深入浅出JavaWeb

极客学院出版

前言

Java Web，是用Java技术来解决相关web互联网领域的技术总和。web包括：web服务器和web客户端两部分。

学习前提

在学习本教程之前建议先对Java、HTML、CSS、JS、ajax、HTTP、Firebug有所了解。

- linux下编写程序。
- 文件均是UTF-8编码。

相关软件的安装

jdk 1.8

netbeans 8.0

tomcat 8.0

更新日期	更新内容
2015-11-13	深入浅出Java Web

目录

1

JSP & Servlet

理解HTTP

HTTP是基于TCP协议的。TCP负责数据传输，而HTTP只是规范了TCP传输的数据的格式，而这个具体的格式，请见后面给出的资料。

HTTP服务的底层实现就是socket编程。

下面基于socket编写一个简单的HTTP server。

```
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.concurrent.Executors;
import java.util.concurrent.ExecutorService;

class SocketHandler implements Runnable {

    final static String CRLF = "\r\n";   // 1

    private Socket clientSocket;

    public SocketHandler(Socket clientSocket) {
        this.clientSocket = clientSocket;
    }

    public void handleSocket(Socket clientSocket) throws IOException {

        BufferedReader in = new BufferedReader(
                new InputStreamReader(clientSocket.getInputStream())
                );
        PrintWriter out = new PrintWriter(
                new BufferedWriter( new OutputStreamWriter(clientSocket.getOutputStream())),
                true
                );

        String requestHeader = "";
        String s;
        while ((s = in.readLine()) != null) {
            s += CRLF;  // 2 很重要，默认情况下in.readLine的结果中`\r\n`被去掉了
            requestHeader = requestHeader + s;
            if (s.equals(CRLF)){ // 3 此处HTTP请求头我们都得到了；如果从请求头中判断有请求正文，则还需要继续获取数据
                break;
            }
        }
        System.out.println("客户端请求头：");
        System.out.println(requestHeader);

        String responseBody = "客户端的请求头是：\n"+requestHeader;

        String responseHeader = "HTTP/1.0 200 OK\r\n" +
                "Content-Type: text/plain; charset=UTF-8\r\n" +
```

```
                "Content-Length: "+responseBody.getBytes().length+"\r\n" +
                "\r\n";
        // 4 问题来了：1、浏览器如何探测编码 2、浏览器受到content-length后会按照什么方式判断？汉字的个数？字节数？

        System.out.println("响应头：");
        System.out.println(responseHeader);



        out.write(responseHeader);
        out.write(responseBody);
        out.flush();

        out.close();
        in.close();
        clientSocket.close();

    }

    @Override
    public void run() {
        try {
            handleSocket(clientSocket);
        } catch(Exception ex) {
            ex.printStackTrace();
        }
    }

}

public class MyHTTPServer {

    public static void main(String[] args) throws Exception {

        int port = 8000;
        ServerSocket serverSocket = new ServerSocket(port);
        System.out.println("启动服务，绑定端口： " + port);

        ExecutorService fixedThreadPool = Executors.newFixedThreadPool(30);  // 5

        while (true) {  // 6
            Socket clientSocket = serverSocket.accept();
            System.out.println("新的连接"
                    + clientSocket.getInetAddress() + ":" + clientSocket.getPort());
            try {
                fixedThreadPool.execute(new SocketHandler(clientSocket));
            } catch (Exception e) {
                System.out.println(e);
            }
        }
    }
}
```

这是一个实现HTTP 1.0的服务器，对于所有的HTTP请求，会把HTTP请求头响应回去。 这个程序说明了web服务器处理请求的基本流程，JSP、Servlet、Spring MVC等只是在 这个基础上嫁了许多方法，以让我们更方面的编写web应用。web服务器不仅可以基于多线程， 也可以基于多进程、Reactor模型等。

测试程序：

运行上面的程序。我们使用curl访问 `http://127.0.0.1` （也可以使用浏览器）：

```
$ curl -i http://127.0.0.1:8000
HTTP/1.0 200 OK
Content-Type: text/plain; charset=UTF-8
Content-Length: 106

客户端的请求头是：
GET / HTTP/1.1
User-Agent: curl/7.35.0
Host: 127.0.0.1:8000
Accept: */*
```

Java程序输出：

```
启动服务，绑定端口： 8000
新的连接/127.0.0.1:36463
新的连接/127.0.0.1:36463客户端请求头：
GET / HTTP/1.1
User-Agent: curl/7.35.0
Host: 127.0.0.1:8000
Accept: */*


响应头：
HTTP/1.0 200 OK
Content-Type: text/plain; charset=UTF-8
Content-Length: 106
```

程序解析：

`// 1`：定义了HTTP头的换行符。

`// 2`：in.readLine()的结果默认不带换行符，这里把它加上。（这不是强制的，主要看你的程序逻辑需不需要， 这个程序的目标是把HTTP请求头响应回去）。

`// 3`：此时s是一个空行，根据HTTP协议，整个请求头都得到了。

`// 4`：Content-Length的值是字节的数量。

`// 5`：线程池。

`// 6`：这个循环不停监听socket连接，使用SocketHandler处理连入的socket，而这个处理是放在线程池中的。

HTTP 1.1：

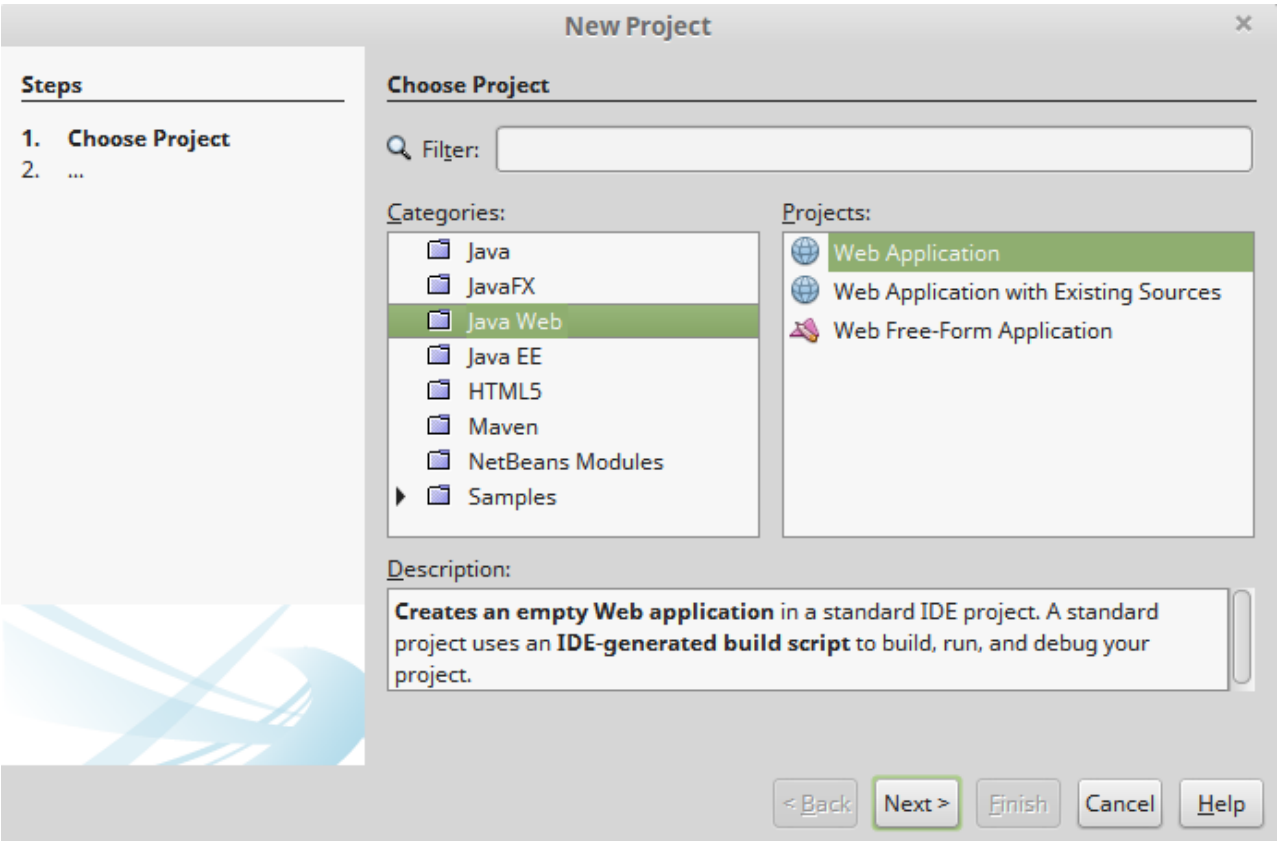
HTTP 1.1也是在这个思路的基础上实现的，即多个HTTP请求都在一个TCP连接中传输。对于HTTP 1.1，如何区分出每个HTTP请求很重要， 比较简单的可以是用过`Content-Length`判断一条请求是否结束。如果一个HTTP请求数据较多，往往采用Chunked方式， 可以参考Chunked transfer encoding。

从JSP开始

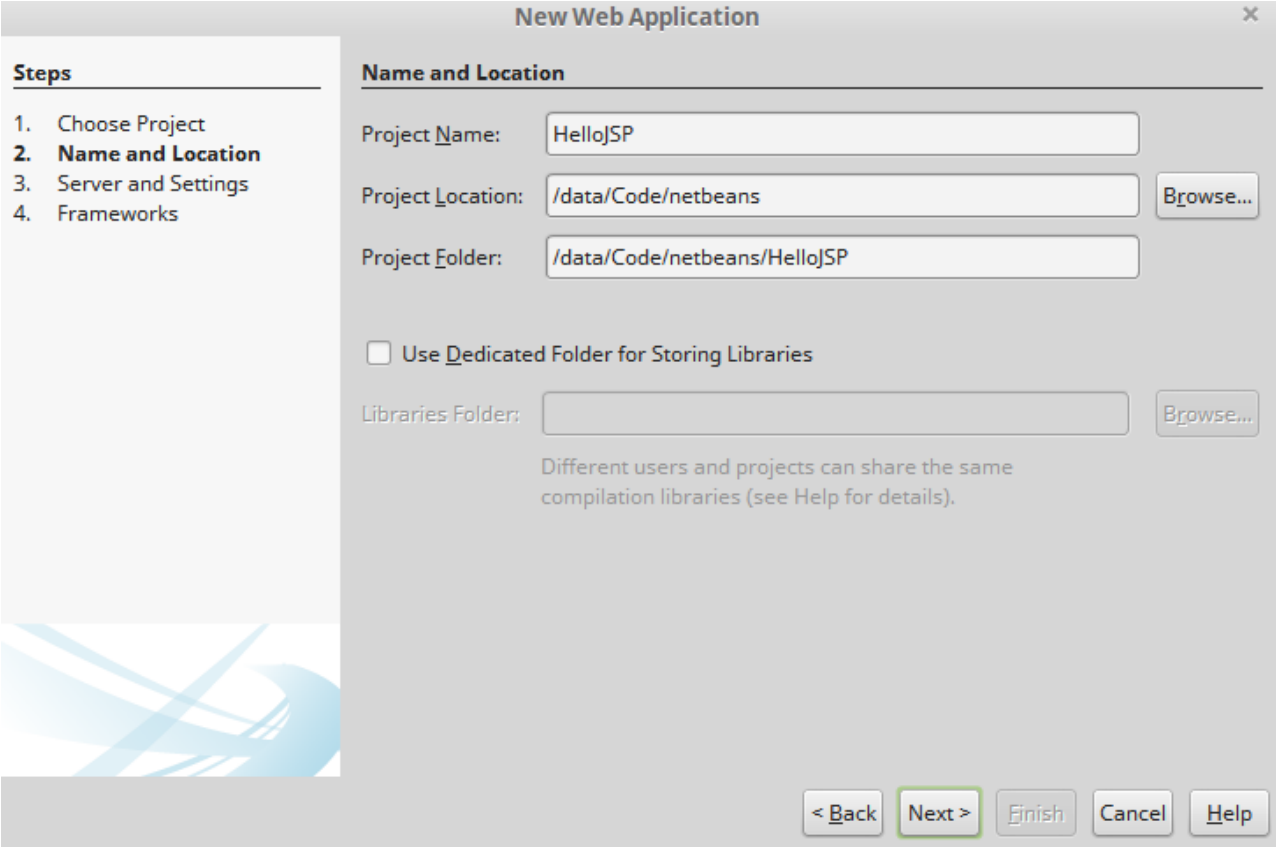
创建web项目

打开netbeans，在菜单栏依次选择“File”、“New Project...”，然后

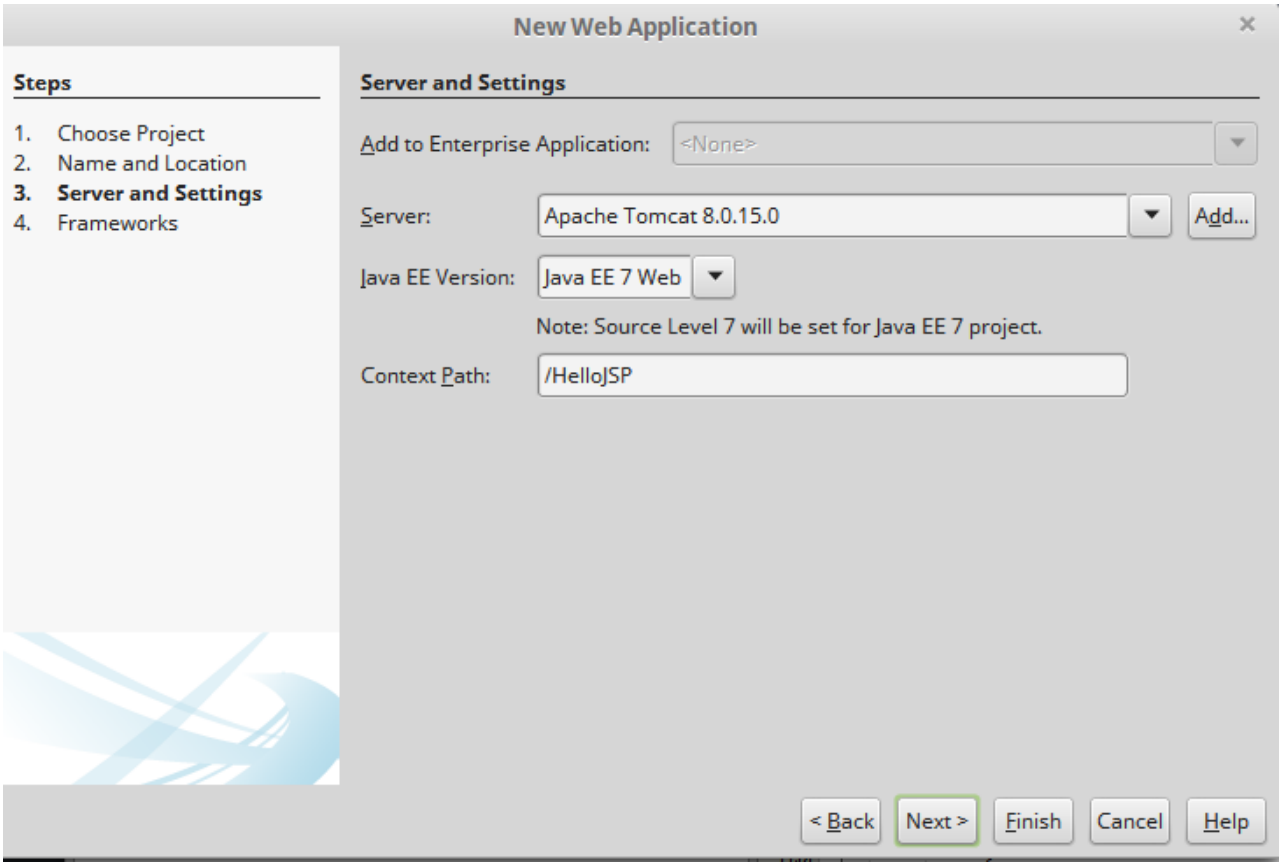
1、选择java web：



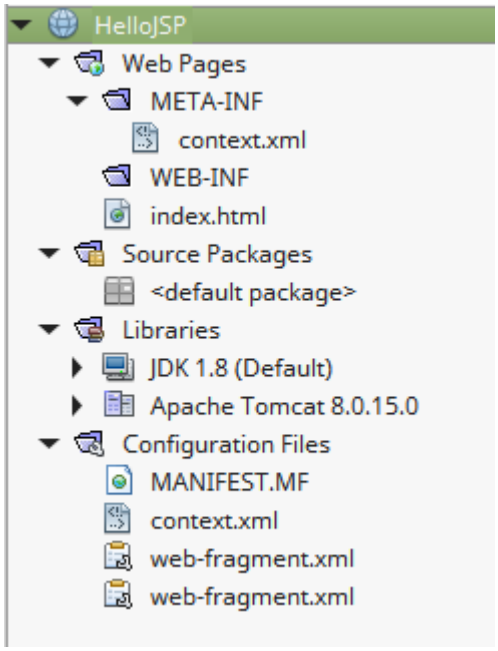
2、项目名称、路径：



3、选择使用的web容器，编写ContextPath：



生成的web项目结构如下：



index.html的内容如下：

```
<!DOCTYPE html>
<!--
To change this license header, choose License Headers in Project Properties.
To change this template file, choose Tools | Templates
and open the template in the editor.
-->
<html>
    <head>
        <title>TODO supply a title</title>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
    </head>
    <body>
        <div>TODO write content</div>
    </body>
</html>
```

`context.xml` 的内容如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<Context antiJARLocking="true" path="/HelloJSP"/>
```

`apache-tomcat-8.0.15/conf/server.xml` 中关于端口的配置如下：

```
<Connector port="8080" protocol="HTTP/1.1"
       connectionTimeout="20000"
       redirectPort="8443" />
```

运行项目后，netbeans的输出信息如下：

```
ant -f /data/Code/netbeans/HelloJSP -Dnb.internal.action.name=run -Ddirectory.deployment.supported=true -DforceRedeploy
init:
deps-module-jar:
deps-ear-jar:
deps-jar:
library-inclusion-in-archive:
library-inclusion-in-manifest:
compile:
compile-jsps:
Incrementally deploying http://localhost:8084/HelloJSP
Completed incremental distribution of http://localhost:8084/HelloJSP
run-deploy:
Browsing: http://localhost:8084/HelloJSP
run-display-browser:
run:
```

使用浏览器访问 `http://localhost:8084/HelloJSP/` ，能看到 `TODO write content` 。

那么问题来了，为什么配置的是8080,访问时却用8084端口？

答案是：这个端口是netbeans启动tomcat时配置的。

看一下启动命令：

```
$ ps -ef | grep 'tomcat' | grep -v grep
letian     390 24314  0 09:24 ?        00:00:08 /data/Apps/jdk1.8.0_20/bin/java -Djava.util.logging.config.file=/home/

$ grep -r '8084' /home/letian/.netbeans/8.0.2 # 可以看到很多结果
...
```

也就是说，实际使用的是 `/home/letian/.netbeans/8.0.2/apache-tomcat-8.0.15.0_base/conf` 下的配置。在``/home/letian/.netbeans/8.0.2/apache-tomcat-8.0.15.0_base/conf/Catalina/localhost/HelloJSP.xml`中有以下内容：

```
<Context antiJARLocking="true" docBase="/data/Code/netbeans/HelloJSP/build/web" path="/HelloJSP"/>
```

这意味着访问 `http://localhost:8084/HelloJSP/` 时对应的web应用部署在HelloJSP项目 的 `build/web/` 目录下。

基于JSP的hello world

删除index.html，新建index.jsp，内容如下：

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>JSP Page</title>
    </head>
```

```
    <body>
        <%
            String data="hello world";
            boolean flag=true;
            if (flag==true) {
                out.println("<h1>" +data.toUpperCase()+ "</h1>");
            }
        %>
    </body>
</html>
```

运行项目，访问 `http://localhost:8084/HelloJSP/` ，可以看到 `HELLO WORLD` 。

JSP在部署时会被转换成servlet， `/home/letian/.netbeans/8.0.2/apache-tomcat-8.0.15.0_base/work/Catalina/localhost/HelloJSP/org/apache/jsp` 中的 `index_jsp.java` 就是对应的servlet。其内容如下：

```
/*
 * Generated by the Jasper component of Apache Tomcat
 * Version: Apache Tomcat/8.0.15
 * Generated at: 2015-09-18 02:43:43 UTC
 * Note: The last modified time of this file was set to
 *       the last modified time of the source file after
 *       generation to assist with modification tracking.
 */
package org.apache.jsp;

import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.jsp.*;

public final class index_jsp extends org.apache.jasper.runtime.HttpJspBase
    implements org.apache.jasper.runtime.JspSourceDependent {

  private static final javax.servlet.jsp.JspFactory _jspxFactory =
          javax.servlet.jsp.JspFactory.getDefaultFactory();

  private static java.util.Map<java.lang.String,java.lang.Long> _jspx_dependants;

  private javax.el.ExpressionFactory _el_expressionfactory;
  private org.apache.tomcat.InstanceManager _jsp_instancemanager;

  public java.util.Map<java.lang.String,java.lang.Long> getDependants() {
    return _jspx_dependants;
  }

  public void _jspInit() {
    _el_expressionfactory = _jspxFactory.getJspApplicationContext(getServletConfig().getServletContext()).getExpressio
    _jsp_instancemanager = org.apache.jasper.runtime.InstanceManagerFactory.getInstanceManager(getServletConfig());
  }

  public void _jspDestroy() {
  }

  public void _jspService(final javax.servlet.http.HttpServletRequest request, final javax.servlet.http.HttpServletRes
        throws java.io.IOException, javax.servlet.ServletException {

final java.lang.String _jspx_method = request.getMethod();
if (!"GET".equals(_jspx_method) && !"POST".equals(_jspx_method) && !"HEAD".equals(_jspx_method) && !javax.servlet.Disp
response.sendError(HttpServletResponse.SC_METHOD_NOT_ALLOWED, "JSPs only permit GET POST or HEAD");
return;
}
```

```
    final javax.servlet.jsp.PageContext pageContext;
    javax.servlet.http.HttpSession session = null;
    final javax.servlet.ServletContext application;
    final javax.servlet.ServletConfig config;
    javax.servlet.jsp.JspWriter out = null;
    final java.lang.Object page = this;
    javax.servlet.jsp.JspWriter _jspx_out = null;
    javax.servlet.jsp.PageContext _jspx_page_context = null;


    try {
      response.setContentType("text/html;charset=UTF-8");
      pageContext = _jspxFactory.getPageContext(this, request, response,
                  null, true, 8192, true);
      _jspx_page_context = pageContext;
      application = pageContext.getServletContext();
      config = pageContext.getServletConfig();
      session = pageContext.getSession();
      out = pageContext.getOut();
      _jspx_out = out;

      out.write("\n");
      out.write("<!DOCTYPE html>\n");
      out.write("<html>\n");
      out.write("    <head>\n");
      out.write("        <meta http-equiv=\"Content-Type\" content=\"text/html; charset=UTF-8\">\n");
      out.write("        <title>JSP Page</title>\n");
      out.write("    </head>\n");
      out.write("    <body>\n");
      out.write("        ");

            String data="hello world";
            boolean flag=true;
            if (flag==true) {
                out.println("<h1>" +data.toUpperCase()+ "</h1>");
            }

      out.write("\n");
      out.write("    </body>\n");
      out.write("</html>\n");
    } catch (java.lang.Throwable t) {
      if (!(t instanceof javax.servlet.jsp.SkipPageException)){
        out = _jspx_out;
        if (out != null && out.getBufferSize() != 0)
          try {
            if (response.isCommitted()) {
              out.flush();
            } else {
              out.clearBuffer();
            }
          } catch (java.io.IOException e) {}
        if (_jspx_page_context != null) _jspx_page_context.handlePageException(t);
        else throw new ServletException(t);
      }
    } finally {
      _jspxFactory.releasePageContext(_jspx_page_context);
    }
  }
}
```

感受一下这段代码吧～

关于JSP就介绍这么多。**需要记住的是：JSP最合适的用途是用作MVC中的视图，而不是和HTML一起混合编码 （例如把从数据库拉取数据也放入JSP中写）**

理解Servlet

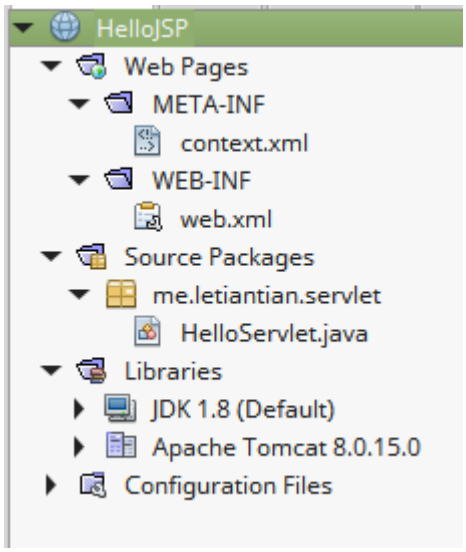
本文依然使用00-03、从JSP开始中创建的项目HelloJSP。

本文主要有以下内容：

- 如何使用Servlet编写Hello Servlet
- 如何将Servlet与URL对应起来
- Servlet如何调用JSP
- Servlet如何返回JSON数据
- 如何编写一个Dispatcher

Hello Servlet

项目结构如下：



`HelloServlet.java` 内容如下：

```
package me.letiantian.servlet;

import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class HelloServlet extends HttpServlet {
```

```
    protected void processRequest(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        try (PrintWriter out = response.getWriter()) {
            out.println("<!DOCTYPE html>");
            out.println("<html>");
            out.println("<head>");
            out.println("<title>Servlet HelloServlet</title>");
            out.println("</head>");
            out.println("<body>");
            out.println("<h1>Servlet HelloServlet at " + request.getContextPath() + "</h1>");
            out.println("</body>");
            out.println("</html>");
        }
    }

    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        processRequest(request, response);
    }

    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        processRequest(request, response);
    }

}
```

HTTP最常见的方法是GET和POST，在一个Servlet中对应的处理方法分别是doGet()和doPost()。 `response.setContentType("text/html;charset=UTF-8");` 用来设置HTTP响应头中的Content-Type。 PrintWriter对象out的输出内容则是响应正文。

`web.xml` 内容如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.1" xmlns="http://xmlns.jcp.org/xml/ns/javaee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    <servlet>
        <servlet-name>HelloServlet</servlet-name>
        <servlet-class>me.letiantian.servlet.HelloServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>HelloServlet</servlet-name>
        <url-pattern>/HelloServlet</url-pattern>
    </servlet-mapping>
    <session-config>
        <session-timeout>
            30
        </session-timeout>
    </session-config>
</web-app>
```

在这个配置中， `me.letiantian.servlet.HelloServlet` 与URL `/HelloServlet` 对应。 `session-timeout` 设置了session的有效时间，单位是分钟（不过目前的程序里还没用过session）。

浏览器访问 `http://127.0.0.1:8084/HelloJSP` 会显示404；访问 `http://127.0.0.1:8084/HelloJSP/HelloServlet` 会显示 `Servlet HelloServlet at /HelloJSP` ，这也正是 `me.letiantian.servlet.HelloServlet` 输出的HTML的渲染结果。

也可以使用注解将Servlet和URL对应起来

首先清空web.xml中关于URL的配置，web.xml最终内容如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.1" xmlns="http://xmlns.jcp.org/xml/ns/javaee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance

    <session-config>
        <session-timeout>
            30
        </session-timeout>
    </session-config>

</web-app>
```

然后对 `me.letiantian.servlet.HelloServlet` 类略做修改：

```
package me.letiantian.servlet;

import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import javax.servlet.annotation.WebServlet;

@WebServlet("/HelloServlet")
public class HelloServlet extends HttpServlet {
    // ......
}
```

重新启动项目，浏览器访问效果和之前是相同的。

Servlet调用JSP

改写me.letiantian.servlet.HelloServlet类，内容如下：

```
package me.letiantian.servlet;

import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
```

```
import javax.servlet.annotation.WebServlet;

@WebServlet("/HelloServlet")
public class HelloServlet extends HttpServlet {

    protected void processRequest(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        request.setAttribute("title", "Hello Servlet");
        request.setAttribute("content", "你好");
        RequestDispatcher rd = request.getRequestDispatcher("/WEB-INF/jsp/hello.jsp");
        rd.forward(request, response);
    }

    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        processRequest(request, response);
    }

    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        processRequest(request, response);
    }

}
```

在 `WEB-INF/` 下创建目录 `jsp`，然后在 `jsp` 目录下新建 `hello.jsp`，内容如下：

```
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>${title}</title>
    </head>
    <body>
        <h1>${content}</h1>
    </body>
</html>
```

重启该项目，访问 `http://127.0.0.1:8084/HelloJSP/HelloServlet`：

```
$ curl -i  http://127.0.0.1:8084/HelloJSP/HelloServlet
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Set-Cookie: JSESSIONID=7CCCFD5467F8330066F827623802FB23; Path=/HelloJSP/; HttpOnly
Content-Type: text/html;charset=UTF-8
Content-Length: 215
Date: Fri, 18 Sep 2015 08:09:58 GMT


<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>Hello Servlet</title>
    </head>
    <body>
        <h1>你好</h1>
```

```
    </body>
</html>
```

CSS等静态文件放在什么地方

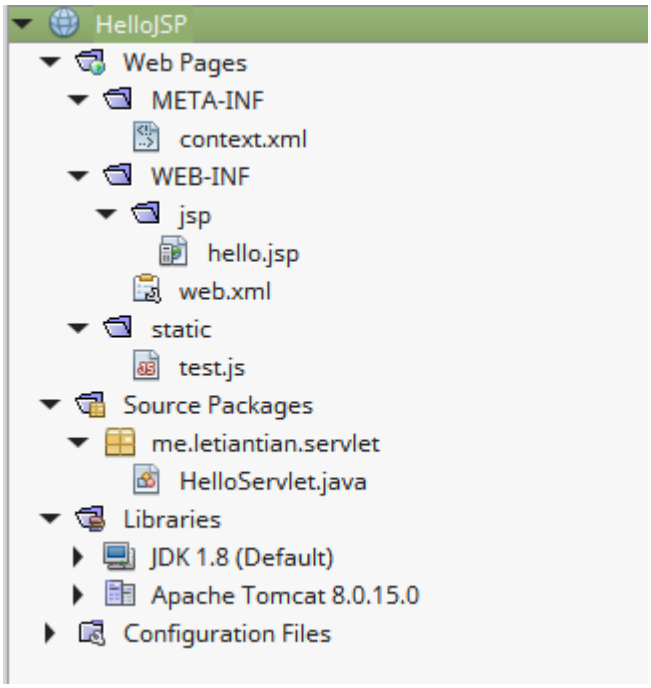
在项目下建立static目录，再这个目录下添加 `test.js`，内容如下：

```
console.log("hello world");
```

在 `web.xml` 添加以下内容：

```
<servlet-mapping>
    <servlet-name>default</servlet-name>
    <url-pattern>*.jpg</url-pattern>
</servlet-mapping>

<servlet-mapping>
    <servlet-name>default</servlet-name>
    <url-pattern>*.png</url-pattern>
</servlet-mapping>

<servlet-mapping>
    <servlet-name>default</servlet-name>
    <url-pattern>*.js</url-pattern>
</servlet-mapping>

<servlet-mapping>
    <servlet-name>default</servlet-name>
    <url-pattern>*.css</url-pattern>
</servlet-mapping>
```

此时，项目结构如下：



启动项目，访问

```
$ curl -i http://localhost:8084/HelloJSP/static/test.js
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Accept-Ranges: bytes
ETag: W/"27-1442566151000"
Last-Modified: Fri, 18 Sep 2015 08:49:11 GMT
Content-Type: application/javascript
Content-Length: 27
Date: Fri, 18 Sep 2015 08:58:00 GMT

console.log("hello world");
```

```
$ curl -i http://localhost:8084/HelloJSP/static/test.js?time=123
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Accept-Ranges: bytes
ETag: W/"27-1442566151000"
Last-Modified: Fri, 18 Sep 2015 08:49:11 GMT
Content-Type: application/javascript
Content-Length: 27
Date: Fri, 18 Sep 2015 08:58:09 GMT

console.log("hello world");
```

Servlet如何返回JSON数据

将

```
response.setContentType("text/html;charset=UTF-8");
```

修改为

```
response.setContentType("application/json;charset=UTF-8");
```

。

`out.println`输出JSON格式的字符串即可。

编写Dispatcher

基于以上的学习，已经可以编写一个分发器了。 将HelloServlet.java修改为DispatcherServlet.java，内容修改为：

```
package me.letiantian.servlet;

import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
```

```
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import javax.servlet.annotation.WebServlet;

@WebServlet("/")
public class HelloServlet extends HttpServlet {

    protected void processRequest(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        response.setContentType("text/plain;charset=UTF-8");
        try (PrintWriter out = response.getWriter()) {
            out.println("context: " + request.getContextPath());
            out.println("request uri: " + request.getRequestURI());
            out.println("params: " + request.getParameterMap());
        }
    }

    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        processRequest(request, response);
    }

    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        processRequest(request, response);
    }

}
```

运行项目，访问结果如下：

```
$ curl http://localhost:8084/HelloJSP/user
context: /HelloJSP
request uri: /HelloJSP/user
params: {}
```

```
$ curl http://localhost:8084/HelloJSP/user?name=letian context: /HelloJSP request uri: /HelloJSP/user par
ams: {name=[Ljava.lang.String;@49ea47b4}
```

```
$ curl http://localhost:8084/HelloJSP/static/test.js
console.log("hello world");
```

（这个代码并没什么用～）

从这段代码中可以看到，我们可以通过request对象得到HTTP请求信息，特别是request URI。在这个程序的基础上，我们 可以继续扩充它，使得其遇到某个URI，就调用指定的处理函数。慢慢地补充，一个框架就出来了。

过滤器与监听器

过滤器

过滤器（Filter），并非必须，但很实用。

过滤器是一种设计模式，主要用来封装Servlet中一些通用的代码。在web.xml中配置哪些URL对应哪些过滤器。

一个过滤器的写法如下：

```
public void doFilter(ServletRequest request , ServletResponse response , FilterChain chain) {
    //处理 request
    chain.doFilter(request, response);
    //处理 response
}
```

假设针对一URL定义了3个过滤器，分别是MyFilter1、MyFilter2、MyFilter3，在web.xml中也是按照这个顺序设置的， 那么过滤器和Servlet的执行顺序如下：

- MyFilter1中处理request的代码;
- MyFilter2中处理request的代码;
- MyFilter3中处理request的代码;
- 相应的Servlet;
- MyFilter3中处理response的代码;
- MyFilter2中处理response的代码;
- MyFilter1中处理response的代码;

之所以能达到这样的效果， `chain.doFilter(request, response);` 起到了很大的作用。 值得注意的是，如果每个Filter没有到达 `chain.doFilter` 就返回了，那么后续的Filter或者Servlet也就不会执行。

监听器

当某个事件发生时候，监听器里的方法会被调用。例如Tomcat容器启动时、销毁时，session创建时、销毁时。

使用 velocity 模板引擎

> Velocity is a Java-based template engine. It permits anyone to use a simple yet powerful template language to reference objects defined in Java code.
>
> When Velocity is used for web development, Web designers can work in parallel with Java programmers to develop web sites according to the Model-View-Controller (MVC) model, meaning that web page designers can focus solely on creating a site that looks good, and programmers can focus solely on writing top-notch code. Velocity separates Java code from the web pages, making the web site more maintainable over its lifespan and providing a viable alternative to Java Server Pages (JSPs) or PHP.

以上内容摘自velocity的官方首页。

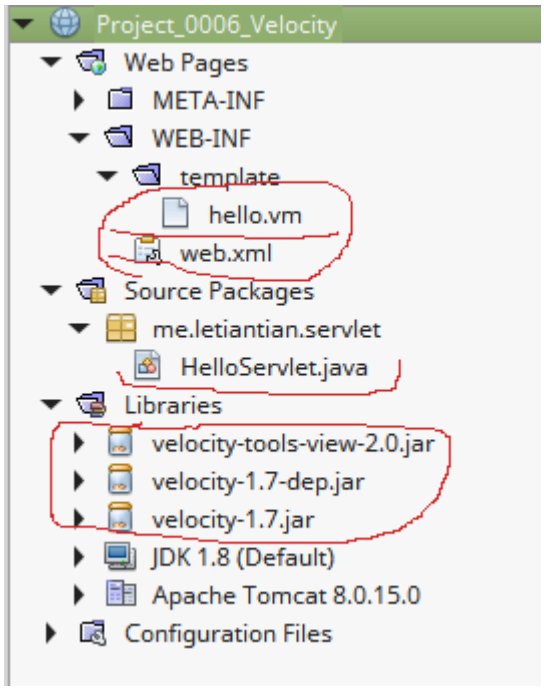
以下通过示例来说明velocity的使用。

项目结构

在http://velocity.apache.org/download.cgi中下载velocity-1.7、velocity-tools-2.0。

参考00-03、从JSP开始所述，创建项目`Project_0006_Velocity`，导入相关的jar，编写代码。

项目结构如下：



图片 1.7 项目结构

对于新增的jar，放到/WEB-INF/lib目录即可。但当多个webApp要使用时，放入CLASSPATH或Servlet容器（如Tomcat）的顶层lib是最好的选择.

代码

web.xml（在这一节，该文件可以忽略）：

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.1" xmlns="http://xmlns.jcp.org/xml/ns/javaee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

    <servlet-mapping>
        <servlet-name>default</servlet-name>
        <url-pattern>*.jpg</url-pattern>
    </servlet-mapping>

    <servlet-mapping>
        <servlet-name>default</servlet-name>
        <url-pattern>*.png</url-pattern>
    </servlet-mapping>

    <servlet-mapping>
        <servlet-name>default</servlet-name>
        <url-pattern>*.js</url-pattern>
    </servlet-mapping>

    <servlet-mapping>
        <servlet-name>default</servlet-name>
        <url-pattern>*.css</url-pattern>
    </servlet-mapping>

    <session-config>
```

```
        <session-timeout>
            30
        </session-timeout>
    </session-config>

</web-app>
```

hello.vm:

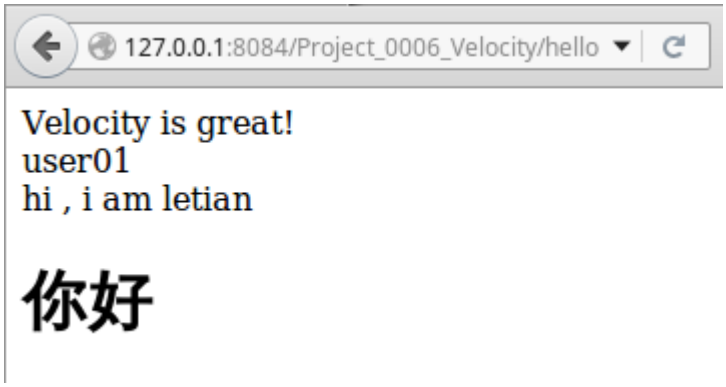
```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
</head>
<body>
 #set( $this = "Velocity")

$this is great!  <br/>
$name  <br/>
hi  , i am letian
<h1>你好</h1>
</body>
</html>
```

HelloServlet.java:

```
package me.letiantian.servlet;

import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import java.util.Properties;
import java.io.StringWriter;
import org.apache.velocity.app.Velocity;
import org.apache.velocity.app.VelocityEngine;
import org.apache.velocity.VelocityContext;


@WebServlet(name = "HelloServlet", urlPatterns = {"/hello"})
public class HelloServlet extends HttpServlet {

    protected void processRequest(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();

        Properties properties=new Properties();
        properties.setProperty("resource.loader", "webapp");
        properties.setProperty("webapp.resource.loader.class", "org.apache.velocity.tools.view.servlet.WebappLoader");
        properties.setProperty("webapp.resource.loader.path", "/WEB-INF/template");
        properties.setProperty(Velocity.ENCODING_DEFAULT, "UTF-8");
        properties.setProperty(Velocity.INPUT_ENCODING, "UTF-8");
        properties.setProperty(Velocity.OUTPUT_ENCODING, "UTF-8");
        VelocityEngine velocityEngine = new VelocityEngine(properties);
        velocityEngine.setApplicationAttribute("javax.servlet.ServletContext", request.getServletContext());

        VelocityContext context=new VelocityContext();
        context.put("name", "user01");
```

```
        StringWriter sw = new StringWriter();
        velocityEngine.mergeTemplate("hello.vm", "utf-8", context, sw);
//        velocityEngine.mergeTemplate("hello.vm", "utf-8", context, sw);      //如果这行不注释，hello.vm的内容会出现两
        out.println(sw.toString());

    }

    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        processRequest(request, response);
    }

    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        processRequest(request, response);
    }

}
```

运行项目，用浏览器访问 `http://127.0.0.1:8084/Project_0006_Velocity/hello` ：



改进上面的代码

在WEB-INF目录下创建 `velocity.properties` 文件，其内容如下：

```
resource.loader=webapp
webapp.resource.loader.class=org.apache.velocity.tools.view.servlet.WebappLoader
webapp.resource.loader.path=/WEB-INF/template/
input.encoding=utf-8
output.encoding=utf-8
```

修改HelloServlet.java：

```
package me.letiantian.servlet;

import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
```

```
import java.util.Properties;
import java.io.StringWriter;
import org.apache.velocity.app.VelocityEngine;
import org.apache.velocity.VelocityContext;


@WebServlet(name = "HelloServlet", urlPatterns = {"/hello"})
public class HelloServlet extends HttpServlet {

    protected void processRequest(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();

        Properties properties=new Properties();
        properties.load(getServletContext().getResourceAsStream("/WEB-INF/velocity.properties"));

        VelocityEngine velocityEngine = new VelocityEngine(properties);
        velocityEngine.setApplicationAttribute("javax.servlet.ServletContext", request.getServletContext());

        VelocityContext context=new VelocityContext();
        context.put("name", "user01");
        StringWriter sw = new StringWriter();
        velocityEngine.mergeTemplate("hello.vm", "utf-8", context, sw);

        out.println(sw.toString());

    }

    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        processRequest(request, response);
    }

    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        processRequest(request, response);
    }

}
```

使用数据库连接池

目前比较常见的连接池实现有DBCP、C3P0,Tomcat_JDBC等。

本文使用的连接池是DBCP。

进入http://commons.apache.org/proper/commons-dbcp/download_dbcp.cgi下载 `Apache Commons DBCP for JDBC` ，http://commons.apache.org/proper/commons-pool/download_pool.cgi中下载 `Apache Commons Pool` ，http://dev.mysql.com/downloads/connector/j/下载MySQL的JDBC驱动。

若下载出现问题，可以到一些Maven仓库中下载。例如http://mvnrepository.com/、http://maven.oschina.net。

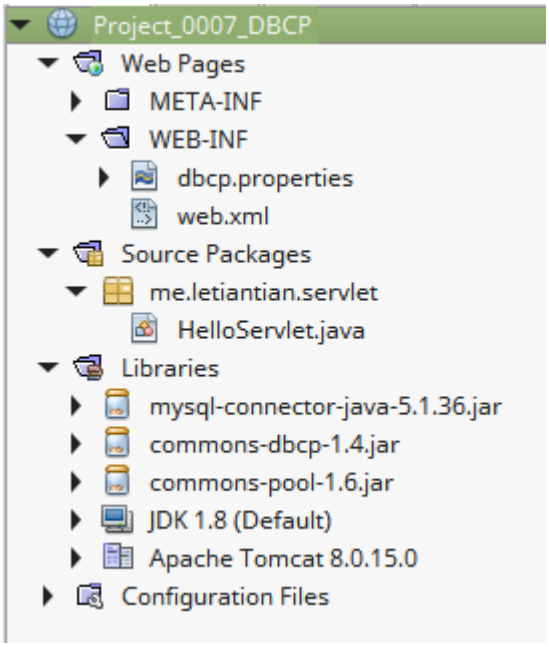
数据库准备

MySQL 5.6。

```
--创建数据库
CREATE DATABASE IF NOT EXISTS `test` DEFAULT CHARSET utf8 COLLATE utf8_general_ci;
USE `test`;
--创建table
CREATE TABLE IF NOT EXISTS USER
(
    `id` INT AUTO_INCREMENT,
    `name` VARCHAR(255),
     `email` VARCHAR(255),
     `age` VARCHAR(255),
    `passwd` VARCHAR(255),
    PRIMARY KEY (`id`),
     UNIQUE KEY (`name`),
    UNIQUE KEY (`email`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
---插入若干数据
INSERT INTO USER (`name`, `email`, `age`, `passwd`)
VALUES ('user01', 'user01@163.com', 20, password('123'));

INSERT INTO USER (`name`, `email`, `age`, `passwd`)
VALUES ('user02', 'user02@163.com', 20, password('456'));
```

示例1

目录结构如下：



web.xml源码：

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.1" xmlns="http://xmlns.jcp.org/xml/ns/javaee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

    <servlet-mapping>
        <servlet-name>default</servlet-name>
        <url-pattern>*.jpg</url-pattern>
    </servlet-mapping>

    <servlet-mapping>
        <servlet-name>default</servlet-name>
        <url-pattern>*.png</url-pattern>
    </servlet-mapping>

    <servlet-mapping>
        <servlet-name>default</servlet-name>
        <url-pattern>*.js</url-pattern>
    </servlet-mapping>

    <servlet-mapping>
        <servlet-name>default</servlet-name>
        <url-pattern>*.css</url-pattern>
    </servlet-mapping>

    <session-config>
        <session-timeout>
            30
        </session-timeout>
    </session-config>

</web-app>
```

dbcp.properties源码：

```
driverClassName=com.mysql.jdbc.Driver
url=jdbc:mysql://localhost:3306/test
username=root
```

```
password=123456
initialSize=2
maxActive=15
maxIdle=2
minIdle=1
maxWait=30000
```

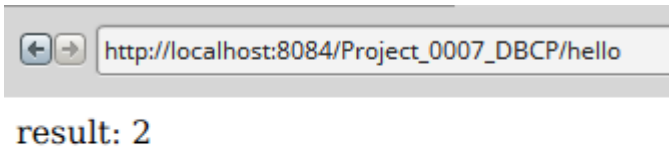
这些配置的解释请见BasicDataSource Configuration Parameters。

HelloServlet.java源码：

```java
package me.letiantian.servlet;

import java.io.IOException;
import java.io.PrintWriter;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.util.Properties;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.sql.DataSource;

import org.apache.commons.dbcp.BasicDataSourceFactory;


@WebServlet(name = "HelloServlet", urlPatterns = {"/hello"})
public class HelloServlet extends HttpServlet {

    protected void processRequest(HttpServletRequest request, HttpServletResponse response) throws IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        try{
            Properties properties=new Properties();
            properties.load(getServletContext().getResourceAsStream("/WEB-INF/dbcp.properties"));
            DataSource dataSource = BasicDataSourceFactory.createDataSource(properties);
            Connection conn = dataSource.getConnection();
            String sql = "select 1+1 as result;";
            PreparedStatement pstmt = conn.prepareStatement(sql);
            ResultSet rs = pstmt.executeQuery();

            if (rs.next()) {
                int result = rs.getInt("result");
                out.println("result: " + result);
            }

            rs.close();
            pstmt.close();
            conn.close();

        } catch (Exception ex) {
            out.println(ex.getMessage());
        }
    }

    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws IOException {
        processRequest(request, response);
    }
```

```
    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws IOException {
        processRequest(request, response);
    }

}
```

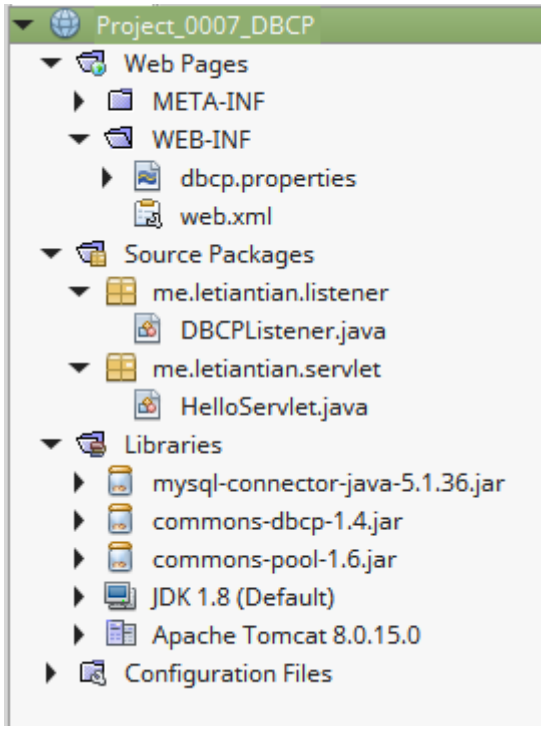
运行项目，浏览器访问 `http://localhost:8084/Project_0007_DBCP/hello` ：



改进：将初始化的连接池放到Servlet上下文中

上面代码中是再servlet中初始化连接池，更好的方法是再Listener中初始化，并将连接池作为属性放入servlet上下文中。

源文件以及代码有所变化，项目结构如下：



DBCPListener.java内容如下：

```
package me.letiantian.listener;
```

```java
import java.util.Properties;
import javax.servlet.ServletContext;
import javax.servlet.ServletContextEvent;
import javax.servlet.ServletContextListener;
import javax.servlet.annotation.WebListener;
import javax.sql.DataSource;
import org.apache.commons.dbcp.BasicDataSourceFactory;

@WebListener
public class DBCPListener implements ServletContextListener{

    // 应用启动时，该方法被调用
    @Override
    public void contextInitialized(ServletContextEvent sce) {
        try {
            System.out.println("设置数据库连接池");
            ServletContext application = sce.getServletContext();
            Properties properties=new Properties();
            properties.load(application.getResourceAsStream("/WEB-INF/dbcp.properties"));
            DataSource dataSource = BasicDataSourceFactory.createDataSource(properties);
            application.setAttribute("dataSource", dataSource);
        }
        catch(Exception ex) {
            System.err.println("数据库连接池设置出现异常：" + ex.getMessage());
        }
    }

    // 应用关闭时，该方法被调用
    @Override
    public void contextDestroyed(ServletContextEvent sce) {

    }

}
```

HelloServlet.java内容如下：

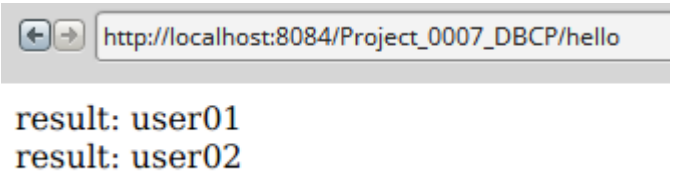
```java
package me.letiantian.servlet;

import java.io.IOException;
import java.io.PrintWriter;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.sql.DataSource;

@WebServlet(name = "HelloServlet", urlPatterns = {"/hello"})
public class HelloServlet extends HttpServlet {

    protected void processRequest(HttpServletRequest request, HttpServletResponse response) throws IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        try {
            DataSource dataSource = (DataSource) getServletContext().getAttribute("dataSource");
            Connection conn = dataSource.getConnection();
            String sql = "select name from user;";
            PreparedStatement pstmt = conn.prepareStatement(sql);
            ResultSet rs = pstmt.executeQuery();
```

```
            while (rs.next()) {
                String name = rs.getString("name");
                out.println("result: " + name + "</br>");
            }

            rs.close();
            pstmt.close();
            conn.close();

        } catch (Exception ex) {
            out.println(ex.getMessage());
        }
    }

    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws IOException {
        processRequest(request, response);
    }


    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws IOException {
        processRequest(request, response);
    }

}
```

启动项目，可以看到Tomcat输出：

```
设置数据库连接池
```

浏览器输出：



查看一下mysql的连接：

```
mysql> show processlist;
+----+------+-----------------+------+---------+------+-------+------------------+
| Id | User | Host            | db   | Command | Time | State | Info             |
+----+------+-----------------+------+---------+------+-------+------------------+
| 45 | root | localhost       | test | Query   |    0 | init  | show processlist |
| 77 | root | localhost:41770 | test | Sleep   |  300 |       | NULL             |
| 78 | root | localhost:41771 | test | Sleep   |  300 |       | NULL             |
| 83 | root | localhost:41790 | test | Sleep   |  274 |       | NULL             |
| 84 | root | localhost:41791 | test | Sleep   |   69 |       | NULL             |
+----+------+-----------------+------+---------+------+-------+------------------+
5 rows in set (0.00 sec)
```

关闭Tomcat，查看数据库连接：

```
mysql> show processlist;
+----+------+-----------+------+---------+------+-------+------------------+
| Id | User | Host      | db   | Command | Time | State | Info             |
```

```
+----+------+-----------+------+---------+------+-------+------------------+
| 45 | root | localhost | test | Query   |    0 | init  | show processlist |
+----+------+-----------+------+---------+------+-------+------------------+
1 row in set (0.00 sec)
```

DBUtils

使用DBUtils可以更加方便的操作数据库，可以参考DBUtils简明教程。

Tomcat 的运行机制

在00-02、理解HTTP中给出了一个简单的服务器代码，Tomcat的设计思路也是类似的。

Tomcat是一个servlet容器。

http://www.kaifajie.cn/tomcat6/7454.html中的内容值得参考：

> 先不去关技术细节，对一个servlet容器，我觉得它首先要做以下事情： 1:实现Servlet api规范。这是最基础的一个实现，servlet api大部分都是接口规范。如request、response、session、cookie。为了我们应用端能正常使用，容器必须有一套完整实现。
> 2：启动Socket监听端口，等待http请求。
> 3：获取http请求，分发请求给不同的协议处理器，如http和https在处理上是不一样的。
> 4：封装请求，构造HttpServletRequest。把socket获取的用户请求字节流转换成java对象httprequest。构造HttpResponse。
> 5：调用(若未创建，则先加载)servlet，调用init初始化，执行servlet.service()方法。
> 6：为httpResponse添加header等头部信息。
> 7：socket回写流，返回满足http协议格式的数据给浏览器。
> 8：实现JSP语法分析器，JSP标记解释器。JSP servlet实现和渲染引擎。
> 9：JNDI、JMX等服务实现。容器一般额外提供命名空间服务管理。
> 10：线程池管理，创建线程池，并为每个请求分配线程。

Tomcat有自己的类加载机制。可以参考：

Java类加载原理解析

深入探讨 Java 类加载器

Tomcat类加载器体系结构

Tomcat 8 权威指南

2

Spring MVC

Spring 与依赖注入

依赖注入是反转控制的一种。

什么是反转控制？

我们平常写程序，需要什么对象，就在代码里显式地new一个出来然后使用，这是我们自己去控制对象的生成。而反转控制是让Spring（或者类似的其他工具）帮忙去生成我们需要的对象，也就是说对象的生成的控制权交给Spring了。

当然，Spring需要依据一定的规则去生成对象，这个规则就在我们写的xml配置文件、或者代码中添加的注解之中。 换句话说，我们不要生成对象，但是要去写配置。

据说，反转控制可用于解耦。这个在小型的项目中很难看出来，项目越大越能感受得到。（我是没写过这方面的大的项目，想着xml配置就头疼）

反转控制的实现中应用了大量的反射。

依赖注入

声明依赖关系，Spring将对象A需要的对象B注入到对象A中。

建议阅读

google `Spring 依赖注入`。

Spring与面向切面编程

面向切面编程是一种编程模式。

使用动态代理可以实现面向切面编程。

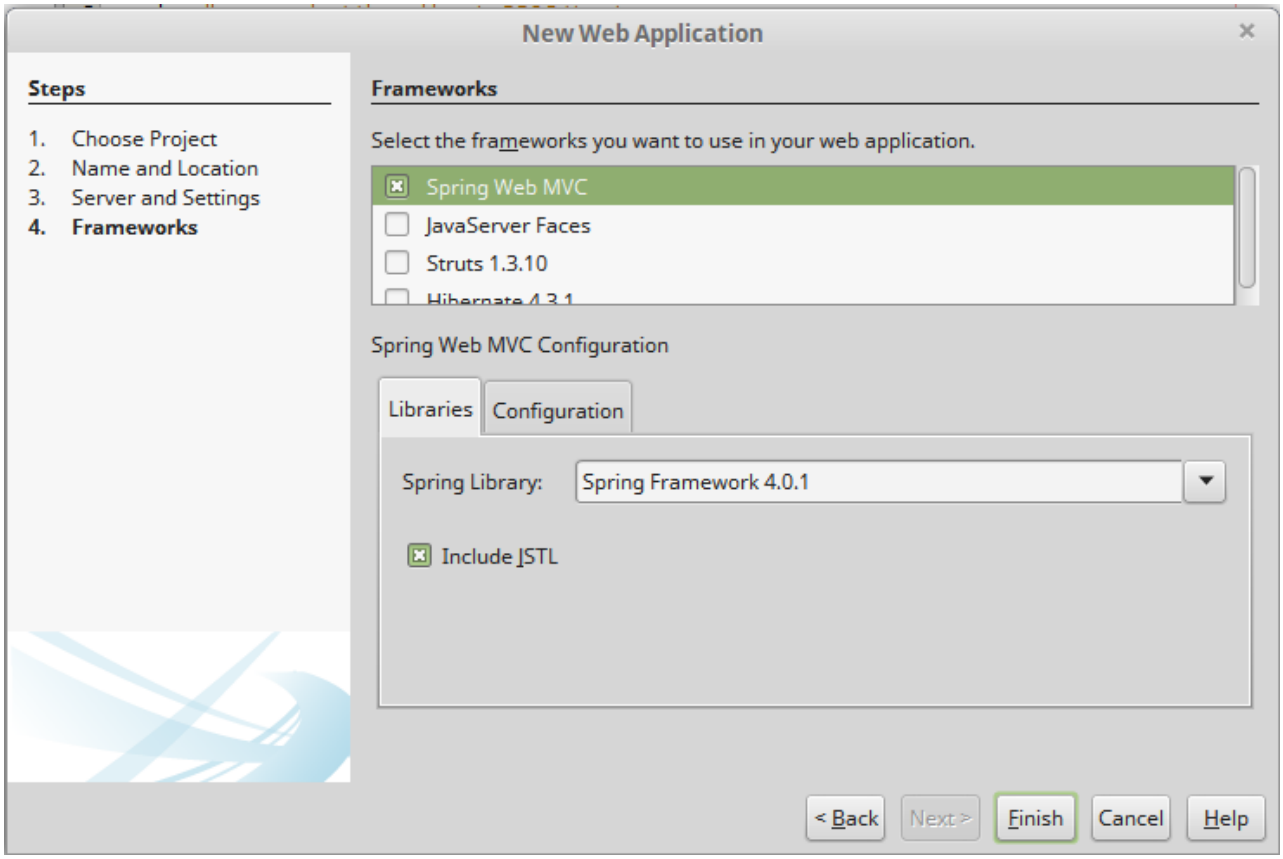
google：`java 设计模式 代理`、`java 动态代理`、`Spring AOP`、`Spring 面向切面`。

使用Spring MVC构建Hello World

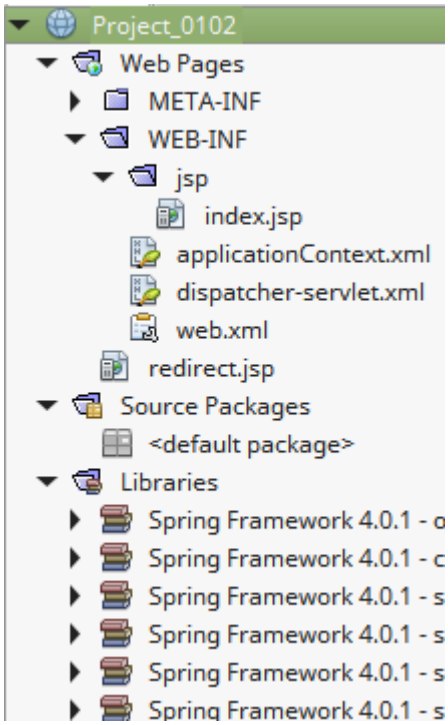
本文演示如何使用Spring MVC做出最简单的Hello World应用。

示例1

项目创建和之前一样，不过在最后一步要选择Spring Web MVC：



项目结构如下：



web.xml源码：

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.1" xmlns="http://xmlns.jcp.org/xml/ns/javaee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    <context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>/WEB-INF/applicationContext.xml</param-value>
    </context-param>

    <listener>
        <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
    </listener>

    <servlet>
        <servlet-name>dispatcher</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
        <load-on-startup>2</load-on-startup>
    </servlet>

    <servlet-mapping>
        <servlet-name>dispatcher</servlet-name>
        <url-pattern>*.htm</url-pattern>
    </servlet-mapping>

    <session-config>
        <session-timeout>
            30
        </session-timeout>
    </session-config>
    <welcome-file-list>
        <welcome-file>redirect.jsp</welcome-file>
    </welcome-file-list>
</web-app>
```

如果遇到匹配*.htm的URL，会使用`org.springframework.web.servlet.DispatcherServlet`来处理。

applicationContext.xml源码：

```
<?xml version='1.0' encoding='UTF-8' ?>
<!-- was: <?xml version="1.0" encoding="UTF-8"?> -->
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:p="http://www.springframework.org/schema/p"
       xmlns:aop="http://www.springframework.org/schema/aop"
       xmlns:tx="http://www.springframework.org/schema/tx"
       xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spri
       http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-4.0.xsd
       http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-4.0.xsd">

    <!--bean id="propertyConfigurer"
          class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer"
          p:location="/WEB-INF/jdbc.properties" />

<bean id="dataSource"
    class="org.springframework.jdbc.datasource.DriverManagerDataSource"
    p:driverClassName="${jdbc.driverClassName}"
    p:url="${jdbc.url}"
    p:username="${jdbc.username}"
    p:password="${jdbc.password}" /-->

    <!-- ADD PERSISTENCE SUPPORT HERE (jpa, hibernate, etc) -->

</beans>
```

applicationContext.xml是Spring的配置文件。

dispatcher-servlet.xml源码：

```
<?xml version='1.0' encoding='UTF-8' ?>
<!-- was: <?xml version="1.0" encoding="UTF-8"?> -->
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:p="http://www.springframework.org/schema/p"
       xmlns:aop="http://www.springframework.org/schema/aop"
       xmlns:tx="http://www.springframework.org/schema/tx"
       xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spri
       http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-4.0.xsd
       http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-4.0.xsd">

    <bean class="org.springframework.web.servlet.mvc.support.ControllerClassNameHandlerMapping"/>

    <!--
    Most controllers will use the ControllerClassNameHandlerMapping above, but
    for the index controller we are using ParameterizableViewController, so we must
    define an explicit mapping for it.
    -->
    <bean id="urlMapping" class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
        <property name="mappings">
            <props>
                <prop key="index.htm">indexController</prop>
```

```
            </props>
        </property>
    </bean>

    <bean id="viewResolver"
          class="org.springframework.web.servlet.view.InternalResourceViewResolver"
          p:prefix="/WEB-INF/jsp/"
          p:suffix=".jsp" />

    <!--
    The index controller.
    -->
    <bean name="indexController"
          class="org.springframework.web.servlet.mvc.ParameterizableViewController"
          p:viewName="index" />

</beans>
```

在 `<bean id="viewResolver" .../>` 定义了JSP模板文件的位置和后缀（这样其他地方就可以省略后缀了）。

URL为 `index.htm` 时，对应的控制器是 `indexController` ，其调用了 `/WEB-INF/jsp/` 下的模板 `index.jsp` 。

redirect.jsp源码

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<% response.sendRedirect("index.htm"); %>
```

index.jsp源码

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">

<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>Welcome to Spring Web MVC project</title>
    </head>

    <body>
        <p>Hello! This is the default welcome page for a Spring Web MVC project.</p>
        <p><i>To display a different welcome page for this project, modify</i>
            <tt>index.jsp</tt> <i>, or create your own welcome page then change
                the redirection in</i> <tt>redirect.jsp</tt> <i>to point to the new
                welcome page and also update the welcome-file setting in</i>
            <tt>web.xml</tt>.</p>
    </body>
</html>
```

运行项目，打开浏览器访问 `http://localhost:8084/Project_0102/` ，会自动跳转到 `http://localhost:8084/Project_0102/index.htm` ，并显示 `index.jsp` 的内容。

Hello World

修改dispatcher-servlet.xml，将 `<bean id="urlMapping" .../>` 修改为：

```
<bean id="urlMapping" class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
    <property name="mappings">
        <props>
            <prop key="index.htm">indexController</prop>
            <prop key="hello.htm">helloController</prop>
        </props>
    </property>
</bean>
```

并添加：

```
<bean name="helloController"
      class="me.letiantian.controller.HelloController" />
```

HelloController.java的源码如下：

```
package me.letiantian.controller;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.springframework.web.servlet.ModelAndView;
import org.springframework.web.servlet.mvc.Controller;

public class HelloController implements  Controller{

    @Override
    public ModelAndView handleRequest(HttpServletRequest request, HttpServletResponse response) throws Exception {

        ModelAndView mv = new ModelAndView();
        mv.addObject("message", "Hello World!你好");
        mv.setViewName("hello");
        return mv;
    }

}
```

模板 `hello.jsp` 的源码如下：

```
<html>
    <head>
        <title>Hello world</title>
    </head>
    <body>
        <h1>${message}</h1>
    </body>
</html>
```

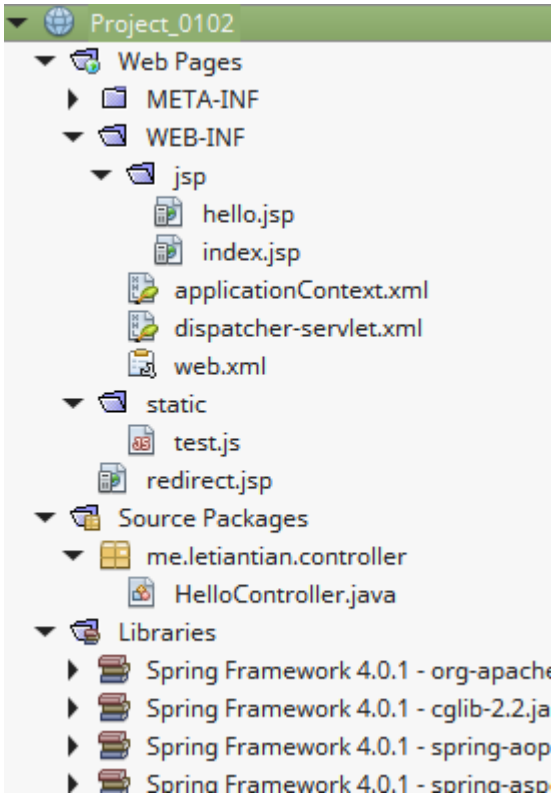
创建static目录，在static目录下创建test.js，内容如下：

```
console.log("hello world");
```

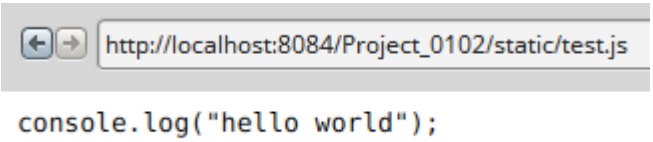
在 web.xml 中添加：

```
<servlet-mapping>
    <servlet-name>default</servlet-name>
    <url-pattern>*.jpg</url-pattern>
</servlet-mapping>

<servlet-mapping>
    <servlet-name>default</servlet-name>
    <url-pattern>*.png</url-pattern>
</servlet-mapping>

<servlet-mapping>
    <servlet-name>default</servlet-name>
    <url-pattern>*.js</url-pattern>
</servlet-mapping>

<servlet-mapping>
    <servlet-name>default</servlet-name>
    <url-pattern>*.css</url-pattern>
</servlet-mapping>
```

好了，现在的项目结构如下：



浏览器访问结果：



乱码了~囧~

解决方法：

在 `HelloController.java` 加入 `response.setContentType("text/html;charset=UTF-8");` ：

```
package me.letiantian.controller;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.springframework.web.servlet.ModelAndView;
import org.springframework.web.servlet.mvc.Controller;

public class HelloController implements  Controller{

    @Override
    public ModelAndView handleRequest(HttpServletRequest request, HttpServletResponse response) throws Exception {
        response.setContentType("text/html;charset=UTF-8");  // 新加入的内容
        ModelAndView mv = new ModelAndView();
        mv.addObject("message", "Hello World!你好");
        mv.setViewName("hello");
        return mv;
    }

}
```

示例2

换种方法配置静态资源

删掉在 `web.xml` 中的：

```
<servlet-mapping>
    <servlet-name>default</servlet-name>
    <url-pattern>*.jpg</url-pattern>
</servlet-mapping>

<servlet-mapping>
    <servlet-name>default</servlet-name>
    <url-pattern>*.png</url-pattern>
</servlet-mapping>

<servlet-mapping>
    <servlet-name>default</servlet-name>
    <url-pattern>*.js</url-pattern>
</servlet-mapping>

<servlet-mapping>
    <servlet-name>default</servlet-name>
```

```
    <url-pattern>*.css</url-pattern>
</servlet-mapping>
```

在 `dispatcher-servlet.xml` 中增加以下内容：

```
<?xml version='1.0' encoding='UTF-8' ?>
<!-- was: <?xml version="1.0" encoding="UTF-8"?> -->
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:p="http://www.springframework.org/schema/p"
       xmlns:aop="http://www.springframework.org/schema/aop"
       xmlns:tx="http://www.springframework.org/schema/tx"
       xmlns:mvc="http://www.springframework.org/schema/mvc"
       xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spri
       http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-4.0.xsd
       http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-4.0.xsd
       http://www.springframework.org/schema/mvc http://www.springframework.org/schema/mvc/spring-mvc-4.0.xsd">

    <!-- 其他内容 -->
    <mvc:resources mapping="/static/**" location="/static/"/>

</beans>
```

注意，在beans的属性中增加了 `xmlns:mvc="http://www.springframework.org/schema/mvc"` ，属性 `xsi:schemaLocation` 中增加了 `http://www.springframework.org/schema/mvc http://www.springframework.org/schema/mvc/spring-mvc-4.0.xsd` 。

不再使用任何后缀（例如.html，.jsp）

将 `redirect.jsp` 修改为：

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<% response.sendRedirect("index"); %>
```

将web.xml中的：

```
<servlet-mapping>
    <servlet-name>dispatcher</servlet-name>
    <url-pattern>*.htm</url-pattern>
</servlet-mapping>
```

修改为：

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

将 `dispatcher-servlet.xml` 中的：

```
<bean id="urlMapping" class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
    <property name="mappings">
        <props>
            <prop key="index.htm">indexController</prop>
            <prop key="hello.htm">helloController</prop>
```

```
        </props>
    </property>
</bean>
```

修改为：

```
<bean id="urlMapping" class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
    <property name="mappings">
        <props>
            <prop key="index">indexController</prop>
            <prop key="hello">helloController</prop>
        </props>
    </property>
</bean>
```

然后，浏览器访问 `http://localhost:8084/Project_0102/hello` 。

示例3

这个示例展示如何获取URL中的数据。

修改 `HelloController.java` ：

```
package me.letiantian.controller;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.springframework.web.servlet.ModelAndView;
import org.springframework.web.servlet.mvc.Controller;

public class HelloController implements  Controller{

    @Override
    public ModelAndView handleRequest(HttpServletRequest request, HttpServletResponse response) throws Exception {
        response.setContentType("text/html;charset=UTF-8");
        ModelAndView mv = new ModelAndView();
        mv.addObject("name", request.getParameter("name"));
        mv.setViewName("hello");
        return mv;
    }

}
```

修改 `hello.jsp` ：

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<html>
    <head>
        <title>Hello world</title>
    </head>
    <body>
        <h1>${pageContext.request.contextPath}</h1>

        <form action="${pageContext.request.contextPath}/hello" method="GET">
            <input name="name" />
            <input type="submit" value="提交"/>
```

```
        </form>

        <h2>提交的数据： ${name}</h2>

    </body>
</html>
```

`${pageContext.request.contextPath}` 的输出是 `/Project_0102` 。

浏览器访问：

再编辑JSP文件时候遇到了这样的问题：

> The header.jspf contains characters which will probably be damaged during conversion to the ISO-8859-1 character set. Do you want to save the file using this character set?

解决办法见http://stackoverflow.com/questions/15499182/netbeans-forces-me-to-save-in-specific-encoding。

资料

- Chapter 13. Web MVC framework
- Spring MVC - How to include JS or CSS files in a JSP page
- dispatcher-servlet.xml and application-context.xml的区别
- Spring MVC SimpleUrlHandlerMapping example
- springMVC中文乱码问题
- SpringMVC 基于注解的Controller @RequestMapping @RequestParam..
- urlMapping也可以通过注解来定义，例如Spring 4 MVC Hello World Tutorial - Full Example。

JdbcTemplate

JdbcTemplate是Spring MVC内置的对JDBC的一个封装。

数据库准备

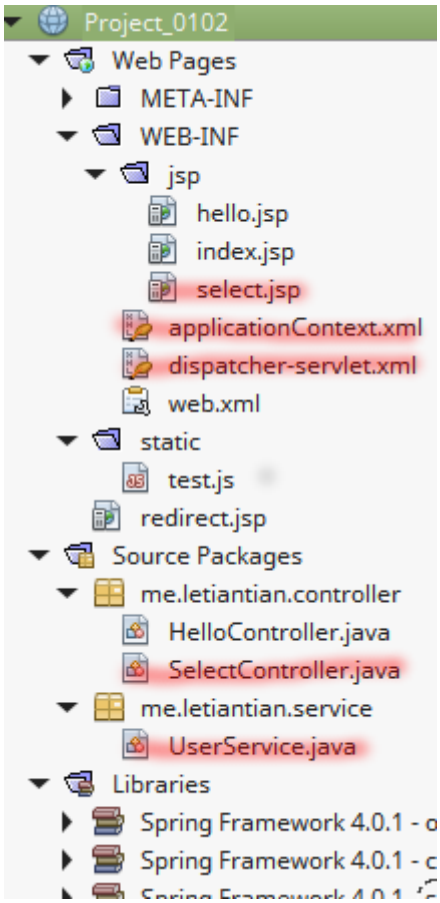
MySQL 5.6。

```
--创建数据库
CREATE DATABASE IF NOT EXISTS `test` DEFAULT CHARSET utf8 COLLATE utf8_general_ci;
USE `test`;
--创建table
CREATE TABLE IF NOT EXISTS USER
(
    `id` INT AUTO_INCREMENT,
    `name` VARCHAR(255),
     `email` VARCHAR(255),
     `age` VARCHAR(255),
    `passwd` VARCHAR(255),
    PRIMARY KEY (`id`),
     UNIQUE KEY (`name`),
    UNIQUE KEY (`email`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
---插入若干数据
INSERT INTO USER (`name`, `email`, `age`, `passwd`)
VALUES ('user01', 'user01@163.com', 20, password('123'));

INSERT INTO USER (`name`, `email`, `age`, `passwd`)
VALUES ('user02', 'user02@163.com', 20, password('456'));

INSERT INTO USER (`name`, `email`, `age`, `passwd`)
VALUES ('用户03', 'user03@163.com', 20, password('456'));
```

示例1

继续使用的上一节01-02、使用Spring MVC构建Hello World中创建的项目。

项目结构如下：



图中红线下的文件是新增或者修改的文件。

MySQL的JDBC封装 `mysql-connector-java-**.jar` 别忘了放到Libraries里。

源码

SelectController.java源码：

```
package me.letiantian.controller;

import java.util.List;
import java.util.Map;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.servlet.ModelAndView;
import org.springframework.web.servlet.mvc.Controller;
import org.springframework.jdbc.core.JdbcTemplate;

import me.letiantian.service.UserService;

public class SelectController implements  Controller{

    @Autowired
    private UserService userDao;
```

```
    @Autowired
    private JdbcTemplate jdbcTemplate;

    @Override
    public ModelAndView handleRequest(HttpServletRequest request, HttpServletResponse response) throws Exception {
        response.setContentType("text/html;charset=UTF-8");
        ModelAndView mv = new ModelAndView();

        List users = jdbcTemplate.queryForList("SELECT * FROM user");
        mv.addObject("users", users);

        Map user1 = userDao.getUserById(1);
        mv.addObject("user1", user1);

        Map user2 = jdbcTemplate.queryForMap("SELECT * FROM user WHERE id=2");
        mv.addObject("user2", user2);

        mv.addObject("message", "无错误信息");
        mv.setViewName("select");
        return mv;
    }

}
```

**UserService.java源码： **

```
package me.letiantian.service;

import java.util.Map;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.stereotype.Service;

@Service
public class UserService {

    @Autowired
    private JdbcTemplate jdbcTemplate;

    public Map getUserById(int id) {

        Map user = jdbcTemplate.queryForMap("SELECT * FROM user WHERE id=?", new Object[] {id});
        return user;

    }

}
```

**select.jsp源码： **

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>JSP Page</title>
    </head>
    <body>
        <h1>Hello World!</h1>
    <c:if test="${not empty users}">
            <ul>
```

```
                <c:forEach var="user" items="${users}">
                    <li>
                        <c:forEach var="entry" items="${user}">
                            <c:out value="${entry.key}" /> :
                            <c:out value="${entry.value}" />
                        </c:forEach>
                    </li>
                </c:forEach>
            </ul>
    </c:if>

        <hr/>

        <c:if test="${not empty user1}">
            <ul>
                <c:forEach var="entry" items="${user1}">
                    <li>
                        <c:out value="${entry.key}" />
                        <c:out value="${entry.value}" />
                    </li>
                </c:forEach>
            </ul>
    </c:if>

        <hr/>

        <c:if test="${not empty user2}">
            <ul>
                <c:forEach var="entry" items="${user2}">
                    <li>
                        <c:out value="${entry.key}" />
                        <c:out value="${entry.value}" />
                    </li>
                </c:forEach>
            </ul>
    </c:if>
        <h2>${message}</h2>
    </body>
</html>
```

dispatcher-servlet.xml源码：

```
<?xml version='1.0' encoding='UTF-8' ?>
<!-- was: <?xml version="1.0" encoding="UTF-8"?> -->
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:p="http://www.springframework.org/schema/p"
       xmlns:aop="http://www.springframework.org/schema/aop"
       xmlns:tx="http://www.springframework.org/schema/tx"
       xmlns:mvc="http://www.springframework.org/schema/mvc"
       xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spri
       http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-4.0.xsd
       http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-4.0.xsd
       http://www.springframework.org/schema/mvc http://www.springframework.org/schema/mvc/spring-mvc-4.0.xsd">

    <bean class="org.springframework.web.servlet.mvc.support.ControllerClassNameHandlerMapping"/>

    <bean class="org.springframework.beans.factory.annotation.AutowiredAnnotationBeanPostProcessor"/>

    <bean id="urlMapping" class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
        <property name="mappings">
            <props>
                <prop key="index">indexController</prop>
```

```
                <prop key="hello">helloController</prop>
                <prop key="select">selectController</prop>
            </props>
        </property>
    </bean>

    <bean id="viewResolver"
          class="org.springframework.web.servlet.view.InternalResourceViewResolver"
          p:prefix="/WEB-INF/jsp/"
          p:suffix=".jsp" />

    <bean name="indexController"
          class="org.springframework.web.servlet.mvc.ParameterizableViewController"
          p:viewName="index" />

    <bean name="helloController"
          class="me.letiantian.controller.HelloController" />

    <bean name="selectController"
          class="me.letiantian.controller.SelectController" />

    <mvc:resources mapping="/static/**" location="/static/"/>

</beans>
```

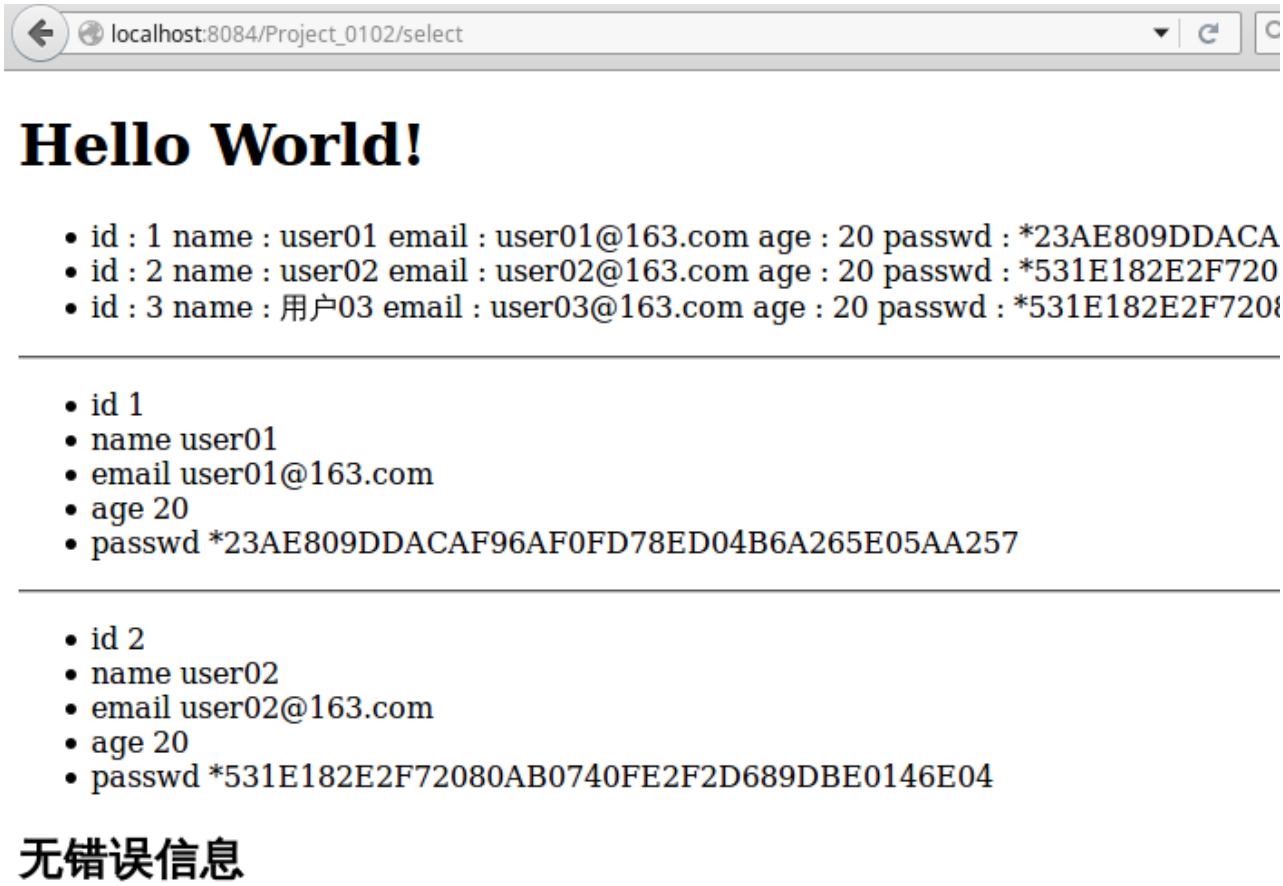
该文件中新增加了 `selectController` ，以及 `<bean class="org.springframework.beans.factory.annotation.AutowiredAnnotationBeanPostProcessor"/>` 以使得 `@Autowired` 能够工作。

applicationContext.xml源码：

```
<?xml version='1.0' encoding='UTF-8' ?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:p="http://www.springframework.org/schema/p"
       xmlns:aop="http://www.springframework.org/schema/aop"
       xmlns:tx="http://www.springframework.org/schema/tx"
       xmlns:context="http://www.springframework.org/schema/context"
       xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spri
       http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-4.0.xsd
       http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-4.0.xsd
       http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-4.0.x

    <context:component-scan base-package="me.letiantian.controller" />
    <context:component-scan base-package="me.letiantian.service" />

    <bean id="dataSource"
          class="org.springframework.jdbc.datasource.DriverManagerDataSource"
          p:driverClassName="com.mysql.jdbc.Driver"
          p:url="jdbc:mysql://localhost:3306/test"
          p:username="root"
          p:password="123456" />

    <bean id="jdbcTemplate" class="org.springframework.jdbc.core.JdbcTemplate">
        <property name="dataSource" ref="dataSource"/>
    </bean>

</beans>
```

`dataSource` 设置程MySQL，并注入到 `jdbcTemplate` 。

运行项目，浏览器访问：



资料

JdbcTemplate中的有多种查询方法，可以参考：

JdbcTemplate 查询

Spring JdbcTemplate方法详解

上面的JSP中用到了JSTL，以下几篇文件可以看一下：

在JSTL EL中处理java.util.Map，及嵌套List的情况

JdbcTemplate也可以使用事务，有声明式和编程式两种方法：

Spring Declarative Transactions

Spring Programmatic Transactions

Spring Programmatic Transaction Management

Spring JdbcTemplate 与 事务管理

Transactions with JdbcTemplate

如何使用连接池？

JDBC Database connection pool in Spring FrameWork - How to SetUp Example

Setup Connection Pooling in Spring MVC

其他：

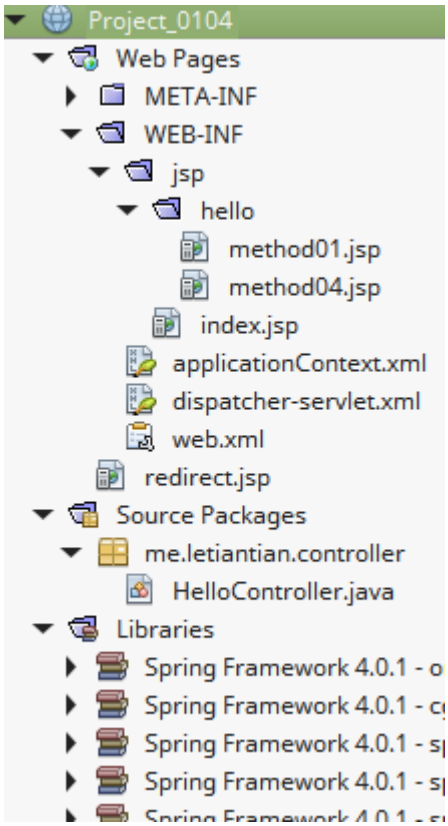
Spring MVC with JdbcTemplate Example

Spring MVC and List Example

基于注解的 URL 映射

项目结构

创建项目 `Pcrojet_0104`，最终结构如下：



源码

applicationContext.xml

```
<?xml version='1.0' encoding='UTF-8' ?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:p="http://www.springframework.org/schema/p"
       xmlns:aop="http://www.springframework.org/schema/aop"
       xmlns:tx="http://www.springframework.org/schema/tx"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:mvc="http://www.springframework.org/schema/mvc"
       xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spri
       http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-4.0.xsd
       http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-4.0.xsd
       http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-4.0.x
       http://www.springframework.org/schema/mvc http://www.springframework.org/schema/mvc/spring-mvc-4.0.xsd
```

```
        http://www.springframework.org/schema/mvc http://www.springframework.org/schema/mvc/spring-mvc-4.0.xsd">

</beans>
```

该配置文件什么都没做。

dispatcher-servlet.xml

```
<?xml version='1.0' encoding='UTF-8' ?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:p="http://www.springframework.org/schema/p"
       xmlns:aop="http://www.springframework.org/schema/aop"
       xmlns:tx="http://www.springframework.org/schema/tx"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:mvc="http://www.springframework.org/schema/mvc"
       xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spri
       http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-4.0.xsd
       http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-4.0.xsd
       http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-4.0.x
       http://www.springframework.org/schema/mvc http://www.springframework.org/schema/mvc/spring-mvc-4.0.xsd">

    <context:component-scan base-package="me.letiantian.controller" />
    <mvc:annotation-driven/>

    <bean class="org.springframework.web.servlet.mvc.support.ControllerClassNameHandlerMapping"/>

    <bean id="urlMapping" class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
        <property name="mappings">
            <props>
                <prop key="index">indexController</prop>
            </props>
        </property>
    </bean>

    <bean id="viewResolver"
          class="org.springframework.web.servlet.view.InternalResourceViewResolver"
          p:prefix="/WEB-INF/jsp/"
          p:suffix=".jsp" />

    <bean name="indexController"
          class="org.springframework.web.servlet.mvc.ParameterizableViewController"
          p:viewName="index" />

</beans>
```

`<beans>` 中增加了属性 `xmlns:context` 、 `xmlns:mvc` ，对应的在 `xsi:schemaLocation` 增加了：

```
http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-4.0.xsd
http://www.springframework.org/schema/mvc http://www.springframework.org/schema/mvc/spring-mvc-4.0.xsd
```

关于 `<mvc:annotation-driven/>` 的意义，可参考What's the difference between and in servlet?。

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.1" xmlns="http://xmlns.jcp.org/xml/ns/javaee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    <context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>/WEB-INF/applicationContext.xml</param-value>
    </context-param>
    <listener>
        <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
    </listener>
    <servlet>
        <servlet-name>dispatcher</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
        <load-on-startup>2</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>dispatcher</servlet-name>
        <url-pattern>/</url-pattern>
    </servlet-mapping>
    <session-config>
        <session-timeout>
            30
        </session-timeout>
    </session-config>
    <welcome-file-list>
        <welcome-file>redirect.jsp</welcome-file>
    </welcome-file-list>
</web-app>
```

模板文件

index.jsp：

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">

<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>Welcome to Spring Web MVC project</title>
    </head>

    <body>
        <p>Hello! This is the default welcome page for a Spring Web MVC project.</p>
        <p><i>To display a different welcome page for this project, modify</i>
            <tt>index.jsp</tt> <i>, or create your own welcome page then change
                the redirection in</i> <tt>redirect.jsp</tt> <i>to point to the new
                welcome page and also update the welcome-file setting in</i>
            <tt>web.xml</tt>.</p>
    </body>
</html>
```

hello/method01.jsp：

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>JSP Page</title>
    </head>
    <body>
        <h1>Hello Method01</h1>
    </body>
</html>
```

hello/method04.jsp：

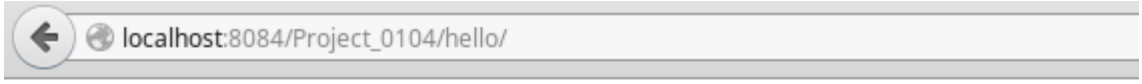
```
<%@page contentType="text/plain" pageEncoding="UTF-8"%>

name: ${name}
```

HelloController.java

```
package me.letiantian.controller;

import java.io.PrintWriter;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;

@Controller
@RequestMapping("/hello")
public class HelloController{

    @RequestMapping(value = "")
    public String index() {
        return "index";
    }

    @RequestMapping(value = "/method01")
    public String method01() {
        return "hello/method01";
    }

    // 仅支持HTTP POST方法
    @RequestMapping(value = "/method02", method = {RequestMethod.POST})
    public String method02() {
        return "hello/method01";
    }

    @RequestMapping(value = "/method03")
    public void method03(HttpServletRequest request, HttpServletResponse response) {
        response.setContentType("text/plain;charset=UTF-8");
        try (PrintWriter out = response.getWriter()) {
            out.println("contextPath: " + request.getContextPath());
            out.println("name: " + request.getParameter("name"));
        } catch (Exception ex) {
            System.out.println(""+ex.getMessage());
        }
```

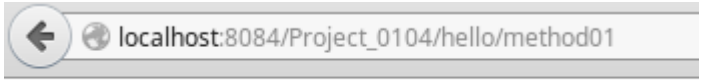
```
    }

    @RequestMapping(value = "/method04/{name}")
    public String method04(@PathVariable String name, Model model) {
        model.addAttribute("name", name);
        return "hello/method04";
    }

    @RequestMapping(value = "/method05/{id}")
    public String method05(@PathVariable int id, Model model) {
        model.addAttribute("name", id);
        return "hello/method04";
    }
}
```

测试

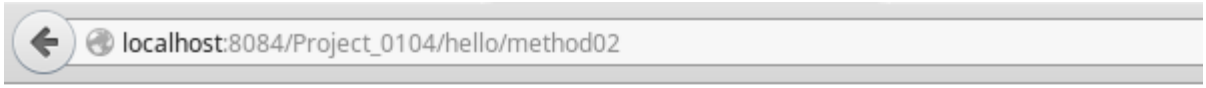


Hello! This is the default welcome page for a Spring Web MVC projec

To display a different welcome page for this project, modify `index.jsp` , `redirect.jsp` *to point to the new welcome page and also update the wel*



Hello Method01



HTTP Status 405 - Request method 'GET' not supported

---

type Status report

message Request method 'GET' not supported

description The specified HTTP method is not allowed for the requested resource.

Apache Tomcat/8.0.15

localhost:8084/Project_0104/hello/method03?name=letian

```
contextPath: /Project_0104
name: letian
```

localhost:8084/Project_0104/hello/method04/letian

```
name: letian
```

localhost:8084/Project_0104/hello/method05/letian

HTTP Status 400 -

type Status report

message

description The request sent by the client was syntactically incorrect.

Apache Tomcat/8.0.15

localhost:8084/Project_0104/hello/method05/12345667788888888888888888888

HTTP Status 400 -

type Status report

message

description The request sent by the client was syntactically incorrect.

Apache Tomcat/8.0.15

localhost:8084/Project_0104/hello/method05/123456

```
name: 123456
```

其他

如何让一个方法映射多个URL？

很简单，例如 `@RequestMapping(value = {"/hello", "/hi"})` 。可参考Spring MVC: Mapping Multiple URLs to Same Controller。

如何自定义错误页面？

Spring MVC : How To Return Custom 404 Error Pages

Spring MVC Exception Handling Example

Exception Handling in Spring MVC

@PathVariable是绑定数据的其中一种方法，还有绑定Cookie中数据，将表单数据绑定到对象中等方法：

Spring MVC Cookie example

Injecting and Binding Objects to Spring MVC Controllers

JSON

如何响应JSON数据

Spring 3 MVC and JSON example

Spring MVC - Easy REST-Based JSON Services with @ResponseBody

如何处理HTTP请求中的JSON数据

How to pass Json object from ajax to spring mvc controller?

校验器

校验器，Validator。

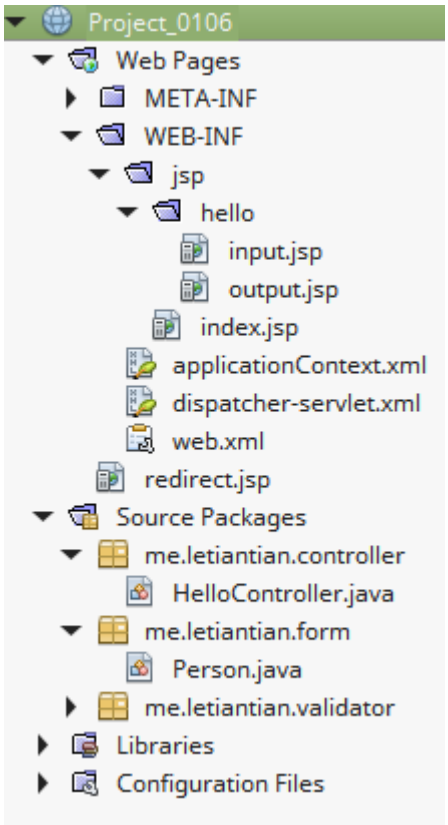
在处理带有表单数据的HTTP请求时，通常这样做：

```
if (表单数据符合要求) {
    处理数据，返回结果；
} else {
    返回结果，提示用户重新输入数据；
}
```

判断表单数据是否符合要求这就是校验器该做的事情。我们可以自己编写校验类，也可以使用Spring MVC自带的相关类。

将表单数据绑定到对象中

项目结构如下：



源码

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.1" xmlns="http://xmlns.jcp.org/xml/ns/javaee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    <context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>/WEB-INF/applicationContext.xml</param-value>
    </context-param>
    <listener>
        <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
    </listener>
    <servlet>
        <servlet-name>dispatcher</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
        <load-on-startup>2</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>dispatcher</servlet-name>
        <url-pattern>/</url-pattern>
    </servlet-mapping>
    <session-config>
        <session-timeout>
            30
        </session-timeout>
    </session-config>
    <welcome-file-list>
        <welcome-file>redirect.jsp</welcome-file>
    </welcome-file-list>
</web-app>
```

applicationContext.xml

```
<?xml version='1.0' encoding='UTF-8' ?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:p="http://www.springframework.org/schema/p"
       xmlns:aop="http://www.springframework.org/schema/aop"
       xmlns:tx="http://www.springframework.org/schema/tx"
       xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spri
       http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-4.0.xsd
       http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-4.0.xsd">

</beans>
```

dispatcher-servlet.xml

```
<?xml version='1.0' encoding='UTF-8' ?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:p="http://www.springframework.org/schema/p"
       xmlns:aop="http://www.springframework.org/schema/aop"
       xmlns:tx="http://www.springframework.org/schema/tx"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:mvc="http://www.springframework.org/schema/mvc"
       xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spri
       http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-4.0.xsd
```

```
        http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-4.0.xsd
        http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-4.0.x
        http://www.springframework.org/schema/mvc http://www.springframework.org/schema/mvc/spring-mvc-4.0.xsd">

    <context:component-scan base-package="me.letiantian.controller" />
    <mvc:annotation-driven/>

    <bean class="org.springframework.web.servlet.mvc.support.ControllerClassNameHandlerMapping"/>

    <bean id="urlMapping" class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
        <property name="mappings">
            <props>
                <prop key="index">indexController</prop>
            </props>
        </property>
    </bean>

    <bean id="viewResolver"
          class="org.springframework.web.servlet.view.InternalResourceViewResolver"
          p:prefix="/WEB-INF/jsp/"
          p:suffix=".jsp" />

    <bean name="indexController"
          class="org.springframework.web.servlet.mvc.ParameterizableViewController"
          p:viewName="index" />

</beans>
```

hello/input.jsp

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>JSP Page</title>
    </head>
    <body>
        <form method="POST" action="${pageContext.request.contextPath}/hello/output">
            First Name: <input type="text" name="firstName"> <br/>
            Second Name: <input type="text" name="secondName"> <br/>
            <input type="submit" />

        </form>
    </body>
</html>
```

hello/output.jsp

```
<%@page contentType="text/plain" pageEncoding="UTF-8"%>
first name: ${person.firstName}
second name: ${person.secondName}
```

Person.java

```
package me.letiantian.form;

public class Person {

    private String firstName;
    private String secondName;
```

```
    public String getFirstName() {
        return firstName;
    }

    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }

    public String getSecondName() {
        return secondName;
    }

    public void setSecondName(String secondName) {
        this.secondName = secondName;
    }

}
```

HelloController.java

```
package me.letiantian.controller;

import java.io.PrintWriter;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;

import me.letiantian.form.Person;
import org.springframework.web.bind.annotation.ModelAttribute;

@Controller
@RequestMapping("/hello")
public class HelloController{

    @RequestMapping(value = "")
    public String index() {
        return "index";
    }

    @RequestMapping(value = "/input")
    public String input() {
        return "hello/input";
    }

    @RequestMapping(value = "/output")
    public String output(Person person, Model model) {
        model.addAttribute("person", person);
        return "hello/output";
    }

//    @RequestMapping(value = "/output")
//    public String output(@ModelAttribute(value="person") Person person, Model model) {
//        return "hello/output";
//    }

}
```

注意，

```
    @RequestMapping(value = "/output")
    public String output(Person person, Model model) {
        model.addAttribute("person", person);
        return "hello/output";
    }
```

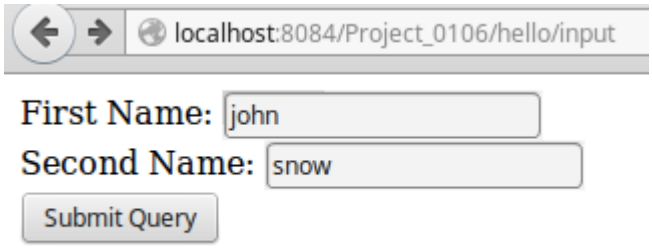
和

```
    @RequestMapping(value = "/output")
    public String output(@ModelAttribute(value="person") Person person, Model model) {
        return "hello/output";
    }
```

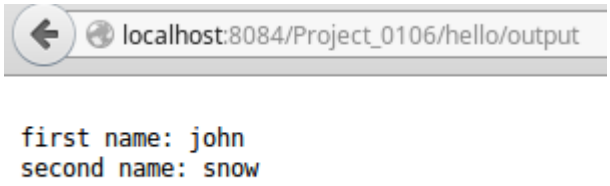
是一样的。

浏览器访问

表单填入信息：



提交表单后：



校验数据

hello/input.jsp修改如下

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@taglib prefix="form" uri="http://www.springframework.org/tags/form" %>
<!DOCTYPE html>
<html>
```

```
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title> JSP Page</title>
        <style type="text/css">
            .error {color: red;}
        </style>
    </head>
    <body>

        <form:form modelAttribute="person" method="POST" action="${pageContext.request.contextPath}/hello/output">
            firstName: <form:input path="firstName" /> <form:errors path="firstName" cssClass="error"/> <br/>
            secondName: <form:input path="secondName" /> <form:errors path="secondName" cssClass="error"/> <br/>
            <input type="submit" value="提交" />
        </form:form>

    </body>
</html>
```

注意 `<form:form modelAttribute="person"` 。

HelloController.java修改如下

```
package me.letiantian.controller;

import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;

import me.letiantian.form.Person;
import me.letiantian.validator.PersonValidator;
import org.springframework.validation.BindingResult;

@Controller
@RequestMapping("/hello")
public class HelloController{


    @RequestMapping(value = "")
    public String index() {
        return "index";
    }

    @RequestMapping(value = "/input")
    public String input(Model model) {
        model.addAttribute("person", new Person());  // 很重要
        return "hello/input";
    }

    @RequestMapping(value = "/output")
    public String output(Person person, BindingResult bindingResult, Model model) {
        model.addAttribute("person", person);  // 很重要
        PersonValidator pv = new PersonValidator();
        pv.validate(person, bindingResult);

        if (bindingResult.hasErrors()) {  // 如果有错误,BindingResult是Errors的子类
            return "hello/input";
        }

        return "hello/output";
```

```
    }

}
```

注意每个方法中的 `model.addAttribute("person")` 是和 `hello/input.jsp` 中的 `<form:form modelAttribute="person"` 对应的。

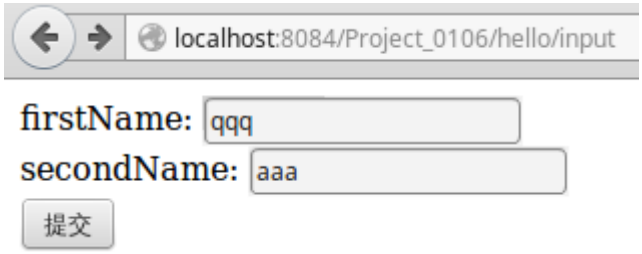
添加PersonValidator.java

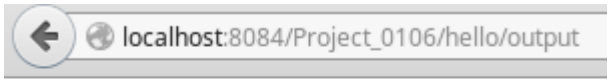
该文件在包 `me.letiantian.validator` 下，内容为：

```
package me.letiantian.validator;

import me.letiantian.form.Person;
import org.springframework.validation.Errors;
import org.springframework.validation.Validator;
import org.springframework.validation.ValidationUtils;

public class PersonValidator implements Validator{

    @Override
    public boolean supports(Class<?> type) {
        return Person.class.isAssignableFrom(type);
    }

    @Override
    public void validate(Object o, Errors errors) {
        Person person = (Person) o;
        ValidationUtils.rejectIfEmptyOrWhitespace(errors, "firstName", null, "firstName不能为空");
        ValidationUtils.rejectIfEmptyOrWhitespace(errors, "secondName", null, "secondName不能为空");
        if (person.getFirstName().length() < 2) {
            errors.rejectValue("firstName", null, "firstName太短");
        }
    }
}
```

测试

第1组：





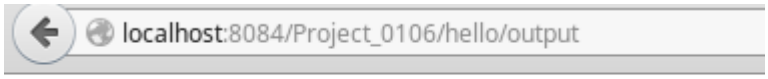

```
first name: qqq
second name: aaa
```

第2组：

firstName: q

secondName: aaa

提交

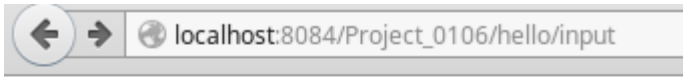


firstName: q firstName太短

secondName: aaa

提交

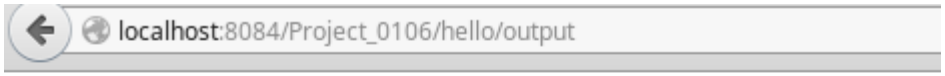
第3组：



firstName:

secondName:

提交



firstName: firstName不能为空

firstName太短

secondName: secondName不能为空

提交

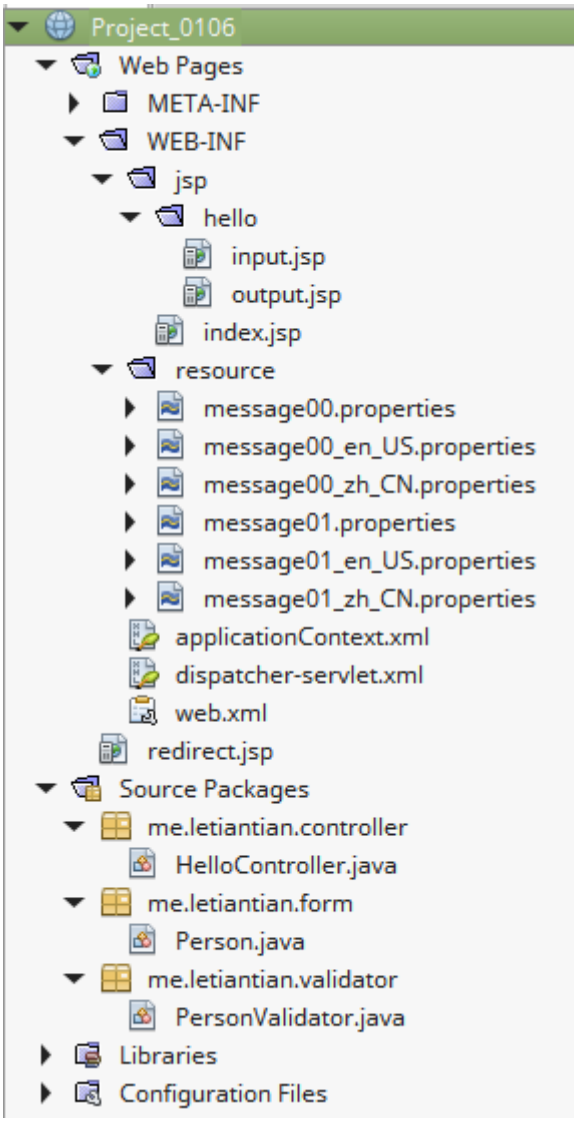
国际化

本节的项目以为01-06、校验器创建的项目 `Project_0106` 为基础。

所谓国际化，是指根据浏览器HTTP请求头中 `Accept-Language` 中指定的语言、或者用户指定的语言（Cookie、session中指定）， 将web页面中的一些文本使用该语言展示出来。

根据浏览器HTTP请求头中 `Accept-Language` 指定的语言进行国际化

项目结构如下：



源码

这里只展示改动或者新增的文件。

dispatcher-servlet.xml

```
<?xml version='1.0' encoding='UTF-8' ?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:p="http://www.springframework.org/schema/p"
       xmlns:aop="http://www.springframework.org/schema/aop"
       xmlns:tx="http://www.springframework.org/schema/tx"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:mvc="http://www.springframework.org/schema/mvc"
       xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spri
       http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-4.0.xsd
       http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-4.0.xsd
       http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-4.0.x
       http://www.springframework.org/schema/mvc http://www.springframework.org/schema/mvc/spring-mvc-4.0.xsd">

    <context:component-scan base-package="me.letiantian.controller" />
    <mvc:annotation-driven/>

    <bean class="org.springframework.web.servlet.mvc.support.ControllerClassNameHandlerMapping"/>

    <bean id="urlMapping" class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
        <property name="mappings">
            <props>
                <prop key="index">indexController</prop>
            </props>
        </property>
    </bean>

    <bean id="viewResolver"
          class="org.springframework.web.servlet.view.InternalResourceViewResolver"
          p:prefix="/WEB-INF/jsp/"
          p:suffix=".jsp" />

    <bean name="indexController"
          class="org.springframework.web.servlet.mvc.ParameterizableViewController"
          p:viewName="index" />

    <!-- 国际化 -->
    <bean id="messageSource" class="org.springframework.context.support.ReloadableResourceBundleMessageSource">
        <property name="basenames">
            <list>
                <value>/WEB-INF/resource/message00</value>
                <value>/WEB-INF/resource/message01</value>
            </list>
        </property>
    </bean>

    <bean id="localeResolver" class="org.springframework.web.servlet.i18n.AcceptHeaderLocaleResolver">

    </bean>
</beans>
```

message00.properties

```
welcome=hello
person.firstName.notempty=firstName can not be empty
person.secondName.notempty=secondName can not be empty
person.firstName.tooshort=firstName is too short
```

message00_en_US.properties

```
welcome=hello
person.firstName.notempty=firstName can not be empty
person.secondName.notempty=secondName can not be empty
person.firstName.tooshort=firstName is too short
```

message00_zh_CN.properties

```
welcome=你好
person.firstName.notempty=firstName不能为空
person.secondName.notempty=secondName不能为空
person.firstName.tooshort=firstName太短
```

message01*.properties

这三个文件为空。

PersonValidator.java

```
package me.letiantian.validator;

import me.letiantian.form.Person;
import org.springframework.validation.Errors;
import org.springframework.validation.Validator;
import org.springframework.validation.ValidationUtils;

public class PersonValidator implements Validator{

    @Override
    public boolean supports(Class<?> type) {
        return Person.class.isAssignableFrom(type);
    }

    @Override
    public void validate(Object o, Errors errors) {
        Person person = (Person) o;
        ValidationUtils.rejectIfEmptyOrWhitespace(errors, "firstName", "person.firstName.notempty");
        ValidationUtils.rejectIfEmptyOrWhitespace(errors, "secondName", "person.secondName.notempty");
        if (person.getFirstName().length() < 2) {
            errors.rejectValue("firstName", "person.firstName.tooshort");
        }
    }

}
```

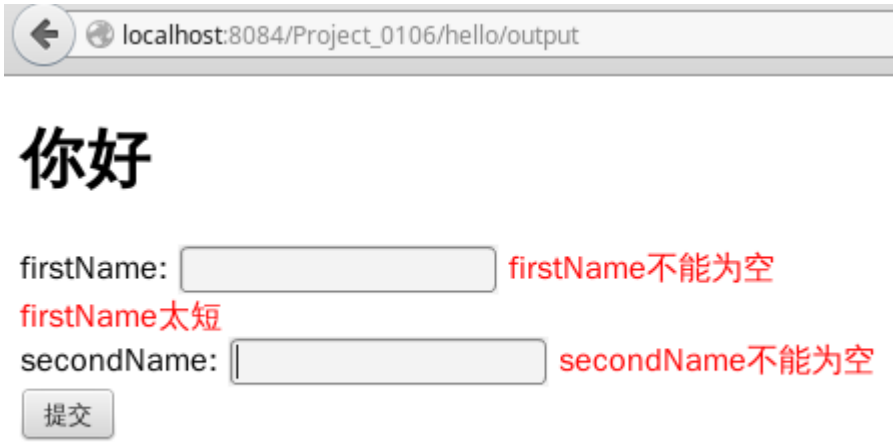
hello/input.jsp

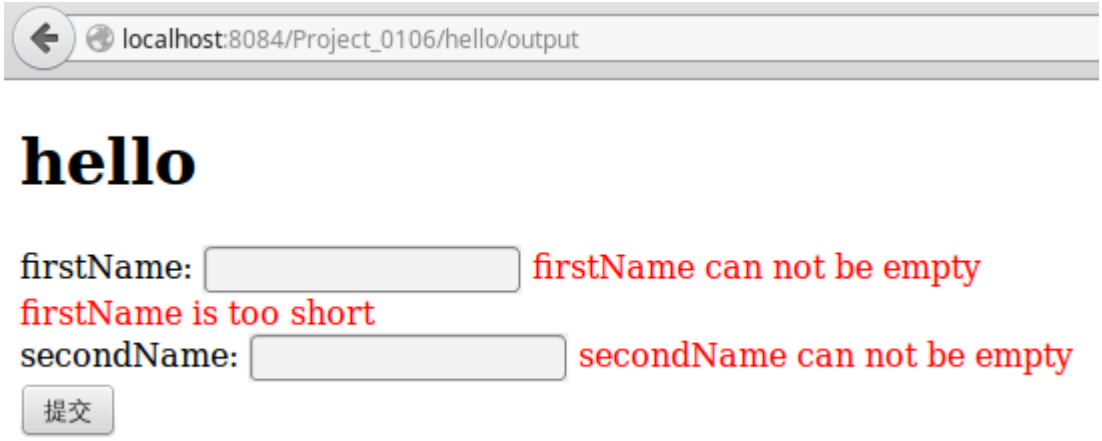
```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@taglib prefix="form" uri="http://www.springframework.org/tags/form" %>
<%@taglib prefix="spring" uri="http://www.springframework.org/tags"%>
<!DOCTYPE html>
<html>
    <head>
```

```
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title> JSP Page</title>
        <style type="text/css">
            .error {color: red;}
        </style>
    </head>
    <body>
        <h1><spring:message code="welcome" /></h1>
        <form:form modelAttribute="person" method="POST" action="${pageContext.request.contextPath}/hello/output">
            firstName: <form:input path="firstName" /> <form:errors path="firstName" cssClass="error"/> <br/>
            secondName: <form:input path="secondName" /> <form:errors path="secondName" cssClass="error"/> <br/>
            <input type="submit" value="提交" />
        </form:form>

    </body>
</html>
```

效果

可以参考Change Mozilla Firefox language settings修改火狐浏览器的Accept-Language。 如果没有效果，可以使用netbeans重启项目，再查看效果。





其他方式的国际化

上面的程序中 `localeResolver` 使用的 `AcceptHeaderLocaleResolver` （见配置文件 `dispatcher-servlet.xml` ）。另外，Spring还给出 `SessionLocaleResolver` 、 `CookieLocaleResolver` 来实现国际化。也可以根据URL中的指定的Locale进行国际化。

可以参考：

SpringMVC学习系列（8） 之 国际化

Spring MVC internationalization example

拦截器

拦截器（interceptor），类似servlet中的过滤器。

Spring 3 MVC Interceptor tutorial with example

Spring MVC Interceptors Example - HandlerInterceptor and HandlerInterceptorAdapter

Spring MVC handler interceptors example

Spring MVC Handler Interceptor

暂不支持注解。Is it possible to wire a Spring MVC Interceptor using annotations?

文件上传

先看一下Servlet是如何处理文件上传的：

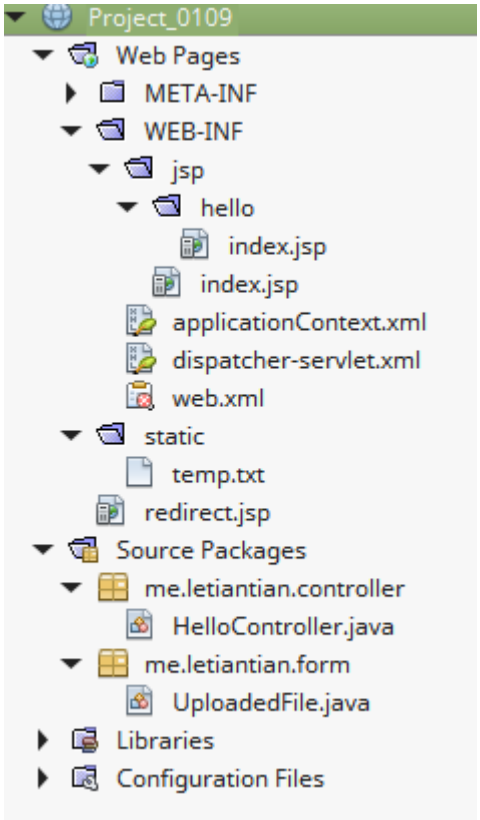
Servlets - File Uploading

Java File Upload Example with Servlet 3.0 API

Spring MVC下的处理是类似的。

下面展示一个简单的实现。

项目结构与源码



web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.1" xmlns="http://xmlns.jcp.org/xml/ns/javaee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance
    <context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>/WEB-INF/applicationContext.xml</param-value>
    </context-param>
    <listener>
```

```
        <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
    </listener>
    <servlet>
        <servlet-name>dispatcher</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
        <load-on-startup>2</load-on-startup>
        <multipart-config>
            <file-size-threshold>1000000</file-size-threshold>
            <max-file-size>2000000</max-file-size>
            <max-request-size>4000000</max-request-size>
        </multipart-config>
    </servlet>
    <servlet-mapping>
        <servlet-name>dispatcher</servlet-name>
        <url-pattern>/</url-pattern>
    </servlet-mapping>
    <session-config>
        <session-timeout>
            30
        </session-timeout>
    </session-config>
    <welcome-file-list>
        <welcome-file>redirect.jsp</welcome-file>
    </welcome-file-list>
</web-app>
```

注意其中的限制文件大小的配置：

```
<multipart-config>
    <file-size-threshold>1000000</file-size-threshold>
    <max-file-size>2000000</max-file-size>
    <max-request-size>4000000</max-request-size>
</multipart-config>
```

applicationContext.xml

```
<?xml version='1.0' encoding='UTF-8' ?>
<!-- was: <?xml version="1.0" encoding="UTF-8"?> -->
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:p="http://www.springframework.org/schema/p"
       xmlns:aop="http://www.springframework.org/schema/aop"
       xmlns:tx="http://www.springframework.org/schema/tx"
       xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spri
       http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-4.0.xsd
       http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-4.0.xsd">

</beans>
```

dispatcher-servlet.xml

```
<?xml version='1.0' encoding='UTF-8' ?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:p="http://www.springframework.org/schema/p"
       xmlns:aop="http://www.springframework.org/schema/aop"
       xmlns:tx="http://www.springframework.org/schema/tx"
```

```
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:mvc="http://www.springframework.org/schema/mvc"
       xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spri
       http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-4.0.xsd
       http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-4.0.xsd
       http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-4.0.x
       http://www.springframework.org/schema/mvc http://www.springframework.org/schema/mvc/spring-mvc-4.0.xsd">

    <context:component-scan base-package="me.letiantian.controller" />
    <context:component-scan base-package="me.letiantian.form" />
    <mvc:annotation-driven/>

    <bean id="multipartResolver"
          class="org.springframework.web.multipart.support.StandardServletMultipartResolver">
    </bean>

    <bean class="org.springframework.web.servlet.mvc.support.ControllerClassNameHandlerMapping"/>

    <bean id="urlMapping" class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
        <property name="mappings">
            <props>
                <prop key="index">indexController</prop>
            </props>
        </property>
    </bean>

    <bean id="viewResolver"
          class="org.springframework.web.servlet.view.InternalResourceViewResolver"
          p:prefix="/WEB-INF/jsp/"
          p:suffix=".jsp" />

    <bean name="indexController"
          class="org.springframework.web.servlet.mvc.ParameterizableViewController"
          p:viewName="index" />

     <mvc:resources mapping="/static/**" location="/static/"/>

</beans>
```

注意，这里增加了一个multipart解析器 `StandardServletMultipartResolver` ，用来处理上传的文件。 `/static` 是存放静态资源的目录， 我们也准备将上传的文件放到这个目录里。

UploadedFile.java

```
package me.letiantian.form;

import org.springframework.web.multipart.MultipartFile;

public class UploadedFile {
    private String fileName;
    private MultipartFile multipartFile;

    public String getFileName() {
        return fileName;
    }

    public void setFileName(String fileName) {
        this.fileName = fileName;
    }
```

```
    public MultipartFile getMultipartFile() {
        return multipartFile;
    }

    public void setMultipartFile(MultipartFile multipartFile) {
        this.multipartFile = multipartFile;
    }

}
```

HelloController.java

```
package me.letiantian.controller;

import java.io.File;
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import me.letiantian.form.UploadedFile;
import org.springframework.web.bind.annotation.ModelAttribute;

import org.springframework.web.multipart.MultipartFile;

@Controller
@RequestMapping("/hello")
public class HelloController{


    @RequestMapping(value = "")
    public String index() {
        return "hello/index";
    }

    @RequestMapping(value = "/upload")
    public void output(@ModelAttribute UploadedFile uploadedFile,
            HttpServletRequest request,
            HttpServletResponse response) throws IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        MultipartFile multiPartFile = uploadedFile.getMultipartFile();
        System.out.println("文件原始名称: "+multiPartFile.getOriginalFilename());
        System.out.println("表单给定的文件名称: "+uploadedFile.getFileName());

        try{
            System.out.println("上传目录: "+request.getServletContext().getRealPath("/static"));
            File file = new File(request.getServletContext().getRealPath("/static"),
                    uploadedFile.getFileName());
            multiPartFile.transferTo(file);  // 将文件写入本地
            out.println("<h2>上传成功</h2>");
        } catch (Exception ex) {
            System.out.println(""+ex.getMessage());
            out.println("<h2>上传失败</h2>");
        }
    }

}
```

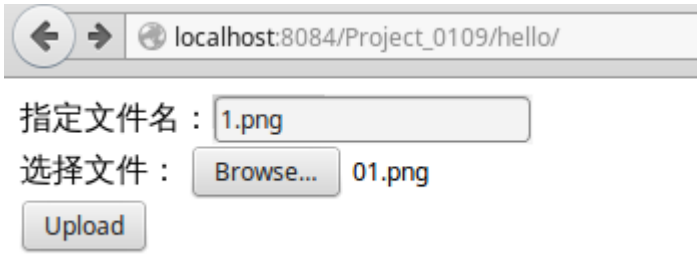
注意，表单数据被绑定到了 `uploadedFile` 对象中。

hello/index.jsp

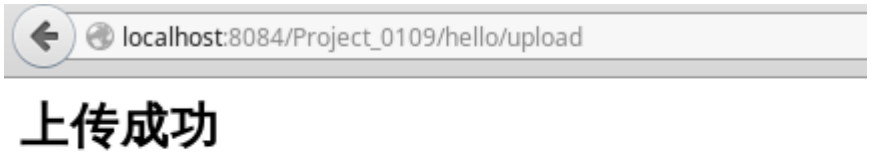
```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>JSP Page</title>
    </head>
    <body>
        <form method="post" action="${pageContext.request.contextPath}/hello/upload" enctype="multipart/form-data">
            指定文件名：<input type="text" name="fileName" /> <br/>
            选择文件： <input type="file" name="multipartFile" size="60"/> <br/>
            <input type="submit" value="Upload" />
        </form>
    </body>
</html>
```

测试程序

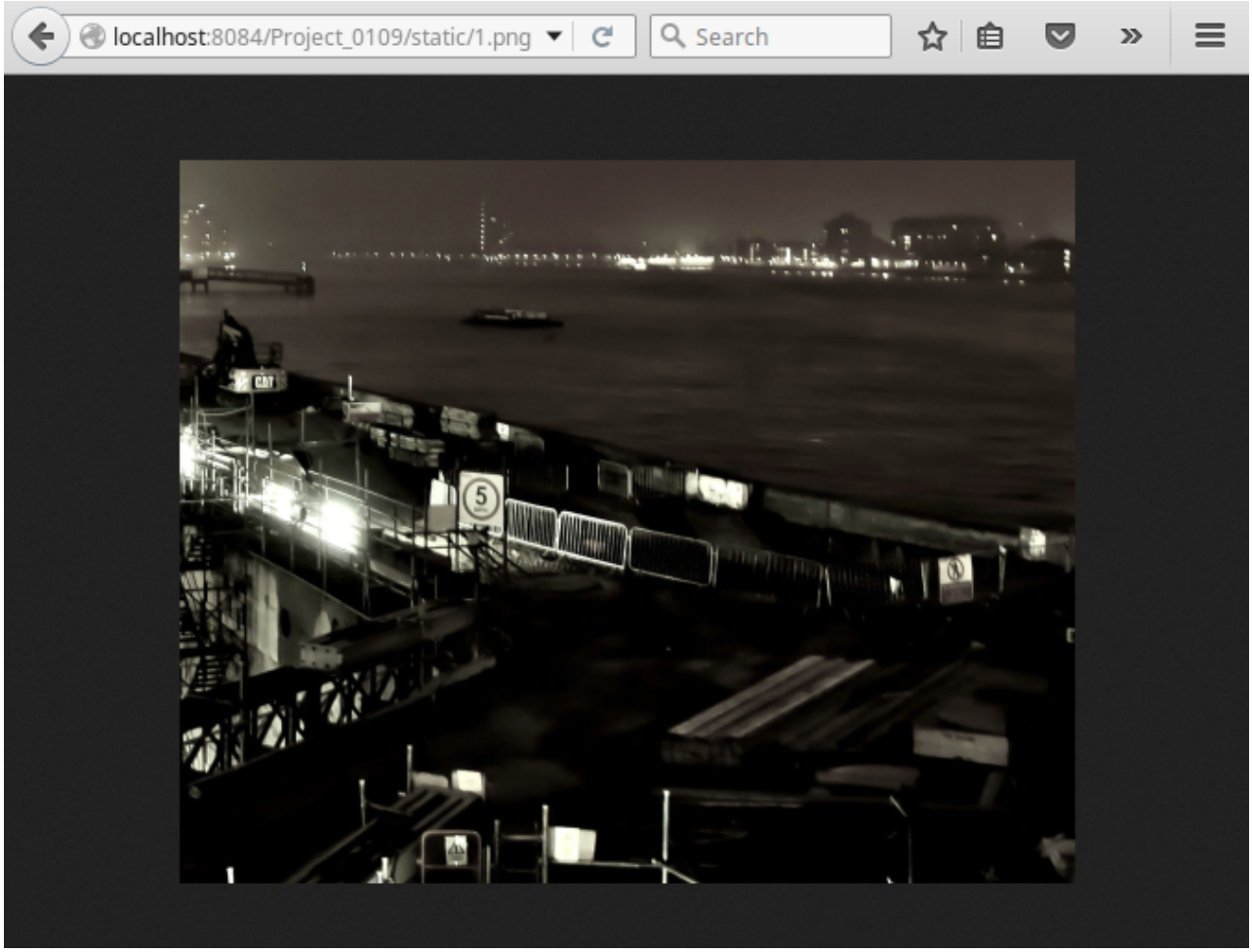
选择文件，指定名称：



上传成功：



查看上传的文件：



Tomcat输出：

```
文件原始名称：01.png
表单给定的文件名称：1.png
上传目录：/data/Code/netbeans/Project_0109/build/web/static
```

资料

《Spring MVC学习指南》 第11章

转换器与格式化

转换器：converter

格式化：Format

在01-06、校验器 (页 0)中的例子中都使用了数据绑定 （将表单数据绑定到bean对象中），例如：

```
@RequestMapping(value = "/output")
public String output(Person person, Model model) {
    model.addAttribute("person", person);
    return "hello/output";
}
```

表单的数据都是String类型，如果我们的bean类中的属性是其他类型，例如Date、int，这时候就需要写一个工具，将String转换成Date、int。

这就是转换器与格式化做的事情：类型转换。字符串转换成数字类型是内置的。

资料：

8. Validation, Data Binding, and Type Conversion

Spring MVC request parameter conversion with minimal configuration

Introduction to Spring Converters and Formatters

极客学院

jikexueyuan.com

中国最大的IT职业在线教育平台

更多信息请访问

http://wiki.jikexueyuan.com/project/java-web/