JSF 2 の魅力: HTML5 複合コンポーネント: 第 2 回

ドラッグ・アンド・ドロップを実装する

David Geary

2010年 11月 23日

連載「JSF 2 の魅力」の今回の記事では、著者の David Geary が前回に引き続き、JSF (Java™Server Faces) 2 技術と HTML5 を組み合わせることによってもたらされる威力を紹介します。今回は、HTML5 のドラッグ・アンド・ドロップをカプセル化した JSF 複合コンポーネントを実装する方法を説明します。

このシリーズの他の記事を見る

この連載について

連載「JSF 2 の魅力」では、David Geary が JSF 2 について紹介する 3 回からなる同名の連載の続編として、皆さんが JSF 2 フレームワークのスキルを開発して磨きをかけるためのお手伝いをします。この連載では JSF 2 フレームワークとそれを取り巻くエコシステムの詳細を探るとともに、このフレームワークの外部にも目を向け、CDI (Contexts and Dependency Injection) などの Java EE 技術を JSF に統合する方法についても紹介します。

この記事を読む前に、連載「JSF 2 の魅力」の前回の記事を必ず読んでください。前回の記事では HTML5 の機能を紹介し、HTML5 キャンバスを簡単に使えるようにする JSF 複合コンポーネントを作成する方法を説明しました。今回の記事では HTML5 のイベント・ベースのドラッグ・アンド・ドロップ・メカニズム (「参考文献」を参照) を利用した 2 つの JSF 複合コンポーネント、`<h5:drag>` と `<h5:drop>` を実装する方法を説明します。

この 2 つのドラッグ・アンド・ドロップ複合コンポーネントが持つ顕著な特徴は、以下の 5 つです。

- 使いやすさ
- 条件付きドラッグ
- Ajax
- 部分レンダリング
- ペイロード

最も基本的なレベルで言うと、`<h5:drag>` と `<h5:drop>` はどちらも HTML5 のドラッグ・アンド・ドロップに伴う特異性をある程度カプセル化して、この機能を使いやすくします。例えば、ブラウザーはデフォルトでドラッグ・アンド・ドロップを拒否します。そのため、drag-enter および drag-over イベント・ハンドラー内でそれぞれのイベントをキャンセルすることで、ブラウザーが

ドラッグ・アンド・ドロップに対してこのようなデフォルト動作をしないようにする必要があります。こうした直観で理解するのが困難な落とし穴については `<h5:drag>` コンポーネントの中で対処することによって、ページ作成者がより意味のある作業に専念できるようにします。

`<h5:drag>` および `<h5:drop>` コンポーネントは条件付きドラッグをサポートします。これにより、ページ作成者は転送データとドロップ・ターゲットに応じて、ドロップを許容または拒否することができます。

サンプル・コードの実行

この連載で使用するコードは、GlassFish や Resin などのエンタープライズ・コンテナー内で実行される JSF 2 をベースとしています。GlassFish でコードをインストールおよび実行する方法については、ステップバイステップのチュートリアルになっている連載の最初の記事「JSF 2 fu: Ajax components」を参照してください。また、この記事のサンプル・コードを入手するには、「ダウンロード」を参照してください。

JSF アプリケーションでは、ユーザーが操作するほとんどのデータは一般に MBean としてサーバー上に保管されます。このため、`<h5:drop>` コンポーネントはドロップを受け入れるときに Ajax 呼び出しを行います。ページ作成者は Ajax 呼び出しがリターンした時点で JSF にレンダリングさせるコンポーネントを指定することができます。

`<h5:drag>` コンポーネントと `<h5:drop>` コンポーネントは、ドラッグ・アンド・ドロップ操作へのデータの添付 (一般にペイロードと呼ばれます) もサポートします。ペイロードとなる Bean プロパティーは、ページ作成者が指定します。ドロップが発生すると、`<h5:drop>` コンポーネントによって Ajax 呼び出しが開始されます。この Ajax 呼び出しの際に、JSF はペイロードの Bean プロパティーのセッター・メソッドを呼び出します。したがって、JSF はドラッグ・アンド・ドロップのペイロードを `<h:inputText>` 要素の値と同じように処理します。

ドラッグ・ソースとドロップ・ターゲットの使用方法

`<h5:drag>` と `<h5:drop>` はそれぞれ、HTML5 のドラッグ・ソース、ドロップ・ターゲットを表します。この 2 つのコンポーネントを使用する JSF アプリケーションでは、以下のようにドラッグ・ソースを使用します。

```
<script>
   function dragStart(event) {
      event.dataTransfer.setData('text', "transfer this string");
   }
</script>

<h5:drag ondragstart="dragStart(event)">

   ...

</h5:drag>
```

ページ作成者は `<h5:drag>` コンポーネント内に複数のコンポーネント (HTML 要素) を組み込み、上記のマークアップで行っているように `ondragstart` 関数に転送データをセットアップすることができます。

ドロップ・ターゲットは以下のように使用します。

```
<h5:drop id="dropzone"
payload="#{dragDrop.payload}"
render="@this"

   ...

</h5:drop>
```

ドラッグ・ソースの場合と同じように、ページ作成者は `<h5:drop>` コンポーネントに複数のコンポーネント、つまり HTML 要素を組み込むことができます。ページ作成者は、ドラッグ・アンド・ドロップのペイロードとなる Bean プロパティー、そしてドロップ後に実行される Ajax 呼び出しがサーバーからリターンした時点で JSF にレンダリングさせるコンポーネントも指定します。

ページ作成者はオプションで、JavaScript を使用してドラッグ・アンド・ドロップ操作に介入することもできます。

```
<h5:drop id="dropzone"
    payload="#{dragDrop.payload}"
     render="@this"
ondragover="dragover(event)"
ondragenter="dragenter(event)"
ondragleave="dragleave(event)"
ondrop="drop(event)">

   ...

</h5:drop>
```

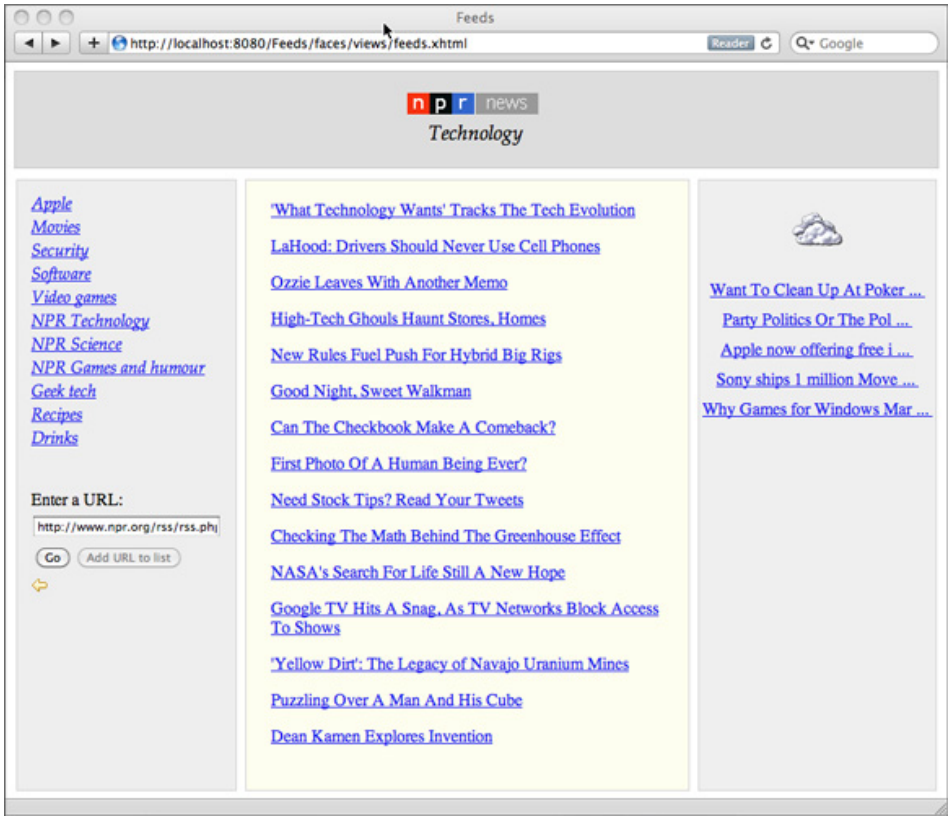
上記のマークアップでは、`dragover()`、`dragenter()`、`dragleave()`、および `drop()` 関数が (表示されていませんが) ページ作成者によって実装されています。

この記事の目指す方向がわかったところで、次のセクションではこの記事でのドラッグ・アンド・ドロップの使用例として、Feeds アプリケーションについて説明します。

Feeds アプリケーション

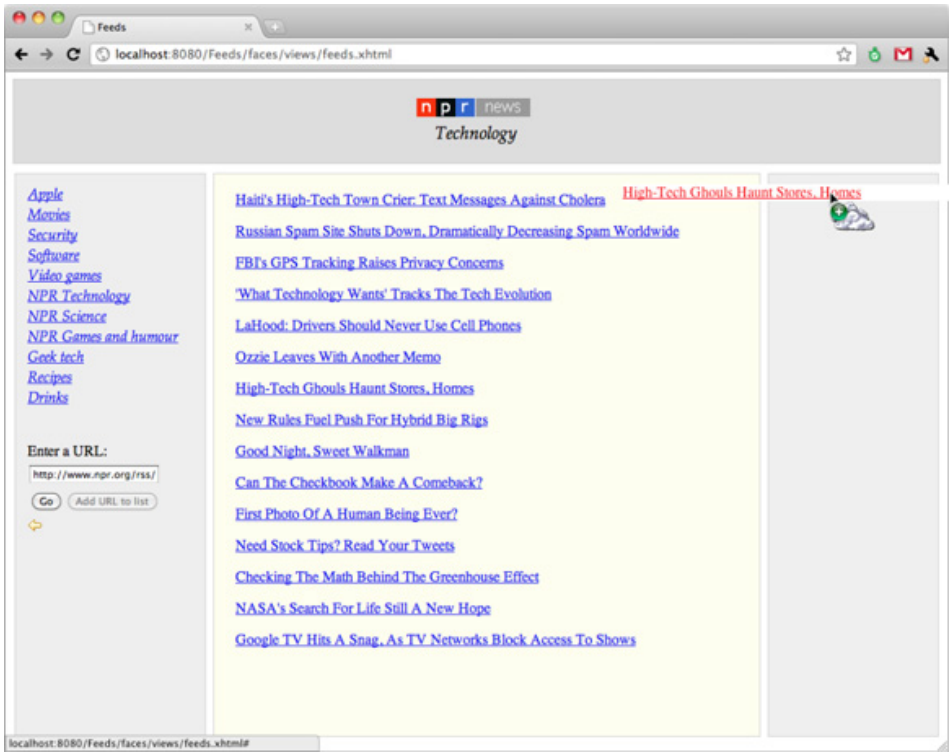
図 1 に示す Feeds アプリケーションは、RSS フィード・リーダーです。左側のメニューに表示される RSS フィードの一覧には、ユーザーがフィードを追加できるようになっています。ページの中央に示されるのは、最新フィードの記事へのリンクです。ユーザーが記事のリンクをクリックすると、アプリケーションがリンクに関連付けられた記事をブラウザーにロードします。その後、ユーザーが「戻る」ボタンを押せば、アプリケーションに戻ることができます。

図 1. Feeds アプリケーション



RSS フィードの記事のリストは定期的に更新されるので、時間のないユーザーは後で読めるようにリンクを保存できるようになっています。保存するには、アプリケーションの中央に表示されている記事のリンクを右側のメニューにドラッグします (図 2 を参照)。

図 2. Feeds アプリケーションでのドラッグ・アンド・ドロップ



このアプリケーションには、RSS フィードを読み取って RSS 項目のリストを表示する MBean があります (リスト 1 を参照)。

リスト 1. RSS フィードの取得と構文解析

```
package com.clarity;

import java.io.Serializable;

import java.net.URL;
import java.util.LinkedList;

import org.gnu.stealthp.rsslib.RSSChannel;
import org.gnu.stealthp.rsslib.RSSHandler;
import org.gnu.stealthp.rsslib.RSSItem;
import org.gnu.stealthp.rsslib.RSSParser;
import javax.enterprise.context.SessionScoped;
import javax.inject.Named;

@Named("rssFeed")@SessionScoped
public class RSSFeed implements Serializable {
   private static final long serialVersionUID = 2L;

   private String feed, displayName;
   private RSSChannel channel;
   private LinkedList<RSSItem> savedItems = new LinkedList<RSSItem>();

   public void fetch(String f, String dn) {
      assert f != null;
      assert dn != null;

      feed = f;
      displayName = dn;

      RSSHandler handler = new RSSHandler();
```

```
      channel = handler.getRSSChannel();

      try {
RSSParser.parseXmlFile(new URL(feed), handler, true);
      } catch (Exception e) {
         channel = null;
         e.printStackTrace();
      }
   }

   public LinkedList<RSSItem> getItems() {
      return channel == null ? null : channel.getItems();
   }
   public LinkedList<RSSItem> getSavedItems() {
      return savedItems;
   }

   public RSSChannel getChannel() { return channel; }
   public String getFeed() { return feed; }
   public String getDisplayName() { return displayName; }
}
```

RSSFeed クラスは RSSLib4J を使用することで、RSS フィードを簡単に取得して構文解析できるようにしています (「参考文献」を参照)。リスト 1 の @Named 属性と @SessionScoped 属性により、アプリケーションにはセッション・スコープの MBean として、RSSFeed のインスタンスである rssFeed が設定されます。

アプリケーションの左側のメニューに表示されているすべてのリンクには、rssFeed.fetch() を呼び出すアクションがあります。例えば、左側のメニューにある Apple リンクは、以下のように実装されています。

```
<h:commandLink value="Apple"
  action="#{rssFeed.fetch('http://rss.news.yahoo.com/rss/applecomputer',
                          'Apple Computer')}"/>
```

ユーザーがこのリンクをクリックすると、JSF は rssFeed MBean の fetch() メソッドを呼び出し、続いてアプリケーションの中央に表示されたリンクのリストをリロードします。

```
<ui:repeat value="#{rssFeed.items}" var="item">
  <h5:drag ondragstart="dragStart(event)">
   <a href="#{item.link}">#{item.title}</a>
  </h5:drag>
<ui:repeat>
```

アプリケーションは、右側のメニューに保存済みリンクも表示します。

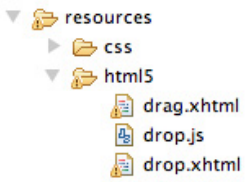
```
<ui:repeat value="#{rssFeed.savedItems}" var="item">
   <a href="#{item.link}">
      #{ fn:substring(#{item.title}, 0, 25) } ...
   </a>
<ui:repeat>
```

Feeds アプリケーションが RSS フィードから項目を取得して表示する方法は以上のとおりです。ここからは、この記事のメイン・イベントに目を向けます。

<h5:drag> および <h5:drop> コンポーネント

`<h5:drag>` コンポーネントと `<h5:drop>` コンポーネントは全部で 3 つのファイルに実装されます。具体的に言うと、コンポーネントごとのファイル (drag.xhtml および drop.xhtml) と、`<h5:drop>` コンポーネントの JavaScript 用のファイル (drop.js) です (図 3 を参照)。

図 3. **<h5:drop>** および **<h5:drag>** コンポーネントのファイル



これらのコンポーネントは Ajax 化したドラッグ・アンド・ドロップを容易にするものの、その実装は大規模なものではありません。どちらのコンポーネントも facelets マークアップと JavaScript だけで実装されており、マークアップは約 100 行、JavaScript は約 50 行を数えるに過ぎません。以降のセクションでは以下の 3 つの作業にわけて、コンポーネントの開発方法を説明します。

- クライアントでのドラッグ・アンド・ドロップの実装
- ドロップ発生時の Ajax 呼び出しの追加
- 条件付きドラッグのサポート

クライアントでのドラッグ・アンド・ドロップ

まず始めは、クライアントでの作業に専念します。クライアントではただ単に、ユーザーがリンクをドロップ・ターゲットにドロップすると、記事のタイトルに続けてその URL を示すアラートを表示します (図 4 を参照)。

図 4. ドロップ・ターゲットにドロップされたリンクを表示するアラート

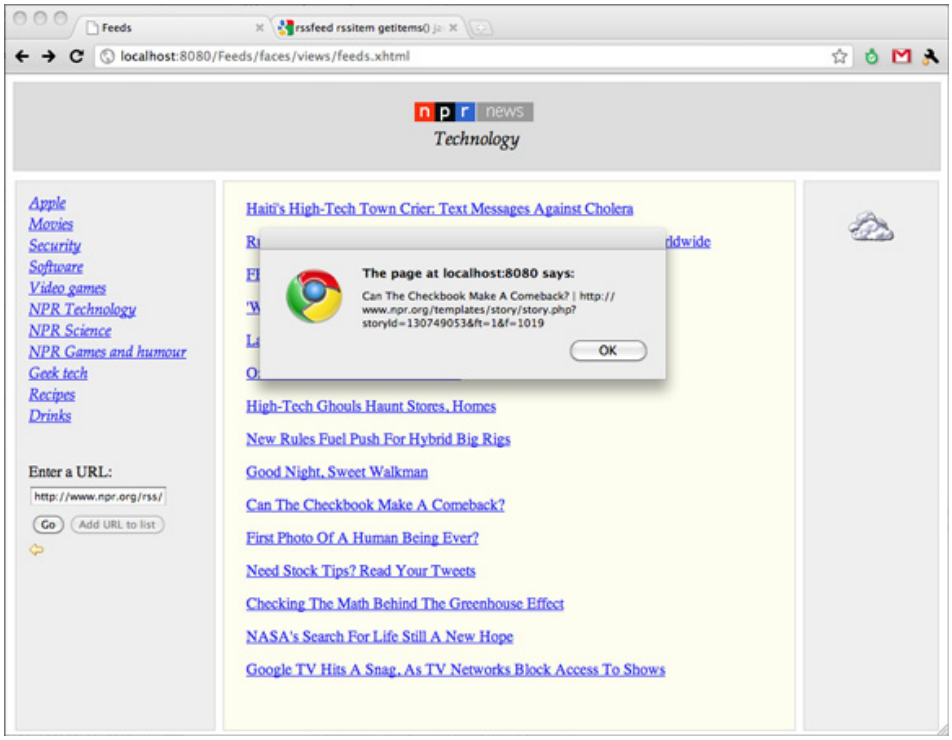


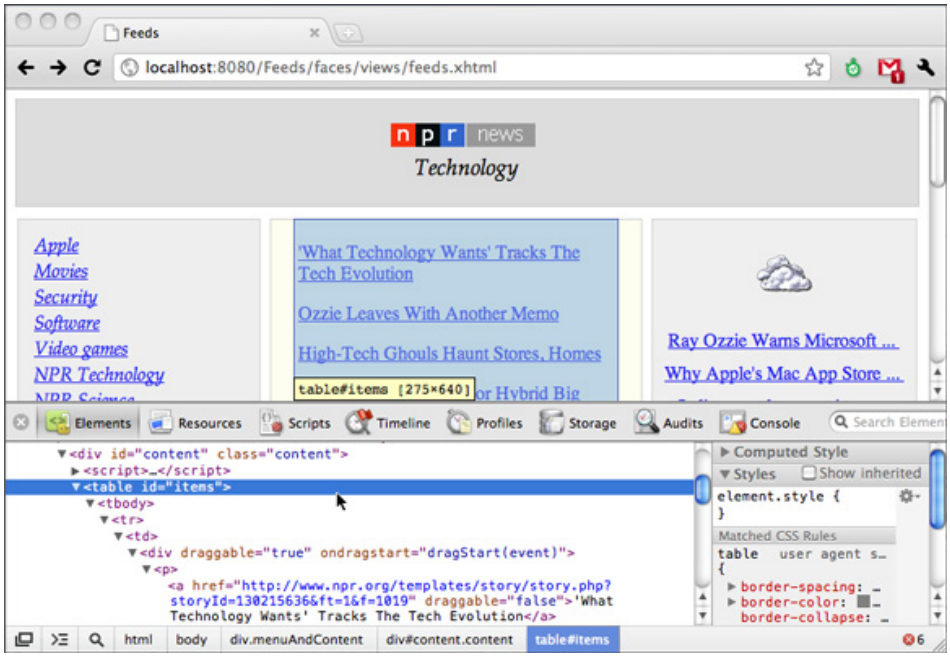
図 4 のドラッグ・アンド・ドロップの実装方法を説明するため、以下の成果物を順に見ていきます。

- ドラッグ・ソース
- ドラッグ・ソース・コンポーネント
- ドロップ・ターゲット
- ドロップ・ターゲット・コンポーネント
- ドロップ・ターゲット・コンポーネントの JavaScript

ドラッグ・ソース

図 5 に強調表示されているアプリケーション中央のリンクのそれぞれが、ドラッグ・ソースです。

図 5. ドラッグ・ソース



リスト 2 に、ドラッグ・ソースのマークアップと JavaScript を記載します。

リスト 2. ドラッグ・ソースのマークアップと JavaScript

```
<ui:composition xmlns="http://www.w3.org/1999/xhtml"
   xmlns:h="http://java.sun.com/jsf/html"
   xmlns:ui="http://java.sun.com/jsf/facelets"
   xmlns:h5="http://java.sun.com/jsf/composite/html5"
   xmlns:places="http://java.sun.com/jsf/composite/places">

   <script>
      // event.target is one of the h5:drag elements generated by ui:repeat below
       function dragStart(event) {
          var linkref = event.target.firstElementChild.firstElementChild; // anchor
          var link = linkref.href;
          var title = linkref.textContent;

          event.dataTransfer.setData('text', title + " | " + link + " ");
      }
    </script>


   <h:panelGrid id="items" columns="1" >
     <ui:repeat value="#{rssFeed.items}" var="item">


      <h5:drag ondragstart="dragStart(event)">
        <p><a href="#{item.link}">#{item.title}</a> <br /></p>
      </h5:drag>

     </ui:repeat>
   </h:panelGrid>

</ui:composition>
```

リスト 2 のマークアップは `<h5:drag>` コンポーネントを使用して、ドラッグの開始時に JSF に呼び出させる JavaScript 関数を指定しています。指定されたこの JavaScript 関数に、ブラウザーはイ

ベントを渡します。イベントのターゲットは、`<h5:drag>` コンポーネントです。関数は `<h5:drag>` 要素からアンカー要素までドリルダウンして、リンクのテキストと参照を取得します。そしてこの情報を、データ転送タイプ `text` と関連付けられたストリングに埋め込みます。その後は HTML5 のドラッグ・アンド・ドロップ・システムが、ドロップが受け入れられた時点でこのストリングをドロップ・ターゲットに転送します。

ドラッグ・ソース・コンポーネント

`<h5:drag>` コンポーネントの実装は、リスト 3 のとおり単純なものです。

リスト 3. `<h5:drag>` コンポーネント

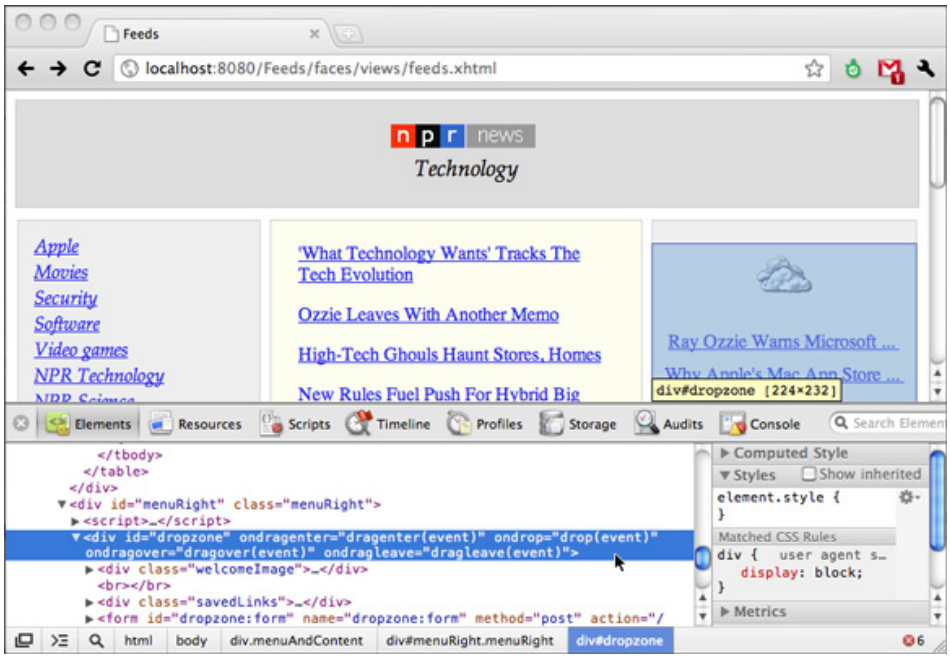
```
<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml"
   xmlns:h="http://java.sun.com/jsf/html"
   xmlns:composite="http://java.sun.com/jsf/composite">

   <composite:interface>
     <composite:attribute name="ondragstart"/>
   </composite:interface>

   <composite:implementation>
     <div draggable="true"
         ondragstart="#{cc.attrs.ondragstart}">
        <composite:insertChildren />
     </div>
   </composite:implementation>
</html>
```

`<h5:drag>` コンポーネントはドラッグ可能な DIV を作成します。この `DIV` の `ondragstart` JavaScript は、リスト 2 でページ作成者が指定した値です。さらに、このコンポーネントは `<composite:insertChildren>` を使用して、あらゆるマークアップを `<h5:drag>` タグの本体に挿入します。リスト 2 でそのマークアップに該当するのはリンクのアンカーです。

ドロップ・ターゲット

ドロップ・ターゲット (図 6 を参照) は、アプリケーションの右側のメニューにあります。

図 6. ドロップ・ターゲット



リスト 4 に、ドロップ・ターゲットの実装を記載します。

リスト 4. ドロップ・ターゲット

```
<ui:composition xmlns="http://www.w3.org/1999/xhtml"
   xmlns:ui="http://java.sun.com/jsf/facelets"
   xmlns:f="http://java.sun.com/jsf/core"
   xmlns:h="http://java.sun.com/jsf/html"
   xmlns:util="http://java.sun.com/jsf/composite/util"
   xmlns:fn="http://java.sun.com/jsp/jstl/functions"
   xmlns:h5="http://java.sun.com/jsf/composite/html5">

   <script>
        function drop(event) { alert(event.dataTransfer.getData("text")); }
    </script>

   <h5:drop id="dropzone"
      ondrop="drop(event)">

      <div class="welcomeImage">
          <h:graphicImage id="welcomeImage"
             library="images" name="cloudy.gif"/>
      </div>

   </h5:drop>
</ui:composition>
```

上記のドロップ・ターゲットは `<h5:drop>` コンポーネントからなります。このコンポーネントに含まれるのは、`DIV` 内にラップされた雲の画像です。`<h5:drop>` コンポーネントの `ondrop` 属性が参照する JavaScript 関数が、ドラッグ・ソースから転送されたストリングを含んだアラートを表示します。

ドロップ・ターゲット・コンポーネント

`<h5:drop>` コンポーネントの実装は、リスト 5 のとおりです。

リスト 5. `<h5:drop>` コンポーネント

```
<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml"
   xmlns:f="http://java.sun.com/jsf/core"
   xmlns:h="http://java.sun.com/jsf/html"
   xmlns:composite="http://java.sun.com/jsf/composite">

   <composite:interface>
      <composite:attribute name="ondragenter"/>
      <composite:attribute name="ondragover"/>
      <composite:attribute name="ondragleave"/>
      <composite:attribute name="ondrop"/>
   </composite:interface>


   <composite:implementation>
      <div id="#{cc.id}" ondragenter="#{cc.attrs.ondragenter}"
                         ondrop="#{cc.attrs.ondrop}"
                         ondragover="#{cc.attrs.ondragover}"
                         ondragleave="#{cc.attrs.ondragleave}">

        <composite:insertChildren />

      </div>

      <script> html5.jsf.init("#{cc.id}"); </script>
   </composite:implementation>
</html>
```

`<h5:drop>` コンポーネントも `<h5:drag>` のように `DIV` を作成し、その `DIV` に、`<h5:drop>` タグの本体に含まれるマークアップを挿入します。

`<h5:drop>` コンポーネントは、作成されると同時に `html5.jsf.init()` を呼び出します。この関数が、drag-enter イベント・ハンドラーと drag-over イベント・ハンドラーを追加してコンポーネントの `DIV` を初期化します。リスト 6 に、この両方のイベント・ハンドラーが実装されています。

リスト 6. `<h5:drop>` コンポーネントの JavaScript

```
if (!html5)
   var html5 = {}
if (!html5.jsf) {
html5.jsf = {
init : function(ccid) {
         var dropzone = $(ccid);

         dropzone.addEventListener("dragenter", function(event) {
            event.preventDefault();
         }, false);

         dropzone.addEventListener("dragover", function(event) {
            event.preventDefault();
         }, false);
      }
   };
}
```

リスト 6 の JavaScript (名前の衝突を避けるため、名前空間を設定してあります) によって、drag-enter および drag-over イベントに対するブラウザーのデフォルトの動作が行われるのを防ぐことができます。ブラウザーはデフォルトでは、これらのイベントを受け付けないことを思い出して

ください。そのため、リスト 6 の JavaScript で、ブラウザーがドロップを拒否しないようにしているというわけです。`preventDefault()` 呼び出しを行わなければならないことが多少不可解であるのは間違いありませんが、だからこそ再利用可能なコンポーネントにカプセル化するのにふさわしい候補であると言えます。

この JavaScript は無条件でブラウザーが drag-over および drag-enter イベントを取り消さないようにします。したがって、すべてのドロップを無条件で受け入れるということですが、それでは実用には即していません。この問題には「条件付きドラッグ」のセクションで対処するとして、次はドロップ・ターゲット・コンポーネントに Ajax を追加する方法を説明します

Ajax の追加とサーバーへのペイロードの送信

「クライアントでのドラッグ・アンド・ドロップ」セクションでは、リンクのタイトルと URL をアラートで表示するという作業により、ドロップされたリンクをクライアント上でのみ処理しました (図 4 を参照)。Feeds アプリケーションだけでなく、ドラッグ・アンド・ドロップ機能を提供する大半の単純ではない JSF アプリケーションでは、サーバー・サイドのデータにペイロードを組み込むために、ドロップ操作にサーバーへの転送を伴うことがよくあります。この要件に対処するため、このセクションでは `<h5:drop>` コンポーネントに Ajax を追加して、ドロップが行われるとコンポーネントが自動的に Ajax 呼び出しを行い、Ajax 呼び出しと一緒にペイロードが送信されるようにします。

具体的には、ユーザーが右側のメニューにリンクをドロップするたびに、`<h5:drop>` コンポーネントが Ajax 呼び出しを行います。この呼び出しにより、ドロップされたリンクをサーバー上のアプリケーションの保存済みリンクのリストに追加します。Ajax 呼び出しがリターンすると、JSF はドロップ・ターゲットを更新して新しく追加されたリンクを反映します。

リスト 7 に、更新後のドロップ・ターゲットを記載します。

リスト 7. ドロップ・ターゲット (テイク II)

```
<ui:composition xmlns="http://www.w3.org/1999/xhtml"
   xmlns:ui="http://java.sun.com/jsf/facelets"
   xmlns:h="http://java.sun.com/jsf/html"
   xmlns:fn="http://java.sun.com/jsp/jstl/functions"
   xmlns:h5="http://java.sun.com/jsf/composite/html5">

   <script>
        function dragenter(event) { /* Implement as desired */ }
        function dragleave(event) { /* Implement as desired */ }
        function dragover(event)  { /* Implement as desired */ }
        function drop(event)      { /* Implement as desired */ }
    </script>

   <h5:drop id="dropzone" payload="#{dragDrop.payload}"
render="@this"
      ondragover="dragover(event)"
      ondragenter="dragenter(event)"
      ondragleave="dragleave(event)"
      ondrop="drop(event)">

      <div class="welcomeImage">
          <h:graphicImage id="welcomeImage"
             library="images" name="cloudy.gif"/>
      </div>
```

```
      <br />

      <div class="savedItems">
           <ui:repeat value="#{rssFeed.savedItems}" var="item">
             <div class="savedLink">
              <a href="#{item.link}">
                 #{ fn:substring(item.title, 0, 25) } ...
               </a>

               <br/>
           </div>
         </ui:repeat>
      </div>

   </h5:drop>
</ui:composition>
```

更新後のドロップ・ターゲットでは、ドラッグ・アンド・ドロップのペイロードを `#{dragDrop.payload}` という Bean プロパティーに関連付けました。そして、ドロップによって開始された Ajax 呼び出しがリターンした時点で `@this` (ドロップ・ターゲット自体を意味します) をレンダリングするよう、ドロップ・ターゲットに対して指示をしています。

リスト 8 に、更新後の `<h5:drop>` コンポーネントの実装を記載します。

リスト 8. `<h5:drop>` コンポーネント (テイク II)

```
<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml"
   xmlns:f="http://java.sun.com/jsf/core"
   xmlns:h="http://java.sun.com/jsf/html"
   xmlns:composite="http://java.sun.com/jsf/composite">

   <composite:interface>
      <composite:attribute name="ondragenter"/>
      <composite:attribute name="ondragover"/>
      <composite:attribute name="ondragleave"/>
      <composite:attribute name="ondrop"/>
      <composite:attribute name="render"/>
      <composite:attribute name="payload"/>
      <composite:attribute name="payloadType"/>
   </composite:interface>

   <composite:implementation>
      <h:outputScript library="javax.faces" name="jsf.js" target="head" />
      <h:outputScript library="html5" name="drop.js" target="head" />

      <div id="#{cc.id}" ondragenter="#{cc.attrs.ondragenter}"
                         ondrop="#{cc.attrs.ondrop}"
                         ondragover="#{cc.attrs.ondragover}"
                         ondragleave="#{cc.attrs.ondragleave}">


        <composite:insertChildren />

        <h:form id="form">
          <h:inputText id="payload"
value="#{cc.attrs.payload}"
style="display: none"/>
        </h:form>

        </div>

        <script> html5.jsf.init("#{cc.id}",
```

```
                                "#{cc.attrs.payloadType}",
                                "#{cc.attrs.render}"); </script>
   </composite:implementation>
</html>
```

更新後のドロップ・ターゲットは、以下の動作をしなければなりません。

- ドロップの発生時に Ajax 呼び出しを行う
- Ajax 呼び出しの間、サーバー上でペイロードを使用可能にする

`<h5:drop>` コンポーネントの JavaScript (リスト 9 を参照) では、JSF の `jsf.ajax.request()` JavaScript 関数を使用して Ajax 呼び出しを行っています。

リスト 9. `<h5:drop>` コンポーネントの JavaScript (テイク II)

```
if (!html5)
   var html5 = {}
if (!html5.jsf) {
   html5.jsf = {
      init : function(ccid, payloadType, renderIds) {
         var dropzone = $(ccid);

         dropzone.payloadInput = $(ccid + ":form:payload");

         dropzone.addEventListener("drop", function(event) {
            if (payloadType == "")
               payloadType = "text";

            if (renderIds == "" || renderIds == "@this")
               renderIds = ccid;


dropzone.payloadInput.value = event.dataTransfer
                  .getData(payloadType);jsf.ajax.request(dropzone.payloadInput, event, {
               render : renderIds,
               onevent : function(data) {
                   if (data.status == "success")
                      html5.jsf.init(ccid, payloadType, renderIds);
                }
            });
         }, false);

         dropzone.addEventListener("dragenter", function(event) {
            event.preventDefault();
         }, false);

         dropzone.addEventListener("dragover", function(event) {
            event.preventDefault();
         }, false);
      }
   };
}
```

リスト 7 では、ペイロードが以下のように指定されていたことを思い出してください。

```
<h5:drop...payload="#{dragDrop.payload}">
```

ペイロードを使用可能にするために使用しているのは、リスト 8 の `<h5:drop>` コンポーネントに含まれる非表示テキスト入力フィールドです。ペイロードをクライアントに転送するには、クライアントとサーバーとの間で値を受け渡しする手段が必要ですが、JSF にはすでにその手段

が `<h:inputText>` によって提供されています。そこで、非表示テキスト入力フィールドをドロップ・ターゲットに追加し、リスト 9 の JavaScript からわかるように、ドロップが行われて Ajax リクエストが実行される直前に、この非表示テキスト入力フィールドの値を設定します。

入力の値は Ajax 呼び出しを行う前に設定するので、入力の値に保管されたドラッグ・アンド・ドロップのペイロードが、Ajax 呼び出しの際にサーバーで使用できるようになります。テキスト入力フィールドは Bean プロパティーに関連付けられているため、JSF は Ajax 呼び出し中に、ペイロードをそのプロパティーのセッター・メソッドに渡します。

その値が、リスト 8 の `<h5:drop>` コンポーネントで非表示テキスト入力フィールドの値として使用されます。

```
<h:inputText id="payload" value="#{cc.attrs.payload}" style="display: none"/>
```

リスト 9 では、非表示テキスト入力フィールドが Ajax リクエストのソースとして指定されています。

```
jsf.ajax.request(dropzone.payloadInput, event, {...});
```

非表示テキスト入力が Ajax リクエストのソースであることから、JSF はこの入力をサーバー上で処理します。これはつまり、テキスト入力フィールドに関連付けられたプロパティーのセッター・メソッドを呼び出すことを意味します。この例の場合、これに該当するメソッドは `DragDrop.setPayload()` です (リスト 10 を参照)。

リスト 10. **payload** プロパティーの実装

```
package com.clarity;

import java.util.Iterator;
import java.util.LinkedList;
import java.util.StringTokenizer;

import javax.enterprise.context.SessionScoped;
import javax.inject.Inject;
import javax.inject.Named;

import org.gnu.stealthp.rsslib.RSSItem;

@Named
@SessionScoped
public class DragDrop implements Serializable {
@Inject private RSSFeed rssFeed;

   public DragDrop() {
   }

   public String getPayload() {
      // JSF requires both getters and setters for input properties
      return "";
   }

   public void setPayload(String payload) {
      // creates a new saved item, based on the payload. Payload
      // was set in the drop event listener for the h5:drop component
      // in /sections/feeds/menuLeft.xhtml
      StringTokenizer st = new StringTokenizer(payload);
      RSSItem item = new RSSItem();
```

```
        item.setTitle(st.nextToken("|"));
        st.nextToken(" ");
        item.setLink(st.nextToken(" "));

        rssFeed.getSavedItems().add(item);
    }
}
```

JSF はドラッグ・アンド・ドロップのペイロードを `DragDrop.setPayload()` に渡します。`DragDrop.setPayload()` メソッドは渡されたペイロードに基づき、新しい RSS 項目を作成して、その項目を `rssFeed` の保存済み項目リストに追加します。payload プロパティーにはゲッター・メソッドも組み込んでいることに注意してください。そのようにしたのは、JSF では入力値のセッターとゲッターの両方が必要になるためです。

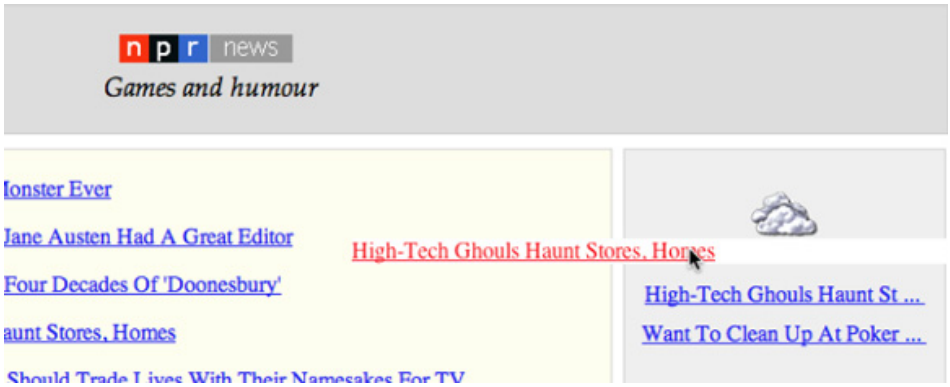
既存のコンポーネントの機能は、複合コンポーネントでは簡単に再利用することができます。この例では、クライアントとサーバーの間でデータを転送する `<h:inputText>` の機能を再利用します。`<h5:drop>` コンポーネントの非表示テキスト入力フィールドを使用してクライアントからサーバーにデータを転送するには、テキスト入力フィールドを複合コンポーネントに追加し、Ajax 呼び出しを行う前に入力の値を設定するだけでよいのです。

これで、ドロップに応答して Ajax 呼び出しを行い、クライアントから転送されたデータをサーバーに送信するという、かなり高度なドロップ・ターゲットになりました (ドラッグ・ソースは導入された当初のまま、変更されていないことに注意してください)。このドロップ・ターゲットでは、Ajax 呼び出しがリターンした時点でレンダリングするコンポーネントをページ作成者が指定することもできます。しかし、このドラッグ・アンド・ドロップ・コンポーネントにはまだ 1 つ欠けている機能があります。それは、条件付きドラッグです。

条件付きドラッグ

リスト 1 では、Feeds アプリケーションの保存済みリンクのリストをリンク付きリストとして実装しました。これは、ユーザーが同じ記事を重複して追加する可能性があることを意味します。そこで、重複して追加されることがないようにします。タイトルをドラッグするカーソルは、図 2 では正常にドロップできることを示すプラス記号になりますが、図 7 ではそうならないことに注目してください。

図 7. ドロップの禁止 (マウス・カーソルが、コピー可能であることを示すアイコンに変わりません)



重複するドロップを禁止するため、`<h5:drop>` コンポーネントの drag-enter および drag-over イベント・ハンドラーを変更して、ドロップを条件付きで受け入れるようにします (リスト 11 を参照)。

リスト 11. `<h5:drop>` コンポーネントの JavaScript (テイク III)

```
if (!html5)
   var html5 = {}
if (!html5.jsf) {
   html5.jsf = {
      init : function(ccid, payloadType, renderIds) {
         var dropzone = $(ccid);


         if (dropzone.serverPayload) // already initialized
            return;

         dropzone.payloadInput = $(ccid + ":form:payload");
         dropzone.acceptDrop = false;
         dropzone.serverPayload = function() {
            return dropzone.payloadInput.value;
         };

         dropzone.addEventListener("drop", function(event) {
            if (payloadType == "")
               payloadType = "text";

            if (renderIds == "" || renderIds == "@this")
               renderIds = ccid;

            dropzone.payloadInput.value = event.dataTransfer
                  .getData(payloadType);

            jsf.ajax.request(dropzone.payloadInput, event, {
               render: renderIds
               onevent : function(data) {
                   if (data.status == "success")
                      html5.jsf.init(ccid, payloadType, renderIds);
                }
            });
         }, false);

         dropzone.addEventListener("dragenter", function(event) {
if (dropzone.acceptDrop)
               event.preventDefault();
         }, false);

         dropzone.addEventListener("dragover", function(event) {
if (dropzone.acceptDrop)
               event.preventDefault();
         }, false);
      }
   };
   }
```

@Inject

リスト 12 の `@Inject` アノテーションに注目してください。これは、`rssFeed` MBean へのアクセスを提供するアノテーションです。JSF アプリケーションが Java コード内で MBean にアクセスする必要があることは珍しくありません。その場合、JSF および JSP Expression Language API を使用して MBean にアクセスするという方法もあれば、単純にこの `@Inject` アノテーションを使用するという方法もあります。

「Ajax の追加とサーバーへのペイロードの送信」セクションでは、ドラッグ・アンド・ドロップのペイロードをクライアントからサーバーに転送することに重点を置きましたが、ドロップを条件付きで受け入れるには、その反対のことを行わなければなりません。つまり、情報をサーバーからクライアントに送信するということです。この場合、リンクが重複しているかどうかを判別するには、保存済みリンクのリストにアクセスしなければなりません。そこで、必要なデータをサーバーからクライアントに転送するために、`serverPayload` 変数を `dropzone` に追加します。

保存済みリンクのリストをサーバーに送信するには、`DragDrop.getPayload()` を実装して、保存された各リンクのタイトルが含まれるストリングをドロップ・ターゲットの非表示テキスト入力フィールドに保管します (リスト 12 を参照)。

リスト 12. `payload` プロパティーのゲッター・メソッド

```
package com.clarity;

// imports omitted. See Listing 10.

@Named
@SessionScoped
public class DragDrop implements Serializable {
   @Inject private RSSFeed rssFeed;

   public DragDrop() {
   }

   public String getPayload() {
      // sends a string that is a concatenation of the saved
      // item's titles, back to the client
      LinkedList<RSSItem> savedItems = rssFeed.getSavedItems();
      Iterator<RSSItem> it = savedItems.iterator();
      String s = "";

      while (it.hasNext()) {
         RSSItem item = it.next();
         s += item.getTitle() + " | ";
      }
      return s;
   }

   public void setPayload(String payload) {
      // creates a new saved item, based on the payload. Payload
      // was set in the drop event listener for the h5:drop component
      // in /sections/feeds/menuLeft.xhtml
      StringTokenizer st = new StringTokenizer(payload);
      RSSItem item = new RSSItem();

      item.setTitle(st.nextToken("|"));
      st.nextToken(" ");
      item.setLink(st.nextToken(" "));

      rssFeed.getSavedItems().add(item);
   }
}
```

次にドラッグが開始されるときには、アプリケーションのドラッグ・ソースがサーバーからのペイロードをチェックして (これは当然、非表示テキスト入力フィールドの値です)、ドラッグされているリンクが重複しているかどうかを調べます (リスト 13 を参照)。

リスト 13. ドラッグ・ソース (テイク II)

```
<ui:composition xmlns="http://www.w3.org/1999/xhtml"
   xmlns:h="http://java.sun.com/jsf/html"
   xmlns:ui="http://java.sun.com/jsf/facelets"
   xmlns:h5="http://java.sun.com/jsf/composite/html5"
   xmlns:places="http://java.sun.com/jsf/composite/places">

   <script>
      // event.target is one of the h5:drag elements generated by ui:repeat below
       function dragStart(event) {
          var linkref = event.target.firstElementChild.firstElementChild; // anchor
          var link = linkref.href;
          var title = linkref.textContent;

          var dropzone = $("dropzone"); // h5:drop in dropZone.xhtml
          dropzone.acceptDrop = true;

          var serverPayload = dropzone.serverPayload();
          if (serverPayload.indexOf(title) != -1)
           dropzone.acceptDrop = false; // link already present

          event.dataTransfer.setData('text', title + " | " + link + " ");
      }
    </script>

   <h:panelGrid columns="1" id="items">
     <ui:repeat value="#{rssFeed.items}" var="item">

       <h5:drag ondragstart="dragStart(event)">
        <p>
          <a href="#{item.link}">#{item.title}</a> <br />
        </p>
      </h5:drag>

      </ui:repeat>
   </h:panelGrid>

</ui:composition>
```

ドラッグされているリンクが重複する場合、ドラッグ・ソースの JavaScript がドロップ・ターゲットの `acceptDrop` 属性を `false` に設定します。これにより、ドロップはドロップ・ターゲットによって取り消されることになります。

まとめ

今回の記事では、HTML5 のドラッグ・アンド・ドロップをカプセル化した JSF 2 複合コンポーネントの実装方法を説明しました。Java コードを作成したり、構成を行ったりしなくても、再利用可能なコンポーネントを実装できるようにする複合コンポーネントは、JSF 開発者のツール・ボックスに是非とも追加したい強力なツールです。記事ではまた、JSF の組み込み `<h:inputText>` 要素を例に、既存のコンポーネントを再利用して独自の複合コンポーネントを実装する際に必要となる作業を減らす方法も説明しました。

いくつかの複合コンポーネントを開発した後は、他の開発者も使用できるように、開発した複合コンポーネントを 1 つの JAR ファイルにまとめるのが賢明です。複合コンポーネントを JAR ファイルに圧縮して、WEB-INF の代わりに META-INF ディレクトリーに格納してください。

「JSF2 の魅力」の次回の記事では、この連載でこれまで取り上げてきた話題を振り返り、JSF 2 に推奨されるベスト・プラクティスについて説明します。

ダウンロード可能なリソース

内容	ファイル名	サイズ
Sample code for this article	j-jsf2fu-1110.zip	5.39MB

関連トピック

- JSF の Web サイト: JSF での開発に関する豊富な資料が揃っています。
- 「Exploring HTML5 with JavaServer Faces 2.0」: JSF 仕様のリーダーの一人、Roger Kitain によるスライドのプレゼンテーションを見てください。
- HTML5 仕様: HTML5 標準の公式な仕様が記載されています (2010年11月の時点ではまだ作成中です)。「Drag and drop」セクションを読んでください。
- 「ネイティブのドラッグ & ドロップ」(Eric Bidelman 著、HTML5Rocks、2010年9月): HTML5 のドラッグ・アンド・ドロップについて詳しく学んでください。
- HTML5 タグ・リファレンス: W3schools に HTML5 の要素についてのマニュアルが用意されています。
- HTML5 TUTORIAL: このサイトでは、ドラッグ・アンド・ドロップのチュートリアルを含め、多数の HTML5 チュートリアルを公開しています。
- JavaScript DOM: Javadoc 風の JavaScript DOM のマニュアルです。
- 「XHTML5 in a nutshell」(Sergey Mavrody 著、The WHATWG Blog、2010年7月): ポリグロット HTML5 について説明しています。
- developerWorks Java technology ゾーン: Java プログラミングのあらゆる側面を網羅した記事が豊富に用意されています。
- JSF: JSF 2.0 をダウンロードしてください。
- RSSLib4J: RSSLib4J をダウンロードしてください。