Mục lục

SERVLET

javax.servlet.Servlet

Ta có thể tạo một servlet bằng cách **implements** từ interface **javax.servlet.Servlet**. Ví dụ:

```
package com.test.servlet;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;

public class Servlet1 implements javax.servlet.Servlet {

	public void destroy() {
	}

	public ServletConfig getServletConfig() {
		return null;
	}

	public String getServletInfo() {
		return null;
	}

	public void init(ServletConfig config) throws ServletException {
	}

	public void service(ServletRequest req, ServletResponse res)
			throws ServletException, IOException {
		PrintWriter out = res.getWriter();
		out.println("Day la servlet implements tu
	javax.servlet.Servlet");
	}
}
```

Để có thể chạy một servlet ta cần các bước sau:

1) Tạo file Servlet1.java với nội dung như trên và đặt trong package `com.test.servlet`

2) Khai báo trong web.xml

```
<servlet>
  <servlet-name>Serv1</servlet-name>
  <servlet-class>com.test.servlet.Servlet1</servlet-class>
</servlet>
```

```
<servlet-mapping>
  <servlet-name>Serv1</servlet-name>
  <url-pattern>/Serv1</url-pattern>
</servlet-mapping>
```

3) Chạy ứng dụng trên browser: http://localhost:8080/Demo/Serv1

GenericServlet:

Bằng cách triển khai servlet như trên thì mọi việc đều ổn nhưng có 2 vấn đề khó chịu là:

1) Như ta thấy ở file Servlet1 ở VD trên luôn có 5 phương thức là

```
public void destroy()
public ServletConfig getServletConfig()
public String getServletInfo()
public void init(ServletConfig config)
public void service(ServletRequest req, ServletResponse res)
```

của interface **javax.servlet.Servlet** bắt buộc phải được triển khai mặc dù phần lớn ta chỉ sử dụng 1 phương thức (các phương thức khác để trống – ko làm gì cả).

2) Nếu ta muốn lưu giữ giá trị của ServletConfig thì ta phải khai báo và lưu giá trị của nó thong qua phương thức init() như bên dưới. Điều này không khó nhưng trong trường hợp này lại thêm một bước phải thực hiện.

```
ServletConfig servletConfig;

public void init(ServletConfig config) throws ServletException {
      servletConfig = config;
}
```

Điều này được giải quyết bằng cách kế thừa từ GenericServlet. Generic class đã thực thi tất cả các phương thức, hầu hết là để trống. ServletConfig được thực hiện bằng cách gọi ***getServletConfig()*** trong ***service();***. Và giờ đây khi ta tạo servlet kế thừa từ lớp này sẽ trông đơn giản và sáng sủa hơn nhiều:

```
package com.test.servlet;

import javax.servlet.*;
import java.io.IOException;
import java.io.PrintWriter;

public class Servlet1 extends javax.servlet.GenericServlet {

  public void service(ServletRequest request, ServletResponse response)
```

```
        throws ServletException, IOException {
            PrintWriter out = response.getWriter();
            out.println("Day la servlet extends tu
javax.servlet.GenericServlet");
    }
}
```

HttpServlet

Được extends (kế thừa) từ javax.servlet.GenericServlet và thêm vào một số các phương thức. Quan trọng nhất là 6 method doxxx được gọi khi Http request được sử dụng đó là: doPost(), **doGet()**, doTrace(), doOption(), **doDelete()**. Trong bài này ta chỉ chú trọng vào doGet() và doPost();

```
package com.test.servlet;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class Servlet2 extends javax.servlet.http.HttpServlet {
      public void doGet(HttpServletRequest request, HttpServletResponse
response)
                  throws ServletException, IOException {
            PrintWriter out = response.getWriter();
            out.println("Day la servlet extends tu
javax.servlet.http.HttpServlet");
      }
}
```

doPost() được gọi khi method được đặt trong form là POST và doGet khi method = GET. Mặc định doGet() sẽ được gọi. VD: <FORM METHOD=POST>

ServletRequest interface:

ServletRequest interface cung cấp các phương thức quan trọng cho phép bạn truy cập thong tin về user. Ví dụ, phương thức getParameterNames trả về kiểu Enumeration chứa tập hợp các tên parameter của request hiện thời. Để lấy thông tin của các request này ta sử dụng phương thức getParameter.

VD1: in ra danh sách tất cả các Parameter của request hiện thời

```
Enumeration parameters = request.getParameterNames();
while (parameters.hasMoreElements()) {
    String parameterName = (String) parameters.nextElement();
    System.out.println("Parameter Name: " + parameterName);
```

```
        System.out.println("Parameter Value: " +
                           request.getParameter(parameterName));
    }
```

Hoặc ta có thể lấy địa chỉ IP của user = phương thức getRemoteAddress, tên máy = phương thức getRemoteHost. Ta có thể xem và chạy thử tất cả các phương thức của ServletRequest trong bảng danh mục các phương thức của ServletRequest ở cuối tài liệu này:

VD2: Lấy địa chỉ máy của user

```
out.println("UserRemoteHost:" + request.getRemoteHost());
```

ServletResponse interface

Phương thức quan trọng nhất là java.io.PrintWriter, với đối tượng này bạn có thể in ra các thẻ HTML hoặc dữ liệu dạng text lên trình duyệt.

VD:

```
PrintWriter out = response.getWriter();
out.println("<HTML>");
.....
out.println("</HTML>");
```

HttpServletRequest interface:

Interface này được kế thừa từ javax.servlet.ServletRequest để cung cấp các thông tin về request cho HTTP servlet. HttpServletRequest được thêm vào một số các phương thức mà ServletRequest ko có phục vụ cho HttpServlet.

VD: getAuthType(): trả về kiểu xác thực để bảo vệ servlet. Kết quả trả về có thể là "BASIC", "SSL" hoặc NULL nếu servlet ko được bảo vệ.

getContextPath(): trả về 1 phần URL của context của request (VD: /ServetDemo).

Các phương thức HTTP based khác là: getCookies(), getHeaderxxx()….

Tất cả các phương thức của interface này được liệt kê trong phần phụ lục.

HttpServletResponse interface:

Kế thừa từ ServletResponse interface để cung cấp các chức năng đặc tả của HTTP servlet trong việc gửi response. Ví đụ các phương thức truy cập HTTP headers và cookies.

Tất cả các phương thức của interface này được liệt kê trong phần phụ lục.

ServletConfig interface

Được sử dụng bởi servlet container để truyền thông tin tới servlet trong khi khởi tạo.

VD: Sử dụng ServletContext để đọc các tham số khởi tạo của servlet.

1) Các tham số này được khai báo trong web.xml như sau:

```
<web-app>
      .…….
       <servlet>
            <servlet-name>Serv1</servlet-name>
            <servlet-class>com.test.servlet.Servlet1</servlet-class>
            <init-param>
                  <param-name>adminEmail</param-name>
                  <param-value>admin@aptech.com</param-value>
            </init-param>
            <init-param>
                  <param-name>adminContactNumber</param-name>
                  <param-value>04298371237</param-value>
            </init-param>
      </servlet>
      ……
</web-app>
```

2) Đọc các tham số:

```
package com.test.servlet;

import java.io.IOException;
import java.io.PrintWriter;
import java.util.Enumeration;
import javax.servlet.ServletConfig;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class Servlet2 extends javax.servlet.http.HttpServlet {
   public void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
         PrintWriter out = response.getWriter();
```

```
                ServletConfig sc = getServletConfig();
                Enumeration parameters = sc.getInitParameterNames();
                int count = 0;
                while (parameters.hasMoreElements()) {
                      count++;
                      String parameter = (String) parameters.nextElement();
                      out.println("Para name: " + parameter +" value: " +
          sc.getInitParameter(parameter));
                }
                if (count == 0)
                      out.println("Ko co parameter nao duoc cai dat trong
          web.xml cua servlet: " + sc.getServletName());
             }
          }
```

Tất cả các phương thức của interface này được liệt kê trong phần phụ lục.

ServletContext

Định nghĩa tập hợp các phương thức mà một servlet sử dụng để giao tiếp với Servlet Container.

VD: setAttribute cho phép gọi thuộc tính này ở tất cả các servlet trong cùng một context

```
package com.test.servlet;

import javax.servlet.*;
import javax.servlet.http.HttpServlet;
import java.util.Enumeration;
import java.io.IOException;
import java.io.PrintWriter;

public class Servlet1 extends HttpServlet {
      public void service(ServletRequest request, ServletResponse response)
                  throws ServletException, IOException {
            PrintWriter out = response.getWriter();
            ServletContext servletContext =
                              getServletConfig().getServletContext();
            Enumeration attributes = servletContext.getAttributeNames();
            while (attributes.hasMoreElements()) {
                  String attribute = (String) attributes.nextElement();
                  out.println("AttributeName: " + attribute);
                  out.println("AttributeValue: "+
                              servletContext.getAttribute(attribute));
            }
            out.println("MajorVersion: "+ servletContext.getMajorVersion());
            out.println("MinorVersion: "+ servletContext.getMinorVersion());
            out.println("Server info : " + servletContext.getServerInfo());
      }
}
```

Tất cả các phương thức của interface này được liệt kê trong phần phụ lục.

Một ví dụ về mã lệnh lắng nghe sự kiện (listener)

Mỗi khi một sự kiện gì đó xảy ra thi các phương thức tương ứng sẽ được gọi

```
package com.yourcompany.listener;

import javax.servlet.ServletContextAttributeEvent;
import javax.servlet.ServletContextAttributeListener;

public class MyServletContextAttributeListener implements
      ServletContextAttributeListener {
public void attributeAdded(ServletContextAttributeEvent scab) {
      System.out.println("An attribute was added to the "
                  + "ServletContext object");
}

public void attributeRemoved(ServletContextAttributeEvent scab) {
      System.out.println("An attribute was removed from "
                  + "the ServletContext object");
}

public void attributeReplaced(ServletContextAttributeEvent scab) {
      System.out.println("An attribute was replaced in the "
                  + "ServletContext object");
}
}
```

Cấu hình trong web.xml

```
<listener>
  <listener-class>
          com.yourcompany.listener.MyServletContextAttributeListener
  </listener-class>
</listener>
```

RequestDispatcher

Gồm 2 phương thức là include (để lấy nội dung của một trang khác vào servlet hiện thời. Phương thức forward để chuyển đến 1 URL khác. Bạn có thể lấy RequestDispatcher theo 1 trong 3 cách sau:

- Sử dụng phương thức ***getRequestDispatcher*** của ***javax.servlet.ServletContext*** interface, tham số truyền vào là chuỗi chứa đường dẫn tới tài nguyên khác. Đường dẫn là đường dẫn tương đối với gốc của **ServletContext**.
- Sử dụng phương thức ***getRequestDispatcher*** của ***javax.servlet.ServletRequest*** interface, tham số truyền vào là chuỗi chứa đường dẫn tới tài nguyên khác. Đường dẫn là tương đối

với HTTP request hiện hành. Đường dẫn có thể là tương đối nhưng nó ko thể mở rộng ra ngoài servlet context hiện hành.

- Sử dụng phương thức ***getNamedDispatcher*** của ***javax.servlet.ServletContext*** interface, tham số truyền vào là chuỗi chứa tên của tài nguyên khác (VD: tên 1 servlet được định nghĩa trong web.xml).

Sự khác biệt giữa 2 phương thức này là javax.servlet.Context.getRequestDispatcher() có thể dùng đường dẫn tương đối.

```
request.getRequestDispatcher("/index.jsp").include(request, response);

request.getRequestDispatcher("/index.jsp").forward(request, response);
```

Error log và send error

```
log("Test log error", new Exception());

response.sendError(response.SC_BAD_GATEWAY);
```

Session Tracking – Lưu dấu session

Vì phương thức HTTP là stateless (ko lưu các thông tin lịch sử), điều này ảnh hưởng sâu sắc đến lập trình ứng dụng web. Có 4 kỹ thuật để lưu dấu session:

1) URL rewritting – Thêm các tham số vào cuối URL
2) Hidden fields – Sử dụng các trường ẩn
3) Cookies – Sử dụng cookie để trao đổi dữ liệu giữa client và server
4) Session *objects* – Sử dụng các đối tượng có phạm vi (scope) là session để truyền thông tin.

URL Rewritting:

VD: Http://localhost:8080/Demo/test?x=abc&y=xyz

Phần sau ?x=abc&y=xyz là phần được thêm vào để truyền 2 parameter x và y

Để lấy giá trị này ta dùng lệnh **request.getParameter("x")**, tương tự với y.

Hidden fields:

<INPUT TYPE=HIDDEN Name=hField VALUE="abc”>

Để lấy giá trị này ta dùng lệnh **request.getParameter(“hField”)**.

Cookies:

Cookie được lưu bởi server và gửi về client cùng response. Request được gửi tới server cùng với cookie nhưng ko thay đổi giá trị của cookie. Giá trị của cookie được lưu trong bộ nhớ (ổ cứng) của client.

VD1: lưu cookie (sử dụng response)

```
Cookie c1 = new Cookie("userName", "Helen");
Cookie c2 = new Cookie("password", "Keppler");
c2.setMaxAge(300);
response.addCookie(c1);
response.addCookie(c2);
```

Cookie được chia làm 2 loại là cookie bị xóa ngay sau khi đóng trình duyệt. Loại thứ 2 gọi là persisting cookies, loại là c2 trong ví dụ trên. Ta sử dụng setMaxage(int) để đặt tuổi theo giây cho cookie.

VD2: Đọc giá trị của cookies:

```
Cookie[] cookies = request.getCookies();
for (int i = 0; i < cookies.length; i++) {
   Cookie cookie = cookies[i];
   out.println("Name->" + cookie.getName() + " Value->" + cookie.getValue());
}
```

Session objects:

```
HttpSession session = request.getSession(true);
session.setAttribute("loggedIn", new String("true"));
```

Vì có phạm vi (scope) là sesion nên giá trị này có thể được đọc bởi các servlet (hoặc JSP) khác.

Filter và Servlet communication

Filter là một bổ xung mới cho Servlet 2.3, cho phép bạn chắn 1 request trước khi nó tới các tài nguyên. Nói cách khác, filter cho bạn quyền truy cập đến các

HttpServletRequest và HttpServletResponse trước khi chúng được chuyển đến 1 servlet.

Filter có thể rất đắc dụng. Ví dụ, bạn có thể viết 1 filter để lưu tất cả các request được gửi đến và log địa chỉ IP của các máy gửi request. Bạn cũng có thể sử dụng filter như một phương tiện để mã hóa và giải mã. Cũng có thể dùng để xác thực user, nén dữ liệu, xác thực dữ liệu..vv...

Khi viết 1 filter, về cơ bản bạn sẽ làm việc với 3 interface trong package javax.servlet:

1) Filter
2) FilterConfig
3) Filter Chain

Filter interface

Vòng đời (life circle của 1 filter gồm 3 method):

1) public void init(FilterConfig filterConfig)
2) public void doFilter(HttpServletRequest request, HttpServletResponse response, FilterChain chain)
3) public void destroy()

Ví dụ1: Trong ví dụ này sẽ demo vòng đời của filter:

TestFilter.java

```
import java.io.*;
import java.net.*;
import java.util.*;
import java.text.*;
import javax.servlet.*;
import javax.servlet.http.*;

import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;

public class TestFilter implements Filter {
```

```
    private FilterConfig filterConfig = null;

    public TestFilter() {
    }

    public void doFilter(ServletRequest request, ServletResponse response,
            FilterChain chain)
            throws IOException, ServletException {
        System.out.printlnThis is doFilter");
        //In ra 2 parameter của filter
        System.out.println("Init p1:" + filterConfig.getInitParameter("p1"));
        System.out.println("Init p2:" + filterConfig.getInitParameter("p2"));
        //Lấy giá trị của parameter para1 để sửa và lưu vào attribute att1
        String para1 = request.getParameter("para1");
        request.setAttribute("att1", para1 + "-xxx|");
        chain.doFilter(request, response);
    }

    public void setFilterConfig(FilterConfig filterConfig) {
        this.filterConfig = filterConfig;
    }

    public void destroy() {
        System.out.println("This is destroy");
    }

    public void init(FilterConfig filterConfig) {

        System.out.println("TestFilter: Initializing filter");
    }
}
```

Web.xml

Có 2 cách thiết lập mapping filter với servlet

1) Theo tên servlet (*không có dấu "/"*)

```
<filter-mapping>
      <filter-name>TestFilter</filter-name>
      <servlet-name>FilteredServlet</servlet-name>
</filter-mapping>
```

2) Theo URL pattern:

```
<filter-mapping>
      <filter-name>TestFilter</filter-name>
      <url-pattern>/FilteredServlet</url-pattern>
</filter-mapping>
```

Dưới đây là khai báo (theo URL) trong web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
      <filter>
            <filter-name>TestFilter</filter-name>
            <filter-class>com.aptech.filter.TestFilter</filter-class>
            <init-param>
                  <param-name>p1</param-name>
                  <param-value>v1</param-value>
            </init-param>
            <init-param>
                  <param-name>p2</param-name>
                  <param-value>v2</param-value>
            </init-param>
      </filter>
      <filter-mapping>
            <filter-name>TestFilter</filter-name>
            <servlet-name>/FilteredServlet</servlet-name>
      </filter-mapping>

      <servlet>
            <servlet-name>FilteredServlet</servlet-name>
            <servlet-class>
                  com.aptech.servlet.FilteredServlet
            </servlet-class>
      </servlet>
      <servlet-mapping>
            <servlet-name>FilteredServlet</servlet-name>
            <url-pattern>/FilteredServlet</url-pattern>
      </servlet-mapping>
</web-app>
```

Servlet

```
package com.aptech.servlet;

import java.io.*;
import java.net.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class FilteredServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse
response)
    throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        out.println("Before:'" + request.getParameter("para1") +"'<BR>");
        out.println("After:'" + request.getAttribute("att1") +"'<BR>");
    }
}
```

FilterConfig interface

Bao gồm các phương thức:

1) getFilterName: Trả về tên của filter
2) getInitParameter(String): trả về giá trị của param tương ứng của para-name.

```
<init-param>
    <param-name>p1</param-name>
    <param-value>v1</param-value>
</init-param>
```

3) getInitParameterNames(): trả về tập giá trị của tên tham số khởi tạo của filter. Dữ liệu trả về là Enumeration của String.
4) getServletContext(): trả về tham chiếu tới ServletContext của filter.

FilterChain interface

Có thể sử dụng nhiều hơn 1 filter. VD: 2 filter Filter và Filter1 áp dụng cho cùng 1 servlet FilteredServlet.

```
<filter-mapping>
      <filter-name>Filter</filter-name>
      <servlet-name>FilteredServlet</servlet-name>
</filter-mapping>

<filter-mapping>
      <filter-name>Filter1</filter-name>
      <servlet-name>FilteredServlet</servlet-name>
</filter-mapping>
```

Security Configuration

Web.xml

```
<security-constraint>
        <web-resource-collection>
            <web-resource-name>
                Restricted Area
            </web-resource-name>
            <url-pattern>/*</url-pattern>
        </web-resource-collection>
        <auth-constraint>
            <role-name>manager</role-name>
        </auth-constraint>
 </security-constraint>
```

BASIC

Web.xml

```
<login-config>
        <auth-method>BASIC</auth-method>
        <realm-name>User Basic Authentication</realm-name>
</login-config>
```

DIGEST

```
<login-config>
    <auth-method>DIGEST</auth-method>
</login-config>
```

FORM

Web.xml

```
<login-config>
      <auth-method>FORM</auth-method>
      <form-login-config>
            <form-login-page>/Login.jsp</form-login-page>
            <form-error-page>/Error.jsp</form-error-page>
      </form-login-config>
</login-config>
```

Login.jsp

- method của form phải là **POST**.
- Giá trị của ACTION attribute phải là "**j_security_check**".
- Form phải có 2 textbox là **j_username** và **j_password**.

```
<HTML>
      <HEAD>
            <TITLE>Login Page</TITLE>
      </HEAD>
      <BODY>
            <CENTER>
                  <H2>
                        Please enter your user name and password
                  </H2>
                  <FORM ACTION="j_security_check" METHOD="POST">
                        <TABLE>
                              <TR>
                                    <TD>
```

```
                                    User name:
                              </TD>
                              <TD>
                                    <INPUT TYPE=TEXT NAME="j_username">
                              </TD>
                        </TR>
                        <TR>
                              <TD>
                                    Password:
                              </TD>
                              <TD>
                                    <INPUT TYPE=PASSWORD
NAME="j_password">
                              </TD>
                        </TR>
                        <TR>
                              <TD>
                                    <INPUT TYPE=RESET>
                              </TD>
                              <TD>
                                    <INPUT TYPE=SUBMIT VALUE="Login">
                              </TD>
                        </TR>
                  </TABLE>
            </FORM>
      </BODY>
</HTML>
```

Error.jsp

```
<HTML>
      <HEAD>
            <TITLE>Error Page</TITLE>
      </HEAD>
      <BODY>
            Login failed. Click
            <A HREF="<%=request.getContextPath()%>/Login.jsp">here</A> to try
            again.
      </BODY>
</HTML>
```

SSL (Socket Secure layer)

Để cài đặt SSL trong tomcat bạn phải thực hiện 3 bước sau:

1. Tạo file Keystore.
2. Cấu hình Tomcat để sử dụng Keystore.
3. Cấu hình web application để có thể làm việc với SSL.

Tạo Keystore

Keystore chứa các chi tiết về Certificates cần thiết để giao thức được bảo mật. Sử dụng command window và thực hiện lệnh:

*keytool -genkey -alias **keyAlias** -keypass **certPass** -keystore **c:/testCertificate** -storepass **certPass***

Chú ý: phần in đậm bạn có thể thay đổi tùy ý nhưng –keypass và –storepass phải có giá trị giống nhau

Bạn sẽ phải trả lời một số câu hỏi:

```
C:\Documents and Settings\DANKO>keytool -genkey -alias keyAlias -
keypass certPas
s -keystore c:/testCertificate -storepass certPass
What is your first and last name?
  [Unknown]:  Hai Nguyen Thanh
What is the name of your organizational unit?
  [Unknown]:  home
What is the name of your organization?
  [Unknown]:  Aptech
What is the name of your City or Locality?
  [Unknown]:  Hanoi
What is the name of your State or Province?
  [Unknown]:  Metri
What is the two-letter country code for this unit?
  [Unknown]:  84
Is CN=Hai Nguyen Thanh, OU=home, O=Aptech, L=Hanoi, ST=Metri, C=84
correct?
  [no]:  yes
```

Đặt file keystore vào thư mục webapps của tomcat

Cấu hình trong tomcat

Mở file <*CATALINA_HOME*>/conf/server.xml và thiết lập như sau:

```
<Connector port="8443" maxThreads="200" scheme="https" secure="true"
      SSLEnabled="true" keystoreFile="webapps/testCertificate"
      keystorePass="certPass" clientAuth="false" sslProtocol="TLS" />
```

Web.xml

```
<security-constraint>
      <web-resource-collection>
            <web-resource-name>securedapp</web-resource-name>
            <url-pattern>/*</url-pattern>
      </web-resource-collection>
      <user-data-constraint>
            <transport-guarantee>CONFIDENTIAL</transport-guarantee>
      </user-data-constraint>
</security-constraint>
```

Applet

TestApplet.java

```
import java.applet.*;
import java.awt.*;
public class TestApplet extends Applet {
        public void paint(Graphics g) {
                g.drawString("Welcome in Java Applet.", 40, 20);
        }
         public int myMethod(int a) {
                return (5 + a);
        }

}
```

TestApplet.html

```
<HTML>
    <HEAD><title>Applet In Jsp</title>
        <SCRIPT>
            function add() {
                var result = document.applets[0].myMethod("3");
                alert("The sum is " + result);
            }
        </SCRIPT>
    </HEAD>
    <BODY>
        <APPLET
            CODE     = "TestApplet.class"
            NAME     = "TestApplet"
            WIDTH    = 200
            HEIGHT   = 50
            HSPACE   = 0
            VSPACE   = 0
            ALIGN    = middle
            > </APPLET>
        <INPUT TYPE=BUTTON onClick='add();'
               VALUE="Run Applet's Method ">

    </BODY>
</HTML>
```

Biên dịch TestApplet.java thành testapplet.class và đưa vào cùng thu mục với TestApplet.html (Chọn TestApplet.java press F9).

Applet kết nối với các đối tượng khác:

1) Applet có thể gọi và được gọi từ Javascript.
2) Applet có thể kết nối với applet khác sử dụng javascript object (JSObject).
3) Applet có thể liên hệ với servlet

JSP - Java Server Pages

Giới thiệu về JSP

Jsp được sử dụng để tạo web động (dynamic web), trong ứng dụng web đa tầng (multi layered web app), JSP được sử dụng làm tầng trình diễn. Nó cũng có khả năng tương tác với tầng ứng dụng (application layer) và tạo ra web động dựa trên business logic.

Khi JSP được gọi lần đầu, nó được trình biên dịch phân tích và biên dịch thành servlet class. Nếu biên dịch thành công, jsp servlet class này sẽ được nạp vào bộ nhớ. Trong lần gọi tiếp theo, servlet clas này đã ở sẵn trong bộ nhớ, tuy nhiên nó có thể được update. Bởi vậy, trình biên dịch sẽ so sánh timestamp của jsp servlet với jsp. Nếu jsp mới hơn thì nó sẽ biên dịch lại. Vì vậy bạn có thể cảm thấy mỗi khi vào trang jsp lần đầu bao giờ cũng thấy chậm hơn các lần sau.

Vòng đời của JSP (Life circle)

1) Translation: JSP được dịch thành JSP servlet với 3 phương thức: jspInit(), _jspService() và jspDestroy(). VD: *public class SimplePage_jsp extends HttpJspBase.*
2) Compilation: JSP servlet được biên dịch sang class
3) Thực thi các phương thức init() và service() và chuyển về dạng html để hiển thị lên browser.

Expresssion (biểu thức)

<%=3+2%>

Scriptlets

<% String a= “abc”;

Int i = 0;

out.println(“This is scriptlets sample”);

%>

Declarations

Declarations cho phép bạn định nghĩa các method và biến mà bạn có thể sử dụng ở bất cứ đâu trong trang jsp.

```
<%@page import="java.util.Calendar"%>
<%!
    String getSystemTime() {
        return Calendar.getInstance().getTime().toString();
    }
%>

<%
    out.println("Current Time: " + getSystemTime());
%>
```

Comments

Server side: <%-- --%>

Client side: <!-- -->

Directives

Directives là chỉ dẫn tới JSP container chứa thông tin chỉ dẫn JSP container phải biên dịch JSP thành servlet tương ứng.

Attribute	Kiểu giá trị	Giá trị mặc định
language	Scripting language name	"java"
info	String	Depends on the JSP container
contentType	MIME type, character set	"text/html;charset=ISO-8859-1"
extends	Class name	None
import	Fully qualified class name or package name	None
buffer	Buffer size or false	8192
autoFlush	Boolean	"true"
session	Boolean	"true"
isThreadSafe	Boolean	"true"
errorPage	URL	None
isErrorPage	Boolean	"false"

VD: **errorPage**

Đây là trang web phát sinh lỗi. Khi phát sinh lỗi sẽ gọi đến trang errorPage.jsp (directive của trang này phải đặt là isErrorPage = true)

```
<%@ page errorPage="errorPage.jsp" %>

<%
   int i = 10;
   i = i / 0;
%>
```

Trang hiển thị lỗi errorPage.jsp

```
<%@ page isErrorPage="true"%>
<html>
      <head>
            <title>Error Page</title>
      </head>
      <body>
            <b>Exception:</b>
            <br>
            <%=exception.toString()%>
      </body>

</html>
```

autoFlush Attribute

autoFlush liên quan đên page buffer. Khi giá trị được đặt là true, JSP container sẽ tự động flush buffer khi buffer đầy. Nếu false, có thể sẽ có exception throw ra nếu buffer bị tràn (overflow). Ta có thể flush buffer manually qua việc sử dụng method của JspWriter, VD:

```
out.flush();
```

VD, đoạn code sau set autoFlush attribute bằng false:

```
<%@ page autoFlush="false" %>
```

Session Attribute

Nếu đặt giá trị là true thì session object sẽ trỏ tới session hiện tại hoặc tạo mới. Ngược lại sẽ ko có session nào được định nghĩa hoặc tạo mới. Do đó implicit object cho session ko sẵn sàng để sử dụng.

```
<%@ page session="false"%>
```

Action

jsp:useBean

Để tạo 1 instance cho java bean để có thể sử dụng trong jsp:

VD:

JavaBean

```
package com.aptech.bean;

public class Calc {
    private int a;
    private int b;
    public int add() {
        return getA()+getB();
    }

    public int getA() {
        return a;
    }

    public void setA(int a) {
        this.a = a;
    }

    public int getB() {
        return b;
    }

    public void setB(int b) {
        this.b = b;
    }
}
```

jsp:

```
<jsp:useBean id="calc" class="com.aptech.bean.Calc" scope="request" />
```

jsp:setProperty

Gán giá trị cho thuộc tính của java bean

```
<jsp:setProperty name="calc" property="a" value="4"/>
<jsp:setProperty name="calc" property="b" value="2"/>
```

jsp:getProperty

Hiển thị giá trị thuộc tính của java bean

```
<jsp:getProperty name="calc" property="b"/>
<%=calc.add()%> <!—- cho kết quả = 6-->
```

jsp:param

Xem phần jsp:forward

jsp:include

Dùng để nhúng nội dung của 1 trang khác vào trang hiện thời. VD:

```
<jsp:include page="/success.jsp"/>
```

jsp:forward

forward đến trang khác, có thể truyền tham số sử dụng thẻ jsp:param

```
<jsp:forward page="/success.jsp">
     <jsp:param name="a" value="abc"/>
</jsp:forward>
```

Tạo success.jsp với nội dung như sau:

```
Parameter a = <%=request.getParameter("a")%>
```

jsp:plugin

jsp:plugin action được dùng để tạo ra thể HTML <OBJECT> hoặc <EMBED> chứa các cấu trúc tương ứng để chỉ định trình duyệt dowload phần mềm Java plugin (nếu cần), và khởi tạo Java applet hoặc các thành phần javabean được chỉ định.

```
<jsp:plugin type="applet" code="Clock.class"
      codebase="http://localhost:8080/home/applets" jreversion="1.4.1">

      <jsp:params>
            <jsp:param name="scriptable" value="false" />
      </jsp:params>

      <jsp:fallback>Sorry, we are unable to start the Java plugin/>
</jsp:fallback>

</jsp:plugin>
```

jsp:params

Là thẻ con nằm trong jsp:plugin

jsp:fallback

Là thẻ con nằm trong jsp:plugin dùng để thông báo lỗi nếu không thể khởi tạo được plugin.

JSP EL Expressions

EL – Expression language (biểu thức)

Disable EL

Muốn vô hiệu hóa EL trong JSP ta sử dụng directives như sau:

```
<%@page isELIgnored="true" %>
```

Or configure in web.xml

```
    <jsp-config>
        <jsp-property-group>
            <url-pattern>*.jsp</url-pattern>
            <el-ignored>false</el-ignored>
        </jsp-property-group>
    </jsp-config>
```

Truy cập JavaBean

Sử dụng JavaBean trong ví dụ về jsp:useBean (Calc.java)

```
<jsp:useBean id="calc" class="com.aptech.bean.Calc"
                    scope="request" />

${calc.a} <!--hiển thị kết quả là giá trị của property a-->
```

Hiển thị attribute

1) Hiển thị attribute

```
<%request.setAttribute("b", "abc");%>
${b} <!--hiển thị giá trị của attribute b-->
```

2) Hiển thị collection

```
<%
   String[] firstNames = { "Bill", "Scott", "Larry" };
   request.setAttribute("first", firstNames);
%>

<UL>
   <LI>${first[0]}
   <LI>${first[1]}
   <LI>${first[2]}
</UL>
```

Sử dụng implicit object

```
<b> Session ID:</b> ${pageContext.session.id}
<br/>
<b>Header accept:</b> ${header.Accept}
<br />
or
<br />
${header["Accept"]}
<br/>
<b>header accept-encoding:</b> ${header["Accept-Encoding"]}
```

Sử dụng các EL operator (toán tử EL)

Arithmetic Operators (toán tử số học)

+, –, *, /(div), %(mod)

Relational Operators (Toán tử quan hệ)

== (eq), != (ne), < (lt), > (gt), <= (le), >= (ge)

Logical Operators (toán tử logic)

&&(and), ||(or), !(not)

The empty Operator (toán tử empty)

Nếu đối tượng có giá trị là null thì trả về true và ngược lại. Cú pháp: `${empty x}`

Biểu thức điều kiện (conditionally expression)

${ test ? expression1 : expression2 }

VD: `${1==2 ? "aaa" : "bbb"}`

JSTL - Java Server Pages Standard Tag Library

Các thành phần và chức năng của:

- ***Iteration*** ***(vòng lặp) và*** ***conditional*** ***(điều kiện):*** Iteration và conditional logic được dùng phổ biến bởi JSP; tuyn nhiên, chỉ một số phương thức tiêu chuẩn của loại này được thực hiện bằng các thành phần script. Đôi khi ta không muốn sử dụng các thành phần, Vì vậy custom tags được tạo ra như 1 sự thay thế. JSTL cung cấp tập các thẻ custom tag cho lặp và điều kiện logic.

- ***Expression Language (ngôn ngữ biểu thức):*** 1 trong các thuộc tính được đề cập nhiều của JSTL là JSP EL được hỗ trợ.
- ***URL manipulation***: JSP hỗ trợ rất giới hạn về xử lý ỦRL. Các tài nguyên bên ngoài 1 ứng dụng JSP không thể được forward hoặc include sử dụng các JSP action tiêu chuẩn, JSP cũng không cung cấp các phương thức chắc chắn rằng URL được mã hóa đúng với các parameter và thông tin về session. JSTL giải quyết tất cả các vấn đề này với các thẻ xử lý URL.
- ***Hỗ trợ I18N (quốc tế hóa):*** hỗ trợ Internationalization là một nhiệm vụ khó và JSP không đưa ra các hỗ trợ mặc định cho nó. Các giải pháp phổ biến nhất hỗ trợ các vấn đề I18N thông qua JSP được chuẩn hóa bởi tập thẻ custom tag của JSTL
- ***XML manipulation***: XML được sử dụng rộng rãi trong java, vì vậy JSTL cung cấp tập các custom tag được thiết kế riêng để xử lý XML.
- ***Truy cập Database:*** Databases là một giải pháp tiêu chuẩn để lưu các dữ liệu phức tạp trong các trang web động. JSTL đưa ra tập các thẻ SQL cho phép truy vấn CSDL.

Tại sao bạn nên sử dụng JSTL

JSTL khá mới nhưng các chức năng mà nó thay thế không mới. Mọi thứ mà JSTL cung cấp có thể được thực hiện và thường xuyên được thực hiện với các custom tag của 1 framework nào đó hoặc JSP container. Là vô nghĩa để giả bộ rằng JSTL luôn luôn được sử dụng để chống lại các giải pháp trước. JSTL là một tiêu chuẩn, vì nó được xây dựng chung bởi các thành viên của cộng đồng JSP và nó mang lại một sự bổ xung tốt và hữu dụng với nhiều hàm hữu ích. JSTL cung cấp các thành phần chung cơ bản để xây dựng một dự án mà các developer ko cần phải xây dựng lại.

Một lý do lớn khác là bằng việc sử dụng JSTL bạn có thể không cần biết nhiều về java hoặc bạn không mấy quan tâm đến kiến trúc MVC liên quan đế việc sử dụng servlet, filter, javabean và JSP.

Cài đặt JSTL

Copy 2 .jar file vào thu mục lib:

```
1)    standard.jar

2)    jstl.jar
```

Core tags

General-Purpose Tags

Các thẻ này cùng với EL được sử dụng cho các tác vụ đơn giản thông thường. Tập thẻ này bao gồm: `out`, `set`, `remove`, and `catch` actions.

<c:out>

Thẻ out thực hiện JSTL expression (biểu thức) và đánh giá kết quả, gửi kết quả tới đối tượng JspWriter hiện hành. Thẻ này hữu dụng như 1 sự thay thế của getProperty action. Trong trường hợp JavaBean không hiện hữu hoặc developer không muốn sử dụng bean, out action đưa ra các chức năng tương đương. Tuy nhiên, trong jsp 2.0 sử dụng out là không thích hợp vì bạn có thể đơn giản đưa một biểu thức tương đương trực tiếp vào trang và có hiệu ứng tương tự.

out action có attributes:

1. ***value***: giá trị của `value` attribute là biểu thức để đánh giá. Kết quả của biểu thức được gửi tới client sử dụng đối tượng JspWriter của trang hiện hành. Giá trị của thuộc tính `value` có thể được chỉ định lúc chạy (runtime).
2. ***escapeXml***: chỉ ra rằng nếu kết quả chuỗi hiển thị được chuyển sang dạng mã lệnh đối với các 1 số ký tự đặc biệt nếu giá trị là false. Giá trị mặc định là `true`. Giá trị của `escapeXml` có thể được chỉ định khi đang chạy (runtime).

   VD: <c:out value="${'&**lt**;, &**gt**;, &, ', "'}" escapeXml="false"/>

   Trả về kết quả là: <, >, &, ', "

   <c:out value="${'&**lt**;, &**gt**;, &, ', "'}" />

   Trả về kết quả là: <, >, &, ', "

3. ***default***: định nghĩa giá trị mặc định được sử dụng khi biểu thức không chạy hoặc kết quả là null.

<c:set>

Set action tương tự với setProperty action. Thẻ này đắc dụng vị nó đánh giá biểu thức và sử dụng kết quả của javaBean hoặc 1 đối tượng java.util.Map.

set action có các thuộc tính sau:

1. value: là giá trị gán. Thuộc này có thể là biểu thức runtime.
2. var: là tên của tên được set giá trị.
3. scope: là phạm vi của biến `page`, `request`, `session`, và `application`, page là giá trị mặc định.
4. target: Là tên của đối tượng có thuộc tính được set giá. Thường là JavaBean object với phương thức setter tương ứng hoặc a `java.util.Map` object. Giá trị này có thể là giá trị runtime.
5. property: là tên của thuộc tính của đối tượng được chỉ định bởi `target` attribute. Thuộc tính này có thể là giá trị runtime.

VD:

```
<%
    com.aptech.bean.User user = new com.aptech.bean.User();
%>

 <c:set var ="name" value="MichaelJs" property="username" target="user"/>
 <c:out value="${name}" default="xxx"/>
```

<c:remove>

Sử dụng để xóa biến.

C:remove có 2 attribute:

- scope: định nghĩa phạm vi tìm kiếm biến để xóa. Nếu không được chỉ định thì biến bị xóa bởi phương thức `PageContext removeAttribute(varName)`. Nếu được chỉ định thì phương thức được gọi là `PageContext removeAttribute(varName, scope`.
- var: tên biến để remove.

VD:

```
<c:set var ="a" value="abc" scope="session"/>
<c:set var ="b" value="xyz" scope="session"/>
<c:out value="${a}" default="xxx"/>

<c:remove var="a" scope="session"/>
a after removing:<c:out value="${a}"/>
```

```
<c:remove var="b" scope="request"/>
b after removing: <c:out value="${b}"/>
```

<c:catch>

Iteration (vòng lặp)

<forEach>

forEach tag hỗ trợ các kiểu dữ liệu *java.util.Collection*, *java.util.Iterator*, *java.util.Enumeration*, hoặc *java.util.Map*. forEach tag có các thuộc tính sau:

- var: tên của đối tượng đặt ra để sử dụng trong vòng lặp (tùy ý).
- items: đối tượng được chỉ định để lặp (Iterator, Enumeration hoặc Map). Giá trị có thể là runtime.
- varStatus: tên biến lưu trạng thái của vòng lặp. Đối tượng có kiểu `javax.servlet.jsp.jstl.LoopTagStatus`.
- begin: chỉ định phần tử đầu tiên được duyệt.
- end: Phần tử cuối được duyệt.
- step: bước duyệt.

VD:

```
<%@ page import="java.util.*" %>

<%
    List list = new ArrayList();
    list.add("a");
    list.add("b");
    list.add("c");

    request.setAttribute("set", list.iterator());
%>


<c:forEach var="item" begin="0" items="${set}">
                The value is <c:out value="${item}" />
</c:forEach>
```

<forTokens>

Tương tự forEach nhưng phần tử tập hợp do lập trình viên tự định nghĩa trong thẻ items, phân tách bởi ký tự được định nghĩa trong thẻ demims. Các thuộc tính khác tương tự.

VD:

```
<c:forTokens var="item" delims="~" items="token1~token2~token3">
            The value is <c:out value="${item}" />
</c:forTokens>
```

Conditionals (câu lệnh điều kiện)

<c:if>

- test: biểu thức điều kiện
- var: tên biến lưu giá trị của biểu thức trong test. Biến này là biến scope có thể sử dụng sau này
- scope: là phạm vị của biến được định nghĩa trong var

VD:

```
<%
    request.setAttribute("y", "abc");
%>
<c:if test="${y=='abc'}" var="a">
    Welcome
</c:if>
<br />--> <c:out value="${a}" />.
```

<choose>, <when>, và <otherwise>

```
<c:choose>
      <c:when test="${y == 'aabc'}">
                  y == 'aabc'
            </c:when>
      <c:otherwise>
                  y != 'aabc'
            </c:otherwise>
</c:choose>
```

URL Manipulation

<c:import> and <c:param>

The `import` tag has the following attributes:

- url: Giá trị của URL import vào bao gồm cả tuyệt đối và tương đối. Giá trị có thể chỉ định runtime.
- context: The `context` attribute can be used to provide the name of a context when accessing a relative URL resource that belongs to a foreign context. The value may be specified by a runtime value. A foreign context is another Web Application that is executing on the same Web server. This value should be used if the URL is relative to that named foreign context. Note that this is the equivalent of a request dispatch across

Web Application boundaries and may not work on particular containers due to security or other restrictions.

- var: The `var` attribute can be used to name a scoped variable to contain the resource's content. The type of the variable is String.
- scope: The `scope` attribute defines the scope of a variable named by the `var` attribute. Valid values are `page`, `request`, `session`, and `application`.
- charEncoding: The `charEncoding` attribute can be used to specify the character encoding of the content at the input resource. By default it is assumed the standard, `ISO-8859-1`, encoding is used.
- varReader: The `varReader` attribute is available to create a `Reader` object for use by content in the import tag's body. The value of the `varReader` attribute is the name of the scoped variable; the type of the variable is `java.io.Reader`.

VD:
```
<c:import url="/jstl.jsp" var="value" />
<c:out value="${value}" />
```

c:param

Là thẻ con của c:import với name là tên parameter, value là giá trị.

c:url

Là phương thức đưa ra để tự động mã hóa URL với các thông tin về session và parameters. Tác dụng tương đương với phương thức `HttpServletResponse encodeURL()`.

VD:

```
<a href='<%= response.encodeURL("/index.jsp")%>'>Book Site</a>

Hoặc

<c:url value="/index.jsp" var="u" />
<a href="${u}">Book Site</a>
```

`url` tag có các thuộc tính sau:

- value: The `value` attribute is used to provide the URL to be processed.
- context: The `context` attribute defines the name of the context when specifying a relative URL resource of a foreign context.
- var: The `var` attribute can be used to export the rewritten URL to a scoped variable. The value of the `var` variable is the processed URL. The type of the variable is String. The variable's scope depends on the `scope` attribute.
- scope: The `scope` attribute determines the scope of the object named by the `var` attribute. Valid values are `page`, `request`, `session`, and `application`.

<c:redirect>

Tương đương `HttpServletResponse sendRedirect()`

VD: <c:redirect url="/jstl.jsp"/>

SQL tag Library

```
<sql:setDataSource driver="com.microsoft.sqlserver.jdbc.SQLServerDriver"
      url="jdbc:sqlserver://localhost\\hai:1443;databaseName=test;" user="sa"
      password="****" var="conn" />
<sql:query var="users" dataSource="${conn}">
            SELECT * FROM [USER]
</sql:query>

<table border="1">
      <tr>
            <c:forEach var="colName" items="${users.columnNames}">
                  <th>
                        <c:out value="${colName}" />
                  </th>
            </c:forEach>
      </tr>
      <c:forEach items="${users.rowsByIndex}" var="row">
            <tr>
                  <c:forEach var="col" items="${row}">
                        <td>
                              <c:out value="${col}" />
                        </td>
                  </c:forEach>
            </tr>
      </c:forEach>
</table>
```

I18N – Internationalisation (quốc tế hóa)

1) Thay đổi ngôn ngữ theo khu vực (locale) mà không cần thay đổi nhiều trong công nghệ.

2) Localization: giúp tạo ra ứng dụng đa ngôn ngữ, định dạng tiền tệ (currency), ngày tháng (date-time) theo khu vực địa lý (geographic region). Sử dụng java.util.Locale.

3) Unicode: là bảng mã chứa các ký tự và các biểu tượng của tất cả các nước chính trên thế giới.

4) Resource bundle: là .properties file chứa dữ liệu dạng key và value thể hiển đa ngôn ngữ. Định dạng xx_YY với xx là 2 chữ viết tắt của ngôn ngữ (chữ thường; YY là 2 chữ viết tắt cho tên quốc gia (chữ hoa). VD MessageBundle_en_US.properties là file cấu hình cho tiếng anh Mỹ. MessageBundle_ko_KR cho Seul Hàn Quốc.

MessageBundle.properties

MessageBundle_fr.properties

Áp dụng I18N trong JSP:

VD: 1

test.jsp

```
<%@ taglib uri="http://java.sun.com/jstl/fmt" prefix="fmt"%>
<fmt:bundle basename="com.aptech.conf.MessageBundle">
<html>
  <head>
    <title><fmt:message key="title"/></title>
  </head>
  <body>
  <fmt:message key="welcome"/>
  </body>
</html>
</fmt:bundle>
```

MessageBundle.properties

```
title=Welcome!
welcome=Welcome to the I18N resource bundle sample.
```

VD2:

```
<%@ page import="java.util.*, com.aptech.bean.Welcome"%>
<%
  Locale locale = Locale.FRANCE;//request.getLocale();
  ResourceBundle  rb =
ResourceBundle.getBundle("com.aptech.conf.MessageBundle", locale);
  Welcome welcome = new Welcome();
  welcome.setTitle(rb.getString("title"));
  welcome.setWelcome(rb.getString("welcome"));
  request.setAttribute("content", welcome);
%>
<%@ taglib uri="http://java.sun.com/jstl/core" prefix="c" %>
<html>
  <head>
    <title>${content.title}</title>
  </head>
```

```
  <body>
  ${content.welcome}
  </body>
</html>
```

Welcome.java

```
package com.aptech.bean;

public class Welcome {
    protected String title = null;
    protected String welcome = null;
```

-----thêm các phương thức get và set vào

```
}
```

Thay đổi Locale từ default sang Locale.FRANCE sẽ thấy kết quả hiển thị thay đổi.

Áp dụng I18N trong Servlet:

```
protected void doGet(HttpServletRequest request,
                    HttpServletResponse response) throws
ServletException, IOException {
            ResourceBundle bundle = ResourceBundle.getBundle(
                        "com.aptech.conf.MessageBundle",
Locale.KOREAN);
            Enumeration keys = bundle.getKeys();
            PrintWriter out = response.getWriter();
            while (keys.hasMoreElements()) {
                  String key = (String) keys.nextElement();
                  out.println("key:" + key);}}
```

ResourceBundle class

getBundle():

Tạo đối tượng bundle với vùng địa lý nào đó (locale):

VD: `ResourceBundle bundle = ResourceBundle.getBundle("com.aptech.conf.MessageBundle", Locale.KOREAN);`

getKeys()

Trả về tập các key trong .properties theo kiểu Enumeration

```
VD: Enumeration keys = bundle.getKeys();
```

getLocale()

Trả về đối tượng locale bundle

getObject(key)

Tham số là tên key trong .properties file. Kết quả trả về là value của key tương ứng. Nếu key không tồn tại sẽ throw exception:

```
java.util.MissingResourceException
```

getString(key)

Trả về value tương ứng với tham số key trong .properties file.

Date Formatting (Định dạng ngày tháng)

```
Date today = new Date();
DateFormat df = DateFormat.getDateInstance(DateFormat.DEFAULT,
Locale.FRENCH);
DateFormat df1 = DateFormat.getDateInstance(DateFormat.DEFAULT,
Locale.ENGLISH);
PrintWriter out = response.getWriter();
out.println(df.format(today));
out.println(df1.format(today));
SimpleDateFormat sf = new SimpleDateFormat("yyy-MM-dd", Locale.FRENCH);
out.println(sf.format(today));
```

Curencies Formatting (định dạng tiền tệ):

Currency Class

getInstance(Locale)

Lấy instance (đối tượng) kiểu Currency theo locale được chỉ định.

```
VD: Currency cr = Currency.getInstance(Locale.US);
```

getCurrencyCode():

trả về mã tiền tệ theo locale

VD:

```
out.println("Currency code:" + cr.getCurrencyCode());//kết quả trả về là USD
```

getSymbol():

Trả về biểu tượng tiền tệ theo locale.

VD: `out.println("Currency symbol):" + cr.getSymbol());//kết quả trả về là $`

NumberFormat Class:

format(double):

```
NumberFormat nf = NumberFormat.getPercentInstance(Locale.FRANCE);
out.println("nf:" + nf.format(0.51));
```

getInstance():

Trả về đối tượng NumberFormat với locale mặc định

getCurrency():

Trả về đối tượng kiểu Currency .

```
Currency cr = NumberFormat.getInstance().getCurrency();
out.println("CR symbol:" + cr.getSymbol());
```

parse(String):

Chuyển từ chuỗi thành số

```
NumberFormat nf = NumberFormat.getInstance();
out.println("nf:" + nf.parse("1.999.999,01"));//Hiển thị kết quả 1.999
```

setCurrency(Currency):

```
Currency c = Currency.getInstance(Locale.JAPAN);
nf.setCurrency(c);
out.println("nf:" + nf.getCurrency().getCurrencyCode());
```

MessageFormat

```
String string = "Hello {0}, it is {1,date} and you have {2, number, currency}
in your pocket.";
Object[] objects = {"foo", new Date(), new Integer(10)};
String result = MessageFormat.format(string, objects);
```

JSP custom tags

Tag library descriptor

Đây là một file XML mô tả các custom tags trong ứng dụng JSP. Nó định nghĩa 1 custom tag và được lưu với phần mở rộng .tld.

Tag handler

Đây là một file java đơn giản chứa code cho các chức năng của custom tag. Có 2 loại ***classic*** và ***simple***.

Các thuật ngữ quan trọng liên quan

Classic Custom Tag

Được định nghĩa bởi classic tag handler class.

Simple Custom Tag

Được định nghĩa bởi simple tag handler class.

Basic tags

Basic tag thông thường không chứa bất cứ body nào. Kể cả khi nó chứa body, phần body cũng ko được xử lý bởi tag này. Tag handler class của basic tag implements từ Tag interface và extends TagSupport class.

Iteration tags

Các thẻ này thường là các thẻ rỗng. Tag handler cho iteration tag implements IterationTag interface. Interface này extends từ Tag interface.

Complex tags

Các thẻ này hỗ trợ body. Thẻ này implements từ BodyTag và extends BodyTag Support class.

Jasper 2

JSP engine là 1 thành phần của web container thực hiện chuyển từ JSP sang servlet code. Jasper engine là tên của JSP engine trong tomcat 4. Trong tomcat5, nó có tên là Jasper 2. Nó là sự triển khai của đặc tả Sun Microsystem's JSP2.0.

Từng bước tạo 1 custom tag:

Tạo tag library descriptor (.tld file).

TLD là một xml file định nghĩa thư viện thẻ và các thẻ. Dòng đầu tiên là xml header chuẩn:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE taglib
PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library 1.1//EN"
"http://java.sun.com/j2ee/dtds/web-jsptaglibrary_1_1.dtd">
```

TLD file chứa thẻ gốc <taglib. Thẻ này có các thẻ con sau:

- tlibversion
- jspversion
- shortname
- info
- uri
- tag

1) tlibversion chỉ định version của thư viện thẻ có định dạng sau:

   ([0-9])* ("." [0-9])? ("." [0-9])? ("." [0-9])?

2) The jspversion chỉ định JSP version. Định dạng giống với tlibversion:

   ([0-9])* ("." [0-9])? ("." [0-9])? ("." [0-9])?

3) shortname chỉ định shortname của thu viện tag library. Giá trị này phải bắt đầu bằng chữ và không chứa khoảng trắng.

4) info chứa thông tin cho mục đích làm tài liệu. Dữ liệu mặc định là rỗng.

5) uri chỉ định link trỏ đến thư viện thẻ.

6) tag là thành phần quan trọng nhất của tag library. Bạn có thể chỉ định hơn 1 thẻ tag.

VD1 Classic custom tag without body

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE taglib
PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library 1.1//EN"
"http://java.sun.com/j2ee/dtds/web-jsptaglibrary_1_1.dtd">
<taglib>
    <tlibversion>1.0.123.123.234</tlibversion>
    <jspversion>1.2</jspversion>
    <shortname></shortname>
    <info>Example tags</info>
    <uri></uri>
    <tag>
        <name>myTag</name>
        <tagclass>com.aptech.customtag.HelloTag</tagclass>
```

```
            <attribute>
                <name>name</name>
                <required>true</required>
            </attribute>
        </tag>
    </taglib>
```

Khai báo trong web.xml

```
<jsp-config>
    <taglib>
        <taglib-uri>/hello</taglib-uri>
        <taglib-location>/WEB-INF/tlds/hello.tld</taglib-location>
    </taglib>
</jsp-config>
```

Tạo Tag Handler

```
package com.aptech.customtag;

import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;

public class HelloTag extends TagSupport {

    private String name;

    public int doEndTag() throws JspException {
        JspWriter out = pageContext.getOut();
        try {
            out.println("<hr/>");
        } catch (Exception e) {
        }
        return super.doEndTag();
    }

    /**
     * @return the name
     */
    public String getName() {
        return name;
    }

    /**
     * @param name the name to set
     */
    public void setName(String name) {
        this.name = name;
    }

    public int doStartTag() {
        try {
            JspWriter out = pageContext.getOut();
            out.println("<hr/>");
```

```
                out.println("Hello " + name);
        } catch (Exception ex) {
            throw new Error("All is not well in the world.");
        }
        return SKIP_BODY;
    }
}
```

Xử lý tag trong JSP

```
<%@ taglib uri="/WEB-INF/tlds/hello.tld" prefix="hello" %>
<html>
        <head>
                <title>Your Standard Hello World Demo</title>
        </head>
        <body bgcolor="#ffffff">
                <hello:myTag name="Hai"/>
        </body>
</html>
```

VD2 classic custom tag with body

Tag handler ClassicTagDemo.java

```
package com.aptech.taghandler;

import java.io.IOException;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.servlet.http.*;
import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;

public class ClassicTagDemo extends BodyTagSupport {
    //These variable represent the custom tag's attributes
    private String heading = null;
    private String image = null;
    private String width = null;
    private String height = null;

    public int doStartTag() throws JspException {
        JspWriter out = pageContext.getOut();
        try {
            int h = Integer.parseInt(heading);
            if (!(h > 0 && h < 7)) {
                out.println("<font color='red'>The 'heading' value must
between 1 and 6 inclusive.</font>");
            }
        } catch (Exception e) {
            throw new JspException(e.getMessage());
        }

        return EVAL_BODY_BUFFERED;
    }
```

```
    public int doEndTag() throws JspException {
        JspWriter out = pageContext.getOut();
        String imgDir = ((HttpServletRequest)
pageContext.getRequest()).getContextPath() + "/img/";

        String message = "<font color='#FF0000'>" +
getBodyContent().getString().trim() + "</font>";
        try {
            out.println("<img src=\"" + imgDir + image +
                    "\" width=\"" + width + "\" height=\"" + height +
                    "\" align=\"left\">" + "<H" + heading + ">" +
                    message + "</H" + heading + ">");
        } catch (IOException ex) {

Logger.getLogger(ClassicTagDemo.class.getName()).log(Level.SEVERE, null, ex);
        }
        return EVAL_PAGE;

    }

    public void setHeading(String level) {
        this.heading = level;
    }

    public void setImage(String name) {
        this.image = name;
    }

    public void setWidth(String width) {
        this.width = width;
    }

    public void setHeight(String height) {
        this.height = height;
    }

    //the JSP container may cache and reuse tag handler objects.
    //this method releases instance variables so that the tag handler
    //can be reused afresh
    public void release() {
        heading = null;
        image = null;
        width = null;
        height = null;
    }
}
```

Tag descriptor

```
<?xml version="1.0" encoding="ISO-8859-1" ?>

<!DOCTYPE taglib
PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library 1.2//EN"
"http://java.sun.com/dtd/web-jsptaglibrary_1_2.dtd">
```

```
<taglib>
    <tlib-version>1.0</tlib-version>
    <jsp-version>1.2</jsp-version>
    <short-name>clsTagDemo</short-name>

    <!-- Here is the URI you use with the 'taglib' directive in the JSP -->
    <uri>/aptech/Classic demo...</uri>

    <description>Aptech demo classic tag</description>

    <tag>
        <name>logo</name>
        <!-- make sure to use the fully qualifed class name -->
        <tag-class>com.aptech.taghandler.ClassicTagDemo</tag-class>

        <body-content>JSP</body-content>

        <description>This tag writes a logo inside the JSP.</description>

        <attribute>
            <name>heading</name>
            <!-- The logo tag requires this attribute -->
            <required>true</required>
            <!-- The attribute can take a JSP expression as a value -->
            <rtexprvalue>true</rtexprvalue>
            <description>The heading level for the logo; 1 through 6.
            </description>
        </attribute>

        <attribute>
            <name>image</name>
            <required>true</required>
            <rtexprvalue>true</rtexprvalue>
            <description>The image name for the logo.</description>
        </attribute>

        <attribute>
            <name>width</name>
            <required>true</required>
            <rtexprvalue>true</rtexprvalue>
            <description>The image width for the logo.</description>
        </attribute>

        <attribute>
            <name>height</name>
            <required>true</required>
            <rtexprvalue>true</rtexprvalue>
            <description>The image height for the logo.</description>
        </attribute>
    </tag>
</taglib>
```

Web.xml declaration

```
      <jsp-config>
        <taglib>
            <taglib-uri>/aptech/Classic demo...</taglib-uri>
```

```
            <taglib-location>/WEB-INF/tlds/ClassicDemo.tld</taglib-location>
        </taglib>
    </jsp-config>
```

Use tag in JSP

```
<%@ taglib uri="/aptech/Classic demo..." prefix="clsTagDemo"%>
<%
    //If required= false -> error for image attribute
    String imgName = "island_hula.gif";
%>

<clsTagDemo:logo heading="6" image="<%=imgName%>" width="335" height="150">
    Come here come here, babe!
</clsTagDemo:logo>
```

Simple Tag

Descriptor

```
<taglib
    xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
    http://java.sun.com/xml/ns/j2ee/web-jsptaglibrary_2_0.xsd"
    version="2.0">
    <tlib-version>1.0</tlib-version>
    <jsp-version>2.0</jsp-version>
    <short-name>Example TLD</short-name>
    <uri>/aptech/simple/demo...</uri>
    <tag>
        <name>date</name>
        <tag-class>com.aptech.taghandler.SimpleTagDemo</tag-class>
        <body-content>empty</body-content>
        <attribute>
            <name>format</name>
        </attribute>
    </tag>

    <tag>
        <name>dynamicAttribute</name>
        <tag-class>com.aptech.taghandler.DynamicAttributeTag</tag-class>
        <body-content>empty</body-content>
        <dynamic-attributes>true</dynamic-attributes>
    </tag>

    <tag>
        <name>body</name>
        <tag-class>com.aptech.taghandler.BodySimpleTag</tag-class>
        <body-content>scriptlet</body-content>
    </tag>
</taglib>
```

Tag Handler

BodySimpleTag

```
package com.aptech.taghandler;

import java.io.IOException;
import java.text.SimpleDateFormat;
import java.util.Date;
import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;

public class BodySimpleTag extends SimpleTagSupport {
    public void doTag() throws JspException, IOException {
        PageContext context = (PageContext) getJspContext();
        JspWriter out = context.getOut();
        out.write("hello");
    }
}
```

DynamicAttributeTag

```
package com.aptech.taghandler;

import javax.servlet.jsp.tagext.*;
import javax.servlet.jsp.*;
import java.util.*;
import java.io.IOException;

public class DynamicAttributeTag
    extends SimpleTagSupport implements DynamicAttributes {
  protected Hashtable map = new Hashtable();

  public void setDynamicAttribute(String uri, String name,
      Object value) throws JspException{
    map.put(name, value);
  }

  public void doTag() throws JspException, IOException {
    JspWriter out = getJspContext().getOut();
    for (Enumeration keys = map.keys();
         keys.hasMoreElements();) {
      Object key = keys.nextElement();
      Object value = map.get(key);
      out.print("<b>Attribute:</b><br>");
      out.print("name: " + key.toString() + "<br>");
      out.print("value: " + value.toString() + "<br>");
    }
  }
}
```

SimpleTagDemo

```
package com.aptech.taghandler;
```

```
import java.io.IOException;
import java.text.SimpleDateFormat;
import java.util.Date;
import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;

public class SimpleTagDemo extends SimpleTagSupport {

    private String format;

    public void setFormat(String format) {
        this.format = format;
    }

    public void doTag() throws JspException, IOException {
        JspWriter out = getJspContext().getOut();
        if (format != null) {
            SimpleDateFormat sdf = new SimpleDateFormat(format);
            out.println(sdf.format(new Date()));
        } else {
            out.println(new Date().toString());
        }
    }
}
```

Web.xml

```
        <taglib>
            <taglib-uri>/aptech/Simple demo...</taglib-uri>
            <taglib-location>/WEB-INF/tlds/SimpleDemo.tld</taglib-location>
        </taglib>
```

JSP

```
The current date is <simple:date format="dd/MM/yy" />.
 <hr>
 <simple:dynamicAttribute name="test" value="a value"/>
 <hr>
 <simple:body/>
```

Using Tag file

caps.tag

```
<%@tag body-content="scriptless"%>

<jsp:doBody var = "theBody" scope="session"/>
<%String bc = (String) session.getAttribute("theBody");%>
<%=bc.toUpperCase()%>
```

heading.tag

```
<%@attribute name="heading1"%>
<%@attribute name="heading2"%>
<%@attribute name="heading3"%>
<%@attribute name="heading4"%>

<tr>
    <td>${heading1}</td>
    <td>${heading2}</td>
    <td>${heading3}</td>
    <td>${heading4}</td>
</tr>
```

column.tag

```
<%@attribute name="col1"%>
<%@attribute name="col2"%>
<%@attribute name="col3"%>
<%@attribute name="col4"%>

<tr>
    <td>${col1}</td>
    <td>${col2}</td>
    <td>${col3}</td>
    <td>${col4}</td>
</tr>
```

JSP

```
<tags:caps> <h1>This is tags demo</h1></tags:caps>
 <table border="1">
     <tags:heading heading1 ="1" heading2 ="2" heading3 ="3" heading4 ="4"/>
     <tags:column col1 ="v11" col2 ="v12" col3 ="v13" col4 ="v14"/>
     <tags:column col1 ="v21" col2 ="v22" col3 ="v23" col4 ="v24"/>
</table>
```

Phụ lục

Các phương thức của Servlet	
void	**destroy**() Called by the servlet container to indicate to a servlet that the servlet is being taken out of service.
ServletConfig	**getServletConfig**() Returns a ServletConfig object, which contains initialization and startup parameters for this servlet.
java.lang.String	**getServletInfo**() Returns information about the servlet, such as author, version, and copyright.
void	**init**(ServletConfig config) Called by the servlet container to indicate to a servlet that the servlet is being placed into service.
void	**service**(ServletRequest req, ServletResponse res) Called by the servlet container to allow the servlet to respond to a request.

Các phương thức của ServletRequest interface:	
java.lang.Object	**getAttribute**(java.lang.String name) Returns the value of the named attribute as an Object, or null if no attribute of the given name exists.
java.util.Enumeration	**getAttributeNames**() Returns an Enumeration containing the names of the attributes available to this request.
java.lang.String	**getCharacterEncoding**()

	Returns the name of the character encoding used in the body of this request.
int	**getContentLength**() Returns the length, in bytes, of the request body and made available by the input stream, or -1 if the length is not known.
java.lang.String	**getContentType**() Returns the MIME type of the body of the request, or null if the type is not known.
ServletInputStream	**getInputStream**() Retrieves the body of the request as binary data using a ServletInputStream.
java.util.Locale	**getLocale**() Returns the preferred Locale that the client will accept content in, based on the Accept-Language header.
java.util.Enumeration	**getLocales**() Returns an Enumeration of Locale objects indicating, in decreasing order starting with the preferred locale, the locales that are acceptable to the client based on the Accept-Language header.
java.lang.String	**getParameter**(java.lang.String name) Returns the value of a request parameter as a String, or null if the parameter does not exist.
java.util.Enumeration	**getParameterNames**() Returns an Enumeration of String objects containing the names of the parameters contained in this request.
java.lang.String[]	**getParameterValues**(java.lang.String name) Returns an array of String objects containing all of the values the given request parameter has, or null if the

	parameter does not exist.
java.lang.String	**getProtocol**() Returns the name and version of the protocol the request uses in the form *protocol/majorVersion.minorVersion*, for example, HTTP/1.1.
java.io.BufferedReader	**getReader**() Retrieves the body of the request as character data using a BufferedReader.
java.lang.String	**getRealPath**(java.lang.String path) **Deprecated.** *As of Version 2.1 of the Java Servlet API, use ServletContext.getRealPath(java.lang.String) instead.*
java.lang.String	**getRemoteAddr**() Returns the Internet Protocol (IP) address of the client that sent the request.
java.lang.String	**getRemoteHost**() Returns the fully qualified name of the client that sent the request, or the IP address of the client if the name cannot be determined.
RequestDispatcher	**getRequestDispatcher**(java.lang.String path) Returns a RequestDispatcher object that acts as a wrapper for the resource located at the given path.
java.lang.String	**getScheme**() Returns the name of the scheme used to make this request, for example, http, https, or ftp.
java.lang.String	**getServerName**() Returns the host name of the server that received the request.

int	**getServerPort**() Returns the port number on which this request was received.
boolean	**isSecure**() Returns a boolean indicating whether this request was made using a secure channel, such as HTTPS.
void	**removeAttribute**(java.lang.String name) Removes an attribute from this request.
void	**setAttribute**(java.lang.String name, java.lang.Object o) Stores an attribute in this request.

Các phương thức của ServletResponse interface:	
void	**flushBuffer**() Forces any content in the buffer to be written to the client.
int	**getBufferSize**() Returns the actual buffer size used for the response.
java.lang.String	**getCharacterEncoding**() Returns the name of the charset used for the MIME body sent in this response.
java.util.Locale	**getLocale**() Returns the locale assigned to the response.
ServletOutputStream	**getOutputStream**() Returns a ServletOutputStream suitable for writing binary data in the response.
java.io.PrintWriter	**getWriter**() Returns a PrintWriter object that can send character

	text to the client.
boolean	**isCommitted**() Returns a boolean indicating if the response has been committed.
void	**reset**() Clears any data that exists in the buffer as well as the status code and headers.
void	**setBufferSize**(int size) Sets the preferred buffer size for the body of the response.
void	**setContentLength**(int len) Sets the length of the content body in the response In HTTP servlets, this method sets the HTTP Content-Length header.
void	**setContentType**(java.lang.String type) Sets the content type of the response being sent to the client.
void	**setLocale**(java.util.Locale loc) Sets the locale of the response, setting the headers (including the Content-Type's charset) as appropriate.

Các phương thức của HttpRequest interface	
java.lang.String	**getAuthType**() Returns the name of the authentication scheme used to protect the servlet, for example, "BASIC" or "SSL," or `null` if the servlet was not protected.
java.lang.String	**getContextPath**() Returns the portion of the request URI that

	indicates the context of the request.
Cookie[]	**getCookies**() Returns an array containing all of the Cookie objects the client sent with this request.
long	**getDateHeader**(java.lang.String name) Returns the value of the specified request header as a long value that represents a Date object.
java.lang.String	**getHeader**(java.lang.String name) Returns the value of the specified request header as a String.
java.util.Enumeration	**getHeaderNames**() Returns an enumeration of all the header names this request contains.
java.util.Enumeration	**getHeaders**(java.lang.String name) Returns all the values of the specified request header as an Enumeration of String objects.
int	**getIntHeader**(java.lang.String name) Returns the value of the specified request header as an int.
java.lang.String	**getMethod**() Returns the name of the HTTP method with which this request was made, for example, GET, POST, or PUT.
java.lang.String	**getPathInfo**() Returns any extra path information associated with the URL the client sent when it made this request.
java.lang.String	**getPathTranslated**() Returns any extra path information after the servlet name but before the query string, and translates

	it to a real path.
java.lang.String	**getQueryString**() Returns the query string that is contained in the request URL after the path.
java.lang.String	**getRemoteUser**() Returns the login of the user making this request, if the user has been authenticated, or null if the user has not been authenticated.
java.lang.String	**getRequestedSessionId**() Returns the session ID specified by the client.
java.lang.String	**getRequestURI**() Returns the part of this request's URL from the protocol name up to the query string in the first line of the HTTP request.
java.lang.String	**getServletPath**() Returns the part of this request's URL that calls the servlet.
HttpSession	**getSession**() Returns the current session associated with this request, or if the request does not have a session, creates one.
HttpSession	**getSession**(boolean create) Returns the current HttpSession associated with this request or, if if there is no current session and create is true, returns a new session.
java.security.Principal	**getUserPrincipal**() Returns a java.security.Principal object containing the name of the current authenticated user.

boolean	**isRequestedSessionIdFromCookie**() Checks whether the requested session ID came in as a cookie.
boolean	**isRequestedSessionIdFromUrl**() **Deprecated.** *As of Version 2.1 of the Java Servlet API, use* *isRequestedSessionIdFromURL()* *instead.*
boolean	**isRequestedSessionIdFromURL**() Checks whether the requested session ID came in as part of the request URL.
boolean	**isRequestedSessionIdValid**() Checks whether the requested session ID is still valid.
boolean	**isUserInRole**(java.lang.String role) Returns a boolean indicating whether the authenticated user is included in the specified logical "role".

Các hằng số của HttpServletResponse

static int	**SC_ACCEPTED** Status code (202) indicating that a request was accepted for processing, but was not completed.
static int	**SC_BAD_GATEWAY** Status code (502) indicating that the HTTP server received an invalid response from a server it consulted when acting as a proxy or gateway.
static int	**SC_BAD_REQUEST** Status code (400) indicating the request sent by the client was syntactically incorrect.

static int	**SC_CONFLICT** Status code (409) indicating that the request could not be completed due to a conflict with the current state of the resource.
static int	**SC_CONTINUE** Status code (100) indicating the client can continue.
static int	**SC_CREATED** Status code (201) indicating the request succeeded and created a new resource on the server.
static int	**SC_EXPECTATION_FAILED** Status code (417) indicating that the server could not meet the expectation given in the Expect request header.
static int	**SC_FORBIDDEN** Status code (403) indicating the server understood the request but refused to fulfill it.
static int	**SC_GATEWAY_TIMEOUT** Status code (504) indicating that the server did not receive a timely response from the upstream server while acting as a gateway or proxy.
static int	**SC_GONE** Status code (410) indicating that the resource is no longer available at the server and no forwarding address is known.
static int	**SC_HTTP_VERSION_NOT_SUPPORTED** Status code (505) indicating that the server does not support or refuses to support the HTTP protocol version that was used in the request message.
static int	**SC_INTERNAL_SERVER_ERROR** Status code (500) indicating an error inside the HTTP server which prevented it from fulfilling the request.
static int	**SC_LENGTH_REQUIRED**

	Status code (411) indicating that the request cannot be handled without a defined *Content-Length*.
static int	**SC_METHOD_NOT_ALLOWED** Status code (405) indicating that the method specified in the *Request-Line* is not allowed for the resource identified by the *Request-URI*.
static int	**SC_MOVED_PERMANENTLY** Status code (301) indicating that the resource has permanently moved to a new location, and that future references should use a new URI with their requests.
static int	**SC_MOVED_TEMPORARILY** Status code (302) indicating that the resource has temporarily moved to another location, but that future references should still use the original URI to access the resource.
static int	**SC_MULTIPLE_CHOICES** Status code (300) indicating that the requested resource corresponds to any one of a set of representations, each with its own specific location.
static int	**SC_NO_CONTENT** Status code (204) indicating that the request succeeded but that there was no new information to return.
static int	**SC_NON_AUTHORITATIVE_INFORMATION** Status code (203) indicating that the meta information presented by the client did not originate from the server.
static int	**SC_NOT_ACCEPTABLE** Status code (406) indicating that the resource identified by the request is only capable of generating response entities which have content characteristics not acceptable according to the accept headerssent in the request.

static int	**SC_NOT_FOUND** Status code (404) indicating that the requested resource is not available.
static int	**SC_NOT_IMPLEMENTED** Status code (501) indicating the HTTP server does not support the functionality needed to fulfill the request.
static int	**SC_NOT_MODIFIED** Status code (304) indicating that a conditional GET operation found that the resource was available and not modified.
static int	**SC_OK** Status code (200) indicating the request succeeded normally.
static int	**SC_PARTIAL_CONTENT** Status code (206) indicating that the server has fulfilled the partial GET request for the resource.
static int	**SC_PAYMENT_REQUIRED** Status code (402) reserved for future use.
static int	**SC_PRECONDITION_FAILED** Status code (412) indicating that the precondition given in one or more of the request-header fields evaluated to false when it was tested on the server.
static int	**SC_PROXY_AUTHENTICATION_REQUIRED** Status code (407) indicating that the client *MUST* first authenticate itself with the proxy.
static int	**SC_REQUEST_ENTITY_TOO_LARGE** Status code (413) indicating that the server is refusing to process the request because the request entity is larger than the server is willing or able to process.
static int	**SC_REQUEST_TIMEOUT** Status code (408) indicating that the client did not produce a

	requestwithin the time that the server was prepared to wait.
static int	**SC_REQUEST_URI_TOO_LONG** Status code (414) indicating that the server is refusing to service the request because the *Request-URI* is longer than the server is willing to interpret.
static int	**SC_REQUESTED_RANGE_NOT_SATISFIABLE** Status code (416) indicating that the server cannot serve the requested byte range.
static int	**SC_RESET_CONTENT** Status code (205) indicating that the agent *SHOULD* reset the document view which caused the request to be sent.
static int	**SC_SEE_OTHER** Status code (303) indicating that the response to the request can be found under a different URI.
static int	**SC_SERVICE_UNAVAILABLE** Status code (503) indicating that the HTTP server is temporarily overloaded, and unable to handle the request.
static int	**SC_SWITCHING_PROTOCOLS** Status code (101) indicating the server is switching protocols according to Upgrade header.
static int	**SC_UNAUTHORIZED** Status code (401) indicating that the request requires HTTP authentication.
static int	**SC_UNSUPPORTED_MEDIA_TYPE** Status code (415) indicating that the server is refusing to service the request because the entity of the request is in a format not supported by the requested resource for the requested method.
static int	**SC_USE_PROXY** Status code (305) indicating that the requested resource *MUST*

	be accessed through the proxy given by the *Location* field.

Các phương thức của HttpServletResponse	
void	**addCookie**(Cookie cookie) Adds the specified cookie to the response.
void	**addDateHeader**(java.lang.String name, long date) Adds a response header with the given name and date-value.
void	**addHeader**(java.lang.String name, java.lang.String value) Adds a response header with the given name and value.
void	**addIntHeader**(java.lang.String name, int value) Adds a response header with the given name and integer value.
boolean	**containsHeader**(java.lang.String name) Returns a boolean indicating whether the named response header has already been set.
java.lang.String	**encodeRedirectUrl**(java.lang.String url) **Deprecated.** *As of version 2.1, use encodeRedirectURL(String url) instead*
java.lang.String	**encodeRedirectURL**(java.lang.String url) Encodes the specified URL for use in the sendRedirect method or, if encoding is not needed, returns the URL unchanged.
java.lang.String	**encodeUrl**(java.lang.String url) **Deprecated.** *As of version 2.1, use encodeURL(String url) instead*
java.lang.String	**encodeURL**(java.lang.String url) Encodes the specified URL by including the session ID in

	it, or, if encoding is not needed, returns the URL unchanged.
void	**sendError**(int sc) Sends an error response to the client using the specified status.
void	**sendError**(int sc, java.lang.String msg) Sends an error response to the client using the specified status code and descriptive message.
void	**sendRedirect**(java.lang.String location) Sends a temporary redirect response to the client using the specified redirect location URL.
void	**setDateHeader**(java.lang.String name, long date) Sets a response header with the given name and date-value.
void	**setHeader**(java.lang.String name, java.lang.String value) Sets a response header with the given name and value.
void	**setIntHeader**(java.lang.String name, int value) Sets a response header with the given name and integer value.
void	**setStatus**(int sc) Sets the status code for this response.
void	**setStatus**(int sc, java.lang.String sm) **Deprecated.** *As of version 2.1, due to ambiguous meaning of the message parameter. To set a status code use `setStatus(int)`, to send an error with a description use `sendError(int, String)`. Sets the status code and message for this response.*

Các phương thức của ServletConfig	
java.lang.String	**getInitParameter**(java.lang.String name) Returns a String containing the value of the named initialization parameter, or null if the parameter does not exist.
java.util.Enumeration	**getInitParameterNames**() Returns the names of the servlet's initialization parameters as an Enumeration of String objects, or an empty Enumeration if the servlet has no initialization parameters.
ServletContext	**getServletContext**() Returns a reference to the ServletContext in which the servlet is executing.
java.lang.String	**getServletName**() Returns the name of this servlet instance.

Các phương thức của Servlet Context	
java.lang.Object	**getAttribute**(java.lang.String name) Returns the servlet container attribute with the given name, or null if there is no attribute by that name.
java.util.Enumeration	**getAttributeNames**() Returns an Enumeration containing the attribute names available within this servlet context.
ServletContext	**getContext**(java.lang.String uripath) Returns a ServletContext object that corresponds to a specified URL on the server.
java.lang.String	**getInitParameter**(java.lang.String name) Returns a String containing the value of the named context-wide initialization parameter, or null if the

	parameter does not exist.
java.util.Enumeration	**getInitParameterNames**() Returns the names of the context's initialization parameters as an Enumeration of String objects, or an empty Enumeration if the context has no initialization parameters.
int	**getMajorVersion**() Returns the major version of the Java Servlet API that this servlet container supports.
java.lang.String	**getMimeType**(java.lang.String file) Returns the MIME type of the specified file, or null if the MIME type is not known.
int	**getMinorVersion**() Returns the minor version of the Servlet API that this servlet container supports.
RequestDispatcher	**getNamedDispatcher**(java.lang.String name) Returns a RequestDispatcher object that acts as a wrapper for the named servlet.
java.lang.String	**getRealPath**(java.lang.String path) Returns a String containing the real path for a given virtual path.
RequestDispatcher	**getRequestDispatcher**(java.lang.String path) Returns a RequestDispatcher object that acts as a wrapper for the resource located at the given path.
java.net.URL	**getResource**(java.lang.String path) Returns a URL to the resource that is mapped to a specified path.
java.io.InputStream	**getResourceAsStream**(java.lang.String path) Returns the resource located at the named path as an

	`InputStream` object.
`java.lang.String`	**`getServerInfo`**`()` Returns the name and version of the servlet container on which the servlet is running.
`Servlet`	**`getServlet`**`(java.lang.String name)` **Deprecated.** *As of Java Servlet API 2.1, with no direct replacement.* *This method was originally defined to retrieve a servlet from a `ServletContext`. In this version, this method always returns `null` and remains only to preserve binary compatibility. This method will be permanently removed in a future version of the Java Servlet API.* *In lieu of this method, servlets can share information using the `ServletContext` class and can perform shared business logic by invoking methods on common non-servlet classes.*
`java.util.Enumeration`	**`getServletNames`**`()` **Deprecated.** *As of Java Servlet API 2.1, with no replacement.* *This method was originally defined to return an `Enumeration` of all the servlet names known to this context. In this version, this method always returns an empty `Enumeration` and remains only to preserve binary compatibility. This method will be permanently removed in a future version of the Java Servlet API.*
`java.util.Enumeration`	**`getServlets`**`()` **Deprecated.** *As of Java Servlet API 2.0, with no replacement.* *This method was originally defined to return an `Enumeration` of all the servlets known to this servlet context. In this version, this method always returns an empty enumeration and remains only to preserve binary compatibility. This method will be permanently removed in a future version of the Java Servlet API.*
`void`	**`log`**`(java.lang.Exception exception, java.lang.String msg)` **Deprecated.** *As of Java Servlet API 2.1, use `log(String message, Throwable throwable)` instead.*

	This method was originally defined to write an exception's stack trace and an explanatory error message to the servlet log file.
void	**log**(java.lang.String msg) Writes the specified message to a servlet log file, usually an event log.
void	**log**(java.lang.String message, java.lang.Throwable throwable) Writes an explanatory message and a stack trace for a given Throwable exception to the servlet log file.
void	**removeAttribute**(java.lang.String name) Removes the attribute with the given name from the servlet context.
void	**setAttribute**(java.lang.String name, java.lang.Object object) Binds an object to a given attribute name in this servlet context.

Kết nối cơ sở dữ liệu

Trong mô hình MVC, model là nhân của chương trình, nó chứa business logic của cả ứng dụng. Một dự án được quản lý tốt thì không có business logic trong View và Controller. Nhiệm vụ của controller chỉ là nhận request, xử lý request, gọi model để thực hiện business logic rồi chọn view để thị dữ liệu.

Một ví dụ về UserBusiness kết nối sqlserver 2005

Lớp DBConnection:

Lớp này có nhiệm vụ tạo connection đến Database

```
private static String USER_NAME = "sa";
    private static String PASSWORD = "123456";
    private static String URL
="jdbc:sqlserver://localhost\\hai:1443;databaseName=test;";

    public static Connection getDBConnection() throws ClassNotFoundException,
SQLException, InstantiationException, IllegalAccessException {
        Connection con = null;
        Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver");
```

```
        con = DriverManager.getConnection(URL, USER_NAME, PASSWORD);
        return con;
    }
```

Trong đó Nếu SqlServer được cài dưới dạng instance thì server name sẽ có định dạng là: `server-name\instance-name.`

1443 là cổng TCP-IP cấu hình trong SqlServer (nếu công này chưa khai báo thì phải khai báo thì mới connect được).

Lớp UserBusiness

Lớp này chứa tất cả các nghiệp vụ liên quan đến bảng User trong Database như INSERT, SELECT, UPDATE, DELETE...

Ví dụ phương thức ***checkLogin*** được gọi trong form login để kiểm tra user/password đăng nhập.

```
public static boolean checkLogin(User user) throws ClassNotFoundException,
SQLException, InstantiationException, IllegalAccessException {
        Connection conn = DBConnection.getDBConnection();
        PreparedStatement st = conn.prepareStatement("SELECT * FROM [USER]
                               WHERE username = ? AND password = ?");
        int i = 0;
        st.setString(++i, user.getUsername());
        st.setString(++i, Encryptation.crypt(user.getPassword()));

        int count = 0;
        ResultSet rs = st.executeQuery();
        while (rs.next()) {
            count++;
        }
        return count == 1 ? true : false;
    }
```

Chú ý: `Encryptation.crypt(user.getPassword()` thực hiện mã hóa password sử dụng MD5 128bit. Về nguyên tắc, kể cả người lập trình cũng không được biết các thông tin nhạy cảm như password của khách hàng. Vì vậy password được lưu trong Database phải được mã hóa theo 1 thuật toán đáng tin cậy.

Đoạn mã thực hiện mã hóa như sau:

```
public static String crypt(String str) {
        if (str == null || str.length() == 0) {
            throw new IllegalArgumentException("String to encrypt cannot be
null or zero length");
        }
        StringBuffer hexString = new StringBuffer();
        try {
```

```
            MessageDigest md = MessageDigest.getInstance("MD5");
            md.update(str.getBytes());
            byte[] hash = md.digest();
            for (int i = 0; i < hash.length; i++) {
                if ((0xff & hash[i]) < 0x10) {
                    hexString.append("0" + Integer.toHexString((0xFF &
hash[i])));
                } else {
                    hexString.append(Integer.toHexString(0xFF & hash[i]));
                }
            }
        } catch (NoSuchAlgorithmException e) {
            e.printStackTrace();
        }
        return hexString.toString();
    }
```

Có thể test kết quả bằng cách thêm vào hàm main như bên dưới đây:

```
    public static void main(String[] a) {
        System.out.println(crypt("abc"));
        System.out.println(crypt("aaaaaaa"));
    }
```

Lấy danh mục các User để áp dụng vào User Search form chẳng hạn, ta có thể sử dụng phương thức bên dưới. Phương thức này cho search chính xác và search gần đúng tùy vào biến wholeWord truyền vào.

```
public static List<User> searchUserByName(String username, boolean wholeWord)
throws SQLException, ClassNotFoundException, InstantiationException,
IllegalAccessException {
        String query = "SELECT username, role, description FROM [USER] WHERE
username = ?";
        String condition = username;
        ArrayList list = new ArrayList();
        if (!wholeWord) {
            query = "SELECT username, role, description FROM [USER] WHERE
username LIKE ?";
            condition = username + "%";
        }
        Connection conn = DBConnection.getDBConnection();
        PreparedStatement st = conn.prepareStatement(query);
        st.setString(1, condition);
        ResultSet rs = st.executeQuery();
        while (rs.next()) {
            User user = new User();
            user.setUsername(rs.getString("username"));
            user.setRole(rs.getString("role"));
            user.setDescription(rs.getString("description"));
            list.add(user);
        }
        return list;
    }
```

JavaBean User.java

Đóng vai trò là DTO – Data Transfer Object. Nó bao gồm các thuộc tính để thể hiện từng trường trong bảng User và các method get/set tương ứng:

```
private String username;
private String password;
private String role;
private String description;
```

Gọi Business trong controller/servlet

Thay vì gán hardcode:

```
loginForm. getUsername().equals("adam") &&
loginForm.gePassword().equals("12345678")....
```

Ta có thể gọi UserBusiness để xác thực login như sau:

```
User user = new User();
user.setUsername(loginForm.getUsername());
user.setPassword(loginForm.gePassword());

if (checkLogin(user)) {
	//login successfully
} else {
	//login failed, show error if necessary
}
```

Code sample

UserBusiness.rar