国際化JSPアプリケーションを構築する

サーバー側アプリケーション特有の困難を理解する

Sing Li
Author
Makawave

2005年 3月 29日

国際的な対象者向けのJSP（Java®Server Pages）アプリケーション設計は、科学と言うよりも職人仕事であり、見た目以上の要素を含むものです。成功の鍵は、国際化に関連した、サーバー側特有の問題をよく理解することです。この記事ではJava開発者のSing Liが、鍵となる問題は何かを明確にし、実践的手法に基づく2つの解決手段を示します。

世界の経済が、より国際的に拡大するにつれ、国際的なユーザーが利用できるWebベースのソフトウェアの必要性が高まっています。こうしたユーザーが必要とする言語や表示方法、データ入力方法、表示形式、文化的な要求には、大きな幅があります。国際化（省略してi18n）というのは、そうした多様なユーザーに適したアプリケーションを作るための作業です。

少し驚かれるかも知れませんが、J2SEに組み込まれた国際化サポートは、サーバー側にカスタム修正を加えないと機能不足なのです。一般的に言ってサーバー側アプリケーションの国際化は相変わらず、科学と言うよりも、独自形式あるいは専用ソリューションを伴う職人仕事なのです。

この記事では、サーバー側のJSPベース・アプリケーションで必要なことと、J2SEアプリケーションで必要なことを区別します。ここでは様々なクライアント/サーバー・トポロジーを調べ、それぞれによってサーバー側の要求が大幅に異なることを示します。その後で、基本的な問題を解決する手段として広く使われている、2つの方法の動作コードを検証して行きます。

J2SEロケールを超えて

J2SEでは、国際化に対してロケール(locale) という概念を使用しています（ロケールの表記を見てください）。単一のマシンの場合では、ロケールは、表示に使う言語（例えば英語かスペイン語かなど）や日付、時間、通貨などのフォーマットに関するユーザーの好みを示します。通常は、下にあるOS（オペレーティング・システム）がロケール情報を管理し、実行時にJ2SEに渡します。

ローカル・マシン上でサーバーを実行している場合、あるいは、同じLAN上にある別マシンにサーバーがある場合には、マシン固有のロケールはうまく動作します。全てのクライアント・マシンとサーバー・マシンは同じロケールを持つので、同じ表示言語を使い、同じ日付フォーマッ

トを使い、等々というわけです。これらのシナリオはどれも、面倒な国際化問題を提起することはありません。ところが、同じサーバーを使って、複数の国にまたがるユーザーにサービスを提供しようとすると、単純とは言えない状況になってきます。

ロケールの表記

ロケールは、言語コードの後に国コードが続いたものとして規定されます（こうしたコードの情報については、参考文献を見てください）。標準的なフォーマットはxx_YYであり、xxは2文字の言語コード（小文字）、YYは2文字の国コード（大文字）です。また、言語コードと国コードの間にハイフンがつく場合や、国コードが小文字の場合もあります。国と言語を分離しているのは、使う言語は同じでも、日付や時間、通貨フォーマットなどは異なる国があるためです。

サーバー側にまつわる国際化の問題

国際的な利用に向けてサーバー・アプリケーションを展開する場合には、様々な場所を同時にサポートしなければなりません。図1は、起こり得るシナリオを示しています。各マシン、つまりサーバーと、任意の時間にサーバーにアクセスするクライアントは、それぞれ独自のロケール設定を持ち、しかもこうしたロケールは異なっている可能性が高いのです。

図1. 様々なロケールを持つユーザーに対してサービスするサーバー

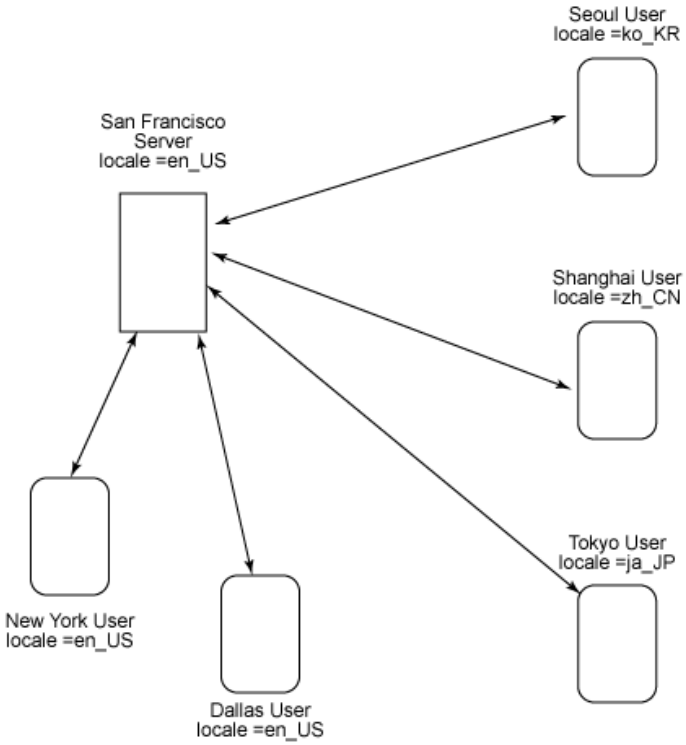


図1では、サーバー・マシンはサンフランシスコにあり、マシン固有のロケールはen_US（英語、アメリカ合衆国）です。ニューヨークとダラスのユーザーは全員en_USを使うため、それ以上の国際化は必要ありません。ところが、ソウルからアクセスするユーザーは、ロケールko_KRとして、アプリケーションが韓国語で表示されることを期待します。一方、上海からアクセスするユーザーはロケールzh_CNとして、アプリケーションの文字が中国語で表示されることを要求します。東京のユーザーは、ロケールja_JPとして日本語で表示されることを期待します。こうした

ユーザーそれぞれの要求は、サンフランシスコにあるサーバーで実行しているJSPアプリケーションで対応しなければなりません。

サーバー側では、サーバー・マシン自体のロケールは完全に制御できますが、クライアントを特定なロケールに変更させたり、強制させたりすることはできません。アプリケーションは各ユーザーのロケールを見つけ、適切なローカライズ方法を使って、JSPページが適切に表示されるようにする必要があります（クライアント・ロケールを検出する、を見てください）。

クライアント・ロケールを検出する

ユーザーのブラウザーから入力されるHTTPヘッダーを調べれば、ユーザーのロケールを自動的に検出することができます。ただしこの情報は、ユーザー、またはOSによって正しく設定されていない可能性があります。しかも、一部のブラウザーには、ロケール情報の処理にバグがあります。ですから、ユーザーに好みのロケールを明示的に尋ねるようにした方が、より妥当だということが理解できるでしょう。

ロケール特定にまつわる、もっと複雑な問題

クライアントのロケールを単純に特定しても（そして、対応したJSPを表示しても）安全だと思ったところで、新しい問題に突き当たります。非常に現実的な、次のシナリオを考えてみてください。

東京からの出張者が、中国の上海にある支店のマシン（ロケールはzh_CN）を使います。中国語の表示は分からないからと、この出張者は日本語でWebアプリケーションにアクセスしようとします。この状況を説明した、図2の（a）を見てください。

ユーザーが、クライアント・マシンとは異なるロケールを要求する

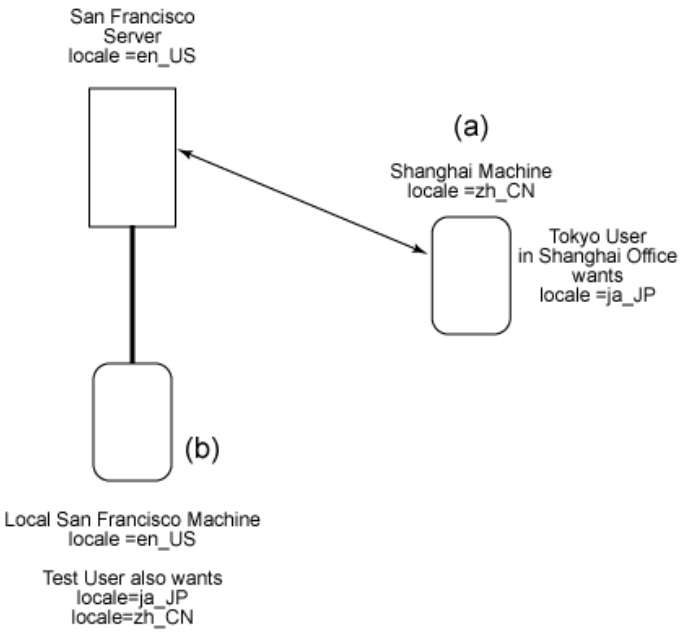


図2の（a）では、サンフランシスコにあるサーバーは、ロケールとしてen_USを持っています。上海にある、クライアント・マシンのロケールはzh_CNです。しかしJSPアプリケーションがユーザーの要求に応えるためには、ja_JPロケールで表示しなくてはなりません。

もう一つのシナリオを挙げましょう（ちょっと変わっていますが、あり得るシナリオです）。国際化JSPアプリケーションの開発者として、あなたはデバッグを行っています。en_USロケールを持つクライアント・システムで、JSPアプリケーションを実行する3つのブラウザー・インスタンスを開きます。サーバー・マシンはLAN上にあり、そのロケールもen_USです。ところがあなたは、そのアプリケーションが中国語や日本語をどう処理するかをテストするのです。ですから1台のen_USクライアント・マシン上で、1つのブラウザー・インスタンスは英語（en_US）、1つは日本語（ja_JP）、1つは中国語（zh_CN）となります。図2の（b）は、この状況を示しています。

ここまでで、国際化にまつわる基本的な問題が明らかになったと思います。国際化アプリケーションの特定なインスタンスを表示する上では、クライアント・ロケールもサーバー・ロケールも、実際に要求されるロケールとは関係ないのです。そのページに対して、どのロケールで表示すべきかを決められるのは、ユーザーのみだということです。

幸いなことに、ユーザーはそのアプリケーションで作業している間は表示言語を変更しない、と想定しても、通常は問題ありません。従って、ロケールは通常、セッションに関連付けることができます。

ロケールに依存した言語表示の問題を解決する

JSPアプリケーション用に、異なる言語で表示するための処理方法として受け入れられているのは、次の2つです。

- それぞれ別の言語でエンコードした複数のJSPセットを保存し、ユーザーのロケール選択に応じて、こうしたセットの間で切り替えを行う。
- 全ての文字列使用を分離し、代わりにリソース・バンドルから、ロケール特有のストリングを取得する（この手法では、J2SEのロケール特有リソース・バンドル処理を使います）。

このそれぞれに対する2種類のサンプル・コードを、お見せしましょう。このサンプル・アプリケーションは、developerWorks Emailという、架空のeメール・サービス用のログイン画面です。このeメール・システムのユーザーは最初に、カレント・セッションに使用するロケールを決定するための言語選択画面を見せられます。選択肢は、図3に示すように、英語、韓国語、日本語、そして中国語です。このコードを試してみたい方は、URL http://<server address>/dwi18n/multdir/index.jsp を使って、このページをアクセスしてください。

最初の言語選択に画像を使う

先方のクライアントが、どんなフォントをサポートしているか、事前に確実に知る方法が無いことがよくあります。最近のOSでは、ストレージやメモリーを節約するために、デフォルトでは全Unicode文字の完全なフォント・サポートをプリインストールしません。一部のブラウザーでは、インストールされていない文字を含んだWebページが最初にアクセスされた場合のみ、必要なフォントをダウンロードし、インストールしようとします。言語選択画面で、外国の文字や、国旗を示すために絵を使うのは現実的な方法です。

明示的にロケール選択を行うための言語選択画面

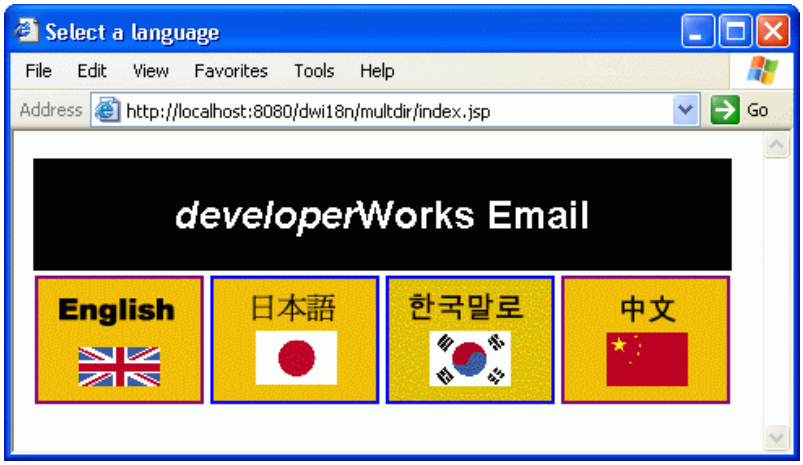
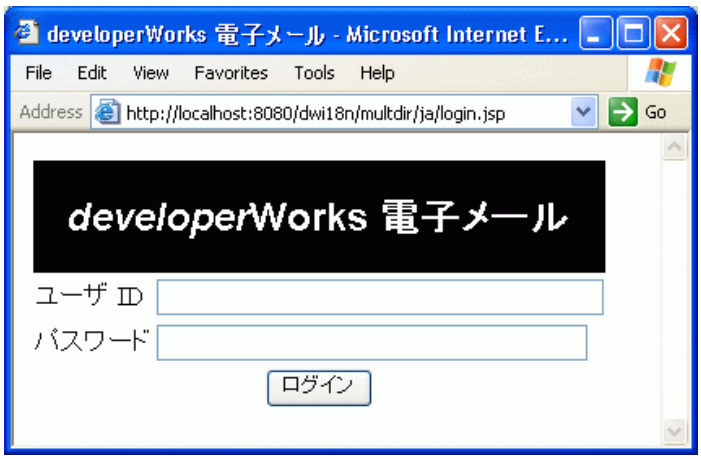


図3では、言語選択画面で4つの画像を使い、4つのロケール選択を表現しています（最初の言語選択に画像を使う、を参照）。ログイン画面は、ここでユーザーが選択する言語を使って表示されます。図4は、日本語でのログイン画面を表しています。

日本語でのサイン・オン画面



冗長に、言語専用のJSPセットを複数使う

サンプル・コード・ディストリビューションのディレクトリー、webapps/dwi18n/multdir/ にある最初の例では、複数のJSPページ・セットを使っています。図5は、このアプリケーションのディレクトリー階層構造を示しています。

dwi18n/multdirのディレクトリー階層構造で、ロケール専用ディレクトリーを示す

図5で、各ロケールには、それぞれに対応したサブディレクトリーがあります。en_ENロケール用に英語でエンコードされたJSPは、enサブディレクトリーにあります。ko_KRロケールに対応する、韓国語でエンコードされたJSPは、koサブディレクトリーにあります。ja_JPロケールに対しては、日本語でエンコードされたJSPが、jaサブディレクトリーにあります。zh_CNロケールは、zhサブディレクトリーにある、中国語でエンコードされたJSPで表現されます。これらそれぞれのサブディレクトリーには、サイン・オン画面のJSP（login.jsp）と、データ確認のJSP（confirm.jsp）が含まれています。

データ確認のJSPは単純に、この単純な例で入力されたデータを表示します。例えば、中国語のサイン・イン画面で情報を入力し、ボタンをクリックすると、データ確認のJSPは、入力されたデータを表示します（図6）。

中国語での確認画面

JSPをコーディングする

index.jspという名前のロケール選択JSPは、言語専用JSPセットの内の一つに直接リンクされます。この場合のindex.jspのコードを、リスト1に示します。

リスト1. 言語専用のページに直接リンクするロケール選択JSP

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<html>
<head>
<title>Select a language</title>
</head>
<body>
<table>
<tr>
<td colspan=4 bgcolor="black">
<br/>
<center><font face="arial" size=+2 color="white">
    <b><i>developer</i>Works Email</b></font>
</center>
<br/>
</td>
</tr>
 <tr><td>
 <c:url value="en/login.jsp" var="englishURL"/>
```

```
 <a href="${englishURL}">
    <img src="english.gif"/>
 </a>
</td>
<td>
 <c:url value="ja/login.jsp" var="japaneseURL"/>
  <a href="${japaneseURL}">
       <img src="japanese.gif"/>
  </a>
</td>
<td>
  <c:url value="ko/login.jsp" var="koreanURL"/>
  <a href="${koreanURL}">
       <img src="korean.gif"/>
  </a>
</td>
<td>
  <c:url value="zh/login.jsp" var="chineseURL"/>
  <a href="${chineseURL}">
       <img src="chinese.gif"/>
  </a>
</td>
</tr>
</table>
</body>
</html>
```

リスト1では、JSTL（JSP Standard Tag Library）からの`<c:url>`タグを使って、リンク先のURLを作っていることに注意してください。これによって、セッション管理が適切に処理されることになります（JSTLと`<c:url>`タグについては、参考文献を見てください）。

login.jspとconfirm.jspのセットは、それぞれロケールに合わせた言語でコード化されます。リスト2は、（図4に対応した）ja_JPロケールに対するlogin.jspを示しています。

リスト2. ja_JPロケールに対するサイン・オン・ページ（login.jsp）

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html;charset=utf-8" >
<title>developerWorks #####</title>
</head>
<body>
<c:url value="confirm.jsp" var="actionURL"/>
<form action="${actionURL}" method="post">
<table>
<tr>
<td colspan=2 bgcolor="black">
<br/>
<center><font face="arial" size=+2 color="white">
    <b><i>developer</i>Works #####</b></font>
</center>
<br/>
</td>
</tr>
<tr>
<td>### ID</td>
<td><input type="text" name="userid" size="40"/></td>
</tr>
<tr>
<td>#####</td>
<td><input type="password" name="pass" size="40"/></td>
</tr>
```

```
<tr>
<td colspan="2" align="center">
<input type="submit" value="####"/></td>
</tr>

</table>
</form>
</body>
</html>
```

同様に、（図6に対応した）zh_CNロケールに対して、confirm.jspは中国語でエンコードされています。これをリスト3に示します。

リスト3. zh_CNロケールに対するデータ確認ページ（confirm.jsp）

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html;charset=utf-8" >
<title>developerWorks ##</title>
</head>
<body>

<table border="1">
<tr>
<td colspan=2 bgcolor="black">
<br/>
<center><font face="arial" size=+2 color="white">
     <b><i>developer</i>Works ##</b></font>
</center>
<br/>

</td>
</tr>


<tr>
<td>####</td>
<td>${param.userid}</td>
</tr>

<tr>
<td>##</td>
<td>${param.pass}</td>
</tr>




</table>


</body>
</html>
```

上でお見せした、冗長に用意した複数セットによる手法は、次のようなアプリケーションでは現実的な解決手段となります。

- 主に一つの言語でアクセスされ、他のロケールからアクセスされることは稀なアプリケーション
- あまり頻繁に変更することのない、プレゼンテーション・レイヤーJSPを下に持つアプリケーション

J2SEリソース・バンドルを使う

前のセクション、冗長に、言語専用のJSPセットを複数使うで示した解決手段の最大の欠点は、ある言語専用のJSPセットに更新が必要になると、冗長にコード化されたJSPセット全てに対して、その更新を行う必要があることです。そのため、中規模程度のプロジェクトであっても、時間のかかる、間違いを起こしやすい更新作業が必要になってきます。

これからお見せする解決手段は、先の解決手段と同じように見え、同じように動作しますが、今度はlogin.jspとconfirm.jspのセットを一つしか使いません。この方法はリソース・バンドルにあるJ2SEサポートを利用し、必要な時にロケール専用の文字列を引き出すのです（J2SEのリソース・バンドルに関しては、参考文献を見てください）。この方法に対するコードの例は、webapps\dwi18n\javares ディレクトリーの下にあります。皆さんがこのサンプル・コードを展開する場合には、URL http://<server address>/dwi18n/javares/index.jsp を使ってください。

図7は、同じクライアント・マシンで実行している、4つのブラウザー・セッションを示しています（それぞれのセッションが、別々のロケールを要求しています）。

同じマシン上の、4つ別々のロケール

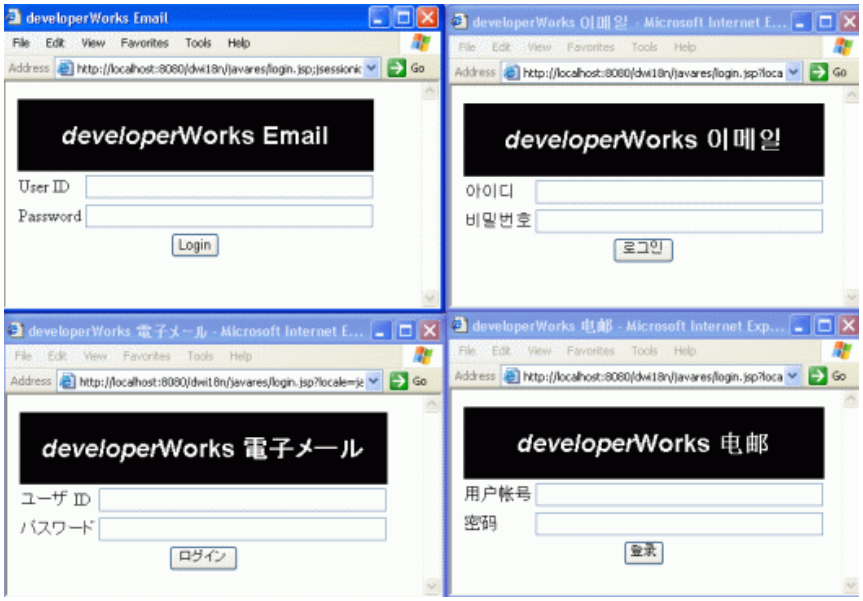


図7を見ると、国際化JSPアプリケーションが同時に複数ロケールを処理する様子が、はっきり分かると思います。

JSPをコーディングする

この場合のindex.jspは、1つのlogin.jspにリンクされているだけなので、少し異なります。リスト4は、この場合のindex.jspのコードを示しています。

リスト4.　1つのlogin.jspのみにリンクした、ロケール選択ページ

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

```
<head>
<title>Select Language</title>
</head>
<body>
<table>
<tr>
<td colspan=4 bgcolor="black">
<br/>
<center><font face="arial" size=+2 color="white">
    <b><i>developer</i>Works Email</b></font>
    </center>
<br/>
</td>
</tr>
 <tr><td>
 <c:url value="login.jsp" var="englishURL">
   <c:param name="locale" value="en_US"/>
 </c:url>
 <a href="${englishURL}">
    <img src="english.gif"/>
 </a>
</td>
<td>
  <c:url value="login.jsp" var="japaneseURL">
     <c:param name="locale" value="ja_JP"/>
  </c:url>
  <a href="${japaneseURL}">
       <img src="japanese.gif"/>
  </a>
</td>
<td>
  <c:url value="login.jsp" var="koreanURL">
     <c:param name="locale" value="ko_KR"/>
  </c:url>
  <a href="${koreanURL}">
       <img src="korean.gif"/>
  </a>
</td>
<td>
  <c:url value="login.jsp" var="chineseURL">
     <c:param name="locale" value="zh_CN"/>
  </c:url>
  <a href="${chineseURL}">
       <img src="chinese.gif"/>
  </a>
</td>
</tr>
</table>
</body>
</html>
```

リスト4では、JSTL`<c:param>`タグを使って、`locale`というURLリクエスト・パラメーターを設定していることに注意してください。このパラメーターは、ユーザーが言語選択の上でクリックすると、login.jspに渡されます。リスト5は、login.jspのコードを示しています。

リスト5. J2SEリソース・バンドルを使ったサイン・イン・ページ（ login.jsp ）

```
<%@ page pageEncoding="UTF-8" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<html>
<c:set var="loc" value="en_US"/>
<c:if test="${!(empty param.locale)}">
  <c:set var="loc" value="${param.locale}"/>
</c:if>
```

```
<fmt:setLocale value="${loc}" />

<fmt:bundle basename="app">
<head>
<title>developerWorks <fmt:message key="email"/></title>
</head>
<body>
<c:url value="confirm.jsp" var="formActionURL" />
<form action="${formActionURL}" method="post">
<table>
<tr>
<td colspan=2 bgcolor="black">
<br/>
<center><font face="arial" size=+2 color="white"><b>
       <i>developer</i>Works <fmt:message key="email"/>
         </b></font></center>
<br/>
</td>
</tr>
<tr>
<td><fmt:message key="userid"/></td>
<td>
<input type="hidden" name="locale" value="${loc}"/>
<input type="text" name="userid" size="40"/></td>
</tr>
<tr>
<td><fmt:message key="password"/></td>
<td><input type="text" name="pass" size="40"/></td>
</tr>

<tr>
<td colspan="2" align="center">
<input type="submit" value="<fmt:message key='login'/>"/></td>
</tr>
</table>
</form>
</body>
</fmt:bundle>
</html>
```

リスト5では、JSTL国際化ヘルパー・タグ・ライブラリー（参考文献）を使っています。入力されるparam.localeが空の場合には、ロケールはデフォルトでen_USに設定されます。リソース・バンドルで作業する場合には、<fmt:setLocale>タグを使ってロケールを設定します。

ロケールが設定されると、<fmt:message>タグは、指定されたキーに応じて、バンドルのプロパティー・ファイルから文字列を引き出します。バンドルのベース名は、<fmt:bundle>JSTLタグを使ってappに設定されます。dwi18n/WEB-INF/classesディレクトリーの下を見ると、リソース・バンドルのプロパティー・ファイル（とその他のファイル）がすべて、そこにあることが分かります。表1は、こうしたファイルを表しています。J2SEリソース・バンドルの使い方については、参考文献を見てください。

表1. リソース・バンドルにあるファイル

ファイル名	説明
app.properties	デフォルトで使用するプロパティー・ファイル。en_USロケールに対応。
app_zh.properties	zh_CNロケールに対するプロパティー・ファイル。中国語でエンコードした文字列を含む。

app_ko.properties	ko_KRロケールに対するプロパティー・ファイル。韓国語でエンコードした文字列を含む。
app_ja.properties	ja_JPロケールに対するプロパティー・ファイル。日本語でエンコードした文字列を含む。
*.ucd	プロパティー・ファイルを作るためのUnicodeソース・ファイル
convacii.bat<\/td>	ucdファイルをプロパティー・ファイルに変換するためのバッチ・ファイル

一例として、`app_ko.properties` ファイルの内容をリスト6に示します。

リスト6. リソース・バンドルからのapp_ko.propertiesファイル

```
email=\uc774\uba54\uc77c
userid=\uc544\uc774\ub514
password=\ube44\ubc00\ubc88\ud638
login=\ub85c\uadf8\uc778
```

リスト6では、Unicode文字が全てエスケープされていることに注意してください。Javaリソース・バンドル機構では、ASCIIコードでのプロパティー・ファイルしか受け付けないのため、こうする必要があるのです。このファイルを作るには、IDEにあるストリング・リソース・エディターを使うか、Unicodeエディターを使ってUnicodeファイルを作ってから、JDKの`nativetoascii` ユーティリティーを使って変換します。この場合では、`convascii.bat` ファイルで変換を行っています。

まとめ

国際化JSPアプリケーションを設計する際には、特有の要件を意識する必要があります。様々なロケール要求を持つユーザーによる同時複数アクセスをサポートするように、アプリケーションを用意する必要があります。この記事では、ロケール特有の言語による文字を、国際化JSPアプリケーションの中でどのように表示するか、という問題への対処方法として、2つを示しました。しかしこれでも、国際化対応のサーバー側アプリケーションを構築する上での技術に関して、ごく表面的なことに触れたにすぎません。その他にも、日付や通貨に関する様々なフォーマットの問題や、GUIレイアウトの処理方法、特別なIME（Input Method Editors、外国の文字を入力するユーティリティー・ソフトウェア）への対応方法など、重要な問題が数多くあります。さらに国際化に関して調べるには、参考文献を見てください。

ダウンロード

内容	ファイル名	サイズ
Tomcat 5.5.7でテストした、この記事用のサンプル・コード	code.zip	25KB

著者について

Sing Li

Sing LiはWrox Pressから出版されている多数の本の著者で、Professional Apache Tomcat 、Early Adopter JXTA 、Professional Jini などを執筆しています。技術雑誌に頻繁に寄稿しており、P2P発展に関する熱心なエバンジェリスト（伝道者）でもあります。