GWT の魅力: 第 2 回 上級編

高度な Google Web Toolkit の機能を実装する

David Geary
President
Clarity Training, Inc.

2009年 10月 20日

GWT (Google Web Toolkit) では、ブラウザーで動作するデスクトップ風のアプリケーションを実装することができます。この 2 回の連載の後半では、David Geary がイベントの受信、タイマーの使用、そしてイベント・プレビューなど、さらに高度な GWT の機能を使用する方法を説明します。

このシリーズの他の記事を見る

Swing のような API を備えた GWT では、Java™ Web Start などのソフトウェアを追加することなく、ブラウザーで動作するリッチ・クライアントのユーザー・インターフェースを実装することができます。この 2 回の連載の第 1 回では、ウィジェットの使い方、リモート・プロシージャー呼び出し (RPC) を使用する方法、そして複合ウィジェットの実装方法など、GWT の基本をおさえました。

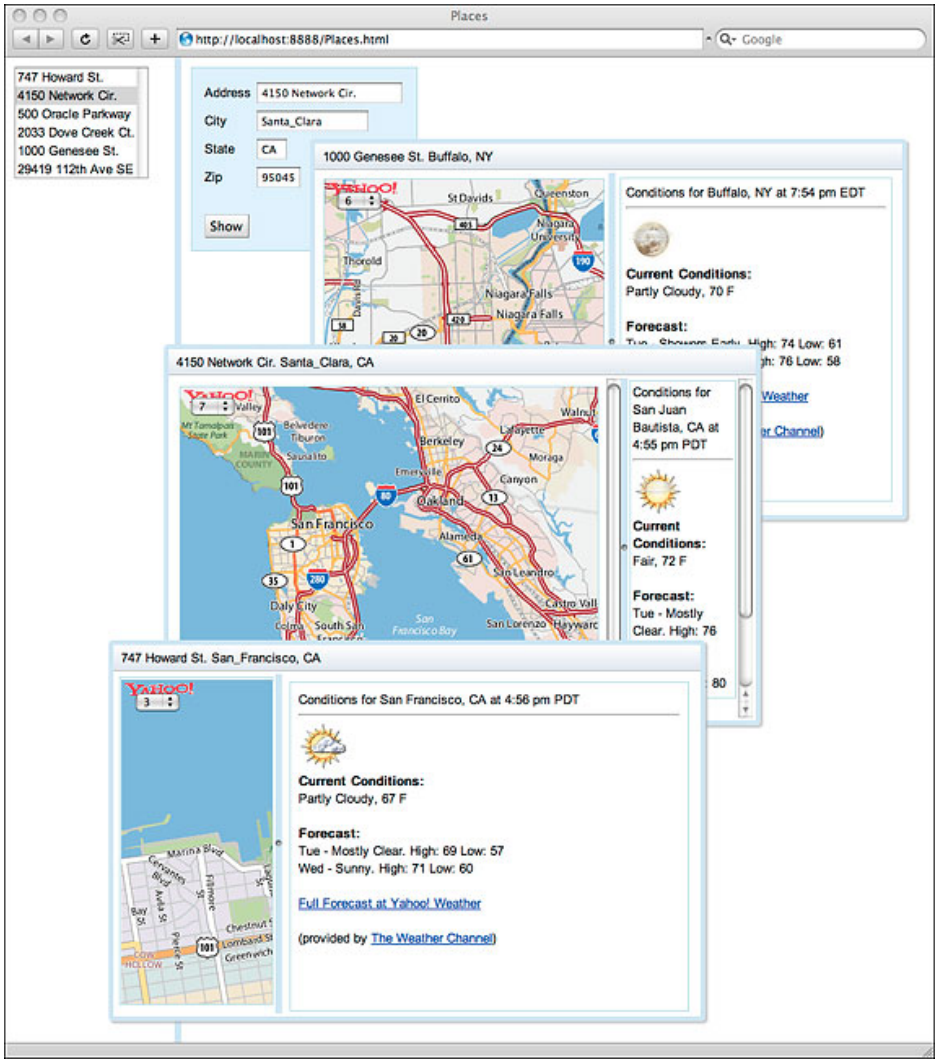
今回の記事では、第 1 回では後回しとしていた高度な GWT の機能について説明します。この記事で取り上げる話題は以下のとおりです。

- ダイアログ・ボックス
- イベントの受信と DOM (Document Object Model) 要素の属性の操作
- 画像のロード (およびビジー・カーソル)
- モジュール
- イベント・プレビュー
- タイマー

Places アプリケーション (復習)

第 1 回では、図 1 に示す Places アプリケーションを導入しました。

図 1. Places アプリケーション



このアプリケーションは起動時にデータベースから 6 つの住所を取得し、左上にあるリスト・ボックスにこれらの住所を表示します。リスト・ボックス内に表示された住所のいずれかをクリックすると、アプリケーションはリスト・ボックス右側のグリッドに、選択された住所をロードします。住所がロードされたグリッドの Show ボタンをクリックすると、アプリケーションはその住所の地図と、その地域の天気の情報を Yahoo! の Web サービスからロードして、ダイアログ・ボックスに表示します。ダイアログ・ボックスでは、地図と天気の情報が左右に分割されたパネルに表示されます。この分割位置は調整可能です (図 1 を参照)

第 1 回では、データベースから住所をロードしてリスト・ボックスに表示し、リスト・ボックスで選択された項目をグリッドに取り込むまでの作業を行いました。この一連の作業が終わった時点での Places アプリケーションのコードは、第 1 回のリスト 9 に記載されています。今回焦点とするのは、ダイアログ・ボックス (`PlaceDialog` のインスタンス) と、地図が表示されるカスタム・ビューポート・ウィジェットです。アプリケーションはサーバーに対する 2 つの RPC によって Yahoo! の Web サービスから地図および天気の情報を取得した後、以下のコードに示すように `PlaceDialog` のインスタンスを作成し、その表示位置を設定したら、このダイアログ・ボックスを表示します。

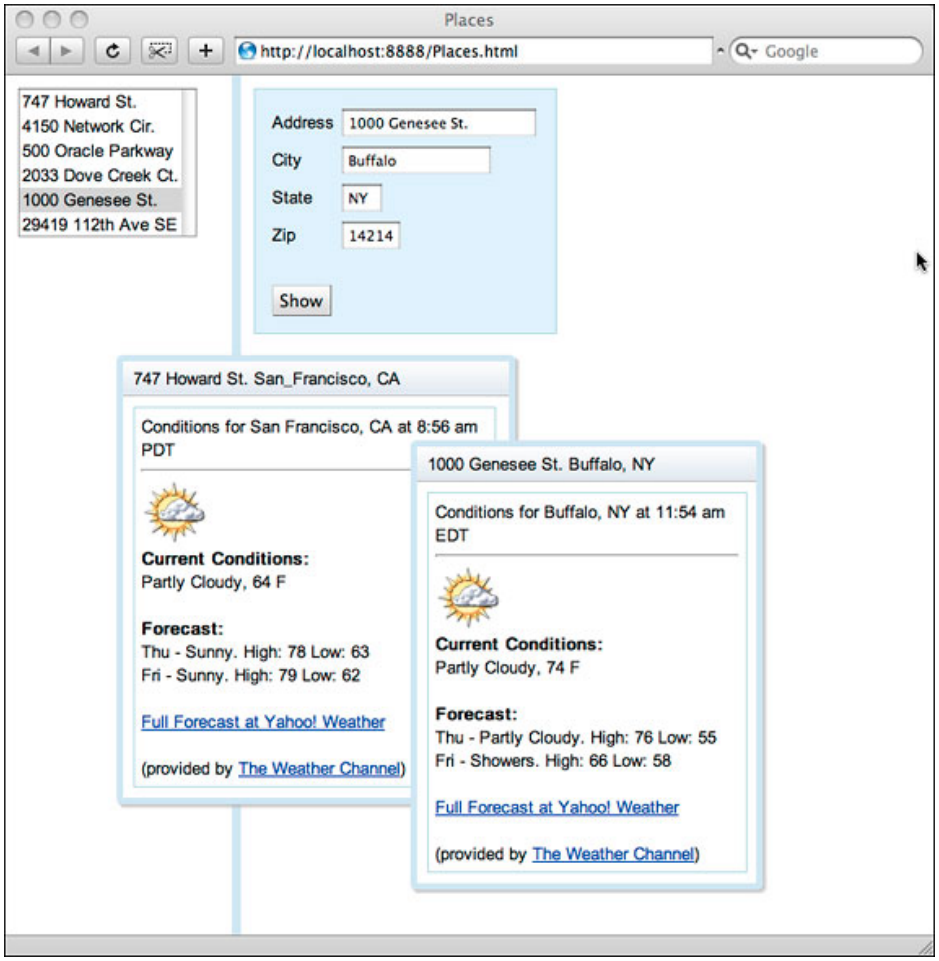
```
PlaceDialog dialog = new PlaceDialog(addressGrid.getAddress(), urls, weatherHTML);
dialog.setPopupPosition(200, 200);
dialog.show();
```

ダイアログ・ボックスとその内部のビューポートを実装する手順をとおして、GWT の高度な機能のうちのいくつかを詳しく探っていきます。

ダイアログ・ボックス

記事の残りでは、引き続き Places アプリケーションの実装作業を行います (完全なアプリケーションのソース・コードはダウンロードすることができます)。最初に行う作業は、選択された場所の天気の情報を表示する単純な `PlacesDialog` を実装することです (図 2 を参照)。

図 2. Places ダイアログの最初の状態



リスト 1 に、この最初の `PlacesDialog` に対応するコードを記載します。

リスト 1. PlacesDialog.java、テイク 1

```
package com.clarity.client;
package com.clarity.client;

import com.google.gwt.user.client.ui.DialogBox;
import com.google.gwt.user.client.ui.HTML;

public class PlacesDialog extends DialogBox {

  public PlacesDialog(Address address, String[] imageUrls, String weather) {
    super(false, false); // no auto-hide, and no modal

    setText(address.getAddress() + " " + address.getCity() + ", " + address.getState());

    HTML weatherHTML = new HTML();
    weatherHTML.setHTML(weather);

    setWidget(weatherHTML);
  }
}
```

リスト 1 の内容は至って単純です。`PlacesDialog` は `DialogBox` を継承し、Web サービスから取得した天気の情報を表示する HTML ウィジェットを設定します。`PlacesDialog` コンストラクターからはスーパークラス・コンストラクターを呼び出して、ダイアログを作成しています。このダイアログは、モーダル・ダイアログではなく、またユーザーがダイアログ・ボックスの外側をクリックしても自動的に非表示にされることがないダイアログです。

しかし、リスト 1 のダイアログ・ボックスには 1 つの問題があります。それは、ダイアログをクリックしても最前面に表示できないことです。例えば図 2 では、San Francisco のダイアログを Buffalo のダイアログの前面に表示することはできません。そのための機能を実装するには、ダイアログ・ボックスでのマウス・イベントを受信し、マウスダウン・イベントを処理する必要があります。

イベントの受信と DOM 要素の属性の操作

リスト 2 に更新後の `PlacesDialog` クラスを記載します。ここでは、ダイアログ・ボックスでのマウス・イベントを受信するために `sinkEvents()` を呼び出しています。`sinkEvents()` を呼び出した後、ダイアログ・ボックスでマウス・イベントが発生すると、GWT はダイアログ・ボックスの `onBrowserEvent()` メソッドを呼び出します。このメソッドのなかで `DOM.setIntStyleAttribute()` を呼び出すことによって、ダイアログ・ボックスに関連付けられた DOM 要素の `z` インデックスを増加させます。

リスト 2. PlacesDialog.java、テイク 2

```
package com.clarity.client;

import com.google.gwt.user.client.DOM;
import com.google.gwt.user.client.Event;
import com.google.gwt.user.client.ui.DialogBox;
import com.google.gwt.user.client.ui.HTML;

public class PlacesDialog extends DialogBox {
  private static int z;

  public PlacesDialog(Address address, String[] imageUrls, String weather) {
    setText(address.getAddress() + " " + address.getCity() + ", " + address.getState());
```

```
    HTML weatherHTML = new HTML();
    weatherHTML.setHTML(weather);

    setWidget(weatherHTML);

    sinkEvents(Event.MOUSEEVENTS);
  }

  public void onBrowserEvent(Event event) {
   if (event.getTypeInt() == Event.ONMOUSEDOWN) {
      DOM.setIntStyleAttribute(PlacesDialog.this.getElement(), "zIndex", z++);
    }
    super.onBrowserEvent(event);
  }
}
```

リスト 2 のダイアログ・ボックスであれば、クリックすることによってダイアログを最前面に表示することができます。

画像のロードとビジー・カーソル

リスト 3 では、左右に分割されたパネルを追加し、地図を左側に、天気の情報を右側に配置することによって `PlacesDialog` クラスの実装を仕上げています。

リスト 3. PlacesDialog.java、テイク 3

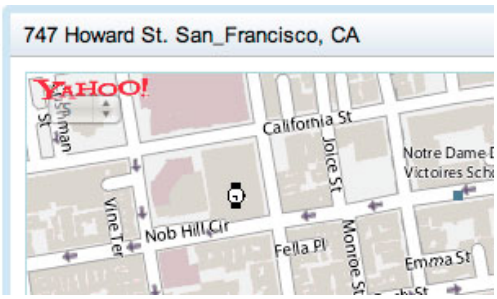
```
package com.clarity.client;

import com.google.gwt.event.dom.client.ChangeEvent;
import com.google.gwt.event.dom.client.ChangeHandler;
import com.google.gwt.event.dom.client.LoadEvent;
import com.google.gwt.event.dom.client.LoadHandler;
import com.google.gwt.user.client.DOM;
import com.google.gwt.user.client.Event;
import com.google.gwt.user.client.ui.AbsolutePanel;
import com.google.gwt.user.client.ui.DialogBox;
import com.google.gwt.user.client.ui.HTML;
import com.google.gwt.user.client.ui.HorizontalSplitPanel;
import com.google.gwt.user.client.ui.Image;
import com.google.gwt.user.client.ui.ListBox;
import com.google.gwt.user.client.ui.Panel;

import com.clarity.widgets.client.Viewport;

public class PlacesDialog extends DialogBox {
    private static int z;
    private static final String[] zoomLevelItems = {
      "1", "2", "3", "4", "5", "6", "7", "8", "9", "10", "11", "12"
    };

    private final Viewport viewport = new Viewport();
    private final Image[] images = new Image[12];
    private String[] imageUrls;

    final ListBox zoomLevels = new ListBox();

    public PlacesDialog(Address address, String[] imageUrls, String weather) {
      super(false, false); // no auto-hide, and no modal
      setText(address.getAddress() + " "
          + address.getCity() + ", "
          + address.getState());

      this.imageUrls = imageUrls;
```

```
      HorizontalSplitPanel hsp = new HorizontalSplitPanel();
      hsp.setPixelSize(600, 350);
      hsp.add(createMapPanel());
      hsp.add(createWeatherPanel(weather));

      zoomLevels.addChangeHandler(new ChangeHandler() {
        public void onChange(ChangeEvent event) {
          String v = zoomLevels.getItemText(zoomLevels.getSelectedIndex());
          setZoom(new Integer(v).intValue() - 1);
        }
      });

      setWidget(hsp);
      sinkEvents(Event.MOUSEEVENTS);
    }

    public void onBrowserEvent(Event event) {
      if (event.getTypeInt() == Event.ONMOUSEDOWN) {
        DOM.setIntStyleAttribute(PlacesDialog.this.getElement(), "zIndex", z++);
      }
      super.onBrowserEvent(event);
    }

    private Panel createMapPanel() {
      AbsolutePanel viewportPanel = new AbsolutePanel();

      for (String item : zoomLevelItems) {
        zoomLevels.addItem(item);
      }

      images[0] = new Image();
      images[0].setUrl(imageUrls[0]);

      viewport.setView(images[0]);

      viewportPanel.add(viewport);
     viewportPanel.add(zoomLevels, 10, 10);  // 10, 10 are x,y coordinates from ulhc

      DOM.setIntStyleAttribute(zoomLevels.getElement(), "zIndex",
      DOM.getIntStyleAttribute(viewport.getElement(), "zIndex") + 1);

      return viewportPanel;
    }

    private HTML createWeatherPanel(String weather) {
      HTML weatherHTML = new HTML();
      weatherHTML.setHTML(weather);
      return weatherHTML;
    }

    public void setZoom(final int index) {
      if (images[index] == null) {
        images[index] = new Image();
        images[index].addLoadHandler(new LoadHandler() {
          public void onLoad(LoadEvent event) {
            zoomLevels.setEnabled(true);
            viewport.removeStyleName("waitCursor");
          }
        });
      zoomLevels.setEnabled(false);
      viewport.addStyleName("waitCursor");
    }
    images[index].setUrl(imageUrls[index]);
    viewport.setView(images[index]);
  }
}
```

リスト 3 の `createMapPanel()` メソッドでは、ビューポートを作成しています。import 文を見るとわかるように、このビューポートは `com.clarity.widgets.client.Viewport` のインスタンスです。このビューポートとともに、ズーム・レベルのプルダウンを `AbsolutePanel` のインスタンスに追加します。絶対パネルでは、その内部に包含するウィジェットを、ピクセル値で指定した位置に配置することができます。上記では、ズーム・レベル・プルダウンをパネル上端から 10 ピクセル下、パネル左上隅から 10 ピクセル左の位置に配置しています。

図 3 に示されているように、ズーム・レベルを選択すると、アプリケーションは選択されたズーム・レベルに対応する画像をロードする間、カーソルを待機状態のカーソルに変更します。そして画像が取り込まれた時点で、カーソルは元の状態に戻ります。

図 3. 画像のロード



リスト 4 に記載するのは、リスト 3 からの抜粋です、ここに、画像のロードをモニターしてロード状況に応じてカーソルを操作する方法が示されています。

リスト 4. 画像のロード

```
package com.clarity.client;
public class PlacesDialog extends DialogBox {
    private static int z;
    private static final String[] zoomLevelItems = {
      "1", "2", "3", "4", "5", "6", "7", "8", "9", "10", "11", "12"
    };

    private final Viewport viewport = new Viewport();
    private final Image[] images = new Image[12];
    private String[] imageUrls;

    final ListBox zoomLevels = new ListBox();

    public PlacesDialog(Address address, String[] imageUrls, String weather) {
      ...
      zoomLevels.addChangeHandler(new ChangeHandler() {
        public void onChange(ChangeEvent event) {
          String v = zoomLevels.getItemText(zoomLevels.getSelectedIndex());
          setZoom(new Integer(v).intValue() - 1);
        }
      });
      ...
    }

    ...

    public void setZoom(final int index) {
      if (images[index] == null) {
        zoomLevels.setEnabled(false);
       viewport.addStyleName("waitCursor");
```

```
        images[index] = new Image();

        images[index].addLoadHandler(new LoadHandler() {
          public void onLoad(LoadEvent event) {
            zoomLevels.setEnabled(true);
           viewport.removeStyleName("waitCursor");
          }
        });
    }
    images[index].setUrl(imageUrls[index]);
    viewport.setView(images[index]);
  }
```

リスト 4 では、ズーム・レベル・プルダウンに変更ハンドラーを追加しました (イベント・ハンドラーについての詳細は、第 1 回を参照してください)。この変更ハンドラーが呼び出す `setZoom()` メソッドが、画像がすでにロードされているかどうかをチェックします。まだロードされていない場合、`setZoom()` メソッドは画像にロード・ハンドラーを追加し、カーソルを待機カーソルに変更し、ズーム・プルダウンを無効にします。その後、画像がロードされた時点で GWT がロード・ハンドラーの `onLoad` メソッドを呼び出すと、ロード・ハンドラーはカーソルを元の表示に戻し、ズーム・プルダウンを再び有効にします。`waitCursor` の CSS スタイルは、アプリケーションのスタイル・シートに以下のように定義されます。

```
.waitCursor {
  cursor: wait;
}
```

モジュール

第 1 回で説明したように、すべての GWT アプリケーションはその 1 つひとつがモジュールですが、その逆のことはありません。つまり、すべてのモジュールが GWT アプリケーションであるわけではありません。モジュールは再利用するためのメカニズムであり、GWT アプリケーションは他のモジュールを必要なだけ利用することができるのです。

モジュールは、単に成果物にモジュール定義 XML ファイルが付随しただけのものにすぎません。例えばカスタム GWT ウィジェットであることもあれば、JavaScript や何らかの CSS であることもあります。ここで使用する `Viewport` クラスは独自の Widgets モジュールに含めてあります。図 4 に、この Widgets モジュールのソース・ファイルを記載します。

図 4. Widgets モジュールのソース・ファイル

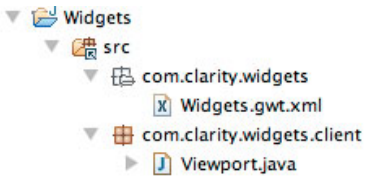


図 4 に記載したモジュールはこの上なく単純な形のモジュールで、このモジュールにあるのはモジュール構成ファイル (Widgets.gwt.xml) とビューポートの実装 (Viewport.java) だけです。構成ファイルにしても、この上なく単純な内容になっています (リスト 5 を参照)。

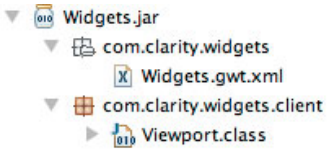
リスト 5. Widgets.gwt.xml

```
<module rename-to='widgets'>
  <inherits name='com.google.gwt.user.User'/>
</module>
```

Widgets モジュールは、GWT のコア・コードが含まれる GWT User モジュールを継承しています。

モジュール構成ファイルを作成して `Viewport` クラスを実装した後、この Widgets モジュールが含まれる JAR ファイルを作成しました (図 5 を参照)。

図 5. Widgets モジュールの JAR



Widgets モジュールを Places アプリケーションで使用するためには、以下の 2 つの作業を行います。

- Widgets の JAR ファイルをクラス・パスに設定
- Widgets モジュールを Places アプリケーションの XML 構成ファイルで継承

モジュールを使用することによって、カスタム・ウィジェットとその関連成果物をパッケージ化し、他の開発者が使用できるようにできます。アプリケーションはモジュールを必要な数だけ継承することが可能であり、モジュールには他のモジュールを含めることもできます。実際、モジュールは必要な深さにまでネストすることができます。

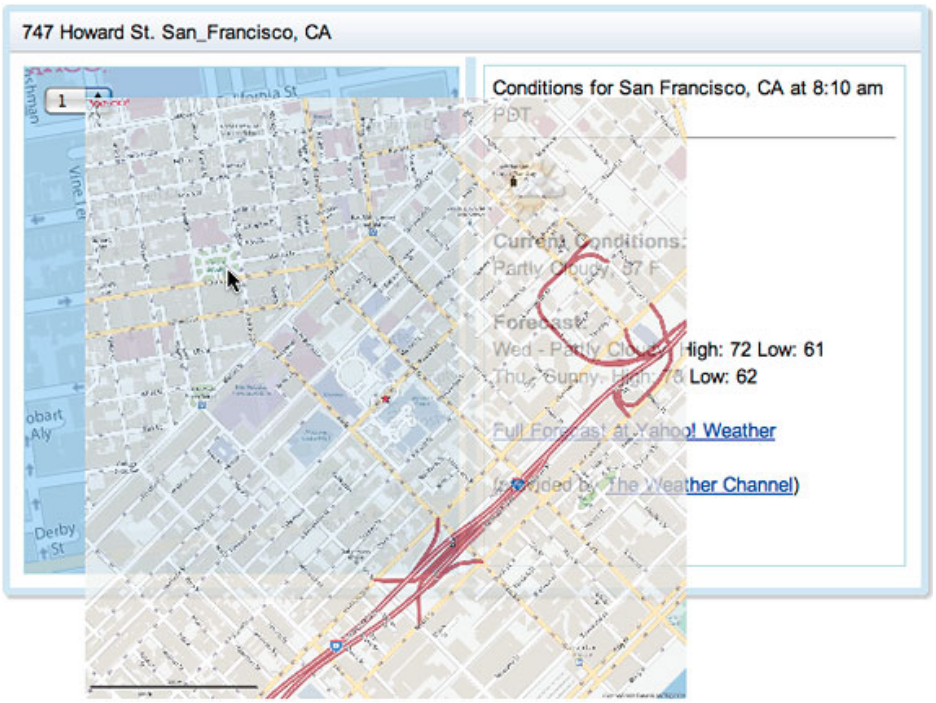
イベント・プレビュー

`Viewport` クラスでは、図 6 に示すようにビューポートのビューをドラッグすることができます。

図 6. ビューポート内での地図のドラッグ

もうご存知だと思いますが、GWT にはマウス・ハンドラーと、パネル内でウィジェットをピクセル単位の位置で配置できる絶対パネルが付属しています。マウス・ハンドラーと絶対パネルを組み合わせることで、図 6 に示すように、絶対パネル内でのウィジェットのドラッグ操作が可能になります。その一方で、画像をドラッグしようとすると、ブラウザーがそのドラッグ操作に干渉してきます (図 7 を参照)

図 7. 画像のドラッグ操作に対するブラウザーの干渉



私が使い慣れているブラウザーでは、いずれも画像をドラッグすることが可能で、ユーザーが絶対パネル内で画像をドラッグすると、ブラウザーが図 7 のように画像をドラッグしてユーザーの操作を支援しようとします。けれどもこのアプリケーションの場合、画像をドラッグする際のブラウザーによる支援は、ユーザーによる画像のドラッグ操作の妨げになるので望ましくありません。

そこで必要となるのは、ブラウザーにアプリケーションでの画像のドラッグ操作に干渉しないように指示する方法です。その方法としては、イベントをプレビューすることによるものがあります (リスト 6 を参照)。

リスト 6. イベントのプレビュー

```
public class Viewport extends AbsolutePanel {
  ...
  private HandlerRegistration handlerRegistration = null;

  private Event.NativePreviewHandler preventDefaultMouseEvents =
    new Event.NativePreviewHandler() {
    public void onPreviewNativeEvent(NativePreviewEvent event) {
      if (event.getTypeInt() == Event.ONMOUSEDOWN
          || event.getTypeInt() == Event.ONMOUSEMOVE) {
        event.getNativeEvent().preventDefault();
      }
    }
  };

  addDomHandler(new MouseOverHandler() {
    public void onMouseOver(MouseOverEvent event) {
      handlerRegistration = Event.addNativePreviewHandler(preventDefaultMouseEvents);
    }
  }, MouseOverEvent.getType());

  addDomHandler(new MouseOutHandler() {
```

```
      public void onMouseOut(MouseOutEvent event) {
        if (handlerRegistration != null) {
          handlerRegistration.removeHandler();
        }
      }
    }, MouseOutEvent.getType());
    ...
  }
}
```

リスト 6 で実装しているのは、ネイティブ・プレビュー・ハンドラーです。その名前が示すように、ネイティブ・プレビュー・ハンドラーを使うと、GWT やブラウザーに干渉する隙を与える前にイベントをプレビューすることができます。イベント・プレビューでは、マウスダウン・イベントやマウス移動イベントにブラウザーが反応しないようにするため、ネイティブ・イベントに対して `preventDefault()` を呼び出します。このメソッドはマウス・イベントに対するブラウザーのデフォルトの応答を防ぐことから、`preventDefault()` という名前になっています。

カーソルがビューポート内に入ると、ネイティブ・プレビュー・ハンドラーを GWT のイベント・スタックの先頭に追加し、カーソルがビューポートの外に出ると、ネイティブ・プレビュー・ハンドラーをスタックから削除して通常のイベント処理に戻るようにしています。つまり、カーソルがビューポート内にある間は、マウスダウン・イベントまたはマウス移動イベントに対してブラウザーのデフォルトの応答動作が実行されるのを GWT が防いでいます。そうすることにより、ブラウザーがユーザーによる画像のドラッグを干渉しないようにするという仕組みです。

タイマー

ビューポート内でマウスを素早く (0.5 秒未満) ドラッグすると、ビューポートはマウスがドラッグされた方向に地図をアニメーション化して移動します。このとき地図が移動する速度は、ドラッグ中に移動したピクセル数に比例します。アニメーション化を使った地図の移動は、ビューポート内でマウスをクリックするまで続きます。このアニメーション化には GWT タイマーを使っています (リスト 7 を参照)。

リスト 7. GWT タイマーによるビューポート内でのビューのアニメーション化

```
private static final int TIMER_REPEAT_INTERVAL = 50;
private static final int SPEED_FACTOR_MULTIPLIER = 20;

...

timer = new Timer() {
  public void run() {
    // Calculate new X and Y locations for the
    // mouse panel, and reposition it.
    int newX = getWidgetLeft(view) - (int) (unitVectorX * speedFactor);
    int newY = getWidgetTop(view) - (int) (unitVectorY * speedFactor);

    repositionView(newX, newY);
  }
};

...

addDomHandler(new MouseUpHandler() {
  public void onMouseUp(MouseUpEvent event) {
    if (deltaTime < gestureTimeThreshold &
       (deltaX > gesturePixelThreshold || deltaY > gesturePixelThreshold)) {
```

```
      speedFactor = ((deltaX + deltaY) / (timeUp - timeDown)) * SPEED_FACTOR_MULTIPLIER;
      timer.scheduleRepeating(TIMER_REPEAT_INTERVAL);
      timerRunning = true;
    }
  }
}
...

addDomHandler(new MouseDownHandler() {
 public void onMouseDown(MouseDownEvent event) {
    System.out.println("mouse down " + event.getX() + ", " + event.getY());
    int x = event.getX(), y = event.getY();

    if (isGesturesEnabled() & timerRunning) {
      // On a mouse down, if the timer is running, stop it.
      timerRunning = false;
      timer.cancel();
    }
}
...
private void repositionView(int newX, int newY) {
  // Check to see if the view scrolled out of sight;
  // if so, bring it back in view
  if (newX > 0) {
    newX = 0;
    unitVectorX = 0 - unitVectorX;
  } else if (newX < 0 - view.getOffsetWidth() + getOffsetWidth()) {
    newX = 0 - view.getOffsetWidth() + getOffsetWidth();
    unitVectorX = 0 - unitVectorX;
  }

  if (newY > 0) {
    newY = 0;
    unitVectorY = 0 - unitVectorY;
  } else if (newY < 0 - view.getOffsetHeight() + getOffsetHeight()) {
    newY = 0 - view.getOffsetHeight() + getOffsetHeight();
    unitVectorY = 0 - unitVectorY;
  }

  if (isRestrictDragVertical())
    setWidgetPosition(view, xstart, newY);
  else if (isRestrictDragHorizontal())
    setWidgetPosition(view, newX, ystart);
  else
    setWidgetPosition(view, newX, newY);
}
```

リスト 7 の内容はとても簡単になっています。ここで実装しているタイマーは、ビューポート内での地図の新しい位置を計算し、それからヘルパー・メソッド (`repositionView()`) を呼び出して地図の位置を変更します。

ビューポートのマウスアップ・ハンドラーでは、タイマーの `scheduleRepeating` メソッドを呼び出しています。このメソッドがタイマーを開始し、`TIMER_REPEAT_INTERVAL` (50 ミリ秒) の間隔の周期タイマーを実行します。その後、ビューポート内でマウス・ボタンが押下されたときに周期タイマーが実行中であれば、タイマーの `cancel()` メソッドを呼び出して周期タイマーを停止します。

`Viewport` クラスは約 350 行に及ぶ長いものなので記事には記載しませんが、Places アプリケーションのソースをダウンロードすればコードを見ることはできます。

まとめ

GWT は想像し得るものを何でも実装できる魅力的なフレームワークです。そしてこのフレームワークに備わった強力な Swing 風の API を使えば、ブラウザーで動作するリッチ・クライアントのユーザー・インターフェースを作成することができます。この 2 回の連載では、GWT の基本機能と、タイマーの使用やイベント・プレビューといった高度な機能の両方を紹介しました。連載をひと通り読んでサンプル・アプリケーションをダウンロードすれば、早速 GWT を使って独自のリッチ・クライアントのユーザー・インターフェースを実装する作業に取り掛かれます。

この短い連載では重要な基本機能と高度な機能をいくつか取り上げただけですが、ご想像どおり、このフレームワークには他にも豊富な機能が備わっています。その一例として、最近では Google App Engine が統合されました。さらに GWT の今後のバージョンには、ドラッグ・アンド・ドロップの組み込みサポートなどのより魅力的な機能が次々と追加されていくはずです。「参考文献」を調べて最新情報を入手してください。

ダウンロード

内容	ファイル名	サイズ
Source code for the places application	j-gwtfu-part1-code	690KB

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、Java University では講座を受け持っています。彼は JavaOne ロック・スターに 2 回選ばれました。