Part1：これから始める人のためのTomcat超入門

第1章 Tomcat 4.0で学ぶWebアプリケーション

まずは基礎の基礎！

フリーライター & エンジニア●橋本 修一 HASHIMOTO Osamu
osm@yha.att.ne.jp http://www.hashimoto-net.jp

はじめに

みなさん，こんにちは．本章では，認証，DBアクセスとアイテム取得の表示などの基本的な機能を持ったWebアプリケーションの作成を通して，サーブレット/JSPの基本と，Jakartaプロジェクトの提供する定番サーブレット/JSPコンテナ，Tomcat[注1]の基本的な機能を解説していきます．最新バージョン4.x以降で可能になった応用的な使い方もご紹介しますので，以前からTomcatを利用している方にも，参考になると思います．

なお，今回のアプリケーション製作で使用する環境は下記のとおりです．

OS：Windows 2000 SP2
コンテナ：Tomcat 4.0.4
Java環境：J2SE（Java 2, Standard Edition）SDK 1.4.0-01
データベース：PostgreSQL 7.2.1 Windows-Native版
JDBCドライバ：pgjdbc2.jar

本稿では，Windows上で手軽に使えるDBMS環境として，Windows-Native版PostgreSQLを使用します．

Tomcatのインストールや環境設定については Appendix 1「Tomcat 4.0セットアップ完全ガイド」（169p.）をご参照ください．またPostgreSQLの導入は，Appendix 2「Windows-Native版PostgreSQLのインストール」（172p.）を参照してください．

まずは，サーブレットとJSP（JavaServer Pages）の基礎知識から説明します．

Servlet/JSP Container

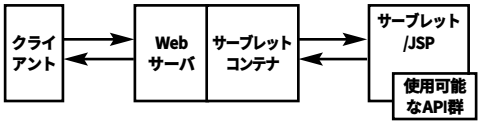
一からおさらい サーブレット/JSP
Jakartaプロジェクト攻略の第一歩

サーブレットのしくみ

サーブレットは，「サーバ上で動くアプレット」と表現されることもあるサーバサイドJavaの最も基本的な技術です．サーブレットコンテナは，Webブラウザなどのクライアントから HTTP プロトコルで送信されてくるメッセージをオブジェクトとして受けとり，必要な処理を行い，再度クライアントにオブジェクトとして返します．これがサーブレットの基本的なしくみです．

注1）ご存知の方が大半かもしれませんが，Tomcatは，Servlet API，JSPの仕様に基づくサーブレット/JSPコンテナのリファレンスインプリメンテーションであり，Jakartaプロジェクトで開発されているソフトウェアの中でも，もっとも基本的なプロダクトということができます．

◆図1 サーブレットのしくみ



もう少し説明すると，HTTP要求オブジェクトとしてクライアントからのリクエストを受けたサーブレットは，使用可能なさまざまなAPI群を呼び出し，開発者の手によって実装されたさまざまなクラス群の機能を利用して，応答オブジェクトとしてクライアントに応答を返すことができます（図1）．

ここで，「使用可能なさまざまなAPI群」というのは具体的には，Servlet APIと通常はJavaの標準ライブラリを指します．これらを用いて以下のような処理が行われます．

- サーブレット起動時に1度だけ実行されるinit( )メソッドで何らかの初期処理，たとえばDBアクセスの際にJDBCドライバのロードなどの処理を行う
- サーブレットが呼び出された際に起動される任意のメソッドで，送信されたHTMLフォームオブジェクトの内容を取り出す
  ➡javax.servlet.http.HttpServletRequest#getParameter(String)にて送信文字列を取り出すなど
- 入力・送信されたパラメータに基づいて処理をする
  ➡ビジネスロジックとプレゼンテーションロジック，MVCモデルで言われるところのModelやViewに処理を振り分けるなど
- サーバサイドにあるライブラリや，内部的に持っているメソッドを使用するなどして，データベースアクセスなどを行う（これはクライアントから送信されたパラメータに基づくもので，パラメータには個人情報などが含まれる場合もある[注2]）．

また「開発者の手によって実装されたさまざまなクラス群」としては，

- Javaの基本的なAPIと環境に依存した情報を利用してデータベースへの接続を行う
- Webページのテキストボックスやコンボボックス，ラジオボタンなどを用いたフォームに入力後，サーバに送信されたパラメータを元に，データベース検索を行う

などが考えられます．この部分は開発者であるみなさんの手によって設計・実装が行われることになります．

JSPのしくみ

JSPは，HTMLの中にプログラムを埋め込んで使うための技術です．JSPのソースは，たとえばTomcatの設定ファイルserver.xmlで指定された任意のディレクトリに配置を行うと，クライアントからの最初のアクセスがあったとき，Tomcatによってサーブレットのプログラムに変換され，それをコンパイルしたものが実行されて，ブラウザなどにHTMLベースの画面などが表示されます．JSPとサーブレットは実行結果としては結局，同じものができますが，JSPの根本的な存在意義であり，仕様が策定された意図としては，

「サーブレットから表示のためのロジックを分離する」

という点が挙げられます．JSPとして表示ロジックが分離されることで，サーブレットは主にサーバサイドの制御処理を行う立場に回ることになります．

実際にサーバサイドJavaの開発経験がある方ならおわかりと思いますが，JSPが今ほどポピュラーでなく，クライアントへのHTML文字列を出力するためにサーブレットを用いていたころ（といってもそんなに昔ではありません）には，Webブラウザへの画面出力のためには非常に可読性の低いロジックを実装する必要がありました．このとき，サーブレットはクライアントに対してHTML形式のテキストを送信しますが，HTMLのタグをコードに埋め込まなければならず，実際のコードは，製作するときならまだしも，見返してみたり自分以外の人が作成したソースを見る必要があったりすると，これは大変です．HTMLも（マークアップ）言語としては成立していますが，サーブレットの文法とは当然重なるため，どこからどこまでがHTMLタグの対象範囲にあるのか，ぱっと見て判断す

注2）データベースのデータに電話番号その他の個人情報などがある場合には，「脆弱性」や「クロスサイトスクリプティング」などのキーワードで指摘されるセキュリティ上の問題を回避するため，可能な限りサーバサイドで処理を完結することなども求められます．

リスト1 HelloWorld.jsp

```
<!-- HelloWorld.jsp -->
<%@ page pageEncoding="Shift_JIS"%>
<%@ page contentType="text/html; charset=Shift_JIS" %>
<html>
<head><title>Hello World!</title></head>
<body>
<h1>世界よこんにちは. </h1>
</body>
</html>
```

◆図2 HelloWorld.jsp

るのは大変なものでした.

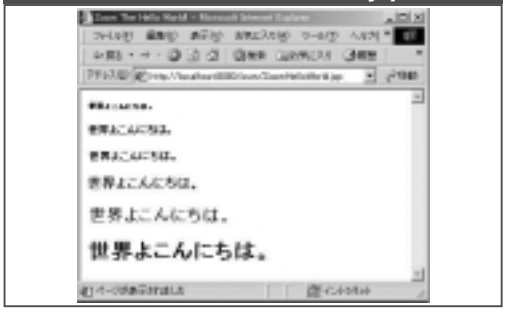
そこで登場したのがJSPです. JSPの記述方法は,一見したところHTMLタグの羅列にも見えます（リスト1）. この例ではJSPファイル, HTMLデータの文字エンコーディング, contentTypeの設定を行っているだけですが, <%@>～<%>で囲まれた部分がJSPタグになります. リスト1をブラウザ上で実行すると, 図2のようになります.

リスト1はなんら変わりばえのしないHTML文書にも見えますが, このファイルを, .html（.htm）ではなく .jsp という拡張子を付けて保存し, サーブレットコンテナで有効となる任意のフォルダに置いてコンテナを起動し, Webブラウザからアクセスをすると, 前述のように拡張子 .javaのファイルが生成され, さらにコンパイルされてサーブレットとして文字列をクライアントであるWebブラウザへ送信します.

リスト2 ZoomHelloWorld.jsp

```
<!-- ZoomHelloWorld.jsp -->
<%@ page pageEncoding="Shift_JIS"%>
<%@ page contentType="text/html; charset=Shift_JIS" %>
<html>
<head><title>Zoom The Hello World!</title></head>
<body>
<% for ( int i = 6; i >= 1; i-- ) { %>
    <h<%= i %>>世界よこんにちは. </h<%= i %>>
<% } %>
</body>
</html>
```

◆図3 ZoomHelloWorld.jsp



次にもう少しJSP“らしい”プログラムにするために, 少々手を入れてみましょう. ループ処理を使ってみます（リスト2）.

リスト2では, スクリプトレットと呼ばれる <% ～ %> で囲まれた部分にJavaのソースを記述しています. こちらも, 先ほどと同じようにブラウザ上で実行すると, 図3のようになります.

Servlet/JSP Container

Tomcatによるサーブレット/JSPシステムの基礎知識
顧客情報閲覧システムを作る

サーブレット/JSPのもっとも基礎的なしくみを説明したところで, 今度はTomcat 4.xを用いてWebアプリケーションを実際に作りながら, その基礎を学んでいきましょう. まずは, 今回作成するWebアプリケーションの概要から説明します. なお, 本特集で解説に使用するソースコードは, 本書サポートページ http://www.gihyo.co.jp/wdpress/jakartaよりダウンロードしてご覧ください. またサンプルアプリケーションの導入方法は, 174ページのコラムに記載しました.

要件定義と設計

サンプルアプリケーションの概要

とある会社（以下, A社）では顧客のニーズなどをつかむために, 顧客情報をデータベース化しています.

第1章◎Tomcat 4.0で学ぶWebアプリケーション
まずは基礎の基礎！

今回，A社のIT部門担当者に，

「顧客の情報一覧をWebブラウザで表示させたい」

という要望が届けられました．

「前提として，一部のユーザ以外には公開しないプライベートなデータもあるので，ユーザごとに閲覧する部分を制限したい」

ということを考えているのだそうです．

ここから，システムのユーザのタイプが2つ想定されます．1つは「一般ユーザ」で，顧客の情報としては

- 性別
- 年齢
- 職業
- 都道府県
- メモ

を公開することにします（図4）．

- 氏名（ローマ字）
- 氏名（漢字）
- 会社名
- 部署
- 役職
- メールアドレス
- 住所
- 電話番号
- 携帯番号

というデータはプライバシの問題があるので，もう1つのユーザ群「管理ユーザ」にしか見せません．

これらの顧客情報データはデータベースに格納してあるものを表示しますので，Javaでデータベースへ接続するためのロジックが必要になります．その際は，J2SE（Java 2, Standard Edition）に標準搭載されているAPI，java.sqlのパッケージをimportして使うことによってJDBCドライバの機能を呼び出して接続するのが一般的です．

サーブレット/JSPに行わせる処理

今回は，表示を行うユーザインタフェース部分とロジック部分，それらに処理を振り分ける部分の3つに分けて設計・実装を行うことにします．

アプリケーションの動作としてはまず，クライアントから送信される入力パラメータを元にデータベースから閲覧するデータを取得して，データをいったん構造化します．そして表示を行うモジュールへ送信し，ブラウザ上で画面出力を行います．

ここでは，サーブレット，JSPにそれぞれ以下のような処理を行わせることにします．

サーブレットで行う処理

- 認証画面からコールされた際のDBアクセスや，表示される内容のJSPへの振り分けをする
- 認証を経ていないアクセスは，認証画面に戻してログインを促す
- 認証画面で得たユーザアカウント情報より，ユーザの閲覧範囲を決定し，パラメータにより表示するデータをDBから取得する
- アカウント情報をキーとしてデータベースにアクセスし，検索を行う
- JSPに処理を委譲する以前に，サーブレットでのアクセスは成功しているか確かめる（デバッグ時の処理なので，通常はオフとする）
- Javaの標準ライブラリのオブジェクトを使用して，取得データのビューを格納する
- 例外が発生した場合は，JSPでメッセージを表示できるようにする
- HTTP応答オブジェクトのタイプを設定する

◆図4　今回作成するWebアプリケーション

リスト3 pageEncoding，contentTypeの設定（SearchResult.jsp 1行目から5行目）

```
<%@ page language="java"
         import="java.util.*"
         pageEncoding="Shift_JIS"
         contentType="text/html; charset=Shift_JIS"
%>
```

リスト4 展開・表示を行うためのオブジェクトの使用宣言（SearchResult.jsp 6行目）

```
<jsp:useBean id="hitResults" scope="request" class="java.util.Vector" />
```

リスト5 例外メッセージ格納するオブジェクトの使用宣言（SearchResult.jsp 7行目）

```
<jsp:useBean id="message" scope="request" class="java.lang.String" />
```

リスト6 ログイン名の表示（SearchResult.jsp 21行目）

```
<h2>ようこそ，<%= request.getAttribute( "accId" )%>さん．</h2>
```

リスト7 権限情報による表示の切り替え（SearchResult.jsp 29行目）

```
<% if ( level != null && level.intValue() >= 2 ) { %>
```

- JSPに処理を委譲する

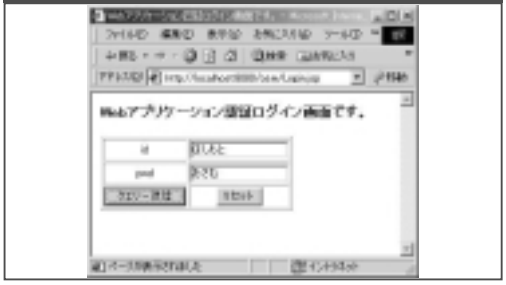
JSPで行う処理

- pageEncoding，contentTypeの設定により，正しいエンコーディングを指定する（リスト3）
- サーブレットより送信されたオブジェクトを展開・表示するために取得できるようにする（リスト4）
- サーブレット内で起きた例外のメッセージを取得できるようにする（リスト5）
- ログイン名（ID）を表示する（リスト6）
- ログイン認証の際に取得している権限情報により，表示を切り替える．一般ユーザには年齢・性別・メモだけを見せるようにして，プライベートなデータは一部の，権限を持ったユーザのみが見られるようにする（リスト7）

これらの処理を，作成するWebアプリケーションに盛り込みます．

なお，今回のサンプルでは，ログインIDとパスワードに，日本語が使用できるようにしました（図5）．Webアプリケーションのユーザアカウントには英数字を用いるのが一般的ですが，ここでは，Servlet API 2.3から追加されたフィルタ機能を紹介する目的で，あえて日本語を使っています．なお，Webアプリケーションのログイン機能については，本特集第3章「ロール別認証とシングルサインオン」でも解説していますのであわせてご参照ください．また，パスワード（図3の“おさむ”）を伏せ字にしたい際には，かなや漢字は使えなくなりますが，パスワードに指定されている<input>オブジェクトのtype属性に"password"を指定してください．

◆図5 ログイン画面



今回，ログイン画面の認証機能を受け持つコンポーネントクラスは，ブラウザからの入力パスワードを元にデータベース検索を行うものです．こちらのソースについて詳しい説明はしませんが，ここまで説明してきたことで理解できる内容になっています．このコンポーネントクラスは検索キーであるIDとパスワードをパラメータとして受け取り，データ閲覧の範囲指定を行う情報を取得します．これはLogin.jspとLoginBean.javaとして定義していますが，JSPからコンポーネントを利用する典型的な例と言えると思います．

サーブレットの実装

上記をふまえてサーブレットの実装を行うポイントとしては，以下の点が挙げられます．

文字化けを回避する

サーバサイドJavaを扱うときに避けて通れない問題を回避するために，次のような設定も行いましょう（具体的には環境を作る際に設定を行うことになります）．JSPでは暗黙に宣言されており，変数名requestとして使用可能なオブジェクトであるjavax.servlet.http.HttpServletRequestのgetParamter(String)メソッドでは，2バイトキャラクタが文字化けを起こす場合があります．このような現象への対処として，Servlet API

2.3から新たに採用されたフィルタ機能を使用することで，エンコーディングの設定を行うSetCharacterEncodingFilterクラスを使用して文字化けを回避することが可能です．これらについては，167ページのコラムを参照してください．

サーブレットの初期処理でのJDBCドライバの参照

Tomcatが起動されると，前述のように最初のアクセスがあったタイミングでサーブレットの初期化メソッドinit( ) がコールされます．サーブレットは一度ロードされるとサーブレットコンテナの終了もしくは自身の持っているメソッドdestroy( ) がコールされるまでコンテナによってメモリ上に常駐します．このinit( )メソッドにはサーブレットのライフサイクルの中でずっと継続されるべき処理を書きます．ここでは，JDBCドライバをロードすることにします（リスト8）．

サーブレットがコールされた際の処理を行う

ログイン画面からコールされた際の処理をdoPost( ) メソッドに記述します．ここで行うことを順番に説明していきます．

まず，セキュリティ保護上，このサーブレットへのアクセスが，URL（http://localhost:8080/tomcat_jk/servlet/tomcat_jk.servlet.SearchServlet）を直接キックしたものではなく，認証画面からログインしたものであることを判定するために，メソッドに対するパラメータであるHttpServletRequestオブジェクトからHttpSessionオブジェクトを取り出し，isNew( )メソッドを使ってこのオブジェクトが生成されたばかりのものか否かを判定します（リスト9）．もし，このURLが直接キックされた場合には，HttpSessionは生成されたばかりのステータスなのでboolean型のtrueが返ってきます．その際には画面を認証画面へいったん戻します．

認証を正常に終了させたユーザがログインしてきた際は，まず，ログインユーザの名前と権限情報を取得します．これは遷移先のJSPで表示する内容として，データベース取得データのどこまでを閲覧範囲とするかを判定するために使います．

リスト8　JDBCドライバのロード（SearchServlet.java）46行目・init( )における処理

```
Class.forName( "org.postgresql.Driver" );
```

リスト9　サーブレットがコールされた際の処理（SearchServlet.java　64行目・doPost( )における処理）

```
HttpSession session = request.getSession();
if ( session.isNew() || request.getParameter( "id" ) == null ) {
    try {
        response.sendRedirect( "/tomcat_jk/Login.jsp" );
        return;
    } catch ( IOException ex ) {
        ex.printStackTrace();
        this.msg = ex.getMessage();
    }
}
```

リスト10　DBへの接続（SearchServlet.java）83行目・doPost( )における処理

```
conn = DriverManager.getConnection( dbUrl, dbId, dbPwd );
```

データベースとの連携

今回のアプリケーションでは，サーブレットはデータベース接続を行います．JDBCドライバを管理するオブジェクト（java.sql.）DriverManagerのパラメータとして，アカウントに対するユーザ名・パスワード・URLを指定して．データベースへの接続オブジェクト（java.sql.）Connectionを開きます（リスト10）．ユーザ名とパスワードはJDBC接続を行う上で不可欠な情報ですが，本章で使用するPostgreSQL Windows-Native版のJDBCドライバによる接続では，Windowsのログオンアカウントとダブらない限り，適当なものを設定すれば良いようです．PostgreSQLのURLの書式ではテーブル名まで含んだものとなっていますので注意してください．これについては本特集コラム「サンプルWebアプリケーションの導入方法」で説明します．

データベース接続と処理

データベース接続がうまくいったら，今度は取得済みのユーザアカウント情報をキーにしてSQL文を発行します（リスト11）．基本的な事柄ですが，Javaにおいてはコマンドは（java.sql.）Connectionオブジェクトより（java.sql.）Statementを生成し，Statementオブジェクトの持っている実行メソッドの引数にSQL文を指定して発行します．

正常処理として，（java.sql.）ResultSet（SQLアクセス発行による結果セット）が取得できたら，このオブジ

ェクトの内容を（java.util.）Vectorと（java.util.）Hashtableにデータの構造化を行う意味で格納しましょう．データベースからのフィールドの数と名前は明らかになっているので，Hashtableに1レコードを格納し，取得できた分だけVectorに追加してやることにします．

ResultSetオブジェクトは明示的にclose( )メソッドを発行して，取得結果を閉じましょう．これを怠ると，JDBCリソースの開放が行われず，一方データベースの側では接続を完了していないものだと判定されて，マシン間の認識の違いが発生する可能性がありますので，注意してください．

ここまでが済んだら，JSPへ処理を渡すための準備をします．そのためにはまず，データベースから取得した文字列などが文字化けしないよう，これから送信を行うオブジェクトに対してcontentType属性を設定します．Windows環境では代表的なエンコーディングはShift_JISであり，特殊な設定を行わない限り，Shift_JISにてソースの記述を行います．それに対してLinuxなどのUNIX環境での代表的なエンコーディングはEUCです．今回はWindows環境を前提にしていますのでShift_JISを使用します．このSearchServlet.javaでは，応答オブジェクトのcontentType属性に対して("text/html; charset=Shift_JIS")を設定します（SearchServlet.java 95行目）．これを行わないとJSPが，送信してきた側がどのcontentTypeで送信を行ってきたのか判定できず，表示文字列が"????"などと表示されてしまう場合があります．注意してください．

そして，送信するオブジェクトをsetAttibute(String, Object)を使用して，doPost( )の引数であるHttpServletRequestに対して設定します．ここまでが済めば，JSPに遷移する準備は完了です．JSPはサーブレットにて取得された情報を待ち構えています．

リスト11　DB検索と取得結果の構造化を行いオブジェクトへのバインドをする処理（SearchServlet.java）

```
public void doPost( HttpServletRequest request,
                HttpServletResponse response ) {
    // (snip)
    String sqlStatement = "select * from customer";
    Vector searchResult = new Vector();
    searchResult = searchDatabase( sqlStatement );
    // (snip)
    response.setContentType( "text/html; charset=Shift_JIS" );
    request.setAttribute( "hitResults", searchResult );
    request.setAttribute( "message", this.msg );
    // (snip)
}

private Vector
        searchDatabase( String sqlStatement ) {
    Vector vResult = new Vector();
    ResultSet rResult = null;
    try {
        Statement stmt =
            conn.createStatement();
        rResult =
            stmt.executeQuery( sqlStatement );
        vResult = setResult( rResult );
        rResult.close();
    } catch ( SQLException ex ) {
        ex.printStackTrace();
        this.msg = ex.getMessage();
    }
    rResult = null;
    return vResult;
}

private Vector setResult( ResultSet rResult ) {
    Vector vResult = new Vector();
    try {
        while( rResult.next() ) {
            try {
                Hashtable htRow = new Hashtable();
                htRow.put("level", new Integer(
                        rResult.getInt( "Account_Level")));
                htRow.put("sex", rResult.getString("sex"));
                // (snip)
                htRow.put("memo",rResult.getString("memo"));
                vResult.addElement( htRow );
            } catch ( NullPointerException ex ) {
                ex.printStackTrace();
                this.msg = ex.getMessage();
            }
        }
    } catch ( SQLException ex ) {
        ex.printStackTrace();
        this.msg = ex.getMessage();
    }
    return vResult;
}
```

リスト12　文字セットの設定（SearchResult.jsp　1行目から5行目）

```
<%@ page language="java"
         import="java.util.*"
         pageEncoding="Shift_JIS"
         contentType="text/html; charset=Shift_JIS"
%>
```

JSPの実装

JSPファイルの冒頭で行う指定

JSPではまず，

- 正しい文字エンコーディングを行うためのpage EncodingとcontentTypeのなどのPage Directiveの設定
- 遷移元であるサーブレットから必要な情報を取り出

すためにuseBeanの宣言

を行います．

まずは日本語での2バイトキャラクタのための文字エンコーディングの設定です．Tomcat 4.xから使用可能になったpageEncoding属性には，記述を行うJSPページに対する文字エンコーディングの指定を行います．具体的には，今回はWindows環境を前提にしていますのでShift_JISを使用します．そして以前からおなじみの，JSPレスポンスのための設定であるcontentType属性には，"text/html;charset=Shift_JIS"を設定しましょう（**リスト12**）．

文字コードの問題に対しては，Servlet API 2.3対応のTomcat 4.xで導入されたフィルタ機能を利用するSetCharctorEncodingクラスを環境設定段階で組み込むことによって，以前までのサーバサイドJavaプログラミングにおける悩みであった文字化けへの対処が可能です．これに関してはコラム「フィルタ機能で実現できること」（167p.）を参照してください．

JSPファイルの冒頭におけるその他の宣言文

JSPページの冒頭では，JSPページ，HTMLファイルに対する文字コードセットの設定や，使用できるJavaライブラリのパッケージのimport宣言をpageディレクティブと呼ばれるエリアで行います．そして遷移元から送られてくる情報である，Beanに格納された値を参照するために，前述のようにuseBean宣言を行います．サーブレット側でsetAttribute( )メソッドによって参照する際の名称が設定されたものを，useBeanでは，id="設定された名称"という形で設定を行うことにより参照が可能となりますので，ここではサーブレットからのデータベース取得データ情報と例外が発生した際のメッセージを取得できるようにしておきます．ということで2つのBeanを使用することを宣言します．

JSPのuseBeanタグで使用するBeanのスコープ（適用範囲）は範囲の順番から並べると，page，request，session，applicationの4つがあります．最適なスコープを選択しましょう．一般に，スコープは広すぎるとWebアプリケーションの負荷の問題につながりやすく「遅いサイトだ」などという悪評を得られかねません．

"page"はクライアントからのリクエストを受け取ったJSPから返す際に使います．今回はユーザのリクエスト→サーブレット→JSPと遷移される中から受け渡される情報を受け取りますので，もう少し広いスコープのほうがよいでしょう．"request"は，同一のリクエストを処理するページからアクセス可能で，暗黙オブジェクトであるrequestからの情報を使用ができ，これらを受け止めることができます．"session"はクライアントからのセッションが何らかの理由においても破棄されず有効な限り，アクセス可能なスコープです．さらに"appllication"は同一のアプリケーション内にあるリクエストを処理するページから参照が可能で，ディレクティブの設定によるなどの事情にてJSPページのsessionが有効になっていない場合にもアクセスすることができます．今回のアプリケーションで最適なスコープとしてはrequestが良さそうです（**リスト13**）．

リスト13　スコープの指定（SearchResult.jsp 6行目と7行目）

```
<jsp:useBean id="hitResults" scope="request" class="java.util.Vector" />
<jsp:useBean id="message"  scope="request" class="java.lang.String" />
```

リスト14　送信フォームの内容が2バイトなので文字化けを起こす例

```
request.getParameter( "paramName" )
```

以下のように，いったんバイト文字列にしてエンコーディングの指定を行う必要があった

```
new String( request.getParameter( "paramName" ).getBytes( "8859_1" ), "Shift_JIS" )
```

アカウント情報の処理

今回のアプリケーションでは，JSP側ではサーブレットから送られてきたアカウント情報を，

リスト15　表示／一部非表示の判定と表示される内容（SearchResult.jsp）

```
        <tr nowrap>
            <th nowrap>性別</th>
            <th nowrap>年齢</th>
            <th nowrap>職業</th>
<%      if ( level != null && level.intValue() >= 2 ) { %>
            <th nowrap>名前（ローマ字）</th>
            <th nowrap>名前（漢字）</th>
            <th nowrap>会社名</th>
            <th nowrap>部署</th>
            <th nowrap>役職</th>
            <th nowrap>メールアドレス</th>
<% } %>
            <th nowrap>都道府県</th>
<%      if ( level != null && level.intValue() >= 2 ) { %>
            <th>住所</th nowrap>
            <th nowrap>電話番号</th>
            <th nowrap>携帯番号</th>
<% } %>
            <th nowrap>メモ</th>
        </tr>
```

● ログイン名の表示
● データベース取得データの閲覧範囲の特定

の2つに使用します.

ログイン名の表示

まず，前述のように日本語によるログイン名を表示します.

サーブレット/JSPの文字コードで，HTMLフォームオブジェクトから送信された文字列が漢字やひらがななどのなどの2バイト文字の場合，しばしば，「入力された文字列」と出力するつもりが「?????????」のようになってしまう文字化けがおきるのが常でした．これを防ぐために文字列はバイト文字列に解析を行い，エンコーディングの指定をして変換を行うというやりかた（**リスト14**）でしたが，前述の，Servlet API 2.3で取り入れられたフィルタ機能を用いると，このような文字列変換のためのコーディングを意識せずに2バイト文字列の出力が行えます．今回はServlet API 2.3準拠のTomcat 4.0.4を使用しているので，コーディングの前段階の環境設定を一度行えば，それ以降に文字コードの意識をする必要はありません.

リスト16　データベースの値の取り出し（SearchResult.jsp）

```
<% for (int i =0;i <hitResults.size();i++) { %>
    <tr>
    <td nowrap align=center>
        <%=((Hashtable)hitResults.get(i)).get("sex")%>
    </td>
<%
    if (level !=null &&level.intValue()>=2 ) {
%>
        <td nowrap align=center>
            <%=((Hashtable)hitResults.get(i)).get("name")%>
        </td>
        <td nowrap align=center>
            <%=((Hashtable )hitResults.get(i)).get("name_jp")%>
        </td>
    <%
        if (level !=null &&level.intValue()>=2 ) {
    %>
            <td nowrap align=center>
                <%=((Hashtable)
                        hitResults.get(i)).get("address")%>
            </td>
        <% } %>
        <td nowrap>
        <%=((Hashtable )hitResults.get(i)).get("memo")%>
        </td>
        </tr>
    <% } %>
<% } %>
```

JSPからBeanにアクセスしてデータを取り出し，表示する

データベースから取り出したい情報，見せたい情報についてはアイテム格納済みのBeanから取り出します．格納された値は格納時の形そのままで入っていますので，これを取り出す処理を行います．ここでは，最初の要件である，表示／一部非表示のデータ判定を行うため，アカウントの持っている権限情報を参照します（**リスト15**）.

<%と%>で囲まれたスクリプトレットでは，まずパラメータを取り出し，内容によって条件判定をして「何を表示するのか」を決めています．表示／一部非表示は，サーブレットから得たパラメータによって切り替えています.

以降，JSPソースの続く部分，スクリプトレットに囲まれている部分では，データベースの値を取り出しています．これは，サーブレット側でVectorオブジェクト内部にキーを設定したHashtableを持つという形でフォーマットを規定しているので，オブジェクトに格納されて渡されてきたビューを，渡されてきたレコード数分だけ表現することができます（**リスト16**）.

以上で，認証で得たパラメータであるログイン名とユーザアカウントレベルによる閲覧範囲の限定を行った表示を実現できます.

おわりに

以上，サーブレット，JSPの機能の基礎知識，それらを利用したシンプルなWebアプリケーションの作成と，コラムでは，Tomcat 4.xで新たに取り入れられたフィルタ機能について解説してきました．本稿が読者の皆さんのTomcatへの理解を得るきっかけとなり，また少しでも興味を持っていただけたら幸いです．■

COLUMN

フィルタ機能で実現できること

フリーライター & エンジニア●橋本 修一 *Osamu Hashimoto*
osm@yha.att.ne.jp

●フィルタ機能の活用

Servlet API 2.3から新たに，フィルタ機能が追加されました．フィルタ機能を用いると，サーブレットの実行前後に，ある特定の処理を実行することができるようになります．たとえば，

- 入出力文字コードの変換
- 送信データの圧縮
- 携帯端末からのアクセスには画像のフォーマットを変えてサイズ小さくする

などの処理が可能になります．フィルタを用いると，サーブレットによるロジックの実行とは切り離して入出力の処理を行えるので，モジュール性が向上します．ここでは，ブラウザから受け取ったリクエストの文字列の文字コードを変換するSetCharacterEncodingFilterを例にとり，フィルタの機能についてご説明します．

●SetCharctorEncodingクラスの使用とWebアプリケーションの配置

Javaでは，内部的な文字コードにUnicodeを使用しています．このため，前述のようにサーバサイドJavaのプログラマは，サーブレットやJSPの文字出力を行う際には入出力形式を最初から想定し，2バイトの文字コードに常に意識をしなければならず，本文でも触れているようにHTMLフォームから送信された文字列を出力する際などは，文字化け問題への策を講じる必要がありました．

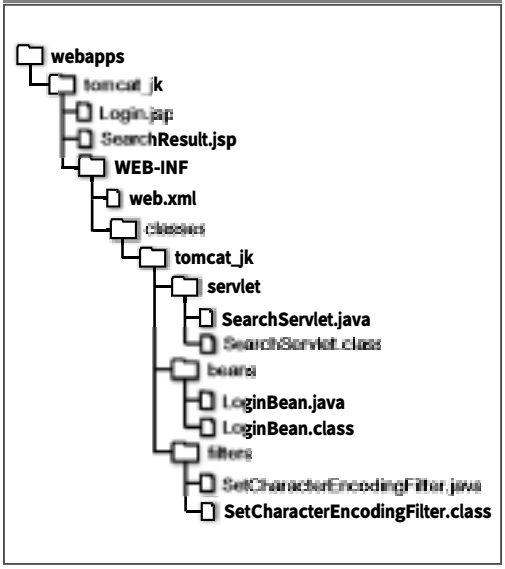
たとえば，JSPでは暗黙オブジェクトであるrequestのgetParameter(String)メソッドを使用して，HTMLフォームオブジェクトから送信されたHTTPパラメータである2バイト文字列を取得，出力する場合，文字コードに対する配慮を忘れると"???"といういわゆる「文字化け」がおこるため，正規の取得と出力のためには文字列を規定のフォーマットにするためにバイト列変換を行い，正しいエンコーディングを指定するといった処理を行う必要がありました．

Servlet API 2.3で導入され，Tomcatではバージョン4.xから標準で使用可能となっているフィルタ機能を使用すると，この文字化けの問題を解消することができます．具体的には，Tomcat 4.0.xに梱包されており，Filterの実用例として提供されている，フィルタ機能を扱うSetCharacterEncodingFilterクラスを使用することで，2バイト文字に対するエンコーディングへの対処の問題が大幅に改善されます．

Tomcatでは，フィルタ機能を使用したサンプルは以下の位置に配置されています（%CATALINA_HOMEはTomcatをインストールしたC:\Program Files\Apache Tomcat4.0などのディレクトリを指します）．

```
%CATALINA_HOME%\webapps\examples\WEB-INF\classes\filters
```

図a 今回作成したWebアプリケーションとfiltersフォルダの配置



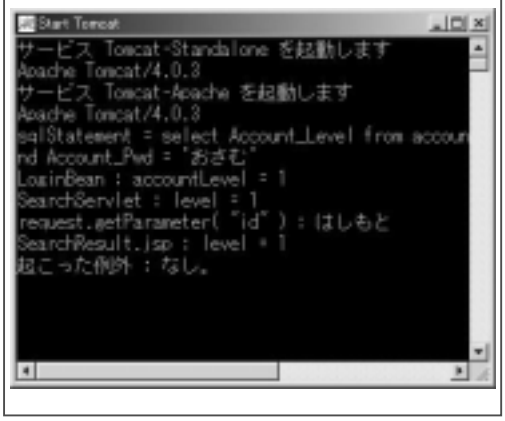
リストa SetCharcterEncodingを使用するためのweb.xmlの記述（文字コードをShift_JISに設定する記述を抜粋）

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app
    PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
    "http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>
    <!-- Example filter to set character encoding on each request -->
    <filter>
        <filter-name>Set Character Encoding</filter-name>
        <filter-class>filters.SetCharacterEncodingFilter</filter-class>
        <init-param>
            <param-name>encoding</param-name>
            <param-value>Shift_JIS</param-value>
        </init-param>
    </filter>
    <filter-mapping>
        <filter-name>Set Character Encoding</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>
</web-app>
```

リストb IDの取り出し（SearchServlet.java 79行目）

```
System.out.println( "request.getParameter( \"id\" ) : " + request.getParameter( "id" ) );
```

◆図b Tomcatコンソールへの出力



今回作成したWebアプリケーションでは，**図a**の位置にこのフォルダを配置します．

●web.xmlの記述

%CATALINA_HOME/webapps/examples/WEB-INF/web.xmlには**リストa**のようなSetCharcter Encodingを使用するための記述があります．

この記述では，filter要素とfilter-mappingの要素を指定しています．今回のサンプルアプリケーションで使用しているのはShift_JISなので，エンコーディングには上記のとおりShift-JISを指定します．対象とするURLはリストaでは「すべて」となっています．

今回作成したサンプルアプリケーションでは，SearchServlet.javaというプログラムの78行目で，フィルタ機能を使ってコーディングをしています．

```
String accId  = request.getParameter( "id" );
```

ここで取得されるのは遷移元のHTMLフォームから送信された文字列ですが，SetCharacter EncodingFilterによって2バイト文字列の文字化け問題が回避されています．日本語（2バイト）のアカウント名で，認証のLogin.jspからログイン名とパスワードを入力すると，直接値を取り出した時点で文字化けせずに，きちんと取得できることがわかります（**リストb**，**図b**）．

などと出力されます．

フィルタ機能を利用することで，このように2バイト文字の文字化けを防いで，開発の効率も大幅に上げることが可能です．■