JSF 2 の魅力: HTML5 複合コンポーネント: 第 1 回

JSF 2 を使って HTML5 コンポーネント・ライブラリーの実装に取り掛かる

David Geary
President
Clarity Training, Inc.

2010年 10月 12日

HTML5 は、ブラウザー・ベースのアプリケーションに、デスクトップ・ソフトウェアの機能に匹敵するリッチな機能を提供します。連載「JSF 2 の魅力」では今回、JSF (JavaServer Faces) 2 を使ってHTML5 複合コンポーネントを実装し、Java™ と HTML5 両方の利点を生かす方法を紹介します。

このシリーズの他の記事を見る

ソフトウェア開発における次なる目玉とされている HTML5 は、ついにドラッグ・アンド・ドロップ、キャンバス、動画、音声などの魅力を完備して、当初 Web アプリケーションとして知られていた、ブラウザーで表示するアプリケーションに、デスクトップ・アプリケーションに匹敵する威力をもたらすようになりました。HTML5 とは技術 (具体的には仕様) の集合であり、これらがまとまって、HTML、JavaScript、および CSS (Cascading Stylesheets) を包含した強力な API を形作ります。HTML5 には、以下の注目すべき機能があります。

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記事の続編として、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、カンフー・マスター並みにこのフレームワークの達人にな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。

- キャンバス
- ドラッグ・アンド・ドロップ
- ジオロケーション*
- インライン編集
- Web ワーカー*
- Web ストレージ*
- メッセージング

© Copyright IBM Corporation 2010

- オフライン・アプリケーション
- 動画と音声*
- Web ソケット*

ジオロケーションやオフライン・アプリケーションなどの将来を見越した機能に注目してください (アスタリスクでマークを付けた機能は、厳密に言うと HTML5 仕様に含まれていませんが、HTML5 という用語は上記に挙げたすべての技術を網羅して言い表すために使われています。詳細については、「参考文献」を参照してください)。

ある意味、HTML5 は Java の最大の特徴を受け継ぐものと言えます。Java 言語は 1990年代後半に非常によく使われるようになりました。その理由は少なからず、Java では「Write once, run anywhere (一度書けば、どこでも実行できる)」ことから、開発者が Windows®、Mac、Linux® のいずれかを選択する (あるいはいずれかに移植する) 必要がなくなったところにあります。HTML5 の場合は、「Write once, run in any (modern) browser (一度書けば、どの (最近の) ブラウザーでも実行できる)」ので、iOS、Android、Chrome の中から 1 つだけ選ぶ必要はありません。

一度書けば、どこでもデバッグできるものなのか？

Java 技術を使用することで、複数のオペレーティング・システムに対応する 1 つのアプリケーションを作成することができますが、すべてが完璧に行われるというわけではありません。それは、HTML5 にしても同じです。HTML5 には、ネイティブ OS が提供する機能のうち、提供していない機能 (加速度センサーを利用する機能など) があります (ただし、PhoneGap (「参考文献」を参照) をはじめ、それをカバーするツールキットはあります)。このような不完全さを理由に、開発者が HTML5 を避けてネイティブ・アプリケーションを選ぶのはやむを得ませんが、多くのアプリケーションでは、HTML5 を使用することで投資収益率が改善されます。

Java 技術を採り入れる

HTML5 は Java の最大の特徴を受け継ぐものかもしれませんが、Java に置き換わることはありません。Java 技術はサーバー・サイド・プログラミングにリッチなエコシステムを提供します。しかも JSF は、そもそも HTML をベースとしたフレームワークです。したがって、今まで HTML4 で簡単に JSF を使用してきたように、HTML5 でも簡単に JSF を使用することができます。この 2 つを組み合わせることで、HTML5 の機能に加え、Facelets テンプレート、複合コンポーネント、組み込み Ajax など、JSF の強力な機能をすべて使用できるようになります。

この記事では、JSF2 を使用して HTML5 複合コンポーネントを作成する方法を説明します。そして連載「JSF 2 の魅力」の次回の記事では、HTML5 コンポーネントのライブラリーを作成する方法を紹介します。

HTML5 の導入

HTML5 を使用する場合、実際には HTML よりも JavaScript のほうが大きく関わってきます。つまり、HTML5 を使用するには優れた JavaScript デバッガーが必要であるということです。私がお勧めするデバッガーは、Google Chrome に組み込まれている Developer Tools (「参考文献」を参照) に付属のデバッガーです (図 1 を参照)

図 1. Chrome Developer Tools による JavaScript のデバッグ

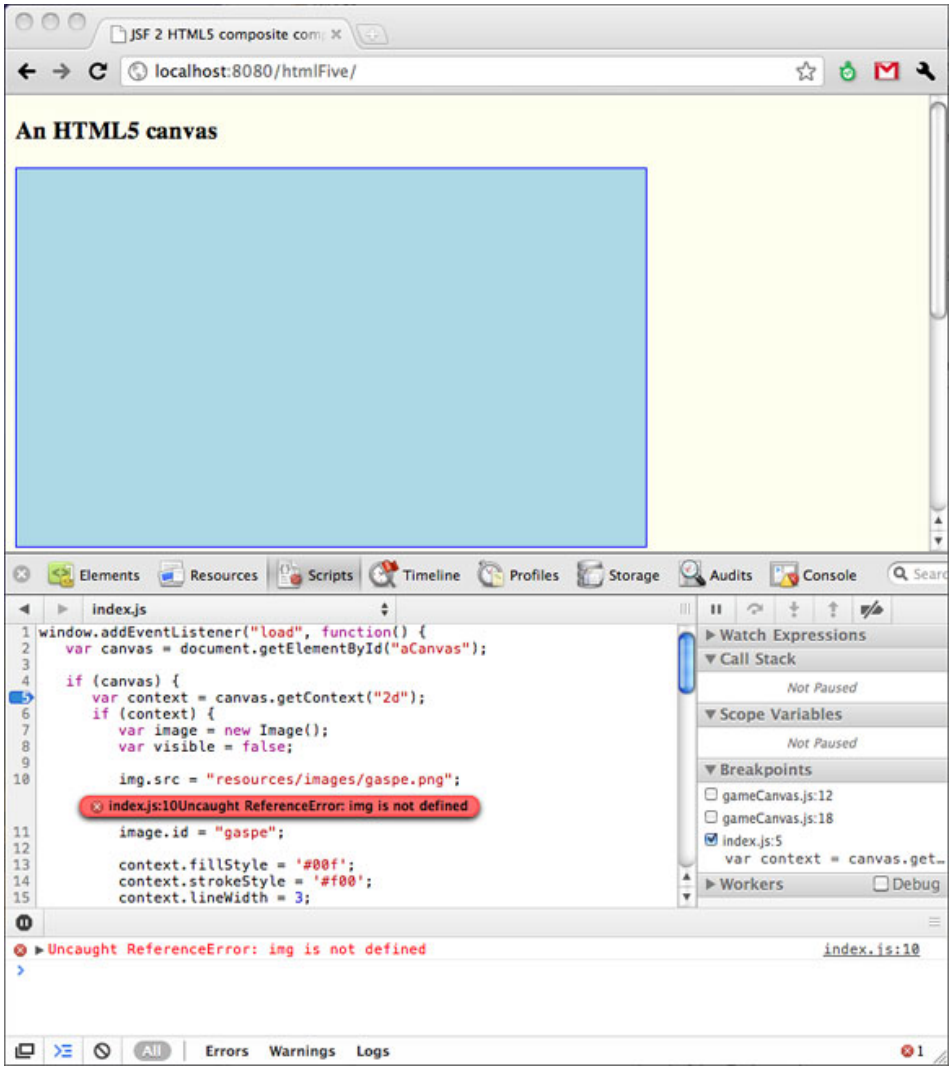


図 1 に示されているように、Chrome デバッガーの上部にはキャンバス・コンポーネントが表示され、その下のパネルに JavaScript コードが表示されます。

優れた JavaScript デバッガーの用意ができたら、あとは HTML5 対応のブラウザーがあればよいのです。よく使われているブラウザーの最新バージョンであれば、ほとんどの場合、HTML5 を十分にサポートしています (Microsoft では近日リリース予定の Internet Explorer 9 に、優れた HTML5 サポートを組み込むようです)。

HTML5 キャンバスの使用方法

HTML5 キャンバスは、本格的な 2D 描画サーフェイスです。Plants vs. Zombies や Quake II などのゲームをサポートするにも十分なほど充実した機能が用意されています。私がこの HTML5 キャンバスを使用して描画した例 (図 2 を参照) は、ゲームほど魅力的ではないかもしれませんが、このキャンバスの使い方を説明するには十分です

図 2. 単純な HTML5 キャンバスの例

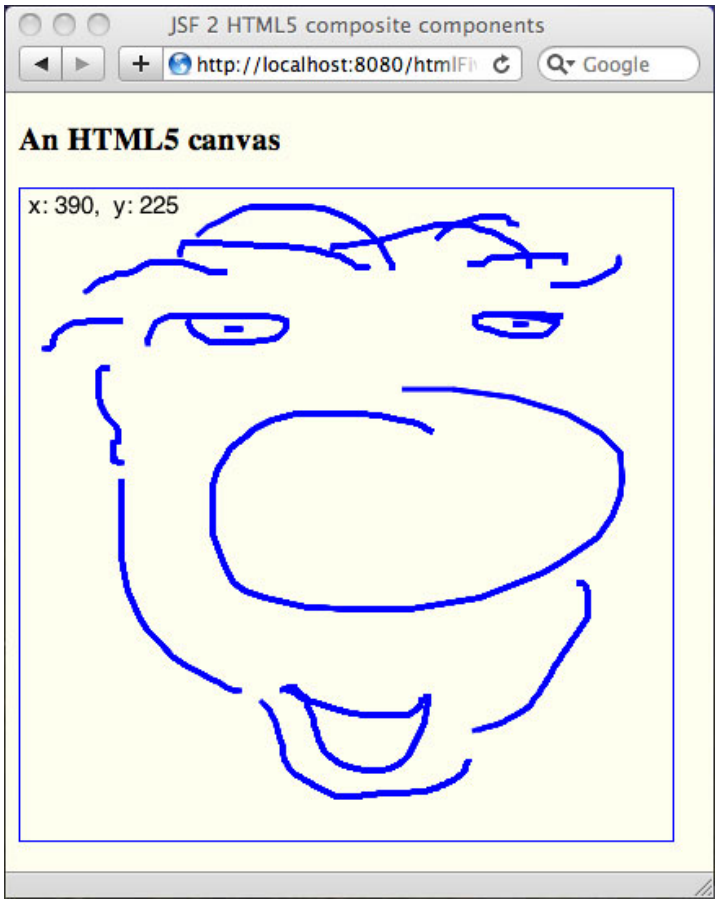
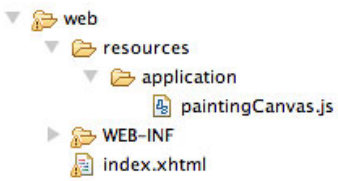


図 2 の単純なペイント・アプリケーションは、HTML5 キャンバスに JavaScript を追加して実装されています。マウスを動かすと、キャンバス左上隅にマウスの座標の読み取り値が表示され、キャンバス内でマウスをドラッグすると、青いブラシで線がペイントされるという仕組みです。

図 2 に示したアプリケーションは、JSF アプリケーションです。図 3 に、このアプリケーションのディレクトリー構造を示します。

図 3. サンプル・キャンバスのディレクトリー構造



このアプリケーション唯一の Facelet は web/WEB-INF/index.xhtml に定義されていて、アプリケーションの JavaScript は web/resources/application/paintingCanvas.js に定義されています。リスト 1 にまず、前者の index.xhtml を記載します。

リスト 1. `<canvas>` タグの使用 (WEB-INF/index.xhtml)

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<!DOCTYPE html>

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

   <h:head>
      <title>#{msgs.windowTitle}</title>
   </h:head>

   <h:body style="background: #fefeef">
      <h:outputScript library="application" name="paintingCanvas.js"
                      target="head" />

      <h3>#{msgs.heading}</h3>


      <canvas width="400px" height="400px" id="paintingCanvas">

         Canvas not supported.

      </canvas>
   </h:body>

</html>
```

リスト 1 の index.xhtml ファイルは、HTML5 ポリグロット (polyglot) 文書として知られています (「参考文献」を参照)。このファイルには、HTML5 の文書タイプ/名前空間と整形式 XHTML 構文が含まれるからです。これがまさに、この例で Facelets と HTML5 を組み合わせて使用するために必要となるものです。

対応する JavaScript を `<h:outputScript>` タグに設定してインポートした後は、いよいよ HTML5 キャンバス要素の登場です。ブラウザーが `<canvas>` タグを認識しなければ、「Canvas not supported (キャンバスはサポートされていません)」というメッセージが表示されます。`<canvas>` タグ自体は控えめなもので、その興味深いコードはすべて、対応する JavaScript の中にあります (リスト 2 を参照)。

リスト 2. キャンバスのペイントを行う JavaScript (resources/application/paintingCanvas.js)

```
window.addEventListener("load", function() {
   var canvas, context, painting;

   function init() {
      canvas = document.getElementById("paintingCanvas");
      if (canvas == null) return;

      context = canvas.getContext("2d");
      if (context == null) return;

      painting = false;

      context.strokeStyle = "#00f";
      context.lineWidth = 3;
      context.font = "15px Helvetica";
   }

   init();

   canvas.addEventListener("mousedown", function(ev) {
      painting = true;
      context.beginPath();
```

```
      context.moveTo(ev.offsetX, ev.offsetY);
  }, false);

  canvas.addEventListener("mousemove", function(ev) {
     updateReadout(ev.offsetX, ev.offsetY);

     if (painting) {
       paint(ev.offsetX, ev.offsetY);
     }
     function updateReadout(x, y) {
        context.clearRect(0, 0, 100, 20);
        context.fillText("x: " + x + ",  y: " + y, 5, 15);
     }
     function paint(x, y) {
        context.lineTo(ev.offsetX, ev.offsetY);
        context.stroke();
     }

  }, false);

  canvas.addEventListener("mouseup", function() {
     painting = false;
     context.closePath();
  }, false);


}, false);
```

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。

リスト 2 で実装しているのは、図 2 に示したマウス・カーソルの位置を読み取る機能が付いた単純なペイント機能です。ページのロード時に、`document.getElementById()` によってキャンバスを参照し、キャンバスから、そのキャンバスのコンテキストを参照します。このコンテキストは、以降のイベント・ハンドラーで使用します。これらのハンドラーを実装するために使用するのは、JavaScript クロージャー、あるいは、Java 開発者が匿名内部クラスと呼ぶものです。

AWT(Abstract Window Toolkit) を使った経験のある方は、キャンバスのコンテキストから即座に AWT のグラフィックス・コンテキストを連想することでしょう。結局のところ、形状、画像、そしてテキストを 2 次元で描画する方法はいくつかに限られているのです。リスト 2 では、ストローク (描画) スタイルを青色に設定してコンテキストを初期化し、線幅とフォントを設定します。その後は、マウス・ボタンの押下、マウスのドラッグ、マウス・ボタンのリリースのそれぞれに対応する、ストローク開始位置への移動、ストローク、ストロークの終了と次のストロークへの準備、といった操作が続いているだけです。

HTML5 キャンバスの基本を押さえたところで、次は JSF 2 と HTML5 の複合コンポーネントを作成する方法を説明します。

HTML5 キャンバスを使用した JSF2 複合コンポーネント

ここからは、HTML5 キャンバスを使用した JSF 2 複合コンポーネントを実装します。この複合コンポーネントは以下の要件を満たさなければなりません。

- 幅、高さ、ペンの色、線幅、CSS スタイルを (タグの属性によって) 構成可能にすること
- コンポーネント・タグの本体を「Canvas not supported (キャンバスはサポートされていません)」というメッセージとして使用すること
- キャンバスの JavaScript を自動的に組み込むこと
- 1 つのページで複数のキャンバス・コンポーネントをサポートすること

図 4 に、キャンバス複合コンポーネントを使用したアプリケーションを示します。

図 4. 動作中のキャンバス複合コンポーネント

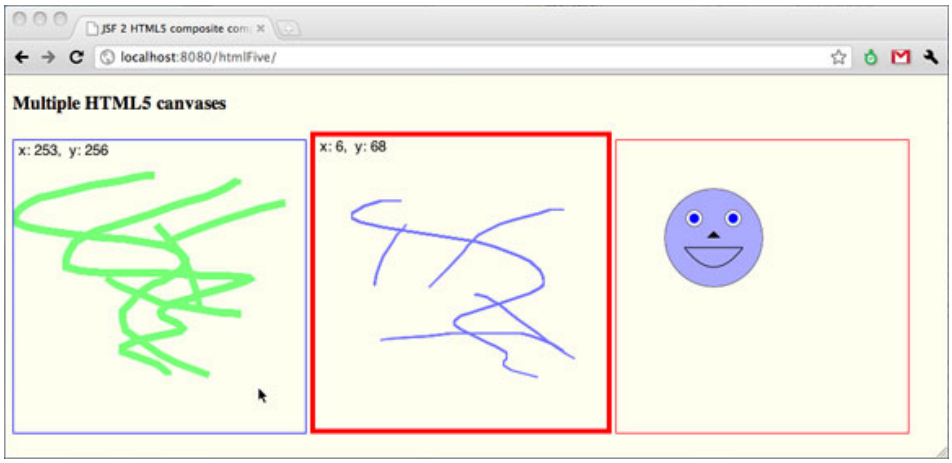


図 4 に示すアプリケーションには、それぞれに構成が異なる 3 つのキャンバス・コンポーネントがあります。左側にある 2 つは、図 2 に示したようなペイント用キャンバスです。右端にあるキャンバスでは、単にスマイリー・フェイスをペイントするだけです。

リスト 3 に、図 4 に示したページのマークアップを記載します。

リスト 3. キャンバス複合コンポーネントを使用するコード (WEB-INF/index.xhtml)

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html>

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

   <h:head>
      <meta charset="UTF-8" />
      <title>#{msgs.windowTitle}</title>
   </h:head>

   <h:body style="background: #fefeef">

      <h3>#{msgs.heading}</h3>

      <h5:canvas id="paintingCanvas" width="300px" height="300px"
                 penColor="#7F7" lineWidth="7">

         #{msgs.canvasUnsupported}

      </h5:canvas>

      <h5:canvas id="secondPaintingCanvas" width="300px" height="300px"
                 style="border: thick solid red">
```

```
         #{msgs.canvasUnsupported}

      </h5:canvas>

      <h5:canvas id="smileyCanvas" library="application" script="smiley.js"
                 width="300px" height="300px"
                 style="border: thin solid red">

         #{msgs.canvasUnsupported}

      </h5:canvas>
   </h:body>

</html>
```

リスト 1 では HTML5 を手作業で使用したので、関連する JavaScript を明示的にインポートしなければなりませんでしたが、リスト 3 はそれとは異なり、キャンバス・コンポーネントが適切な JavaScript をインポートします。ページ作成者は、キャンバス・コンポーネントのオプションの属性 `library` および `script` を使ってキャンバスの JavaScript を指定することも、デフォルトの JavaScript を使用することもできます。リスト 3 では、スマイリー・キャンバスには `script` 属性を使用している一方、上記のサンプルに示されている左の 2 つのキャンバスにはデフォルトの JavaScript (ペイント用キャンバスを実装する resources/html5/canvasDefault.js) を使用しています。

このキャンバス複合コンポーネントの実装をリスト 4 に記載します。

リスト 4. キャンバス複合コンポーネント (WEB-INF/index.xhtml)

```
<?xml version="1.0" encoding="UTF-8"?>
<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="id"/>
    <composite:attribute name="width"/>
    <composite:attribute name="height"/>
    <composite:attribute name="library"  default="html5"/>
    <composite:attribute name="script"   default="canvasDefault.js"/>
    <composite:attribute name="style"    default="border: thin solid blue"/>
    <composite:attribute name="penColor" default="#7777FF"/>
    <composite:attribute name="lineWidth" default="2"/>
  </composite:interface>

  <composite:implementation>
      <canvas id="#{cc.id}"
           width="#{cc.attrs.width}"
          height="#{cc.attrs.height}"
           style="#{cc.attrs.style}">

      <composite:insertChildren/>

     </canvas>

<h:outputScript library="#{cc.attrs.library}"
                       name="#{cc.attrs.script}"/>

    <script>
#{cc.attrs.script}.init('#{cc.id}',
                              '#{cc.attrs.penColor}',
                              '#{cc.attrs.lineWidth}')
    </script>
```

```
  </composite:implementation>
</html>
```

リスト 4 でキャンバス複合コンポーネントに宣言している属性は 8 つあり、そのうちの 5 つにはデフォルトの値が設定されています。コンポーネントの実装に含まれる HTML5 キャンバスの ID、幅、高さ、およびスタイルは、コンポーネントに関連付けられた属性から構成されます。つまり、キャンバス・コンポーネントの最初の要件 (構成可能な属性) は、これで対処されたことになります。

キャンバス複合コンポーネントの子 (`<h5:canvas>` タグの本体に含まれるあらゆる要素) は、HTML5 キャンバスのタグ (`<canvas>`) に挿入されます。これは、ブラウザーが HTML5 キャンバスをサポートしない場合には、タグ本体に含まれるテキストが表示されることを意味するため、2 番目の要件 (「Canvas not supported (キャンバスはサポートされていません)」というメッセージには、コンポーネント・タグの本体を使用すること) を満たすということです。

キャンバス・コンポーネントに含まれる `<h:outputScript>` タグは、キャンバス・コンポーネントの `library` および `script` 属性で指定されたキャンバスの JavaScript をインポートします。`library` 属性と `script` 属性は、デフォルトでそれぞれ `html5`、`canvasDefault.js` に設定されることに注目してください。3 番目の要件 (キャンバスの JavaScript を自動的にインポートすること) には、これで対処することができます。

最後に、キャンバス・コンポーネントは `init()` という JavaScript のメソッドを呼び出し、キャンバスの ID、ペンの色、線幅を渡します。`init()` メソッドはキャンバスのコンテキストを取得して初期化します。ここで、「関数」と呼ばずに「メソッド」という言葉を使用したのは、`init()` はオブジェクトに属するからです。そのオブジェクトの名前は、キャンバス・コンポーネントの `script` 属性から派生します。例えばリスト 3 のスマイリー・キャンバスでは、`script` 属性に `smiley.js` という値を指定しているため、スマイリー・キャンバス・コンポーネントは `smiley.js.init()` (具体的には、`smiley` という名前のオブジェクトに含まれる、`js` という名前のオブジェクトの `init()` メソッド) を呼び出すことになります。明示的に `script` 値を指定しなければ、デフォルトで値は `canvasDefault.js` に設定されるので、この場合のJavaScript メソッドは `canvasDefault.js.init()` となります。グローバル関数ではなく、これらのメソッドを呼び出すことによって 4 番目の要件を満たすこと (つまり 1 つのページで複数のキャンバスをサポートすること) ができます。

リスト 5 に、キャンバス・コンポーネントのデフォルト JavaScript を記載します。

リスト 5. キャンバスのデフォルト JavaScript (resources/html5/canvasDefault.js)

```
if (!canvasDefault) var canvasDefault = {}

if (!canvasDefault.js) {
canvasDefault.js = {
init : function(canvasId, penColor, lineWidth) {
      var canvas, context, painting;

      canvas = document.getElementById(canvasId);
      if (canvas == null) {
        alert("Canvas " + canvasId + " not found")
      }

context = canvas.getContext("2d")
```

```
      if (context == null)
        return;

      painting = false;

      context.strokeStyle = penColor
      context.lineWidth = lineWidth
      context.font = "15px Helvetica"

      canvas.addEventListener("mousedown", function(ev) {
        painting = true
        context.beginPath()
        context.moveTo(ev.offsetX, ev.offsetY)
      }, false)

      canvas.addEventListener("mousemove", function(ev) {
        updateReadout(ev.offsetX, ev.offsetY)

        if (painting) {
          paint(ev.offsetX, ev.offsetY)
        }
        function updateReadout(x, y) {
          context.clearRect(0, 0, 100, 20)
          context.fillText("x: " + x + ",  y: " + y, 5, 15)
        }
        function paint(x, y) {
          context.lineTo(ev.offsetX, ev.offsetY)
          context.stroke()
        }

      }, false)

      canvas.addEventListener("mouseup", function() {
        painting = false
        context.closePath()
      }, false)
    }
  }
}
```

リスト 5 で作成している `canvasDefault` という名前のオブジェクトには `js` というオブジェクトが含まれ、さらにこのオブジェクトには `init()` メソッドが含まれます。このようにして `init()` メソッドの名前空間を設定しているため、このメソッドは他のグローバル `init()` 関数によってオーバーライドされることがありません。したがって、それぞれ独自に `init()` 関数を実装する複数のキャンバスを 1 つのページで使用することができるといわけです。

リスト 6 に、スマイリー・キャンバスの JavaScript を記載します。

リスト 6. スマイリー・キャンバスの JavaScript (resources/application/smiley.js)

```
if (!smiley) var smiley = {}

if (!smiley.js) {
smiley.js = {
init : function(canvasId, penColor, lineWidth) {
      var canvas, context

      canvas = document.getElementById(canvasId);
      if (canvas == null) {
        alert("Canvas " + canvasId + " not found")
      }

context = canvas.getContext("2d");
```

```
      if (context == null)
        return

      // smiley face code originally downloaded
      // from thinkvitamin.com

      // Create the face

      context.strokeStyle = "#000000";
      context.fillStyle = "#AAAAFF";
      context.beginPath();
      context.arc(100,100,50,0,Math.PI*2,true);
      context.closePath();
      context.stroke();
      context.fill();

      // eyes
      context.strokeStyle = "#000000";
      context.fillStyle = "#FFFFFF";
      context.beginPath();
      context.arc(80,80,8,0,Math.PI*2,true);
      context.closePath();
      context.stroke();
      context.fill();

      context.fillStyle = "#0000FF";
      context.beginPath();
      context.arc(80,80,5,0,Math.PI*2,true);
      context.closePath();
      context.fill();

      context.strokeStyle = "#000000";
      context.fillStyle = "#FFFFFF";
      context.beginPath();
      context.arc(120,80,8,0,Math.PI*2,true);
      context.closePath();
      context.stroke();
      context.fill();

      context.fillStyle = "#0000FF";
      context.beginPath();
      context.arc(120,80,5,0,Math.PI*2,true);
      context.closePath();
      context.fill();

      // nose
      context.fillStyle = "#000000";
      context.beginPath();
      context.moveTo(93,100);
      context.lineTo(100,93);
      context.lineTo(107,100);
      context.closePath();
      context.fill();

      // smile
      context.strokeStyle = "#000000";
      context.beginPath();
      context.moveTo(70,110);

      context.quadraticCurveTo(100,150,130,110);
      context.closePath();
      context.stroke();
    }
  }
}
```

リスト 6 の名前空間の設定方法は、リスト 5 で使用した規則に従っています。他のキャンバスの JavaScript についても、これと同じ規則に従う必要があります。

まとめ

この記事では HTML5 について紹介し、JSF 2 と HTML5 キャンバスの複合コンポーネントを実装する方法を実際に示しました。この方法を使えば、JSF 開発者とページ作成者は HTML5 キャンバスを簡単に使えるようになります。また、JSF 式言語から取得した情報を、複合コンポーネントに関連付けられた JavaScript に渡す方法、そして同じ名前が付けられた関数が互いに上書きしないように JavaScript 関数に名前空間を設定する方法についても説明しました。連載「JSF 2 の魅力」の次回の記事では、別の HTML5 コンポーネントを実装する方法を紹介し、複数の HTML5 コンポーネントを再利用可能なライブラリーに組み込んで、JAR ファイルで他の開発者にも配布できるようにします。

ダウンロード

内容	ファイル名	サイズ
Sample code for this article	j-jsf2fu-1010.zip	49KB

著者について

David Geary

著者、講演者、コンサルタントとして活躍する David Geary は、Clarity Training, Inc. の社長です。彼はこの会社で、開発者に JSF および GWT (Google Web Toolkit) を使用した Web アプリケーションの実装を指導しています。JSTL 1.0 および JSF 1.0/2.0 Expert Group のメンバー、そして Sun の Web 開発者認定試験の共同制作者としての経験を持つ彼は、Apache Struts や Apache Shale などのオープンソース・プロジェクトにも貢献しています。彼の著書『グラフィック Java2 〈Vol.2〉 Swing 編』は Java 関連の本のなかでは史上に残るベスト・セラーの 1 つで、『Core JSF』(Cay Horstman との共著) は JSF 関連のベストセラー本となっています。コンファレンスやユーザー・グループで定期的に講演を行っている他、2003年以来 NFJS ツアーの常連で、これまでに 3 回 Java University の講師になり、JavaOne ロック・スターに 2 回選ばれました。

© Copyright IBM Corporation 2010
(www.ibm.com/legal/copytrade.shtml)
商標
(www.ibm.com/developerworks/jp/ibm/trademarks/)