JSTL入門: プレゼンテーションがすべて

カスタム・タグを使用した書式設定と国際化対応

Mark Kolb
Software Engineer

2003年 4月 15日

作成したウェブ・アプリケーションを世界各国に対応させたいと考えている開発者にとって、サイト訪問者向けコンテンツをローカライズすることは重要です。JDK 1.1以降、Javaプログラミング言語には国際化対応機能が組み込まれており、JSP標準タグ・ライブラリー (JSTL) の`fmt`ライブラリーに含まれている中心的なカスタム・タグ・セットを使用すると、これら国際化対応機能のすべてに簡単にアクセスすることができます。Mark Kolb氏は、全4回シリーズの第3回となる今回の記事で、再びJSTLを取り上げ、データを書式設定および国際化対応させるための`fmt`タグについて説明します。

本シリーズのこれまでの記事では、JSTLとその式言語 (EL) について説明しました。また、`core`ライブラリーで定義されているカスタム・タグについても考察しました。具体的には、「式言語」では、ELが、JSPアプリケーション内のデータにアクセスして操作するとともに、これらのデータを動的属性値としてJSTLのカスタム・タグで使用できるようにするための、単純化された言語であることを述べました。「核心(core)に触れる」では、スコープ付き変数を管理したり、EL値を表示したり、反復コンテンツや条件付きコンテンツを実装したり、URLを操作したりするためのカスタム・タグが含まれた`core`ライブラリーについて説明しました。

今回の記事で紹介するJSTLライブラリーは、`fmt`ライブラリーです。`fmt`ライブラリー内のカスタム・タグは、リソース・バンドルによるテキスト・コンテンツのローカライズ、および数値や日付の表示と構文解析をサポートしています。これらのタグでは、`java.util`パッケージや`java.text`パッケージで実現されているように、Java言語の国際化対応APIが活用されるため、`ResourceBundle`、`Locale`、`MessageFormat`、`DateFormat`などのクラスをすでに使い慣れている場合は、`fmt`ライブラリーは役に立ちます。そうでない場合でも、`fmt`ライブラリーのタグでは、国際化対応APIが直感的に扱えるような形でカプセル化されているため、作成するJSPアプリケーションにローカライズ機能を簡単に組み込むことができます。

ローカリゼーション

Java言語の国際化対応APIには、データのローカライズ方法を左右する2つの重要な要素があります。1つはユーザーのロケールであり、もう1つはユーザーの時間帯です。ロケールは、日付、数値、金額の書式設定法など、特定の地域や文化の言語規約を表します。ロケールには、必ず1つの言語が関連付けられており、多くの場合この言語は、複数のロケールで共用されている言語の

方言です。たとえば、アメリカ、イギリス、オーストラリア、カナダの英語方言ごとに異なるロケールが用意されており、フランス、ベルギー、スイス、カナダのフランス語方言ごとに異なるロケールが用意されています。

本シリーズの他の記事もお読みください

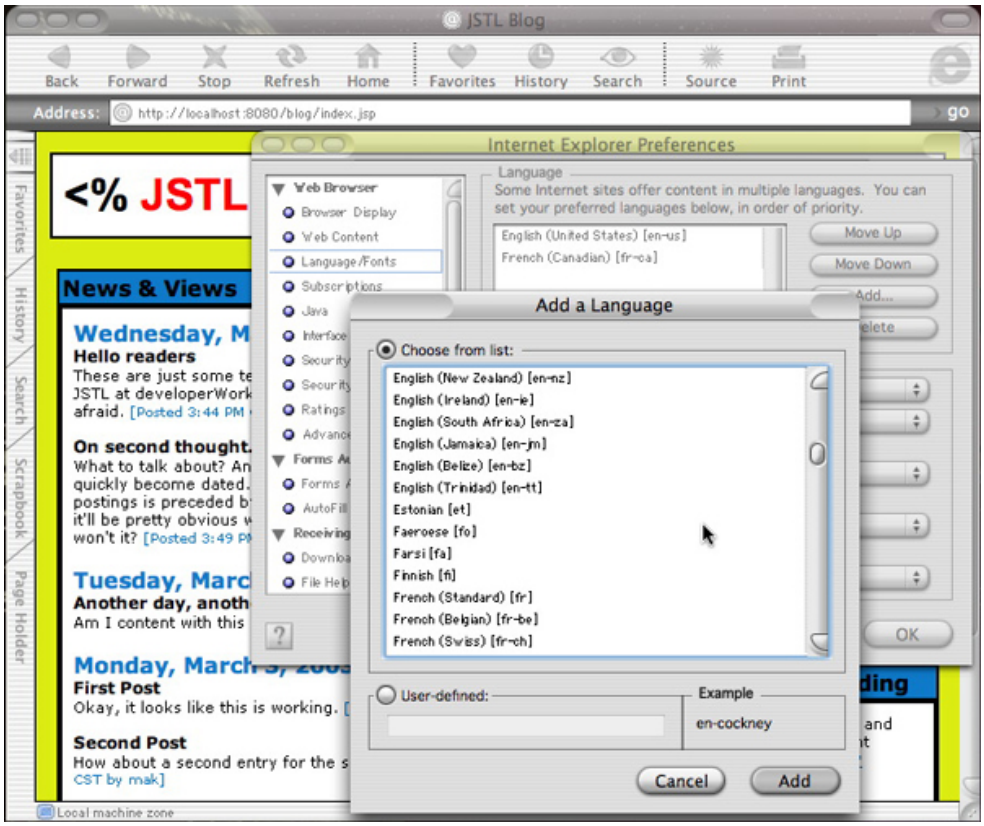
第1回 「式言語」(2003年2月)
第2回 「核心(core)に触れる」(2003年3月)

データのローカライズにおける2つ目の要素は時間帯です。これは、一部のロケールは非常に広範囲な地域にまたがっているからです。オーストラリア英語のように大陸全体にわたるロケールの時刻情報を表示する場合は、ユーザーの時間帯に応じた時刻を表示することは、時刻を適切に書式設定することと同じくらい重要です。

しかしここで、アプリケーション側では、どのようにしてユーザーのロケールと時間帯を判断できるのか？という疑問が生じます。Javaアプリケーションの場合は、当該地域のオペレーティング・システムと連携して、JVMによってデフォルトのロケールと時間帯が設定されます。この方法は、デスクトップ・アプリケーションの場合は有効ですが、サーバー・サイドのJavaアプリケーションの場合はあまり適していません。これらのアプリケーションでは、地球の反対側の場所から送信されてくる要求が処理されることもあるからです。

幸い、HTTPプロトコルには、`Accept-Language` 要求ヘッダーを使用して、ブラウザーからサーバーにローカライズ情報を転送するための機能が備わっています。多くのWebブラウザーでは、図1に示すように、言語の優先順位をカスタマイズすることができます。一般に、任意の数の希望ロケールを明示的に設定できないブラウザーでは、代わりに、オペレーティング・システムに問い合わせて、`Accept-Language` ヘッダーで送信する値が決定されます。サーブレットの仕様では、`javax.servlet.ServletRequest` クラスの`getLocale()` メソッドと`getLocales()` メソッドを使用して、このHTTPプロトコルの機能が自動的に活用されます。次に、JSTLの`fmt` ライブラリー内のカスタム・タグでは、これらのメソッドを利用して、ユーザーのロケールが自動的に判断され、これに応じて出力が調整されます。

図1. ブラウザーの言語の優先順位を設定してロケールを選択する



しかし、残念なことに、ユーザーの時間帯をブラウザーからサーバーに転送するための標準HTTP要求ヘッダーは用意されていません。このため、Webアプリケーションの時間データをローカライズするには、ユーザー固有の時間帯を判断して常時監視するための独自のメカニズムを実装する必要があります。たとえば、本シリーズの第2回記事「核心(core)に触れる」で紹介したウェブログ・アプリケーションには、ユーザーの時間帯の設定をクッキーに格納するためのフォームが含まれています。

fmtライブラリー

JSTLの`fmt`ライブラリー内のカスタム・タグは、大きく4つのグループに分類されます。1つ目のグループでは、他のタグの処理の基準となるローカリゼーション・コンテキストを設定することができます。つまり、このタグ・グループを使用すると、ページ作成者はロケールと時間帯を明示的に設定することができ、他の`fmt`タグでは、このロケールと時間帯に基づいてデータが書式設定されます。2つ目と3つ目のタグ・グループでは、それぞれ日付と数値を書式設定および構文解析することができます。4つ目のタグ・グループでは、テキスト・メッセージをローカライズすることができます。

以上が基本事項の説明ですが、次に、これら4つのタグ・グループを個別に順番に説明して、これらの使用法を例示します。

ローカリゼーション・コンテキスト・タグ

前述のように、JSTLタグによるデータの書式設定時に使用されるロケールは、通常、ユーザーのブラウザーからHTTP要求の一部として送信される`Accept-Language`ヘッダーの値に基づいて決定

されます。このようなヘッダーが存在しない場合のために、JSTLでは、デフォルトのロケールを指定するための一連のJSP構成変数が用意されています。これらの構成変数が設定されていない場合は、JSPコンテナーの実行元オペレーティング・システムから取得される、JVMのデフォルト・ロケールが使用されます。

`fmt` ライブラリーには、上記のユーザー・ロケール決定処理に優先してロケールを指定するための独自のカスタム・タグ`<fmt:setLocale>` が含まれています。次のコード例で示すように、`<fmt:setLocale>` アクションは3つの属性をサポートしています。

```
<fmt:setLocale value="expression"
    scope="scope" variant="expression"/>
```

これらの属性のうち、`value` 属性だけが必須です。この属性の値には、ロケールの文字列名または`java.util.Locale` クラスのインスタンスを指定します。ロケール名は、2つの小文字からなるISO国別コードであり、必要に応じてその後ろに、下線またはハイフン、および2つの大文字からなるISO言語コードを付加します。

たとえば、`en` は英語の言語コードであり、`US` はアメリカの国別コードであるため、`en_US` (または`en-US`) は、アメリカ英語のロケール名になります。同様に、`fr` はフランス語の言語コードであり、`CA` はカナダの国別コードであるため、`fr_CA` (または`fr-CA`) は、カナダ・フランス語のロケール名になります (すべての有効なISO言語コードと国別コードへのリンクについては、参考文献を参照してください)。もちろん、国別コードは省略できるため、`en` や`fr` だけでも有効なロケール名であり、これらのロケール名は、対応する言語の個別方言が区別されないアプリケーションには適しています。

`<fmt:setLocale>` の任意指定の`scope` 属性では、ロケールのスコープを指定します。この属性の値が`page` の場合、設定したロケールは現在のページだけに適用され、値が`request` の場合、設定したロケールは、同一要求の最中にアクセスされたすべてのJSPページに適用されます。`scope` 属性の値が`session` の場合、設定したロケールは、ユーザーのセッション中にアクセスされたすべてのJSPページに適用されます。値が`application` の場合、設定したロケールは、当該Webアプリケーションのすべてのユーザーを対象にして、このアプリケーションのすべてのJSPページに対するすべての要求について適用されます。

`variant` 属性 (これも任意指定) を使用すると、Webブラウザーの特定のプラットフォームやベンダーに合わせて、ロケールをさらにカスタマイズすることができます。たとえば、`MAC` と`WIN` は、それぞれApple MacintoshとMicrosoft Windowsプラットフォームを示すvariant属性値です。

次のコード例では、`<fmt:setLocale>` タグを使用して、ユーザー・セッション用のロケールを明示的に設定する方法を示しています。

```
<fmt:setLocale value="fr_CA" scope="session"/>
```

JSPコンテナーによってこのJSPコードが処理されると、ユーザーのブラウザーで設定されている言語の優先順位が無視されて、代わりに上記の設定が適用されます。

<fmt:setLocale> と同様に、<fmt:setTimeZone> アクションを使用すると、他のfmt カスタム・タグで使用されるデフォルトの時間帯を設定することができます。このアクションの構文は次のとおりです。

```
<fmt:setTimeZone value="expression"
    var="name" scope="scope"/>
```

<fmt:setLocale> の場合と同様に、value 属性だけが必須ですが、この属性の値には、時間帯名またはjava.util.TimeZone クラスのインスタンスを指定します。

残念ながら、一般的に使用されている時間帯名の表記規則はありません。したがって、<fmt:setTimezone> タグのvalue 属性で指定できる時間帯名は、Javaプラットフォーム独自のものです。有効な時間帯名のリストを取得するには、java.util.TimeZone クラスのgetAvailableIDs() staticメソッドを呼び出します。有効な時間帯名としては、US/Eastern、GMT+8、Pacific/Guam などがあります。

<fmt:setLocale> の場合と同様に、任意指定のscope 属性を使用して、設定した時間帯のスコープを指定することができます。次のコードでは、<fmt:setTimeZone> を使用して、個別ユーザーのセッションに適用される時間帯を指定しています。

```
<fmt:setTimeZone value="Australia/Brisbane" scope="session"/>
```

また、<fmt:setTimeZone> アクションを使用して、TimeZone インスタンスの値をスコープ付き変数に格納することもできます。この場合は、var 属性でスコープ付き変数を指定し、scope 属性でこの変数のスコープを指定します (たとえば、<c:set> アクションや<c:if> アクションでこれら2つの属性を使用する場合と同様です)。<fmt:setTimeZone> アクションをこのような形式で使用する場合、指定した変数が設定されるだけです。var 属性を指定した場合、JSP環境では、他のJSTLタグで使用される時間帯は変更されません。

このグループの最後のタグは、<fmt:timeZone> アクションです。

```
<fmt:timeZone value="expression">
body content
</fmt:timeZone>
```

<fmt:setTimeZone> と同様に、このタグでは、他のJSTLタグで使用される時間帯を指定します。ただし、<fmt:timeZone> アクションのスコープは、その本文コンテンツ内に限定されています。<fmt:timeZone> タグの本文内では、JSP環境で設定されている他の時間帯の代わりに、このタグのvalue 属性で指定された時間帯が使用されます。

<fmt:setTimeZone> の場合と同様に、<fmt:timeZone> タグのvalue 属性の値には、時間帯名またはjava.util.TimeZone クラスのインスタンスを指定します。<fmt:timeZone> の使用法については、後述のリスト1 で例示しています。

日付タグ

fmt ライブラリーには、日付と時刻を操作するための<fmt:formatDate> と<fmt:parseDate> という2つのタグが含まれています。それぞれの名前から分かるように、<fmt:formatDate> は、日付

と時刻を書式設定して表示するためのタグであり (データの出力)、<fmt:parseDate> は、日付値と時刻値を構文解析するためのタグです (データの入力)。

<fmt:formatDate> の構文は次のとおりです。

```
<fmt:formatDate value="expression" timeZone="expression"
    type="field" dateStyle="style" timeStyle="style"
    pattern="expression"
    var="name" scope="scope"/>
```

value 属性だけが必須です。この値には、java.util.Date クラスのインスタンスを指定します。これにより、書式設定して表示する日付や時刻を指定します。

任意指定のtimeZone 属性では、表示される日付や時刻の基準となる時間帯を指定します。timeZone 属性が明示的に指定されていない場合は、<fmt:formatDate> を囲んでいる<fmt:timeZone> タグで指定された時間帯が使用されます。<fmt:formatDate> アクションが<fmt:timeZone> タグで囲まれていない場合は、関連する<fmt:setTimeZone> アクションで設定された時間帯が使用されます。関連する<fmt:setTimeZone> アクションがない場合は、JVMのデフォルト時間帯が使用されます (当該地域のオペレーティング・システムで設定されている時間帯)。

type 属性では、指定したDate インスタンスのどのフィールドを表示するのかを指定し、その値には、time、date、both のいずれかを指定します。この属性のデフォルトの値はdate であるため、type 属性が指定されていない場合は、<fmt:formatDate> タグでは、その名前のとおり、value 属性で指定されたDate インスタンスに関連付けられた日付情報だけが表示されます。

dateStyle 属性とtimeStyle 属性では、それぞれ日付情報と時刻情報の書式を指定します。有効な値は、default、short、medium、long、およびfull です。デフォルトの値はもちろんdefault であり、この場合は、ロケール固有の書式が使用されます。他の4つの書式値のセマンティクスは、java.text.DateFormat クラスで定義されているとおりです。

用意されている書式を使用する代わりに、pattern 属性を使用して独自の書式を指定することもできます。pattern 属性を使用する場合、その値には、java.text.SimpleDateFormat クラスの規則に従ったパターン文字列を指定する必要があります。これらのパターンを使用した場合、パターン内の指定文字は、対応する日付フィールドや時刻フィールドに置き換えられます。たとえば、MM/dd/yyyy というパターンを指定した場合は、2桁の月と日および4桁の年がスラッシュで区切られて表示されます。

var 属性を指定した場合は、この属性で指定した変数に、書式設定済みの日付が格納されたString 値が代入されます。指定しない場合、<fmt:formatDate> タグは、書式設定の結果を出力します。var 属性を指定した場合、scope 属性では、var 属性で指定された変数のスコープが指定されます。

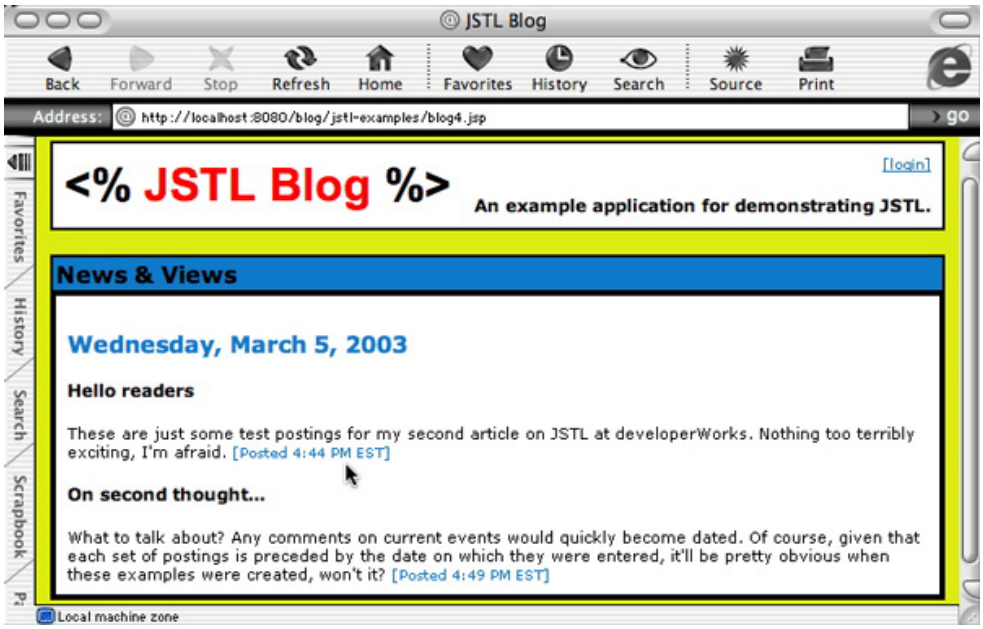
リスト1 (本シリーズ第2回のリスト8を拡張したもの) では、2つの<fmt:formatDate> タグを使用しています。1つ目の<fmt:formatDate> タグでは、最初のウェブログ項目の作成タイム・スタンプの日付部分だけを表示するように設定しています。また、dateStyle 属性にfull という値を指定しているため、すべての日付フィールドがロケール固有の書式で表示されます。

リスト1. <fmt:formatDate> タグを使用して日付値と時刻値を表示する

```
<table>
<fmt:timeZone value="US/Eastern">
<c:forEach items="${entryList}" var="blogEntry" varStatus="status">
<c:if test="${status.first}">
        <tr><td align="left" class="blogDate">
<fmt:formatDate value=
              "${blogEntry.created}" dateStyle="full"/>
        </td></tr>
</c:if>
      <tr><td align="left" class="blogTitle">
<c:out value="${blogEntry.title}" escapeXml="false"/>
      </td></tr>
    <tr><td align="left" class="blogText">
<c:out value="${blogEntry.text}" escapeXml="false"/>
      <font class="blogPosted">
        [Posted<fmt:formatDate value="${blogEntry.created}"
                               pattern="h:mm a zz"/>]
      </font>
    </td></tr>
</c:forEach>
  </fmt:timeZone>
</table>
```

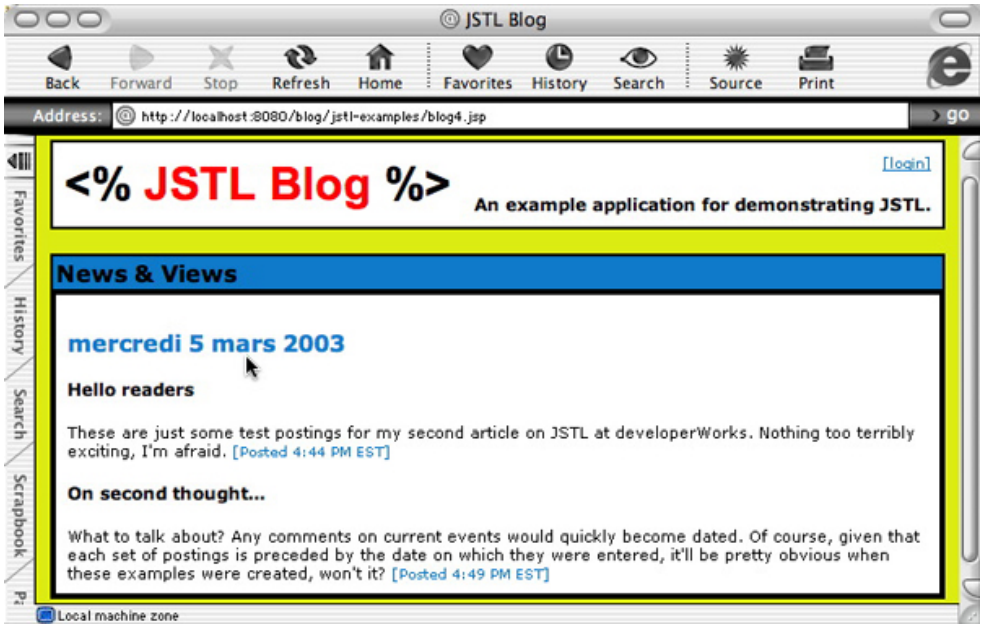
<c:forEach> ループの本文では、2つ目の<fmt:formatDate> アクションを使用して、各項目の作成日の時刻部分だけを表示するように設定しています。ここでは、pattern 属性を使用して時刻値の書式を設定しています。具体的には、時単位を1桁で表示すること (可能な場合)、12時間制で表示すること、および時間帯の略名を表示することを指定しています。リスト1の出力を図2に示します。

図2. リスト1に基づいたen_USロケールの出力



正確に言うと、図2に示す出力は、ユーザーのブラウザーで英語が優先言語として設定されている場合に生成されます。しかし、<fmt:formatDate> はユーザーのロケールに依存しているため、ブラウザーのロケール設定を変更すると、異なる内容が生成されます。たとえば、フランス語のロケールが優先設定されている場合は、代わりに、図3のような内容が出力されます。

図3. リスト1に基づいたfr_CAロケールの出力



`<fmt:formatDate>` では、`java.util.Date` インスタンスのローカライズ済み文字列表現が生成されるのに対して、`<fmt:parseDate>` アクションでは逆の処理が実行されます。すなわち、このアクションでは、日付や時刻を表す文字列を指定すると、対応する`Date` オブジェクトが生成されます。`<fmt:parseDate>` アクションには、次に示すように2種類の書式があります。

```
<fmt:parseDate value="expression"
    type="field" dateStyle="style" timeStyle="style"
    pattern="expression"
    timeZone="expression" parseLocale="expression"
    var="name" scope="scope"/>
<fmt:parseDate
    type="field" dateStyle="style" timeStyle="style"
    pattern="expression"
    timeZone="expression" parseLocale="expression"
    var="name" scope="scope">
body content
</fmt:parseDate>
```

1つ目の書式では、`value` 属性だけが必須であり、この値には、日付か時刻またはこれらの両方を表す文字列を指定します。2つ目の書式では、必須の属性はなく、構文解析対象の値を表す文字列は、`<fmt:parseDate>` タグの必須本文コンテンツとして指定します。

`type` 属性、`dateStyle` 属性、`timeStyle` 属性、`pattern` 属性、および`timeZone` 属性の役割は、`<fmt:parseDate>` でも`<fmt:formatDate>` の場合と同じですが、この場合、これらの属性では、日付値の表示ではなく構文解析について設定します。`parseLocale` 属性では、このタグの値を構文解析する際の基準となるロケールを指定し、その値には、ロケール名または`Locale` クラスのインスタンスを指定します。

`var` 属性と`scope` 属性では、`<fmt:parseDate>` の処理の結果として`Date` オブジェクトが代入されるスコープ付き変数を指定します。`var` 属性が指定されていない場合、結果は、`Date` クラス

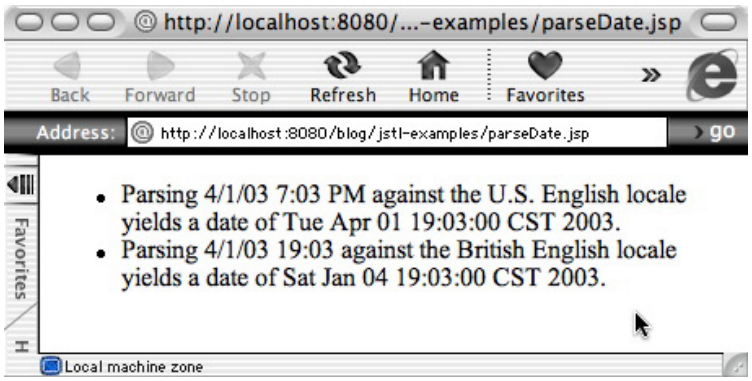
のtoString()メソッドを使用してJSPページに書き込まれます。リスト2には、<fmt:parseDate>アクションの記述例を示しています。

リスト2. <fmt:parseDate> タグを使用して日付と時刻の構文を解析する

```
<c:set var="usDateString">4/1/03 7:03 PM</c:set>
<fmt:parseDate value="${usDateString}" parseLocale="en_US"
               type="both" dateStyle="short" timeStyle="short"
               var="usDate"/>
<c:set var="gbDateString">4/1/03 19:03</c:set>
<fmt:parseDate value="${gbDateString}" parseLocale="en_GB"
               type="both" dateStyle="short" timeStyle="short"
               var="gbDate"/>
<ul>
<li> Parsing <c:out value="${usDateString}"/> against the
U.S. English
     locale yields a date of <c:out value="${usDate}"/>.</li>
<li> Parsing <c:out value="${gbDateString}"/> against the
British English
     locale yields a date of <c:out value="${gbDate}"/>.</li>
</ul>
```

リスト2の出力を図4に示します。

図4. リスト2の出力



<fmt:parseDate>で実行される構文解析は非常に厳密であるので注意が必要です。リスト2から分かるように、構文解析対象の値は、指定した (ロケール固有の) 書式やパターンに完全に準拠している必要があります。もちろんこれは、かなり厳しい制限です。その一方で、データの構文解析は、プレゼンテーション層に適した処理ではありません。実用コードの場合、テキスト入力の検証や変換は、JSPのカスタム・タグではなく、バックエンド・コード (サーブレットなど) で処理する方が適切です。

数値タグ

<fmt:formatDate>タグと<fmt:parseDate>タグでは、日付を対象にして書式設定と構文解析が実行されるのに対して、<fmt:formatNumber>タグと<fmt:parseNumber>タグでは、数値データを対象にして同様の処理が実行されます。

<fmt:formatNumber>タグを使用すると、金額やパーセント値などの数値データを、ロケール固有の書式で表示することができます。<fmt:formatNumber>アクションでは、たとえば数値の整数部と小数部の区切り文字としてピリオドかコンマのどちらを使用するかが、ロケールに基づいて決定されます。構文は次のとおりです。

```
<fmt:formatNumber value="expression"
    type="type" pattern="expression"
    currencyCode="expression" currencySymbol="expression"
    maxIntegerDigits="expression" minIntegerDigits="expression"
    maxFractionDigits="expression" minFractionDigits="expression"
    groupingUsed="expression"
    var="name" scope="scope"/>
```

<fmt:formatDate> の場合と同様に、value 属性だけが必須です。この属性では、書式設定対象の数値を指定します。var 属性とscope 属性の役割は、<fmt:formatNumber> アクションでも<fmt:formatDate> の場合と同じです。

type 属性の値には、number、currency、またはpercentage のいずれかの値を指定し、これにより、書式設定対象の数値のタイプを指定します。この属性のデフォルト値はnumber です。pattern 属性はtype 属性よりも優先されます。pattern 属性を使用すると、java.text.DecimalFormat クラスのパターン規則に従って、より厳密に数値を書式設定することができます。

type 属性の値がcurrency の場合は、currencyCode 属性を使用して、表示する数値の通貨単位を明示的に指定することができます。言語コードや国別コードと同様に、通貨コードもISO規格で規定されています (すべての有効なISO通貨コードへのリンクについては、参考文献を参照してください)。通貨コードに基づいて、書式設定後の値に付加表示される通貨記号が決定されます。

別の方法として、currencySymbol 属性を使用して通貨記号を明示的に指定することもできます。JDK 1.4およびこれに伴って追加されたjava.util.Currency クラスでは、currencySymbol 属性よりも<fmt:formatNumber> アクションのcurrencyCode 属性が優先されるので注意してください。しかし、これより前のバージョンのJDKでは、currencySymbol 属性が優先されます。

maxIntegerDigits 属性、minIntegerDigits 属性、maxFractionDigits 属性、およびminFractionDigits 属性では、小数点の前後に表示される有効桁数を指定します。これらの属性の値には、整数値を指定する必要があります。

groupingUsed 属性にはブール値を指定し、これにより、小数点より前の桁を分割するかどうかを指定します。たとえば、英語のロケールでは、大きな数値は3桁ずつに分割され、3桁ごとにコンマで区切られます。他のロケールでは、分割された桁はピリオドやスペースで区切られます。この属性のデフォルト値はtrue です。

リスト3に、リスト1 を拡張した簡単な通貨表示例を示します。ここでは、currencyCode 属性もcurrencySymbol 属性も指定していません。代わりに、通貨は、ロケール設定に基づいて決定されます。

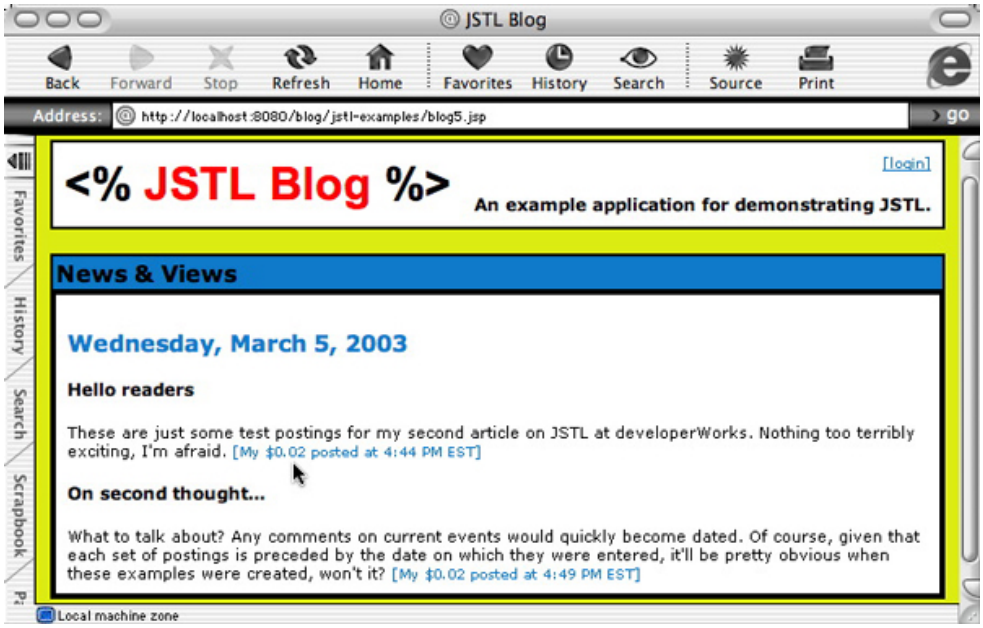
リスト3. <fmt:formatNumber> タグを使用して通貨値を表示する

```
<table>
<fmt:timeZone value="US/Eastern">
<c:forEach items="${entryList}" var="blogEntry"
varStatus="status">
<c:if test="${status.first}">
        <tr><td align="left" class="blogDate">
<fmt:formatDate value=
              "${blogEntry.created}" dateStyle="full"/>
        </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"/>
      <font class="blogPosted">
        [My<fmt:formatNumber value="0.02" type="currency"/>
         posted at<fmt:formatDate value="${blogEntry.created}"
                                  pattern="h:mm a zz"/>]
      </font>
    </td></tr>
</c:forEach>
  </fmt:timeZone>
</table>
```

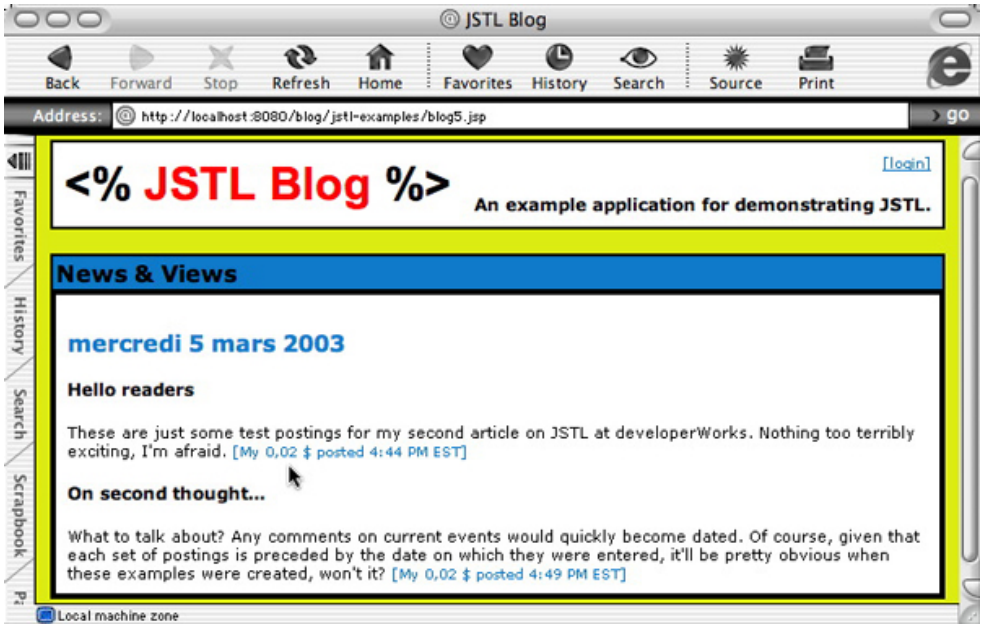
en_US ロケールの場合の出力を図5に示します。

図5. リスト3に基づいたen_USロケールの出力



fr_CA ロケールの場合の出力を図6に示します。

図6. リスト3に基づいたfr_CAロケールの出力



次に示す`<fmt:parseNumber>` アクションでは、その`value` 属性または本文コンテンツで指定した数値が、ロケール固有の方法で構文解析され、その結果が、`java.lang.Number` クラスのインスタンスとして返されます。`type` 属性と`pattern` 属性の役割は、`<fmt:parseNumber>` でも`<fmt:formatNumber>` の場合と同じです。同様に、`parseLocale` 属性、`var` 属性、および`scope` 属性の役割は、`<fmt:parseNumber>` でも`<fmt:parseDate>` の場合と同じです。

```
<fmt:parseNumber value="expression"
    type="type" pattern="expression"
    parseLocale="expression"
    integerOnly="expression"
    var="name" scope="scope"/>
<fmt:parseNumber
    type="type" pattern="expression"
    parseLocale="expression"
    integerOnly="expression"
    var="name" scope="scope">
body content
</fmt:parseNumber>
```

`<fmt:parseDate>` の説明で、データの構文解析はプレゼンテーション層に適した処理ではないことを述べましたが、これは、`<fmt:parseNumber>` の場合にも同様に当てはまります。データの構文解析と検証を、アプリケーションのビジネス・ロジックの一部として実装すると、ソフトウェアの保守が簡単になります。このため、実用のJSPページでは、原則的に`<fmt:parseDate>` と`<fmt:parseNumber>` を使用しないことをお勧めします。

`integerOnly` だけが、`<fmt:parseNumber>` に固有の属性です。この属性にはブール値を指定し、これにより、指定値の整数部だけを構文解析するかどうかを指定します。この属性の値が`true` の場合は、構文解析する文字列内の小数点以下の桁は解析対象にはなりません。この属性のデフォルト値は`false` です。

メッセージ・タグ

JSTLでテキストをローカライズするには、`<fmt:message>` タグを使用します。このタグを使用すると、ロケール固有のリソース・バンドルからテキスト・メッセージを取得して、JSPページに表示することができます。また、このアクションでは、`java.text.MessageFormat` クラスにより提供される機能が活用されるため、上記のようなテキスト・メッセージにパラメーター化された値を代入して、ローカライズ済みコンテンツを動的にカスタマイズすることができます。

ロケール固有のメッセージを格納するためのリソース・バンドルは、ベース名とロケール名を組み合わせるという標準命名規則に従ったクラスまたはプロパティー・ファイルの形で提供されます。たとえば、Weblogアプリケーションのクラスパス内の、`com.taglib.weblog` パッケージに対応するサブディレクトリーに格納されている`Greeting.properties` という名前のプロパティー・ファイルについて見てみましょう。このプロパティー・ファイルとして提供されているリソース・バンドルを英語とフランス語にローカライズするには、それぞれの言語コードを名前に付加した2つの新たなプロパティー・ファイルを同じディレクトリーに作成します。具体的には、これら2つのファイルには、それぞれ`Greeting_en.properties` と`Greeting_fr.properties` という名前を付けます。さらに、このリソース・バンドルをフランス語のカナダ方言にローカライズするには、該当する国別コードを名前に付加した3つ目のプロパティー・ファイルを作成します (`Greeting_fr_CA.properties` など)。

これらの各ファイルでは、同じプロパティーを定義しますが、これらのプロパティーの値は、対応する言語や方言に合わせてカスタマイズします。この方法については、`Greeting_en.properties` ファイルと`Greeting_fr.properties` ファイルのコンテンツ例を示したリスト4と5を参照してください。これらの例では、2つのローカライズされたメッセージを定義しています。これらのメッセージは、`com.taglib.weblog.Greeting.greeting` キーと`com.taglib.weblog.Greeting.return` キーで指定しています。ただし、これらのキーに関連付けられた値は、それぞれのファイル名で指定された言語にローカライズしています。`com.taglib.weblog.Greeting.greeting` メッセージの両方の値に含まれている`{0}` パターンを使用すると、パラメーター化された値を、コンテンツ生成時に動的にメッセージに挿入することができます。

リスト4. ローカライズ済みリソース・バンドルGreeting_en.propertiesのコンテンツ

```
com.taglib.weblog.Greeting.greeting=Hello {0}, and welcome to the JSTL Blog.
com.taglib.weblog.Greeting.return=Return
```

リスト5. ローカライズ済みリソース・バンドルGreeting_fr.propertiesのコンテンツ

```
com.taglib.weblog.Greeting.greeting=Bonjour {0}, et bienvenue au JSTL Blog.
com.taglib.weblog.Greeting.return=Retournez
```

このようなローカライズ済みコンテンツをJSTLを使用して表示するには、まずリソース・バンドルを指定します。`fmt` ライブラリーには、リソース・バンドルを指定するための`<fmt:setBundle>` と`<fmt:bundle>` という2つのカスタム・タグが含まれています。これらのタグの役割は、すでに紹介した`<fmt:setTimeZone>` タグおよび`<fmt:timeZone>` タグと似ています。`<fmt:setBundle>` アクションでは、特定スコープ内の`<fmt:message>` タグ用のデフォルト・リソース・バンドルを指定し、`<fmt:bundle>` では、その本文コンテンツ内でネストされているすべての`<fmt:message>` アクション用のリソース・バンドルを指定します。

次のコードでは、<fmt:setBundle> タグの構文を示しています。basename 属性は必須であり、これにより、デフォルトのリソース・バンドルを指定します。basename 属性の値には、ローカライズ・サフィックスもファイル名拡張子も含めてはいけません。リスト4と5に示しているリソース・バンドル例のベース名はcom.taglib.weblog.Greeting です。

```
<fmt:setBundle basename="expression"
    var="name" scope="scope"/>
```

任意指定のscope 属性では、デフォルト・リソース・バンドルの設定が適用されるJSPスコープを指定します。この属性を明示的に指定していない場合は、page という値が指定されているものと見なされます。

任意指定のvar 属性を指定した場合は、この属性で指定した変数に、basename 属性で指定したリソース・バンドルが代入されます。この場合、scope 属性では、この変数のスコープが指定され、デフォルト・リソース・バンドルは対応するJSPスコープに割り当てられません。

<fmt:bundle> タグでは、その本文コンテンツのスコープ内のデフォルト・リソース・バンドルを指定します。このタグの構文は次のとおりです。<fmt:setBundle> の場合と同様に、basename 属性だけが必須です。任意指定のprefix 属性では、すべてのネストされた<fmt:message> アクションのkey 値に付加するデフォルトのプレフィックスを指定します。

```
<fmt:bundle basename="expression"
prefix="expression">
body content
</fmt:bundle>
```

リソース・バンドルを指定したら、ローカライズ済みメッセージを実際に表示するのは、<fmt:message> タグの役割です。このアクションでは、ネストされた<fmt:param> タグが必要であるかどうかに応じて、次の2つの構文を使い分けることができます。

```
<fmt:message key="expression" bundle="expression"
    var="name" scope="scope"/>
<fmt:message key="expression" bundle="expression"
    var="name" scope="scope">
  <fmt:param value="expression"/>
  ...
</fmt:message>
```

<fmt:message> では、key 属性だけが必須です。key 属性の値に基づいて、リソース・バンドルで定義されているメッセージのいずれを表示するかが決定されます。

bundle 属性を使用すると、key 属性で指定したメッセージの検索先となるリソース・バンドルを明示的に指定することができます。この属性の値には、var 属性が指定された際に<fmt:setBundle> アクションで割り当てられたリソース・バンドルなど、実際のリソース・バンドルを指定する必要があります。<fmt:bundle> や<fmt:setBundle> のbasename 属性などの文字列値は、<fmt:message> のbundle 属性には指定できません。

<fmt:message> タグのvar 属性を指定した場合は、このタグで生成されるテキスト・メッセージは、JSPページに書き込まれる代わりに、var 属性で指定した変数に代入されます。これまでのタ

グと同様に、var 属性を指定した場合は、任意指定のscope 属性では、var 属性で指定された変数のスコープが指定されます。

<fmt:param> タグでは、テキスト・メッセージ用のパラメーター化された値をvalue 属性で指定します (必要な場合)。別の方法として、この値を、<fmt:param> タグの本文コンテンツで指定することもできます。この場合はvalue 属性を省略します。パラメーター化された値パターンが、リソース・バンドルから取得されたテキスト・メッセージに含まれている場合は必ず、<fmt:param> タグで指定した値が、java.text.MessageFormat クラスの動作に従ってこのメッセージに挿入されます。パラメーター化された値は、各値のインデックスによって指定されるため、ネストされた<fmt:param> タグの順番は重要です。

リスト6では、<fmt:bundle> タグ、<fmt:message> タグ、および<fmt:param> タグを組み合わせて使用しています。このリストの<fmt:bundle> タグでは、2つのネストされた<fmt:message> タグによって取得されるローカライズ済みメッセージの取得先バンドルを指定しています。これら2つの最初の<fmt:message> タグは、パラメーター化された値が1つ含まれたメッセージに対応しており、このタグ内には、この値に対応する<fmt:param> タグが含まれています。

リスト6. <fmt:message> タグを使用してローカライズ済みメッセージを表示する

```
<fmt:bundle basename="com.taglib.weblog.Greeting">
<fmt:message key="com.taglib.weblog.Greeting.greeting">
<fmt:param value="${user.fullName}"/>
</fmt:message>
  <br>
  <br>
  <center>
    <a href=
      "<c:url value='/index.jsp'/>"><fmt:message key="com.taglib.weblog.Greeting.return"/></a>
  </center>
</fmt:bundle>
```

リスト7では、<fmt:bundle> のprefix 属性を使用しています。prefix 属性で指定した値は、ネストされた<fmt:message> アクション内のすべてのkey 値の先頭に自動的に付加されます。したがって、リスト7はリスト6と同じ意味ですが、リスト7では、この便利な機能を利用しているため、2つの<fmt:message> タグ内のkey 値を略記しています。

リスト7. <fmt:bundle> のprefix属性を使用して <fmt:message> タグを簡素化する

```
<fmt:bundle basename="com.taglib.weblog.Greeting"
            prefix="com.taglib.weblog.Greeting.">
<fmt:message key="greeting">
<fmt:param value="${user.fullName}"/>
</fmt:message>
  <br>
  <br>
  <center>
    <a href="<c:url value='/index.jsp'/>"><fmt:message key="return"/></a>
  </center>
</fmt:bundle>
```

図7と8では、fmt ライブラリーのメッセージ関連タグを使用した出力を示しており、これらの出力は、リスト7のコードおよびリスト4 とリスト5 のローカライズ済みリソース・バンドルに基づいて生成されています。図7では、ブラウザーで英語ロケールが優先設定されている場合の出力を示しています。

図7. リスト7によるen_USロケールの出力

図8には、フランス語のロケールが設定されている場合の出力を示しています。

図8. リスト7によるfr_CAロケールの出力

まとめ

JSTLの`fmt`ライブラリー内のカスタム・タグを使用すると、JSP開発者は、Javaプラットフォームの国際化対応APIに簡単にアクセスすることができます。これにより、テキスト・メッセージ、数値、日付のすべてをロケールに応じた形式で表示できるとともに、時刻を特定の時間帯に合わせて調整することもできます。特定ユーザー向けのロケールは、ユーザーのブラウザー設定に基づいて自動的に決定することも、ページ作成者が明示的に指定することもできます。さらに、`fmt`ライブラリーには、書式設定されたデータを生成して表示するためのアクションだけでなく、数値データや時間関連データを構文解析する特製のタグも含まれています。

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。