JSPベスト・プラクティス: JSPのインクルード・インクルード・メカニズムを使用してルック・アンド・フィールを向上させる

Webサイトのコンテンツを処理しやすく分解する

Brett D. McLaughlin, Sr.
Author and Editor
O'Reilly Media, Inc.

2003年 4月 15日

新しいJSPベスト・プラクティス・シリーズの第1回となるこの記事では、JavaServer Pagesの`include`のメカニズムをご紹介します。Javaプログラミングの専門家であるBrett McLaughlin氏が、`include`を使用してWebサイトやWebアプリケーションのページに静的なヘッダーおよびフッター・ファイルを組み込む方法を説明します。

Javaゾーン最新のベスト・プラクティス・シリーズが始まりました。これまでのシリーズを購読された読者にはすでにお馴染みのとおり、ベスト・プラクティスの記事はさまざまなJavaテクノロジーの有益な特徴をすばやく理解することを目的としています。ベスト・プラクティスのこのシリーズでは、J2EEの中核的なテクノロジーの1つであるJavaServer Pages (JSP) テクノロジーを専門に扱います。

一言で言えば、JSPテクノロジーとはJavaプラットフォーム上にWebページやWebアプリケーションのインターフェースを構築するためのツールです。JSPテクノロジーにより、要求データに対する動的な対応、複雑なXMLとHTMLの表示、視覚に訴えて動的に変化するWebサイトの作成などを行うことができます。このシリーズでは、JSPテクノロジーを使用したWebサイト構築のための基本を理解することができます。ここで取り上げる最も一般的なJSPのメカニズムをとおして、テンプレートの作成、動的コンテンツの操作、イメージ・ホスティング、ユーティリティー・コード・ライブラリーの作成などの重要なWeb開発技術を習得しましょう。

シリーズ第1回のこの記事では、ローカルHTMLページのコンテンツの組み込みを可能にするJSPの`include`メカニズムに焦点を当てます。まず最初に、Webページのインクルードが発展してきた背景を、特にフレームとサーバー・サイド・インクルードの使用を中心に説明します。その後、JSPの`include`メカニズムを使用して、WebページやWebアプリケーションの複数の画面に同一のヘッダーとフッターを追加する方法を説明します。

JSPベスト・プラクティス・シリーズ

このシリーズは、JSPテクノロジーのすべてを紹介したり、特定のタイプのアプリケーションの開発方法を説明するためのものではありません。シリーズの各記事では、JSPテクノロジーを使用したプログラミングのある側面に焦点を当て、それをかみ砕いて詳しく説明することに重点を置いています。JSPテクノロジー全般の概要や、特定の結果を得るためのより詳細な使用方法については、参考文献を参照してください。

必要なもの

このシリーズのすべてのベスト・プラクティスはJavaServer Pagesテクノロジーに基づいています。これらを実行するためには、JSP対応のWebコンテナー (Apache Tomcatなど) をローカル・マシンかテスト・サーバーのどちらかに設定する必要があります。また、JSPページのコードを作成するには、テキスト・エディターまたはIDEが必要です。Tomcatへのリンク、およびJSP対応WebコンテナーとIDEのリストについては、参考文献を参照してください。

より優れたルック・アンド・フィール

Webページのデザインとレイアウトを一貫したものにすることは、本格的なルック・アンド・フィールを実現する最も簡単な方法の1つです。多くのWebページを見るとわかりますが、1つのサイト内のほとんどのページで、同一のヘッダーとフッター、およびナビゲーション・バーといったものが使用されています。うまくデザインされたサイトでは、各ページでこれらの要素に同じレイアウト、コンテンツ、および機能を持たせながら、メイン・パネル (多くの場合コンテンツ・ペインと呼ばれる) の表示内容をページごとに変えています。

このようなレイアウトは、かつてほとんどがフレームかフレームセットだけで実装されていました。このような実装では、静的なコンテンツの各部をフレームごとに配置し、ページのメイン・コンテンツを中央のフレームに配置していました。フレームで問題になるのは、ブラウザーによって異なった表示になることが多いために、見た目の統一性に欠ける点です。また、サイト内のページからフレームを使用した外部サイトへのリンクが、フレームを使用しない場合に比べると難しくなるという点も挙げられます。外部サイトへのリンクの目的は、ユーザーが元のサイトを去ることなく外部コンテンツを表示できるようにすることですが、元のサイトとの連携を残した結果が得られることは稀です。サイト全体がとても小さなフレームに押し込められてしまうか、悪くすると元のサイトが別のサイトのフレーム内にネストされてしまう場合もあります。このような混乱により、Webデザイナーはより優れたソリューションを追い求めるようになりました。そして、私たちはその解決策をサーバー・サイド・インクルード (SSI) の中に見出したのです。

サーバー・サイド・インクルード

最近まで、SSIは共有コンテンツを作成するための最も一般的な選択肢でした。単純なSSIディレクティブにより、他のページから自分のページにコンテンツ (ヘッダーやフッターなど) を組み込むことができます。リスト1はその例です。

リスト1. SSIの動作

```
<html>
     <head>
      <title>Simple SSI test</title>
     </head>
     <body>
       This content is statically in the main HTML file.<br />
       <!--#include virtual="included.html" -->
     </body>
</html>
```

このファイルは、この後の演習で使用します。今のところは、このファイルをtest-ssi.shtmlとして保存しておいてください。ほとんどの環境では、SSIファイルは.shtml で終わる必要があります。.shtmlが付くことにより、Webサーバーはこれらを SSIディレクティブとして解析する対象であると認識します。リスト2は、インクルードされる側のファイルです。名前はincluded.htmlとします。

リスト2. インクルードされるコンテンツ

```
This content is in another file, included.html
```

test-ssi.shtmlを要求すると、ユーザーにはincluded.htmlのコンテンツとともに、このファイルのコンテンツが表示されます。これらのファイルは、Apache TomcatなどのSSI対応Webコンテナーで表示できます (参考文献を参照)。

ユーザーの観点からすれば、SSIはフレームにとっての大きな進歩です。それは、ユーザーにとっては、単一のファイルであるかインクルードした他のファイルから取り込んだコンテンツであるかという違いをまったく意識する必要がないからです。逆に、SSIには特殊なサーバー設定が必要で、多くの場合Java開発者はそれを使用できないという欠点があります。また、最近のSSIのバージョンでは動的コンテンツを組み込むこともできますが、それ以前は一般にインクルードされるコンテンツは静的でなければなりませんでした。

SSIは異なる種類のコンテンツをWebサイトやWebアプリケーションに組み込む上で実用的な解決策ですが、Java開発者にとって最善の選択肢ではありません。JavaServer Pagesテクノロジーは、SSIの代替手段をJavaのみで実現したものです。SSIとJSPは簡単に組み合わせることはできません。JSPページの拡張子は.jsp なので、SSIディレクティブが正しく解釈されるようにするためには、SSIの設定を変更してJSPファイルも解析するようにする (各JSPページの解析でオーバーヘッドが増加します) か、JSPの設定を変更して`.shtml` 拡張子を持つファイルもJSPページとして扱うようにする (これは良い方法ではありません) 必要があります。JSPテクノロジーはJava開発者にとってコンテンツ管理における最善の解決策であり、幸いにもその`include` メカニズムは簡単に習得できます。

JSPインクルード

JSPの`include` ディレクティブは、同じような機能を持つSSIに非常に似ています。リスト3は、リスト1に示したSSIディレクティブに相当するJSPです。JSP対応のWebコンテナー (リンクについては参考文献を参照してください) であれば、このJSPページを処理して表示できます。このファイルは、test-include.jspとして保存しておいてください。

リスト3. JSPのincludeディレクティブ

```
<%@ page language="java" contentType="text/html" %>
<html>
     <head>
      <title>JSP include element test</title>
     </head>
     <body>
      This content is statically in the main JSP file.<br />
      <%@ include file="included.html" %>
     </body>
</html>
```

include ディレクティブにより、非常に簡単に同一のヘッダー・ファイルとフッター・ファイルをサイトに組み込むことができます。リスト4は、インクルード・ファイルをいくつか含むメインのインデックス・ページです。

リスト4. メインのインデックス・ページ用のJSP includeディレクティブ

```
<%@ page language="java" contentType="text/html" %>
<html>
<head>
    <title>newInstance.com</title>
    <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
    <link href="/styles/default.css" rel="stylesheet" type="text/css" />
</head>
<body>
<%@ include file="header.jsp" %>
<%@ include file="navigation.jsp" %>
<%@ include file="bookshelf.jsp" %>
<%@ include file="/mt-blogs/index.jsp" %>
<%@ include file="footer.jsp" %>
</body>
</html>
```

このコードを見ると、JSPインクルードの使用法に関して多くのことが理解できます。それを皆さん自身で試すことにより、JSPインクルードのしくみを学んでください。

動的コンテンツの追加

ヘッダー、フッター、ナビゲーション・ファイルなどの静的コンテンツに加えて、リスト4にはウェブログ (/mt-blogs/index.jsp) の呼び出しが含まれています。これを題材にして、これから動的コンテンツについて説明します。動的コンテンツに関しては、SSIのinclude ディレクティブ同様に、JSPのinclude メカニズムにも問題があります。JSPのinclude ディレクティブを使用すると動的コンテンツを取り込むことができますが、そのコンテンツに対する変更内容は反映されません。これは、インクルードされたファイルが元の (インクルード元の) ページの一部としてWebコンテナーに読み込まれるためです。コンテナーは、結果を複数のJSPコンポーネントではなく、単一のファイルとしてキャッシュします。Webコンテナーは、インクルードされたファイルに対する変更をポーリングしないため、その変更をまったく認識できず、更新されたページではなくキャッシュしてあるページを自動的に表示します。このしくみを確認するために、簡単な演習を行います。まず、保存したincluded.htmlページをリスト5のように変更します。

リスト5. インクルードされたコンテンツの変更

```
This content is in another file, included.html.
<br />
Some new content...
```

次に、これらの変更内容を保存して、test-include.jspファイルに移動し、ブラウザーをリフレッシュします。included.htmlの新しいコンテンツが表示されないことに注意してください。インクルードされたファイルのコンテンツは、変更が加えられる前にキャッシュされているため、新しいコンテンツは表示されません。サイトに動的なコンテンツがある場合や、頻繁に変更されるコンテンツがある場合は、これが問題になります。幸いにも、これには回避策があります。次回は、<jsp:include> タグを使用してWebページに動的コンテンツを組み込む方法を説明します。

それまでの間、参考文献で紹介されているコードをいろいろ試してみてください。それでは、次回をお楽しみに。

著者について

Brett D. McLaughlin, Sr.

Brett McLaughlin氏は、Logo (小さな三角形を覚えていますか?) の時代からコンピューターの仕事をしています。現在の専門は、JavaおよびJava関連のテクノロジーを使ったアプリケーション・インフラストラクチャーの構築です。ここ数年は、Nextel Communications and Allegiance Telecom, Inc. でこれらのインフラストラクチャーの実装に携わっています。Brett氏は、Javaサーブレットを使ってWebアプリケーション開発のための再利用可能なコンポーネント・アーキテクチャーを構築するJava Apacheプロジェクトturbineの共同設立者の1人です。同氏はまた、オープン・ソースのEJBアプリケーション・サーバーであるEJBossプロジェクトと、オープン・ソースのXML Web公開エンジンであるCocoonにも貢献しています。