JSTL入門: 式言語

スクリプト要素を回避することによってJSPアプリケーションのソフトウェア保守を単純化する

Mark Kolb
Software Engineer

2003年 2月 01日

JSP標準タグ・ライブラリー (JSTL) はカスタム・タグ・ライブラリーのコレクションで、反復、条件付け、データ管理フォーマット設定、XML操作、データベース・アクセスなど、Webアプリケーションに共通の汎用機能を実装するものです。developerWorksのこの新シリーズの第1回として、ソフトウェア・エンジニアのMark Kolbは、JSPページでスクリプト要素の使用を避けるためにJSTLタグを使用する方法を示します。また、プレゼンテーション層からソース・コードを除去することによってソフトウェア保守を単純化する方法についても学びます。最後に、JSTLの単純化された式言語について学びます。これにより、本格的なプログラム言語を使用せずにJSTLアクションに動的な属性値を指定することができます。

JavaServer Pages (JSP) テクノロジーは、J2EEプラットフォームにおける標準的なプレゼンテーション層のテクノロジーです。JSPテクノロジーは、動的にページ・コンテンツを生成するための計算を実行する、スクリプト要素とアクションの両方を提供します。スクリプト要素を使用すると、ユーザー要求に応えてページがレンダリングされる際に実行されるように、プログラムのソース・コードをJSPページに組み込むことができます。アクションは、一般にJSPページのテンプレート・テキストを構成するHTMLやXMLマークアップによく似たタグの中に、計算演算子をカプセル化するものです。JSP仕様で標準として定義されたアクションは数えるほどしかありませんが、JSP 1.1からは、カスタム・タグ・ライブラリーの形で開発者が独自のアクションを作成できるようになっています。

JSP標準タグ・ライブラリー (JSTL) は、幅広いサーバー・サイドJavaアプリケーションに共通の基本機能を実装する、JSP 1.2のカスタム・タグ・ライブラリーのコレクションです。JSTLは、データ・フォーマット設定、反復や条件付きコンテンツなどの一般的なプレゼンテーション層タスクの標準実装を提供することによって、JSPの作成者がこれらの一般操作を「一から再開発する」のではなく、アプリケーション固有の開発ニーズに注力できるようにします。

もちろん、スクリプトレット、式、宣言などのJSPスクリプト要素を使用してこのようなタスクを実装することもできます。例として、条件付きコンテンツは、リスト1に強調表示されている3つのスクリプトレットを使用して実装することができます。しかし、それらはページ内に埋め込まれるプログラムのソース・コード (通常はJavaコード) に依存するため、スクリプト要素はそれら

を使用するJSPページのソフトウェア保守タスクを非常に複雑にしてしまいがちです。例えば、リスト1のスクリプトレット例は、中括弧の正しいマッチングに大きく依存します。条件付きコンテンツ内にさらなるスクリプトレットをネストさせると、うっかり構文エラーを発生させた場合に大混乱が生じるおそれがあります。また、ページがJSPコンテナーによりコンパイルされたときに、結果のエラー・メッセージを理解するのが非常に困難な場合もあります。

リスト1. スクリプトレットを通じて条件付きコンテンツを実装する

```
<% if (user.getRole() == "member")) { %>
    <p>Welcome, member!</p>
<% } else { %>
    <p>Welcome, guest!</p>
<% } %>
```

このような問題を解決するためには、一般にかなりのプログラミング経験が必要になります。通常、JSPページ内のマークアップはページ・レイアウトやグラフィック・デザインに長けたデザイナーによって開発、保守されるのに対し、同じページのスクリプト要素には、問題が 生じた場合にプログラマーの介在が必要となります。このように単一ファイル内のコードに対し共同で責任を負うと、JSPページの開発、デバッグ、拡張が厄介なものになってしまいます。JSTLは、共通機能をカスタム・タグ・ライブラリーの標準化セットにパッケージ化することによって、JSP作成者からスクリプト要素の必要性を削減または除去し、関連する保守コストを回避してくれます。

JSTL 1.0

2002年6月にリリースされたJSTL 1.0は、4つのカスタム・タグ・ライブラリー (`core`、`format`、`xml`、および`sql`) と、1対の汎用タグ・ライブラリー・バリデーター (`ScriptFreeTLV` と`PermittedTaglibsTLV`) から成ります。`core` タグ・ライブラリーは、スコープ付き変数を通じてデータを管理したり、ページ・コンテンツの反復や条件付けを実行したりするための、カスタム・アクションを提供します。また、URLの生成や操作を行うためのタグも提供します。`format` タグ・ライブラリーは、その名が示すとおり、データのフォーマット設定 (具体的には数字や日付) のためのアクションを定義します。また、ローカライズされたリソース・バンドルを使用してJSPページの国際化サポートも提供します。`xml` ライブラリーには、XMLを通じて表されるデータを操作するためのタグが含まれており、`sql` ライブラリーはリレーショナル・データベースを照会するためのアクションを定義します。

2つのJSTLタグ・ライブラリー・バリデーターは、開発者がJSPアプリケーション内でコーディング標準を強制できるようにするものです。`ScriptFreeTLV` バリデーターを構成して、スクリプトレット、式、宣言など、さまざまなタイプのJSPスクリプト要素をJSPページ内で使用することを禁止することができます。同様に、`PermittedTaglibsTLV` バリデーターは、あるアプリケーションのJSPページによってアクセスされる可能性のあるカスタム・タグ・ライブラリー (JSTLタグ・ライブラリーを含む) のセットを制限するために使用できます。

JSTLはJ2EEプラットフォームの必要コンポーネントに、いずれはなるにもかかわらず、これが組み込まれているアプリケーション・サーバーは今のところごく少数です。Apache Software FoundationのJakarta Taglibプロジェクトの一部として、JSTL 1.0の参考実装が入手できます (参考文献 を参照)。JSTLのサポートを追加するために、参考実装のカスタム・ タグ・ライブラリーを、JSP 1.2およびServlet 2.3仕様をサポートする任意のアプリケーション・サーバーに組み込むことができます。

式言語

JSP 1.2では、静的文字列または (許可されている場合は) 式のどちらかを使用して、JSPアクションの属性を指定することができます。例として、リスト2では、`<jsp:setProperty>` アクションの`name` および`property` 属性には静的な値が指定され、`value` 属性の指定には式が使用されています。このアクションには、名前の指定されたBeanプロパティーに要求パラメーターの現行値を代入する効果があります。この形で使用された式は要求時属性値 と呼ばれ、属性値を動的に指定するためにJSP仕様に組み込まれている唯一のメカニズムです。

リスト2. 要求時属性値を組み込むJSPアクション

```
<jsp:setProperty name="user" property="timezonePref"
                    value='<%= request.getParameter("timezone") %>'/>
```

要求時属性値は式を使用して指定されるため、他のスクリプト要素と同じソフトウェア保守問題が生じやすくなっています。このため、JSTLのカスタム・タグでは動的属性値指定のために代替的な機構をサポートしています。JSTLアクションの属性値は、本格的なJSP式を使用するのではなく、単純化した式言語 (EL) を使用して指定することができます。ELは、JSPコンテナー内に存在するデータを検索したり操作したりするためのID、アクセサー、および演算子を提供します。ELは、大まかにはEcmaScript (参考文献 を参照) とXML Path Language (XPath) を基にしているため、その構文はページ・デザイナーとプログラマー双方にとってなじみがあるはずです。ELは、オブジェクトやプロパティーのルックアップとそれらに対する簡単な操作に適合しています。これはプログラム言語でないばかりか、スクリプト言語ですらありません。ただし、JSTLタグと結合すれば、簡単で便利な表記法を使用して複雑な振る舞いを表すことができます。EL式は、リスト3に強調表示されているように、先頭のドル記号 ($) および先頭と末尾の中括弧 ({}) によって区切られます。

リスト3.EL式の区切りを示すJSTLアクション

```
<c:out value="${user.firstName}"/>
```

また、リスト4に示すように、文字列連結を通じて静的テキストを持つ複数の式を結合することによって、動的属性値を構成することもできます。個々の式は、ID、アクセサー、リテラル、および演算子で構成されます。IDは、データ・センターに保管されたデータ・オブジェクトを参照するのに使用されます。ELには、ELで暗黙的に存在する11のオブジェクトに対応した11の予約IDがあります。他のすべてのIDは、スコープ付き変数 を指すものと想定されます。アクセサーは、あるオブジェクトのプロパティーまたはあるコレクションの要素を検索するのに使用されます。リテラルは、固定値 (数字、文字列、ブール式、ヌル値) を表します。演算子を使用すると、データやリテラルを結合したり比較したりできます。

リスト4. 静的テキストと複数のEL式を結合して、動的属性値を指定する

```
<c:out value="Hello${user.firstName} ${user.lastName}"/>
```

スコープ付き変数

JSP APIは、`<jsp:useBean>` アクションを通じて、JSPコンテナー内の4つの異なるスコープにデータを保管したり、検索したりできるようにします。JSTLは、これらのスコープ内でオブジェクトを

割り当てたり除去したりするための追加アクションを提供することによって、この機能を拡張しています。さらに、ELではこれらのオブジェクトをスコープ付き変数として検索するための、組み込みサポートを提供しています。具体的には、あるEL式に出現する任意のIDで、ELの暗黙的オブジェクトのいずれかに対応していないものは、次の4つのJSPスコープのどれかに保管されたオブジェクトを参照するものと自動的に想定されます。

- ページ・スコープ
- 要求スコープ
- セッション・スコープ
- アプリケーション・スコープ

ご存じのように、ページ・スコープに保管されたオブジェクトは、そのページがある特定の１回の要求に応じて処理されている間のみ検索できます。要求スコープに保管されたオブジェクトは、ある要求の処理に参加するすべてのページが処理される間のみ検索できます(ある要求の処理が1つ以上の`<jsp:include>` または`<jsp:forward>` アクションに出くわした場合など)。あるオブジェクトがセッション・スコープに保管されている場合、そのオブジェクトはWebアプリケーションとの単一の対話セッションの間のみ (つまり、そのユーザーの対話に関連付けられた`HttpSession` オブジェクトが無効になるまで)、ユーザーがアクセスする任意のページによって検索できます。アプリケーション・スコープに保管されたオブジェクトは、(一般にはJSPコンテナーがシャットダウンされた結果として) そのWebアプリケーション自身がアンロードされるまで、すべてのページからすべてユーザーがアクセスできます。

オブジェクトは、希望するスコープ内のオブジェクトに文字列をマッピングすることによって、スコープに保管されます。その後、同じ文字列を指定することによって、スコープからそのオブジェクトを検索することができます。その文字列は、スコープのマッピングでルックアップされ、マップされたオブジェクトが戻されます。Servlet API内では、そのようなオブジェクトは対応するスコープの属性 と呼ばれます。ただしELのコンテキストでは、ある属性に関連付けられた文字列は変数名と考えることもできます。この変数は、属性マッピングの要領で特定の値にバインドされます。

ELでは、暗黙的オブジェクトに関連付けられていないIDは、4つのスコープに保管されたオブジェクトを指すものと想定されます。このようなIDは、まずページ・スコープ、次に要求スコープ、そしてセッション・スコープ、最後にアプリケーション・スコープの順にチェックされ、IDの名前がそのスコープ内に保管されたオブジェクトの名前と一致するかどうかが検査されます。最初に一致したものが、そのEL のIDの値として戻されます。こうした意味で、EL のIDはスコープ付き変数を参照していると考えられます。

より専門的な説明をすると、暗黙的オブジェクトにマップされていないIDは、現在処理中の要求に対する式が存在するページの処理を表す`PageContext` インスタンスの`findAttribute()` メソッドを使用して評価されます。そのID名は、引数としてこのメソッドに渡され、そのメソッドによって4つのスコープに同じ名前の属性がないか順に検索されます。最初に一致するもの見つかると、それが`findAttribute()` メソッドの値として戻されます。4つのスコープのいずれにもそのような属性が見つからない場合、`null` が戻されます。

究極的には、スコープ付き変数は、EL のIDとして使用できる名前を保持している4つのJSPスコープの属性です。それらに英数字の名前が割り当てられている限り、スコープ付き変数

はJSP内にある属性設定の任意のメカニズムによって作成することができます。これには、組み込みの<jsp:useBean> アクションや、Servlet APIのいくつかのクラスによって定義されたsetAttribute() メソッドも含まれます。また、4つのJSTLライブラリー内で定義されているカスタム・タグの多くは、それ自身、スコープ付き変数として使用するための属性値を設定することができます。

暗黙的オブジェクト

表1に、11個のEL暗黙的オブジェクトのIDがリストされています。これらをJSPの暗黙的オブジェクト (これらは9個しかありません) と混同しないようにしてください。両者に共通のオブジェクトは1つしかありません。

表1. EL暗黙的オブジェクト

カテゴリー	ID	説明
JSP	pageContext	現行ページの処理に対応するPageContext インスタンス
スコープ	pageScope	ページ・スコープ付き属性の名前と値を関連付けるMap
	requestScope	要求スコープ付き属性の名前と値を関連付けるMap
	sessionScope	セッション・スコープ付き属性の名前と値を関連付けるMap
	applicationScope	アプリケーション・スコープ付き属性の名前と値を関連付けるMap
要求パラメーター	param	要求パラメーターの主要な値を名前別に保管するためのMap
	paramValues	要求パラメーターのすべての値をString 配列として保管するためのMap
要求ヘッダー	header	要求ヘッダーの主要な値を名前別に保管するためのMap
	headerValues	要求ヘッダーのすべての値をString 配列として保管するためのMap
Cookie	cookie	要求に伴うCookieを名前別に保管するためのMap
初期化パラメーター	initParam	Webアプリケーションのコンテキスト初期化パラメーターを名前別に保管するためのMap

JSP暗黙的オブジェクトとEL暗黙的オブジェクトには、共通のオブジェクトが1つしかありません (pageContext) が、他のJSP暗黙的オブジェクトもELからアクセスすることは可能です。その理由は、pageContext には、他の8つのJSP暗黙的オブジェクトすべてにアクセスするためのプロパティーがあるからです。実際、これがEL暗黙的オブジェクトにpageContext が含まれている主な理由です。

残りのEL暗黙的オブジェクトはすべてマップで、ある名前に対応するオブジェクトの検索に使用できます。最初の4つのマップは、先ほど説明したさまざまな属性スコープを表します。ELがデフォルトで使用する順次検索プロセスに頼る代わりに、これらのマップを使用して特定のスコープ内のIDをルックアップすることができます。

次の4つのマップは、要求パラメーターや要求ヘッダーの値を取り出すためのものです。HTTPプロトコルは、要求パラメーターと要求ヘッダーのいずれにも多値を持たせられるため、それぞれについて2種類のマップがあります。2種類のうちの最初のマップは、要求パラメーターや要求ヘッダーの主要な値を単に戻します。通常は、実際の要求でたまたま最初に指定された値になります。2番目のマップでは、あるパラメーターまたはヘッダーのすべての値を取り出すことができます。このマップのキーはパラメーターやヘッダーの名前であり、値は`String` オブジェクトの配列で、その各要素が1つひとつのパラメーターまたはヘッダーの値になっています。

Cookie暗黙的オブジェクトは、要求によって設定されたCookieへのアクセスを提供します。このオブジェクトは、ある要求に関連付けられたすべてのCookieの名前を、それらのCookieのプロパティーを表す`Cookie` オブジェクトにマップします。

最後のEL暗黙的オブジェクト、`initParam` は、そのWebアプリケーションに関連付けられた任意のコンテキスト初期化パラメーターの名前と値を保管するマップです。初期化パラメーターは、そのアプリケーションの`WEB-INF` ディレクトリーにある`web.xml` デプロイメント記述ファイルを通じて指定されます。

アクセサー

EL のIDは、暗黙的オブジェクトかスコープ付き変数 (属性を通じて実装される) のいずれかとして解決されるため、必然的にJavaオブジェクトとして評価されます。ELは、対応するJavaクラス内のプリミティブを自動的にラップおよびアンラップすることができますが (たとえば`int` を裏側で`Integer` クラスへと強制することができ、その逆も可能)、大部分のIDは本格的なJavaオブジェクトへのポインターになります。

結果として、通常、これらのオブジェクトのプロパティーにアクセスするか、配列やコレクションの場合であればその要素にアクセスすることが望ましいと言えます。ELには、この目的だけのためにドット演算子 (`.`) とブラケット演算子 (`[]`) という2つのアクセサーがあります。これらは同時に、ELを通じてプロパティーや要素を演算できるようにします。

ドット演算子は一般に、あるオブジェクトのプロパティーにアクセスするために使用されます。例えば`${user.firstName}` という式では、`user` IDによって参照されるオブジェクトの`firstName` というプロパティーにアクセスするのにドット演算子が使用されています。ELは、Java Bean規約を使用してオブジェクトのプロパティーにアクセスするので、この式が正しく評価されるためには、このプロパティーのgetter (通常は`getFirstName()` というメソッド) を定義しておく必要があります。アクセスされるプロパティー自身がオブジェクトである場合、ドット演算子をさらに適用することができます。例えば、ここで仮に取りあげる`user` オブジェクトにはJavaオブジェクトとして実装された`address` プロパティーがあるとすると、このオブジェクトのプロパティーにアクセスするためにもドット演算子を使用できます。例えば`${user.address.city}` という式は、このアドレス・オブジェクトのネストされた`city` プロパティーを戻します。

ブラケット演算子は、配列やコレクションの要素を検索するのに使用されます。配列や順序付きコレクション (すなわち`java.util.List` インターフェースを実装するコレクション) の場合、検索される要素のインデックスはブラケット内に現れます。例えば、`${urls[3]}` という式は、`urls` IDによって参照される配列またはコレクションの4番目の要素を戻します (Java言語やJavaScriptと同様に、ELのインデックスはゼロ・ベースです)。

java.util.Map インターフェースを実装するコレクションの場合、ブラケット演算子は、関連するキーを使用してマップに保管された値をルックアップします。キーはブラケット内で指定され、対応する値がその式の値として戻されます。例えば、${commands["dir"]} という式は、commands IDによって参照されるMap の"dir" キーに関連付けられた値を戻します。

どちらの場合も、ブラケット内に式が現れることが許容されます。ネストした式の評価の結果は、コレクションまたは配列の適切な要素を検索するためのインデックスまたはキーとして役立ちます。ドット演算子でもそうであったように、ブラケット演算子をさらに適用することができます。これにより、ELは多次元配列やネストされたコレクション、あるいはその2つの組み合わせから要素を検索することができます。さらに、ドット演算子とブラケット演算子を組み合わせることが可能です。例えば、配列の要素がそれ自身オブジェクトである場合、ブラケット演算子を使用してその配列から要素を検索することができるとともに、ドット演算子と組み合わせてその要素のいずれかのプロパティー (例えば${urls[3].protocol}) を検索することもできます。

ELの役割が動的属性値を指定するための簡易言語であるとすると、ELアクセサーの興味深い特性の1つとして、Java言語のアクセサーとは異なり、null を適用したときに例外をスローしないという点があります。ELアクセサーが適用されるオブジェクト (例えば${foo.bar} と${foo["bar"]} 両方にあるfoo ID) がnull の場合、アクセサーの適用結果もnull になります。後ほどわかりますが、これは多くの環境のもとでかなり役立ちます。

最後に、ドット演算子とブラケット演算子はある程度置き換え可能です。例えば、${commands.dir} をcommands マップの"dir" キーに関連付けられた値を取り出すのに使用できますし、${user["firstName"]} をuser オブジェクトのfirstName プロパティーを検索するのに使用できます。

演算子

IDとアクセサーを使用して、ELは (スコープ付き変数を通じて公開される) アプリケーション・データあるいは (ELの暗黙的オブジェクトを通じて公開される) 環境に関する情報の入ったオブジェクト階層を全探索することができます。ただし、多くのJSPアプリケーションに必要なプレゼンテーション・ロジックの実装には、そのようなデータに単純にアクセスすることは不適切な場合がよくあります。

このためELには、EL式によってアクセスされるデータの操作や比較を行うためにいくつかの演算子が組み込まれています。これらの演算子は、表2にまとめられています。

表2. EL演算子

カテゴリー	演算子
算術	+、-、*、/ (またはdiv)、% (またはmod)
関係	== (oreq)、!= (orne)、< (またはlt)、> (またはgt)、<= (またはle)、>= (またはge)
論理	& (またはand)、|| (またはor)、! (またはnot)
検証	empty

算術演算子は、数値の加法、減法、乗法、除法をサポートしています。剰余演算子も用意されています。除法と剰余の演算子には、代替の非記号の名前があることに注意してください (XPathと

の整合性を保つため)。リスト5は、算術演算子の使用を示す式の例です。1対のEL式への算術演算子の適用結果は、これらの式によって戻される数値にその演算子を適用した結果です。

リスト5. 算術演算子を使用したEL式

```
${item.price * (1 + taxRate[user.address.zipcode])}
```

関係演算子では、数値またはテキスト・データを比較できます。比較結果は、ブール値で戻されます。論理演算子では、ブール値を結合して新たなブール値を戻すことができます。したがってリスト6に示すように、ELの論理演算子は、ネストされた関係または論理演算子の結果に適用することができます。

リスト6. 関係演算子と論理演算子を使用したEL式

```
${(x >= min) & (x <= max)}
```

最後のEL演算子は`empty` です。これは、データを検証するのに特に便利です。`empty` 演算子は、引数として単一の式 (つまり`${empty input}`) をとり、その式が「空の」値と評価されたか否かを示すブール値を戻します。`null` と評価された式は空である、つまり要素のないコレクションまたは配列とみなされます。`empty` 演算子は、引数がゼロの長さの`String` であると評価された場合にも`true` を戻します。

EL演算子の演算子優先順位は表3に示すとおりです。リスト5とリスト6が示すように、括弧を使用して式をグループ化し、通常の優先順位規則をオーバーライドすることができます。

表3. ELの演算子優先順位 (上から下、左から右の順)

- `[]`,`.`
- `()`
- 単項の`-`、`not`、`!`、`empty`
- `*`、`/`、`div`、`%`、`mod`
- `+`、2項の`-`
- ()<,>,<=、>=、`lt`、`gt`、`le`、`ge`
- `==`、`!=`、`eq`、`ne`
- `&`、`and`
- `||`、`or`

リテラル

EL式のリテラル値として、数字、文字列、ブール値、および`null` が指定可能です。文字列は単一または二重引用符で区切ります。ブール値は`true` と`false` で指定します。

taglib指示子

以前に説明したように、JSTL 1.0には4つのカスタム・タグ・ライブラリーが組み込まれています。式言語を使用してJSTLタグの相互作用を示すために、ここではJSTL`core` ライブラリーのいくつかのタグを見ていきましょう。JSPカスタム・タグ・ライブラリーでもそうであるように、この

ライブラリーのタグを使用できるようにしたい任意のページに`taglib`指示子を組み込む必要があります。この特定のライブラリー用の指示子は、リスト7に示されています。

リスト7. ELバージョンのJSTL coreライブラリー用taglib指示子

```
<%@ taglib uri="http://java.sun.com/jstl/core" prefix="c" %>
```

JSTL 1.0ではELがオプションであるため、実際にはJSTL`core`ライブラリーに対応する`taglib`指示子は2つあります。JSTL 1.0の4つのカスタム・タグ・ライブラリーすべてには、動的属性値の指定にELではなくJSP式を使用する代替バージョンがあります。式言語を使用するライブラリーがEL ライブラリーと呼ばれるのに対し、これらの代替ライブラリーは、より伝統的なJSPの要求時属性値に依存するためRT ライブラリーと呼ばれます。別々の`taglib`指示子を使うことによって２種類のライブラリーを区別して使えます。RTバージョンを使用する指示子をリスト8に示します。しかし今はELを話題の中心にしているので、必要なのはこれらの指示子のうち最初に示したほうです。

リスト8. JSTL coreライブラリーのRTバージョン用のtaglib指示子

```
<%@ taglib uri="http://java.sun.com/jstl/core_rt" prefix="c_rt" %>
```

変数タグ

ここで検討する最初のJSTLカスタム・タグは`<c:set>`アクションです。すでに示したように、スコープ付き変数はJSTLにおいて主要な役割を演じ、`<c:set>`アクションが、スコープ付き変数を作成および設定するためのタグ・ベースのメカニズムを提供します。このアクションの構文は、リスト9に示すとおりです。ここでは、`var`属性はスコープ付き変数の名前を指定し、`scope`属性はその変数が常駐するスコープを示し、`value`属性はその変数にバインドされる値を指定します。指定された変数がすでに存在する場合、指示された値に単純に割り当てられます。存在しない場合は、新規のスコープ付き変数が作成され、その値に初期化されます。

リスト9. <c:set> アクションの構文

```
<c:set var="name" scope="scope" value="expression"/>
```

`scope`属性はオプションであり、`page`にデフォルト設定されています。

`<c:set>`の2つの例がリスト10に示されています。最初の例では、セッション・スコープ付き変数が`String`値に設定されています。2番目の例では、式を使用して数値が設定されています。`square`というページ・スコープ付き変数に、`x`という要求パラメーターの値を自乗した結果が割り当てられています。

リスト10. <c:set> アクションの例

```
<c:set var="timezone" scope="session" value="CST"/>
<c:set var="square" value="${param['x'] * param['x']}"/>
```

属性を使用するのではなく、`<c:set>`アクションのボディ・コンテンツとしてスコープ付き変数に値を指定することもできます。この方法を使用すれば、リスト10の最初の例をリスト11のように書き換えることができます。さらに、もうすぐわかりますが、`<c:set>`タグのボディ・コンテン

ツでカスタム・タグ自身を使用することも許容されます。<c:set> のボディ内で生成されるコンテンツ全体が、String 値として指定の変数に割り当てられます。

リスト11. ボディ・コンテンツを通じて <c:set> アクションに値を指定する

```
<c:set var="timezone" scope="session">CST</c:set>
```

JSTL coreライブラリーには、スコープ付き変数を管理するための2つ目のタグ、<c:remove> が組み込まれています。その名が示すとおり、<c:remove> アクションはスコープ付き変数を削除するのに使用されるもので、2つの属性をとります。リスト12に示すとおり、var 属性は除去する変数の名前を指定し、オプションのscope 属性はどのスコープから除去するのかを示すもので、デフォルトはpage になっています。

リスト12. <c:remove> アクションの例

```
<c:remove var="timezone" scope="session"/>
```

出力

<c:set> アクションは、ある式の結果をスコープ付き変数に割り当てることを可能にしてくれますが、開発者は式の値を保管するのではなく、単に表示したいだけという場合もよくあります。これは、JSTLの<c:out> カスタム・タグの役割です。その構文はリスト13に表示されています。このタグは、value 属性によって指定された式を評価してから、その結果を印刷します。オプションのdefault 属性が指定されているときに、value 属性の式がnull または空のString として評価された場合、<c:out> アクションは代わりにその値を印刷します。

リスト13. <c:out> アクションの構文

```
<c:out value="expression" default="expression" escapeXml="boolean"/>
```

escapeXml 属性もオプションです。<c:out> タグによる出力の際に、"<"、">"、"&" のような、HTMLとXMLの両方で特別の意味を持つ文字をエスケープするかどうかを制御します。escapeXml をtrueに設定すると、これらの文字は対応するXMLエンティティー (ここで取りあげた文字については、それぞれ<、>、& に) 自動的に変換されます。

例えば、username とcompany という2つのプロパティーを定義するクラスのインスタンスである、user というセッション・スコープ付き変数があるとします。このオブジェクトは、ユーザーがそのサイトにアクセスするたびに自動的にセッションに割り当てられますが、2つのプロパティーはユーザーが実際にログインするまで設定されません。このようなシナリオで、リスト14に示すようなJSPフラグメントについて検討してみます。ユーザーがログインすると、このフラグメントは "Hello" というワードにそのユーザーのユーザー名と感嘆符を続けて表示します。しかしユーザーがログインするまでは、このフラグメントが生成するコンテンツは、この代わりに "Hello Guest!" というフレーズになります。この場合、username プロパティーはまだ初期化されていないため、<c:out> タグは、代わりにdefault 属性の値 (つまり文字列 "Guest") を印刷します。

リスト14. デフォルト・コンテンツ付きの <c:out> アクションの例

```
Hello <c:out value="${user.username}" default="Guest"/>!
```

次に、<c:out> タグのescapeXml 属性を使用したリスト15について検討してみます。company プロパティーが、このケースでは"Flynn &Sons" というJavaString 値に設定されているとすると、このアクションで生成されるコンテンツは実際にFlynn & Sons となります。このアクションがHTMLまたはXMLコンテンツを生成するJSPページの一部である場合、この文字列のまん中にある & 記号がHTMLまたはXML制御文字と解釈され、このコンテンツのレンダリングや構文解析を妨げてしまう可能性があります。しかし、escapeXml 属性の値がtrue に設定されていれば、生成されるコンテンツはFlynn & Sons になります。このようなコンテンツが出てきても、ブラウザーやパーサーには問題なく解釈できるはずです。HTMLやXMLがJSPアプリケーションで最も一般的なコンテンツ・タイプであるとすると、escapeXml 属性のデフォルト値がtrue であることも驚くには値しません。

リスト15. エスケープが使用不可になっている <c:out> アクションの例

```
<c:out value="${user.company}" escapeXml="false"/>
```

デフォルト値付き変数の指定

<c:out> にデフォルト値を指定できることは、動的データの表示を簡単にするだけでなく、<c:set> を通じて変数値を設定する際にも便利です。リスト11 に強調表示されているように、スコープ付き変数に割り当てられた値は、<c:set> タグの値属性を通じて指定できるほか、そのボディ・コンテンツとして指定することもできます。<c:set> タグのボディ・コンテンツの<c:out> アクションをネストさせることにより、変数割り当てでデフォルト値の機能を活用することができます。

この方法は、リスト16に示されています。外側の<c:set> タグの振る舞いは単純です。これは、セッション・スコープ付きのtimezone 変数に、そのボディ・コンテンツに基づいて値を設定します。ただしこのケースでは、ボディ・コンテンツは<c:out> アクションを通じて生成されます。ネストされたこのアクションの値属性は${cookie['tzPref'].value} という式で、これはcookie 暗黙的オブジェクトを通じてtzPref という名前のCookieの値を戻そうとするものです。(cookie 暗黙的オブジェクトは、Cookieの名前を対応するCookie インスタンスに対応付けます。したがって、Cookieに保管された実際の値を検索するためには、ドット演算子を使用して、Cookieオブジェクトのvalue プロパティーを取り出す必要があります。)

リスト16. <c:set> と <c:out> を結合してデフォルトの変数値を得る

```
<c:set var="timezone" scope=="session">
   <c:out value="${cookie['tzPref'].value}" default="CST"/>
</c:set>
```

ところが、ユーザーがこのコードを使用したWebアプリケーションを初めて利用するというケースを検討しましょう。その結果、その要求に供給されるtzPref というCookieはないことになります。つまり、暗黙的オブジェクトを使用したルックアップはnull を戻し、その場合、式全体もnull を戻します。value 属性の評価結果がnull であるため、<c:out> タグも代わりにdefault 属性の評価結果を出力します。ここでは、これは文字列CST です。そして最終的な効果としては、timezone スコープ付き変数はそのユーザーのtzPref Cookieに保管された時間帯に設定されるか、それがない場合はデフォルトの時間帯であるCST が設定されます。

ELとJSP 2.0

現在は、式言語はJSTLカスタム・タグの動的属性値の指定にしか使用できません。JSTL 1.0式言語の拡張が提案されていますが、JSP 2.0仕様への取り込みについては現在最終検討が進められている最中です。この拡張は、独自のカスタム・タグと一緒にELを活用できるようにするものです。ページ作成者は、現在JSP式の仕様が許可されている任意の箇所でEL式を使用できるようになります。例えば、`<p>Your preferred time zone is${timezone}.</p>` というテンプレート・テキストに動的な値を挿入するなどです。

JSTLそのものと同様に、JSP 2.0のこの特性によって、ページ作成者はJSPスクリプト要素への依存を減らすことができ、JSPアプリケーションの保守容易性の向上につながります。

まとめ

ELは、4つのJSTLカスタム・タグ・ライブラリーによって提供されるアクションと連携して、ページ作成者がスクリプト要素に頼らずにプレゼンテーション層ロジックを実装できるようにします。例として、この記事の冒頭にあるリスト1 のJSPコードを、リスト17に強調表示されている、JSTLを通じて実装される同じ機能と対比してみてください。(`<c:choose>` やその子を含むJSTL`core` ライブラリーの残りのタグについては、このシリーズの次回の記事で取り上げます。) 条件付きロジックが実行されることはやはり明らかですが、JSTLバージョンにはJava言語のソース・コードが含まれておらず、タグ同士の関係 (特にネスト要件に関して) はHTML構文を知っている人なら誰にでもなじみがあります。

リスト17. JSTLを用いて条件付コンテンツを実装する

```
<c:choose>
<c:when test="${user.role == 'member'}">
    <p>Welcome, member!</p>
</c:when>
<c:otherwise>
    <p>Welcome, guest!</p>
</c:otherwise>
</c:choose>
```

JSTLは、多くのWebアプリケーションに共通の機能の標準実装を提供することによって、開発サイクルの迅速化に役立ちます。JSTLは、ELと連携してプレゼンテーション層のプログラム・コードの必要性を除去し、JSPアプリケーションの保守を非常に単純化します。

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。