CodeZine
デベロッパーの成長と課題解決に貢献するメディア

Javaで軽快に使える「軽量フレームワーク」特集

Javaで軽快に使える「軽量フレームワーク」特集 ～クールなGUIをシンプルなスクリプトで作成するZK（3）

第22回

Java

掌田 津耶乃[著]

2010/09/02 14:00

注目の軽量フレームワークをフットワーク軽く取り上げていく本連載。今回は、JavaをベースにしたRIA開発のフレームワーク「ZK」の第3回として、コンポーネント以外に用意されている重要な機能について、いくつかピックアップして整理していきます。

はじめに

今回は、これまで紹介したコンポーネント以外の機能から、重要なものをいくつかピックアップして補足していくことにしましょう。ZKは、基本的にプレゼンテーション層のフレームワークであり、コンポーネントによるページの作成を第一に考えられています。しかし、Webページを作る上で、コンポーネント以外にも必要となる機能はいろいろと考えられますし、コンポーネントを活用する上で理解しておきたい機能というのもまたあります。

例えば、GUI関連の機能としては「ドラッグ＆ドロップ」や「メニュー」、「マルチウインドウ」といったものがあげられます。またWebアプリケーションでデータを扱う際に多用される「セッション」の利用方法も重要ですし、「データベース・アクセス」はWebアプリケーション開発では外すことのできない機能でしょう。今回は、これら「コンポーネント以外の機能」について説明していきます。

対象読者

- Javaで手ごろなフレームワークを探している技術者
- 最近のフレームワークをごくざっと理解しておきたい方
- Web開発の手法がどうも気に入らない、と常々考えているJavaプログラマ

ドラッグ＆ドロップ

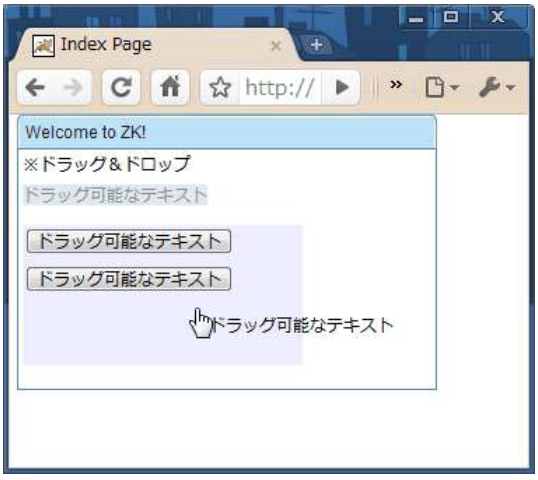
まずは、コンポーネントのドラッグ＆ドロップについてです。ZKのコンポーネントには「draggable」「droppable」という属性が用意されており、これにtrueを設定するだけで、ドラッグ可能・ドロップ可能にできます。

問題は「ドロップされたときにどのようにして処理を行うか」でしょう。これは「onDrop」という属性に処理を用意することで解決できます。では、実際にやってみましょう。

```
<?page title="Index Page"?>
<window title="Welcome to ZK!" border="normal" width="300px">
  <label>※ドラッグ＆ドロップ</label>${gmsg}
  <separator />
  <label draggable="true">ドラッグ可能なテキスト</label>
  <separator /><separator />
  <vbox droppable="true" style="background-color:#eeeeff"
      width="200px" height="100px" onDrop="doDropped(self,event.dragged)">
  </vbox>
```

```
  <separator /><separator />
  <zscript>
  void doDropped(obj1,obj2){
    obj1.appendChild(new Button(obj2.value));
  }
  </zscript>
</window>
```

テキストを四角いエリアにドラッグ＆ドロップすると、ボタンとして追加される



これは、ドラッグ可能なテキスト（ラベル）と、ドロップ可能なエリア（ボックス）を表示するサンプルです。「ドラッグ可能なテキスト」というテキストをマウスでドラッグすると、そのまま動かすことができます。これを下のグレーのエリアにドロップすると、ボタンとして組み込まれます。

ここでは、<vbox>の属性にonDrop="doDropped(self,event.dragged)"という形でドロップ時の処理を設定しています。doDroppedグローバル関数を呼び出し、selfとevent.draggedを渡しています。selfはイベントが発生したコンポーネント（つまりドロップされたコンポーネント）、eventのdraggedはイベントオブジェクトに用意されているプロパティで、ドラッグしたコンポーネントを示すものです。これで、ドロップされた側とドラッグした側の両方のオブジェクトが引数で渡されます。

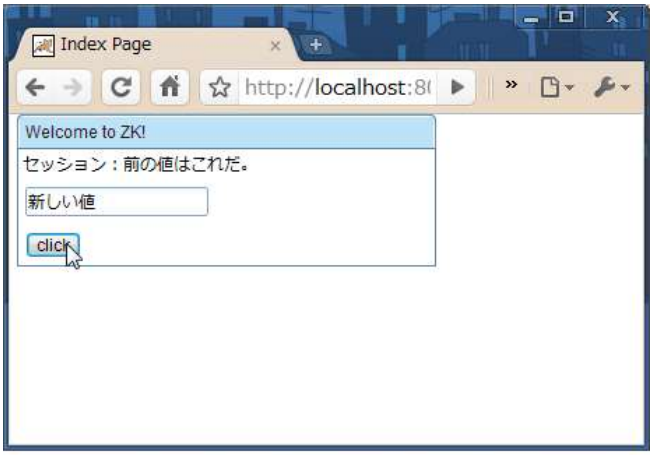
doDropped関数では、ドラッグしたコンポーネントのvalueを取得し、新しいButtonを作成して、これをドロップされた側に組み込みます。組み込みは「appendChild」というメソッドを使います。これはコンテナに引数のコンポーネントを組み込むメソッドです。

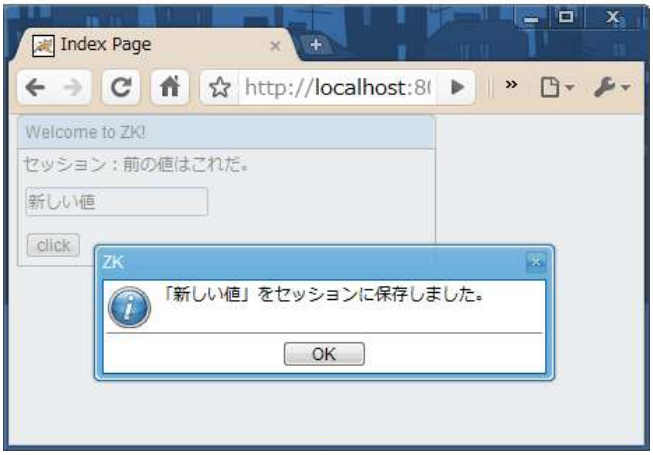
セッションの利用

常に接続を維持し、値などを保持し続けるのに、セッションは不可欠です。ZKでも、もちろんセッションを利用できます。簡単なサンプルを挙げておきましょう。

```
<?page title="Index Page"?>
<window title="Welcome to ZK!" border="normal" width="300px">
  <zscript>
  var gmsg = session.getAttribute("msg");
  void doAction(){
    String s = text1.value;
    session.setAttribute("msg",s);
    alert("「" + s + "」をセッションに保存しました。");
  }
  </zscript>
  <label>セッション：${gmsg}</label>
  <separator/>
  <textbox id="text1" />
  <separator/>
  <button label="click" onClick="doAction()" />
</window>
```

テキストを入力してボタンを押すと、その値がセッションに保管される





入力フィールドにテキストを書き、［save］ボタンを押すと、そのテキストがセッションに保管されます。以後、ページをリロードすると保存した値が表示されるようになります。ここでは、sessionオブジェクトの「setAttribute」「getAttribute」というメソッドを利用してセッションへの値のやり取りを行っています。このsessionは、JSPの暗黙オブジェクトと同様、HttpSessionのインスタンスです。しかし、JSPの暗黙オブジェクトがそのまま使えるわけではありません。例えば、sessionやapplicationなどは使えますが、requestやresponse、outといったもっとも重要なものは使えません。

applicationやsessionについては、ZKのシステムに何ら影響を与えるものではありませんが、requestやresponseなどは「クライアント側とサーバ側を分離しない」というZKのシステムからすれば、使えることができてはまずいものです。outによる出力も、プレゼンテーション層を管理するためのZKというフレームワークからすれば利用できない方がいいものなので、用意されていないのでしょう。

従って、これらは「JSPの暗黙オブジェクトが使える」というより、「ZKが独自に、必要最低限の暗黙オブジェクトを用意してある」と考えた方が良いでしょう。実際、ZKには、JSPにはなかった暗黙オブジェクトが多数揃えられています。現在のイベントを示すeventや、forEachによる繰り返しで用いられるeach、コンポーネント自身を示すselfなども暗黙オブジェクトです。

内部ウインドウの作成

ZKでは、ウインドウを擬似的にWebページ内に配置できました。さらに機能を用意して、パソコンのデスクトップのような表現も可能です。ZKでは、メニューバー、ダイナミックなコンポーネント（ウインドウ）の生成、ウインドウ操作時のイベント処理など、基本的な機能は揃っているので、ちょっとコードを書けば、擬似デスクトップを作りマルチウインドウなGUIを構築することもできます。

実際に簡単なサンプルを作ってみましょう。まず、ページ内に内部ウインドウとして表示させるzulファイルを作成します。ここでは「other.zul」という名前で、Webアプリケーションのルート（index.zulと同じ階層）に配置しておくことにします。ソースコードはざっと次のようにしておきましょう。

※other.zul

```
<window title="ウインドウ" border="normal"
    height="100px" width="300px"
    minheight="100" minwidth="100">
  <zscript>
  void doAction(){
    alert("ok");
```

```
  }
  </zscript>
  <label>※ウインドウ</label>
  <separator/>
  <button label="click" onClick="doAction()" />
</window>
```

内部ウインドウ用に用意したzulファイル。このウインドウが、ブラウザ内にマルチウインドウで現れる

ここでは、<window>タグ内にラベルとボタンを用意してあります。ボタンをクリックすると、簡単なメッセージが表示されるようにしてあります。これは「内部ウインドウ内できちんとスクリプトが機能するか」を確認するためのものなので、内容は適当に変えて構いません。

それよりも重要なのは、<window>タグに用意されている属性です。ここではheight／widthの他に「minheight」「minwidth」というものが用意されています。これらは、最低の高さと幅を示すものです。これにより、指定したサイズより小さくウインドウをリサイズできないようにします。

メニューバーとマルチウインドウ処理

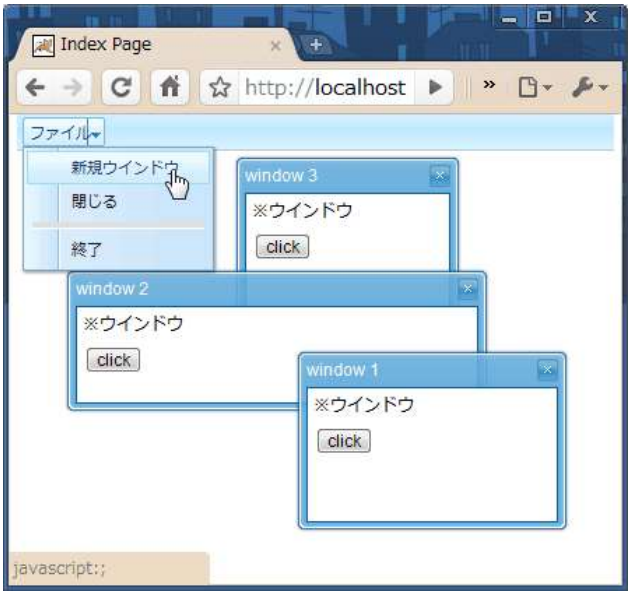
では、このother.zulを内部にマルチウインドウで表示するサンプルを作成しましょう。よりデスクトップらしく見せるためメニューバーを用意し、メニューを選んでウインドウを開いたり閉じたりするようにしてみます。

※index.zul

```
<?page id="other" title="Index Page"?>
<zk>
  <menubar id="menubar" width="100%">
    <menu label="Project" image="/img/Centigrade-Widget-Icons/Briefcase-16x16.png">
      <menupopup>
        <menuitem label="新規ウインドウ" onClick="doAction()" />
        <menuitem label="閉じる" onClick="doClose()" />
        <menuseparator />
        <menuitem label="終了" onClick="alert(self.label)" />
      </menupopup>
    </menu>
  </menubar>
  <zscript>
  var selectedWindow = null;
  var counter = 0;

  void doAction(){
    var mywindow = (org.zkoss.zul.Window)Executions.createComponents
        ("/other.zul",null,null);
    mywindow.setTitle("window " + ++counter);
    mywindow.setSizable(true);
    mywindow.setClosable(true);
    mywindow.doOverlapped();
    mywindow.addEventListener("onZIndex",new MyAdapter());
  }
  void doClose(){
    if (selectedWindow != null){
      selectedWindow.onClose();
    }
  }
```

```
  class MyAdapter implements EventListener {
    public void onEvent(Event event){
      selectedWindow = event.getTarget();
    }
  }
  </zscript>
</zk>
```

［ファイル］メニューから［新規ウインドウ］を選ぶと、次々とウインドウが現れる



ここでは、アクセスすると［ファイル］というメニューを持つメニューバーが現れます。ここから［新規ウインドウ］メニューを選ぶと、画面にother.zulのウインドウが現れます。ウインドウはマウスでクリックして選択したり、ドラッグして移動やリサイズが可能です。またメニューから［閉じる］を選ぶと、現在選択されているウインドウが閉じられます。

メニューの構成

メニューは、全部で4種類のコンポーネントから構成されています。ここで使われているメニュー関係のタグを整理すると次のようになっていることが分かります。

- **<menubar>**：メニューバー。これはバーだけで、何も項目は表示されない
- **<menu>**：メニュー。これがメニューバーに表示されるメニューとなる
- **<menupopup>**：メニューをクリックすると下に現れる部分。ポップアップメニューとして定義される
- **<menuitem>**：ポップアップメニューに表示されるメニュー項目。これが実際に選択されるメニューになる

このように、メニューは4つの部品から構成されます。これらのタグを構造的に記述すれば、それだけでメニューが作成されるため、メニューの表示や動作のためにコードを書く必要はありません。

ウインドウを作成し表示する

続いて、「新規ウインドウ」メニューで新しいウインドウを表示する処理についてです。これは、doActionというグローバル関数として定義されています。ここでは、まずウインドウであるWindowクラスのインスタンスを生成し、そのプロパティなどを設定していきます。

```
var mywindow = (org.zkoss.zul.Window)Executions.createComponents
    ("/other.zul",null,null);
```

Windowコンポーネントの生成は、new Windowなどという形で作成をしません。「Executions」という、クライアントからの要求を処理するための専用クラスが用意されており、ここにあるコンポーネント生成のためのメソッド「createComponents」を利用します。これは、引数にコンポーネントのテンプレートとなるzulファイルのURL、親コンポーネント、初期設定するプロパティをまとめたオブジェクトなどが渡されます。ここでは、テンプレートとなるother.zulのURLだけを渡しておきます。

続いて、作成したWindowオブジェクトのプロパティを設定していきます。

```
mywindow.setTitle("window " + ++counter);
mywindow.setSizable(true);
mywindow.setClosable(true);
```

「setTitle」でタイトル、「setSizable」ではリサイズ可能かどうか、「setClosable」は閉じられるかどうかを設定します。これで一通りの設定はできましたが、まだこの状態ではWindowは独立したウインドウとして使うことはできません。

独立したウインドウとして扱えるようにしているのが次の部分です。

```
mywindow.doOverlapped();
```

「doOverlapped」は、ウインドウをオーバラップ可能にするものです。ウインドウは、その性質などからいくつかのモードに分けて考えることができます。Windowにある以下のメソッドで、ウインドウの動作モードを変更できます。

- **doOverlapped**：オーバーラップ・ウインドウにする
- **doPopup**：ポップアップ・ウインドウにする。オーバーラップ・ウインドウと同じように表示されるが、他のエリアをクリックすると消える
- **doModal**：モーダル・ウインドウにする。表示されている間、他の処理を停止し、閉じるまでウインドウ外にアクセスできなくなる
- **doEmbedded**：デフォルトの状態（ドキュメント内に埋め込まれた状態）に戻す

ZIndex変更時のイベント処理

このスクリプトではもう1つ、「ウインドウを閉じる」ための処理が用意されています。これは、メニューを選んだら、1番手前にあるWindowの「onClose」メソッドを呼び出してウインドウを閉じる、というものです。問題は、どうやって1番手前にあるウインドウを知るかということです。いろいろやり方は考えられますが、ここでは「ZIndex変更時のイベント」を利用することにしました。

ZIndexは、コンポーネントの重なり順を示すプロパティです。ウインドウをクリックして1番手前になると、必ずこのイベントが発生します。これを利用し、ZIndexが変更されるイベントが発生したら、そのイベント発生源となるコンポーネントを、1番手前に移動したコンポーネントとして変数selectedWindowに保管するようにしておく、というわけです。こうすれば、selectedWindowには常に1番手前のWindowが保管されることになります。

イベントの組み込みは、Windowインスタンスを作成したところで行っています。これは「addEventListener」というメソッドを利用します。

```
mywindow.addEventListener("onZIndex",new MyAdapter());
```

ZKのイベント処理は、Swingなどと同様にデリゲーションイベントモデルを採用しています。しかし、Swingなどのようにイベントリスナーが細分化されているわけではありません。イベントリスナーは「EventListener」というクラス1つだけで、組み込みも「addEventListener」があるのみです。

ではどうやってイベントの区別をするのかと言うと、addEventListner時に、組み込むイベント名とイベントリスナーをそれぞれ引数に指定しておくのです。例えば、ここでは"onZIndex"というイベントに、MyAdapterを組み込んでいます。このようにして、特定のイベントにイベントリスナーを割りつけるようになっています。

肝心のイベントリスナーですが、これはEventListenerをimplementsして作成します。クラスには、「doEvent」というメソッドを1つ用意します。

```
class MyAdapter implements EventListener {
  public void onEvent(Event event){
    selectedWindow = event.getTarget();
  }
}
```

onEventでは発生したイベントの情報をまとめたEventクラスのインスタンスが渡されます。この「getTarget」メソッドにより、イベントが発生したコンポーネントを取得できます。ここでは、この値をselectedWindowに設定しています。ZKには、onZIndex以外にも各種のイベントが用意

されており、すべてこれと同様にして処理を組み込むことができます。

JavaDBへのアクセス

最後に、ZKからデータベースにアクセスする方法について整理します。ZKには、データベースアクセスのための特別な機能は用意されていません。しかし、ZKは一般的なWebアプリケーションのようにクライアントとサーバにわかれて設計せず、普通のアプリケーションと同じ感覚でプログラムを設計できます。従って、ごく普通のアプリケーションと同じ感覚で、データベースアクセス用のクラスを作成し、それを必要に応じて呼び出して処理すれば良いのです。

では、サンプルとして、Javaに付属するJavaDBを利用してデータベースアクセスする例を考えてみましょう。JavaDBは、JDK／JREとは別のディレクトリにインストールされています（通常、C:¥Program Files¥Sun¥JavaDB）。データベースを操作する場合には、インストールディレクトリ内の「bin」内にある「ij.bat」を利用するのが良いでしょう。これを起動すると、コマンドプロンプト・ウインドウが開き、JavaDBのコマンドを直接実行できるようになります（なお、そのまま起動すると、JavaDBのログファイルの出力に失敗するエラーが現れることがあります。このような場合には、管理者としてij.batを実行してください）。

起動したら、まずデータベースファイルに接続します。JavaDBでは、データベースサーバとしてJavaDBを起動し、それに接続する方式と、直接データベースファイルにアクセスする方式があります。ここでは扱いが簡単な後者の方式を利用します。ここでは、Cドライブ直下に「DerbyDatabases」というフォルダを用意し、この中に「sampledata」というデータベースファイルを作成して利用することにしましょう。ij.batのコマンドプロンプト・ウインドウで、次のように実行し、C:¥DerbyDatabases¥sampledataというデータベースファイルを作成して接続してください。

```
connect 'jdbc:derby:C:¥DerbyDatabases¥sampledata;create=true';
```

続いて、簡単なテーブルを作成しておきましょう。name,mail,telといった項目を持つSampleRowDataテーブルを作成します。

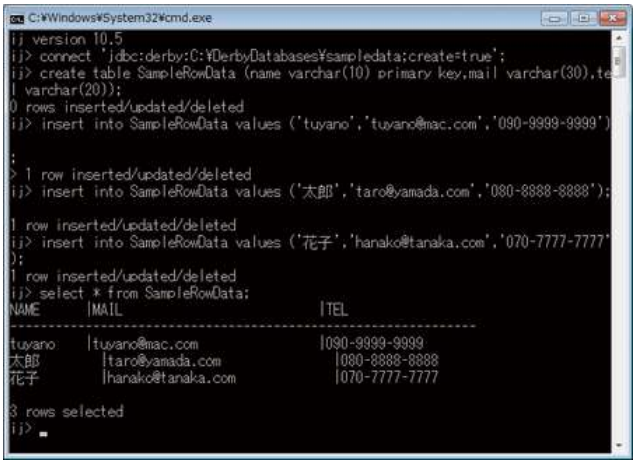
```
create table SampleRowData (name varchar(10) primary key,mail varchar(30),tel varchar(20));
```

ダミーのデータも次のようにいくつか用意しておきます。

```
insert into SampleRowData values ('tuyano','tuyano@mac.com','090-9999-9999');
insert into SampleRowData values ('太郎','taro@yamada.com','080-8888-8888');
insert into SampleRowData values ('花子','hanako@tanaka.com','070-7777-7777');
```

作成後、「select * from SampleRowData;」でデータが保管されていることを確認してから、ij.batを終了してください。とりあえず、これでサンプルのテーブルは用意できました。

ij.batを起動し、サンプルのデータベースファイルにテーブルを用意する



DAOクラスの作成

では、続いて用意したデータベースにアクセスするためのDAO（Data Access Object）クラスを作成しましょう。これは、Javaのクラスとして用意してもいいですが、今回はZKのスクリプトにそのまま埋め込んで利用できる形で作ることにします（スクリプトはこの後に作成します）。

```
public class DAO {
  String url = "jdbc:derby:C:¥¥DerbyDatabases¥¥sampledata;create=true";

  public DAO() {
    try {
      Class.forName("org.apache.derby.jdbc.EmbeddedDriver");
    } catch (ClassNotFoundException e) {
      e.printStackTrace();
    }
  }

  public List findAll(){
    java.sql.Connection conn = null;
    java.sql.Statement stmt = null;
    List allDatas = new ArrayList();
    try {
      conn = java.sql.DriverManager.getConnection(url);
      stmt = conn.createStatement();
      String query = "select * from SampleRowData";
      java.sql.ResultSet rs = stmt.executeQuery(query);
      gmsg = query;
      SampleRowData data;
      while (rs.next()) {
        data = new SampleRowData();
        data.setName(rs.getString(1));
        data.setMail(rs.getString(2));
        data.setTel(rs.getString(3));
        allDatas.add(data);
      }

    } catch (java.sql.SQLException e) {
      e.printStackTrace();
      gmsg = e;
    } finally {
      try {
        stmt.close();
      } catch (java.sql.SQLException e) {
        e.printStackTrace();
        gmsg = e;
      }
      try {
        conn.close();
      } catch (SQLException e) {
        e.printStackTrace();
        gmsg = e;
      }
    }
    return allDatas;
  }

  public boolean delete(String name){
    java.sql.Connection conn = null;
    java.sql.Statement stmt = null;
    boolean result = false;
    try {
      conn = java.sql.DriverManager.getConnection(url);
      stmt = conn.createStatement();
      if (stmt.executeUpdate("delete from SampleRowData where name = '" +
                name + "'") > 0){
        result = true;
      }
    } catch (java.sql.SQLException e) {
      e.printStackTrace();
      gmsg = e;
    }finally {
      try {
        stmt.close();
```

```
      } catch (SQLException e) {
        e.printStackTrace();
        gmsg = e;
      }
      try {
        conn.close();
      } catch (java.sql.SQLException e) {
        e.printStackTrace();
        gmsg = e;
      }
    }
    return result;
  }

  public boolean insert(SampleRowData data){
    java.sql.Connection conn = null;
    java.sql.Statement stmt = null;
    boolean result = false;
    try {
      conn = java.sql.DriverManager.getConnection(url);
      stmt = conn.createStatement();
      String query = "insert into SampleRowData(name,mail,tel) " +
          "values ('" + data.getName() + "','" + data.getMail() +
          "','" + data.getTel() + "')";
      gmsg = query;
      if (stmt.executeUpdate(query) > 0){
        result = true;
      }
    } catch (java.sql.SQLException e) {
      e.printStackTrace();
      gmsg = e;
    } finally {
      try {
        stmt.close();
      } catch (java.sql.SQLException e) {
        e.printStackTrace();
        gmsg = e;
      }
      try {
        conn.close();
      } catch (java.sql.SQLException e) {
        e.printStackTrace();
        gmsg = e;
      }
    }
    return result;
  }

  public boolean update(SampleRowData data){
    java.sql.Connection conn = null;
    java.sql.Statement stmt = null;
    boolean result = false;
    try {
      conn = java.sql.DriverManager.getConnection(url);
      stmt = conn.createStatement();
      if (stmt.executeUpdate("update SampleRowData set  mail = " +
          data.getMail() + ",tel = '" + data.getTel()+
          "' where name = '" + data.getName() + "'") > 0){
       result = true;
      }
    } catch (java.sql.SQLException e) {
      e.printStackTrace();
    }finally{
      try {
        stmt.close();
      } catch (java.sql.SQLException e) {
        e.printStackTrace();
```

```
      }
      try {
        conn.close();
      } catch (java.sql.SQLException e) {
        e.printStackTrace();
      }
    }

    return result;
  }
}
```

DAOクラスには、全データを検索し返す「findAll」、データを追加する「insert」、データを更新する「update」、データを削除する「delete」というメソッドを用意しておきました。いずれも、データをオブジェクトとしてまとめるための「SampleRowData」クラスのインスタンスとしてデータを扱うようにしてあります。以下にざっと整理しておきましょう。

findAllメソッド

データベースにアクセスし、SampleRowDataテーブルのデータを取得し、SampleRowDataインスタンスのListとして返すものです。データベースへのアクセスは、java.sql.DriverManagerのgetConnectionでデータベースに接続し、createStatementでStatementインスタンスを用意し、executeQueryでクエリーを実行する、という形で行います。

```
conn = java.sql.DriverManager.getConnection(url);
stmt = conn.createStatement();
String query = "select * from SampleRowData";
java.sql.ResultSet rs = stmt.executeQuery(query);
```

後は、executeQueryで取得したResultSetから、順にデータを取り出し、SampleRowDataインスタンスを作成してListに保管する、といったことを繰り返していきます。Listの形でデータがまとめられていれば、後でスクリプトから利用する場合も扱いやすくなるでしょう。

```
while (rs.next()) {
  data = new SampleRowData();
  data.setName(rs.getString(1));
  data.setMail(rs.getString(2));
  data.setTel(rs.getString(3));
  allDatas.add(data);
}
```

insert／updateメソッド

insertは追加のためのメソッドです。引数にSampleRowDataを渡すと、これをデータベースに追加します。DriverManager.getConnectionでデータベースに接続し、createStatementでStatementを作成した後、executeUpdateでクエリーを実行する、という手順は同じです。

```
conn = java.sql.DriverManager.getConnection(url);
stmt = conn.createStatement();
String query = "insert into SampleRowData(name,mail,tel) " +
    "values ('" + data.getName() + "','" + data.getMail() +
    "','" + data.getTel() + "')";
if (stmt.executeUpdate(query) > 0){……
```

updateも、基本的な流れは同じです。引数にSampleRowDataインスタンスを渡すと、そのnameのデータを更新します。

deleteメソッド

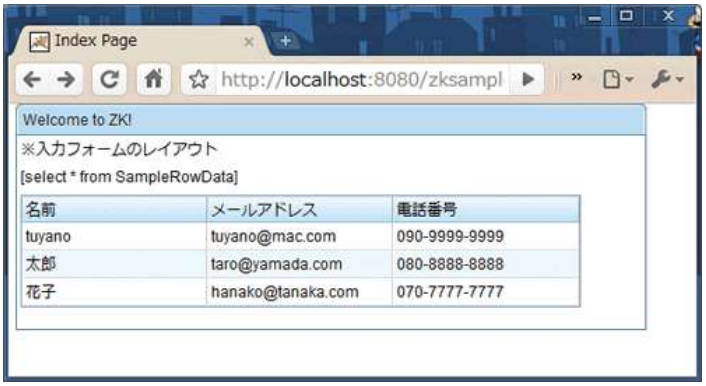
引数に、削除するデータのnameを渡すと、そのデータをデータベースから削除します。これも基本的な流れは同じです。

```
stmt.executeUpdate("delete from SampleRowData where name = '" + name + "'");
```

DAO経由でデータベースにアクセスする

では、このDAOクラスを利用してデータベースアクセスするスクリプトを作成してみることにしましょう。ここでは一例として、findAllを呼び出してデータの一覧を表示させてみます。

```
<?page title="Index Page"?>
<zk>
  <zscript>
  var gmsg = "";

  public class DAO { ……略…… }

  class SampleRowData {
    private String name;
    private String mail;
    private String tel;

    public SampleRowData(){}
    public SampleRowData(String s1,String s2,String s3){
      name = s1;
      mail = s2;
      tel = s3;
    }

    public String getName(){ return name; }
    public void setName(String s){ name = s; }
    public String getMail(){ return mail; }
    public void setMail(String s){ mail = s; }
    public String getTel(){ return tel; }
    public void setTel(String s){ tel = s; }
  }
  class SampleRowRenderer implements RowRenderer {
    public void render(Row row, java.lang.Object obj){
      SampleRowData data = (SampleRowData)obj;
      new Label(data.getName()).setParent(row);
      new Label(data.getMail()).setParent(row);
      new Label(data.getTel()).setParent(row);
    }
  }

  DAO dao = new DAO();
  ArrayList result = dao.findAll();
  ListModelList modellist = new ListModelList(result);
  SampleRowRenderer renderer = new SampleRowRenderer();
  </zscript>

  <window title="Welcome to ZK!" border="normal" width="500px">
    <label>※入力フォームのレイアウト</label>
    <separator />[${gmsg}]<separator />
    <grid width="90%" model="${modellist}" rowRenderer="${renderer}">
      <columns sizable="true">
        <column label="名前" />
        <column label="メールアドレス" />
        <column label="電話番号" />
      </columns>
    </grid>
    <separator /><separator />
  </window>
</zk>
```



アクセスすると、<grid>を使ってデータを表にまとめて表示します。ここではDAOの他、データを保管するためのSampleRowDataクラスと、行データを表示するためのSampleRowRendererクラスを用意してあります。スクリプトでは、DAOインスタンスを作成し、findAllを呼び出してデータをListとして取得し、SampleRowRendererでレンダラーを作成しています。

```
DAO dao = new DAO();
ArrayList result = dao.findAll();
ListModelList modellist = new ListModelList(result);
SampleRowRenderer renderer = new SampleRowRenderer();
```

こうして用意されたデータのListとレンダラーを使い、<grid>にデータを表示させています。それが以下の部分です。

```
<grid width="90%" model="${modellist}" rowRenderer="${renderer}">
  <columns sizable="true">
    <column label="名前" />
    <column label="メールアドレス" />
    <column label="電話番号" />
  </columns>
</grid>
```

見れば分かるように、繰り返しなどは一切使っていませんし、実際にデータを表示する<rows>タグはありません。ここでは、<grid>に「model="${modellist}"」というようにしてmodel属性を設定しています。このようにmodelを使ってListや配列を指定することで、その中身を自動的に表示するようにできるのです。いちいち繰り返しなどを使ってデータを出力する必要はありません。

ここではデータの一覧を表示しているだけですが、DAOには既にデータベースアクセスの基本メソッドは用意されていますから、これを利用してデータベースアクセスするのは簡単です。例えばデータを追加／削除したければ、次のように実行するだけです。

データを追加

```
dao.insert(new SampleRowData("はなこ","hanako@tanaka.com","070-777-777"));
```

データを削除

```
dao.delete("たろう");
```

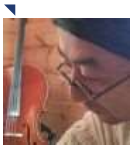
それぞれで、DAOを利用した機能を実装してみるとよいでしょう。ZKには、確かにデータベース専用の機能はありませんが、これは「特別なことは何も無い」と考えることもできます。ごく普通のアプリケーションでデータベースにアクセスできるなら、ZKでもまったく同じやり方でできるはずなのです。

まとめ

3回に渡って、ZKによるWebアプリケーション作成について説明をしてきました。ZKは、厳密に言えば「Javaフレームワーク」とはいえないものでしょう。確かにサーバーサイドJavaの技術を使って実装されており、サーブレットコンテナで動作はしますが、そこで使用するスクリプトはJava（BeanShell）に限定されているわけではなく、JRubyやJythonなどさまざまなスクリプト言語を利用してコーディングできます。Javaの上に構築されていますが、Javaなど知らないRubyやPythonのプログラマがそのままZKで開発を行うこともできるのです。

Javaは、「Javaというプログラミング言語」としてだけでなく、今やさまざまな言語を動かすための「環境」として浸透しつつあります。Javaベースだけれど、Java以外の言語を利用したフレームワークは、今後も増えていくことでしょう。こうした「Javaという環境」を使ったフレームワークの代表格として、ZKは非常に注目すべきものと言えるのではないでしょうか。

著者プロフィール



掌田 津耶乃（ショウダ ツヤノ）

三文ライター＆三流プログラマ。主にビギナーに向けたプログラミング関連の執筆を中心に活動している。 ※現在、入門ドキュメントサイト「libro」、カード型学習サイト「CARD.tuyano.com」を公開中。またGoogle+プロフィールはこちら。

※プロフィールは、執筆時点、または直近の記事の寄稿時点での内容です