだけに的を絞って情報を検索することができます。
その結果、XML文書が普及すると、インターネットの広い情報空間を、一つの膨大な知識データベースとして扱うことができるようになります。

2. XML の利点

・メディアに依存しない電子出版を可能にする。
・産業界は、データ、特に電子商取引データ用のプラットフォームに依存しないプロトコルを定義できる。
・ユーザーエージェントが受信後すぐに自動処理できる形で情報を引き渡すことができる。
・XML を用いると、人々が容易に安価なソフトウエアを使ってデータ処理をすることができる。
・人々が欲しい形で情報を表示することができる。
・情報についてのデータであるメタデータを提供することができ、情報の提供者と消費者でお互いに情報を容易に交換できる。
また、テクニカルには、
・「データ」と「外見」の情報を完全に分離できるので、どちらか一方だけを修正するのが容易である。
・同じデータを何回も書かずに、一度入力したデータを必要に応じて再利用できる。
・HTML 文書作成作業の自動化や、スクリプトによる文書内容の加工・検索が可能である。

■ HTMLについて

HTML は、「ハイパーテキスト構造をもった文書を記述するためのマークアップ言語」であるが、
HTML は、＜＞で囲まれた「タグ」を使用して、テキストデータのままでハイパーテキスト構造を持った文書を作成している。
しかし、HTML はタグの名前と働きが、すべて固定的に決まっている。従って、HTML のタグで表現できないデータを記述することができない。
また、HTML ではタグの記述に際して、ある程度いい加減なタグ付けを行っても、ブラウザが適当に辻褄を合わせてしまう傾向がある。

■ HTML のタグの基本

・タグは、半角の＜＞で囲み、「データ」の部分と区別する、
・データは、タグに挟まれたテキスト、または、属性値として記述する、
・データをタグで挟むか属性値にするかは、自由に選択できる、
・タグや属性の名前は、自由に決めてよい、
・階層構造を持ったデータ記述ができる。階層は、タグを入れ子にすることで表現する、が基本になっている。

ちなみに、「タグ」というのは、「<tag>」あるいは、「</tag>」のことで、「属性」、「属性

値」というのは、「＜IMG　SRC=”foo.jpg”>」の中の「IMG　SRC=”foo.jpg”」のことである。

３．XMLの基本構造

XML は、HTML と違って、タグ付けは厳しく規定されている。

３．１　XML 宣言

「このファイルの中身は XML ですよ」という宣言を最初にする必要がある。
基本構造は、以下の形式である。
<?xml　version=”1.0”?>
また、文書を保存する場合の文字エンコード規則を指定します。

```
<?xml　version=”1.0”　encoding=“Shift__JIS”?>
```

３．２　タグと要素と属性

例

```
<?xml　version=”1.0”　encoding=“Shift__JIS”?>
<メモ>初めてのXML</メモ>
<メモ>これからXMLを学ぶ</メモ>
```

XML で使用されるタグには、「開始タグ」、「終了タグ」、「空要素タグ」の区別がある。
「開始タグ」は「<タグ名>」で、対象のテキストの左に配置される
例

```
 <名前>
 <name>
```

「終了タグ」は「</タグ名>」で、対象のテキストの右に配置される
例

```
  </名前>
  </name>
```

HTML では、テキストを挟まないタグとして、「<HR>」、「
」、
「<LI>」などがあるが、XML　では存在せず、必ず、開始タグで始まり、終了タグで終わる必要がある。
このため、空要素タグがある。

「空要素タグ」は、「<タグ名/>」という形式をとる。
例

```
<hr/>、<IMG　SRC＝”image.jpg”/>
```

XML では、開始タグ、終了タグ、テキストを総称して「要素（element）」という。

３．３　タグの入れ子と階層構造

XML では、ある開始タグと終了タグで構成されるペアの中に、別の開始タグと終了タグのペアを入れ子（ネスト）させることで、データの階層構造を表現できる。
例

```
<?xml version='1.0' encoding='Shift_JIS' ?>
<parent>
  <child>テキスト</child>
</parent>
```


Sample01.xml

```
<?xml version='1.0' encoding='Shift_JIS' ?>
<parent>
    <child>テキスト</child>
    <child>テキスト</child>
</parent>
```


Sample04.xml

３．４　タグや属性名に使用できる文字

XML では、原則としてタグの名前や属性は自由に決めてよいが、
一部に使用できない単語や文字種がある。
・「XML」や「XML～」は使用できない
・スペースを含む名前は使用できない
・「＆」、「＜」、「＞」、「“」、「‘」はタグや属性の名前に使用できない
・数字で始まる名前は使えない

３．５　記号と文字の指定方法

XML では、「＆」、「＜」、「＞」、「“」、「‘」の 5 種類の半角文字はタグで挟まれたテキストや属性のなどに使用できない。それを使用したいときは、以下のように指定すると使用できる。

文字	書き方
&	&amp;
<	&lt;
>	&gr;
“	&quot;
‘	&apos;

3．6　属性の書き方

開始タグに属性を追加できる。

例

```
<tag  attrib  =  “属性値”>テキスト</tag>
```

属性は、名前（name）と値（value）の組で表し、開始タグの中で

名前＝”値“

という形式で記述する。

属性の値は、二重引用符（ダブルクオーテーション　“）または一重引用符（シングルクオーテーション　‘）で囲む。

3．7　コメントの書き方

コメントは、以下のように<!－－で始まり－－>で終了して、改行が必要です。

<!－－　コメント文　－－>

3．8　ウェルフォームド XML 文書

XML 文書は、必ず守らなければならないタグ付けとして、以下のルールがあります。

・ルート要素は 1 つだけ

・開始タグには、必ず対応する終了タグがある。テキストを挟まないタグは、空要素タグとして記述される。

・タグの入れ子構造は、正確に行われている。

この原則を守った XML 文書をウェルフォームド文書という。

3.9　処理命令

アプリケーションのための命令を文書内に入れることができる。

形式

<?ターゲット名　命令内容?>

例

```
<?editor linebreak?>
```

3.10　実際に書いてみる

「日記」を XML で書いてみよう。

構成要素として、以下のものを考える。

・日付

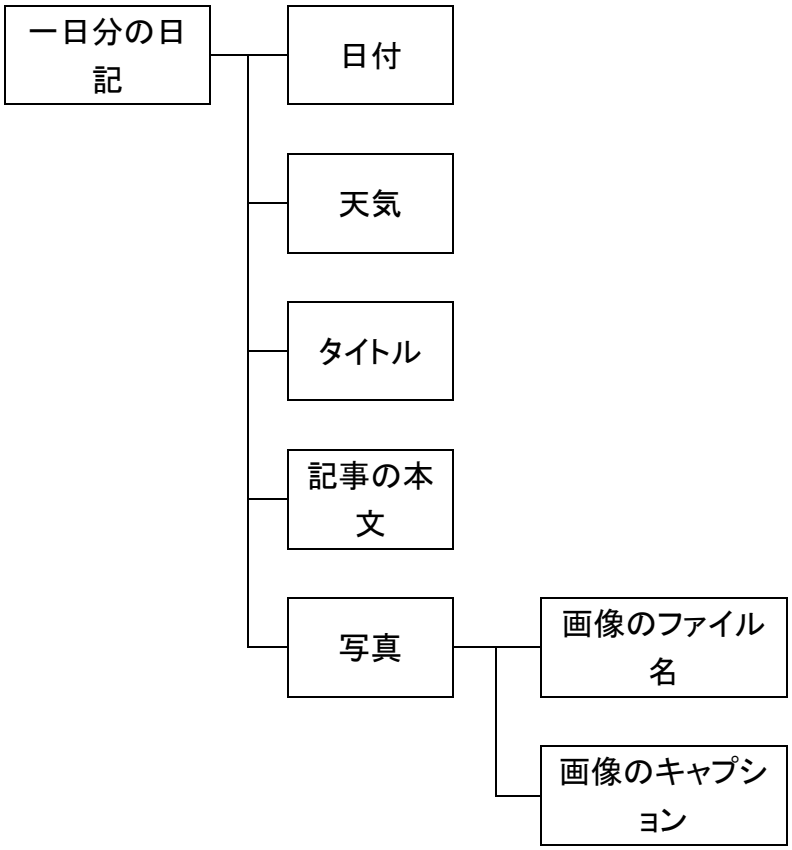
・天気

・タイトル

・記事の本文

・写真

この構成をツリー構造で示すと以下のようになる。



タグと属性の名前を決める

・一日ごとの日記：topic

・記事の日付：date

・天気：weather

・タイトル（その日のキーワード）：keyword

・記事の本文：text

・写真のファイル名：photo

・写真のキャプション：caption

XML ファイルとして、メモ帳で書き、ファイル名と拡張子は、「.xml」とする。

```
<?xml version='1.0'  encoding='Shift_JIS' ?>
<xdoc>

<topic>
<date>2005/4/1</date>
<weather>晴れ</weather>
<keyword>今日はエイプリルフール</keyword>
<text>
久しぶりに子供に帰った気持ちで、大きなうそをついてみよう。誰にしようかな。何にしようかな。名実ともにまっかなうそをつくことにしよう。
</text>
<photo caption="エイプリルフール">s20050401.jpg</photo>
</topic>

</xdoc>
```

図１－１　日記

Ⅱ． XSLT の基本と操作

XML 文書は、それだけでは単なる「データ」に過ぎない。外見をコントロールしてブラウザの画面に出すには、「スタイルシート」が必要になる。逆に、「スタイルシート」によって、同じ「XML 文書」を違った形式で画面に出すことが可能になる。

つまり、「データ」と「画面表現の形式」を分離することが可能になる。

XSLT（XSL Transformation）は、XML文書を別のXML文書へ変換するための言語である。XSLTで記述された変換ルールをXSLTスタイルシートと呼ぶ。

1. XSLTの仕組み

テンプレートルール：xsl:template

XSLT で XML 文書を変換するには、「何を」「どのように」変換するかを指定する。この指定をテンプレートルールと呼ぶ。

テンプレートルールは、「変換規則の適用先となるノードの指定」と「変換規則」を xsl:template で定義する。

変換規則の適用先となる XML 文書ノード指定をパターンと呼び、match 属性で指定する。パターンの指定は XPath を使用する。

変換規則のことを、テンプレートと呼び、XML 文書の変換を記述する。

形式

```
< xsl:template match ="パターン">
  テンプレート
</ xsl:template >
```

はじめに、以下のXML文書を

```
<?xml version="1.0" encoding="Shift_JIS" ?>
<メモ>はじめての XSLT</メモ>
```

次のように変換する。

```
<?xml version="1.0" encoding="Shift_JIS" ?>
<memo>はじめての XSLT</memo>
```

この変換の目的は、「メモ」要素を「memo」要素に置き換えることである。

これを行うXSLTスタイルシートは以下のようになる。

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

```
3:   <xsl:template match="メモ">
4:        <memo><xsl:apply-templates/></memo>
5:   </xsl:template>
6:</xsl:stylesheet>
```

XSLTスタイルシート自身もXML文書である。
ルート要素は、xsl:stylesheet となる。
2行目は、XSL：Stylesheetの Version および XSLT の指定を記述している。
3行目の xsl:template 要素は、テンプレートルール（template rule）と呼ぶ。1つのXSLTスタイルシートは複数個のテンプレートルールから構成される。
テンプレートルールには、XML文書の変換規則を記述する。変換規則をテンプレート(template)と呼ぶ。xsl:template 要素の match 属性には、そのテンプレートを変換元のどのノードに適用するかの条件を書く。この条件のことをパターン（pattern）と呼ぶ。XSLT のパターンは XPath という言語で記述する。上記の例では、match 属性の値は「メモ」がパターンである。
4行目は、テンプレートルールの子要素で、テンプレートである。テンプレートには、パターンに適合したノードの変換規則が記述される。上の例では、<memo><xsl:apply-templates/></memo>がテンプレートになる。xsl:apply-templates 要素は、パターンに適合したノードの子ノード（要素ノード、テキストノード）に他のテンプレートルールを適用するというXSLTの命令要素である。xsl:apply-templates 要素が memo 要素で囲まれているので、このテンプレートの意味は、「パターンに適合したノード（メモ要素）の子ノードを memo タグで囲んだものを出力する」という意味になる。

2. 属性とテキストノードの追加

次に、下記の XML 文書を生成する。

```
<?xml version="1.0" encoding="Shift_JIS" ?>
<memo  class ="para">メモ：はじめての XSLT</memo>
```

memo 要素に class 属性を追加し、テキストノードの先頭に「メモ：」という文字列を追加している。

このための XSLT スタイルシートは次のようになる。

```
1: <?xml version="1.0" encoding="Shift_JIS" ?>
2:<xsl:stylesheet version="1.0"  xmlns:xsl="http://www.w3.org/1999/xsl/Transform"
  >
3:   <xsl:template match="メモ">
4:        <memo class="para">
```

```
5:              <xsl:text>メモ:</xsl:text>
6:              <xsl:apply-templates/>
7:        </memo>
8:   </xsl:template>
9:</xsl:stylesheet>
```

4行目で、<memo class="para">を挿入し、
5行目で、文字列を挿入する xsl:text 要素を用いて「メモ:」を挿入している。

3. XPath の詳細

XPath は、XML 文書のある部分を特定するための言語である。

3. 1 式の基本

1)要素の選択

```
<?xml version="1.0" encoding="Shift_JIS" ?>
<doc>
  <title>XSLT 入門</title>
  <section>
    <title>はじめに</title>
    <para>XSLT は・・・</para>
    <para>このように・・・</para>
  </section>
</doc>
```

① 単純な要素の選択

doc ルート要素である doc 要素とマッチする
title 2つの title 要素とマッチする
para 2つの para 要素とマッチする

② 親子関係

section/para section 要素の子の2つの para 要素とマッチする
section/title section 要素の子の title 要素とマッチする
doc//title doc 要素の子孫の title 要素とマッチする
para/../title para 要素の親の子に当たる title 要素にマッチする
/ は親子関係 // は子孫関係 .. は親ノードを示す

例

```
<?xml version="1.0" encoding="Shift_JIS" ?>
<doc xml:lang="ja">
```

```
 <title>XSLT 入門</title>
 <section type="abstract">
  <para xml:lang="ja">本書では・・・</para>
  <para xml:lang="en">This document・・・</para>
 </section>
 <section type="normal">
  <title>はじめに</title>
  <para>XSLT は・・・</para>
  <para>このように・・・</para>
 </section>
</doc>
```

③ 単純な属性の選択

doc/@xml:lang　ルート要素 doc の xml:lang 属性にマッチする

section/@type　2つの section 要素の type 属性にマッチする

＊/@xml:lang　すべての xml:lang にマッチする

④ ワイルドカード

＊　すべての要素にマッチする

html:＊　"html"という接頭辞で宣言されている要素すべてにマッチする

section/＊　section 要素の子要素にマッチする

section/@＊　section 要素のすべての属性にマッチする

2） 述語

title[@type="abstract"] は、type 属性の値が"abstract"であるような title 要素にマッチする。

[ ] で囲んだ部分を述語(predicate)と呼ぶ。述語でかかれた式を述語式と呼ぶ。

① 値の指定

section[@type＝"abstract"]　type 属性の値が"abstract"である section 要素

section[@type="astract"]/para　type 属性の値が"abstract"である section 要素の子要要素の para 要素にマッチする

section[title＝"はじめに"]　title が"はじめに"である section 要素にマッチする

② 位置の指定

para[1]

para[position()=1]　いずれも para 要素にマッチする

section[last()]

section[position()=last()]　いずれも最後の section 要素にマッチする

para[last()-1]　　最後から2番目の para 要素にマッチする
para[position()>3]　　4番目以降の para 要素にマッチする

3） 冗長な構文
para は XPath では省略構文である。
それに対して、child::para は冗長な構文である。
ここで、“child“を軸（axis）、“para“をノードテスト（node　test）という。
軸には、13個用意されている。

child	文脈ノードの子ノードを選択する
descendant	子孫ノード。子孫ノードは子ノードまたは子ノードの子ノード
parent	親ノード
ancestor	先祖ノード。親ノードまたは親ノードの親ノード
following-sibling	後ろにあるすべての兄弟ノード
preceding-sibling	前にあるすべての兄弟ノード
following	文脈ノード以降にあるすべてのノード
preceding	文脈ノードより前にあるすべての要素
attribute	文脈ノードの属性を選択する
namespace	すべての要素
self	文脈ノード自身
descendant-or-self	文脈ノードと子孫ノード
ancestor-or-self	文脈ノードと祖先ノード

4） 式と演算子
① 演算子
｜　左右の式を評価した結果のノード集合
/　 親子関係
//　子孫関係
②ブール演算子
or　 左右演算の Or 演算
and　 左右演算のAnd演算
③ 比較演算子
＝
！＝
<＝
<
＝>

>
左右のオペランドの型で振る舞いは変わる。
④ 数値演算子
＋ 加算
― 減算
div　浮動小数点の割り算
mod　あまり

4. XSLT の詳細

1)テキストの生成:xsl:text
xsl:text 要素は、テキストノードを生成する。
形式

```
<xsl:text>
  テキスト
</xsl:text>
```

次の XML 文書は、

```
<?xml version="1.0" encoding="Shift_JIS" ?>
<book>
  <title>XML マスターとは</title>
  <price>2500</price>
</book>
```

下記の XSLT スタイルシートを適用すると、

```
<?xml version="1.0" encoding="Shift_JIS" ?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <answer>
      <xsl:text>(1) </xsl:text>
        <xsl:value-of select="book/title" />
    </answer>
  </xsl:template>
</xsl:stylesheet>
```

次のような XML 文書が生成される。

```
<answer>
  (1) XML マスターとは
```

```
</answer>
```

2）テキストの動的生成：xsl:value-of

XML 文書の中から動的に生成する。select 属性に指定した XPath 式が評価され、その結果がテキストノードとして生成される。

形式

```
<xsl:value-of select="XPpath 式"/>
```

次の XML 文書を、

```
<?xml version="1.0" encoding="Shift_JIS" ?>
<book>
  <title code="1234">概説 XML</title>
</book>
```

下記の XSLT スタイルシートを適用すると、

```
<?xml version="1.0" encoding="Shift_JIS" ?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <answer>
      <!-- title 要素のテキストを生成する -->
      <xsl:value-of select="book/title" />
      <!-- code 属性のテキストを生成する -->
      <xsl:value-of select="book/title/@code" />
    </answer>
  </xsl:template>
</xsl:stylesheet>
```

次のような XML 文書が生成される。

```
<answer>
  概説 XML
  1234
</answer>
```

3）要素の生成：xsl:element

xsl:element は要素を生成する。

形式

```
<xsl:element name="">
  テンプレート
```

```
</xsl:element>
```

次の XML 文書を

```
<?xml version="1.0" encoding="Shift_JIS" ?>
<book>
  <title>XML マスターとは</title>
  <price>2500</price>
</book>
```

下記の XSLT スタイルシートを適用すると、

```
<?xml version="1.0" encoding="Shift_JIS" ?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <answer>
      <xsl:element name="data">
        <xsl:value-of select="book/title" />
      </xsl:element>
    </answer>
  </xsl:template>
</xsl:stylesheet>
```

次のような XML 文書が生成される。

```
<answer>
  <data>
    XML マスターとは
  </data>
</answer>
```

4）属性の生成：attribute

xsl:attribute は、属性ノードを生成する。この要素は、要素ノードを生成する場所の最初に記述しなければならない。

形式

```
<xsl:attribute name="属性名">
  テンプレート
</xsl:attribute>
```

次の XML 文書を

```
<?xml version="1.0" encoding="Shift_JIS" ?>
<book>
  <title>XML マスターとは</title>
  <price>2500</price>
</book>
```

下記の XSLT スタイルシートを適用すると、

```
<?xml version="1.0" encoding="Shift_JIS" ?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <answer>
      <xsl:element name="data">
        <xsl:attribute name="price">
          <xsl:value-of select="book/price"/>
        </xsl:attribute>
        <xsl:value-of select="book/title" />
      </xsl:element>
    </answer>
  </xsl:template>

</xsl:stylesheet>
```

次のような XML 文書が生成される。

```
<answer>
    <data price="2500">
        XML マスターとは
    </data>
</answer>
```

5）テンプレートの適用：xsl:apply-templates

select 属性で指定した式を評価する。評価結果、ノードが選択された場合、そのノードにマッチするテンプレートを XSLT スタイルシートから探して変換処理を行う。

select 属性を省略した場合は、カレントノードの子ノードが選択される。

形式

```
<xsl:apply-templates select="次に処理するノード集合"/>
```

例

次の XML 文書を

```
<?xml version="1.0" encoding="Shift_JIS" ?>
<book>
  <title>XML マスターとは</title>
  <price>2500</price>
</book>
```

下記の XSLT スタイルシートを適用すると、

```
<?xml version="1.0" encoding="Shift_JIS" ?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
   <magazine>
     <title-price>
       <xsl:apply-templates select="book" />
     </title-price>
   </magazine>
  </xsl:template>
  <xsl:template match="book">
    <xsl:value-of select="title" />
    <xsl:value-of select="price" />
  </xsl:template>
</xsl:stylesheet>
```

次のような XML 文書が生成される。

```
<magazine>
  <title-price>
    XML マスターとは 2500
  </title-price>
</magazine>
```

6）繰り返し：xsl:for-each

形式

```
<xsl:for-each select="処理の対象となるノードを選択する XPath 式">
  テンプレート
</xsl:for-each>
```

繰り返し動作を指定するときは、xsl:for-each を使用する。xsl:for-each は必ず select

属性を持ち、select 属性には XPath 式が記述され、ノード集合のそれぞれのノードに対して xsl:for-each 要素内のテンプレートが適用される。

例

```
<?xml version="1.0" encoding="Shift_JIS" ?>
<book>
  <title>XML マスターとは</title>
  <title>XML 概説</title>
  <title>Java と XML</title>
</book>
```

上記の XML を下記の XSLT を適用すると、

```
<?xml version="1.0" encoding="Shift_JIS" ?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:template match="/">
      <magazine>
        <title-list>
          <xsl:for-each select="book/title" >
            <xsl:value-of select="." />
          </xsl:for-each>
        </title-list>
      </magazine>
    </xsl:template>
</xsl:stylesheet>
```

次のようになる。

```
<magazine>
  <title-list>
    XML マスターとは
    XML 概説
    Java と XML
  </title-list>
 </magazine>
```

7） 条件分岐：xsl:if

形式

```
<xsl:if test="式">
  テンプレート
```

```
</xsl:if>
```

xsl:if を使用すると、ノードの条件によって結果を変えられる。

次のテンプレートを持ったスタイルシートで、

```
<?xml version="1.0" encoding="shift_jis"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="authorlist">
    <para><xsl:apply-templates/></para>
  </xsl:template>
  <xsl:template match="authorlist/author">
    <xsl:apply-templates/>
    <xsl:if test="not(position()=last())">、</xsl:if>
  </xsl:template>
</xsl:stylesheet>
```

次の XML 文書に適用すると、

```
<?xml version="1.0" encoding="Shift_JIS" ?>
<authorlist>
  <author>HOSEI Akihiro</author>
  <author>Waseda Masahiro</author>
  <author>Tokyo Hirosi</author>
</authorlist>
```

以下のような結果が得られる。

```
<para> HOSEI Akihiro、Waseda Masahiro、Tokyo Hirosi</para>
```

8）多岐条件分岐： xsl:choose

xsl:choose 要素は、複数の選択肢の中から条件を満たしたものを1つ選択して処理を行う。xsl:when 要素に選択条件を記述し、要素内容に処理を記述する。xsl:when 要素の選択条件に該当しない処理は、xsl:otherwise 要素内容に記述。

形式

```
<xsl:choose>
  <xsl:when test="式1">
    テンプレート1
  </xsl:when>
  <xsl:when test="式2">
    テンプレート2
  </xsl:when>
```

```
      ・
      ・
      ・
   <xsl:otherwise>
      テンプレート n
   </xsl:otherwise>
</xsl:choose>
```

次のテンプレートを持ったスタイルシートで、

```
<?xml version="1.0" encoding="shift_jis"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
      <publication>
         <title-list>
            <xsl:apply-templates select="//book"/>
         </title-list>
      </publication>
   </xsl:template>
   <xsl:template match="book">
      <xsl:choose>
         <xsl:when test="price&gt;4000">
            <xsl:value-of select="title"/> --&gt;高い
         </xsl:when>
         <xsl:when test="price&gt;3000 and price&lt;4500">
            <xsl:value-of select="title"/> --&gt;まぁまぁ
         </xsl:when>
         <xsl:otherwise>
            <xsl:value-of select="title"/> --&gt;安い
         </xsl:otherwise>
      </xsl:choose>
   </xsl:template>
</xsl:stylesheet>
```

次の XML 文書に適用すると、

```
<?xml version="1.0" encoding="Shift_JIS" ?>
<booklist>
   <book>
         <title>XML マスターとは</title>
```

```
        <price>2500</price>
    </book>
  <book>
        <title>XML 概説</title>
        <price>5000</price>
    </book>
  <book>
        <title>Java&amp;amp;XML</title>
        <price>3500</price>
    </book>
</booklist>
```

以下のような結果が得られる。

```
<publication>
  <title-list>
    XML マスターとは-->安い  XML 概説-->高い  Java&amp;amp;XML-->まぁまぁ
  </title-list>
</publication>
```

9）テンプレートの呼び出し xsl:call-template

XSLT スタイルシートが大きくなってくると、同じような処理を行うテンプレートを 1 つにまとめた方がよい場合がある。プログラミング言語の関数のようなもので、名前つきテンプレート（named template）という。

形式

```
<xsl:call-template name="呼び出すテンプレートルールの名前"/>
```

呼び出されるテンプレートルールの形式

```
<xsl:template name="テンプレートルールの名前">
  テンプレート
</xsl:template>
```

例

XML 文書

```
<?xml version="1.0" encoding="Shift_JIS" ?>
<book>
  <title>XML マスターとは</title>
  <price>2500</price>
```

```
</book>
```

上記の XML 文書を下記のテンプレートを持ったスタイルシートに適用すると、

```
<?xml version="1.0" encoding="Shift_JIS" ?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
 <xsl:template match="/">
    <magazine>
      <title-price>
        <xsl:call-template name="booktemp" />
      </title-price>
    </magazine>
  </xsl:template>
  <xsl:template name="booktemp">
    <xsl:value-of select="book/title" />
    <xsl:value-of select="book/price" />
  </xsl:template>
</xsl:stylesheet>
```

下記の結果が得られる。

```
<magazine>
  <title-price>
    XML マスターとは
    2500
  </title-price>
</magazine>
```

10)ソート xsl:sort

要素をソートするのに xsl:sort 要素を用いる。

形式

```
<xsl:sort select="ソートキーとなる文字列"/>
```

以下の従業員リストを考える。

```
<?xml version="1.0" encoding="Shift_JIS" ?>
<従業員リスト>
  <従業員>
    <社員番号>345678</社員番号>
    <名前>法政一郎</名前>
    <ふりがな>ほうせいいちろう</ふりがな>
```

```
  </従業員>
  <従業員>
    <社員番号>987654</社員番号>
    <名前>東京三郎</名前>
    <ふりがな>とうきょうさぶろう</ふりがな>
  </従業員>
  <従業員>
    <社員番号>123456</社員番号>
    <名前>早稲田次郎</名前>
    <ふりがな>わせだじろう</ふりがな>
  </従業員>
</従業員リスト>
```

以下のスタイルシートを適用すると「ふりがな」でソートする。

```
<?xml version="1.0" encoding="Shift_JIS" ?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <html>
     <head><title>従業員リスト</title></head>
     <body>
      <ul>
       <xsl:for-each select="//従業員">
         <xsl:sort select="ふりがな"/>
         <li><xsl:value-of select="名前"/></li>
       </xsl:for-each>
      </ul>
     </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

その結果、五十音に並んだ従業員リストが得られる。

```
<html xmlns="http://www.w3.org/1999/xhtml">
  <head><title>従業員リスト</title></head>
  <body>
    <ul>
      <li>東京三郎</li>
```

```
        <li>法政一郎</li>
        <li>早稲田次郎</li>
      </ul>
    </body>
</html>
```

11） ノードのコピー　xsl:copy

ノードのコピーをできる。

形式

```
<xsl:copy/>
または
<xsl:copy>
テンプレート
</xsl:copy>
```

例

次のような HTML 文書を考える。

```
<?xml version="1.0" encoding="Shift_JIS" ?>
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>test</title>
  </head>
  <body>
    <h1>XHTML　メモ</h1>
    <p class="abstract">この文書では・・・</p>
  </body>
</html>
```

以下のようなテンプレートを使えばコピーできる。

```
<?xml version="1.0" encoding="Shift_JIS" ?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="*">
    <xsl:copy><xsl:apply-templates/></xsl:copy>
  </xsl:template>
</xsl:stylesheet>
```

その結果、次のような文書が得られる。

```
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>test</title>
  </head>
  <body>
    <h1>XHTML　メモ</h1>
    <p>この文書では・・・</p>
  </body>
</html>
```

12）変数

XSLT では、2つの変数、xsl:variable と xsl:param がある。両要素とも name 属性と select 属性を持ち、要素内容はテンプレートである。

```
<xsl:variable name="copyright">
    <p class="copy">Copyright © 2001 HOGE Corp. </p>
</xsl:variable>
```

これは、コピーライトの p 要素を copyright という名前の変数として宣言している。この変数は、XPath 式から以下のように参照できる。

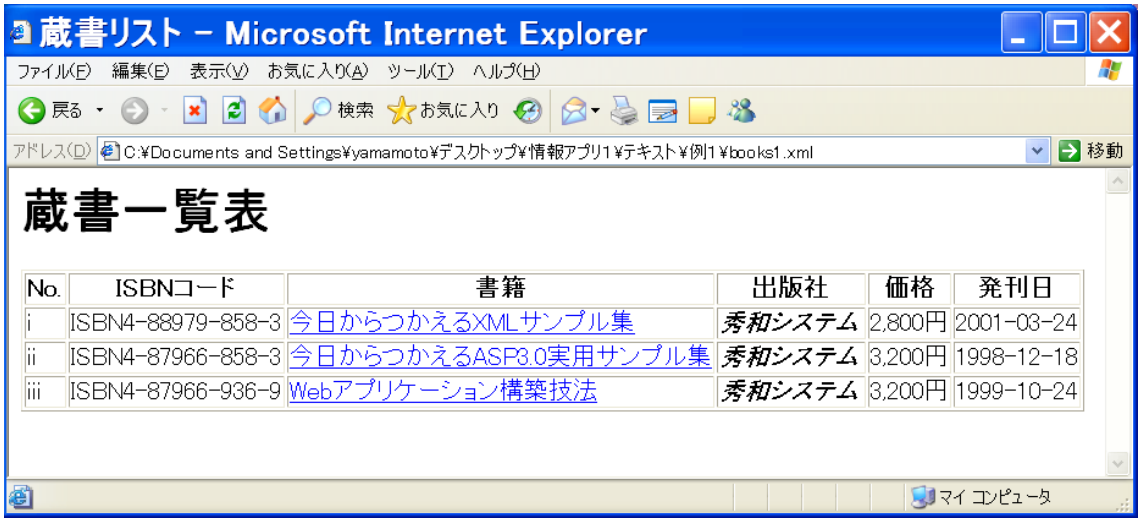
```
<xsl:copy-of  select  "$copyright"/>
```

Ⅲ． XML/XSLT の応用

XML を XSLT を用いた利用について、述べる。

1． 1つの XML ﾄﾞｷｭﾒﾝﾄをテーブル形式で表示する。

1）テーブル形式の表示結果



2） テーブル形式の XML(books1.xml)

```
<?xml version="1.0" encoding="Shift_JIS" standalone="yes" ?>
<?xml-stylesheet type="text/xsl" href="table.xsl" ?>
<books>
        <!--2000.12.24  山田-->
                <book isbn="ISBN4-88979-858-3">
                <name>今日からつかえる XML サンプル集</name>
                <publish>秀和システム</publish>
                <price>2800</price>
                <url>http://member.nifty.ne.jp/Y-Yamada/xml/</url>
                <published>2001-03-24</published>
                <desc>Web 世界を革新する最新技術<keywd>XML(eXtensible Markup Language)</keywd>。
「興味はあるけどいったい何に使えるの？」「とりあえず動いている XML を見てみたい」---そんな貴方のために最新の技術情報
と実用サンプルを多数取り揃え、ご紹介します。MSXML パーサ 3 対応</desc>
        </book>
                <book isbn="ISBN4-87966-858-3">
                <name>今日からつかえる ASP3.0 実用サンプル集</name>
```

```
                <publish>秀和システム</publish>
                <price>3200</price>
                <image>asp.jpg</image>
                <url>http://member.nifty.ne.jp/Y-Yamada/asp/</url>
                <published>1998-12-18</published>
                <desc>Windows2000/NT4.0/Me/98/95 でつかえるサーバサイド技術の Active Server Pages3.0。
潤沢に用意された追加コンポーネントや Access や SQL Server など既存 DB との連携により、強力な Web アプリケーションの
構築を可能とします。Windows2000 の登場でますます充実のパワフルテクニックを、豊富なサンプルとともに</desc>
        </book>
        <book isbn="ISBN4-87966-936-9">
                <name>Web アプリケーション構築技法</name>
                <publish>秀和システム</publish>
                <price>3200</price>
                <image>webware.jpg</image>
                <url>http://member.nifty.ne.jp/Y-Yamada/webWare/</url>
                <published>1999-10-24</published>
                <desc>IE5.0 、 IIS(PWS)4.0/5.0 で 動 作 す る Web グ ル ー プ ウ ェ ア を <ref
url="http://member.nifty.ne.jp/Y-Yamada/webware/">無償ダウンロード提供中！</ref>スケジュール管理や施設予約、ファ
イル共有、メール送受信、電子会議室、検索エンジン、不在ボードなどの豊富な機能を非常に簡単な設定だけで提供します。
ScriptEngine5.5 に対応し、DHTML、ScriptLets、RemoteScripting、XML(VML,HTML+TIME)、DHTML Behavior な
どスクリプト最新技術のケーススタディにも最適。</desc>
        </book>
</books>
```

3）テーブル形式の XSLT（table.xsl）

```
<?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" version="4.0" encoding="Shift_JIS" />
  <xsl:decimal-format NaN="未定" />
  <xsl:template match="/">
    <html>
    <head>
    <title><xsl:text>蔵書リスト</xsl:text></title>
    <link rel="stylesheet" type="text/css" href="books.css" />
    </head>
    <h1><xsl:text>蔵書一覧表</xsl:text></h1>
```

```
    <table border="1">
    <tr>
      <th>No.</th>
      <th>ISBN コード</th>
      <th>書籍</th>
      <th>出版社</th>
      <th>価格</th>
      <th>発刊日</th>
    </tr>
    <xsl:apply-templates select="books" />
    </table>
    </html>
</xsl:template>
<xsl:template match="books">
  <xsl:for-each select="book">
      <xsl:sort select="date" data-type="text" order="descending" />
      <xsl:sort select="price" data-type="number" order="ascending" />
      <tr>
        <td nowrap="nowrap">
            <xsl:number value="position()" format="i"/>
        </td>
        <td nowrap="nowrap">
            <xsl:value-of select="@isbn" />
        </td>
        <td nowrap="nowrap">
            <xsl:element name="a">
                <xsl:attribute name="href">
                    <xsl:value-of select="url" />
                </xsl:attribute>
                <xsl:value-of select="name" />
            </xsl:element>
        </td>
        <td nowrap="nowrap">
            <xsl:choose>
                <xsl:when test="publish[. = '秀和システム']">
                    <div style="font-style:italic;">
```

```
                        <xsl:value-of select="publish" />
                    </div>
                </xsl:when>
                <xsl:otherwise>
                    <xsl:value-of select="publish" />
                </xsl:otherwise>
            </xsl:choose>
        </td>
        <td nowrap="nowrap">
            <xsl:value-of select="format-number(price,'0,000')" />
            <xsl:if test="price[. != '']">円</xsl:if>
        </td>
        <td nowrap="nowrap">
            <xsl:value-of select="published" />
        </td>
    </tr>
  </xsl:for-each>
 </xsl:template>
</xsl:stylesheet>
```

2. 1つの XML ドキュメントを一覧形式で表示する。XMLドキュメントは、テーブル形式のものと同じである。XSLTを変えることで、違った形式に表現することができる。

1） 一覧形式の表示結果

2） 一覧形式の XML(books2.xml)

```
<?xml version="1.0" encoding="Shift_JIS" standalone="yes" ?>

<?xml-stylesheet type="text/xsl" href="description.xsl" ?>

<books>

        <!--2000.12.24　山田-->

                <book isbn="ISBN4-88979-858-3">

                <name>今日からつかえる XML サンプル集</name>

                <publish>秀和システム</publish>

                <price>2800</price>

                <url>http://member.nifty.ne.jp/Y-Yamada/xml/</url>

                <published>2001-03-24</published>

                <desc>Web 世界を革新する最新技術<keywd>XML(eXtensible Markup Language)</keywd>。「興味はあるけどいったい何に使えるの？」「とりあえず動いている XML を見てみたい」---そんな貴方のために最新の技術情報と実用サンプルを多数取り揃え、ご紹介します。MSXML パーサ 3 対応</desc>

        </book>

                <book isbn="ISBN4-87966-858-3">

                <name>今日からつかえる ASP3.0 実用サンプル集</name>
```

```
                <publish>秀和システム</publish>
                <price>3200</price>
                <image>asp.jpg</image>
                <url>http://member.nifty.ne.jp/Y-Yamada/asp/</url>
                <published>1998-12-18</published>
                <desc>Windows2000/NT4.0/Me/98/95 でつかえるサーバサイド技術の Active Server Pages3.0。
潤沢に用意された追加コンポーネントや Access や SQL Server など既存 DB との連携により、強力な Web アプリケーショ
ンの構築を可能とします。Windows2000 の登場でますます充実のパワフルテクニックを、豊富なサンプルとともに</desc>
        </book>
        <book isbn="ISBN4-87966-936-9">
                <name>Web アプリケーション構築技法</name>
                <publish>秀和システム</publish>
                <price>3200</price>
                <image>webware.jpg</image>
                <url>http://member.nifty.ne.jp/Y-Yamada/webWare/</url>
                <published>1999-10-24</published>
                <desc>IE5.0 、 IIS(PWS)4.0/5.0 で 動 作 す る Web グ ル ー プ ウ ェ ア を <ref
url="http://member.nifty.ne.jp/Y-Yamada/webware/">無償ダウンロード提供中！</ref>スケジュール管理や施設予
約、ファイル共有、メール送受信、電子会議室、検索エンジン、不在ボードなどの豊富な機能を非常に簡単な設定だけ
で提供します。ScriptEngine5.5 に対応し、DHTML、ScriptLets、RemoteScripting、XML(VML,HTML+TIME)、DHTML Behavior
などスクリプト最新技術のケーススタディにも最適。</desc>
        </book>
</books>
```

3）一覧形式の XSLT（description.xsl, parts.xsl）

description.xsl

```
<?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" version="4.0" encoding="Shift_JIS" />
  <xsl:variable name="title">秀和システム</xsl:variable>
  <xsl:attribute-set name="imageSize">
    <xsl:attribute name="width">120</xsl:attribute>
    <xsl:attribute name="height">160</xsl:attribute>
  </xsl:attribute-set>
  <xsl:template match="/">
    <html>
```

```
    <head>
    <title>
       <xsl:text disable-output-escaping="no">詳細情報</xsl:text>
    </title>
    <link rel="stylesheet" type="text/css" href="books2.css" />
    </head>
    <body>
    <h1>
       <xsl:text disable-output-escaping="yes">&lt;</xsl:text>
       <xsl:value-of select="$title" />
       <xsl:text disable-output-escaping="yes">出版書籍&gt;</xsl:text>
    </h1>
    <xsl:apply-templates select="books" />
    </body>
    </html>
</xsl:template>
<xsl:template match="books">
  <xsl:for-each select="book[publish = string($title)]">
     <xsl:sort select="date" order="descending" />
       <table border="0">
       <tr>
       <td>
          <img xsl:use-attribute-sets="imageSize">
             <xsl:attribute name="src">
                <xsl:choose>
                   <xsl:when test="image">
                      <xsl:value-of select="image" />
                   </xsl:when>
                   <xsl:otherwise>
                      <xsl:text>default.jpg</xsl:text>
                   </xsl:otherwise>
                </xsl:choose>
             </xsl:attribute>
          </img>
         </td>
         <td valign="bottom">
```

```
            <dl>
               <dt>
                  <xsl:number value="position()" format="1. "/>
                  <xsl:value-of select="name" />
               </dt>
               <dd>
                   <xsl:apply-templates select="desc" />
               </dd>
            </dl>
         </td>
         </tr>
         </table>
         <hr />
      </xsl:for-each>
   </xsl:template>
   <xsl:include href="parts.xsl" />
</xsl:stylesheet>
```

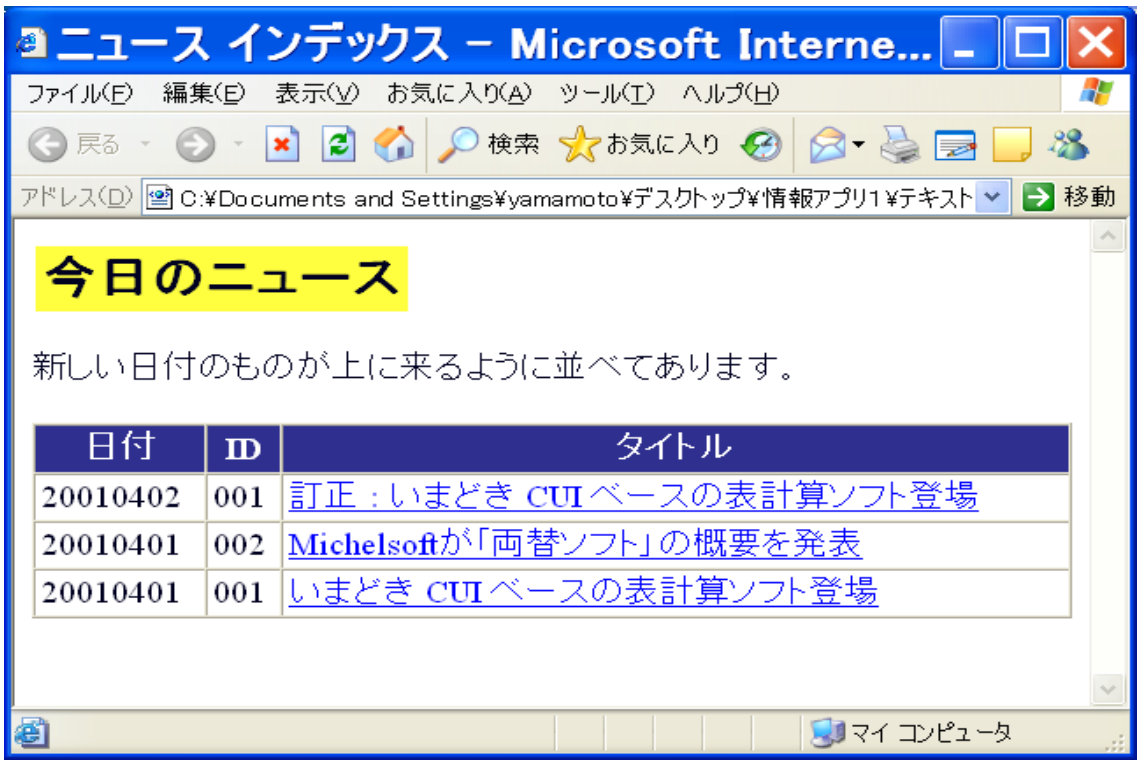
parts.xsl

```
<?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" version="4.0" encoding="Shift_JIS" />
          <xsl:template match="ref">
                    <xsl:element name="a">
                              <xsl:attribute name="href">
                                        <xsl:value-of select="@url" />
                              </xsl:attribute>
                              <xsl:value-of select="." />
                    </xsl:element>
          </xsl:template>
          <xsl:template match="keywd">
                    <span style="font-style:italic;">
                              <xsl:value-of select="." />
                    </span>
          </xsl:template>
          <xsl:template match="text()">
```

```
                    <xsl:value-of select="." />
            </xsl:template>
</xsl:stylesheet>
```

3.　新しい日付のものが上に来るように並べて表示する

1） 表示結果



2） XML（news.xml）

```
<?xml version="1.0" encoding="Shift_JIS" ?>
<?xml-stylesheet type="text/xsl" href="index.xsl" ?>
<xdoc>
<news>
<date>20010401</date>
<ID>002</ID>
<title>Michelsoft が「両替ソフト」の概要を発表</title>
<text>
<P>
マイケルソフト社は 1 日、同社としては初のカテゴリーである「両替ソフト」として、「Michelsoft Exchanger」の概要を発表した。
</P>
```

<P>
同社では、「Exchanger」を使用した両替機は世界各地の空港などへの設置が進むと見込んでおり、年間 5,000 台の販売計画を立てている。
</P>
</text>
<URL>
http://www.michelsoft.com.com/Exchanger/
</URL>
</news>
<news>
<date>20010401</date>
<ID>001</ID>
<title>いまどき CUI ベースの表計算ソフト登場</title>
<text>
<P>
耳鼻コープ社は、表計算ソフトの市場に殴り込みをかけるべく、極秘裏に開発を進めてきた新製品「耳鼻科ルク 2001」を、4 月 1 日に全世界一斉発売した。
</P>
<P>
「耳鼻科ルク 2001」は、外見をシンプル化することによって浮いた余力をワークシートの拡大と計算速度の向上に使われている、としており、「外見よりも物事の本質を見抜く目を持ったユーザーに勧めたい」とのことである。しかし、マウス操作を一切受け付けない「耳鼻科ルク 2001」が、どれだけ市場で受け入れられるかは未知数である。
</P>
</text>
<URL>
http://www.jibicorp.com.com/JibiCalc/
</URL>
</news>
<news>
<date>20010402</date>
<ID>001</ID>
<title>訂正 : いまどき CUI ベースの表計算ソフト登場</title>
<text>
<P>
昨日のニュースで、耳鼻コープ社が「耳鼻科ルク 2001」を発売したというニュースをお知らせしたが、このニュースは「真っ赤な嘘」であることが判明した。

```
</P>
<P>
実は、このニュースは株式公開（IPO）を目前にした耳鼻コープ社が、株価の吊り上げと話題作りの一挙両得を狙って流した、エイプリール・フールだったとのことである。
</P>
</text>
<URL>
</URL>
</news>
</xdoc>
```

3） XSLT（index.xsl）

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

<xsl:output method="xml" omit-xml-declaration="yes" />
<xsl:template match="/">

<HTML>
 <HEAD>
  <TITLE>ニュース インデックス</TITLE>
 </HEAD>

 <BODY BGCOLOR="#FFFFFF" TEXT="#000020">

 <TABLE BORDER="0" CELLPADDING="4">
  <TR>
   <TD BGCOLOR="#FFFF40" NOWRAP="NOWRAP">
    <SPAN STYLE="font-size:18pt; font-weight:bold;">
     今日のニュース
    </SPAN>
   </TD>
  </TR>
```

```
</TABLE>

<P>
 新しい日付のものが上に来るように並べてあります。
</P>

<TABLE BORDER="1" CELLPADDING="3" CELLSPACING="0" WIDTH="100%">

 <!-- 見出し表示行 -->
 <TR>
  <TH BGCOLOR="#2F2F8F"><SPAN STYLE="color:white;">日付</SPAN></TH>
  <TH BGCOLOR="#2F2F8F"><SPAN STYLE="color:white;">ID</SPAN></TH>
  <TH BGCOLOR="#2F2F8F"><SPAN STYLE="color:white;">タイトル</SPAN></TH>
 </TR>

 <!-- 記事一覧表示 -->
 <xsl:for-each select="xdoc/news">
  <!-- 日付と ID で、それぞれ降順に並べ替え -->
  <xsl:sort select="date" order="descending" />
  <xsl:sort select="ID" order="descending" />

  <TR>
   <!-- 日付表示列 -->
   <TD>
    <xsl:value-of select="date"/>
   </TD>
   <!-- ID 表示列 -->
   <TD>
    <xsl:value-of select="ID"/>
   </TD>

   <!-- タイトル表示列（個別記事へのハイパーリンク）-->
   <TD>
    <!-- タイトル文字列をハイパーリンクにする A タグを生成 -->
    <!-- リンク先ファイル名は「日付+ID+".html"」 -->
    <xsl:element name="a">
```

```
        <xsl:attribute name="href">
          <xsl:value-of select="date"/>
          <xsl:value-of select="ID"/>.html
        </xsl:attribute>

        <!-- 実際に表示されるタイトル文字列 -->
        <xsl:value-of select="title"/>
       </xsl:element>
     </TD>
    </TR>
  </xsl:for-each>
 </TABLE>

 </BODY>
</HTML>

</xsl:template>
</xsl:stylesheet>
```

Ⅳ. DOM の応用

XML 文書のデータを変更したり、構造を変更したい場合があります。また、XML 文書のデータを処理したい場合があります。

DOM（Document Object Model）は、XML 文書をメモリ上にツリー構造で管理し、プログラムからアクセスするようにした API（Application Program Interface）です。XML 文書にアクセスするための各種のメソッドが用意されており、データの検索、修正、削除を行うことができます。

1. DOM ツリー

XPath のロケーションパスと同様に、XML 文書をツリー構造で表現したものが DOM ツリーで、それぞれのノードの種類を指定できます。

表4．1 ノードの種類

種類	意味	定義	数値表現
Element	要素	ELEMENT__NODE	1
Attr	属性	ATTRIBUTE__NODE	2
Text	テキスト	TEXT__NODE	3
CDATA Section	CDATA セクション	CDATA__SECTION__NODE	4
Entity References	実態参照	ENTITY__REFERENCE__NODE	5
Entity	実態	ENTITY__NODE	6
Processing Instruction	処理命令	PROCESSING__INSTRUCTION__NODE	7
Comment	コメント	COMMENT__NODE	8
Document	文書	DOCUMENT__NODE	9
Document Type	文書型	DOCUMENT__TYPE__NODE	10
Document Fragment	作業用文書ノード	DOCUMENT__FRAGMENT__NODE	11
Notation	記述	NOTATION__NODE	12

空白の扱い

XML では、空白文字、TAB 文字、改行文字はすべて、そのまま認識される。従って、アプリケーションを作成するときには、それらのデータをどう扱うか組み込む必要がある。

DOM ツリー

DOM ツリーのトップは、Document ノードである。Document ノードの下にルート要素が1つあります。

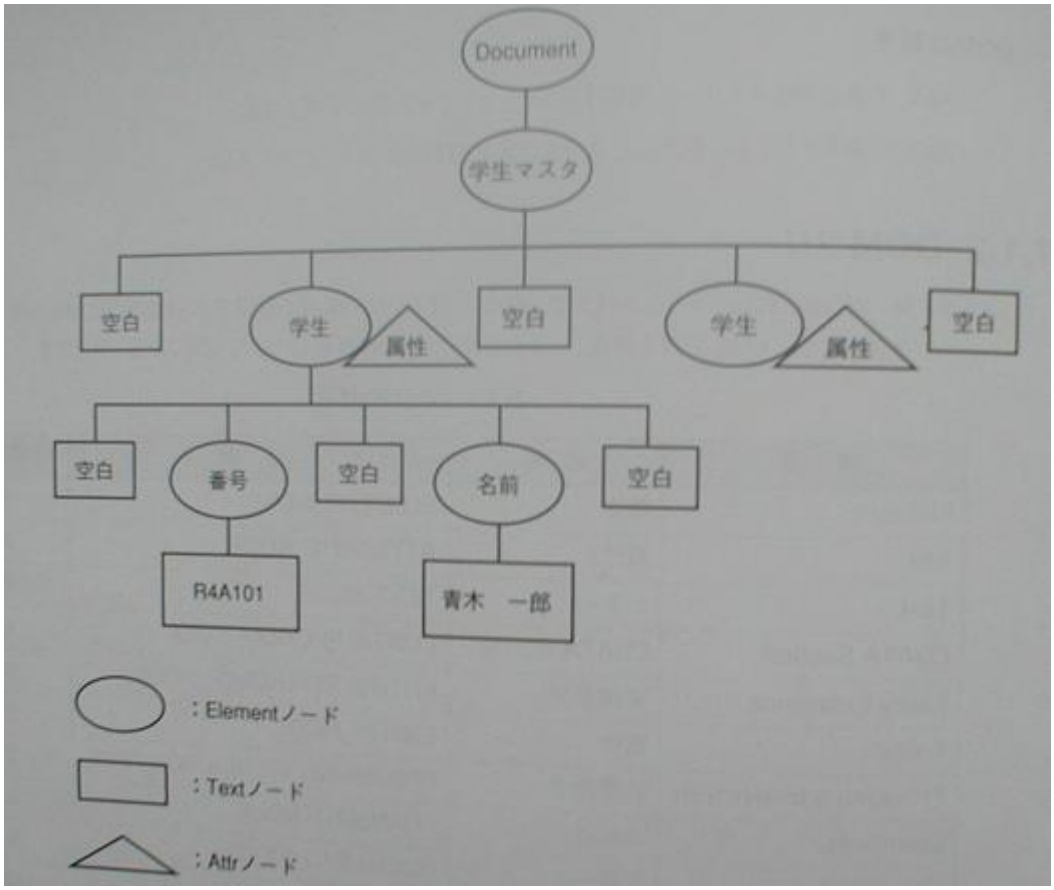


図4．1　DOM ツリーの構造モデル

空白文字、TAB 文字、改行文字だけの部分も Text ノードとして扱われるので注意が必要である。

ノードの親子関係

ノードには親子関係がある。ルートに近い方が親ノードで、親ノードの下のノードが子ノードである。

ノードの関係を次の表に示す。

表4．2　ノードの親子関係

種類	意味
parentNode	親ノード
childNode	子ノード
firstChild	最初の子ノード

lastChild	最後の子ノード
previousSibling	親が同じノードで前のノード
nextSibling	親が同じノードで次のノード

2. JavaScript による DOM の操作

W3C は DOM に対応したスクリプト言語の機能として「ECMA（ヨーロッパ電子計算機工業会）スクリプト言語」を定義している。Microsoft 社のインターネットエクスプローラの JavaScript は ECMA スクリプトに対応している。DOM 操作は、JavaScript を用いて行うことができる。

ここでは、「ボタン」をクリックしたら動作するプログラムを作成する。そして、実行結果を表示するために、処理結果を変数 strMsg に文字列として作成し、innerHTML を用いて文字列を表示する。

これを行う基本プログラムは、下記のようになる。

```
<html>
<head>
<script language="JavaScript">
function onClick() {
      var strMsg     = "";

      strMsg +=  処理結果;

      layer1.innerHTML = strMsg;
}
</script>
</head>
<body>
<form name="InputData">
   <input type="button" value="実行" onclick="onClick()">
</form>
<!-- 検索結果を表示する場所 -->
<div id="layer1"><!--この中が置き換わる--></div>
</body>
</html>
```

3. DOM プログラムの構造

JavaScript で DOM を操作するプログラムの基本構造を下記に示す。

ここでは、XML 文書として、「gakusei.XML」を使用する。

```
<html>
<head>
<script language="JavaScript">
function onClick() {
    var fnamexml = "";      // XML ファイル名
    var strMsg   = "";      //  検索結果、エラーメッセージ
    var objDoc   = null;   // DOM オブジェクト
    var objRoot  = null;   //  ルート要素
    var objErr   = null;   // XML 文書の書式エラー
    try {
        fnamexml        = "gakusei.xml";  //XML 文書のセット
        objDoc          = new ActiveXObject("MSXML2.DOMDocument");  //オブジェクト作成
        objDoc.async    = false;  //同期して読み込む
        objDoc.load(fnamexml);  //DOM に読み込む
        objErr          = objDoc.parseError;    //構造を検査する
        objRoot         = objDoc.documentElement;    //ルートノードを設定する
//処理プログラムを記述する
//  以降では、この部分に処理の記述を行う
    } catch (e) {           //objDoc.parseErro 処理でエラーが発生するとエラー処理を行う
        if (objErr.line == 0) {
            strMsg   = "JavaScript に誤りがあります。";
        } else {
            strMsg   = objErr.line + "行  " + objErr.srcText + "<br/>";//エラー行のセット
            strMsg += objErr.reason;      //エラー理由のセット
        }
    }
    objDoc.Close;       //DOM をクローズ
    objDoc = null;      //オブジェクトをクローズ
    layer1.innerHTML = strMsg;    //処理結果を表示する
}
</script>
</head>
<body>
```

```
<form name="InputData">
  <input type="button" value="実行" onclick="onClick()">    //ボタンにより実行開始
</form>
<!-- 検索結果を表示する場所 -->
<div id="layer1"><!--この中が置き換わる--></div>    //処理結果表示位置
</body>
</html>
```

図4．2　DOM 操作の基本プログラム

また、「gakusei.XML」は、次のようなものである。

```
<?xml  version="1.0"  encoding="Shift_JIS" ?>
<!-- 学生情報マスタ -->
<学生マスタ>
    <学生　学籍番号="02001"　学科="研究科">
        <番号>R4A101</番号>
        <名前>青木　一郎</名前>
        <証明写真>Photo02001.jpg</証明写真>
        <点数>80</点数>
        <資格>基本情報</資格>
        <資格>初級シスアド</資格>
        <資格>ソフト開発</資格>
        <url>xml1.ac.jp</url>
        <email>info@xml1.ac.jp</email>
        <自己紹介>
            自宅から学校までの 12km を約 40 分かけて
            <アピール>自転車</アピール>
            通学をしています。
        </自己紹介>
    </学生>
    <学生　学籍番号="02002"　学科="研究科">
        <番号>R4A201</番号>
        <名前>石田　二郎</名前>
        <証明写真>Photo02002.jpg</証明写真>
        <点数>70</点数>
        <資格>基本情報</資格>
```

```
        <資格>簿記３級</資格>
        <url>xml2.ac.jp</url>
        <email>info@xml2.ac.jp</email>
        <自己紹介>
            <アピール>読書</アピール>
            が趣味です。XML 関連の本が大好きです。
        </自己紹介>
    </学生>
    <学生　学籍番号="03001"　学科="専門科">
        <番号>S3A101</番号>
        <名前>上田　三郎</名前>
        <証明写真>Photo03001.jpg</証明写真>
        <点数>80</点数>
        <資格>基本情報</資格>
        <資格>初級シスアド</資格>
        <資格>ソフト開発</資格>
        <url>xml3.ac.jp</url>
        <email>info@xml3.ac.jp</email>
        <自己紹介>
            時間があると
            <アピール>映画</アピール>
            を見に行きます。最近見た映画では「今日も雨降り」に感動しました。
        </自己紹介>
    </学生>
    <学生　学籍番号="03002"　学科="専門科">
        <番号>S3A102</番号>
        <名前>遠藤　四郎</名前>
        <証明写真>Photo03002.jpg</証明写真>
        <点数>45</点数>
        <url>xml4.ac.jp</url>
        <email>info@xml4.ac.jp</email>
        <自己紹介>
            地元のサッカーチームの「ベアーズ札幌」を応援しています。高校時代は
            <アピール>サッカー</アピール>
            部で活躍していました。
        </自己紹介>
```

```
    </学生>
    <学生　学籍番号="04001"　学科="システム科">
        <番号>J2A101</番号>
        <名前>太田　五郎</名前>
        <証明写真>Photo04001.jpg</証明写真>
        <点数>60</点数>
        <資格>基本情報</資格>
        <資格>初級シスアド</資格>
        <url>xml5.ac.jp</url>
        <email>info@xml5.ac.jp</email>
        <自己紹介>
            通学の電車の中で
            <アピール>読書</アピール>
            は欠かせません。XML の問題が解けたときの満足感は何ものにも代えられません。
        </自己紹介>
    </学生>
</学生マスタ>
```

図4．3　　gakusei.XML

4．DOM の基本インタフェース

4．1　ルートノード名を取り出す

DOM ツリーを操作する JavaScript のメソッド

・ nodeName　：　ノード名を取り出す

「DOM 操作の基本プログラム」に次の命令を挿入する。

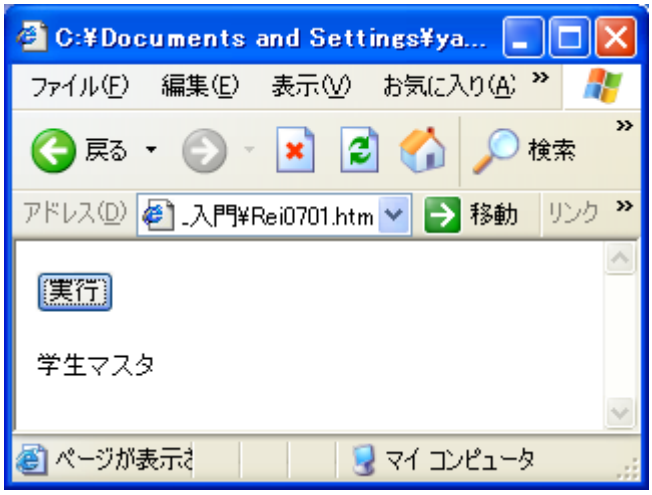
strMsg　=　objRoot.nodeName

Rei0701.htm

```
<html>
<head>
<script language="JavaScript">
function onClick() {
    var fnamexml = "";      // XML ファイル名
    var strMsg    = "";      //  検索結果、エラーメッセージ
    var objDoc    = null;   // DOM オブジェクト
    var objRoot   = null;   //  ルート要素
```

```
    var objErr     = null;   // XML 文書の書式エラー
    try {
        fnamexml          = "gakusei.xml";
        objDoc            = new ActiveXObject("MSXML2.DOMDocument");
        objDoc.async    = false;
        objDoc.load(fnamexml);
        objErr            = objDoc.parseError;
        objRoot           = objDoc.documentElement;
        strMsg = objRoot.nodeName;     //挿入した文;rootnode のノード名の取得
    } catch (e) {
        if (objErr.line == 0) {
            strMsg   = "JavaScript に誤りがあります。";
        } else {
            strMsg   = objErr.line + "行  " + objErr.srcText + "<br/>";
            strMsg += objErr.reason;
        }
    }
    objDoc.Close;
    objDoc = null;
    layer1.innerHTML = strMsg;       //メッセージの表示
}
</script>
</head>
<body>
<form name="InputData">
  <input type="button" value="実行" onclick="onClick()">//onClick()が実行される
</form>
<!-- 検索結果を表示する場所 -->
<div id="layer1"><!--この中が置き換わる--></div>
</body>
</html>
```

実行画面



4．2　子ノードを取り出す（Node による方法）

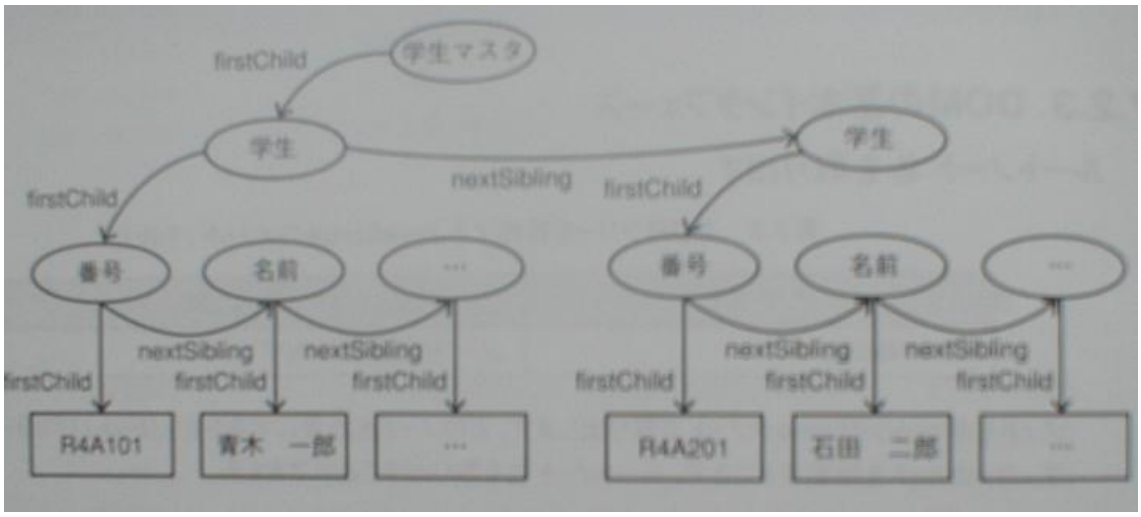


図4．4　子ノードを探索する手順

図4．4において、ルートノード「学生マスタ」の子ノード「学生」の最初のノードは firstChild メソッドを使って指定する。次の「学生」ノードは nextSibling メソッドで指定できる。

さらに、ノード「学生」の子ノード「番号」は firstChild メソッドを使って指定し、次のノード「名前」は nextSibling メソッドを使って順に指定できる。

表4．3に DOM ツリーを操作する JavaScript のメソッドを示す。

表4．3　DOM ツリーを操作する JavaScript のメソッド

メソッド	機能

firstChild	ノードの最初の子ノード
nextSibling	次の子ノード
nodeType	ノードの種類（表４．５）
nodeValue	ノードの内容

また、表４．４にノードの識別子を示す。文字列と数値表現が規定されている。

表４．４　ノードの識別子

種類	定義	数値表現
要素	Node.ELEMENT_NODE	1
属性	Node.ATTRIBUTE_NODE	2
テキスト	Node.TEXT_NODE	3

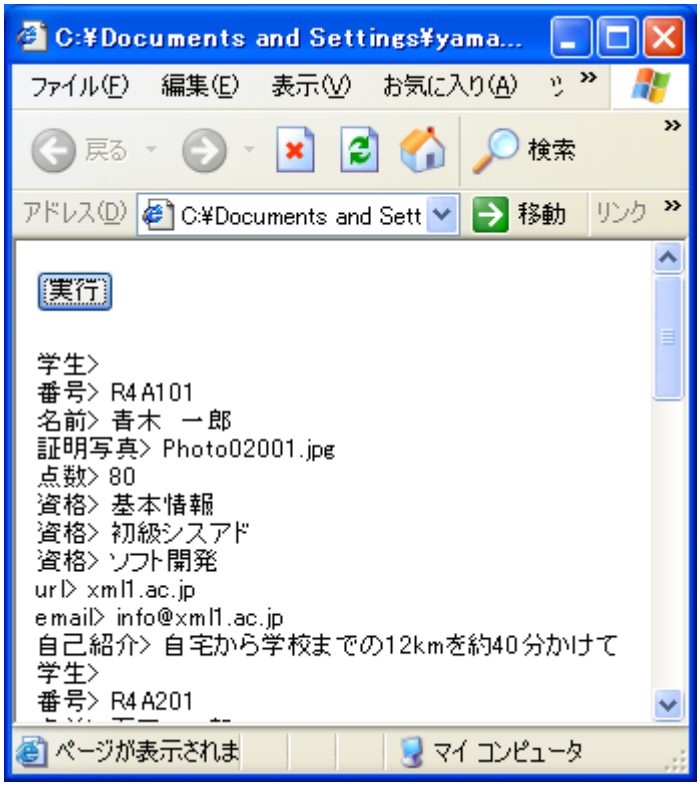
要素「学生」のすべての要素の要素名と内容を取り出すには、次のプログラムを追加する。

```
var gakuNode    = objRoot.firstChild;
while (gakuNode != null) {
    if (gakuNode.nodeType == 1) {    //1は要素を識別する数値表現
        strMsg += gakuNode.nodeName + "><br/>";

        var gakuNode2 = gakuNode.firstChild;
        while (gakuNode2 != null) {
            if (gakuNode2.nodeType == 1) {       //1は要素を識別する
              strMsg += gakuNode2.nodeName + "> ";  //要素名
              strMsg += gakuNode2.firstChild.nodeValue + "<br/>";  //要素の内容
             }
             gakuNode2 = gakuNode2.nextSibling;
        }
       gakuNode = gakuNode.nextSibling;
   }
 }
```

完成プログラムは、Rei0702.htm である。

実行結果



4．3　　属性を取り出す

属性を取り出すには、表4．5の attribute メソッドを使用する。

表4．5　DOM ツリーを操作する JavaScript のメソッド

メソッド	機　能
hasAttributes	ノードに属性があるかどうか調べる
attributes	属性を示す
length	属性の数を取り出す
nodeName	属性名を取り出す
value	属性値を取り出す

属性リストを取り出す書き方

var attrs =　要素のノード.attributes

ノードリスト　　ノードリストでの順番の指定

＜＞ ┬ ＜学籍番号＞　　attrs.item(0)

　　└ ＜学科＞　　attrs.item(1)

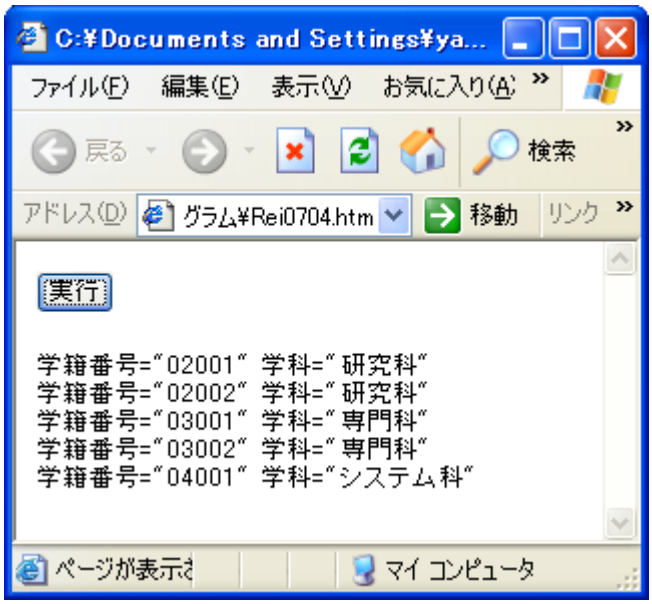
要素「学生」のすべての属性名と属性値を取り出す例

次のプログラムを追加する。

```
if (objRoot.hasChildNodes()) {
   var gakuNode = objRoot.childNodes;
   for (var i = 0; i < gakuNode.length; i++) {
       var gakuNode2 = gakuNode.item(i);
       if (gakuNode2.nodeType == 1) {
          var attrs       = gakuNode2.attributes;
          for (var j = 0; j < attrs.length; j++) {       //属性の数だけ繰り返す
              var attr = attrs.item(j);
              strMsg += " "     + attr.nodeName;     //属性名
              strMsg += "=¥"" + attr.value + "¥"";     //属性値
          }
          strMsg += "<br/>";                              //改行
       }
    }
} else {
    strMsg = "ノードがありません。";
}
```

完成プログラムは、Rei0704.htm である。

実行結果



5. 要素・属性の検索

ここでは、データを1つ入力して XML 文書から指定した要素・属性の検索をするように基本プログラムを変更・追加する。

```
<html>
<head>
<script language="JavaScript">
function onClick() {
    var fnamexml = "";      // XML ファイル名
    var strMsg   = "";      //  検索結果、エラーメッセージ
    var objDoc   = null;   // DOM オブジェクト
    var objRoot  = null;   //  ルート要素
    var objErr   = null;   // XML 文書の書式エラー

    var inStr1 = document.InputData.value1.value;   //入力した値を保存する変数

    try {

        fnamexml        = "gakusei.xml";

        objDoc          = new ActiveXObject("MSXML2.DOMDocument");
        objDoc.async    = false;
        objDoc.load(fnamexml);
        objErr          = objDoc.parseError;
        objRoot         = objDoc.documentElement;

    } catch (e) {
        if (objErr.line == 0) {
            strMsg   = "JavaScript に誤りがあります。";
        } else {
            strMsg   = objErr.line + "行  " + objErr.srcText + "<br/>";
            strMsg += objErr.reason;
        }
    }
    objDoc.Close;
    objDoc = null;

    layer1.innerHTML = strMsg;
```

```
}

function getText(element, tagName) {
      var list        = element.getElementsByTagName(tagName);
      var cElement = list.item(0);
      return    cElement.firstChild.nodeValue;
}
</script>
</head>
<body>
<form name="InputData">
  <table border="1">
    <tr>
      <td>入力</td>  //入力用ガイドメッセージ
      <td><input type="text" name="value1" size="50"></td>  //テキストボックス
    </tr>
  </table>
  <input type="button" value="実行" onclick="onClick()">
</form>
<!--  検索結果を表示する場所  -->
<div id="layer1"><!--この中が置き換わる--></div>
</body>
</html>
```

５．１　子ノードを取り出す

ノード名を指定して、その名前の子ノードを取り出すことができる。

JavaScript のメソッド

・getElementsByTagName(“ノード名”)　「ノード名」で示されるノードリストを取出す

１） 要素の内容を取り出す

次のプログラムを追加する。

```
var gakuNode    = objRoot.getElementsByTagName("学生");
for (var i = 0; i < gakuNode.length; i++) {
    var gakuNode2 = gakuNode.item(i);
    var bangou       = getText(gakuNode2, "番号");   //要素「番号」の内容を取り出す
    var namae        = getText(gakuNode2, "名前"); //要素「名前」の内容を取り出す
```

```
        strMsg += bangou + " " + namae + "<br/>";
}
```

プログラムは Rei0705.htm

実行結果

2） 属性値を取り出す

属性名を指定して属性値を取り出すには、getAttribute メソッドを用いる。

- **getAttribute(“属性名”)　「属性名」で指定した属性の属性値を取り出す**

次のプログラムを追加する。

```
var gakuNode   = objRoot.getElementsByTagName("学生");
for (var i = 0; i < gakuNode.length; i++) {
    var gakuNode2 = gakuNode.item(i);
    var gakuseki   = gakuNode2.getAttribute("学籍番号"); //属性「学生番号」の内容
    var namae       = getText(gakuNode2, "名前");
    strMsg += gakuseki + " " + namae + "<br/>";
}
```

プログラムは Rei0706.htm

実行結果

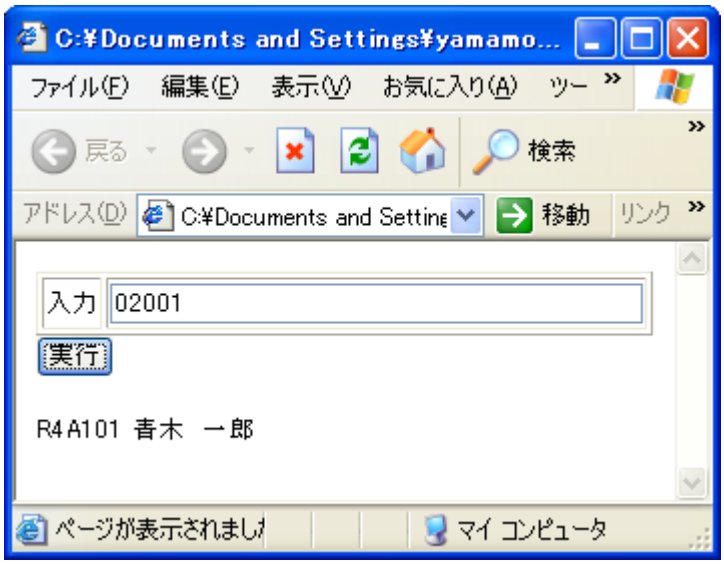
3)属性値「学籍番号」を指定して、対応する「番号」、「名前」の値を取り出す

次のプログラムを追加する。

```
var gakuNode  = objRoot.getElementsByTagName("学生");  //「学生」のノードリスト
var gakuNode2 = null;
var gakuseki  = "";
var fnd       = false;    //fnd:見つかったどうかを識別
for (var i = 0; i < gakuNode.length; i++) {
    gakuNode2 = gakuNode.item(i);
    gakuseki  = gakuNode2.getAttribute("学籍番号");  //学籍番号の属性値を取得
    if (gakuseki == inStr1) {      //入力値(inStr1)と比較
        fnd = true;
        break;
    }
}
if (fnd == false) {
    strMsg = inStr1 + "の学生はいません。";
} else {
   var bangou = getText(gakuNode2, "番号");
   var namae  = getText(gakuNode2, "名前");
   strMsg += bangou + " " + namae + "<br/>";
}
```

プログラムは Toi0705.htm

実行結果



3） 属性を追加・変更

属性を追加/変更するには、setAttribute メソッドを使用する。

・setAttribute（”属性名”、“属性値”）　　要素に属性を追加する

次のプログラムを追加する。

```
if (fnd == false) {                                  //inStr1  学籍番号
   strMsg = inStr1 + "の学生はいません。";     //inStr2  属性名
} else {                                             //inStr3  属性値
   gakuNode2.setAttribute(inStr2, inStr3);       //属性名、属性値の追加
   window.alert(gakuNode2.xml);                      //XML インスタンスの表示
}
```

プログラムは Rei0707.htm

実行結果

データの入力画面

実行結果の画面

4） 属性の削除

属性を削除するには、removeAttribute メソッドを使用する。

・removeAttribute（“属性名”）　　指定した属性（属性名と属性値）が削除される

次のプログラムを追加する。

```
if (fnd == false) {
    strMsg = inStr1 + "の学生はいません。";
} else {
    gakuNode2.removeAttribute(inStr2);          //属性「学科」を削除
```

```
    window.alert(gakuNode2.xml);          //XML インスタンスを表示
}
```

プログラムは Rei0708.htm

実行結果

データの入力画面

実行結果の画面