

JSTL入門: 核心(core)に触れる

カスタム・タグを用いたフロー制御とURL管理

Mark Kolb
Software Engineer

2003年 3月 18日

JSP標準タグ・ライブラリー (JSTL) のcore ライブラリーには、その名が示すとおり、反復や条件付けなどの基本操作のカスタム・タグのほかに、スコープ付き変数の管理やURLとのやり取りなどの基本的機能のカスタム・タグも用意されています。これらのタグは、ページ作成者が直接利用できるだけでなく、他のJSTLライブラリーと組み合わせることで、もっと複雑なプレゼンテーション・ロジックの土台を築くこともできます。Mark Kolb氏は、引き続きJSTLを調べ、core ライブラリー内のフロー制御とURL管理に役立つタグを中心に説明します。

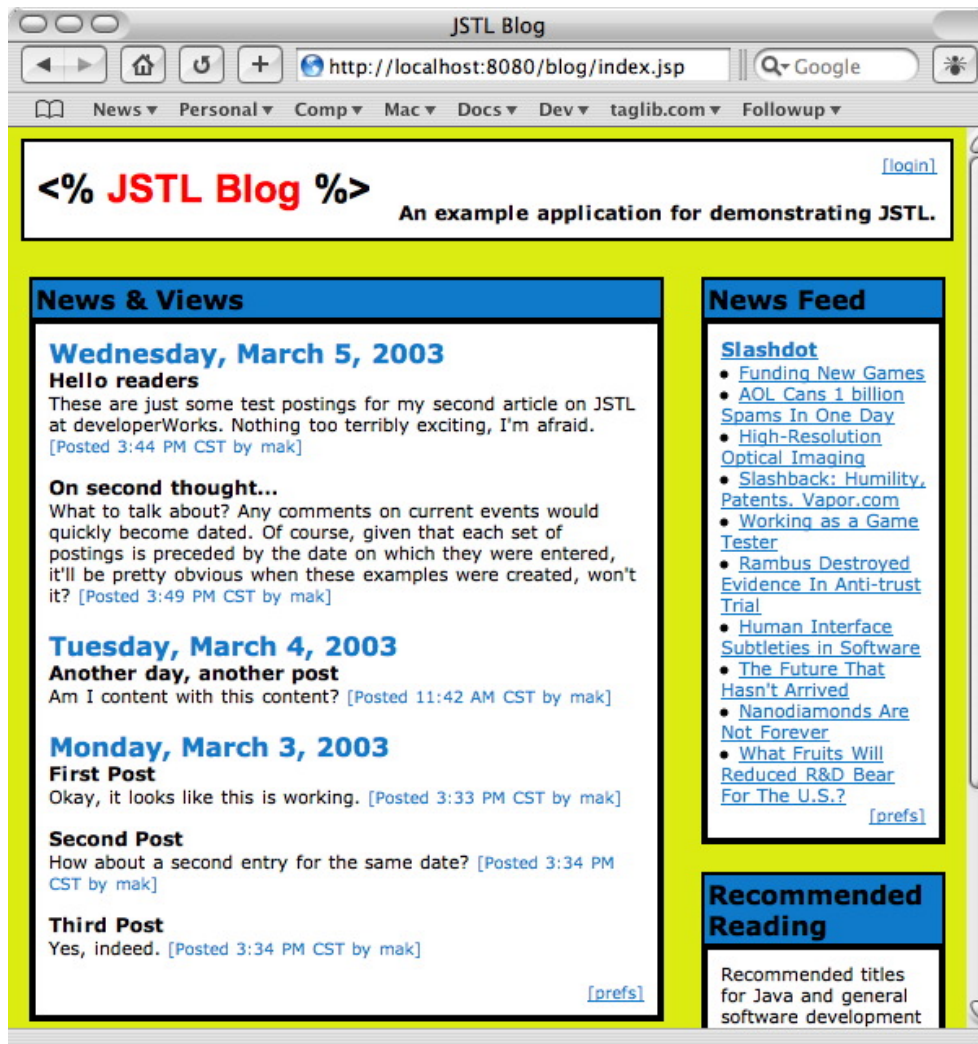
このシリーズの第1回では、JSTLの概要を学びました。式言語 (EL) を使用してデータにアクセスし、データを操作する方法を学習しました。お分かりのように、ELはJSTLカスタム・タグの属性に動的な値を割り当てるためのもので、標準装備のアクションや他のカスタム・タグ・ライブラリーの要求時属性値を指定するJSP式と同じ役割を果たします。

ELの使い方を示すため、core ライブラリーから<c:set>、<c:remove>、<c:out> の3つのタグを紹介しました。<c:set> と<c:remove> はスコープ付き変数を管理するタグで、<c:out> はデータ、特にELを使用して計算した値を表示するタグです。この基礎知識を元に、この記事では、大きく分けるとフロー制御とURL管理の2つのカテゴリーに分類できる、core ライブラリーのその他のタグを中心に説明します。

アプリケーション例

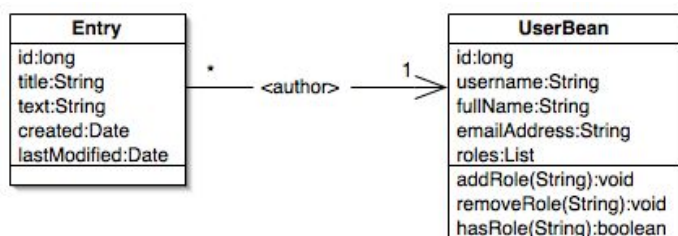
JSTLタグを示すため、このシリーズでこれから利用する作業アプリケーションから例を取り上げることにします。普及度と知名度の理由から、ここでは簡単なJavaベースのWeblogを使用します。このアプリケーションのJSPページとソース・コードのダウンロードについては、[参考文献](#)を参照してください。Weblog (blogとも言います) は、Weblogの作成者に関係があるテーマについての簡単なコメントからなるWebベースのジャーナルで、一般的にはWeb上の関連記事や関連ディスカッションへのリンクが張られています。図1に、実行しているアプリケーションのスクリーンショットを示します。

図1. Weblogアプリケーション



完全実装にはかなりの数のJavaクラスが必要ですが、プレゼンテーション層では、WeblogアプリケーションのEntryとUserBeanの2つのクラスしか参照されません。JSTLの例を理解する場合に重要なのは、これらの2つのクラスだけです。図2に、EntryとUserBeanのクラス図を示します。

図2. Weblogアプリケーションのクラス図



EntryクラスはWeblog内の日付付き項目を表します。id属性は、データベース内の項目の保管および検索に使用するのに対し、title属性とtext属性は、項目の実際のコンテンツを表します。Java言語のDateクラスの2つのインスタンスは、項目が最初に作成されたときを表すcreated

属性と、項目が最後に変更されたときを表す `lastModified` 属性で参照されます。 `author` 属性は `UserBean` インスタンスを参照して、項目の作成者を指定します。

`UserBean` クラスは、アプリケーションの認証済みユーザーに関する情報(ユーザー名、フルネーム、Eメール・アドレスなど)を保管します。また、このクラスには、関連データベースとやり取りする場合の `id` 属性も含まれます。最後の属性 `roles` は、 `String` 値のリストを参照し、対応ユーザーに関連付けられているアプリケーション固有の役割を指定します。Weblogアプリケーションの場合、関係のある役割は "User"(すべてのアプリケーション・ユーザーに共通のデフォルトの役割)と "Author" (Weblog項目の作成と編集が許可されるユーザーを指定する役割)です。

フロー制御

JSP式の代わりにELを使用して動的な属性値を指定することができるので、ページ作成者はスクリプト要素を使用する必要がなくなります。スクリプト要素はJSPページの保守問題の重要な原因になることがあるので、それに代わる簡単な(そして標準の)代替手段を提供することは、JSTLの大きなメリットです。

ELはJSPコンテナからデータを取得し、オブジェクト階層をトラバースして、その結果に対し簡単な操作を実行します。しかし、データへのアクセスと操作のほかに、JSPスクリプト要素には、フロー制御というもう1つの一般的な使い方があります。特に、ページ作成者がスクリプトレットに頼って、反復や条件付きコンテンツを実装することは、よくあることです。しかし、このような操作はELの能力を超えているので、代わりに `core` ライブラリーがカスタム・アクションを提供して、[反復](#)、[条件付け](#)、および[例外処理](#)の形でフロー制御を管理します。

反復

Webアプリケーションの場合、反復は、通常、リストまたはテーブル内の行のシーケンスの形で、主にデータのコレクションを取り出したり表示したりする場合に使用されます。反復コンテンツを実装する場合の基本JSTLアクションは `<c:forEach>` カスタム・タグです。このタグは、整数範囲内の反復 (Java言語の `for` ステートメントに相当) と、コレクション内の反復 (Java言語の `Iterator` クラスと `Enumeration` クラスに相当)の2つの異なるスタイルの反復をサポートしています。

整数範囲内で反復する場合は、リスト1に示す `<c:forEach>` タグの構文を使用します。 `begin` 属性と `end` 属性は、静的な整数値または整数値に評価される式のいずれかでなければなりません。それぞれ、反復で使用する索引の初期値と、反復を中止する索引値を指定します。 `<c:forEach>` を使用して整数範囲内で反復する場合は、これらの2つの属性が必須で、他の属性はすべてオプションです。

リスト1. `<c:forEach>` アクションを用いた数値反復の構文

```
<c:forEach var="name" varStatus="name"
  begin="expression" end="expression" step="expression">
  body content
</c:forEach>
```

`step` 属性も、指定されている場合は、整数値を取らなければなりません。各反復後に索引に追加する量を指定します。反復の索引は、 `begin` 属性の値を先頭として、 `step` 属性の値ずつ増分さ

れ、end 属性の値を超えると反復は中止されます。step 属性を省略した場合は、デフォルトとして1が設定されます。

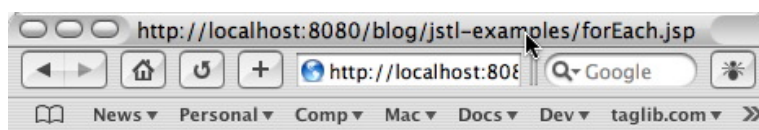
var 属性を指定した場合は、指定された名前を持つスコープ付き変数が作成され、反復が実行されるたびに索引の現在の値が代入されます。このスコープ付き変数の可視性にはネストが掛けられています。つまり、この変数には<c:forEach> タグの本体内でしかアクセスできません(後にオプションのvarStatus 属性の使用について説明します)。リスト2に、限られた一連の整数値内で反復する場合の<c:forEach> アクションの例を示します。

リスト2. <c:forEach> タグを使用して数値の範囲に対応するテーブル形式データを生成する

```
<table>
<tr><th>Value</th>
  <th>Square</th></tr>
<c:forEach var="x" begin="0" end="10" step="2">
  <tr><td><c:out value="{x}" /></td>
    <td><c:out value="{x * x}" /></td></tr>
</c:forEach>
</table>
```

このコード例では、図3に示すように最初の5つの偶数の二乗からなるテーブルが生成されます。ここでは、begin 属性とstep 属性の両方に値2を指定し、end 属性に値10を指定しています。さらに、var 属性を使用して、<c:forEach> タグの本体内で参照される、索引値を保管するためのスコープ付き変数を作成しています。具体的に言えば、2つの<c:out> アクションを使用して索引とその二乗(簡単な式を使って計算する)を表示します。

図3. リスト2の出力



Value	Square
2	4
4	16
6	36
8	64
10	100

コレクションのメンバー内を反復する場合は、<c:forEach> タグのもう1つの属性items を使用します(リスト3を参照)。この形式の<c:forEach> タグを使用する場合は、items 属性が唯一の必須属性となります。items 属性の値は、反復の実行対象であるメンバーが属するコレクションでなければなりません。一般に、これはEL式を使用して指定します。<c:forEach> タグのitems 属性で変数名を指定する場合は、指定された変数には、反復を実行するごとに連続的にコレクションの要素が入ります。

リスト3. <c:forEach> アクションを用いたコレクション内の反復の構文

```
<c:forEach var="name" items="expression" varStatus="name"
  begin="expression" end="expression" step="expression">
body content
</c:forEach>
```

Javaプラットフォームに用意されている標準のコレクション・タイプはすべて、<c:forEach> タグでサポートされています。さらに、このアクションを使用して、プリミティブの配列を含め、配列の要素内を反復することもできます。表1に、items 属性でサポートされる値の全リストを示します。表の最後の行から分かるように、JSTLでは、SQLクエリーの結果内を反復するための、独自のインターフェース `javax.servlet.jsp.jstl.sql.Result` が定義されています(この機能の詳細については、次回以降の記事で改めて紹介します)。

表1. <c:forEach> タグのitems 属性でサポートされるコレクション

items の値	結果的に items に入る値
<code>java.util.Collection</code>	<code>iterator()</code> 呼び出しからの要素
<code>java.util.Map</code>	<code>java.util.Map.Entry</code> のインスタンス
<code>java.util.Iterator</code>	反復子の要素
<code>java.util.Enumeration</code>	列挙の要素
<code>Object</code> インスタンスの配列	配列の要素
プリミティブ値の配列	ラップされた配列の要素
コンマを区切り文字とするString	サブストリング
<code>javax.servlet.jsp.jstl.sql.Result</code>	SQLクエリーからの行

begin 属性、end 属性、step 属性を使用して、反復に含めるコレクションの要素を制限することができます。<c:forEach> による数値反復の場合と同様、コレクションの要素内を反復するときも反復索引は保持されます。実際に<c:forEach> タグで処理されるのは、begin、end、step に指定した値と一致する索引値に対応する要素だけです。

リスト4に、コレクション内を反復する場合に使用する<c:forEach> タグを示します。このJSPコードでは、スコープ付き変数 `entryList` は `Entry` オブジェクトのリスト (具体的に言えば、`ArrayList`) に設定されています。<c:forEach> タグは、このリストの各要素を順番に処理し、`blogEntry` というスコープ付き変数に代入して、2つのテーブル行 (1つはWeblog項目のtitle 用、もう1つはtext 用) を生成します。これらのプロパティは、対応するEL式を持つ2つの<c:out> アクションを用いて `blogEntry` 変数から取得します。Weblog項目のタイトルとテキストの両方にHTMLマークアップが入っている可能性があるので、どちらの<c:out> タグの `escapeXml` 属性もfalseに設定されていることに注意してください。図4に、結果を示します。

リスト4. <c:forEach> タグを使用して指定日のWeblog項目を表示する

```
<table>
<c:forEach items="${entryList}" var="blogEntry">
  <tr><td align="left" class="blogTitle">
<c:out value="${blogEntry.title}" escapeXml="false"/>
  </td></tr>
  <tr><td align="left" class="blogText">
<c:out value="${blogEntry.text}" escapeXml="false"/>
  </td></tr>
</c:forEach>
</table>
```

図4. リスト4の出力



<c:forEach> の最後の属性であるvarStatus は、整数内の反復でも、コレクション内の反復でも、同じ役割を果たします。var 属性と同様、varStatus もスコープ付き変数を作成するためのものです。ただし、現在の索引値や現在の要素を保管するのではなく、varStatus 属性で指定された変数には、javax.servlet.jsp.jstl.core.LoopTagStatus クラスのインスタンスが代入されます。このクラスは、反復の現在の状態を表す一連のプロパティ(表2を参照)を定義するものです。

表2. LoopTagStatusオブジェクトのプロパティ

プロパティ	Getter	説明
current	getCurrent()	現在の反復に対する項目(コレクションから取得)
index	getIndex()	現在の反復に対するゼロベースの索引
count	getCount()	現在の反復に対する1ベースのカウント
first	isFirst()	現在の反復が最初の実行かどうかを示すフラグ
last	isLast()	現在の反復が最後の実行かどうかを示すフラグ
begin	getBegin()	begin 属性の値
end	getEnd()	end 属性の値
step	getStep()	step 属性の値

リスト5に、`varStatus` 属性の使用例を示します。リスト4のコードを変更して、タイトルを表示するテーブル行にWeblog項目の番号を追加します。これは、`varStatus` 属性の値を指定して、結果として生成されるスコープ付き変数の`count` プロパティにアクセスすることで行います。この結果を図5に示します。

リスト5. `varStatus`属性を使用してWeblog項目の番号を表示する

```
<table>
<c:forEach items=
    "${entryList}" var="blogEntry" varStatus="status">
    <tr><td align="left" class="blogTitle">
<c:out value="${status.count}"/>.
    <c:out value="${blogEntry.title}" escapeXml="false"/>
    </td></tr>
    <tr><td align="left" class="blogText">
<c:out value="${blogEntry.text}" escapeXml="false"/>
    </td></tr>
</c:forEach>
</table>
```

図5. リスト5の出力



`<c:forEach>` のほかに、`core` ライブラリーにはもう1つの反復タグ`<c:forTokens>` が用意されています。このJSTLのカスタム・アクションは、Java言語の`StringTokenizer` クラスに相当します。`<c:forTokens>` タグ (リスト6を参照) には、コレクション型バージョンの`<c:forEach>` と同じ属性セットのほかに、もう1つ属性があります。`<c:forTokens>` では、トークン化するストリングは`items` 属性を用いて指定し、トークンを生成する場合に使用する区切り文字セットは、`delims` 属性を用いて指定します。`<c:forEach>` の場合と同様に、`begin` 属性、`end` 属性、`step` 属性を使用して、指定した索引値に一致するトークンだけを処理するように制限することができます。

リスト6. `<c:forTokens>` アクションを用いてストリングのトークン内を反復する場合の構文

```
<c:forTokens var="name" items="expression"
    delims="expression" varStatus="name"
    begin="expression" end="expression" step="expression">
body content
</c:forTokens>
```

条件付け

動的コンテンツが入っているWebページでは、ユーザーのカテゴリごとに異なるコンテンツを表示できなければなりません。たとえば、Weblogの場合、ビジターには項目の閲覧とフィードバックの発信を認めますが、新しい項目の記入や既存のコンテンツの編集は許可ユーザーにしか認めないようにする必要があります。

利便性とソフトウェアの保守性はどちらも、このような機能を同じJSPページ内に実装してから、条件付けロジックを使用して、要求ごとに表示コンテンツを制御することで向上させることができます。core ライブラリーには、これらの機能を実装する `<c:if>` と `<c:choose>` という2つの異なる条件付けタグが用意されています。

2つのアクションを比べると `<c:if>` の方が簡単です。このタグは、単に単一のテスト式を評価して、式がtrue の場合のみ、その本体コンテンツを処理します。それ以外の場合は、タグの本体コンテンツは無視されます。リスト7に示すように、`<c:if>` は、`var` 属性と `scope` 属性を介して、必要に応じてテストの結果をスコープ付き変数に代入することができます(これらの属性の役割は `<c:set>` の場合と同じです)。この機能は、テストのコストが高い場合に特に有効です。テスト結果をスコープ付き変数にキャッシュしておき、後で `<c:if>` またはその他のJSTLタグを呼び出して取得することができます。

リスト7. `<c:if>` 条件付けアクションの構文

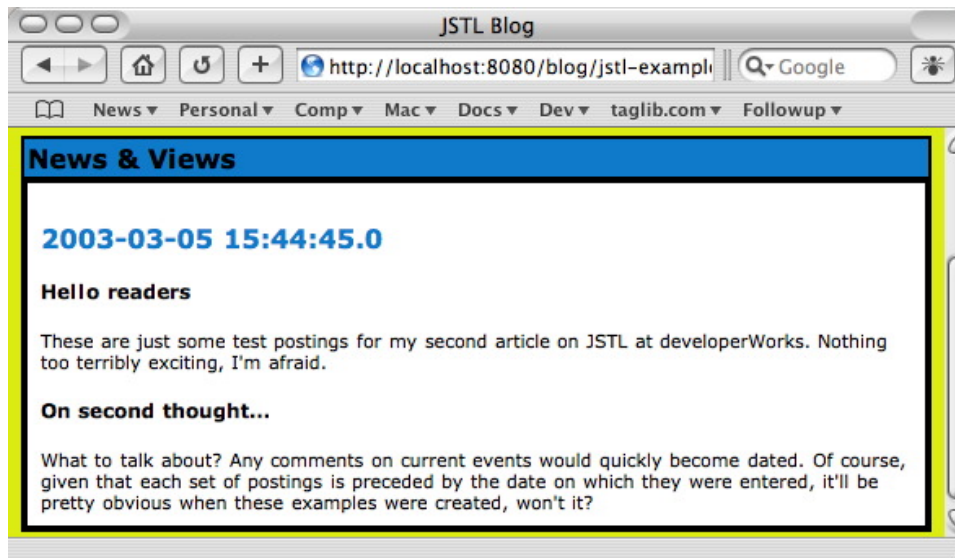
```
<c:if test="expression" var="name" scope="scope">
body content
</c:if>
```

リスト8に、`<c:if>` と `<c:forEach>` タグの `LoopTagStatus` オブジェクトの `first` プロパティを併用した場合を示します。この場合は、図6に示すように、一連のWeblog項目の作成日付は、その日付に最初に対応する項目の真上に表示され、それ以外の項目の前では繰り返されません。

リスト8. `<c:if>` を使用してWeblog項目の日付を表示する

```
<table>
<c:forEach items=
  "${entryList}" var="blogEntry" varStatus="status">
  <c:if test="${status.first}">
    <tr><td align="left" class="blogDate">
<c:out value="${blogEntry.created}"/>
    </td></tr>
  </c:if>
  <tr><td align="left" class="blogTitle">
<c:out value="${blogEntry.title}" escapeXml="false"/>
    </td></tr>
  <tr><td align="left" class="blogText">
<c:out value="${blogEntry.text}" escapeXml="false"/>
    </td></tr>
  </c:forEach>
</table>
```


図6. リスト8の出力



リスト8に示すように、`<c:if>` タグでは、条件付けするコンテンツの単純なケースを非常にコンパクトに表記することができます。また、表示するコンテンツを判別するには相互に排他的なテストが必要なケースに備えて、JSTLのcore ライブラリーには`<c:choose>` アクションも用意されています。リスト9に、`<c:choose>` の構文を示します。

リスト9. `<c:choose>` アクションの構文

```
<c:choose>
  <c:when test="expression">
body content
  </c:when>
  ...
  <c:otherwise>
body content
  </c:otherwise>
</c:choose>
```

テスト対象の各条件は、対応する`<c:when>` タグで表します。このタグは、少なくとも1つは存在しなければなりません。test が true と評価された最初の`<c:when>` タグの本体コンテンツのみが処理されます。どの`<c:when>` テストも true を戻さなかった場合は、`<c:otherwise>` タグの本体コンテンツが処理されます。ただし、`<c:otherwise>` タグはオプションであることに注意してください。`<c:choose>` タグが持つことができるネスト`<c:otherwise>` タグは、多くても1つです。`<c:when>` テストがすべて false で、`<c:otherwise>` アクションが指定されていない場合、`<c:choose>` の本体コンテンツは処理されません。

リスト10に、`<c:choose>` タグの使用例を示します。ここでは、要求オブジェクトから (EL の pageContext 暗黙オブジェクトを用いて) プロトコル情報を取得し、単純なストリング比較を使用してテストします。これらのテストの結果に基づき、対応するテキスト・メッセージが表示されます。

リスト10. <c:choose> を使用したコンテンツの条件付け

```
<c:choose>
  <c:when test="${pageContext.request.scheme eq 'http'}">
    This is an insecure Web session.
  </c:when>
  <c:when test="${pageContext.request.scheme eq 'https'}">
    This is a secure Web session.
  </c:when>
  <c:otherwise>
    You are using an unrecognized Web protocol. How did this happen?!
  </c:otherwise>
</c:choose>
```

例外処理

最後のフロー制御タグは<c:catch> です。このタグでは、JSPページの中で基本的な例外処理を行うことができます。もっと具体的に言えば、このタグの本体コンテンツ内で発生した例外はすべてキャッチされ、無視されます (つまり、標準のJSPエラー処理機構は呼び出されません)。ただし、例外が発生した場合に<c:catch> タグのオプション属性であるvar が指定されていると、その例外は指定の変数 (ページ・スコープ付き) に代入されて、そのページの中でカスタムの例外処理を実行することができます。リスト11に、<c:catch> 構文を示します (例については、[リスト18](#)を参照してください)。

リスト11. <c:catch> アクションの構文

```
<c:catch var="name">
body content
</c:catch>
```

URLアクション

JSTLのcore ライブラリーの残りのタグはURL関係です。まず、URLを生成するには、名前どおりの<c:url> タグを使用します。特に、<c:url> は、J2EE WebアプリケーションのURLを作成する場合に特に重要な3つの機能を提供します。

- 現在のサーブレット・コンテキストの名前を前に付加する
- セッション管理でURLを再作成する
- 要求パラメーターの名前と値をURLでエンコードする

リスト12に、<c:url> タグの構文を示します。基本URLはvalue 属性を使用して指定します。これは、タグにより必要に応じて変換されます。この基本URLの先頭がスラッシュの場合、サーブレット・コンテキスト名が前に付加されます。明示的なコンテキスト名は、context 属性を使用して指定することができます。この属性を省略すると、現在のサーブレット・コンテキストの名前が使用されます。サーブレット・コンテキストの名前は、開発時ではなく、デプロイメント時に決めるため、これは特に有効です (基本URLの先頭がスラッシュではない場合は、コンテキスト名の追加を必要としない相対URLと見なされます)。

リスト12. <c:url> アクションの構文

```
<c:url value="expression" context="expression"
      var="name" scope="scope">
  <c:param name="expression" value="expression"/>
  ...
</c:url>
```

URLの再作成は、`<c:url>` アクションにより自動的に実行されます。JSPコンテナが、ユーザーの現在のセッションIDが保管されているcookieを検出した場合は、再作成は必要ありません。ただし、このようなcookieが存在しない場合は、`<c:url>` で生成されたURLはすべて再作成されてセッションIDがエンコードされます。適切なcookieが後続の要求で検出されると、`<c:url>` は、URLを再作成してこのIDを取り入れる処理を中止することに注意してください。

`var` 属性に値を指定した(必要に応じて、`scope` 属性の対応する値も一緒に指定した) 場合、生成されたURLは、指定されたスコープ付き変数の値として代入されます。それ以外の場合、生成されたURLは現在の`JspWriter`を使用して出力されます。結果を直接出力するこの機能を使用することで、`<c:url>` タグを、たとえば、リスト13のようにHTML`<a>` タグの`href` 属性の値として指定することができます。

リスト13. HTMLタグの属性値としてURLを生成する

```
<a href="<c:url value='/content/sitemap.jsp' />">View sitemap</a>
```

最後に、要求パラメーターをネスト`<c:param>` タグを介して指定した場合、それらのパラメーターの名前と値は、HTTP GET要求の標準表記に従って、生成されたURLに付加されます。さらに、URLエンコードが実行されます。つまり、これらのパラメーターの名前または値のいずれかに、有効なURLを生成するために変換しなければならない文字が含まれている場合、それらの文字は適切に変換されます。リスト14に、`<c:url>` の振る舞いを示します。

リスト14. 要求パラメーターを用いてURLを生成する

```
<c:url value="/content/search.jsp">  
  <c:param name="keyword" value="#{searchTerm}" />  
  <c:param name="month" value="02/2003" />  
</c:url>
```

リスト14のJSPコードは`blog` というサーブレット・コンテキストにデプロイされ、スコープ付き変数`searchTerm` の値は`"core library"` に設定されます。セッションcookieが検出された場合、リスト14で生成されたURLはリスト15のようになります。コンテキスト名が前に付加され、要求パラメーターが付加されることに注意してください。さらに、`keyword` パラメーターの値に含まれるスペースと、`month` パラメーターの値に含まれるスラッシュが、HTTP GETパラメーターの要求に応じてエンコードされています(具体的に言えば、スペースは`+` に変換され、スラッシュはシーケンス`%2F` に変換されています)。

リスト15. セッションcookieが検出された場合のURL生成

```
/blog/content/search.jsp?keyword=foo+bar&month=02%2F2003
```

セッションcookieが検出されない場合、URLはリスト16のようになります。前にも説明したように、サーブレット・コンテキストが前に付加され、URLエンコード済みの要求パラメーターが付加されます。ただし、さらに、基本URLはセッションIDの仕様が組み込まれるように再作成されます。ブラウザが、このように再作成されたURLの要求を送信すると、JSPコンテナは自動的にセッションIDを抽出して、対応するセッションにこの要求を関連付けます。このように、J2EEアプリケーションは、セッション管理を必要とする場合も、アプリケーションのユーザーから渡されるcookieに依存する必要がありません。

リスト16. セッションcookieがない場合のURL生成

```
/blog/content/search.jsp;jsessionid=233379C7CD2D0ED2E9F3963906DB4290
?keyword=foo+bar&month=02%2F2003
```

コンテンツのインポート

JSPには、各種URLからJSPページにコンテンツを組み込むための標準機構が2つ用意されています。include 指示子と<jsp:include> アクションです。ただし、どちらの場合も、組み込むコンテンツは、そのページと同じWebアプリケーション(またはサーブレット・コンテキスト)の一部でなければなりません。これらの2つのタグの主な違いは、include 指示子はページ・コンパイルの際に組み込みコンテンツを取り入れるに対し、<jsp:include> アクションはJSPページの要求時処理の際に取り入れます。

core ライブラリーの<c:import> アクションは本質的に、<jsp:include> の汎用性が高く、強力なバージョンです(言ってみれば、巨大な<jsp:include> です)。<jsp:include> と同様、<c:import> は要求時のアクションで、その基本タスクは他のWebリソースのコンテンツをJSPに挿入することです。構文は、リスト17に示すように、<c:url> の構文とよく似ています。

リスト17. <c:import> アクションの構文

```
<c:import url="expression" context="expression"
  charEncoding="expression" var="name" scope="scope">
  <c:param name="expression" value="expression"/>
  ...
</c:import>
```

インポートするコンテンツのURLは、url 属性を用いて指定します。この属性は、<c:import> の唯一の必須属性です。相対URLも指定でき、現在のページのURLを基準として解決されます。ただし、url 属性の値の先頭がスラッシュの場合は、ローカルJSPコンテナ内の絶対URLとして解釈されます。context 属性の値が指定されていない場合、このような絶対URLは現在のサーブレット・コンテキスト内のリソースを参照するものと見なされます。context 属性を用いて明示的なコンテキストが指定されている場合は、絶対(ローカル)URLは指定されたサーブレット・コンテキストを基準に解決されます。

ただし、<c:import> アクションは、ローカル・コンテンツへのアクセスに限定されるものではありません。プロトコルやホスト名を含む、完全なURIを、url 属性の値として指定することもできます。事実、プロトコルはHTTPだけではありません。java.net.URL クラスでサポートされるすべてのプロトコルを、<c:import> のurl 属性の値として指定することができます。リスト18に、この機能を示します。

ここでは、<c:import> アクションを使用して、FTPプロトコルを介してアクセスする文書のコンテンツを組み込みます。さらに、<c:catch> アクションを使用して、FTPファイル転送時に発生するエラーをローカルで処理します。この場合は、<c:catch> のvar 属性を使用して例外用のスコープ付き変数を指定してから、<c:if> を使用してその値をチェックします。例外が発生すると、スコープ付き変数への代入が行われます。リスト18のEL式から分かるように、値は空にはなりません。FTP文書を取得できないので、その結果に対するエラー・メッセージが表示されます。

リスト18. <c:import> と <c:catch> の組み合わせ例

```
<c:catch var="exception">
  <c:import url="ftp://ftp.example.com/package/README"/>
</c:catch>
<c:if test="${not empty exception}">
  Sorry, the remote content is not currently available.
</c:if>
```

<c:import> アクションの最後の2つの (オプション)属性はvar とscope です。var 属性を指定すると、指定URLから取り出したコンテンツは、現在のJSPページに組み込まれるのではなく、スコープ付き変数に (String 値として) 保管されます。scope 属性は、この変数のスコープ設定を制御し、デフォルトではページ・スコープに設定します。今後の記事で紹介しますが、文書全体をスコープ付き変数に保管する<c:import> のこの機能は、JSTLのxml ライブラリーのタグで利用されます。

また、(オプションの) ネスト<c:param> タグを使用して、インポートするURLの要求パラメーターを指定することもできます。<c:url> のネスト<c:param> タグの場合と同様に、パラメーターの名前と値が必要に応じてURLにエンコードされます。

要求のリダイレクト

最後に説明するcore ライブラリー・タグは<c:redirect> です。このJSTLアクションは、ユーザーのブラウザーにHTTPリダイレクト応答を送信するもので、javax.servlet.http.HttpServletResponse のsendRedirect() メソッドに相当します。このタグのurl 属性とcontext 属性の振る舞いは、リスト19に示すように、ネスト<c:param> タグの結果である、<c:import> のurl 属性とcontext 属性の振る舞いと同じです。

リスト19. <c:redirect>アクションの構文

```
<c:redirect url="expression" context="expression">
  <c:param name="expression" value="expression"/>
  ...
</c:redirect>
```

リスト20に、<c:redirect> アクションを示します。このアクションは、リスト18のエラー・メッセージを指定エラー・ページへのリダイレクトに置き換えます。この例で、<c:redirect> タグの使い方は、標準の<jsp:forward> アクションと同じです。ただし、要求ディスパッチャーを介する転送はサーバー側で実装されるのに対し、リダイレクトはブラウザーで実行されることに注意してください。開発者の観点から見ると、効率性の面ではリダイレクトより転送の方が優れていますが、<jsp:forward> は現在のサーブレット・コンテキスト内の他のJSPページにしかディスパッチできないので、柔軟性の面では<c:redirect> アクションの方が多少優位です。

リスト20. 例外の応答におけるリダイレクト

```
<c:catch var="exception">
  <c:import url="ftp://ftp.example.com/package/README"/>
</c:catch>
<c:if test="${not empty exception}">
  <c:redirect url="/errors/remote.jsp"/>
</c:if>
```

ユーザーの観点から見た主な違いは、リダイレクトの場合にはブラウザで表示されるURLが更新されるので、ブックマークの設定に影響する、ということです。一方、転送の場合は、エンド・ユーザーが意識することはありません。`<c:redirect>` と `<jsp:forward>` の選択は、要求されるユーザー経験によっても異なります。

まとめ

JSTLのcore ライブラリーには、幅広いJSP開発者の要求を満たす、汎用のカスタム・タグが各種用意されています。たとえば、`<jsp:include>` アクションや`<jsp:forward>` アクション、`include` 指示子、`page` 指示子の`errorpage` 属性など、URLタグと例外処理タグは、既存のJSP機能を見事に補足しています。反復アクションと条件付けアクションでは、特に変数タグ (`<c:set>` と `<c:remove>`) とELを組み合わせて、スクリプト要素を必要することなく複雑なプレゼンテーション・ロジックを実装することができます。

著者について

Mark Kolb

Mark Kolbは、テキサス州オースチンで働くソフトウェア・エンジニアです。サーバー・サイドのJavaトピックについての講演多数のほか、[Web Development with JavaServer Pages, 2nd Edition](#) の共著者でもあります。

© Copyright IBM Corporation 2003

(www.ibm.com/legal/copytrade.shtml)

商標

(www.ibm.com/developerworks/jp/ibm/trademarks/)