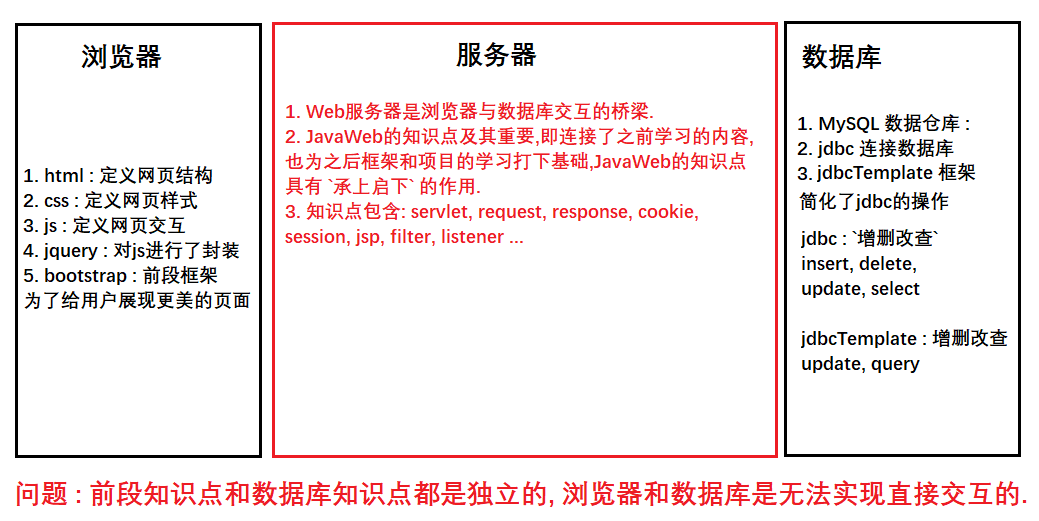
# 知识点回顾与Web服务器引入 :



# 软件架构模式 :

生活中，我们从网络中获取资源的方式通常有两种：

1. 通过浏览器去访问网络资源（B/S）；Broswer / Server

2. 打开特定的软件访问网络资源(C/S)；Client /Server

**BS架构 :** 又称Browser/Server架构，即浏览器和服务器架构模式。它是通过浏览器 和后台服务器 进行交互的，如淘宝，京东，12306等等。

1. 优点：

1. 用户只需要安装一个浏览器即可访问不同的服务器；

2. 系统更新和升级比较方便；

2. 缺点：

1. 动画效果受浏览器限制；

2. 网站的压力集中在服务器端。

**CS架构 :** 又称Client/Server架构，及客户端和服务器架构模式。它是通过客户端 和服务器 进行交互的，如QQ，迅雷，360等等。

1. 优点：

1. 客户端效果炫，用户体验好；

2. 客户端能够进行部分计算功能，减轻服务器的压力；

2. 缺点：

1. 需要安装客户端，占用硬盘空间；

2. 系统升级麻烦；

3. 不跨平台，如win版qq，mac版qq；

【结论】B/S架构将是未来软件架构的趋势。

大家每天都在使用京东, 淘宝, 基本都是直接使用浏览器实现访问与交互的, 在电脑的桌面端, 也从未出现过京东, 淘宝的客户端程序应用.

# BS通信模式 :



**小结 :**

BS通讯特点 :

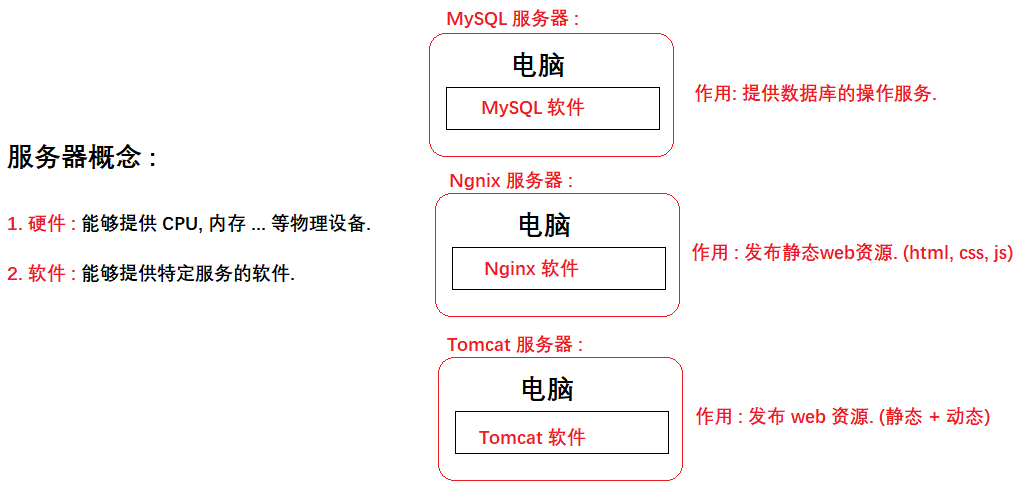
1. 一次请求, 一次响应.
2. 先有请求, 后有响应.
3. 请求和响应成对出现.

HTTP 协议底层依赖于 TCP 协议. TCP 协议就是请求和响应模型.

# URL 统一资源定位符 :



# Tomcat服务器简介 :



服务器就是一个软件，任何电脑只需要安装上了服务器软件，然后该电脑的指定目录下的资源就能提供对外访问。

tomcat服务器是一个免费的开源的web应用服务器。是Apache软件基金会的Jakarta项目中的一个核心项目，由Apache，Sun和其他一些公司及个人共同开发而成。由于有了Sun的参与和支持，最新的Servlet和JSP规范总是能在Tomcat中得到体现。因为tomcat技术先进，性能稳定，而且免费。因而深受Java爱好者的喜爱，是目前比较流行的Web应用服务器。

​

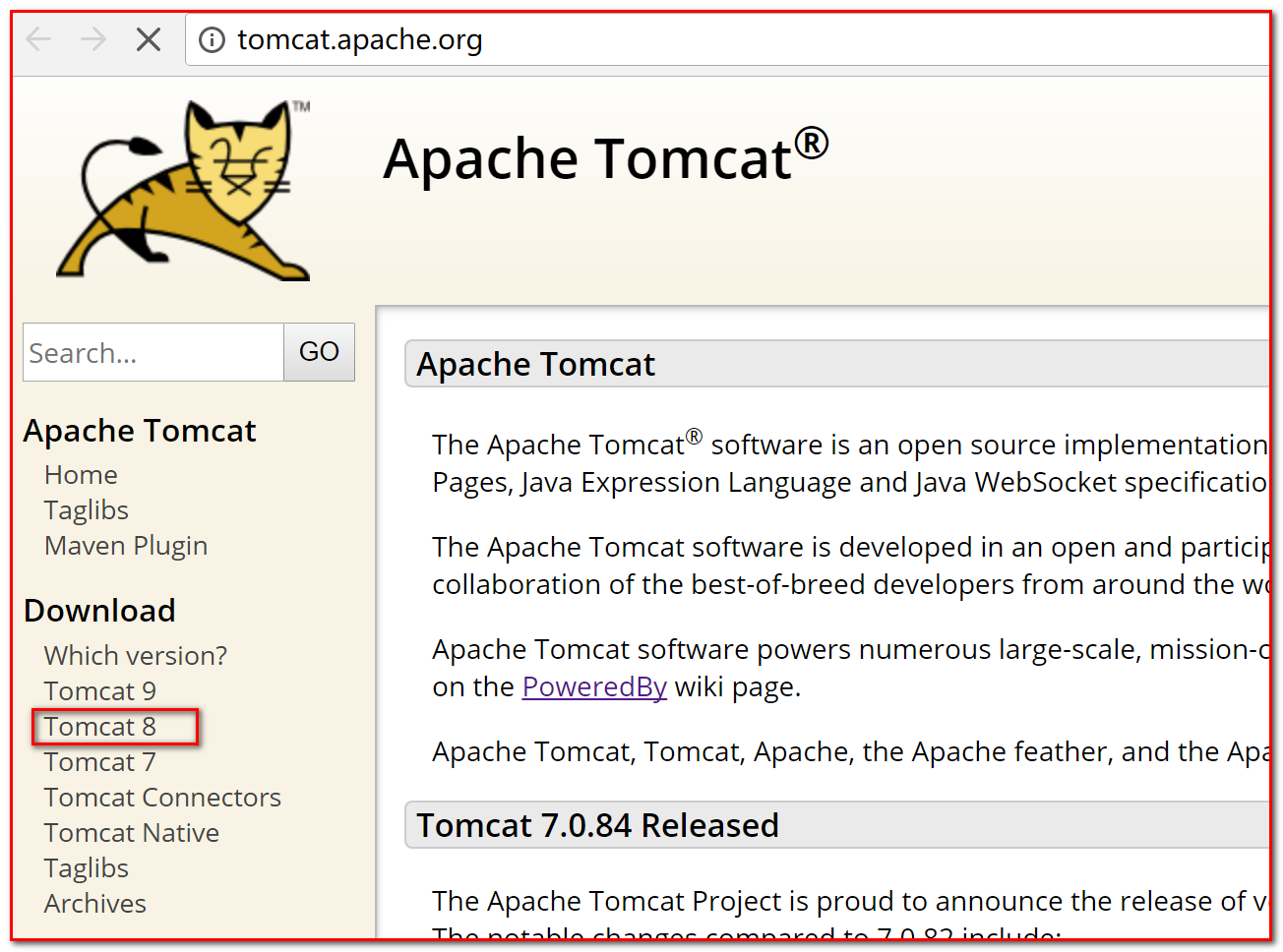
【作用】 运行web项目。

**了解 :** 常见的web服务器软件

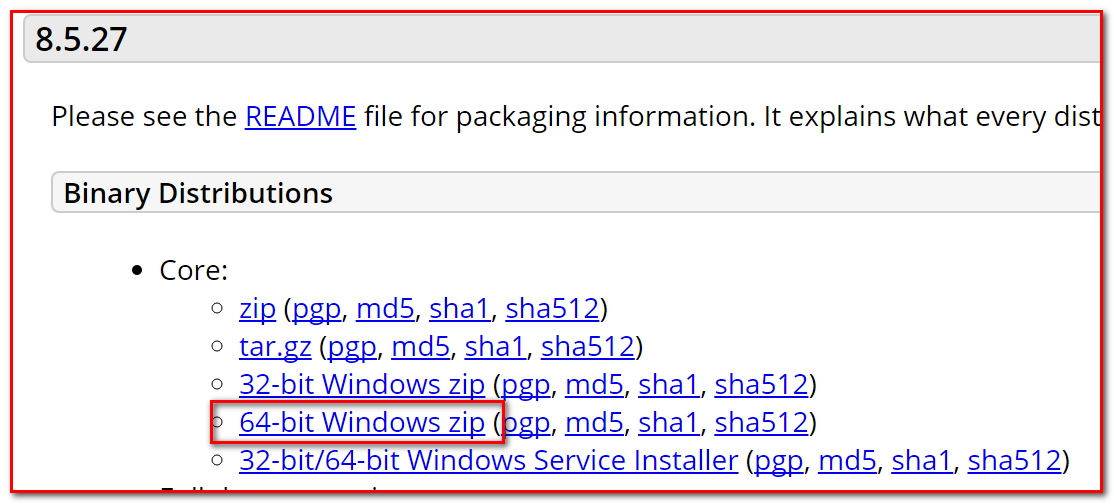
1. Tomcat：Apache组织提供一个免费开源的小型的服务器软件。支持Servlet和JSP规范。
2. WebLogic：Bea公司的一个收费的大型的服务器软件，后被Oracle收购。支持EE的所有的规范
3. WebSphere：IBM公司的一个收费的大型的服务器软件，支持EE的所有的规范。
4. JBoss：是一个基于J2EE的开放源代码的应用服务器。JBoss是一个管理EJB的容器和服务器，JBoss核心服务不包括支持servlet/JSP的WEB容器，一般与Tomcat或Jetty绑定使用。

# 6. Tomcat下载,安装与测试 :

**步骤一 :** 先去官网下载：<http://tomcat.apache.org/>，选择tomcat8版本（红框所示）：



**步骤二 :** 选择要下载的文件（红框所示）



tar.gz 文件 是linux操作系统下的安装版本

exe文件是window操作系统下的安装版本

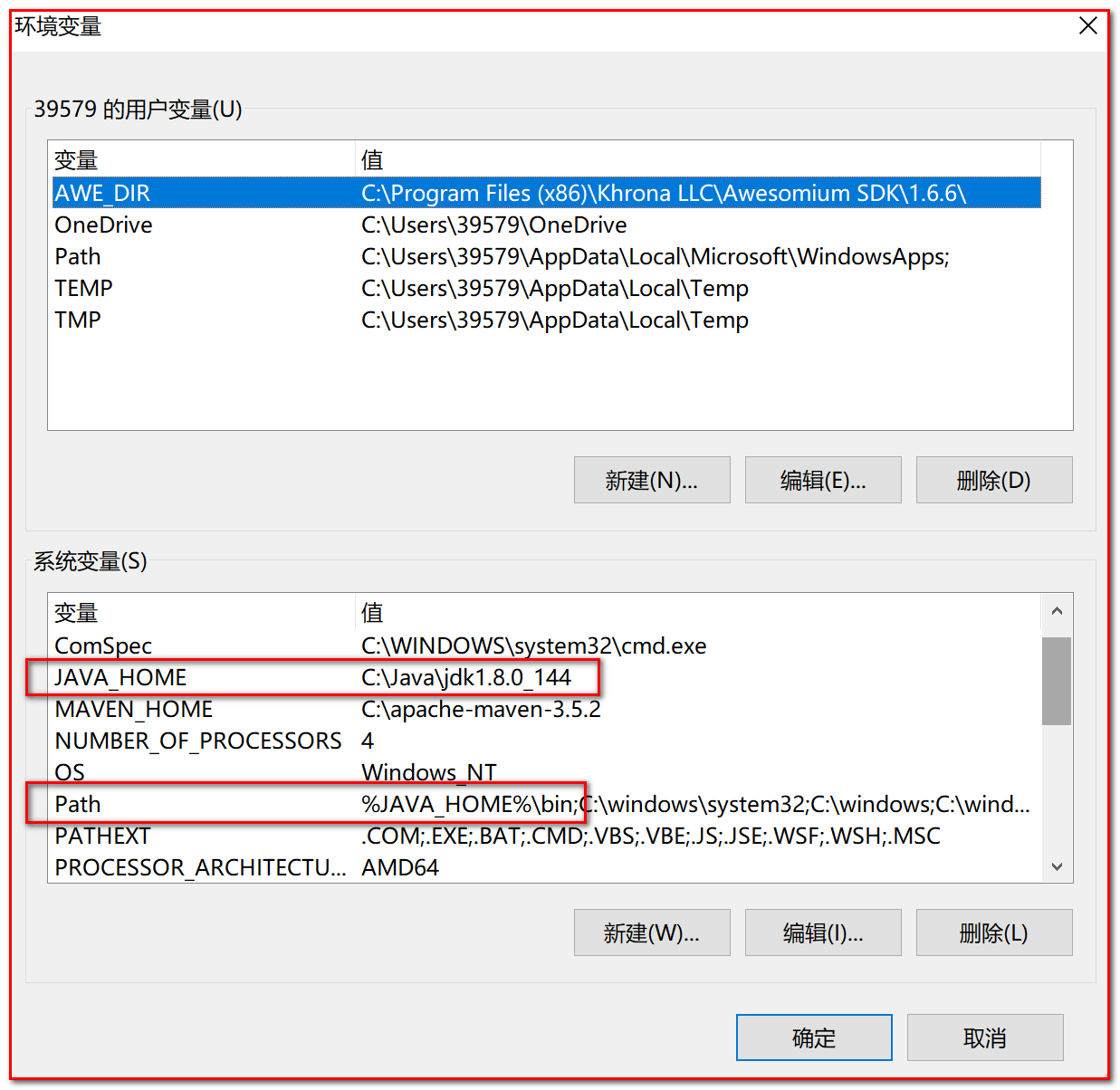
zip文件是window操作系统下压缩版本（我们选择zip文件）

**步骤三 :** 下载完成.



**步骤四 :** 安装.

1. **直接解压当前这个tomcat压缩包**：
2. **配置环境变量**：tomcat运行依赖于java环境：

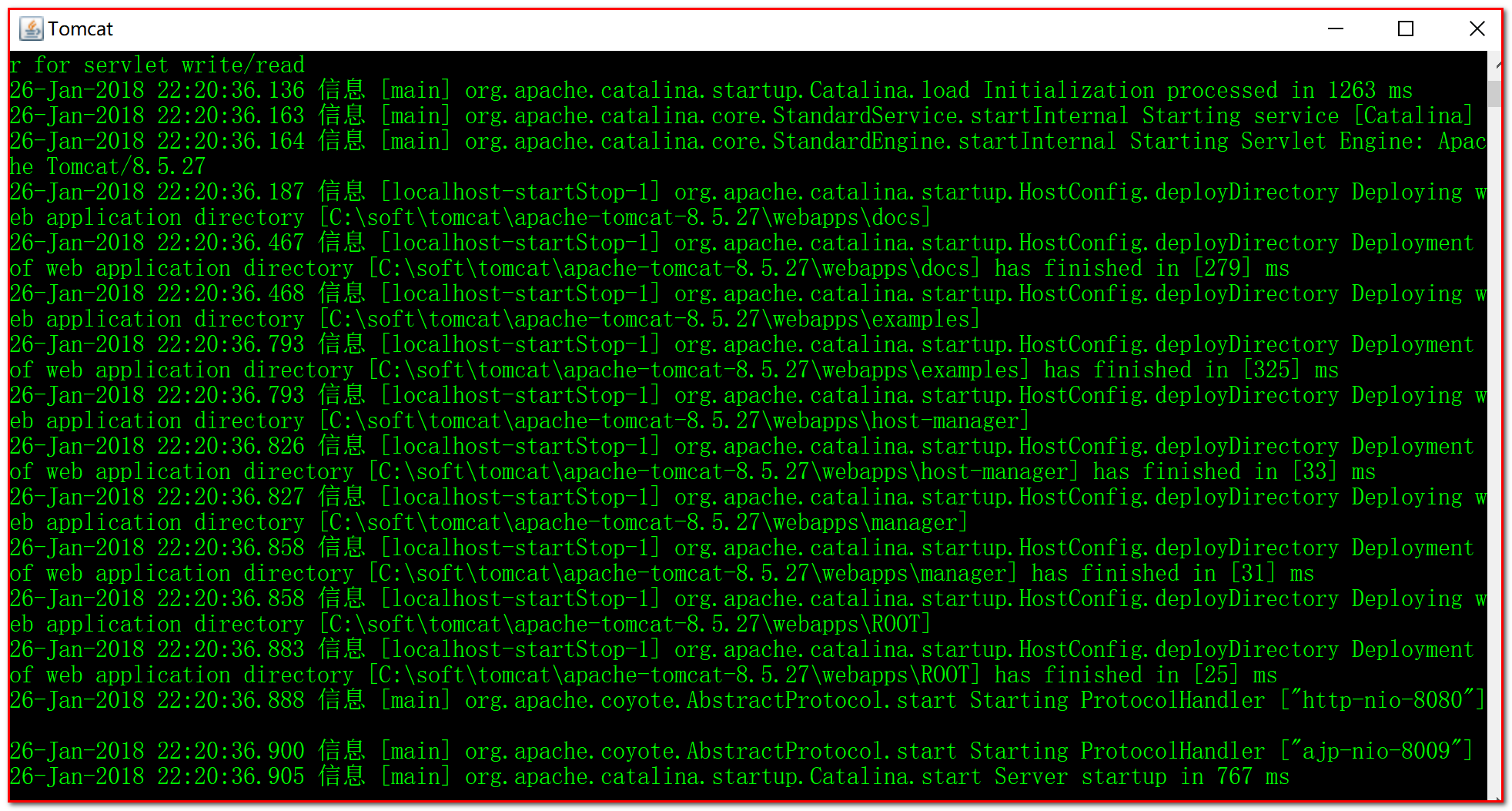


**步骤五 :** 启动tomcat服务器

查找tomcat目录下bin目录，查找其中的startup.bat命令，双击启动服务器：



**启动效果 ：**

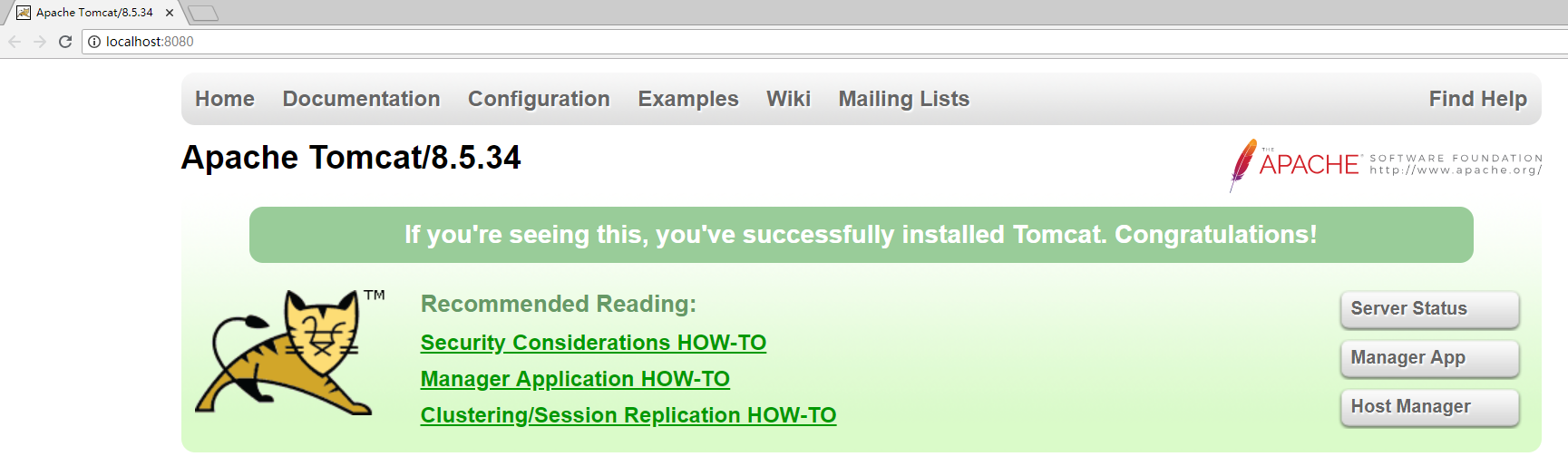


**步骤六 : 测试访问tomcat服务器**

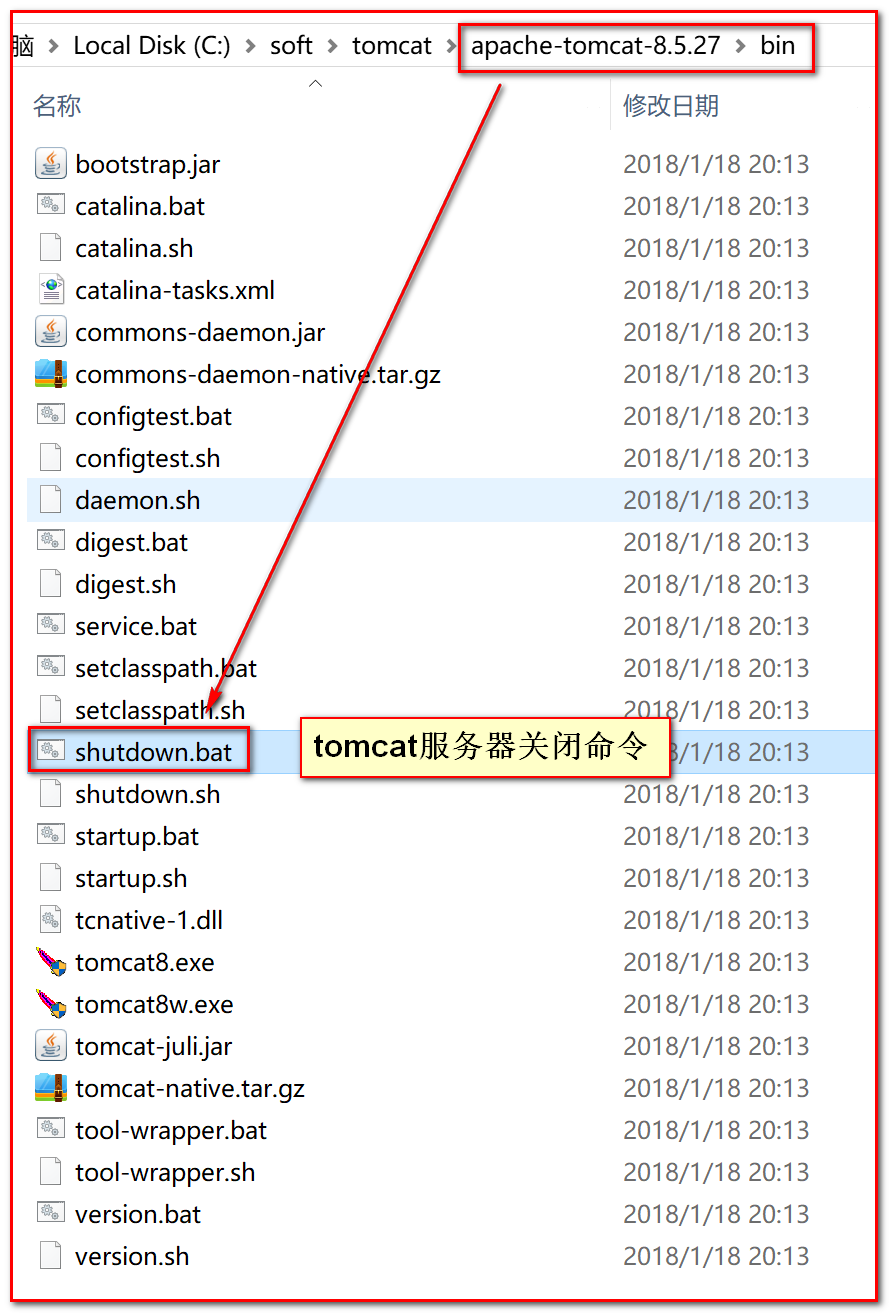
打开浏览器在，在浏览器的地址栏中输入：

​<http://127.0.0.1:8080>

​<http://localhost:8080>

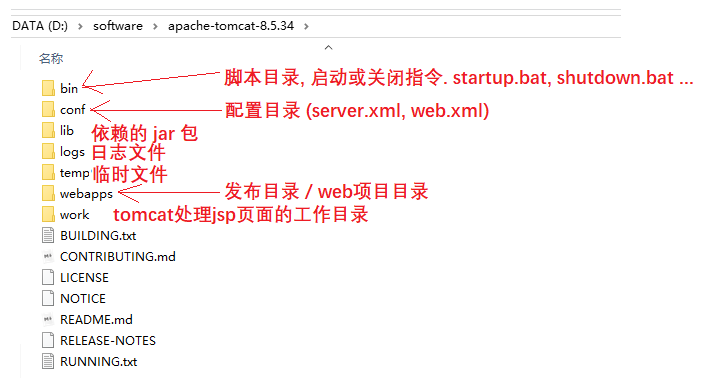


**步骤七 :** 关闭tomcat服务器

常用方式1 : 查找tomcat目录下bin目录，查找其中的shutdown.bat命令，双击关闭服务器：

常用方式2 : 可以点击cmd窗口的退出按钮

# 7. Tomcat安装目录介绍 :

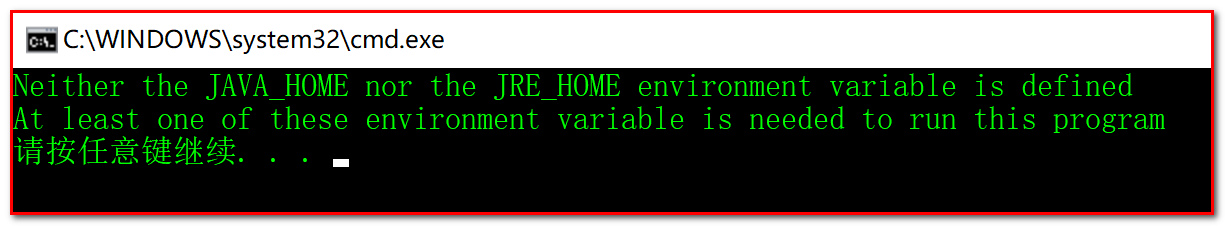


# 8. Tomcat 启动失败的常见原因 :

**1. 无法启动（闪退：cmd命令窗口出现一下，就消失）：**

主要原因：没有配置JAVA\_HOME环境变量。JAVA\_HOME 环境变量中配置的是JDK的安装目录，不包含bin目录，不是tomcat的安装目录。

闪退的原因查看：可以在startup.bat文件末尾书写pause命令。让运行的窗口暂停。

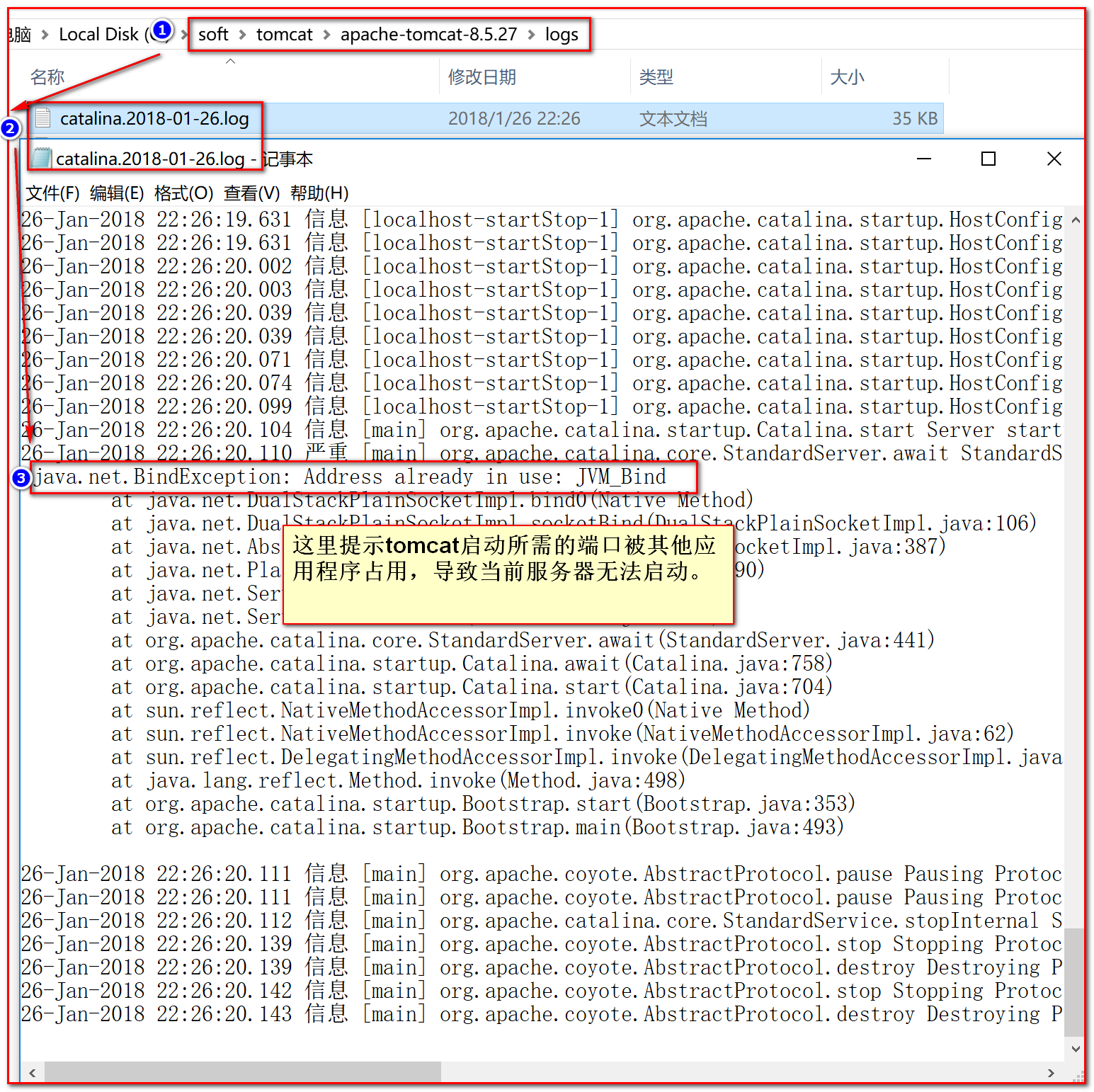


**2. 端口被占用导致启动失败**

如果启动的时候，发生异常问题，这时有可能是端口被占用。

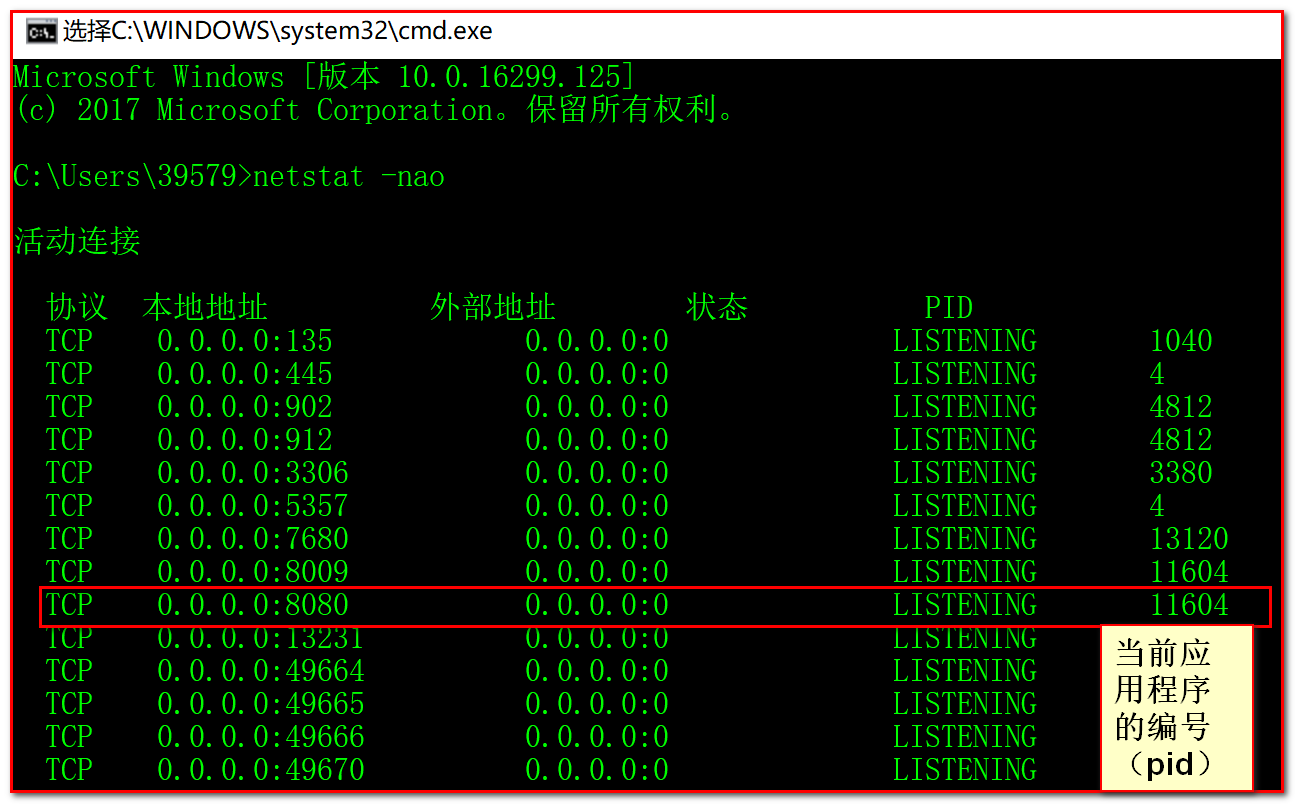
Tomcat服务器在启动的时候默认占用本地的8080端口，如果这个端口被占用，启动的时候就会报错。

报错内容可以通过查询tomcat目录下的logs目录中Catalina.当前系统年月日.log文件查看，如下图：

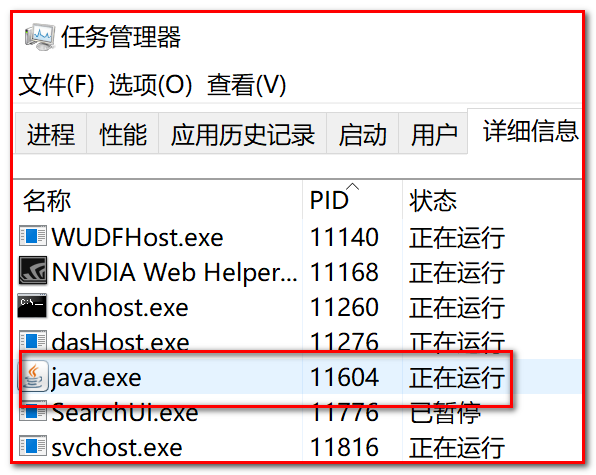


我们需要做的是查看本地端口使用情况，关闭占用端口的程序：

在dos窗口中输入 netstat –nao 就可以查看当前端口的占用情况：



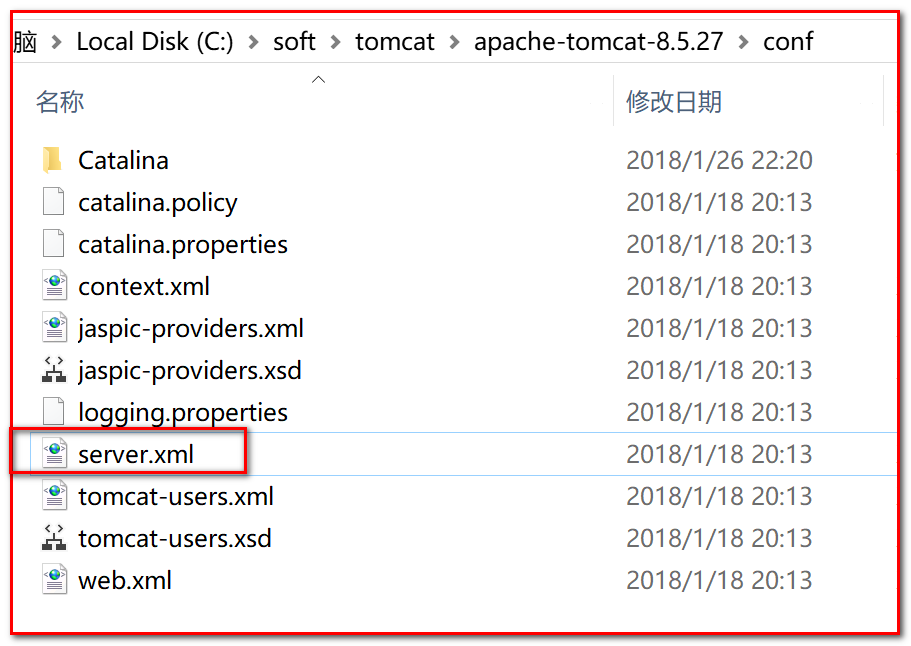
使用任务管理器---->选择详细信息---->查看pid为11604（查询出来的值），关闭结束进程即可：



注意 ：如果这个进程是操作系统的任务进程，这时一般是不能停止这个进程。如果是系统进程占用端口，那么我们只能换一个端口. 下面看如何修改端口：

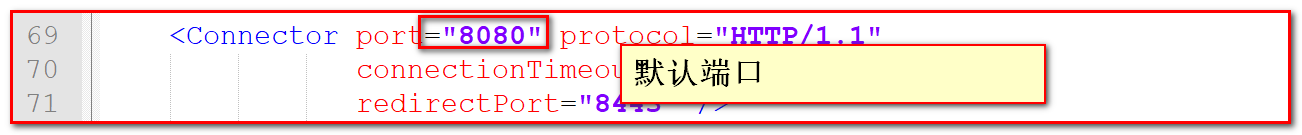
**3. 修改tomcat启动端口.**

Tomcat服务器的配置文件，全部都在tomcat的安装目录下conf目录下：



server.xml : 服务器端口配置、服务器自身配置文件

​ 打开server.xml，修改服务器端口：



修改tomcat的端口为9090：

​ 修改完server.xml文件必须重启服务器才能有效。​

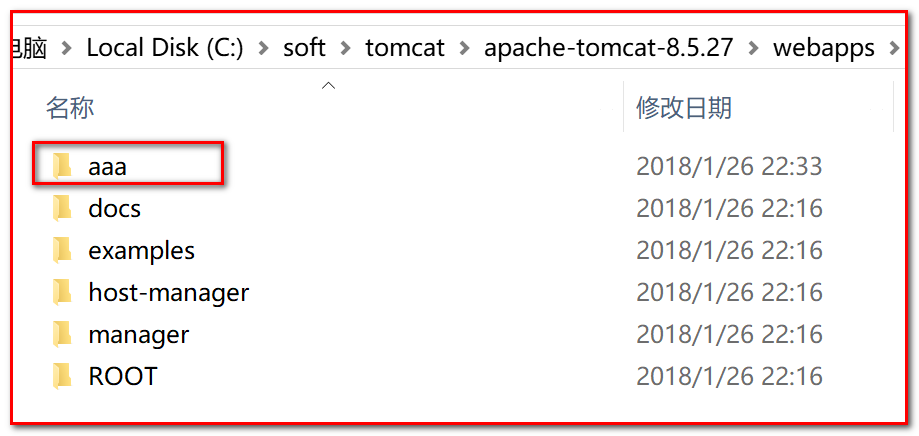
通过浏览器的地址栏访问测试：<http://localhost:9090>

# 9. Tomcat 发布项目 :

**1. 在webapps文件夹下直接发布.**

只要将准备好的web资源直接复制到tomcat/webapps文件夹下，就可以通过浏览器使用http协议访问获取.

一 : 创建aaa文件夹：

二 : 定义一个index.html文件，内容为：

<html>

<head>

<title>百度一下, 你就知道</title>

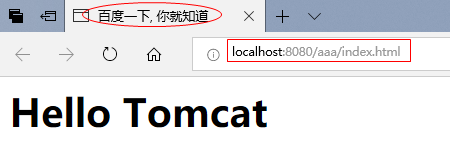
</head>

<body>

<h1>Hello Tomcat</h1>

</body>

</html>



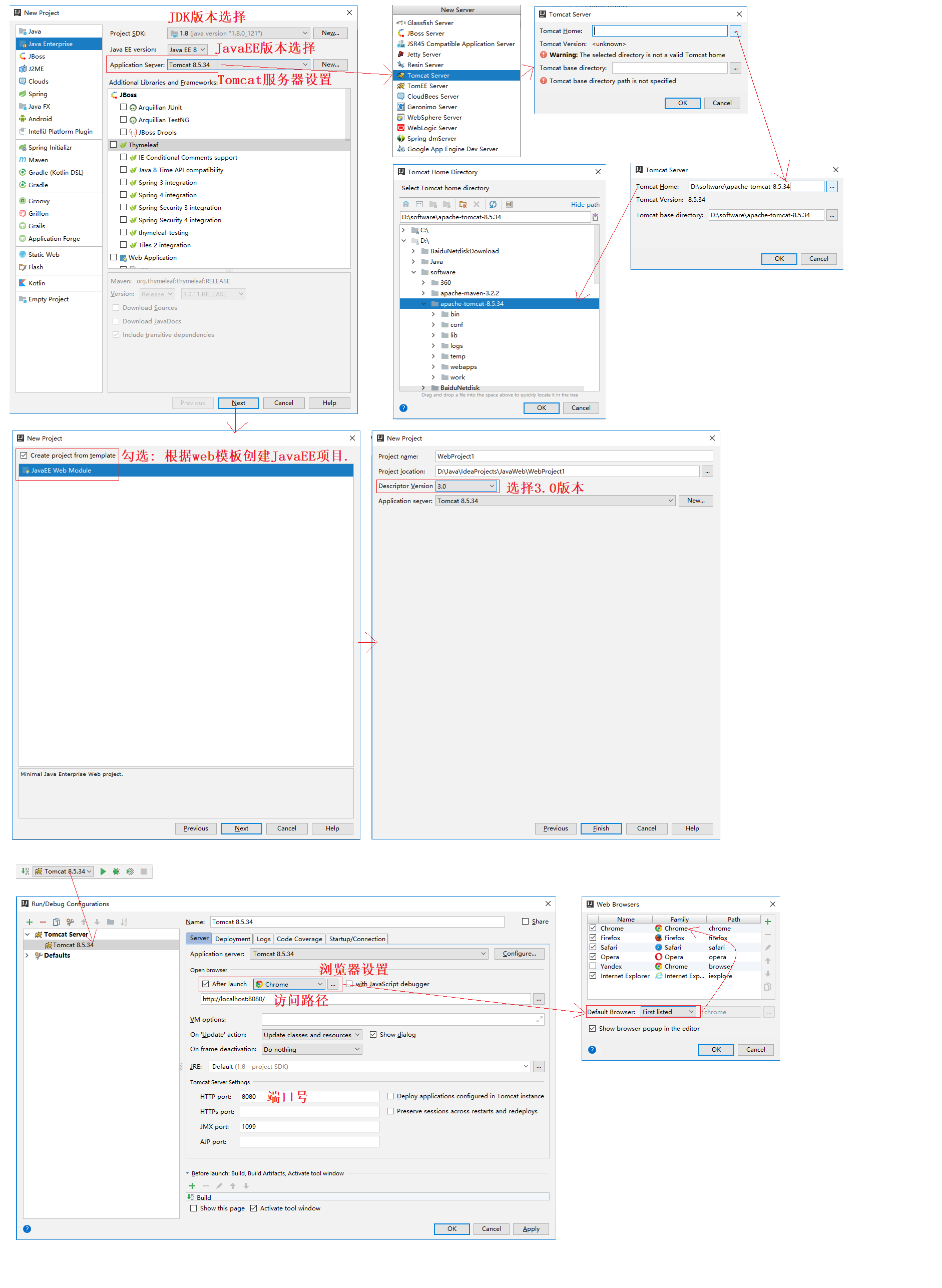
**2. war 包发布 :**

使用压缩软件压缩成zip包，然后修改后缀名为war包，放到webapp目录下.

**3. idea配置Tomcat直接发布 :**

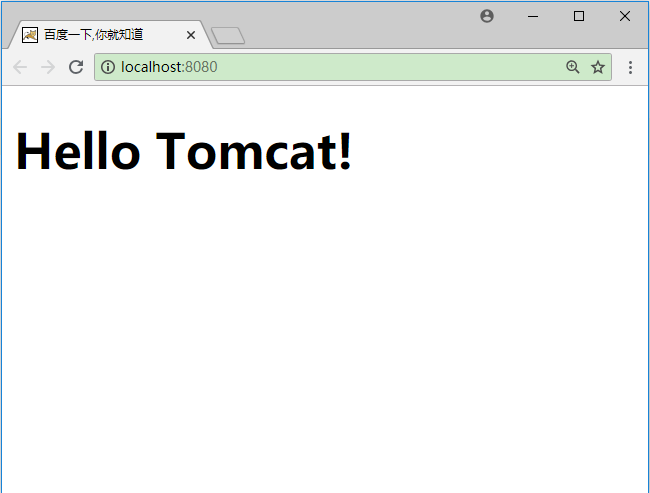
# 10. IDEA集成Tomcat服务器 :

因为我们开发的时候都是用IDE,所有我们要将idea和tomcat集成到一起，可以通过idea就控制tomcat的启动和关闭：



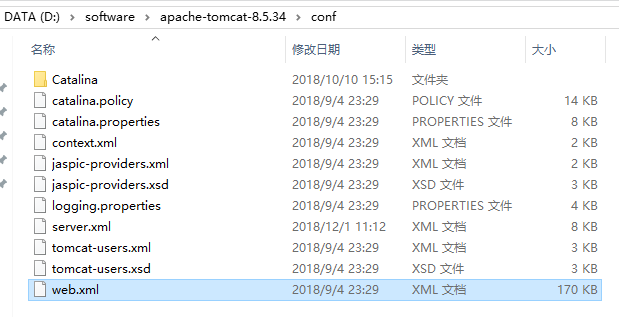
编写一个 index.html 页面, 使用 Tomcat 启动并发布项目 :

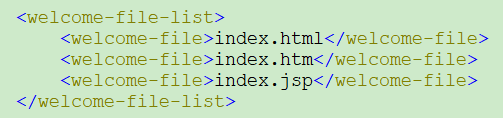
<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <title>百度一下,你就知道</title>  
</head>  
<body>  
 <h1>Hello Tomcat!</h1>  
</body>  
</html>



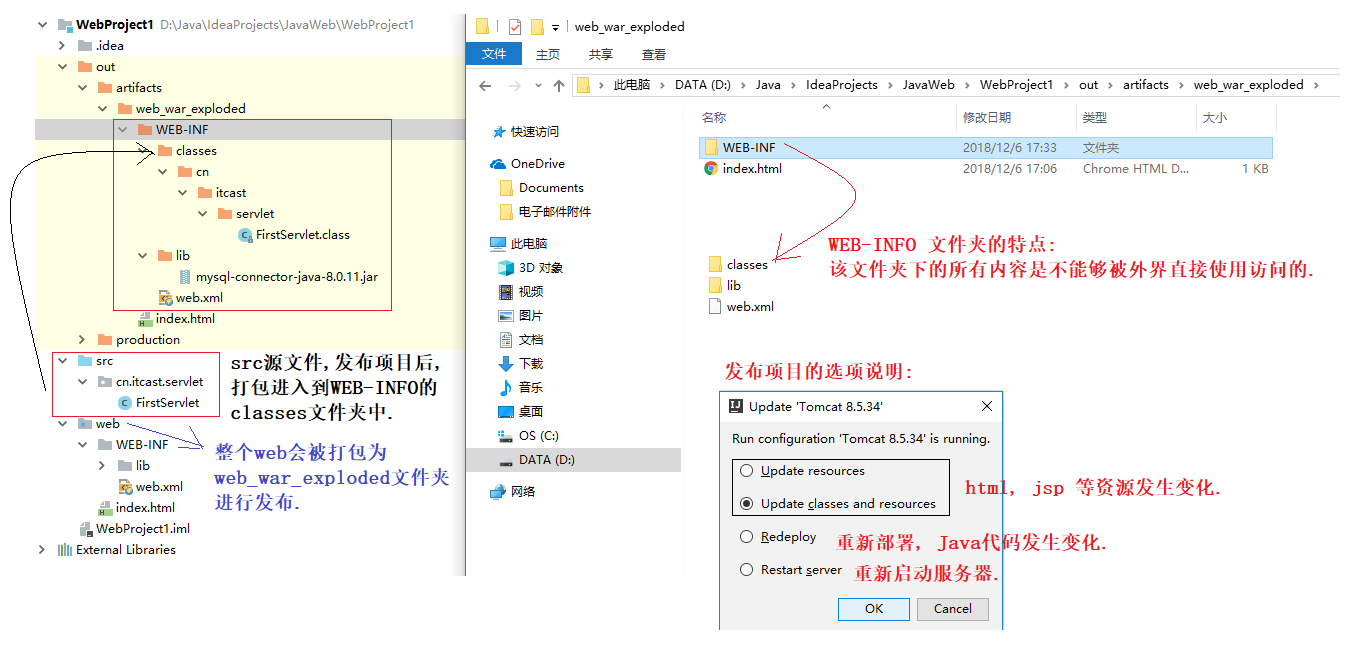
# 11. 项目结构介绍 :

**请问 :** 为什么启动项目就自动访问了 index.html 静态页面呢 ???





**说明 :** 该文件是整体 Tomcat 服务器的配置文件. 而项目中的 web.xml 配置文件也是基于 conf目录下的 web.xml 文件的.



# 12. Servlet入门程序 :

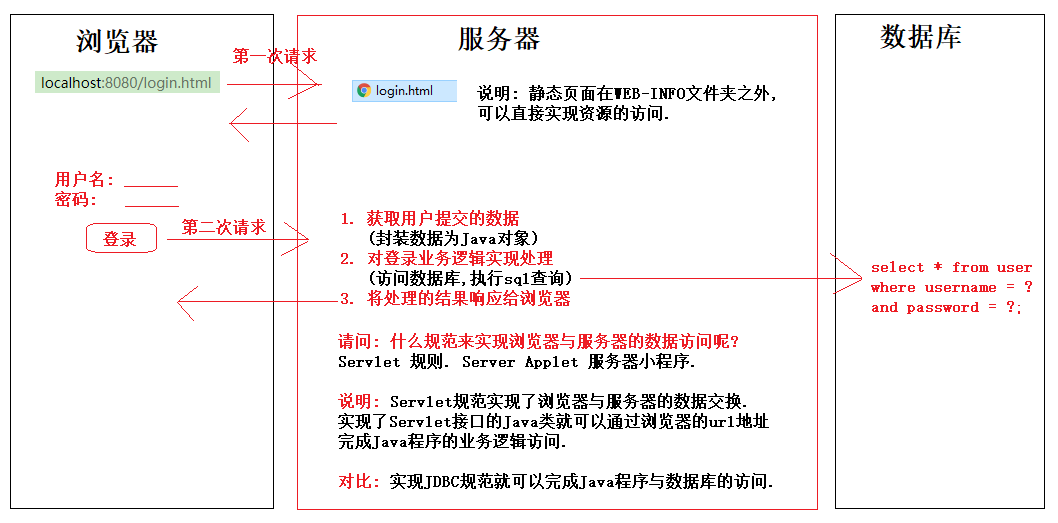
**目标 :** 浏览器实现Java程序访问.

public class LoginServlet {  
 public static void main(String[] args) {  
  
 *// 1. 获取用户提交的数据  
  
 // 2. 对登录业务逻辑实现处理  
  
 // 3. 将处理结果响应给浏览器* }  
}

**目标的实现说明与步骤详解 :**

**说明 :** 普通类(直接继承自Object类)是不能够直接通过浏览器的url实现Java类代码访问,并执行业务逻辑处理.

那么, 请问, 谁能够完成这一使命呢 ???



**Servlet实现的步骤说明 :**

1. 遵守规范, 实现 Servlet 接口. 重写接口中定义的抽象方法.
2. 在web.xml文件中配置Servlet的具体信息.

<servlet>

<servlet-name>自定义名称</servlet-name>

<servlet-class>包名+类名</servlet-class>

</sevlet>

<servlet-mapping>

<servlet-name>自定义名称</servlet-name>

<url-pattern>浏览器访问的映射路径</url-pattern>

</servlet-mapping>

1. 在service方法中实现程序的业务逻辑处理.
2. 启动 tomcat, 根据配置文件的映射路径, 实现servlet程序的访问.

**实现 :**

import javax.servlet.\*;  
import java.io.IOException;  
  
public class FirstServlet implements Servlet {  
 @Override  
 public void init(ServletConfig servletConfig) throws ServletException {  
  
 }  
  
 @Override  
 public ServletConfig getServletConfig() {  
 return null;  
 }  
  
 @Override  
 public void service(ServletRequest servletRequest, ServletResponse servletResponse) throws ServletException, IOException {  
  
 *// 1. 获取用户提交的数据* System.*out*.println("1.获取用户提交的数据");  
  
 *// 2. 对登录业务逻辑实现处理* System.*out*.println("2.对登录业务逻辑实现处理");  
  
 *// 3. 将处理结果响应给浏览器* System.*out*.println("3.将处理结果响应给浏览器");  
 }  
  
 @Override  
 public String getServletInfo() {  
 return null;  
 }  
  
 @Override  
 public void destroy() {  
  
 }  
}

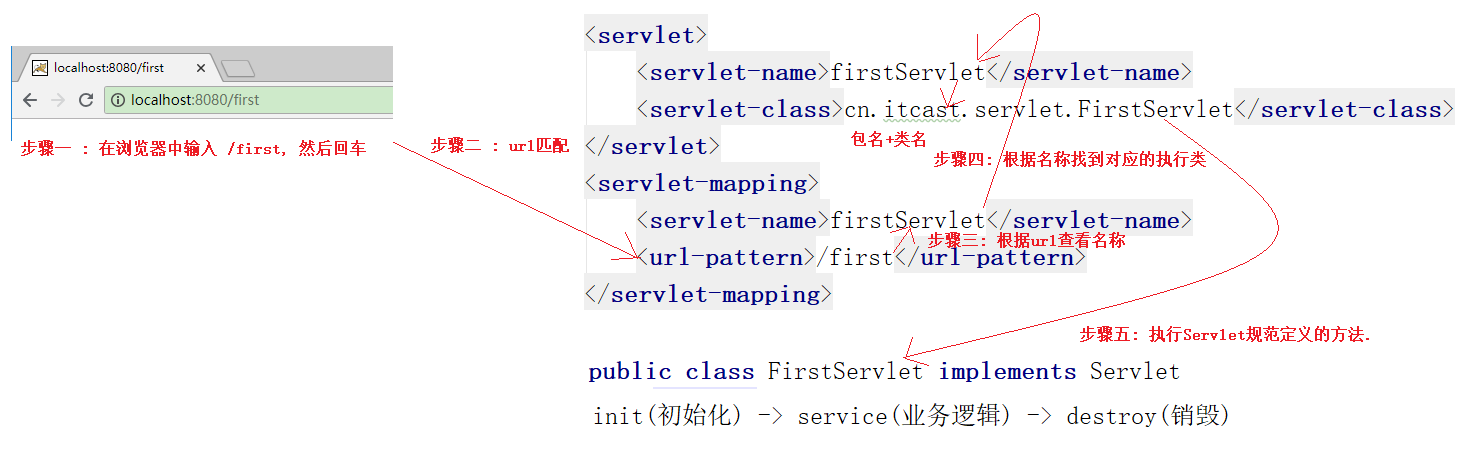
**web.xml 配置文件 :**

*<?*xml version="1.0" encoding="UTF-8"*?>*<web-app xmlns="http://java.sun.com/xml/ns/javaee"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://java.sun.com/xml/ns/javaee  
 http://java.sun.com/xml/ns/javaee/web-app\_3\_0.xsd"  
 version="3.0">  
  
 <servlet>  
 <servlet-name>firstServlet</servlet-name>  
 <servlet-class>cn.itcast.servlet.FirstServlet</servlet-class>  
 </servlet>  
 <servlet-mapping>  
 <servlet-name>firstServlet</servlet-name>  
 <url-pattern>/first</url-pattern>  
 </servlet-mapping>  
  
</web-app>

**小结 :**

Servlet规范 : Servlet就是一个运行在tomcat容器中的应用小程序, 可以直接通过浏览器的url路劲来完成Java代码的访问与执行.

# 13. 访问与执行流程说明 :



# 14. Servlet优化 : GenericServlet

**目标 :** 使用Generic优化Servlet规范的代码编写.

**步骤 :**

一 : 自定义类实现Servlet接口.

二 : 重写 Servlet 接口中的所有抽象方法.

三 : 将 service 方法定义为抽象方法, 因为开发中最重要的步骤就是业务逻辑的实现.

四 : 父类也可以提供一些扩展方法, 方便子类继承时实现直接使用. 如: log 方法等…

五 : 完成上述步骤后, 编写 SecondServlet 类, 继承自 GenericServlet 类, 重写 service 抽象方法即可.

六 : 在 web.xml 文件中配置 SecondServlet 的访问映射路径.

七 : 浏览器访问实现了Servlet规范的Java程序.

**实现 :**

import javax.servlet.\*;  
import java.io.IOException;  
  
public abstract class GenericServlet implements Servlet {  
 @Override  
 public void init(ServletConfig servletConfig) throws ServletException {  
  
 }  
  
 @Override  
 public ServletConfig getServletConfig() {  
 return null;  
 }  
  
 @Override  
 public abstract void service(ServletRequest servletRequest, ServletResponse servletResponse) throws ServletException, IOException;  
  
 @Override  
 public String getServletInfo() {  
 return null;  
 }  
  
 @Override  
 public void destroy() {  
  
 }  
  
 *// 扩展方法 :* public void log() {  
  
 }  
}

import javax.servlet.ServletException;  
import javax.servlet.ServletRequest;  
import javax.servlet.ServletResponse;  
import java.io.IOException;  
  
public class SecondServlet extends GenericServlet {  
 @Override  
 public void service(ServletRequest servletRequest, ServletResponse servletResponse) throws ServletException, IOException {  
 *// 1. 获取用户提交的数据* System.*out*.println("1.获取用户提交的数据");  
  
 *// 2. 对登录业务逻辑实现处理* System.*out*.println("2.对登录业务逻辑实现处理");  
  
 *// 3. 将处理结果响应给浏览器* System.*out*.println("3.将处理结果响应给浏览器");  
 }  
}

**web.xml 配置文件 :**

<servlet>  
 <servlet-name>secondServlet</servlet-name>  
 <servlet-class>cn.itcast.servlet.SecondServlet</servlet-class>  
</servlet>  
<servlet-mapping>  
 <servlet-name>secondServlet</servlet-name>  
 <url-pattern>/second</url-pattern>  
</servlet-mapping>

**小结 :**

使用继承 GenericServlet 类的方式, 可以将代码的重点关注在 `真正的业务逻辑` 实现上. 简化了 Servlet 规范的具体代码编写.

# 15. Servlet优化 : HttpServlet

**目标 :** 使用HttpServlet优化GenericServlet代码的编写.

**步骤 :**

一 : 编写 login.html 静态页面. 提供表单提交数据. (GET / POST两种方式)

二 : 自定义类继承 GenericServlet类.

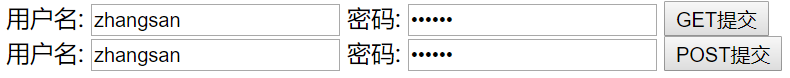
三 : 重写 serivce抽象方法. 将方法划分为 get / post 请求对应的方法. 因为浏览器与服务器交互的协议为 http 协议. 而表单提交数据的大部分方式都为 get / post 请求方法.

四 : 完成上述步骤后, 编写 ThreeServlet 类, 继承自 HttpServlet 类, 重写 get / post方法即可.

**实现 :**

login.html 页面 :

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <title>登录</title>  
</head>  
<body>  
 <form action="/three" method="get">  
 用户名: <input type="text" name="username" value="zhangsan">  
 密码: <input type="password" name="password" value="123456">  
 <input type="submit" value="GET提交">  
 </form>  
  
 <form action="/three" method="post">  
 用户名: <input type="text" name="username" value="zhangsan">  
 密码: <input type="password" name="password" value="123456">  
 <input type="submit" value="POST提交">  
 </form>  
</body>  
</html>



import javax.servlet.ServletException;  
import javax.servlet.ServletRequest;  
import javax.servlet.ServletResponse;  
import javax.servlet.http.HttpServletRequest;  
import javax.servlet.http.HttpServletResponse;  
import java.io.IOException;  
  
public class HttpServlet extends GenericServlet {  
 @Override  
 public void service(ServletRequest servletRequest, ServletResponse servletResponse) throws ServletException, IOException {  
 *// 1. 将 request 和 response 转换为 http 协议类型* HttpServletRequest request = (HttpServletRequest) servletRequest;  
 HttpServletResponse response = (HttpServletResponse) servletResponse;  
  
 this.service(request, response);  
 }  
  
 public void service(HttpServletRequest request, HttpServletResponse response) {  
  
 *// 1. 获取请求的方式* String method = request.getMethod();  
  
 *// 3. 判断* if ("GET".equals(method)) {  
 this.doGet(request, response);  
 } else if ("POST".equals(method)) {  
 this.doPost(request, response);  
 }  
 }  
  
 public void doGet(HttpServletRequest request, HttpServletResponse response) {  
 *// 执行 http 协议 get 请求的设置* }  
  
 public void doPost(HttpServletRequest request, HttpServletResponse response) {  
 *// 执行 http 协议 post 请求的设置* }  
}

import javax.servlet.http.HttpServlet;  
import javax.servlet.http.HttpServletRequest;  
import javax.servlet.http.HttpServletResponse;  
  
public class ThreeServlet extends HttpServlet {  
 @Override  
 public void doGet(HttpServletRequest request, HttpServletResponse response) {  
 System.*out*.println("GET 请求 ...");  
 }  
  
 @Override  
 public void doPost(HttpServletRequest request, HttpServletResponse response) {  
 System.*out*.println("POST 请求 ...");  
 }  
}

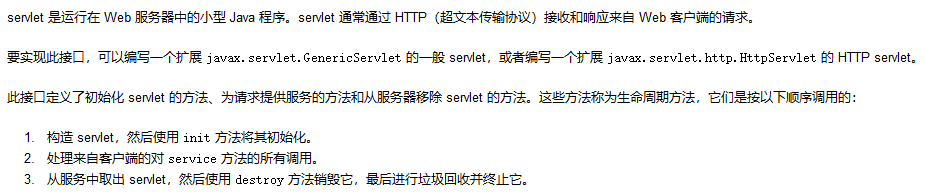
配置文件 :

<servlet>  
 <servlet-name>threeServlet</servlet-name>  
 <servlet-class>cn.itcast.servlet.ThreeServlet</servlet-class>  
</servlet>  
<servlet-mapping>  
 <servlet-name>threeServlet</servlet-name>  
 <url-pattern>/three</url-pattern>  
</servlet-mapping>

**小结 :**

使用 HttpServlet 类, 能够将 GenericServlet 类实现 http 协议的特性划分, 更加面向 http 协议开发.

# 16. Servlet的生命周期 :



import javax.servlet.\*;  
import java.io.IOException;  
  
public class LifeCircleServlet implements Servlet {  
 @Override  
 public void init(ServletConfig servletConfig) throws ServletException {  
 System.*out*.println("init 初始化 ...");  
 }  
  
 @Override  
 public ServletConfig getServletConfig() {  
 return null;  
 }  
  
 @Override  
 public void service(ServletRequest servletRequest, ServletResponse servletResponse) throws ServletException, IOException {  
 System.*out*.println("service 业务逻辑方法 ...");  
 }  
  
 @Override  
 public String getServletInfo() {  
 return null;  
 }  
  
 @Override  
 public void destroy() {  
 System.*out*.println("destroy 销毁方法 ...");  
 }  
}

配置文件 :

<servlet>  
 <servlet-name>lifeCircleServlet</servlet-name>  
 <servlet-class>cn.itcast.servlet.LifeCircleServlet</servlet-class>  
</servlet>  
<servlet-mapping>  
 <servlet-name>lifeCircleServlet</servlet-name>  
 <url-pattern>/lifeCircle</url-pattern>  
</servlet-mapping>

**小结 :**

init 第一次访问该 servlet 时执行.

service每次访问都会执行一次.

destroy关闭服务器时执行一次.

# 17. Servlet映射路径的配置规范 :

在学习Servlet的时候，每个Servlet程序都需要在当前的这个项目的web.xml文件中注册和映射。

在映射中书写的url-pattern标签的书写方式：

**1、全路径匹配**

在书写url-pattern的时候，必须以/开始，后面书写具体浏览器访问时的路径。

<servlet>  
 <servlet-name>pathServlet</servlet-name>  
 <servlet-class>cn.itcast.servlet.PathServlet</servlet-class>  
</servlet>  
<servlet-mapping>  
 <servlet-name>pathServlet</servlet-name>  
 *<!--全路径匹配-->* <url-pattern>/path</url-pattern>  
</servlet-mapping>

<form action="/path" method="get">  
 用户名: <input type="text" name="username" value="zhangsan">  
 密码: <input type="password" name="password" value="123456">  
 <input type="submit" value="GET提交">  
</form>  
  
<form action="/path" method="post">  
 用户名: <input type="text" name="username" value="zhangsan">  
 密码: <input type="password" name="password" value="123456">  
 <input type="submit" value="POST提交">  
</form>

外界的访问方式： http://localhost:8080/path

**2、路径通配符匹配**

在书写url-pattern 的时候，以/开始，后面可以使用\*号表示任意的匹配

<servlet>  
 <servlet-name>pathServlet</servlet-name>  
 <servlet-class>cn.itcast.servlet.PathServlet</servlet-class>  
</servlet>  
<servlet-mapping>  
 <servlet-name>pathServlet</servlet-name>  
 *<!--通配符匹配-->* <url-pattern>/path/\*</url-pattern>  
</servlet-mapping>

<form action="/path/abc" method="get">  
 用户名: <input type="text" name="username" value="zhangsan">  
 密码: <input type="password" name="password" value="123456">  
 <input type="submit" value="GET提交">  
</form>  
  
<form action="/path/aaa/bbb/ccc" method="post">  
 用户名: <input type="text" name="username" value="zhangsan">  
 密码: <input type="password" name="password" value="123456">  
 <input type="submit" value="POST提交">  
</form>

外界在访问的时候，只要能够和/url2匹配上，后面写任何东西都可以

http://localhost:8080/path/abc

**3、扩展名匹配**

在书写url-pattern 的时候，不能以/开始，以\*开始，后面书写扩展名

<servlet>  
 <servlet-name>pathServlet</servlet-name>  
 <servlet-class>cn.itcast.servlet.PathServlet</servlet-class>  
</servlet>  
<servlet-mapping>  
 <servlet-name>pathServlet</servlet-name>  
 *<!--扩展名匹配 : 注意, 不要带 / -->* <url-pattern>\*.action</url-pattern>  
</servlet-mapping>

<form action="path/abc.action" method="get">  
 用户名: <input type="text" name="username" value="zhangsan">  
 密码: <input type="password" name="password" value="123456">  
 <input type="submit" value="GET提交">  
</form>  
  
<form action="path.action" method="post">  
 用户名: <input type="text" name="username" value="zhangsan">  
 密码: <input type="password" name="password" value="123456">  
 <input type="submit" value="POST提交">  
</form>

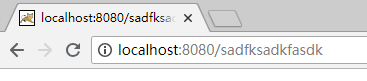
常见的扩展名书写：\*.action \*.do \*.go

访问的方式： http://localhost:8080/path/abc.action

**4、缺省路径**

在书写url-pattern 的时候， /，访问的路径找不到,就会去找缺省路径.

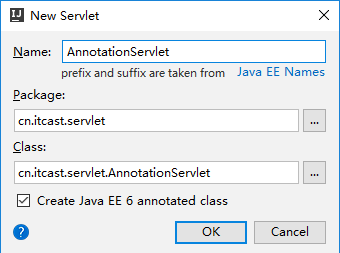
<servlet>  
 <servlet-name>pathServlet</servlet-name>  
 <servlet-class>cn.itcast.servlet.PathServlet</servlet-class>  
</servlet>  
<servlet-mapping>  
 <servlet-name>pathServlet</servlet-name>  
 *<!--缺省路径, 不用 / -->* <url-pattern>/</url-pattern>  
</servlet-mapping>



url-pattern标签中的路径可以按照上述的四种书写，它们的 优先级：

全路径匹配 > 路径通配符匹配 > 扩展名匹配 > 缺省路径

# 18. Servlet注解开发 :

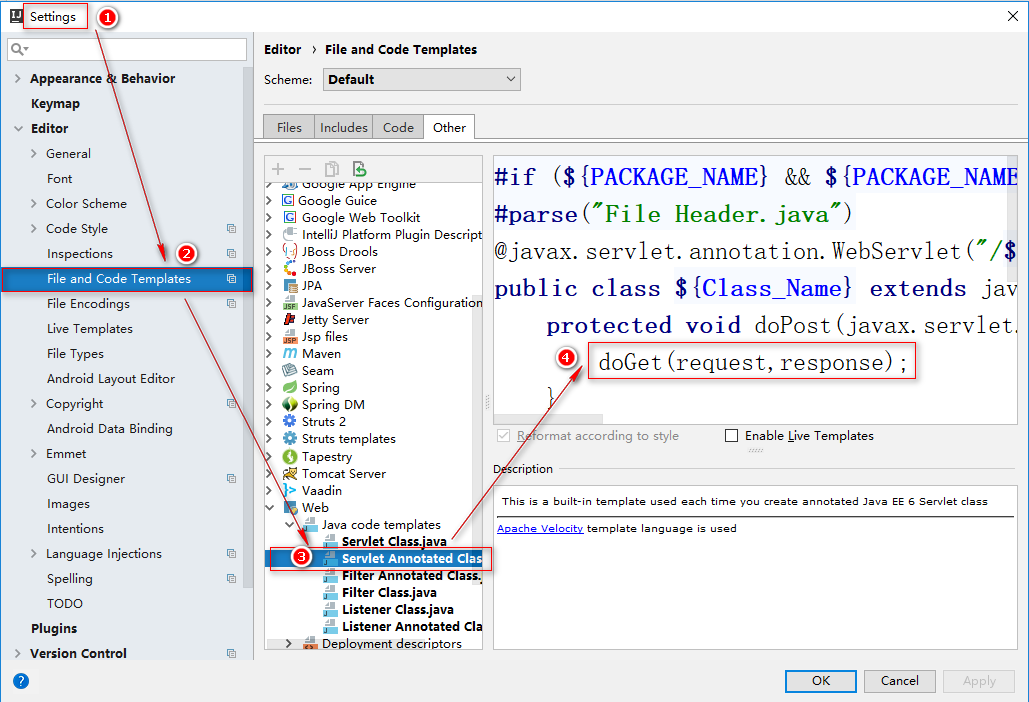


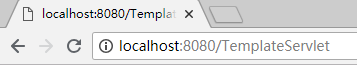
@WebServlet(name = "AnnotationServlet", urlPatterns = "/annotation")  
public class AnnotationServlet extends HttpServlet {  
 protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 System.*out*.println("doPost 方法执行 ...");  
 }  
  
 protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 System.*out*.println("doGet 方法执行 ...");  
 }  
}

# 19. 注解Servlet模板设置 :

模板配置代码 :

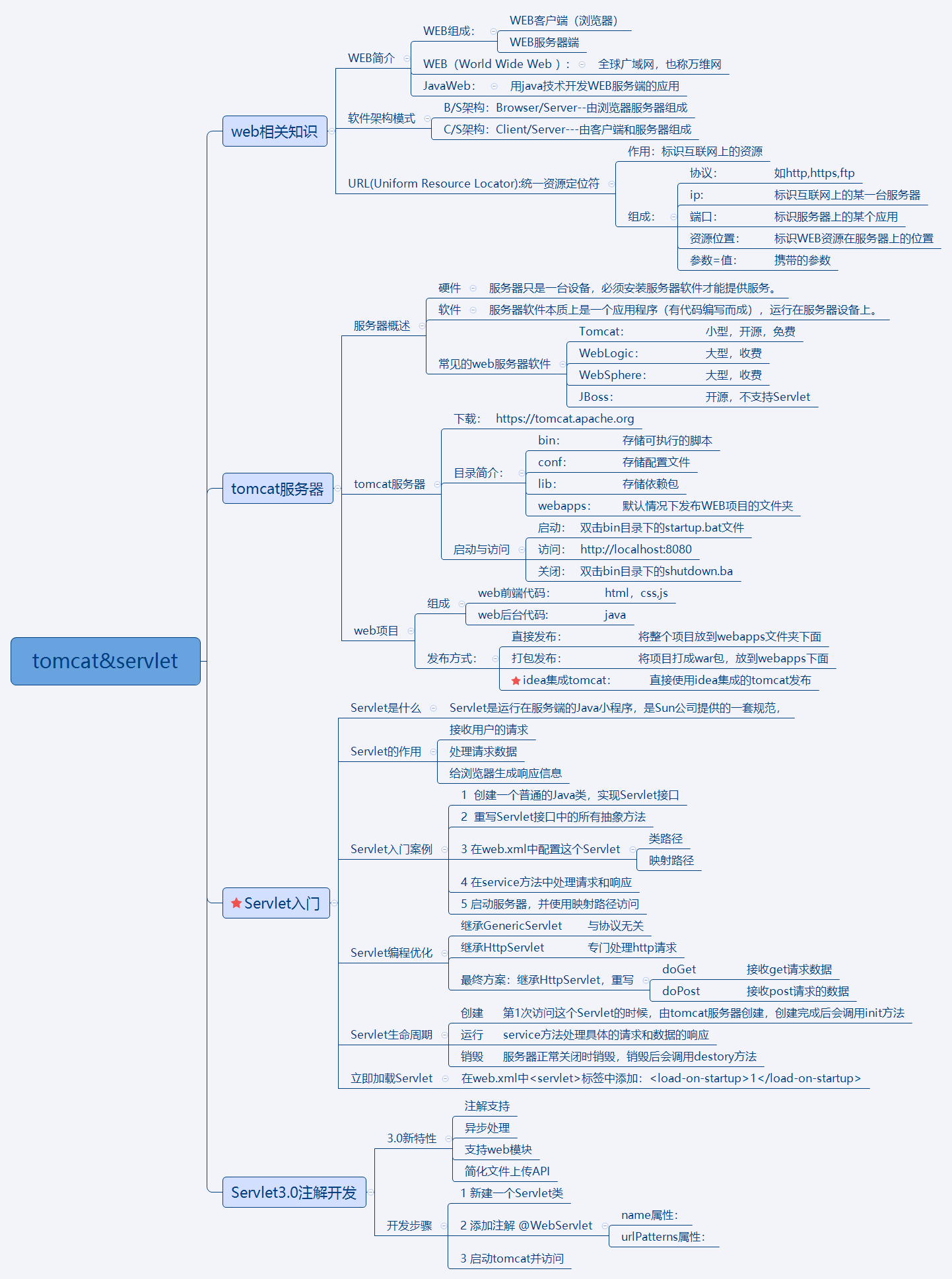
#if (${PACKAGE\_NAME} && ${PACKAGE\_NAME} != "")package ${PACKAGE\_NAME};#end  
#parse("File Header.java")  
@javax.servlet.annotation.WebServlet("/${Entity\_Name}")  
public class ${Class\_Name} extends javax.servlet.http.HttpServlet {  
 protected void doPost(javax.servlet.http.HttpServletRequest request, javax.servlet.http.HttpServletResponse response) throws javax.servlet.ServletException, java.io.IOException {  
 doGet(request,response);  
 }  
  
 protected void doGet(javax.servlet.http.HttpServletRequest request, javax.servlet.http.HttpServletResponse response) throws javax.servlet.ServletException, java.io.IOException {  
  
 }  
}





@WebServlet("/TemplateServlet")  
public class TemplateServlet extends HttpServlet {  
 protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 doGet(request, response);  
 }  
  
 protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 System.*out*.println("执行 ...");  
 }  
}

# 20. Servlet 小结说明 :



# 21. HTTP协议介绍 :

**目标 :** 了解HTTP协议的作用与特点.

**步骤 :**

一 : HTTP协议概念说明.

二 : 抓包查看HTTP协议的 get / post 请求格式.

**实现 :**

协议：是通信双方应该遵守的一个共同的规则。只有双方按照指定的规则，一方发送数据，另外一方才能接收到数据。

基础班学习网络编程的时候学习过TCP传输层的协议。

就业班学习的协议基本都是应用层的协议：主要是针对的是应用程序。

在互联网中，我们通过浏览器访问其他的网站，默认都会使用http协议。客户端和服务器之间进行交互的规则。

**http协议：**

超文本传输协议 (HTTP, HyperText transfer protocol) 是互联网上应用最为广泛的一种网络协议.

设计HTTP最初的目的是为了提供一种发布和接收HTML页面的方法, 它是互联网交互的一种规则.

HTTP 的特点 :

1. 默认端口是 80 (如果端口是80, 可以省略不写)

<http://www.baidu.com:80>

2. http协议永远是客户端 (浏览器或者手机等) 发送请求, 服务器回送响应.



客户端访问服务端称为客户端的请求，服务器给浏览器回送数据称为服务器的响应。

3. http协议是基于TCP协议.

**抓包查看协议的格式 :**

步骤一 : 安装抓包软件.



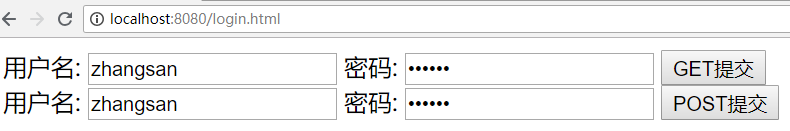
步骤二 : 准备表单页面, 用于发送 get / post 请求.

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <title>登录</title>  
</head>  
<body>  
 <form action="/request" method="get">  
 用户名: <input type="text" name="username" value="zhangsan">  
 密码: <input type="password" name="password" value="123456">  
 <input type="submit" value="GET提交">  
 </form>  
  
 <form action="/request" method="post">  
 用户名: <input type="text" name="username" value="zhangsan">  
 密码: <input type="password" name="password" value="123456">  
 <input type="submit" value="POST提交">  
 </form>  
</body>  
</html>

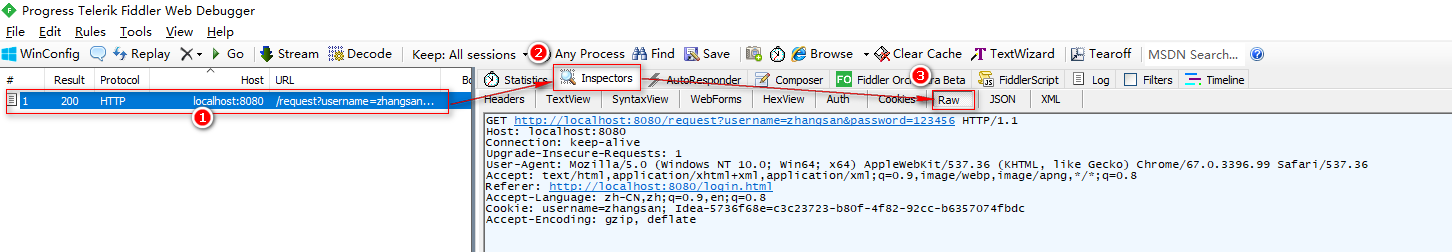
步骤三 : 准备提交的 Servlet.

@WebServlet("/request")  
public class RequestServlet extends HttpServlet {  
 protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 doGet(request, response);  
 }  
  
 protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 System.*out*.println("请求提交成功 ...");  
 }  
}

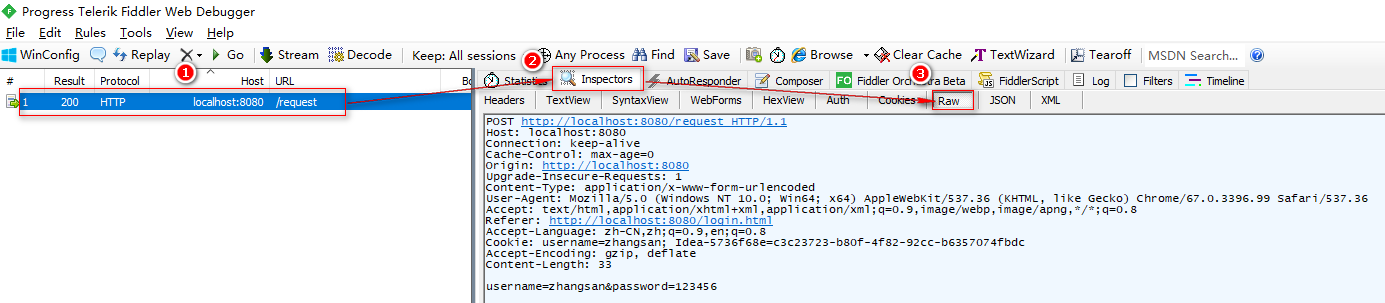
步骤四 : 启动 tomcat 服务器, 访问 login.html 登录页面, 并发送 get / post 请求.



步骤五 : get 请求提交的数据格式.



步骤六 : post 请求提交的数据格式.



**小结 :**

协议就是双方规定的交互方式, 双方都需要同时遵守, 这样才能实现数据的交互. HTTP协议规定了浏览器与服务器的数据交互规则.

# 22. 区分get请求与post请求 :

**目标 :** 能够区分 get与post 请求的区别. (浏览器发送给服务器的数据格式)

**步骤 :**

一 : get请求具体数据格式说明.

二 : post请求具体数据格式说明.

**实现 :**





**小结 :**

在请求中, 只有post请求方式才有请求体, get方式的请求, 请求数据中是没有请求体的.

# 23. 请求对象api介绍与演示 :

**目标 :** 请求对象中关于请求行, 请求头, 请求体的api介绍与演示.

**步骤 :**

一 : 图解api方法名与含义.

二 : 创建form表单, 实现post请求方式提交.

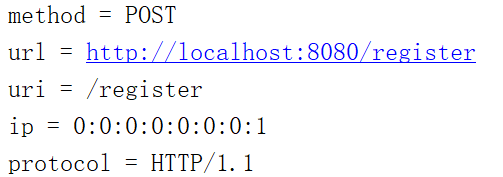
三 : 创建servlet, 调用请求对象api方法, 查看提交的具体数据.

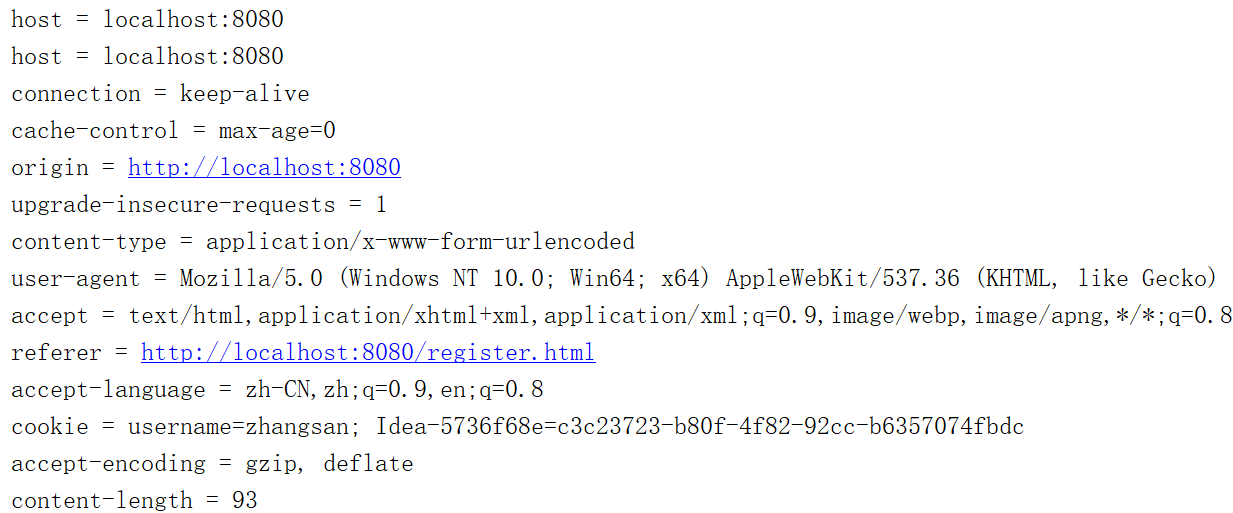
**实现 :**

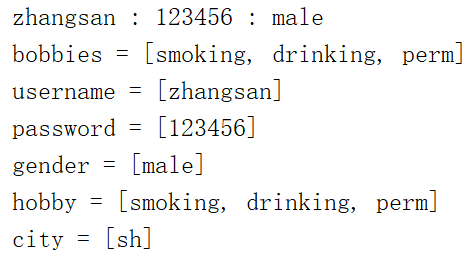


<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <title>注册</title>  
  
 <style>  
 #formDiv {  
 width: 300px;  
 margin: 0 auto;  
 }  
 </style>  
  
</head>  
<body>  
 <div id="formDiv">  
 <h3>注册表单</h3>  
 <form action="/register" method="post">  
 <table>  
 *<!--一对一数据-->* <tr>  
 <td><label for="username">用户名</label></td>  
 <td><input type="text" name="username" id="username"></td>  
 </tr>  
 <tr>  
 <td><label for="password">密码</label></td>  
 <td><input type="password" name="password" id="password"></td>  
 </tr>  
 <tr>  
 <td><label for="gender">性别</label></td>  
 <td>  
 <input type="radio" name="gender" id="gender" checked="checked" value="male"> 男  
 <input type="radio" name="gender" value="female"> 女  
 </td>  
 </tr>  
 *<!--一对多数据-->* <tr>  
 <td><label for="hobby">爱好</label></td>  
 <td>  
 <input type="checkbox" name="hobby" id="hobby" value="smoking">吸烟  
 <input type="checkbox" name="hobby" value="drinking">喝酒  
 <input type="checkbox" name="hobby" value="perm">烫头  
 </td>  
 </tr>  
 <tr>  
 <td><label for="city">城市</label></td>  
 <td>  
 <select name="city" id="city">  
 <option value="bj">北京</option>  
 <option value="sh" selected="selected">上海</option>  
 <option value="gz">广州</option>  
 <option value="sz">深圳</option>  
 </select>  
 </td>  
 </tr>  
 <tr>  
 <td></td>  
 <td><input type="submit" value="提交"></td>  
 </tr>  
 </table>  
 </form>  
 </div>  
</body>  
</html>

@WebServlet("/register")  
public class RegisterServlet extends HttpServlet {  
 protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 doGet(request, response);  
 }  
  
 protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 *// 1. 请求行  
 // 请求方式 :* String method = request.getMethod();  
 System.*out*.println("method = " + method);  
 *// 请求url* StringBuffer url = request.getRequestURL();  
 System.*out*.println("url = " + url);  
 *// 请求资源路径* String uri = request.getRequestURI();  
 System.*out*.println("uri = " + uri);  
 *// 请求ip地址* String ip = request.getRemoteAddr();  
 System.*out*.println("ip = " + ip);  
 *// 请求协议版本号* String protocol = request.getProtocol();  
 System.*out*.println("protocol = " + protocol);  
  
 *// 2. 请求头* String host = request.getHeader("Host");  
 System.*out*.println("host = " + host);  
 Enumeration<String> names = request.getHeaderNames();  
 while (names.hasMoreElements()) {  
 String name = names.nextElement();  
 String value = request.getHeader(name);  
 System.*out*.println(name + " = " + value);  
 }  
  
 *// 3. 请求体  
 // 一对一数据获取* String username = request.getParameter("username");  
 String password = request.getParameter("password");  
 String gender = request.getParameter("gender");  
 System.*out*.println(username + " : " + password + " : " + gender);  
 *// 一对多数据获取* String[] hobbies = request.getParameterValues("hobby");  
 System.*out*.println("bobbies = " + Arrays.*toString*(hobbies));  
 *// 获取表单中提交的所有数据* Map<String, String[]> map = request.getParameterMap();  
 for (Map.Entry<String, String[]> entry : map.entrySet()) {  
 String key = entry.getKey();  
 String[] value = entry.getValue();  
 System.*out*.println(key + " = " + Arrays.*toString*(value));  
 }  
 }  
}







**小结 :**

请求对象的api也可分为 `请求行, 请求头, 请求体` 三部分实现记忆.

# 24. 请求参数中文乱码处理 :

**目标 :** 解决请求参数中的中文乱码问题.

**步骤 :**

一 : 在页面中输出中文数据.

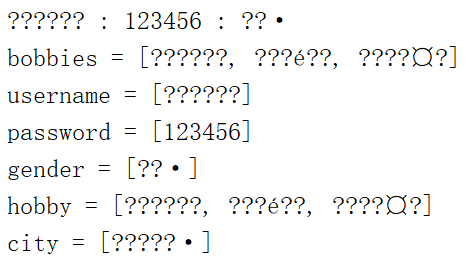
二 : 提交数据.

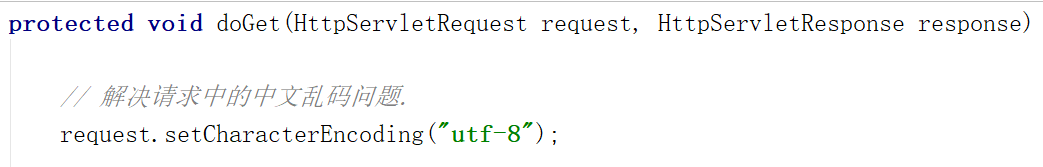
三 : 在对应的 servlet 中使用请求方法, 处理中文乱码.

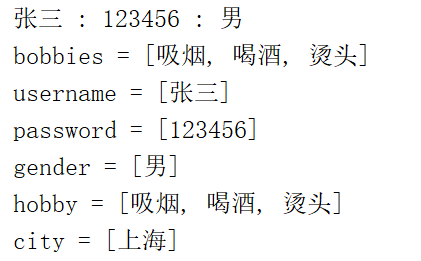
**实现 :**

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <title>注册</title>  
  
 <style>  
 #formDiv {  
 width: 300px;  
 margin: 0 auto;  
 }  
 </style>  
  
</head>  
<body>  
 <div id="formDiv">  
 <h3>注册表单</h3>  
 <form action="/register" method="post">  
 <table>  
 *<!--一对一数据-->* <tr>  
 <td><label for="username">用户名</label></td>  
 <td><input type="text" name="username" id="username"></td>  
 </tr>  
 <tr>  
 <td><label for="password">密码</label></td>  
 <td><input type="password" name="password" id="password"></td>  
 </tr>  
 <tr>  
 <td><label for="gender">性别</label></td>  
 <td>  
 <input type="radio" name="gender" id="gender" checked="checked" value="男"> 男  
 <input type="radio" name="gender" value="女"> 女  
 </td>  
 </tr>  
 *<!--一对多数据-->* <tr>  
 <td><label for="hobby">爱好</label></td>  
 <td>  
 <input type="checkbox" name="hobby" id="hobby" value="吸烟">吸烟  
 <input type="checkbox" name="hobby" value="喝酒">喝酒  
 <input type="checkbox" name="hobby" value="烫头">烫头  
 </td>  
 </tr>  
 <tr>  
 <td><label for="city">城市</label></td>  
 <td>  
 <select name="city" id="city">  
 <option value="北京">北京</option>  
 <option value="上海" selected="selected">上海</option>  
 <option value="广州">广州</option>  
 <option value="深圳">深圳</option>  
 </select>  
 </td>  
 </tr>  
 <tr>  
 <td></td>  
 <td><input type="submit" value="提交"></td>  
 </tr>  
 </table>  
 </form>  
 </div>  
</body>  
</html>









**小结 :**

如果请求提交中有中文数据, 需要使用 request.setCharacterEncoding(“utf-8”); 方法, 告诉服务器使用utf-8编码格式处理请求中的中文数据.

# 25. 请求和响应对象的生命周期 :

**目标 :** 了解请求和响应对象的创建与销毁的生命周期过程.

**步骤 :**

一 : 请求与响应对象的概念说明.

二 : 图解浏览器访问 servlet 的交互过程.

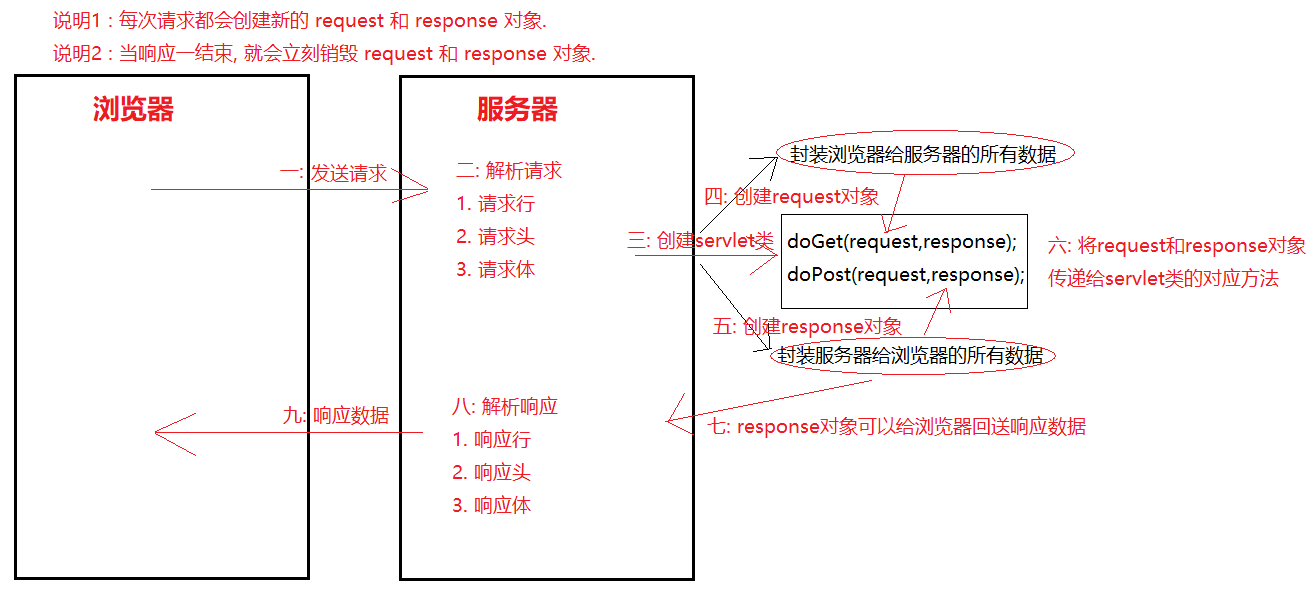
**实现 :**

servlet的最主要的作用就是处理客户的请求，并向客户端做出响应。

为此，针对servlet的每次请求，tomcat在调用doGet或者doPost方法之前，都会创建两个两个对象，分别是HttpServletRequest和HttpServletResponse对象。

其中，HttpServletRequest用于封装HTTP请求消息，简称request对象。

HttpServletResponse用于封装HTTP的响应消息，简称response对象。



**小结 :**

请求与响应对象是由 tomcat 服务器为我们程序创建的, 需要注意的是, 每一次新的请求, 都会创建全新的 request 和 response 对象.

# 26. request域对象与请求转发 :

**目标 :** 使用请求转发实现request 域对象的数据存储与获取.

域对象共有四个, 分别为 page, request, session, context.

当一个Web资源收到客户端的请求后，如果希望服务器通知另外一个资源处理.

可以通过 转发对象 RequestDispatcher的forward(request,response)方法,将当前请求传递给其他的Web资源进行处理，这种方式称为请求转发。

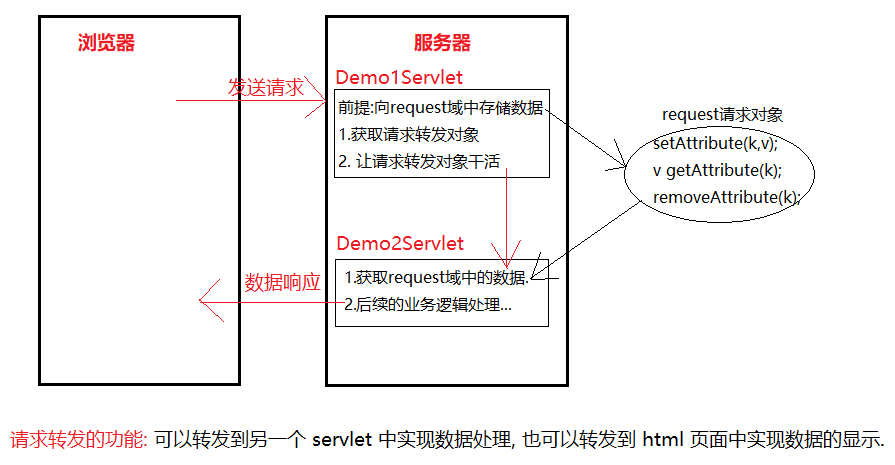
**步骤 :**

一 : 讲解 request 域对象的含义.

二 : 域对象提供的 api.

三 : 请求转发完成向request域对象中存储与获取数据实现.

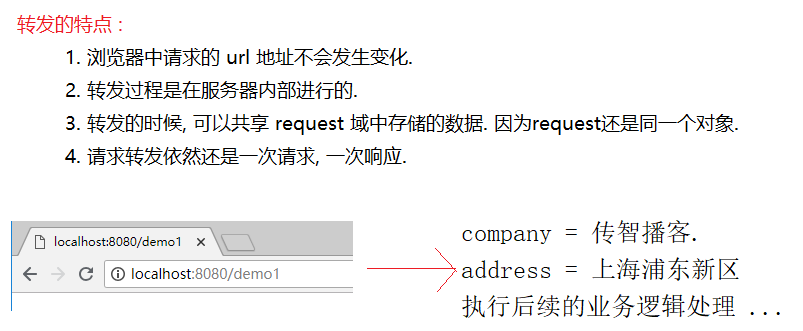
**实现 :**



@WebServlet("/demo1")  
public class Demo1Servlet extends HttpServlet {  
 protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 doGet(request, response);  
 }  
  
 protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 *// 1. 向 request 域中存储数据* request.setAttribute("company", "传智播客.");  
 request.setAttribute("address", "上海浦东新区");  
  
 *// 2. 获取请求转发对象* RequestDispatcher dispatcher = request.getRequestDispatcher("/demo2");  
  
 *// 3. 让请求转发对象干活* dispatcher.forward(request, response);  
 }  
}

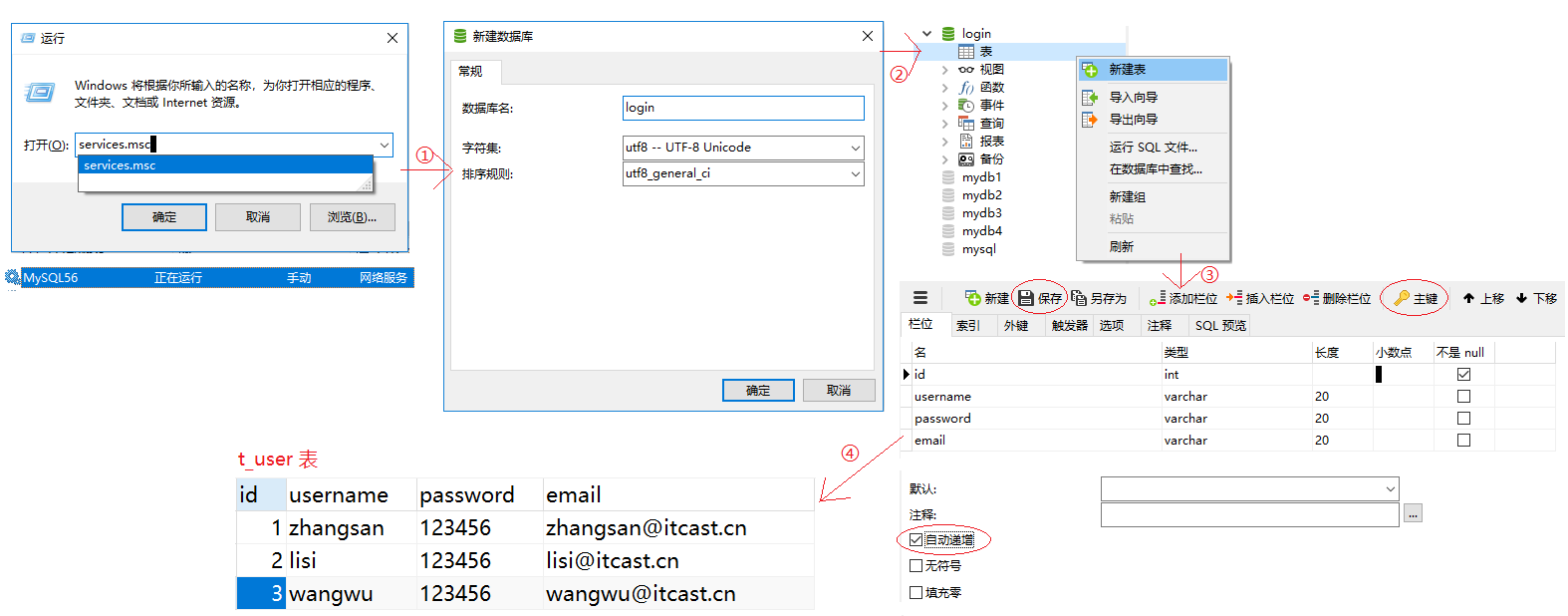
@WebServlet("/demo2")  
public class Demo2Servlet extends HttpServlet {  
 protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 doGet(request, response);  
 }  
  
 protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 *// 1. 获取 request 域中的数据* Object company = request.getAttribute("company");  
 Object address = request.getAttribute("address");  
 System.*out*.println("company = " + company);  
 System.*out*.println("address = " + address);  
  
 *// 2. 后续业务逻辑处理* System.*out*.println("执行后续的业务逻辑处理 ...");  
 }  
}

**小结 :**

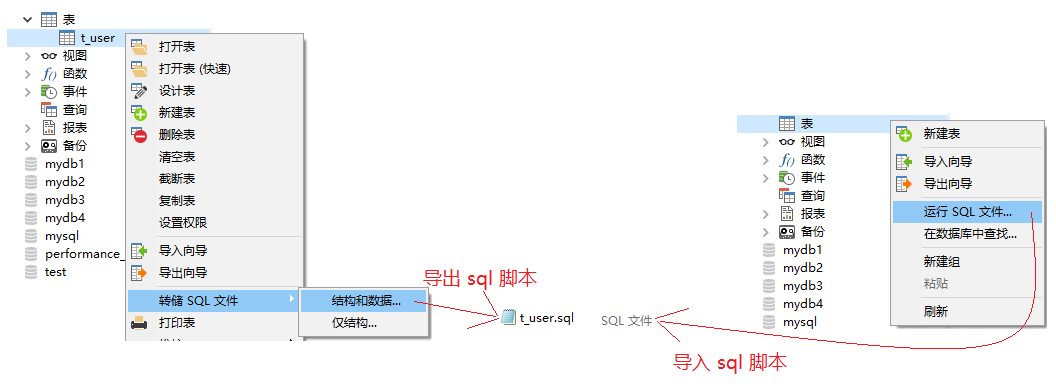


# 27. 登录案例前期准备工作 :

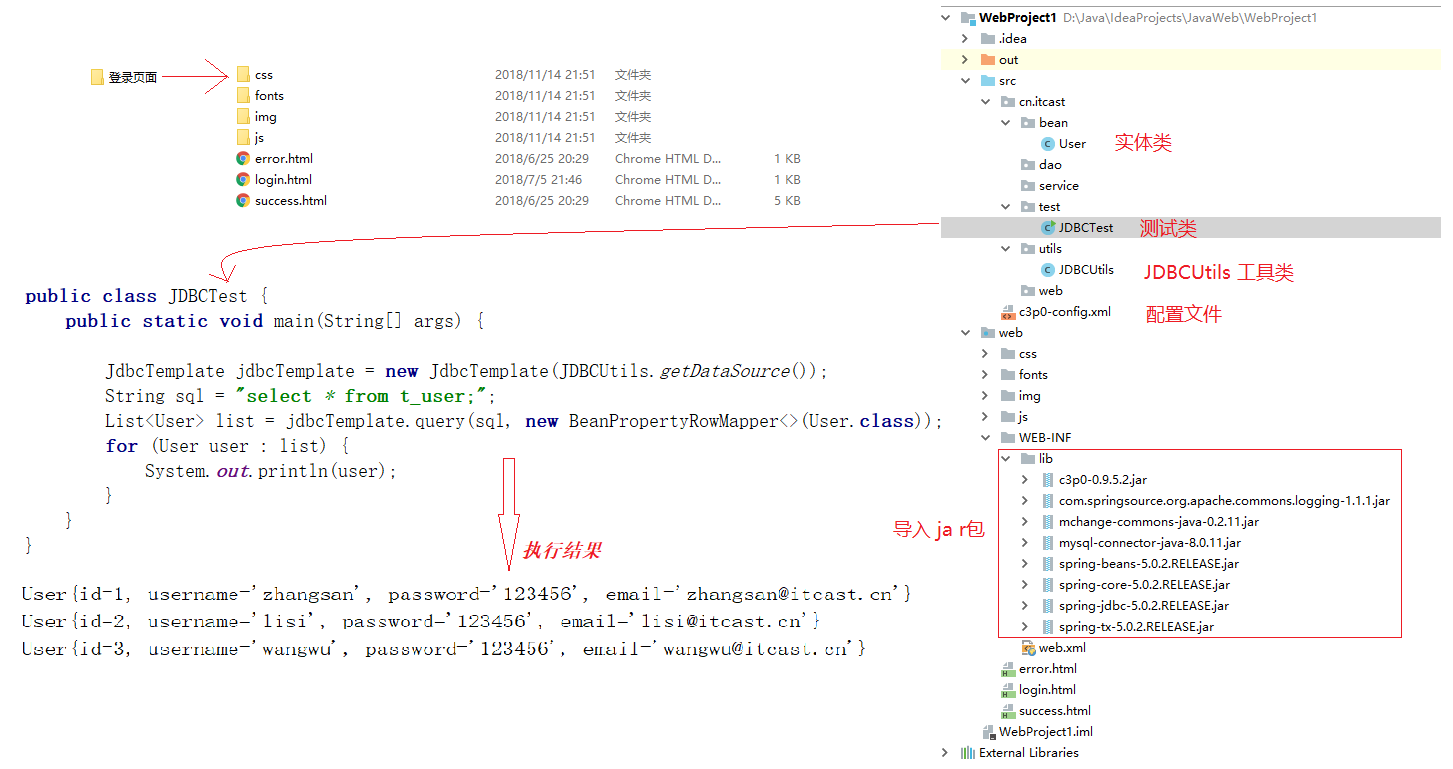
步骤一 : 数据库与数据表的准备.



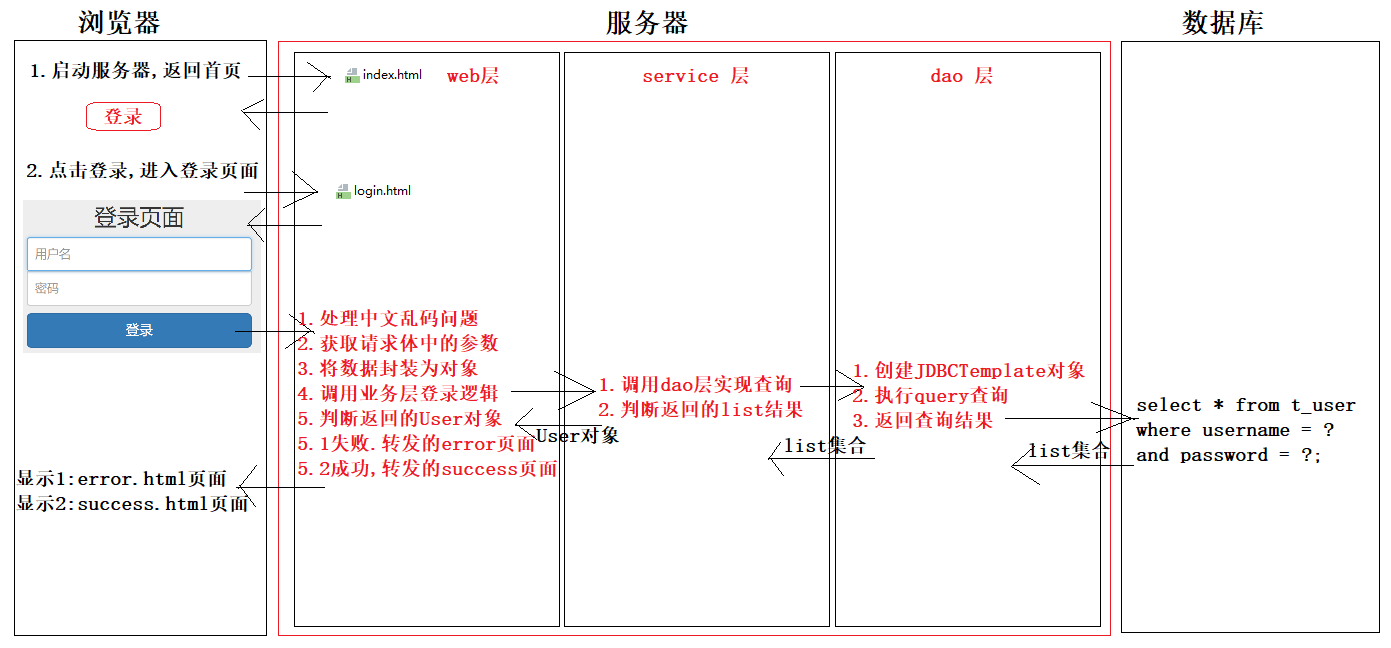
**说明 :** 如果数据脚本已经存在, 则可实现直接导入数据.



第二步 : IDEA 项目搭建, 测试类代码执行.



# 28. 登录案例业务流程分析 :



# 29. 登录案例具体代码实现 :

**login.html 登录页面 :**

<div class="container text-center">  
 <form class="form-signin" action="/login" method="post">  
 <h2 class="form-signin-heading">登录页面</h2>  
 <input type="text" name="username" class="form-control" placeholder="用户名" required autofocus>  
 <input type="password" name="password" class="form-control" placeholder="密码" required>  
 <button class="btn btn-lg btn-primary btn-block" type="submit">登录</button>  
 </form>  
</div>

import cn.itcast.bean.User;  
import cn.itcast.service.UserService;  
  
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.io.IOException;  
  
@WebServlet("/login")  
public class LoginServlet extends HttpServlet {  
 protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 doGet(request, response);  
 }  
  
 protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 *// 0. 处理中文乱码问题* request.setCharacterEncoding("utf-8");  
  
 *// 1. 获取用户名和密码* String username = request.getParameter("username");  
 String password = request.getParameter("password");  
  
 *// 2. 将数据封装为一个 User 对象* User user = new User();  
 user.setUsername(username);  
 user.setPassword(password);  
  
 *// 3. 调用业务层进行登录业务逻辑处理* UserService userService = new UserService();  
 User loginUser = userService.login(user);  
  
 *// 4. 判断返回的结果* if (loginUser == null) {  
 *// 登录失败. 请求转发到 `error.html` 页面* request.getRequestDispatcher("/error.html").forward(request, response);  
 } else {  
 *// 登录成功. 请求转发到 `success.html` 页面.* System.*out*.println(loginUser.getEmail());  
 request.getRequestDispatcher("/success.html").forward(request, response);  
 }  
 }  
}

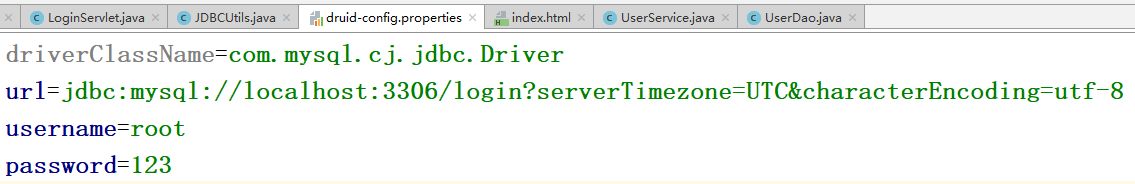
import cn.itcast.bean.User;  
import cn.itcast.dao.UserDao;  
  
import java.util.List;  
  
public class UserService {  
  
 *// 登录业务逻辑* public User login(User user) {  
 *// 1. 调用 dao 层, 实现根据用户名和密码查询用户是否存在* UserDao userDao = new UserDao();  
 List<User> list = userDao.login(user);  
 *// 2. 判断返回的结果* if (list == null || list.isEmpty() == true) {  
 return null;  
 } else {  
 return list.get(0);  
 }  
 }  
}

import cn.itcast.bean.User;  
import cn.itcast.utils.JDBCUtils;  
import org.springframework.jdbc.core.BeanPropertyRowMapper;  
import org.springframework.jdbc.core.JdbcTemplate;  
  
import java.util.List;  
  
public class UserDao {  
  
 *// 登录查询* public List<User> login(User user) {  
 *// 1. 创建一个 JDBCTemplate 对象, 并将数据库连接池作为参数传入* JdbcTemplate jdbcTemplate = new JdbcTemplate(JDBCUtils.*getDataSource*());  
 *// 2. 执行 query 查询方法* String sql = "select *\** from t\_user where username = ? and password = ?;";  
 Object[] params = { user.getUsername(), user.getPassword() };  
 List<User> list = jdbcTemplate.query(sql, new BeanPropertyRowMapper<>(User.class), params);  
 *// 3. 返回 list 集合* return list;  
 }  
}

# 30. Tomcat中读取Properties文件:

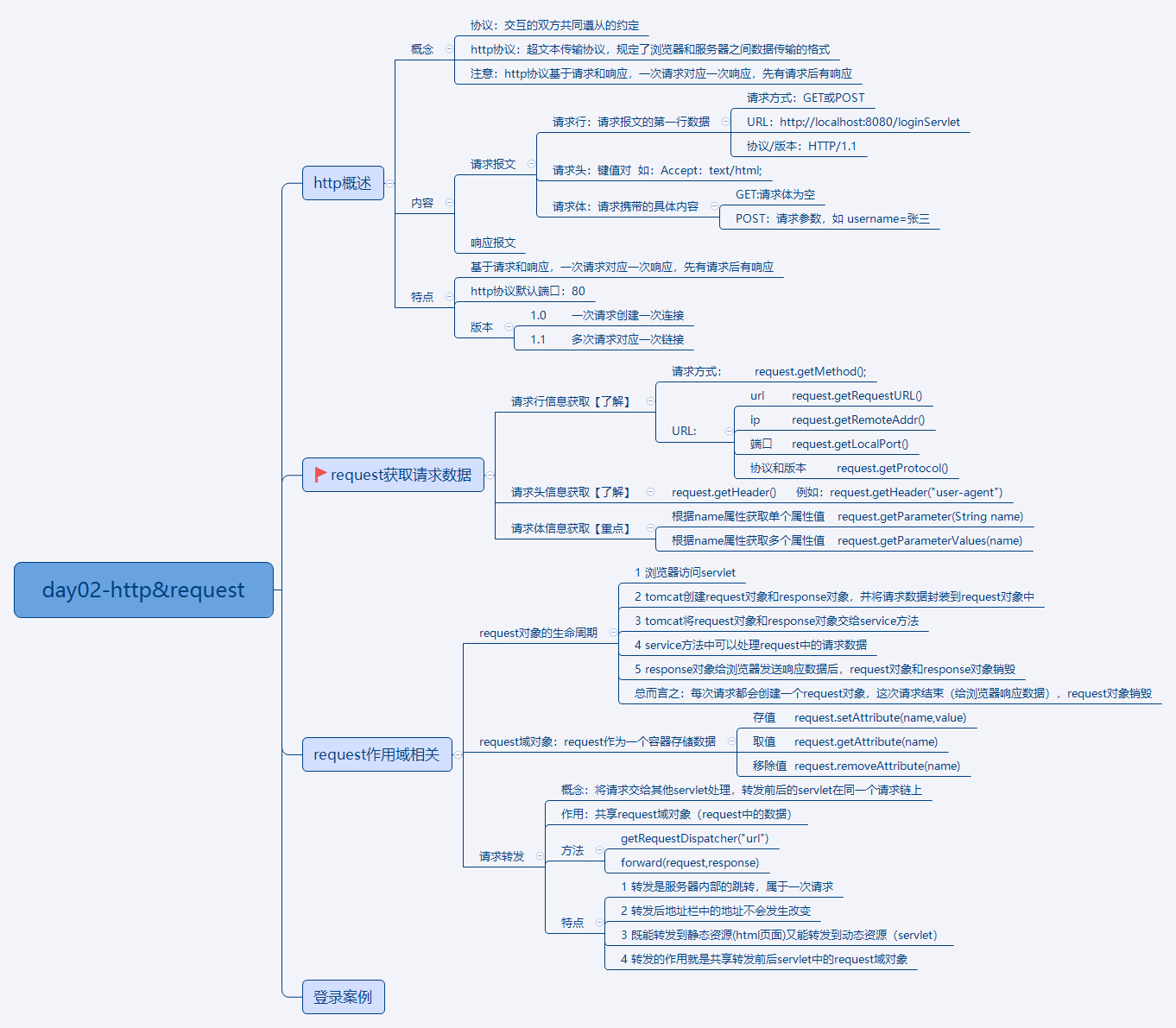
**注意1 :** properites 配置文件必须存放于 src 目录下.

**注意2 :** FileReader类无法读取发布后的配置文件, 需要改写为使用类加载器读取资源配置文件.



import com.alibaba.druid.pool.DruidDataSourceFactory;  
import javax.sql.DataSource;  
import java.io.InputStream;  
import java.sql.Connection;  
import java.sql.ResultSet;  
import java.sql.SQLException;  
import java.sql.Statement;  
import java.util.Properties;  
  
public class JDBCUtils {  
  
 *// 属性* private static final DataSource *dataSource*;  
  
 *// 静态代码块* static {  
 Properties prop = new Properties();  
 try {  
 *// 要求配置文件路径存放于 src 目录下, 并使用类加载器方法能够读取到src目录发布后的配置文件路径* InputStream in = JDBCUtils.class.getClassLoader().getResourceAsStream("druid-config.properties");  
 prop.load(in);  
  
 *dataSource* = DruidDataSourceFactory.*createDataSource*(prop);  
 } catch (Exception e) {  
 throw new RuntimeException("配置文件加载失败!");  
 }  
 }  
   
 *// 返回连接池对象 (JDBCTemplate 类需要在创建时使用)* public static DataSource getDataSource() {  
 return *dataSource*;  
 }  
  
 *// 建立连接* public static Connection getConnection() throws SQLException {  
 return *dataSource*.getConnection();  
 }  
  
 *// 释放资源* public static void release(Connection conn, Statement stmt, ResultSet rs) {  
  
 if (rs != null) {  
 try {  
 rs.close();  
 } catch (SQLException e) {  
 *// ignore* }  
 rs = null;  
 }  
  
 *release*(conn, stmt);  
 }  
  
 public static void release(Connection conn, Statement stmt) {  
  
 if (stmt != null) {  
 try {  
 stmt.close();  
 } catch (SQLException e) {  
 *// ignore* }  
 stmt = null;  
 }  
  
 if (conn != null) {  
 try {  
 conn.close(); *// 数据库连接池底层拦截 close 方法. 将连接对象重新放回到连接池中, 实现复用.* } catch (SQLException e) {  
 *// ignore* }  
 conn = null;  
 }  
 }  
}

# 31. request小结说明 :



# 32. response响应格式抓包 :

**目标 :** 查看响应的基本格式.

**步骤 :**

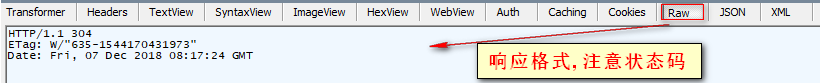
一 : 自定义reponse.html页面.

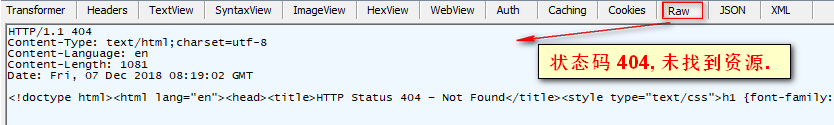
二 : 使用工具实现抓包, 查看.

**实现 :**

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <title>响应</title>  
</head>  
<body>  
 <form action="/response" method="get">  
 用户名: <input type="text" name="username" value="zhangsan">  
 密码: <input type="password" name="password" value="123456">  
 <input type="submit" value="GET提交">  
 </form>  
 <form action="/response" method="post">  
 用户名: <input type="text" name="username" value="zhangsan">  
 密码: <input type="password" name="password" value="123456">  
 <input type="submit" value="POST提交">  
 </form>  
</body>  
</html>







**小结 :**

响应指的是服务器返回给浏览器的数据. 也具有指定的格式.

# 33. response响应体介绍 :

response是专门给客户端响应数据的响应对象。

http协议规定：响应数据也要分成三部分：

响应行：主要学习设置状态码

响应头：主要学习重定向(重点)

响应体：主要是给客户端响应页面上要展示的数据（验证码）



# 34. response解决响应体显示乱码:

**目标 :** 使用响应对象将数据返回给浏览器, 完成显示.

**步骤 :**

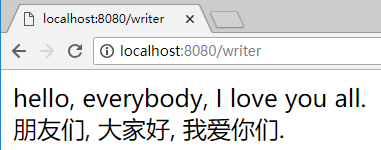
一 : 自定义 WriterServlet类.

二 : 使用response对象的getWriter() 方法将数据写入响应, 返回给浏览器实现.

三 : 使用response对象的 setContentType() 解决中文数据的乱码问题.

**实现 :**

@WebServlet("/writer")  
public class WriterServlet extends HttpServlet {  
 protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 doGet(request, response);  
 }  
  
 protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
  
 *// 设置响应类型与字符集.* response.setContentType("text/html;charset=utf-8");  
  
 *// 需求 : 使用 response 对象调用 getWriter 方法, 实现数据在浏览器中显示.* response.getWriter().println("hello, everybody, I love you all. <br />");  
 response.getWriter().println("朋友们, 大家好, 我爱你们.");  
 }  
}



**小结 :**

使用响应对象的 getWriter 方法可以向响应体中写入数据, 如果数据有中文, 需要使用 setContentType()方法设置响应数据的类型与字符集.

# 35. response实现页面跳转 :

**目标 :** 实现指定秒数后, 进行页面跳转.

应用场景: 经常网页中看到, 页面多少秒后的自动跳转.

例如: 你点击一个页面, 提示您访问的页面不存在, 5秒后返回首页. 通过设置refresh响应头信息来完成这个功能.

格式: response.setHeader(“refresh”, “秒数;网址”);

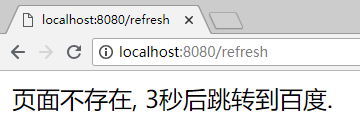
举例: response.setHeader(“refresh”,”5;http://www.baidu.com”)  表示5秒后跳转到 百度首页

**步骤 :**

一 : 自定义一个 RefreshServlet 类.

二 : 调用 response.setHeader 方法实现页面跳转.

**实现 :**



@WebServlet("/refresh")  
public class RefreshServlet extends HttpServlet {  
 protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 doGet(request, response);  
 }  
  
 protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 *// 说明: 向浏览器中输出中文数据* response.setContentType("text/html;charset=utf-8");  
 response.getWriter().println("页面不存在, 3秒后跳转到百度.");  
  
 *// 实现 3 秒后跳转到百度首页* response.setHeader("refresh", "3;url=https://www.baidu.com");  
 }  
}

**小结 :**

通过 response 对象, 调用 setHeader 方法, 指定 refresh的url, 可以实现页面的跳转功能.

# 36. response实现原始数据显示 :

**目标 :** 浏览器显示响应返回的原始数据.

**步骤 :**

一 : 自定义 ContentServlet 类.

二 : 使用response对象的setContentType 方法, 将类型设置为 text/plain.

**实现 :**

@WebServlet("/content")  
public class ContentServlet extends HttpServlet {  
 protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 doGet(request, response);  
 }  
  
 protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
  
 *// 说明: text/html 类型表示页面类型, 浏览器就会实现对响应体的解析.   
 // response.setContentType("text/html;charset=utf-8");  
 // 说明: text/plain 类型表示原始类型, 浏览器不会实现响应体数据的解析.* response.setContentType("text/plain;charset=utf-8");  
  
 response.getWriter().println("<h1>传智播客</h1>");  
 response.getWriter().println("<font color='red'>上海浦东新区</font>");  
 response.getWriter().println("欢迎您.");  
 }  
}





**小结 :**

通过设置 response 对象的 setContentType 方法, 将响应类型设置为 text/plain, 最终会导致浏览器不解析响应体数据.

# 37. response完成重定向 :

**目标 :** 使用response对象完成页面的重定向.

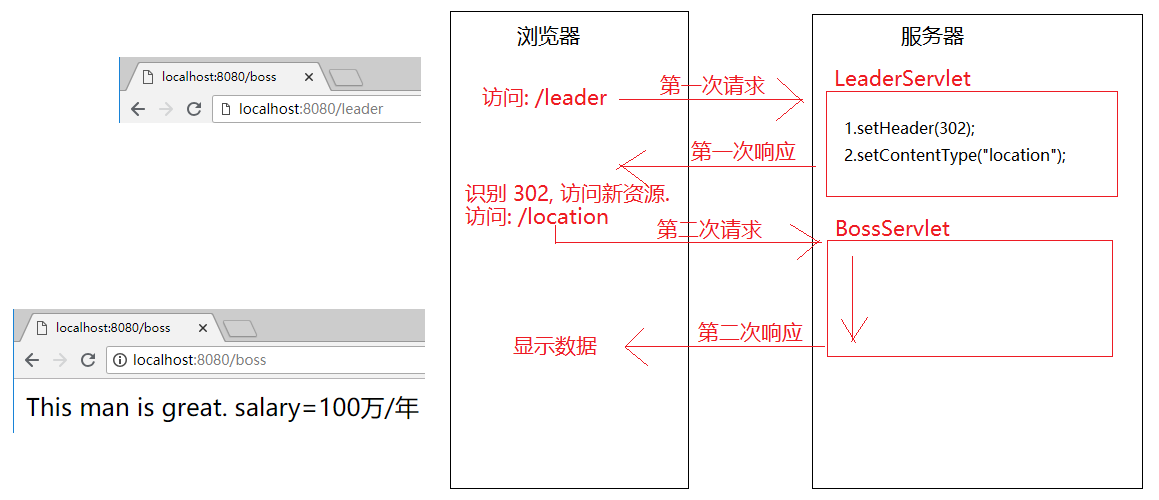
**步骤 :**

一 : 自定义LeaderServlet类.

二 : 访问 leader 时, 将请求状态码设置为 302, 并设置响应头 location 信息, 保存新资源路径.

三 : 自定义BossServlet类. 完成重定向.

**实现 :**



@WebServlet("/leader")  
public class LeaderServlet extends HttpServlet {  
 protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 doGet(request, response);  
 }  
  
 protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
  
 *// 处理逻辑 : 这哥们技术杠杠的, 要年薪 100 万. 要问一下老板同不同意.  
  
 // 1. 设置响应码* response.setStatus(302);  
 *// 2. 设置响应头 location 信息* response.setHeader("location", "/boss");  
 }  
}

@WebServlet("/boss")  
public class BossServlet extends HttpServlet {  
 protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 doGet(request, response);  
 }  
  
 protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 response.setContentType("text/html;charset=utf-8");  
 *// 老板处理的业务逻辑...* response.getWriter().println("This man is great. salary=100万/年");  
 }  
}

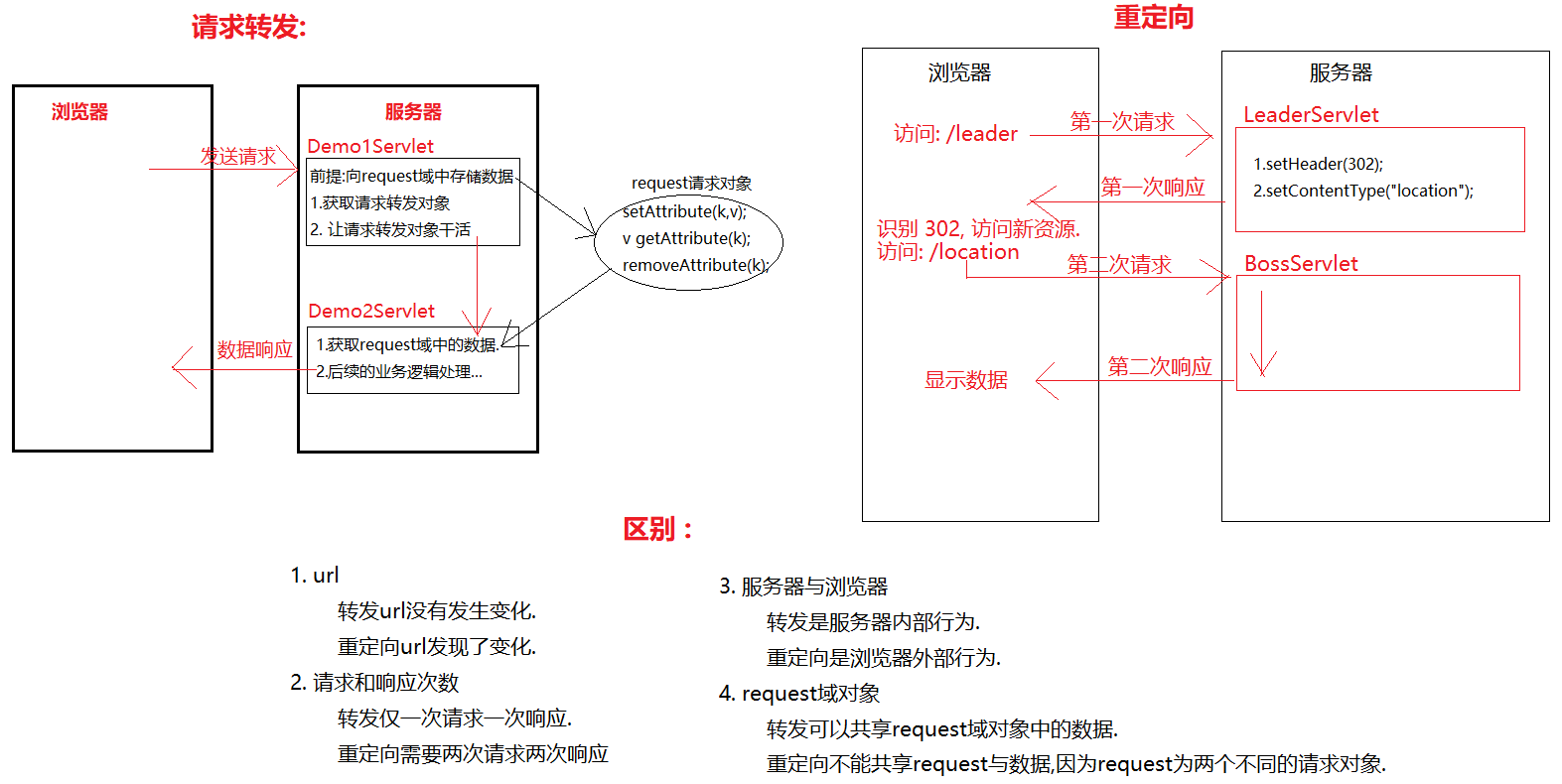
**小结 :**

首先必须是客户端发出请求, 然后服务端告诉客户端继续访问其他资源. 共两个步骤. 1. 设置响应码 2. 设置响应头的location新资源路径.

# 38. response重定向的简化写法 :

@WebServlet("/leader")  
public class LeaderServlet extends HttpServlet {  
 protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 doGet(request, response);  
 }  
  
 protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
  
 *// 处理逻辑 : 这哥们技术杠杠的, 要年薪 100 万. 要问一下老板同不同意.  
  
 // 实现 : 调用 response 对象的 sendRedirect 方法.* response.sendRedirect("/boss");  
 }  
}

# 39. 请求转发与重定向的区别 :



# 40. ServletContext上下文对象讲解 :



# 41. 上下文对象获取文件路径与文件类型 :

**目标 :** 使用上下文对象获取项目发布后的文件路径与文件类型.

**步骤 :**

一 : 自定义ContextServlet 类.

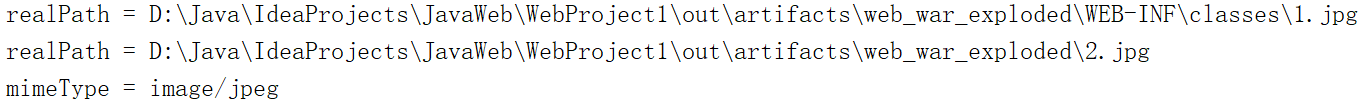
二 : 将资源拷贝到当前项目中. 注意资源的存放位置.

三 : 调用上下文对象的 getRealPath 方法获取资源的真实发布路径.

四 : 调用上下文对象的 getMimeType方法获取资源的类型.

**实现 :**

@WebServlet("/context")  
public class ContextServlet extends HttpServlet {  
 protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 doGet(request, response);  
 }  
  
 protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 *// 资源位置说明: 1.jpg 位于 src 目录中. 2.jpg 位于web文件夹中.  
 // 特殊注意: src 目录一旦发布后, 就会成为 WEB-INFO 目录中 classes 文件夹内的资源. 不能直接访问.  
  
 // 1. 获取文件发布后的真实路径* ServletContext context = this.getServletContext();  
 *// D:\Java\IdeaProjects\JavaWeb\WebProject1\out\artifacts\web\_war\_exploded\WEB-INF\classes\1.jpg* String realPath = context.getRealPath("WEB-INF\\classes\\1.jpg");  
 System.*out*.println("realPath = " + realPath);  
  
 *// D:\Java\IdeaProjects\JavaWeb\WebProject1\out\artifacts\web\_war\_exploded\2.jpg* realPath = context.getRealPath("2.jpg");  
 System.*out*.println("realPath = " + realPath);  
  
 *// 2. 获取文件类型* String mimeType = context.getMimeType("WEB-INF\\classes\\1.jpg");  
 System.*out*.println("mimeType = " + mimeType);  
 }  
}



**小结 :**

通过 servletContext 对象调用 realPath方法可以获取项目的真实发布路径. getMimeType可以获取资源在服务器中的类型格式.

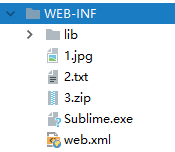
# 42. 下载案例思路讲解 :

**目标 :** 实现文件下载.

**步骤 :**

1. 文件下载资源存放到 WEB-INF 文件夹中, 这样用户就无法实现直接访问, 因此我们就可以使用代码来控制.
2. 编写 download.html 页面. 点击链接, 下载资源.
3. 编写 DownloadServlet 类.
4. 获取文件名称, 使用 ServletContext 对象的 getRealPath 获取文件的真实路径.
5. 根据文件名, 创建 file 对象, 判断该资源是否真实存在.
6. 如果存在, 首先需要设置两个响应头. (Content-Type, Content-Disposition) 告诉浏览器文件的类型, 并告诉浏览器不要解析文件,要以附件的形式下载.
   1. setHeader(“Content-Type”, context.getMimeType(filename)); 告诉浏览器文件的类型
   2. setHeader(“Content-Disposition”,, “attachment;filename=”+filename);
7. 获取字节输入输出流对象, 实现文件复制操作.

# 43. 下载案例代码实现 :



**download.html 页面 :**

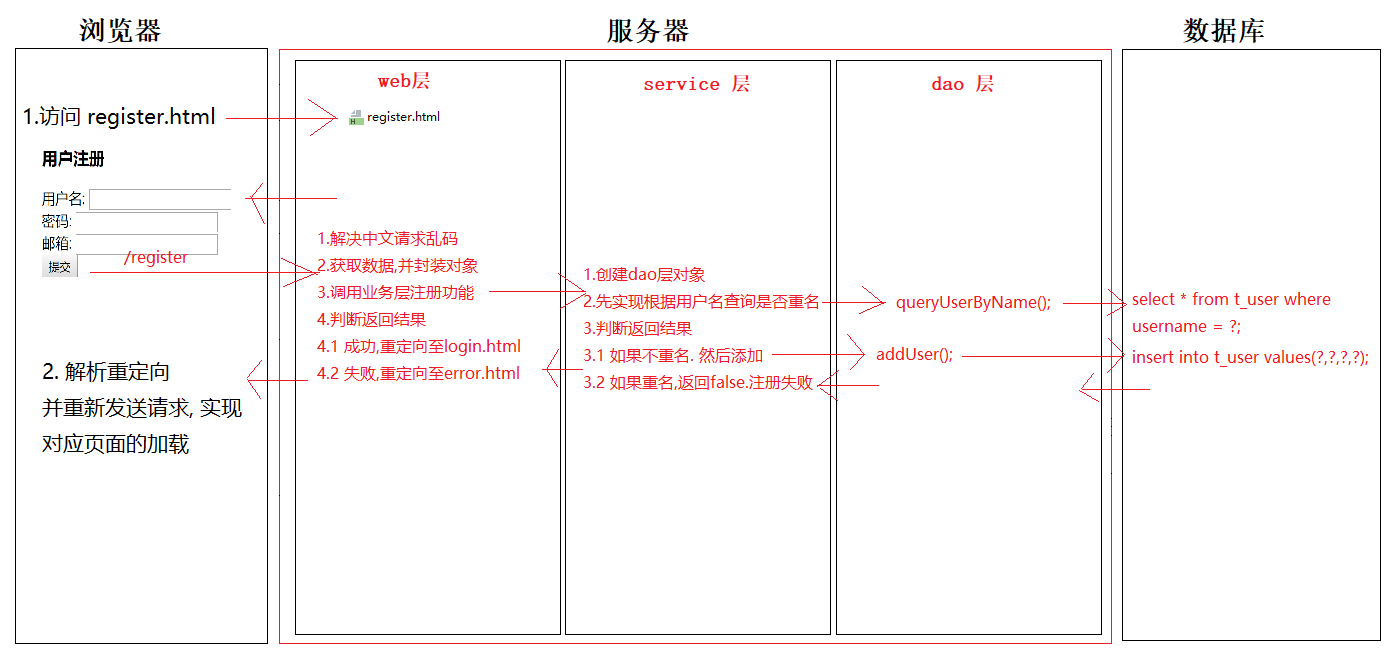
<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <title>文件下载</title>  
</head>  
<body>  
 <a href="1.jpg">美女图</a> <br>  
 <a href="/download?filename=1.jpg">1.jpg</a><br>  
 <a href="/download?filename=2.txt">2.txt</a><br>  
 <a href="/download?filename=3.zip">3.zip</a><br>  
 <a href="/download?filename=Sublime.exe">Sublime.exe</a>  
</body>  
</html>

@WebServlet("/download")  
public class DownloadServlet extends HttpServlet {  
 protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 doGet(request, response);  
 }  
  
 protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 *// 1. 获取文件名* String filename = request.getParameter("filename");  
 *// 2. 获取文件真实路径* ServletContext context = this.getServletContext();  
 String realPath = context.getRealPath("/WEB-INF/" + filename);  
 *// 3. 创建 file 对象* File file = new File(realPath);  
 *// 4. 判断文件是否存在* if (file.exists()) {  
 *// 5. 实现下载  
 // 5.1 获取文件类型* String mimeType = context.getMimeType(filename);  
 *// 5.2 设置内容类型* response.setContentType(mimeType);  
 *// 5.3 告诉浏览器, 以附件的形式实现下载* response.setHeader("Content-Disposition", "attachment;filename=" + filename);  
  
 *// 5.4 创建字节输入, 输出流* BufferedInputStream bis = new BufferedInputStream(new FileInputStream(file));  
 BufferedOutputStream bos = new BufferedOutputStream(response.getOutputStream());  
  
 *// 5.5 读写操作* byte[] buf = new byte[1024];  
 int len = -1;  
 while ((len = bis.read(buf)) != -1) {  
 bos.write(buf, 0, len);  
 bos.flush();  
 }  
  
 *// 5.6 关闭资源* bos.close();  
 bis.close();  
 }  
 }  
}

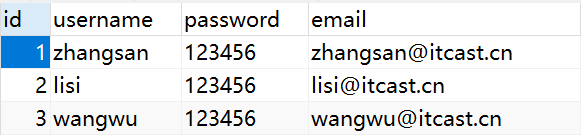
# 44. 注册案例实现 :

**目标 :** 用户注册, 返回注册成功或失败.

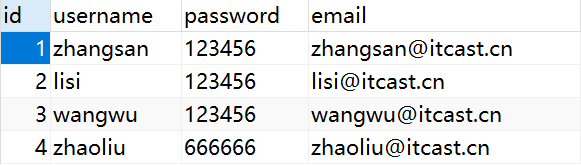
**步骤 :**

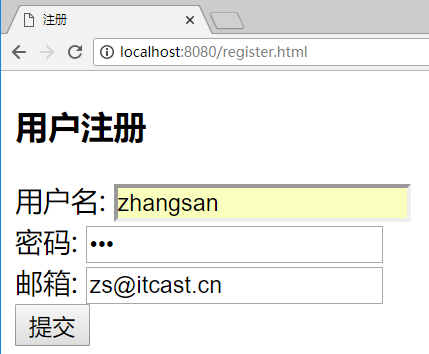


**实现 :**











**register.html**

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <title>注册</title>  
</head>  
<body>  
 <h3>用户注册</h3>  
 <form action="/register" method="post">  
 用户名: <input type="text" name="username"> <br>  
 密码: <input type="password" name="password"> <br>  
 邮箱: <input type="text" name="email"> <br>  
 <input type="submit" value="提交">  
 </form>  
</body>  
</html>

@WebServlet("/register")  
public class RegisterServlet extends HttpServlet {  
 protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 doGet(request, response);  
 }  
  
 protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 *// 1. 解决中文乱码问题* request.setCharacterEncoding("utf-8");  
 *// 2. 获取用户名, 密码, 邮箱* String username = request.getParameter("username");  
 String password = request.getParameter("password");  
 String email = request.getParameter("email");  
 *// 3. 将数据封装为对象* User user = new User();  
 user.setUsername(username);  
 user.setPassword(password);  
 user.setEmail(email);  
 *// 4. 调用业务层, 实现注册功能* UserService userService = new UserService();  
 boolean result = userService.register(user);  
 *// 5. 判断 result 结果* if (result == true) {  
 *// 5.1 注册成功, 去登录页面, 不需要携带数据, 重定向实现* response.sendRedirect("/login.html");  
 } else {  
 *// 5.2 注册失败, 重定向到 error.html 页面* response.sendRedirect("/error.html");}  
 }  
}

*// 用户注册*public boolean register(User user) {  
 *// 说明 : 注册业务要实现 `先查` `后插`  
 // 1. 创建一个 dao 层对象* UserDao userDao = new UserDao();  
 *// 2. 先实现查询* List<User> list = userDao.queryUserByName(user);  
 *// 3. 判断  
 // 3.1 list 结果为 null, 或者 list 集合中没有任何元素, list 集合为空* if (list == null || list.isEmpty() == true) {  
 *// 没有重名的元素, 可以添加新用户* int count = userDao.addUser(user);  
 *// 判断* if (count > 0) {  
 return true;  
 } else {  
 return false;  
 }  
 } else {  
 *// 有重名的用户. 添加失败.* return false;  
 }  
}

*// 根据用户名查询数据库中是否存在该用户*public List<User> queryUserByName(User user) {  
 *// 1. 创建一个 JDBCTemplate 对象, 并将数据库连接池作为参数传入* JdbcTemplate jdbcTemplate = new JdbcTemplate(JDBCUtils.*getDataSource*());  
 *// 2. 执行 query 查询* String sql = "select *\** from t\_user where username = ?;";  
 List<User> list = jdbcTemplate.query(sql, new BeanPropertyRowMapper<>(User.class), user.getUsername());  
 *// 3. 返回结果* return list;  
}  
  
*// 添加新用户*public int addUser(User user) {  
 *// 1. 创建一个 JDBCTemplate 对象, 并将数据库连接池作为参数传入* JdbcTemplate jdbcTemplate = new JdbcTemplate(JDBCUtils.*getDataSource*());  
 *// 2. 执行 udpate 方法* String sql = "insert into t\_user values(?,?,?,?);";  
 Object[] params = { null, user.getUsername(), user.getPassword(), user.getEmail() };  
 int count = jdbcTemplate.update(sql, params);  
 *// 3. 返回结果* return count;  
}

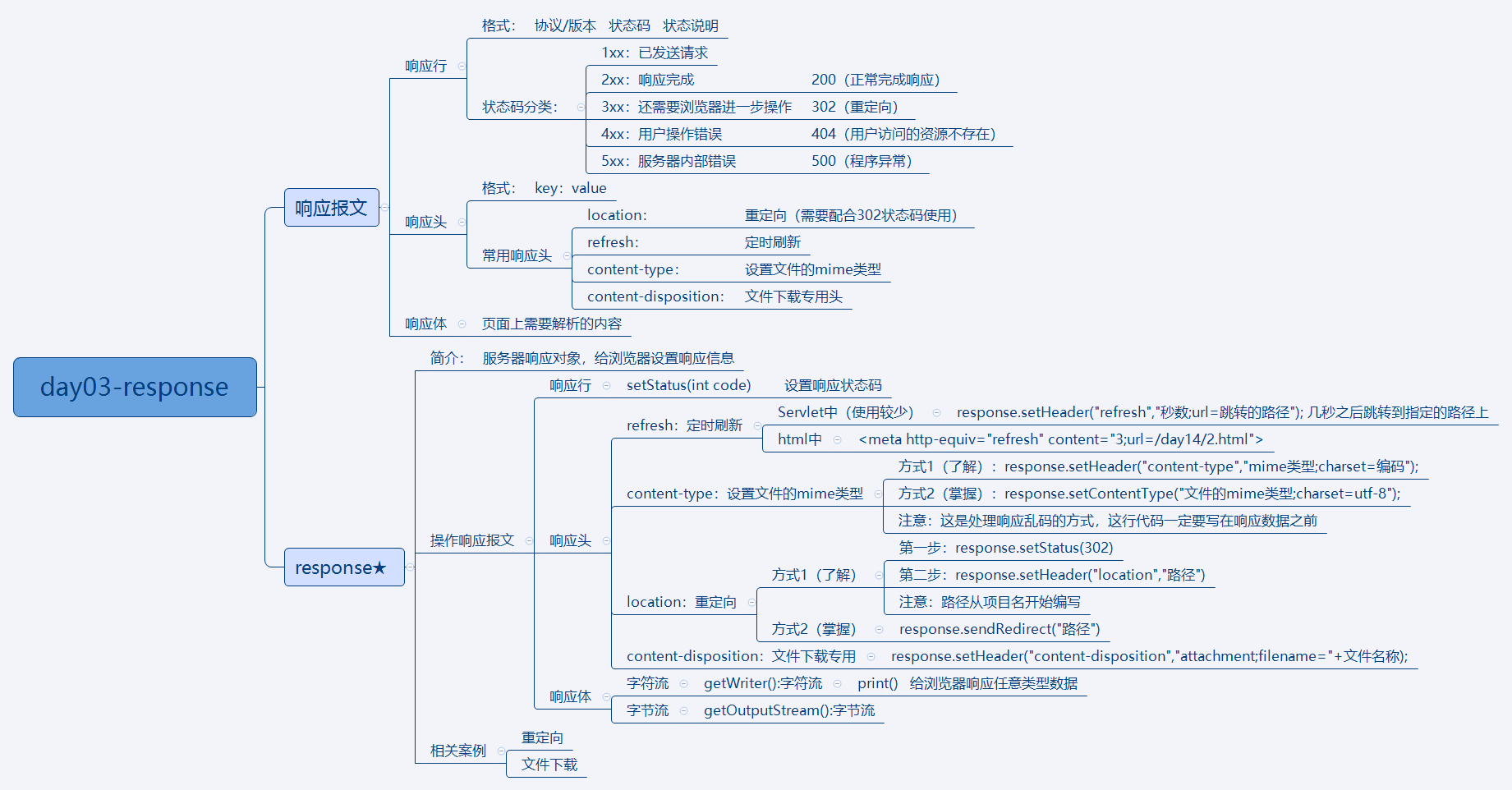
**error.html**

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <title>Title</title>  
</head>  
<body>  
 <h1>用户名或密码错误，请重新<a href="/login.html">登录</a>！</h1>  
 <h1>注册失败，请重新<a href="/register.html">注册</a>！</h1>  
</body>  
</html>

**小结 :**

用户注册一定要 `先查后插`. 如果用户名重复, 就不能添加用户.

# 45. response小结说明 :

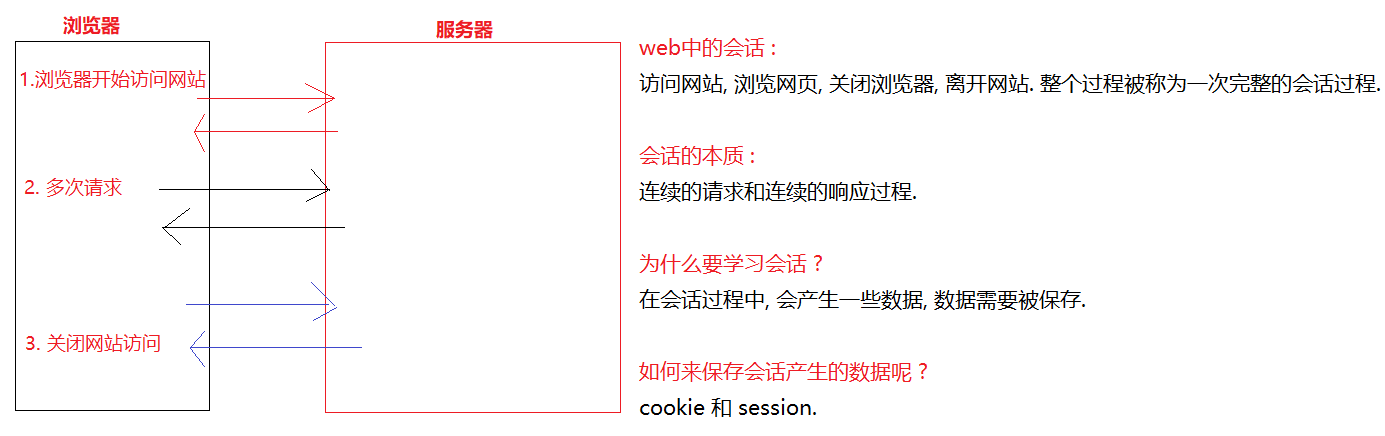


# 46. 会话与Cookie概述 :

**目标 :** 理解会话与cookie.

**步骤 :**

1. 什么是会话?



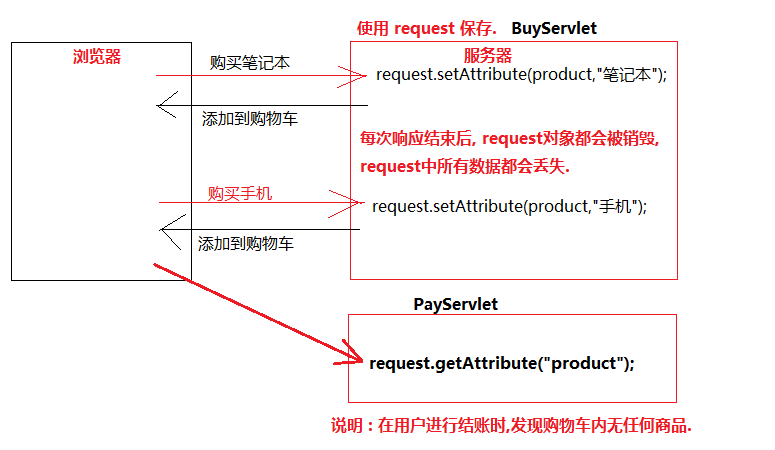
会话可以理解为, 用户打开浏览器, 从使用浏览器来实现服务器访问的那刻开始, 无论中间发生过多少次请求, 到最终关闭浏览器, 结束整个访问. 该整个过程就被称为一次完整的会话.

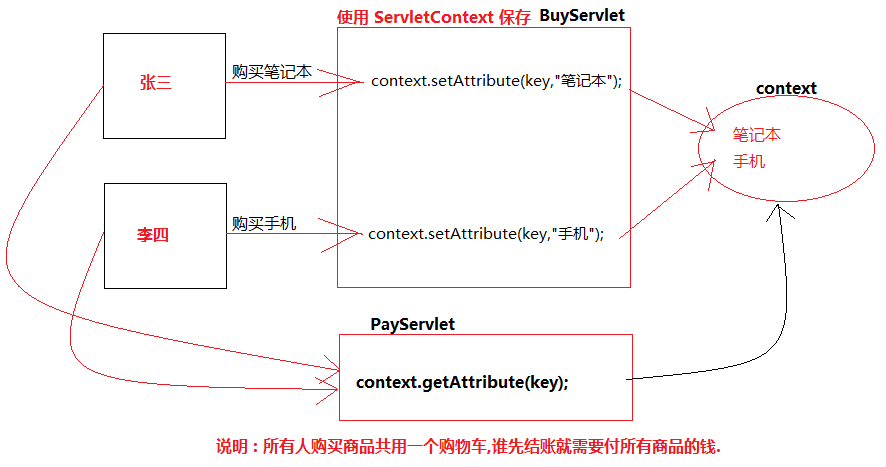
2. 会话过程中需要解决一些问题?

每个用户和服务器交互的会话过程中, 会产生一些数据,需要程序将数据保存起来.

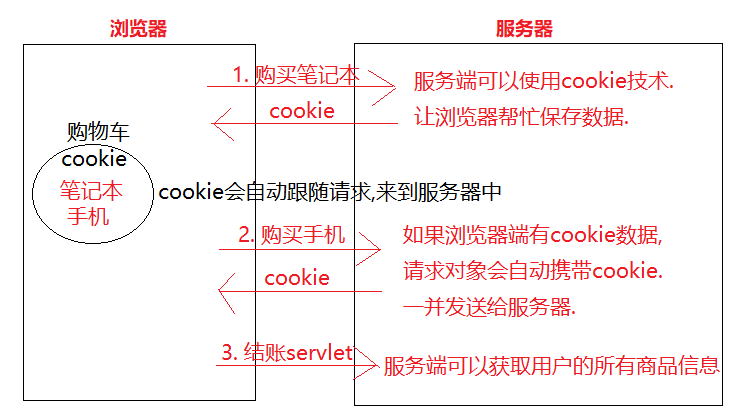
例如：用户点击超链接通过一个servlet购买了一个商品，程序应该保存用户购买的商品，以便于用户点结帐servlet时，结帐servlet可以得到用户商品为用户结帐。

思考：用户购买的商品保存在request或servletContext中行不行？





3. 什么是 cookie ?



Cookie就是服务器实现将数据存储与客户端浏览器的一门存储技术.

好处 : 服务器希望保存数据, 使用浏览器来帮忙保存数据. Cookie只能在服务端创建, 通过响应对象, 将cookie 数据保存在客户端浏览器本地. 这样可以减轻服务器保存数据的压力.

注意 : 如果用户没有登录该网站, 依然是可以实现将数据保存于客户端浏览器本地.

# 47. Cookie记住密码案例 :

**目标 :** 使用cookie技术在客户端浏览器中存储用户名和密码.

**步骤 :**

一 : 创建一个 login.html 登录页面.

二 : 编写 LoginServlet 类. 判断用户是否点击了记住密码.

三 : 编写 GetCookieServlet 类, 来获取浏览器请求时携带的 cookie 数据.



**实现 :**

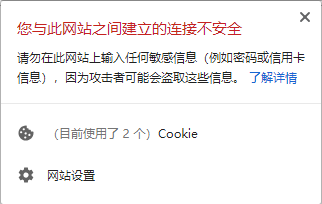
login.html 页面 :

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <title>登录</title>  
</head>  
<body>  
 <h3>用户登录</h3>  
 <hr>  
 <form action="/login" method="post">  
 <table>  
 <tr>  
 <td><label for="username">用户名:</label></td>  
 <td><input type="text" name="username" id="username"></td>  
 </tr>  
 <tr>  
 <td><label for="password">用户名:</label></td>  
 <td><input type="password" name="password" id="password"></td>  
 </tr>  
 <tr>  
 <td colspan="2"><input type="checkbox" name="check">记住密码</td>  
 </tr>  
 <tr>  
 <td></td>  
 <td><input type="submit" value="登录"></td>  
 </tr>  
 </table>  
 </form>  
</body>  
</html>

@WebServlet("/login")  
public class Login2Servlet extends HttpServlet {  
 protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 doGet(request, response);  
 }  
  
 protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 *// 1. 设置请求编码* request.setCharacterEncoding("utf-8");  
 *// 2. 获取用户名, 密码, 记住密码* String username = request.getParameter("username");  
 String password = request.getParameter("password");  
 String check = request.getParameter("check");  
 *// 3. 判断用户是否点击了 `记住密码`* if ("on".equals(check)) {  
 *// 3.1 创建用户名和密码的 cookie* Cookie usernameCookie = new Cookie("username", username);  
 Cookie passwordCookie = new Cookie("password", password);  
 *// 3.2 将创建好的 cookie 对象, 添加到 response 对象中, 响应给浏览器实现存储* response.addCookie(usernameCookie);  
 response.addCookie(passwordCookie);  
 }  
 }  
}

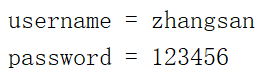
@WebServlet("/getCookie")  
public class GetCookieServlet extends HttpServlet {  
 protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 doGet(request, response);  
 }  
  
 protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 *// 1. 从请求中获取所有的 cookie 对象* Cookie[] cookies = request.getCookies();  
 *// System.out.println("cookie = " + cookies); 说明: 如果没有 cookie 对象, 则返回为 null.  
 // 2. 遍历 cookie 数组* if (cookies != null) {  
 for (Cookie cookie : cookies) {  
 *// 获取用户名 cookie 和密码 cookie* String name = cookie.getName();  
 String value = cookie.getValue();  
 System.*out*.println(name + " = " + value);  
 }  
 } else {  
 System.*out*.println("没有任何 cookie 数据.");  
 }  
 }  
}











**小结 :**

Cookie是服务器实现将数据存储在浏览器存储技术, 当浏览器本地有cookie数据时, 发送的请求中会自动携带cookie.服务器就可以根据浏览器的cookie取出对应的数据.

# 48. Cookie中特殊字符处理 :

**目标 :** cookie中的特殊字符处理.

**步骤 :**

一 : 在用户名文本框中输入特殊字符, 如: 空格.

二 : 使用 `编码与解码` 解决特殊字符.

**实现 :**



@WebServlet("/login")  
public class Login2Servlet extends HttpServlet {  
 protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 doGet(request, response);  
 }  
  
 protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 *// 1. 设置请求编码* request.setCharacterEncoding("utf-8");  
 *// 2. 获取用户名, 密码, 记住密码* String username = request.getParameter("username");  
 String password = request.getParameter("password");  
 String check = request.getParameter("check");  
 *// 3. 判断用户是否点击了 `记住密码`* if ("on".equals(check)) {  
  
 *// 实现数据编码* username = URLEncoder.*encode*(username, "utf-8");  
 password = URLEncoder.*encode*(password, "utf-8");  
  
 *// 3.1 创建用户名和密码的 cookie* Cookie usernameCookie = new Cookie("username", username);  
 Cookie passwordCookie = new Cookie("password", password);  
 *// 3.2 将创建好的 cookie 对象, 添加到 response 对象中, 响应给浏览器实现存储* response.addCookie(usernameCookie);  
 response.addCookie(passwordCookie);  
 }  
 }  
}

@WebServlet("/getCookie")  
public class GetCookieServlet extends HttpServlet {  
 protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 doGet(request, response);  
 }  
  
 protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 *// 1. 从请求中获取所有的 cookie 对象* Cookie[] cookies = request.getCookies();  
 *// System.out.println("cookie = " + cookies); 说明: 如果没有 cookie 对象, 则返回为 null.  
 // 2. 遍历 cookie 数组* if (cookies != null) {  
 for (Cookie cookie : cookies) {  
 *// 获取用户名 cookie 和密码 cookie* String name = cookie.getName();  
 String value = cookie.getValue();  
  
 *// 实现数据的解码* value = URLDecoder.*decode*(value, "utf-8");  
  
 System.*out*.println(name + " = " + value);  
 }  
 } else {  
 System.*out*.println("没有任何 cookie 数据.");  
 }  
 }  
}

**小结 :**

特殊字符必须手动实现 `编码 URLEncoder.encode()` 和 `解码 URLDecoder.decode()` 才能保证数据的正常存储与获取.

# 49. Cookie 持久化设置 :

**目标 :** 持久化cookie的存储.

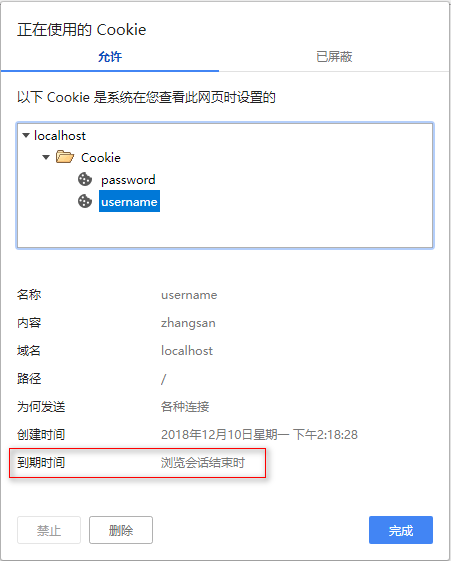
**步骤 :**

一 : 创建cookie对象.

二 : 调用 cookie 对象的 setMaxAge() 方法, 传入对象的秒数.

**实现 :**





每次关闭浏览器, 重新打开时, cookie就已经被销毁了, 因为所有的cookie都是会话临时性cookie.

@WebServlet("/login")  
public class Login2Servlet extends HttpServlet {  
 protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 doGet(request, response);  
 }  
  
 protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 *// 1. 设置请求编码* request.setCharacterEncoding("utf-8");  
 *// 2. 获取用户名, 密码, 记住密码* String username = request.getParameter("username");  
 String password = request.getParameter("password");  
 String check = request.getParameter("check");  
 *// 3. 判断用户是否点击了 `记住密码`* if ("on".equals(check)) {  
  
 *// 实现数据编码* username = URLEncoder.*encode*(username, "utf-8");  
 password = URLEncoder.*encode*(password, "utf-8");  
  
 *// 3.1 创建用户名和密码的 cookie* Cookie usernameCookie = new Cookie("username", username);  
 Cookie passwordCookie = new Cookie("password", password);  
  
 *// cookie 的持久化设置 (存储一星期)* usernameCookie.setMaxAge(60 \* 60 \* 24 \* 7);  
 passwordCookie.setMaxAge(60 \* 60 \* 24 \* 7);  
  
 *// 3.2 将创建好的 cookie 对象, 添加到 response 对象中, 响应给浏览器实现存储* response.addCookie(usernameCookie);  
 response.addCookie(passwordCookie);  
 }  
 }  
}

**小结 :**

通过调用 cookie 对象的 setMaxAge() 方法, 传入对应的时间秒数, 就可以将一个会话时期的cookie对象转换为持久化cookie.

# 50. Cookie 有效路径设置 :

**目标 :** cookie的有效访问路径设置.

一个项目中可能会出现许多不同的模块, 例如: 商城模块, 用户模块, 商品模块, 活动模块 …, 此时, 我们希望有些cookie数据仅在对应模块访问时携带, 就需要设置cookie的有效访问路径了. 如果不设置, 那么cookie的默认有效访问路径为全局.

**步骤 :**

一 : 创建setPathServlet 类, 将 cookie 的有效访问路径设置为 /user.

二 : 创建getPath1Servlet 类, 将servlet的url-pattern设置为 /user/getCookie..

三 : 创建getPath2Servlet 类, 将servlet的url-pattern设置为 /web/getCookie..

四 : 查看访问结果.

**实现 :**

@WebServlet("/setPath")  
public class SetPathServlet extends HttpServlet {  
 protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 doGet(request, response);  
 }  
  
 protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 *// 1. 创建 cookie* Cookie cookie = new Cookie("path", "上海浦东新区");  
 *// 2. 设置访问路径* cookie.setPath("/user");  
 *// 3. 将 cookie 添加到响应中* response.addCookie(cookie);  
 }  
}

@WebServlet("/user/getCookie")  
public class GetPath1Servlet extends HttpServlet {  
 protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 doGet(request, response);  
 }  
  
 protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 *// 1. 获取 cookie 数组* Cookie[] cookies = request.getCookies();  
 *// 2. 判断并遍历 cookie 数组* if (cookies != null) {  
 for (Cookie cookie : cookies) {  
 String name = cookie.getName();  
 String value = cookie.getValue();  
 System.*out*.println(name + " = " + value);  
 }  
 } else {  
 System.*out*.println("没有任何cookie数据.");  
 }  
 }  
}









@WebServlet("/web/getCookie")  
public class GetPath2Servlet extends HttpServlet {  
 protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 doGet(request, response);  
 }  
  
 protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 *// 1. 获取 cookie 数组* Cookie[] cookies = request.getCookies();  
 *// 2. 判断并遍历 cookie 数组* if (cookies != null) {  
 for (Cookie cookie : cookies) {  
 String name = cookie.getName();  
 String value = cookie.getValue();  
 System.*out*.println(name + " = " + value);  
 }  
 } else {  
 System.*out*.println("没有任何cookie数据.");  
 }  
 }  
}





**小结 :**

通过调用 cookie 对象的 setPath() 方法, 设置有效的 cookie 访问路径.

# 51. Cookie 的删除 :

**目标 :** 删除持久化的 cookie 对象.

**步骤 :**

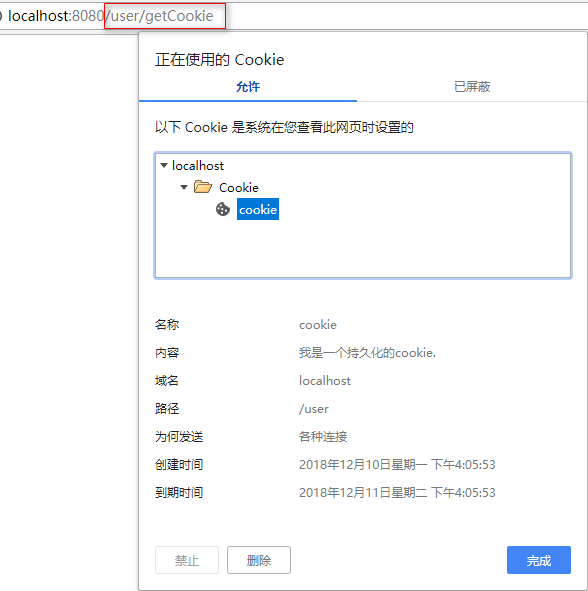
一 : 创建setCookieServlet 类, 将 cookie 的有效访问路径设置为 /user. 并实现 cookie 的持久化

二 : 创建DeleteCookieServlet 类, 完成 cookie 对象的删除.

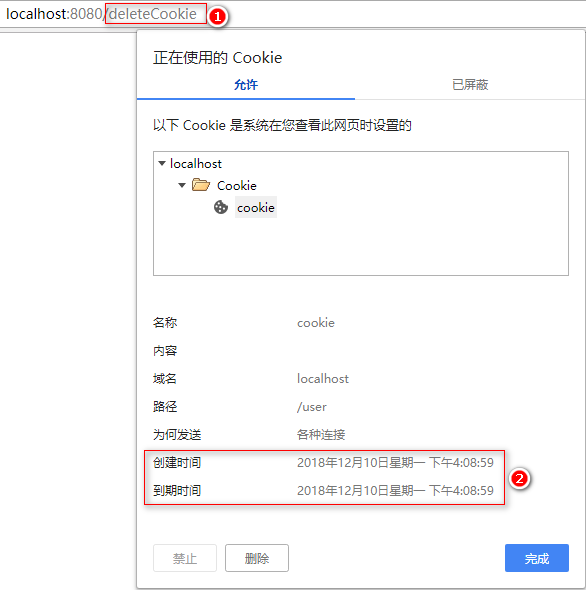
**实现 :**

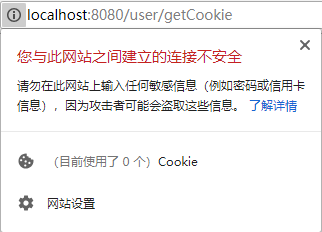


@WebServlet("/setCookie")  
public class SetCookieServlet extends HttpServlet {  
 protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 doGet(request, response);  
 }  
  
 protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 *// 1. 创建一个 cookie 对象* Cookie cookie = new Cookie("cookie", "我是一个持久化的cookie.");  
 *// 2. 持久化 cookie* cookie.setMaxAge(60 \* 60 \* 24);  
 *// 3. 设置有效访问路径* cookie.setPath("/user");  
 *// 4. 将 cookie 添加到响应中, 返回给浏览器存储* response.addCookie(cookie);  
 }  
}



@WebServlet("/deleteCookie")  
public class DeleteCookieServlet extends HttpServlet {  
 protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 doGet(request, response);  
 }  
  
 protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 *// 1. 创建一个与要删除相同名称的 cookie 对象* Cookie cookie = new Cookie("cookie", "");  
 *// 2. 设置存活时间为 0* cookie.setMaxAge(0);  
 *// 3. 设置相同的访问路径* cookie.setPath("/user");  
 *// 4. 将 cookie 响应到浏览器中, 实现 cookie 的自动置换* response.addCookie(cookie);  
 }  
}





**小结 :**

设置相同名称的 cookie 对象, 可以实现 cookie 对象的置换. 从而实现 cookie 对象的删除.

# 52. Session 概述 :

Cookie技术可以将用户的信息保存在各自的浏览器中，

优点：很明显实现了同一个用户不同请求中数据共享。

缺点：黑客可以利用脚本等手段 窃取cookie中的重要数据，从而泄漏个人的隐私，存在巨大的安全隐患。

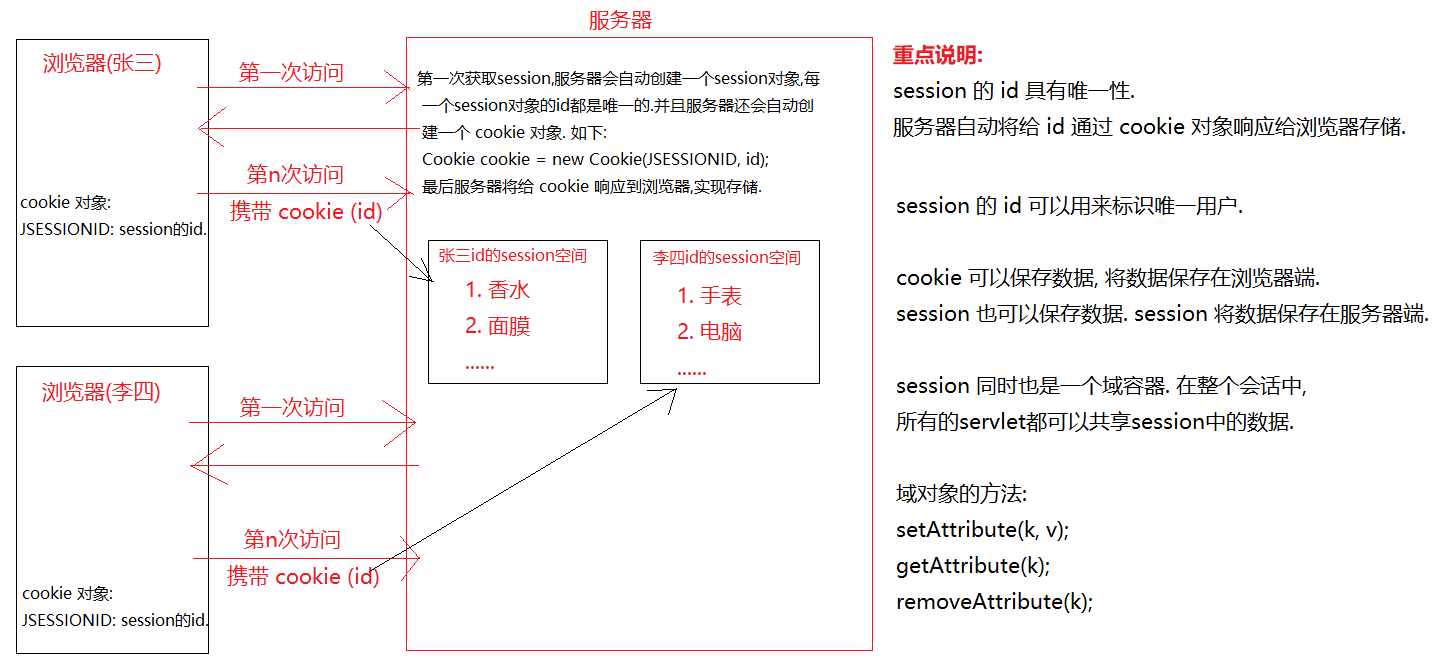
session技术是将会话的数据保存在服务器端的技术。

* 在WEB开发中，服务器可以为每个用户浏览器创建一个会话对象（session对象），注意：一个浏览器独占一个session对象(默认情况下)。因此，在需要保存用户数据时，服务器程序可以把用户数据写到用户浏览器独占的session中，当用户使用浏览器访问其它程序时，其它程序可以从用户的session中取出该用户的数据，为用户服务。
* Session和Cookie的主要区别在于：

Cookie是把用户的数据写给用户的浏览器。

Session技术把用户的数据写到用户独占的session中（服务器端）。

Session对象由服务器创建，开发人员可以调用request对象的getSession方法得到session对象。

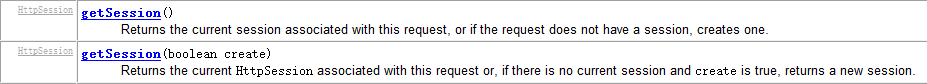


# 53. 获取Session对象并查看id :

Session是基于用户的请求，而把用户的重要信息在服务器端针对这个用户（浏览器）创建了一个容器。

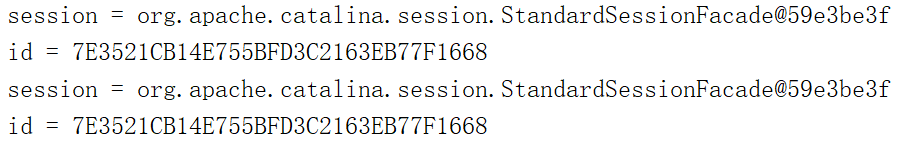
而这个Session容器是由web服务器(tomcat)帮助我们创建的，在程序中我们只能去获取到这个容器，然后给容器添加数据或者取出数据，或者删除数据，而我们是不能创建这个容器对象。

在HttpServletRequest对象中提供了获取session对象的方法：

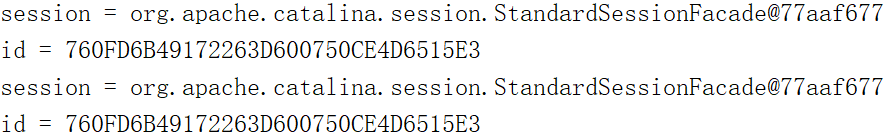


**说明 :** 如果 create 参数为 true, 当前没有 session 对象时, 则会创建一个新的 session 对象返回, 如果 create 参数为 null, 当前如果没有 session 对象, 则返回 null. 不创建 session 对象. 默认 true.

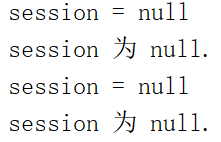
@WebServlet("/session1")  
public class Session1Servlet extends HttpServlet {  
 protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 doGet(request, response);  
 }  
  
 protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 *// 1. 获取 session 对象* HttpSession session = request.getSession();  
 System.*out*.println("session = " + session);  
 *// 2. 查看 session 对象的 id* String id = session.getId();  
 System.*out*.println("id = " + id);  
 }  
}



protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 *// 1. 获取 session 对象* HttpSession session = request.getSession(true);  
 System.*out*.println("session = " + session);  
 *// 2. 查看 session 对象的 id* String id = session.getId();  
 System.*out*.println("id = " + id);  
}



protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 *// 1. 获取 session 对象* HttpSession session = request.getSession(false);  
 System.*out*.println("session = " + session);  
 if (session != null) {  
 *// 2. 查看 session 对象的 id* String id = session.getId();  
 System.*out*.println("id = " + id);  
 } else {  
 System.*out*.println("session 为 null.");  
 }  
}



# 54. session存储与获取数据 :

**目标 :** 实现session域容器的存储与获取数据.

**步骤 :**

一 : 创建一个 SetSessionServlet 类, 向 session 容器中存储数据.

二 : 创建一个 GetSessionServlet 类, 从 session 容器中取出数据.

**实现 :**

@WebServlet("/setSession")  
public class SetSessionServlet extends HttpServlet {  
 protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 doGet(request, response);  
 }  
  
 protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 *// 1. 获取 session 对象* HttpSession session = request.getSession();  
 *// 2. 向 session 域容器中存储数据* session.setAttribute("session", "我是session中的数据.");  
 }  
}

@WebServlet("/getSession")  
public class GetSessionServlet extends HttpServlet {  
 protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 doGet(request, response);  
 }  
  
 protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 *// 1. 获取 session 对象* HttpSession session = request.getSession();  
 *// 2. 从 session 容器中取出数据* Object obj = session.getAttribute("session");  
 System.*out*.println("obj = " + obj);  
 }  
}



**小结 :**

session 可以作为容器使用, 整个会话中的 servlet 可以共享 session 域中的数据.

# 55. session 完善下载案例 :

**目标 :** 下载操作之前, 用户必须先登录.

**步骤 :**

一 : 创建一个 download.html 下载页面.

二 : 创建 DownloadServlet 类. 首先从 session 中是否存在 user 对象, 如果没有, 需要用户先实现登录.

三 : 创建 login.html 登录页面.

四 : 创建 LoginServlet 类, 将用户存储到 session 域中.

**实现 :**

download.html 下载页面

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <title>文件下载</title>  
</head>  
<body>  
 <a href="/download?filename=1.jpg">1.jpg</a><br>  
 <a href="/download?filename=2.txt">2.txt</a><br>  
 <a href="/download?filename=3.zip">3.zip</a><br>  
 <a href="/download?filename=Sublime.exe">Sublime.exe</a>  
</body>  
</html>

@WebServlet("/download")  
public class DownloadServlet extends HttpServlet {  
 protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 doGet(request, response);  
 }  
  
 protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
  
 *// 第一步: 获取 session, 判断 session 中是否存储了 user 信息* HttpSession session = request.getSession();  
 *// 第二步: 从 session 中取出 user 对象* User user = (User) session.getAttribute("user");  
 *// 第三步: 判断* if (user == null) {  
 *// 用户没有登录, 先登录, 重定向到登录页面* response.sendRedirect("/login.html");  
 return;  
 } else {  
 *// 用户已经登录, 实现资源下载  
 // 1. 获取文件名* String filename = request.getParameter("filename");  
 *// 2. 获取文件真实路径* ServletContext context = this.getServletContext();  
 String realPath = context.getRealPath("/WEB-INF/" + filename);  
 *// 3. 创建 file 对象* File file = new File(realPath);  
 *// 4. 判断文件是否存在* if (file.exists()) {  
 *// 5. 实现下载  
 // 5.1 获取文件类型* String mimeType = context.getMimeType(filename);  
 *// 5.2 设置内容类型* response.setContentType(mimeType);  
 *// 5.3 告诉浏览器, 以附件的形式实现下载* response.setHeader("Content-Disposition", "attachment;filename=" + filename);  
  
 *// 5.4 创建字节输入, 输出流* BufferedInputStream bis = new BufferedInputStream(new FileInputStream(file));  
 BufferedOutputStream bos = new BufferedOutputStream(response.getOutputStream());  
  
 *// 5.5 读写操作* byte[] buf = new byte[1024];  
 int len = -1;  
 while ((len = bis.read(buf)) != -1) {  
 bos.write(buf, 0, len);  
 bos.flush();  
 }  
  
 *// 5.6 关闭资源* bos.close();  
 bis.close();  
 }  
 }  
 }  
}

public class User {  
 private String username;  
 private String password;  
  
 public User(String username, String password) {  
 this.username = username;  
 this.password = password;  
 }  
  
 public User() {  
 }  
  
 @Override  
 public String toString() {  
 return "User{" +  
 "username='" + username + '\'' +  
 ", password='" + password + '\'' +  
 '}';  
 }  
  
 public String getUsername() {  
 return username;  
 }  
  
 public void setUsername(String username) {  
 this.username = username;  
 }  
  
 public String getPassword() {  
 return password;  
 }  
  
 public void setPassword(String password) {  
 this.password = password;  
 }  
}

login.html 页面 :

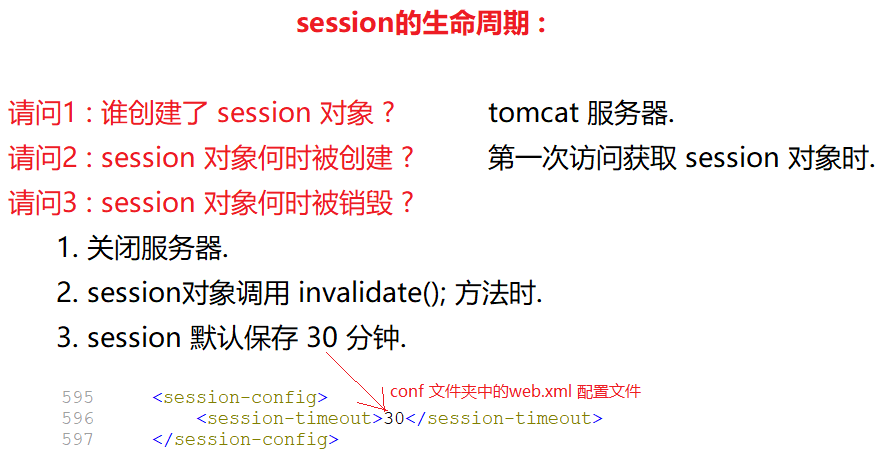
<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <title>登录</title>  
</head>  
<body>  
<h3>用户登录</h3>  
<hr>  
<form action="/login" method="post">  
 <table>  
 <tr>  
 <td><label for="username">用户名:</label></td>  
 <td><input type="text" name="username" id="username"></td>  
 </tr>  
 <tr>  
 <td><label for="password">用户名:</label></td>  
 <td><input type="password" name="password" id="password"></td>  
 </tr>  
 <tr>  
 <td colspan="2"><input type="checkbox" name="check">记住密码</td>  
 </tr>  
 <tr>  
 <td></td>  
 <td><input type="submit" value="登录"></td>  
 </tr>  
 </table>  
</form>  
</body>  
</html>

@WebServlet("/login")  
public class LoginServlet extends HttpServlet {  
 protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 doGet(request, response);  
 }  
  
 protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 *// 1. 设置请求编码* request.setCharacterEncoding("utf-8");  
 *// 2. 获取用户名, 密码, 记住密码* String username = request.getParameter("username");  
 String password = request.getParameter("password");  
 *// 3. 将数据封装为一个 User 对象* User user = new User(username, password);  
 *// 4. 获取 session 对象* HttpSession session = request.getSession();  
 *// 5. 将 user 对象存储到 session 中.* session.setAttribute("user", user);  
 }  
}

**小结 :**

用户登录后, 将用户存储到 session 中, 因此就可以实现, 如果用户需要下载资源, 判断用户是否已经登录.

# 56. session 的生命周期 :



# 57. session 持久化解决方案 :

**目标 :** 保证服务器为每个浏览器仅创建一个session对象.

Session默认存活时间为 30 分钟, 而 cookie 默认存活时间为当前会话时间, 因此, 如果在30分钟内, 重复关闭浏览器, 重复访问服务器, 服务器内部就会为该浏览器创建多个 session 对象, 从而造成服务器内存空间的浪费.

**步骤 :**

一 : 自定义类实现Servlet接口.

二 : 重写 Servlet 接口中的所有抽象方法.

三 : 将 service 方法定义为抽象方法, 因为开发中最重要的步骤就是业务逻辑的实现.

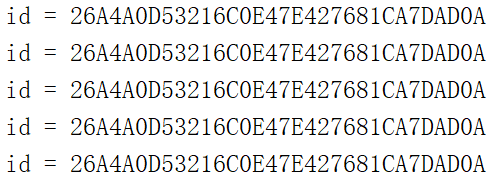
四 : 父类也可以提供一些扩展方法, 方便子类继承时实现直接使用. 如: log 方法等…

五 : 完成上述步骤后, 编写 SecondServlet 类, 继承自 GenericServlet 类, 重写 service 抽象方法即可.

**实现 :**



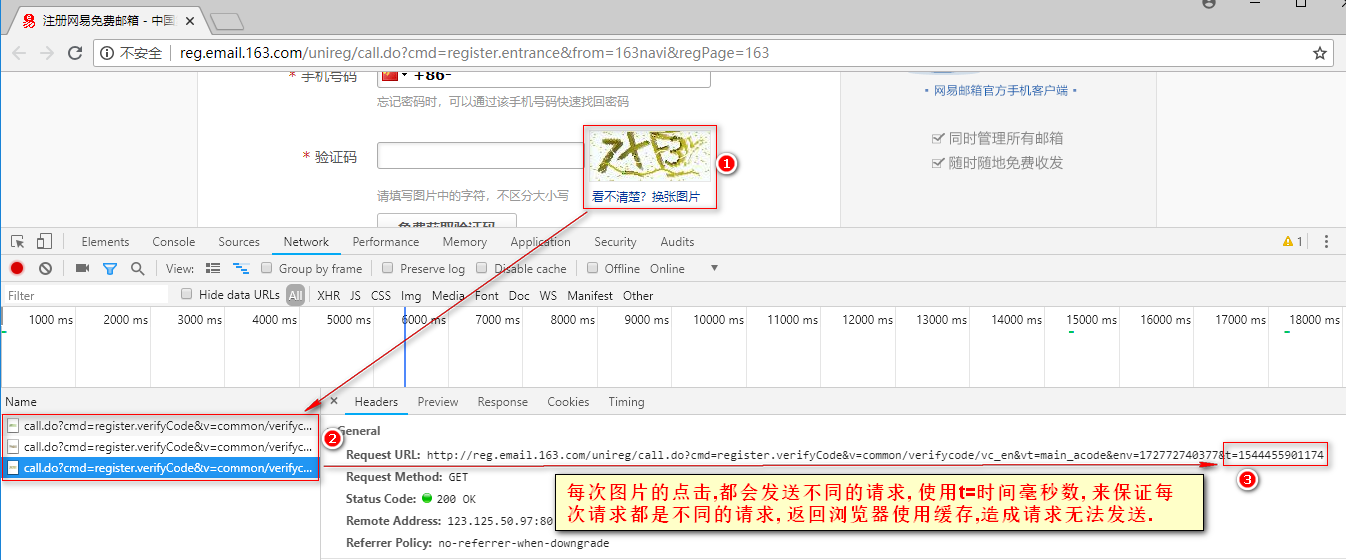
@WebServlet("/sessionId")  
public class SessionIdServlet extends HttpServlet {  
 protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 doGet(request, response);  
 }  
  
 protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 *// 1. 获取 session, 并打印 session 对象的 id.* HttpSession session = request.getSession();  
 String id = session.getId();  
 System.*out*.println("id = " + id);  
  
 *// 2. session 持久化解决方案  
 // 2.1 手动创建 cookie 对象, key 必须定义为 JSESSIONID.* Cookie cookie = new Cookie("JSESSIONID", id);  
 *// 2.2 设置 cookie 的存活时间为 30 分钟, 与 session 存活时间一致.* cookie.setMaxAge(60 \* 30);  
 *// 2.3 将 cookie 响应到浏览器中* response.addCookie(cookie);  
 }  
}



**小结 :**

持久化 session 的方案就是将 cookie 的 key 设置为 JSESSIONID, 然后设置 cookie 的存活时间与session一致. 这样就可以保证服务端为每个浏览器对象仅创建一个 session 对象. 从而减少服务端内存的消耗.

# 58. 验证码 Servlet 创建 :





<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <title>注册</title>  
  
 <style>  
  
 */\* 1. 设置所有元素的内外边距为0，设置box-sizing为 border-box，  
 即：边框的宽和高就是盒子的宽和高，不会因为内边距或内容的变大而撑大。\*/* \* {  
 padding: 0px;  
 margin: 0px;  
 box-sizing: border-box;  
 }  
  
 */\* 2. 创建外面的白色div容器，类名rg\_form，宽886，高534，背景色为白色，上下外边距24，左右自动，  
 边框为8，实线，颜色为#eee\*/* .rg\_form {  
 width: 886px;  
 height: 534px;  
 background-color: white;  
 margin: 24px auto;  
 border: 8px solid #eee;  
  
 }  
  
 */\* 3. 左边的的容器，类名为rg\_form\_left，宽度为256，左浮动，上内边距20，左内边距20\*/* .rg\_form\_left {  
 width: 256px;  
 float: left;  
 padding-top: 20px;  
 padding-left: 20px;  
 }  
  
 */\* 4. "新用户注册"汉字，第1行使用p标签，可以使用伪类样式，p:first-child，字体20，颜色：#ffcd26\*/* .rg\_form\_left p:first-child {  
 font-size: 20px;  
 color: #ffcd26;  
 }  
 */\* 5. "新用户注册"第2行英文使用p标签，内容：USER REGISTER，使用伪类样式，p:last-child，字体20，颜色：#a6a6a6\*/* .rg\_form\_left p:last-child {  
 font-size: 20px;  
 color: #a6a6a6;  
 }  
  
 */\* 6. 右边"已有账号"的容器类名 rg\_form\_right，宽256，右浮动。\*/* .rg\_form\_right {  
 width: 256px;  
 float: right;  
 }  
  
 */\* 7. 右边的文字p标签，右浮动，字大小14，内边距，上20，右12，下0，左0\*/* .rg\_form\_right p {  
 float: right;  
 font-size: 14px;  
 padding: 20px 12px 0px 0px;  
 }  
  
 */\* 8. p内部有个a标签，颜色#fc8989\*/* .rg\_form\_right p a {  
 color: #fc8989;  
 }  
  
 */\* 9. 中间注册窗体的容器类名rg\_form\_center，宽358，左浮动，上内边距10，字体大小14\*/* .rg\_form\_center {  
 width: 358px;  
 float: left;  
 padding-top: 10px;  
 font-size: 14px;  
 }  
  
 */\* 10. 中间容器中的表格类名rg\_table，上外边距25\*/* .rg\_table {  
 margin-top: 25px;  
 }  
  
 */\* 11. 表单中左边的文字所在单元格，类名td\_left，宽65，文本右对齐\*/* .td\_left {  
 width: 65px;  
 text-align: right;  
 }  
  
 */\* 12. 表单中右边的输入框单元格类名td\_right，宽293，高50\*/* .td\_right {  
 width: 293px;  
 height: 50px;  
 }  
  
 */\* 13. 表单中的文本框，日期框，密码框使用属性选择器，宽256，高32，行高32，  
 内边距上下6，左右12，圆角4，边框：1 实线 颜色#a6a6a6，右浮动\*/* input[type="text"], input[type="date"], input[type="password"] {  
 width: 256px;  
 height: 32px;  
 line-height: 32px;  
 padding: 6px 12px;  
 border-radius: 4px;  
 border: 1px solid #a6a6a6;  
 float: right;  
 }  
  
 */\* 14. 性别(男,女选项)类名：gender，左内边距40\*/* .gender {  
 padding-left: 40px;  
 }  
  
 */\* 15. 验证码输入框类名：check，(必须使用.td\_right > .check)宽118，左浮动，左外边距35，右外边距14  
 (验证码文件可以设置最大长度,验证码图片高度32)\*/* .td\_right > .check {  
 width: 118px;  
 float: left;  
 margin-left: 35px;  
 margin-right: 14px;  
  
 }  
  
 */\* 16. 提交按钮类名：submit，宽120，高36，行高36，背景色#ffc900，字体大小14，边框类型为none，左外边距35.\*/* .submit {  
 width: 120px;  
 height: 36px;  
 line-height: 36px;  
 background-color: #ffc900;  
 font-size: 14px;  
 border-style: none;  
 margin-left: 35px;  
 }  
  
 </style>  
  
 <script>  
 function *changeCode*(img) {  
 img.src = "/code?t=" + new *Date*().getTime();  
 }  
 </script>  
  
</head>  
<body>  
*<!--最外层容器-->*<div class="rg\_form">  
 *<!--左容器-->* <div class="rg\_form\_left">  
 <p>新用户注册</p>  
 <p>USER REGISTER</p>  
 </div>  
 *<!--中间容器-->* <div class="rg\_form\_center">  
 *<!--table 容器-->* <div class="rg\_table">  
 *<!--表单域 form-->* <form action="/register" method="post" name="registerForm">  
 *<!--table : 9行2列-->* <table>  
 *<!--第一行-->* <tr>  
 <td class="td\_left"><label for="username">用户名</label></td>  
 <td class="td\_right"><input type="text" name="username" id="username" placeholder="请输入账号..."></td>  
 </tr>  
 *<!--第二行-->* <tr>  
 <td class="td\_left"><label for="password">密码</label></td>  
 <td class="td\_right"><input type="password" name="password" id="password" placeholder="请输入密码..."></td>  
 </tr>  
 *<!--第三行-->* <tr>  
 <td class="td\_left"><label for="email">邮箱</label></td>  
 <td class="td\_right"><input type="text" name="email" id="email" placeholder="请输入邮箱..."></td>  
 </tr>  
 *<!--第四行-->* <tr>  
 <td class="td\_left"><label for="name">姓名</label></td>  
 <td class="td\_right"><input type="text" name="name" id="name" placeholder="请输入真实姓名..."></td>  
 </tr>  
 *<!--第五行-->* <tr>  
 <td class="td\_left"><label for="phone">手机号</label></td>  
 <td class="td\_right"><input type="text" name="phone" id="phone" placeholder="请输入手机号码..."></td>  
 </tr>  
 *<!--第六行-->* <tr>  
 <td class="td\_left"><label for="gender">性别</label></td>  
 <td class="td\_right gender">  
 <input type="radio" name="gender" id="gender" value="男"> 男  
 <input type="radio" name="gender" value="女" checked="checked"> 女  
 </td>  
 </tr>  
 *<!--第七行-->* <tr>  
 <td class="td\_left"><label for="birthday">生日</label></td>  
 <td class="td\_right"><input type="date" name="birthday" id="birthday"></td>  
 </tr>  
 *<!--第八行-->* <tr>  
 <td class="td\_left"><label for="code">验证码</label></td>  
 <td class="td\_right">  
 <input type="text" name="code" id="code" class="check" maxlength="4">  
 <img src="/code" alt="验证码图片" height="32px" onclick="*changeCode*(this);">  
 </td>  
 </tr>  
 *<!--第九行-->* <tr>  
 <td class="td\_left"></td>  
 <td class="td\_right"><input type="submit" value="注册" class="submit"></td>  
 </tr>  
 </table>  
 </form>  
 </div>  
 </div>  
 *<!--右容器-->* <div class="rg\_form\_right">  
 <p>已有账号?<a href="#">立即登录</a></p>  
 </div>  
</div>  
</body>  
</html>

**验证码创建 :**

import javax.imageio.ImageIO;  
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.awt.\*;  
import java.awt.image.BufferedImage;  
import java.io.IOException;  
import java.util.Random;  
  
@WebServlet("/code")  
public class CodeServlet extends HttpServlet {  
 protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 doGet(request, response);  
 }  
  
 protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 *// 1. 定义验证码图片宽高* int width = 60;  
 int height = 30;  
  
 *// 2. 根据宽高创建一张图片* BufferedImage image = new BufferedImage(width, height, BufferedImage.*TYPE\_INT\_RGB*);  
  
 *// 4. 获取画笔* Graphics g = image.getGraphics();  
 *// 5. 设置画笔颜色, 并填充一个白色内部矩形框* g.setColor(Color.*white*);  
 g.fillRect(1, 1, width-2, height-2);  
 *// 6. 设置文字字体, 并拼接随机字符串* g.setFont(new Font("宋体", Font.*BOLD* | Font.*ITALIC*, 25));  
 String data = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789";  
 StringBuilder code = new StringBuilder();  
 Random r = new Random();  
 for (int i = 0; i < 4; i++) {  
 *// 设置随机画笔颜色* g.setColor(new Color(r.nextInt(255), r.nextInt(255), r.nextInt(255)));  
 *// 获取一个随机下标的字符* int index = r.nextInt(data.length());  
 String str = data.charAt(index) + "";  
 code.append(str);  
 *// 将字符写入到画布框中* g.drawString(str, 9 \* (i + 1), 22);  
 }  
  
 *// 7. 随机绘制多条干扰线* for (int i = 0; i < 5; i++) {  
 *// 设置随机画笔颜色* g.setColor(new Color(r.nextInt(255), r.nextInt(255), r.nextInt(255)));  
 *// 干扰线* g.drawLine(r.nextInt(width), r.nextInt(height), r.nextInt(width), r.nextInt(height));  
 *// 随机点* g.drawOval(r.nextInt(width), r.nextInt(height), 2, 2);  
 }  
  
 *// 8. 将验证码保存到 session 域中.* request.getSession().setAttribute("code", code.toString());  
 System.*out*.println("code = " + code.toString()); *// 控制台查看验证码  
   
 // 3. 将图片响应给浏览器* ImageIO.*write*(image, "jpg", response.getOutputStream());  
 }  
}

# 59. 登录验证码校验 :

**目标 :** 校验用户输入的验证码是否正确.

**步骤 :**

一 : 创建 RegisterServlet 类.

二 : 从 request 对象中获取用户输入的验证码.

三 : 再从 request 对象中获取生成的验证码.

四 : 两个验证码实现判断比较即可.

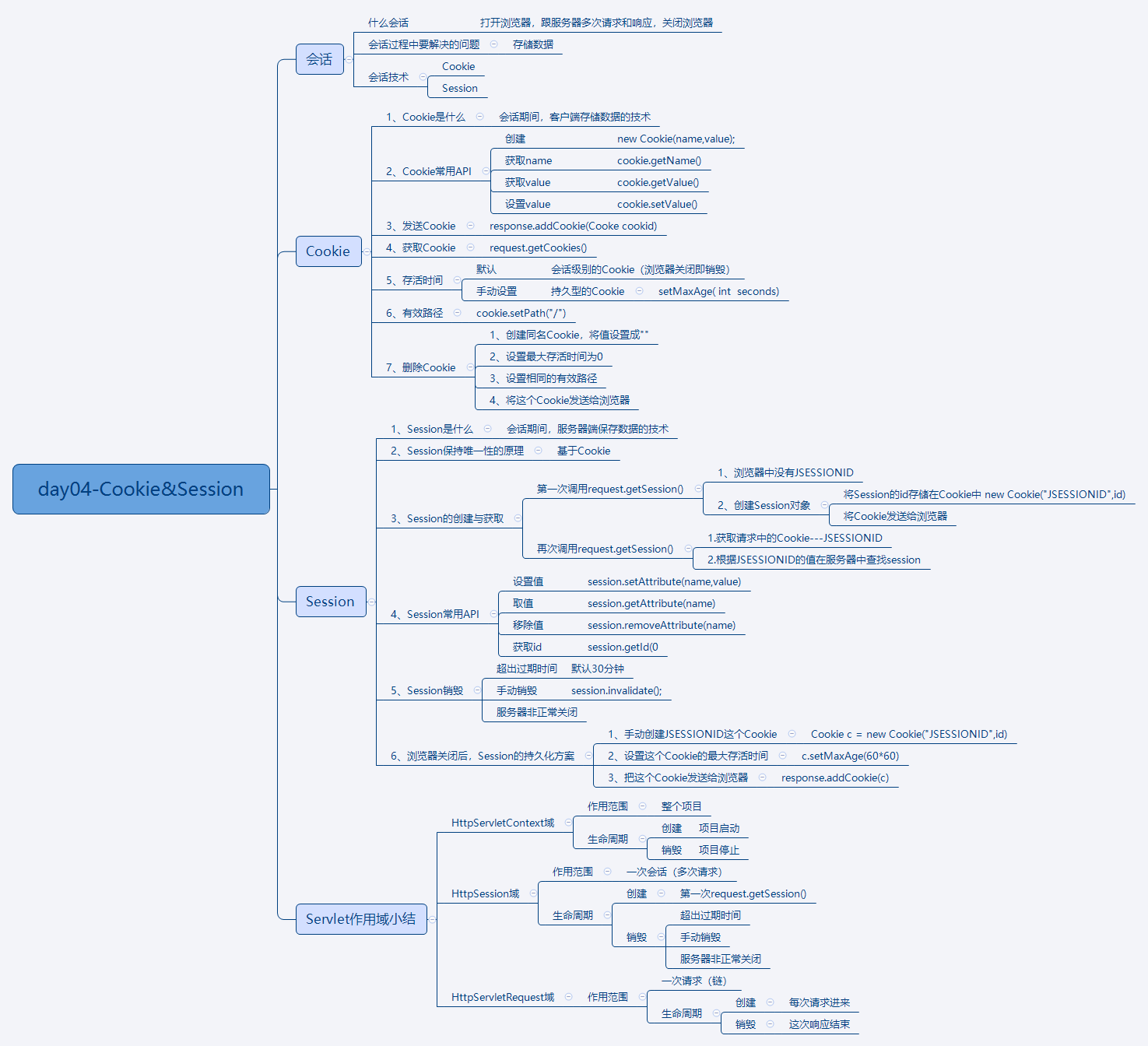
**实现 :**

@WebServlet("/register")  
public class RegisterServlet extends HttpServlet {  
 protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 doGet(request, response);  
 }  
  
 protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 *// 1. 获取用户提交的数据* String username = request.getParameter("username");  
 String password = request.getParameter("password");  
 String code = request.getParameter("code");  
 *// 2. 判断用户输入的验证码和session中随机生成的验证码是否一致* String sessionCode = (String) request.getSession().getAttribute("code");  
 if (sessionCode.equalsIgnoreCase(code)) {  
 *// 验证码输入正确* System.*out*.println("验证码输入正确...");  
 } else {  
 *// 验证码输出错误* System.*out*.println("错误, 错误, 错误...");  
 }  
 }  
}

**小结 :**

从 request 对象中同时获取用户输入的验证码和 session 中存储的生成验证码, 实现验证码判断, 这样就能够解决暴力登录和恶意攻击的问题.

# 60. cookie与session小结 :



# 61. 登录成功后显示昵称 :

**目标 :** 登录成功后, 访问任何一个页面, 都要显示自己的昵称.

**步骤 :**

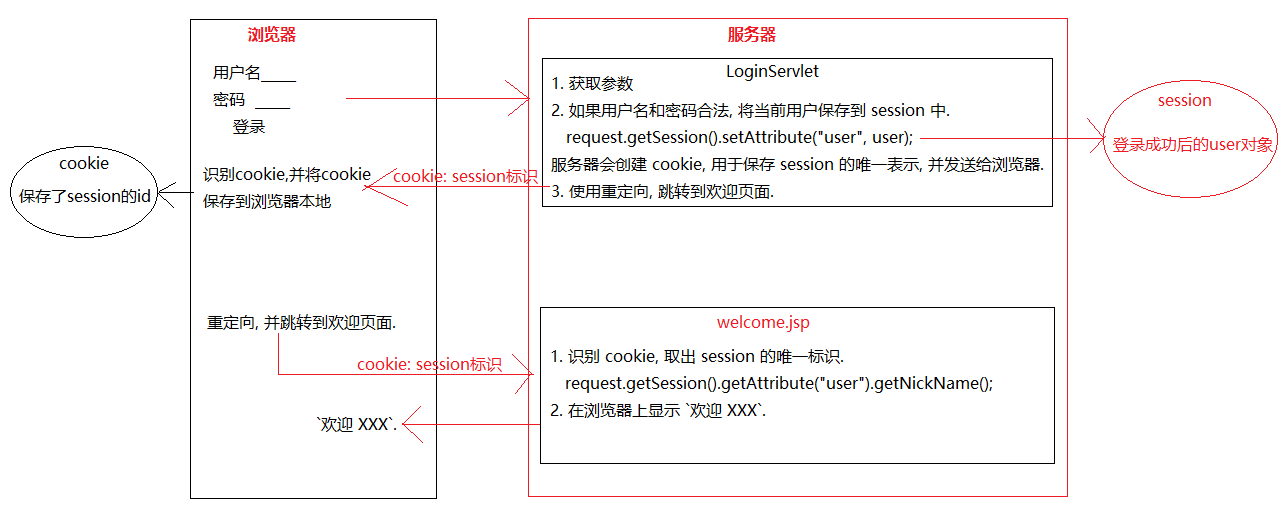
一 : 创建数据库与数据表.

二 : 创建数据表对应的实体类对象.

三 : JDBCUtils工具类与配置文件信息.

四 : 编写 login.jsp 页面.

五 : 创建 LoginServlet 类, 完成对应的业务逻辑处理.



创建数据库与数据表 :

create database sessiondb;

use sessiondb;

create table user (

id int primary key auto\_increment,

username varchar(20),

password varchar(20),

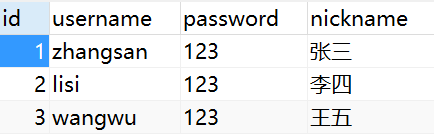
nickname varchar(20)

);

insert into user values(null,'zhangsan','123','张三');

insert into user values(null,'lisi','123','李四');

insert into user values(null,'wangwu','123','王五');



**实体类对象 :**

public class User {  
 *// 属性* private int id;  
 private String username;  
 private String password;  
 private String nickname;  
  
 *// 注意 : 必须要提供一个公开的无参构造方法, 方便框架创建该 JavaBean 对象.* public User() {  
 }  
  
 public User(int id, String username, String password, String nickname) {  
 this.id = id;  
 this.username = username;  
 this.password = password;  
 this.nickname = nickname;  
 }  
  
 public User(String username, String password) {  
 this.username = username;  
 this.password = password;  
 }  
  
 @Override  
 public String toString() {  
 return "User{" +  
 "id=" + id +  
 ", username='" + username + '\'' +  
 ", password='" + password + '\'' +  
 ", nickname='" + nickname + '\'' +  
 '}';  
 }  
  
 public int getId() {  
 return id;  
 }  
  
 public void setId(int id) {  
 this.id = id;  
 }  
  
 public String getUsername() {  
 return username;  
 }  
  
 public void setUsername(String username) {  
 this.username = username;  
 }  
  
 public String getPassword() {  
 return password;  
 }  
  
 public void setPassword(String password) {  
 this.password = password;  
 }  
  
 public String getNickname() {  
 return nickname;  
 }  
  
 public void setNickname(String nickname) {  
 this.nickname = nickname;  
 }  
}

JDBCUtils 工具类与 数据库连接池配置文件. 注意配置文件中的账号, 密码, 与数据库名称.

**实现 :**

编写登录页面 : login.html

<%@ page contentType="text/html;charset=UTF-8" language="java" %>  
<html>  
<head>  
 <title>登录</title>  
  
 <script>  
 function *changeCode*(img) {  
 img.src = "/code?t=" + new *Date*().getTime();  
 }  
 </script>  
</head>  
<body>  
 *<%--登录失败的提示信息--%>* <div style="color: red;">${requestScope.errorMsg }</div>  
 <form action="/login" method="post">  
 <table>  
 <tr>  
 <td><label for="username">用户名</label></td>  
 <td><input type="text" id="username" name="username" value="${cookie.username.value }"></td>  
 </tr>  
 <tr>  
 <td><label for="password">密码</label></td>  
 <td><input type="password" id="password" name="password"></td>  
 </tr>  
 <tr>  
 <td><label for="code">验证码</label></td>  
 <td>  
 <input type="text" id="code" name="code">  
 <img src="/code" alt="" id="验证码" style="cursor: pointer;" onclick="*changeCode*(this);">  
 </td>  
 </tr>  
 <tr>  
 <td></td>  
 <td><input type="submit" value="登录"></td>  
 </tr>  
 </table>  
 </form>  
  
</body>  
</html>

@WebServlet("/login")  
public class LoginServlet extends HttpServlet {  
 protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 doGet(request, response);  
 }  
  
 protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 *// 1. 校验验证码的合法性* String session\_code = (String) request.getSession().getAttribute("code");  
 String code = request.getParameter("code");  
 *// 判断* if (session\_code == null || session\_code.equalsIgnoreCase(code) == false) {  
 *// 不合法, 请求转发跳转到登录页面.提示验证码错误, 并将错误信息存储到 request 中* request.setAttribute("errorMsg", "验证码错误!");  
 request.getRequestDispatcher("/login.jsp").forward(request, response);  
 return; *// 结束执行* }  
  
 *// 2. 验证码正确, 校验用户名和密码的合法性* String username = request.getParameter("username");  
 String password = request.getParameter("password");  
 User user = new User();  
 user.setUsername(username);  
 user.setPassword(password);  
 *// 创建业务层对象* UserService userService = new UserService();  
 User loginUser = userService.login(user);  
 *// 判断返回的结果* if (loginUser != null) {  
 *// 登录成功  
 // 将信息存储到 session 中.* request.getSession().setAttribute("user", loginUser);  
 *// 设置 cookie, 完成重新访问登录页面, 免输入用户名* Cookie cookie = new Cookie("username", loginUser.getUsername());  
 cookie.setPath("/");  
 cookie.setMaxAge(60 \* 60 \* 24 \* 30); *// 有效时间为一个月* response.addCookie(cookie);  
 *// 重定向跳转到欢迎页面  
 // 请问 : 为什么书写 jsp 页面, 因为 jsp 页面即可以书写 html 代码, 也可以书写 Java 代码.* response.sendRedirect("/welcome.jsp");  
 return;  
 } else {  
 *// 登录失败  
 // 如果不存在该用户, 提示错误信息, 并使用请求转发的方式回到登录页面* request.setAttribute("errorMsg", "用户名或密码错误.");  
 request.getRequestDispatcher("/login.jsp").forward(request, response);  
 return;  
 }  
 }  
}

public class UserService {  
 *// 登录业务逻辑* public User login(User user) {  
 UserDao userDao = new UserDao();  
 List<User> list = userDao.login(user);  
 *// 判断* if (list != null && list.isEmpty() == false) {  
 return list.get(0);  
 }  
 return null;  
 }  
}

public class UserDao {  
 *// 登录* public List<User> login(User user) {  
 JdbcTemplate jdbcTemplate = new JdbcTemplate(JDBCUtils.*getDataSource*());  
 String sql = "select *\** from user where username = ? and password = ?;";  
 Object[] params = { user.getUsername(), user.getPassword() };  
 List<User> list = jdbcTemplate.query(sql, new BeanPropertyRowMapper<>(User.class), params);  
 return list;  
 }  
}

**welcome.jsp 页面**

<%@ page import="cn.itcast.bean.User" %>  
<%@ page contentType="text/html;charset=UTF-8" language="java" %>  
<html>  
<head>  
 <title>Title</title>  
</head>  
<body>  
 <h1>欢迎, ${sessionScope.user.nickname}  
 <a href="/invalidate.jsp">注销</a>  
 </h1>  
</body>  
</html>

**invalidate.jsp 页面 :**

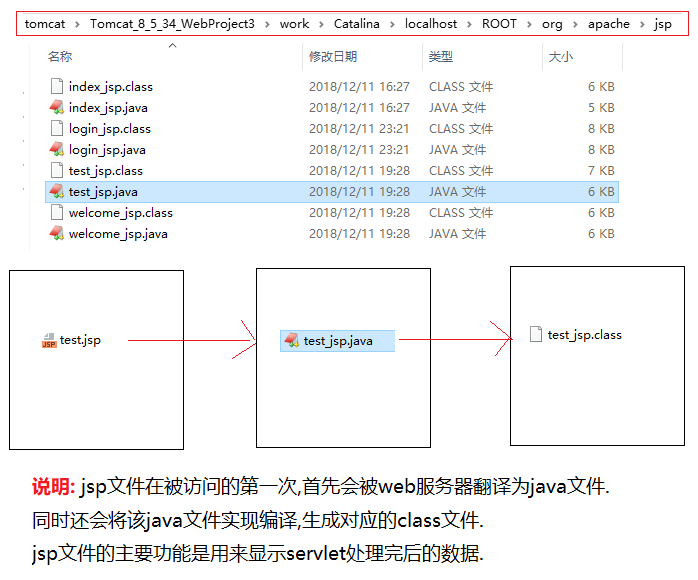
<%@ page contentType="text/html;charset=UTF-8" language="java" %>  
<html>  
<head>  
 <title>注销</title>  
</head>  
<body>  
 <%  
 *// 通知服务器销毁 session 容器* session.invalidate();  
 *// 从定向跳转到登录页面* response.sendRedirect("/login.jsp");  
 %>  
</body>  
</html>

**小结 :**

为什么我们需要书写 jsp 页面, 因为在 html 页面中是不能够书写 Java 代码的, jsp 页面即可以书写 html 代码, 同时也可以实现 Java 代码的书写. 从而实现数据的获取并在页面显示.

jsp 页面本质上就是一个 servlet 类.

# 62. jsp的三种脚本书写格式 :



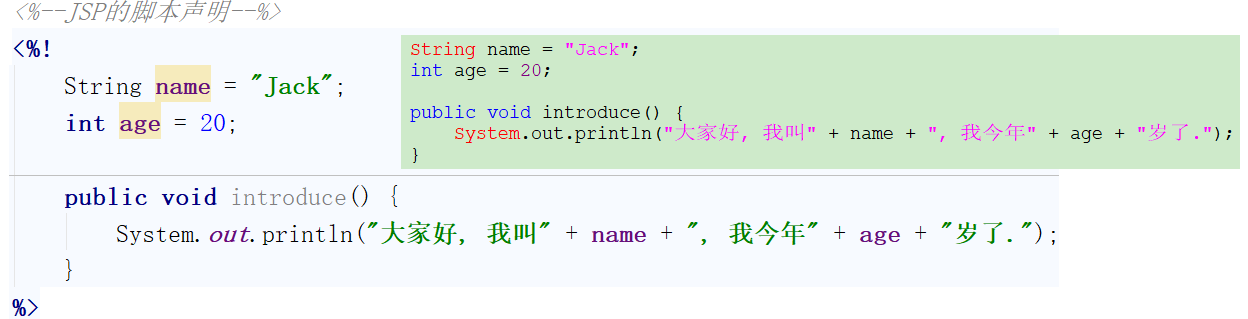
**1、JSP的脚本声明（定义）**

<%!

书写Java代码

%>

**脚本声明： 定义的java代码翻译完之后，相当于类中的成员或方法。**

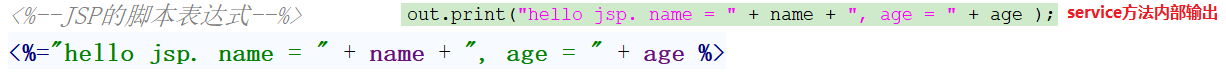


**2、JSP的脚本表达式**

作用：他被翻译后保存在java程序的service方法中，使用print方法输出到页面上.

<%=”hello world!”%>

**脚本表达式 : 代码会在页面上显示出来。**



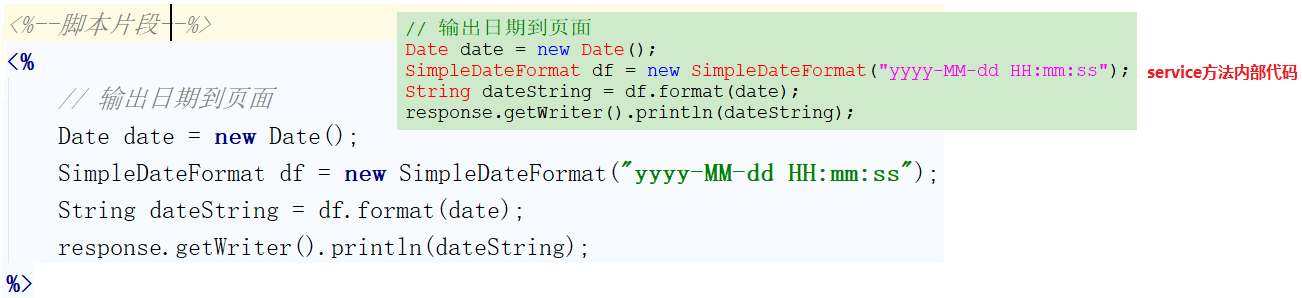
**3、脚本片段**

作用：他被翻译后保存在java程序的service方法中， 直接就是Java代码.

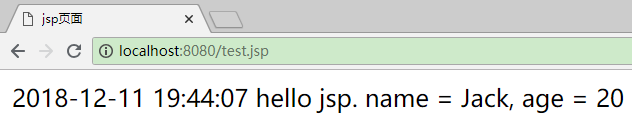
<%

书写Java代码

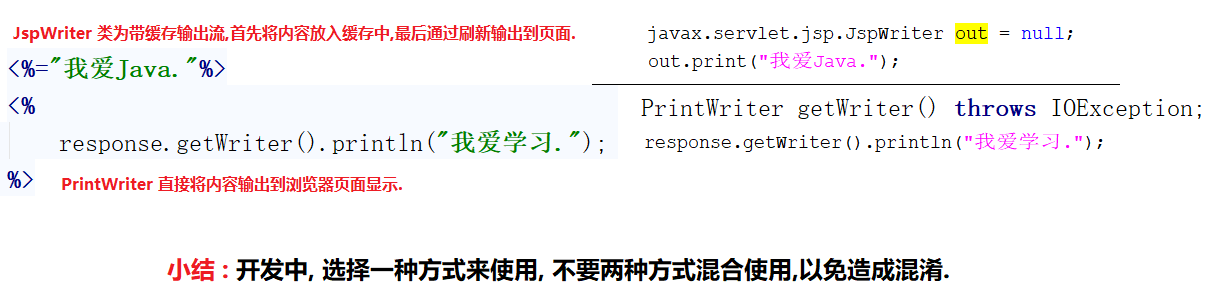
%>



**页面输出效果 :**



# 63. jsp输出方式的说明 :



# 64. jsp脚本完成登录页面信息输出 :

<%@ page contentType="text/html;charset=UTF-8" language="java" %>  
<html>  
<head>  
 <title>登录</title>  
  
 <script>  
 function *changeCode*(img) {  
 img.src = "/code?t=" + new *Date*().getTime();  
 }  
 </script>  
</head>  
<body>  
 <%!  
 String username;  
 %>  
 <%  
 Cookie[] cookies = request.getCookies();  
 if (cookies != null) {  
 for (Cookie cookie : cookies) {  
 if ("username".equals(cookie.getName())) {  
 username = cookie.getValue();  
 }  
 }  
 }  
 %>  
 *<%--登录失败的提示信息--%>  
 <%--<div style="color: red;">${requestScope.errorMsg }</div>--%>* <div style="color: red;"><%=request.getAttribute("errorMsg")%></div>  
 <form action="/login" method="post">  
 <table>  
 <tr>  
 <td><label for="username">用户名</label></td>  
 *<%--<td><input type="text" id="username" name="username" value="${cookie.username.value }"></td>--%>* <td><input type="text" id="username" name="username" value="<%=username%>"></td>  
 </tr>  
 <tr>  
 <td><label for="password">密码</label></td>  
 <td><input type="password" id="password" name="password"></td>  
 </tr>  
 <tr>  
 <td><label for="code">验证码</label></td>  
 <td>  
 <input type="text" id="code" name="code">  
 <img src="/code" alt="" id="验证码" style="cursor: pointer;" onclick="*changeCode*(this);">  
 </td>  
 </tr>  
 <tr>  
 <td></td>  
 <td><input type="submit" value="登录"></td>  
 </tr>  
 </table>  
 </form>  
  
</body>  
</html>

**说明 :** 使用 jsp 的脚本方式取出对应 request 或 cookie 中存储的数据, 代码实现还是较为复杂的. 另外, 如果使用 jsp 脚本实现, 没有数据时, 会显示 null 结果.



# 65. EL表达式获取域对象中的数据 :

**说明 :** 什么是EL表达式 ?

jsp中的EL（express language）表达式技术：它仅仅是表达式 。

EL表达式的功能：获取4个内置对象（域）中的数据，或自定义对象中的数据等。可以完成非常简单的运行。它不能完成循环、复杂的判断等功能。

EL表达式的书写格式：**${ 表达式 }**

在JSP中提供了EL表达式，可以快速的从不同的容器（page、request、session、ServletContext）中**取出**数据。

需求1: 通过el表达式 从指定域中获取数据 (比较和java代码的不同)

需求2: 通过el表达式 从不确定域中获取数据

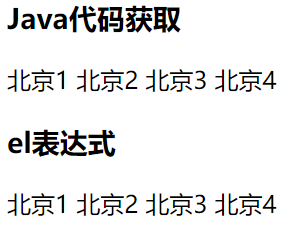
**格式1: ${xxxxScope.key}**

使用el表达式从不同的域中取出数据，需要使用el中的4个内置对象搞定

pageScope、requestScope、sessionScope、 applicationScope

在4个el内置对象xxxxScope取出保存的数据时，相当于使用pageContext.getAttribute(key,scope);取值

<%@ page contentType="text/html;charset=UTF-8" language="java" %>  
<html>  
<head>  
 <title>el表达式</title>  
</head>  
<body>  
 <%  
 *// 需求1: 通过el表达式 从指定域中获取数据 (比较和java代码的不同)* pageContext.setAttribute("city", "北京1", PageContext.*PAGE\_SCOPE*);  
 pageContext.setAttribute("city", "北京2", PageContext.*REQUEST\_SCOPE*);  
 pageContext.setAttribute("city", "北京3", PageContext.*SESSION\_SCOPE*);  
 pageContext.setAttribute("city", "北京4", PageContext.*APPLICATION\_SCOPE*);  
 %>  
  
 <h3>Java代码获取</h3>  
 <%=pageContext.getAttribute("city", PageContext.*PAGE\_SCOPE*)%>  
 <%=pageContext.getAttribute("city", PageContext.*REQUEST\_SCOPE*)%>  
 <%=pageContext.getAttribute("city", PageContext.*SESSION\_SCOPE*)%>  
 <%=pageContext.getAttribute("city", PageContext.*APPLICATION\_SCOPE*)%>  
  
 <h3>el表达式</h3>  
 ${pageScope.city }  
 ${requestScope.city }  
 ${sessionScope.city }  
 ${applicationScope.city }  
</body>  
</html>



**格式2: ${key}**

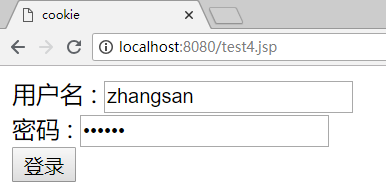
如果不知道数据在哪个范围中，这时可以不用指定范围直接书写key值即可 ${key}

下面代码的底层对应的是 pageContext.findAttribute(name);

<%@ page contentType="text/html;charset=UTF-8" language="java" %>  
<html>  
<head>  
 <title>el表达式</title>  
</head>  
<body>  
 <%  
 *// 需求2: 通过el表达式 从不确定域中获取数据* pageContext.setAttribute("city", "北京1", PageContext.*PAGE\_SCOPE*);  
 pageContext.setAttribute("city", "北京2", PageContext.*REQUEST\_SCOPE*);  
 pageContext.setAttribute("city", "北京3", PageContext.*SESSION\_SCOPE*);  
 pageContext.setAttribute("city", "北京4", PageContext.*APPLICATION\_SCOPE*);  
 %>  
  
 <h3>Java代码获取</h3>  
 <%=pageContext.findAttribute("city")%>  
  
 <h3>el表达式</h3>  
 ${city }  
</body>  
</html>

# 67. EL表达式从cookie中取值 :

**需求 :** 登录成功后, 账号与密码自动实现填充 :



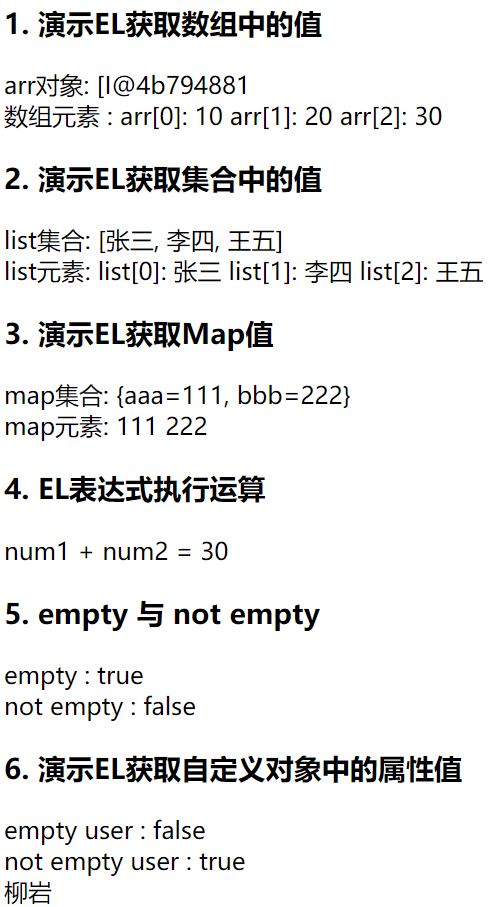
<%@ page contentType="text/html;charset=UTF-8" language="java" %>  
<html>  
<head>  
 <title>cookie</title>  
</head>  
<body>  
 <%--需求 : 免用户名与密码输入--%>  
 <%--EL 如何获取 cookie 中的数值 ???--%>  
 <%--cookie.key名称 : 根据 cookie 的 key 名称, 获取对应的 cookie.--%>  
 <%--在调用 value, 就可以获取对应 cookie 的数值--%>  
 <form action="/test4" method="post">  
 用户名 : <input type="text" name="username" value="${cookie.username.value }"> <br>  
 密码 : <input type="password" name="password" value="${cookie.password.value }"> <br>  
 <input type="submit" value="登录">  
 </form>  
</body>  
</html>

@WebServlet("/test4")  
public class Test4Servlet extends HttpServlet {  
 protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 doGet(request, response);  
 }  
  
 protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 *// 1. 获取用户名与密码* String username = request.getParameter("username");  
 String password = request.getParameter("password");  
 *// 2. 创建 cookie* Cookie usernameCookie = new Cookie("username", username);  
 Cookie passwordCookie = new Cookie("password", password);  
 *// 3. 将 cookie 响应到浏览器中* response.addCookie(usernameCookie);  
 response.addCookie(passwordCookie);  
 }  
}

# 68. EL表达式获取复杂数据 :

这里说的复杂数据是指：数组，集合，自定义对象。

<body>  
  
 <h3>1. 演示EL获取数组中的值</h3>  
 <h3>2. 演示EL获取集合中的值</h3>  
 <h3>3. 演示EL获取Map值</h3>  
 <h3>4. EL表达式执行运算</h3>  
 <h3>5. empty 与 not empty</h3>  
 <h3>6. 演示EL获取自定义对象中的属性值</h3>  
  
</body>



<%@ page contentType="text/html;charset=UTF-8" language="java" %>  
<html>  
<head>  
 <title>el获取复杂数据</title>  
</head>  
<body>  
  
 <h3>1. 演示EL获取数组中的值</h3>  
 <%  
 int[] arr = {10, 20, 30};  
 pageContext.setAttribute("arr", arr);  
 %>  
 arr对象: ${arr } <br>  
 数组元素 : arr[0]: ${arr[0] } arr[1]: ${arr[1] } arr[2]: ${arr[2] } <br>  
  
 <h3>2. 演示EL获取集合中的值</h3>  
 <%  
 ArrayList<String> list = new ArrayList<>();  
 list.add("张三");  
 list.add("李四");  
 list.add("王五");  
 pageContext.setAttribute("list", list);  
 %>  
 list集合: ${list } <br>  
 list元素: list[0]: ${list[0] } list[1]: ${list[1] } list[2]: ${list[2] }  
  
 <h3>3. 演示EL获取Map值</h3>  
 <%  
 HashMap<String, Integer> map = new HashMap<>();  
 map.put("aaa", 111);  
 map.put("bbb", 222);  
 pageContext.setAttribute("map", map);  
 %>  
 map集合: ${map } <br>  
 map元素: ${map.aaa } ${map.bbb }  
  
 <h3>4. EL表达式执行运算</h3>  
 <%  
 pageContext.setAttribute("num1", 10);  
 pageContext.setAttribute("num2", 20);  
 %>  
 num1 + num2 = ${num1 + num2 }  
  
 <h3>5. empty 与 not empty</h3>  
 <%  
 ArrayList<Integer> list2 = new ArrayList<>();  
 %>  
 empty : ${empty list2 } <br>  
 not empty : ${not empty list2}  
  
 <h3>6. 演示EL获取自定义对象中的属性值</h3>  
 <%  
 User user = new User();  
 user.setUsername("LiuYan");  
 user.setNickname("柳岩");  
 pageContext.setAttribute("user", user);  
 %>  
 empty user : ${empty user } <br>  
 not empty user : ${not empty user } <br>  
 ${not empty user ? user.nickname : "匿名登录" }  
  
</body>  
</html>

# 69. JSTL引入与介绍 :

**JSTL介绍**

在JSP页面中即可书写html，也可以书写Java代码，导致页面混乱，维护，修改，升级难度加大.

根据页面的需求将Java代码进行抽离封装，页面上需要的时候，通过标签（和书写HTML标签差不多）的方式引入封装的Java代码。

这套标签库被SUN公司定义为标准标签库（The JavaServer Pages Standard Tag Library），简称JSTL。

总结：JSTL就是一套封装Java代码的标签库，开发人员不用在JSP页面上书写Java代码，直接使用封装好的标签库，就能完成和书写Java代码相同的效果。



<%@ page contentType="text/html;charset=UTF-8" language="java" %>  
<html>  
<head>  
 <title>jstl</title>  
</head>  
<body>  
 <%  
 for (int i = 1; i <= 10; i++) {  
 %>  
 <button style="width: 60px; height: 30px; background-color: skyblue"><%=i%></button>  
 <%  
 }  
 %>  
</body>  
</html>

# 70. JSTL核心库if与out :

<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>  
<%@ page import="cn.itcast.bean.User" %>  
<%@ page contentType="text/html;charset=UTF-8" language="java" %>  
<html>  
<head>  
 <title>out与if</title>  
</head>  
<body>  
 <%  
 *// 向域中存储一个 User 对象* User user = null;  
 request.setAttribute("user", user);  
 %>  
 *<%--value 就是需要输出的具体信息--%>* <c:out value="${not empty user ? user.nickname : '匿名登录' }"></c:out>  
  
 *<%--test 就是 if 语句中的boolean类型判断条件--%>* <c:if test="${user == null}">  
 <c:out value="用户为空."></c:out>  
 </c:if>  
  
 <c:if test="${user != null}">  
 <c:out value="用户不为空."></c:out>  
 </c:if>  
</body>  
</html>



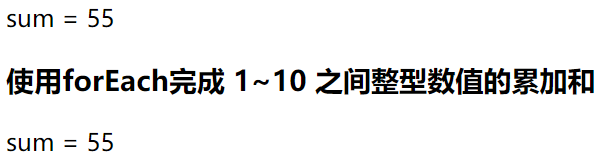
**说明 :** 使用 c:if 标签, 在jsp页面上可以完成 if 判断, 注意, 在 JSTL 核心库中没有 else. 只有 if.

# 71. JSTL核心库forEach与set :

<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>  
<%@ page import="java.util.ArrayList" %>  
<%@ page import="java.util.Collections" %>  
<%@ page import="java.util.HashMap" %>  
<%@ page contentType="text/html;charset=UTF-8" language="java" %>  
<html>  
<head>  
 <title>集合遍历</title>  
</head>  
<body>  
 <%  
 ArrayList<Integer> list = new ArrayList<>();  
 Collections.*addAll*(list, 10, 20, 30, 40, 50);  
 request.setAttribute("list", list);  
  
 *// Java 代码遍历 :* for (Integer num : list) {  
 response.getWriter().println(num);  
 }  
 %>  
  
 <h3>jstl : forEach 遍历 list 集合</h3>  
 <c:forEach var="num" items="${list }">  
 <c:out value="${num }"></c:out>  
 </c:forEach>  
  
 <h3>jstl : forEach 遍历 map 集合</h3>  
 <%  
 HashMap<String, String> map = new HashMap<>();  
 map.put("中国", "北京");  
 map.put("韩国", "首尔");  
 map.put("泰国", "曼谷");  
 request.setAttribute("map", map);  
 %>  
 <c:forEach var="entry" items="${map }">  
 <c:out value="key = ${entry.key }"></c:out>  
 <c:out value="value = ${entry.value }"></c:out>  
 </c:forEach>  
  
</body>  
</html>

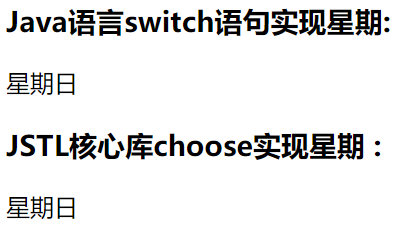


<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>  
<%@ page contentType="text/html;charset=UTF-8" language="java" %>  
<html>  
<head>  
 <title>JSTL核心类</title>  
</head>  
<body>  
  
 <%  
 *// Java 代码实现 :* int sum = 0;  
 for (int i = 1; i <= 10; i++) {  
 sum += i;  
 }  
 response.getWriter().println("sum = " + sum);  
 %>  
  
 <h3>使用forEach完成 1~10 之间整型数值的累加和</h3>  
 <c:set var="sum" value="0"></c:set>  
 <c:forEach var="i" begin="1" end="10" step="1">  
 <c:set var="sum" value="${sum + i}"></c:set>  
 </c:forEach>  
 <c:out value="sum = ${sum }"></c:out>  
</body>  
</html>



# 72. JSTL核心库choose :

<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>  
<%@ page contentType="text/html;charset=UTF-8" language="java" %>  
<html>  
<head>  
 <title>choose标签</title>  
</head>  
<body>  
 <h3>Java语言switch语句实现星期:</h3>  
 <%  
 int day = 7;  
 switch (day) {  
 case 1:  
 out.println("星期一");  
 break;  
 case 2:  
 out.println("星期二");  
 break;  
 case 3:  
 out.println("星期三");  
 break;  
 case 4:  
 out.println("星期四");  
 break;  
 case 5:  
 out.println("星期五");  
 break;  
 case 6:  
 out.println("星期六");  
 break;  
 case 7:  
 out.println("星期日");  
 break;  
 default:  
 out.println("没有对应的星期.");  
 break;  
 }  
 %>  
  
 <h3>JSTL核心库choose实现星期 :</h3>  
 <c:set var="day" value="7"></c:set>  
 <c:choose>  
 <c:when test="${day == 1 }">星期一</c:when>  
 <c:when test="${day == 2 }">星期二</c:when>  
 <c:when test="${day == 3 }">星期三</c:when>  
 <c:when test="${day == 4 }">星期四</c:when>  
 <c:when test="${day == 5 }">星期五</c:when>  
 <c:when test="${day == 6 }">星期六</c:when>  
 <c:when test="${day == 7 }">星期日</c:when>  
 <c:otherwise>没有对应的星期.</c:otherwise>  
 </c:choose>  
  
</body>  
</html>



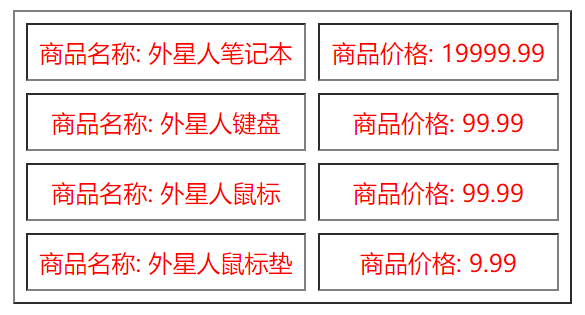
# 73. servlet与jsp练习 :



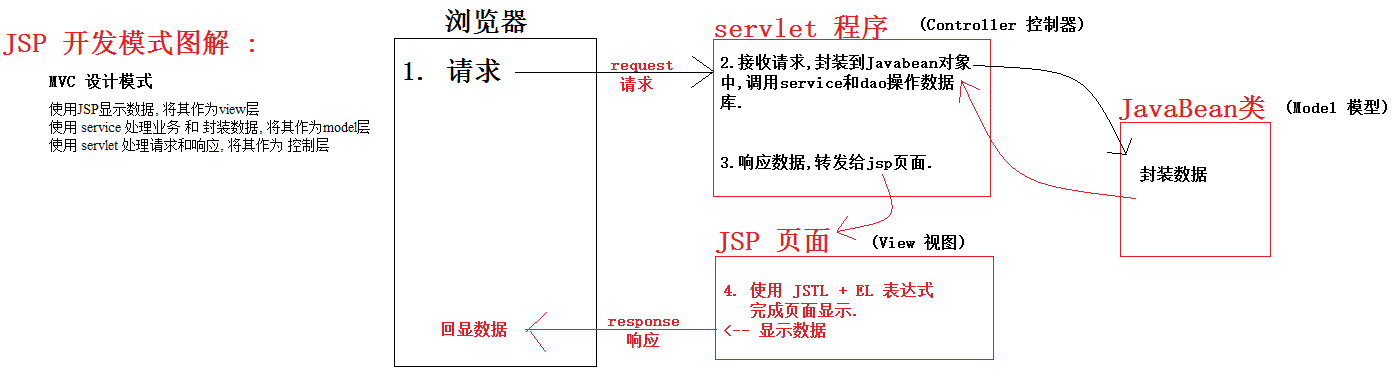
public class Product {  
 private String pname;  
 private String price;  
  
 public Product(String pname, String price) {  
 this.pname = pname;  
 this.price = price;  
 }  
  
 public Product() {  
 }  
  
 @Override  
 public String toString() {  
 return "Product{" +  
 "pname='" + pname + '\'' +  
 ", price='" + price + '\'' +  
 '}';  
 }  
  
 public String getPname() {  
 return pname;  
 }  
  
 public void setPname(String pname) {  
 this.pname = pname;  
 }  
  
 public String getPrice() {  
 return price;  
 }  
  
 public void setPrice(String price) {  
 this.price = price;  
 }  
}

@WebServlet("/product")  
public class ProductServlet extends HttpServlet {  
 protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 doGet(request, response);  
 }  
  
 protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 *// 1. 获取数据  
 // 2. 调用业务层获取处理结果* List<Product> products = new ArrayList<Product>();  
 products.add(new Product("外星人笔记本", "19999.99"));  
 products.add(new Product("外星人键盘", "99.99"));  
 products.add(new Product("外星人鼠标", "99.99"));  
 products.add(new Product("外星人鼠标垫", "9.99"));  
  
 *// 3. 将处理结果存储到 request 域中, 并实现请求转发到 jsp 页面* request.setAttribute("products", products);  
 request.getRequestDispatcher("/list.jsp").forward(request, response);  
 return;  
 }  
}

<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>  
<%@ page contentType="text/html;charset=UTF-8" language="java" %>  
<html>  
<head>  
 <title>展示商品</title>  
  
 <style>  
 td {  
 color: red;  
 text-align: center;  
 }  
 </style>  
  
</head>  
<body>  
  
 <table border="1px" cellspacing="8px" cellpadding="8px" align="center">  
 *<%--使用 el + jstl 对域容器中的数据实现遍历与获取显示--%>* <c:forEach var="product" items="${products }">  
 <tr>  
 <td>商品名称: ${product.pname }</td>  
 <td>商品价格: ${product.price }</td>  
 </tr>  
 </c:forEach>  
 </table>  
  
</body>  
</html>



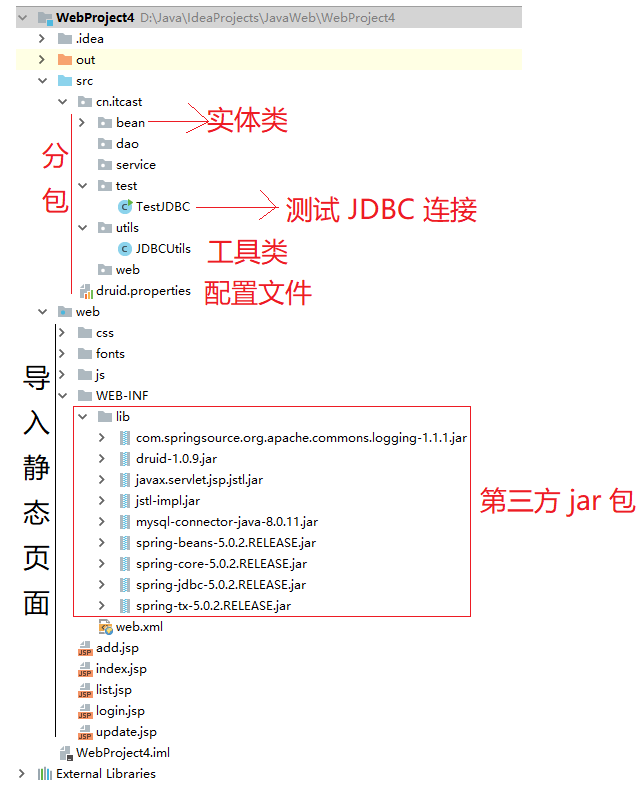
# JSP开发模式图解 :



# 联系人员名单增删改查:

## 准备工作 :

**注意:** 创建一个项目.



**数据库脚本 :**

-- --------------------------------------------------------

-- 主机: 127.0.0.1

-- 服务器版本: 5.5.27 - MySQL Community Server (GPL)

-- 服务器操作系统: Win64

-- HeidiSQL 版本: 8.2.0.4675

-- --------------------------------------------------------

/\*!40101 SET @OLD\_CHARACTER\_SET\_CLIENT=@@CHARACTER\_SET\_CLIENT \*/;

/\*!40101 SET NAMES utf8 \*/;

/\*!40014 SET @OLD\_FOREIGN\_KEY\_CHECKS=@@FOREIGN\_KEY\_CHECKS, FOREIGN\_KEY\_CHECKS=0 \*/;

/\*!40101 SET @OLD\_SQL\_MODE=@@SQL\_MODE, SQL\_MODE='NO\_AUTO\_VALUE\_ON\_ZERO' \*/;

-- 导出 day18 的数据库结构

CREATE DATABASE IF NOT EXISTS `day18` /\*!40100 DEFAULT CHARACTER SET utf8 \*/;

USE `day18`;

-- 导出 表 day18.contact 结构

CREATE TABLE IF NOT EXISTS `contact` (

`id` int(11) NOT NULL AUTO\_INCREMENT COMMENT '编号',

`name` varchar(50) DEFAULT NULL COMMENT '姓名',

`sex` varchar(50) DEFAULT NULL COMMENT '性别',

`age` int(11) DEFAULT NULL COMMENT '年龄',

`address` varchar(50) DEFAULT NULL COMMENT '籍贯',

`qq` varchar(50) DEFAULT NULL COMMENT 'QQ号',

`email` varchar(50) DEFAULT NULL COMMENT '邮箱',

PRIMARY KEY (`id`)

) ENGINE=InnoDB AUTO\_INCREMENT=11 DEFAULT CHARSET=utf8 COMMENT='联系人表';

-- 正在导出表 day18.contact 的数据：~0 rows (大约)

/\*!40000 ALTER TABLE `contact` DISABLE KEYS \*/;

INSERT INTO `contact` (`id`, `name`, `sex`, `age`, `address`, `qq`, `email`) VALUES

(1, '张三', '男', 11, '广东', '766335435', '766335435@qq.com'),

(2, '李四', '男', 12, '广东', '243424242', '243424242@qq.com'),

(3, '王五', '女', 13, '广东', '474574574', '474574574@qq.com'),

(4, '赵六', '女', 18, '广东', '77777777', '77777777@qq.com'),

(5, '钱七', '女', 15, '湖南', '412132145', '412132145@qq.com');

/\*!40000 ALTER TABLE `contact` ENABLE KEYS \*/;

/\*!40101 SET SQL\_MODE=IFNULL(@OLD\_SQL\_MODE, '') \*/;

/\*!40014 SET FOREIGN\_KEY\_CHECKS=IF(@OLD\_FOREIGN\_KEY\_CHECKS IS NULL, 1, @OLD\_FOREIGN\_KEY\_CHECKS) \*/;

/\*!40101 SET CHARACTER\_SET\_CLIENT=@OLD\_CHARACTER\_SET\_CLIENT \*/;

**数据库配置文件 :**

driverClassName=com.mysql.cj.jdbc.Driver  
url=jdbc:mysql://localhost:3306/day18?serverTimezone=UTC&characterEncoding=utf-8  
username=root  
password=123

**数据库工具类 :**

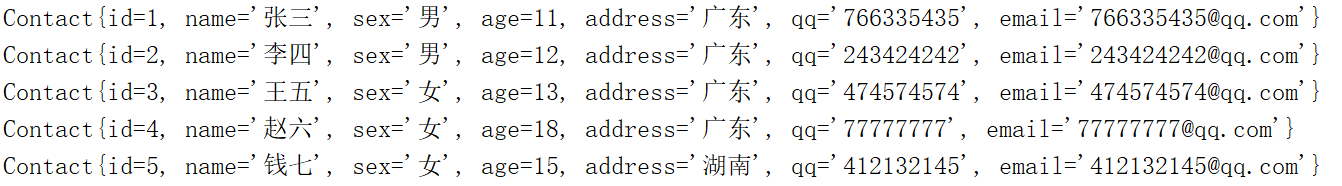
import com.alibaba.druid.pool.DruidDataSourceFactory;  
  
import javax.sql.DataSource;  
import java.io.InputStream;  
import java.sql.Connection;  
import java.sql.ResultSet;  
import java.sql.SQLException;  
import java.sql.Statement;  
import java.util.Properties;  
  
public class JDBCUtils {  
  
 private static DataSource *dataSource*;  
  
 static {  
  
 *// Java 项目 src 目录下的配置文件  
 // Properties prop = new Properties();  
 // prop.load(new FileReader("src/druid.properties"));  
  
 // tomcat 服务器 src 目录下的配置文件* InputStream in = JDBCUtils.class.getClassLoader().getResourceAsStream("druid.properties");  
 Properties prop = new Properties();  
  
 try {  
 prop.load(in);  
  
 *dataSource* = DruidDataSourceFactory.*createDataSource*(prop);  
 } catch (Exception e) {  
 throw new RuntimeException("配置文件加载失败.");  
 }  
 }  
  
 public static DataSource getDataSource() {  
 return *dataSource*;  
 }  
  
 public static void release(Connection conn, Statement stmt, ResultSet rs) {  
 if (rs != null) {  
 try {  
 rs.close();  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 rs = null;  
 }  
  
 *release*(conn, stmt);  
 }  
  
 public static void release(Connection conn, Statement stmt) {  
 if (stmt != null) {  
 try {  
 stmt.close();  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 stmt = null;  
 }  
 if (conn != null) {  
 try {  
 conn.close();  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 conn = null;  
 }  
 }  
}

**实体类 :**

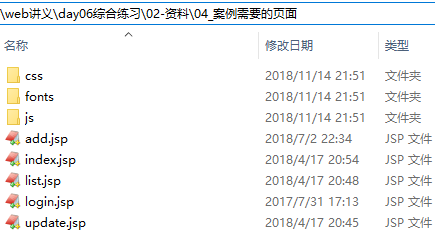
public class Contact {  
 private int id;  
 private String name;  
 private String sex;  
 private int age;  
 private String address;  
 private String qq;  
 private String email;  
  
 @Override  
 public String toString() {  
 return "Contact{" +  
 "id=" + id +  
 ", name='" + name + '\'' +  
 ", sex='" + sex + '\'' +  
 ", age=" + age +  
 ", address='" + address + '\'' +  
 ", qq='" + qq + '\'' +  
 ", email='" + email + '\'' +  
 '}';  
 }  
  
 public int getId() {  
 return id;  
 }  
  
 public void setId(int id) {  
 this.id = id;  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 public void setName(String name) {  
 this.name = name;  
 }  
  
 public String getSex() {  
 return sex;  
 }  
  
 public void setSex(String sex) {  
 this.sex = sex;  
 }  
  
 public int getAge() {  
 return age;  
 }  
  
 public void setAge(int age) {  
 this.age = age;  
 }  
  
 public String getAddress() {  
 return address;  
 }  
  
 public void setAddress(String address) {  
 this.address = address;  
 }  
  
 public String getQq() {  
 return qq;  
 }  
  
 public void setQq(String qq) {  
 this.qq = qq;  
 }  
  
 public String getEmail() {  
 return email;  
 }  
  
 public void setEmail(String email) {  
 this.email = email;  
 }  
}

**JDBC 测试类 :**

import cn.itcast.bean.Contact;  
import cn.itcast.utils.JDBCUtils;  
import org.springframework.jdbc.core.BeanPropertyRowMapper;  
import org.springframework.jdbc.core.JdbcTemplate;  
  
import java.util.List;  
  
public class TestJDBC {  
 public static void main(String[] args) {  
 JdbcTemplate jdbcTemplate = new JdbcTemplate(JDBCUtils.*getDataSource*());  
 String sql = "select *\** from contact;";  
 List<Contact> list = jdbcTemplate.query(sql, new BeanPropertyRowMapper<>(Contact.class));  
 for (Contact contact : list) {  
 System.*out*.println(contact);  
 }  
 }  
}



**导入页面静态文件 :**

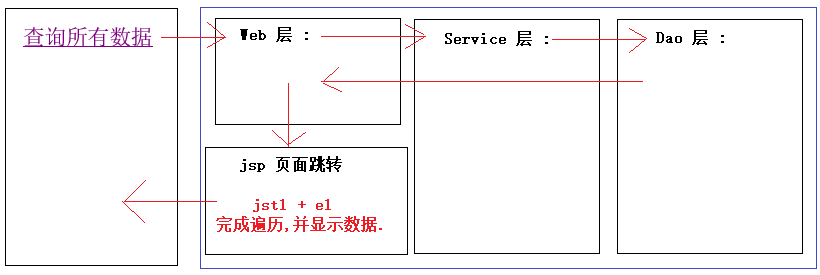


**启动 tomcat 服务器 :**





## 显示所有联系人 :



**index.jsp 页面 :**

<%@ page language="java" import="java.util.\*" pageEncoding="UTF-8"%>  
<%  
String path = request.getContextPath();  
String basePath = request.getScheme()+"://"+request.getServerName()+":"+request.getServerPort()+path+"/";  
%>  
  
<!DOCTYPE html>   
<html lang="zh-CN">  
 <head>  
 <base href="<%=basePath%>"/>  
 <meta charset="utf-8"/>  
 <meta http-equiv="X-UA-Compatible" content="IE=edge"/>  
 <meta name="viewport" content="width=device-width, initial-scale=1"/>  
 <title>首页</title>  
  
 *<!-- 1. 导入CSS的全局样式 -->* <link href="css/bootstrap.min.css" rel="stylesheet">  
 *<!-- 2. jQuery导入，建议使用1.9以上的版本 -->* <script src="js/jquery-2.1.0.min.js"></script>  
 *<!-- 3. 导入bootstrap的js文件 -->* <script src="js/bootstrap.min.js"></script>  
 <script type="text/javascript">  
 </script>  
 </head>  
 <body>  
 <div align="center">  
 <a  
 href="/findAll" style="text-decoration:none;font-size:33px">查询所有用户信息  
 </a>  
 </div>  
 </body>  
</html>

**ContactFindAllServlet.java**

package cn.itcast.web;  
  
import cn.itcast.bean.Contact;  
import cn.itcast.service.ContactService;  
  
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.io.IOException;  
import java.util.List;  
  
@WebServlet("/findAll")  
public class ContactFindAllServlet extends HttpServlet {  
 protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 doGet(request, response);  
 }  
  
 protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 *// 1. 编码* request.setCharacterEncoding("utf-8");  
 *// response.setHeader("content-type", "text/html;charset=utf-8");* response.setContentType("text/html;charset=utf-8");  
  
 *// 2. 获取数据  
  
 // 3. 处理数据, 调用 service 层* ContactService contactService = new ContactService();  
 List<Contact> contactList = contactService.findAll();  
  
 *// 4. 响应数据* System.*out*.println("contactList = " + contactList.size());  
  
 *// 4.1 将数据保存到 request 中* request.setAttribute("contactList", contactList);  
 *// 4.2 请求转发, 使用 list.jsp 页面显示返回的所有数据* request.getRequestDispatcher("/list.jsp").forward(request, response);  
 }  
}

**ContactService.java**

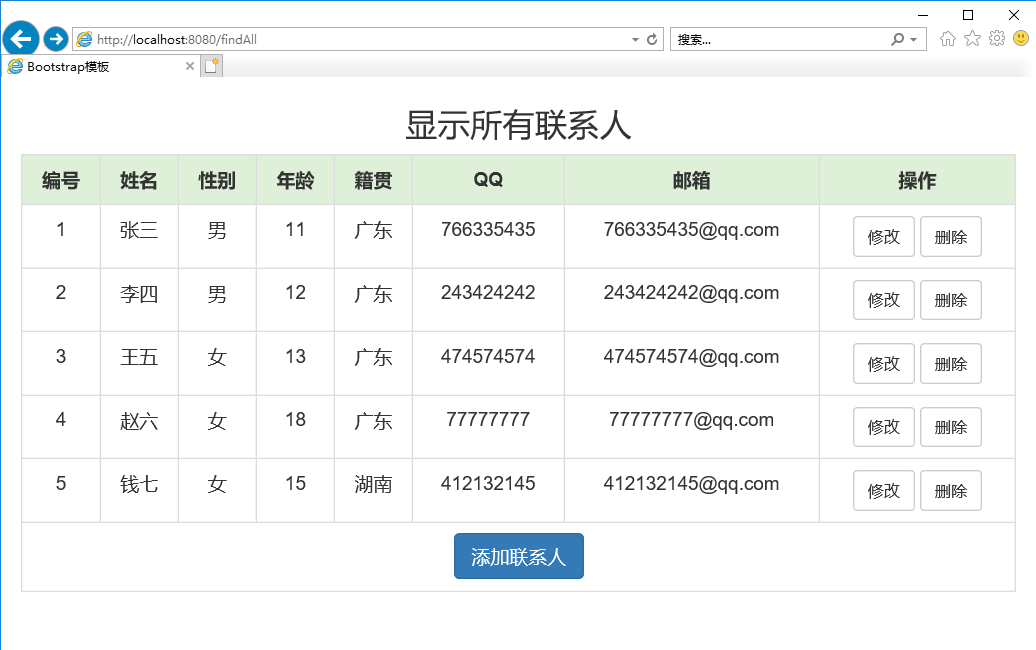
package cn.itcast.service;  
  
import cn.itcast.bean.Contact;  
import cn.itcast.dao.ContactDao;  
  
import java.util.List;  
  
public class ContactService {  
 public List<Contact> findAll() {  
 ContactDao contactDao = new ContactDao();  
 List<Contact> list = contactDao.findAll();  
 return list;  
 }  
}

**ContactDao.java**

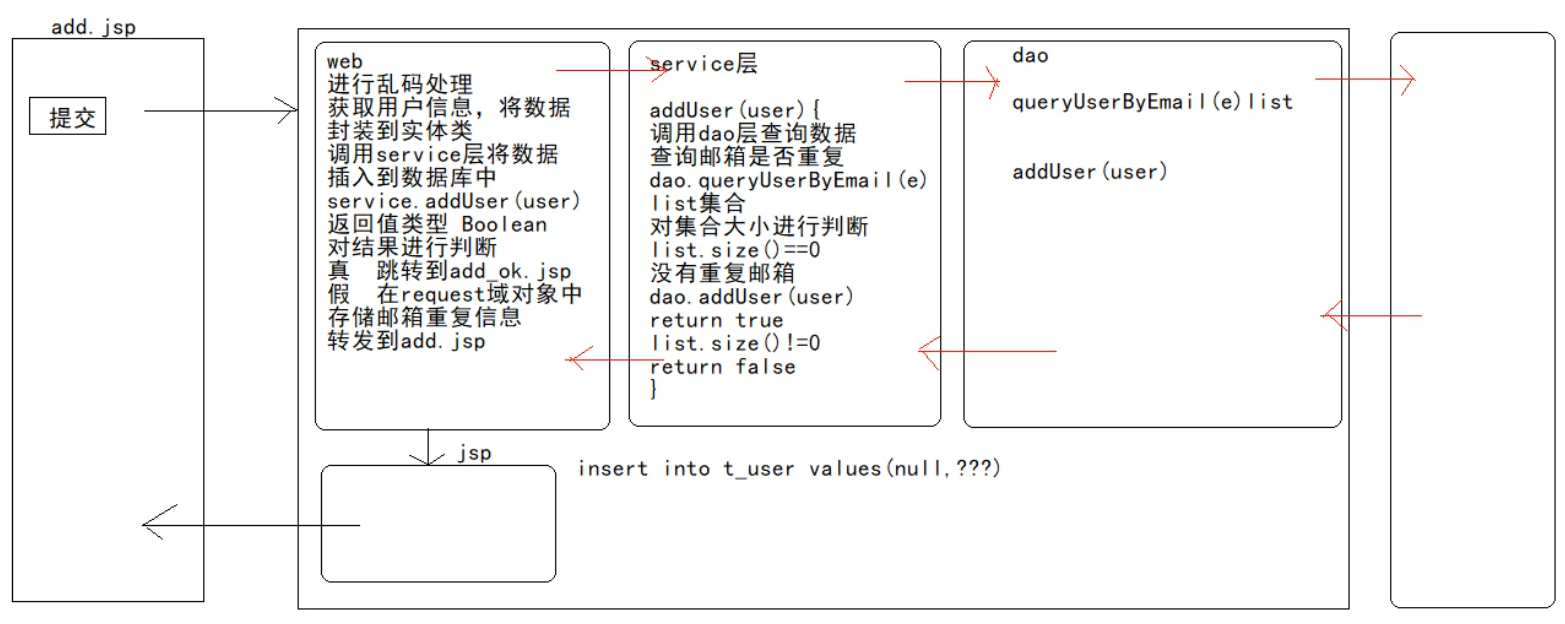
package cn.itcast.dao;  
  
import cn.itcast.bean.Contact;  
import cn.itcast.utils.JDBCUtils;  
import org.springframework.jdbc.core.BeanPropertyRowMapper;  
import org.springframework.jdbc.core.JdbcTemplate;  
  
import java.util.List;  
  
public class ContactDao {  
 public List<Contact> findAll() {  
 JdbcTemplate jdbcTemplate = new JdbcTemplate(JDBCUtils.*getDataSource*());  
 List<Contact> list = jdbcTemplate.query("select *\** from contact;", new BeanPropertyRowMapper<>(Contact.class));  
 return list;  
 }  
}

**list.jsp 页面 :**

<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>  
<%@ page contentType="text/html;charset=UTF-8" language="java" %>  
<!DOCTYPE html>  
*<!-- 网页使用的语言 -->*<html lang="zh-CN">  
<head>  
 *<!-- 指定字符集 -->* <meta charset="utf-8">  
 *<!-- 使用Edge最新的浏览器的渲染方式 -->* <meta http-equiv="X-UA-Compatible" content="IE=edge">  
 *<!-- viewport视口：网页可以根据设置的宽度自动进行适配，在浏览器的内部虚拟一个容器，容器的宽度与设备的宽度相同。  
 width: 默认宽度与设备的宽度相同  
 initial-scale: 初始的缩放比，为1:1 -->* <meta name="viewport" content="width=device-width, initial-scale=1">  
 *<!-- 上述3个meta标签\*必须\*放在最前面，任何其他内容都\*必须\*跟随其后！ -->* <title>用户信息管理系统</title>  
  
 *<!-- 1. 导入CSS的全局样式 -->* <link href="css/bootstrap.min.css" rel="stylesheet">  
 *<!-- 2. jQuery导入，建议使用1.9以上的版本 -->* <script src="js/jquery-2.1.0.min.js"></script>  
 *<!-- 3. 导入bootstrap的js文件 -->* <script src="js/bootstrap.min.js"></script>  
 <style type="text/css">  
 td, th {  
 text-align: center;  
 }  
 </style>  
</head>  
<body>  
<div class="container">  
 <h3 style="text-align: center">用户信息列表</h3>  
 <table border="1" class="table table-bordered table-hover">  
 <tr class="success">  
 <th>编号</th>  
 <th>姓名</th>  
 <th>性别</th>  
 <th>年龄</th>  
 <th>籍贯</th>  
 <th>QQ</th>  
 <th>邮箱</th>  
 <th>操作</th>  
 </tr>  
 <c:forEach items="${contactList }" var="contact">  
 <tr>  
 <td>${contact.id}</td>  
 <td>${contact.name}</td>  
 <td>${contact.sex}</td>  
 <td>${contact.age}</td>  
 <td>${contact.address}</td>  
 <td>${contact.qq}</td>  
 <td>${contact.email}</td>  
 <td><a class="btn btn-default btn-sm" href="update.jsp">修改</a>&nbsp;<a class="btn btn-default btn-sm" href="修改联系人.html">删除</a></td>  
 </tr>  
 </c:forEach>  
 <tr>  
 <td colspan="8" align="center"><a class="btn btn-primary" href="add.jsp">添加联系人</a></td>  
 </tr>  
 </table>  
</div>  
</body>  
</html>



## 添加联系人 :



add.jsp 页面 :

<%@ page contentType="text/html;charset=UTF-8" language="java" %>  
*<!-- HTML5文档-->*<!DOCTYPE html>  
*<!-- 网页使用的语言 -->*<html lang="zh-CN">  
<head>  
 *<!-- 指定字符集 -->* <meta charset="utf-8">  
 *<!-- 使用Edge最新的浏览器的渲染方式 -->* <meta http-equiv="X-UA-Compatible" content="IE=edge">  
 *<!-- viewport视口：网页可以根据设置的宽度自动进行适配，在浏览器的内部虚拟一个容器，容器的宽度与设备的宽度相同。  
 width: 默认宽度与设备的宽度相同  
 initial-scale: 初始的缩放比，为1:1 -->* <meta name="viewport" content="width=device-width, initial-scale=1">  
 *<!-- 上述3个meta标签\*必须\*放在最前面，任何其他内容都\*必须\*跟随其后！ -->* <title>添加用户</title>  
  
 *<!-- 1. 导入CSS的全局样式 -->* <link href="css/bootstrap.min.css" rel="stylesheet">  
 *<!-- 2. jQuery导入，建议使用1.9以上的版本 -->* <script src="js/jquery-2.1.0.min.js"></script>  
 *<!-- 3. 导入bootstrap的js文件 -->* <script src="js/bootstrap.min.js"></script>  
</head>  
<body>  
<div class="container text-left">  
 <center><h3>添加联系人页面</h3></center>  
 <form action="/addUser" method="post" class="form-horizontal">  
 <div class="form-group">  
 <label class="col-lg-2 control-label" for="name">姓名：</label>  
 <div class="col-lg-8">  
 <input type="text" class="form-control" id="name" name="name" placeholder="请输入姓名">  
 </div>  
 <label class="col-lg-2 control-label" id="nameMsg"></label>  
 </div>  
  
 <div class="form-group">  
 <label class="col-lg-2 control-label">性别：</label>  
 <div class="col-lg-3">  
 <input type="radio" name="sex" value="男" checked="checked"/>男  
 </div>  
 <div class="col-lg-3">  
 <input type="radio" name="sex" value="女"/>女  
 </div>  
 <label class="col-lg-2 control-label" id="sexMsg"></label>  
 </div>  
  
 <div class="form-group">  
 <label for="age" class="control-label col-lg-2">年龄：</label>  
 <div class="col-lg-8">  
 <input type="text" class="form-control" id="age" name="age" placeholder="请输入年龄">  
 </div>  
 </div>  
  
 <div class="form-group">  
 <label for="jiguan" class="col-lg-2 control-label">籍贯：</label>  
 <div class="col-lg-8">  
 <select name="address" class="form-control" id="jiguan">  
 <option value="广东">广东</option>  
 <option value="广西">广西</option>  
 <option value="湖南">湖南</option>  
 </select>  
 </div>  
 </div>  
  
 <div class="form-group">  
 <label for="qq" class="control-label col-lg-2">QQ：</label>  
 <div class="col-lg-8">  
 <input type="text" class="form-control" id="qq" name="qq" placeholder="请输入QQ号码"/>  
 </div>  
 </div>  
  
 <div class="form-group">  
 <label for="email" class="col-lg-2 control-label">Email：</label>  
 <div class="col-lg-8">  
 <span style="color: red;">${msg }</span>  
 <input type="text" class="form-control" id="email" name="email" placeholder="请输入邮箱地址"/>  
 </div>  
 </div>  
  
 <div class="form-group" style="text-align: center">  
 <input class="btn btn-primary" type="submit" value="提交" />  
 <input class="btn btn-default" type="reset" value="重置" />  
 <input class="btn btn-default" type="button" value="返回" />  
 </div>  
 </form>  
</div>  
</body>  
</html>

@WebServlet("/addUser")  
public class AddUserServlet extends HttpServlet {  
 protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 doGet(request, response);  
 }  
  
 protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 *// 1. 请求与响应乱码处理* request.setCharacterEncoding("utf-8");  
 response.setContentType("text/html;charset=utf-8");  
 *// 2. 使用 BeanUtils 工具类, 将 map 中的请求数据, 封装到实体类中存储* Contact contact = new Contact();  
 try {  
 BeanUtils.*populate*(contact, request.getParameterMap());  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 *// System.out.println("contact = " + contact);  
 // 3. 调用 service 层添加用户业务逻辑* ContactService contactService = new ContactService();  
 boolean result = contactService.addContact(contact);  
 *// 4. 将结果实现判断* if (result == true) {  
 *// 4.1 添加用户成功, 跳转到 add\_ok.jsp 页面* response.sendRedirect("add\_ok.jsp");  
 } else {  
 *// 4.2 将邮箱重复信息告诉用户, 将错误信息存储到 request 域中, 然后实现请求转发* request.setAttribute("msg", "邮箱重复");  
 request.getRequestDispatcher("/add.jsp").forward(request, response);  
 }  
 }  
}



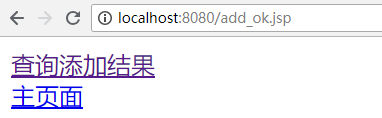
public class ContactService {  
  
 ContactDao contactDao = new ContactDao();  
  
 public List<Contact> findAll() {  
 List<Contact> list = contactDao.findAll();  
 return list;  
 }  
  
 *// 添加用户信息, 先查(邮箱是否重复) 后插* public boolean addContact(Contact contact) {  
 *// 1. 查询邮箱是否重复* List<Contact> list = contactDao.queryContactByEmail(contact.getEmail());  
 *// 2. 对集合大小实现判断* if (list == null || list.size() == 0) {  
 *// 2.1 数据库中没有重复邮箱地址, 插入数据* contactDao.addContact(contact);  
 return true;  
 } else {  
 *// 2.2 有重复邮箱, 插入失败* return false;  
 }  
 }  
}

public class ContactDao {  
  
 JdbcTemplate jdbcTemplate = new JdbcTemplate(JDBCUtils.*getDataSource*());  
  
 public List<Contact> findAll() {  
 List<Contact> list = jdbcTemplate.query("select *\** from contact;", new BeanPropertyRowMapper<>(Contact.class));  
 return list;  
 }  
  
 *// 根据邮箱查询用户信息* public List<Contact> queryContactByEmail(String email) {  
 String sql = "select *\** from contact where email = ?;";  
 List<Contact> list = jdbcTemplate.query(sql, new BeanPropertyRowMapper<>(Contact.class), email);  
 return list;  
 }  
  
 *// 添加用户* public void addContact(Contact contact) {  
 String sql = "insert into contact values(null, ?, ?, ?, ?, ?, ?);";  
 Object[] params = { contact.getName(), contact.getSex(), contact.getAge(), contact.getAddress(), contact.getQq(), contact.getEmail() };  
 jdbcTemplate.update(sql, params);  
 }  
}

**add\_ok.jsp 页面 :**

<%@ page contentType="text/html;charset=UTF-8" language="java" %>  
<html>  
<head>  
 <title>添加成功</title>  
</head>  
<body>  
 <a href="/findAll">查询添加结果</a> <br>  
 <a href="index.jsp">主页面</a>  
</body>  
</html>





## 添加页面数据回显:

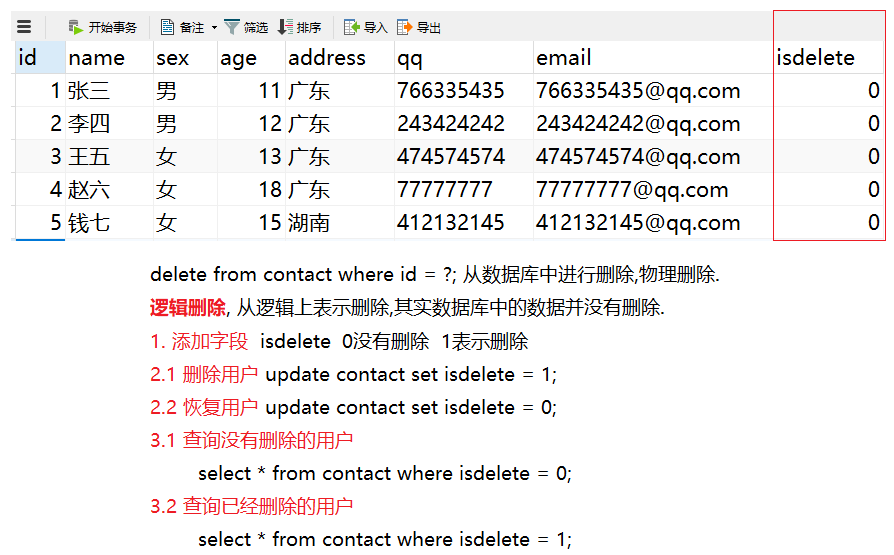


@WebServlet("/addUser")  
public class AddUserServlet extends HttpServlet {  
 protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 doGet(request, response);  
 }  
  
 protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 *// 1. 请求与响应乱码处理* request.setCharacterEncoding("utf-8");  
 response.setContentType("text/html;charset=utf-8");  
 *// 2. 使用 BeanUtils 工具类, 将 map 中的请求数据, 封装到实体类中存储* Contact contact = new Contact();  
 try {  
 BeanUtils.*populate*(contact, request.getParameterMap());  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 *// System.out.println("contact = " + contact);  
 // 3. 调用 service 层添加用户业务逻辑* ContactService contactService = new ContactService();  
 boolean result = contactService.addContact(contact);  
 *// 4. 将结果实现判断* if (result == true) {  
 *// 4.1 添加用户成功, 跳转到 add\_ok.jsp 页面* response.sendRedirect("add\_ok.jsp");  
 } else {  
 *// 4.2 将邮箱重复信息告诉用户, 将错误信息存储到 request 域中, 然后实现请求转发* request.setAttribute("msg", "邮箱重复");  
 request.setAttribute("contact", contact);  
 request.getRequestDispatcher("/add.jsp").forward(request, response);  
 }  
 }  
}

**add.jsp 页面**

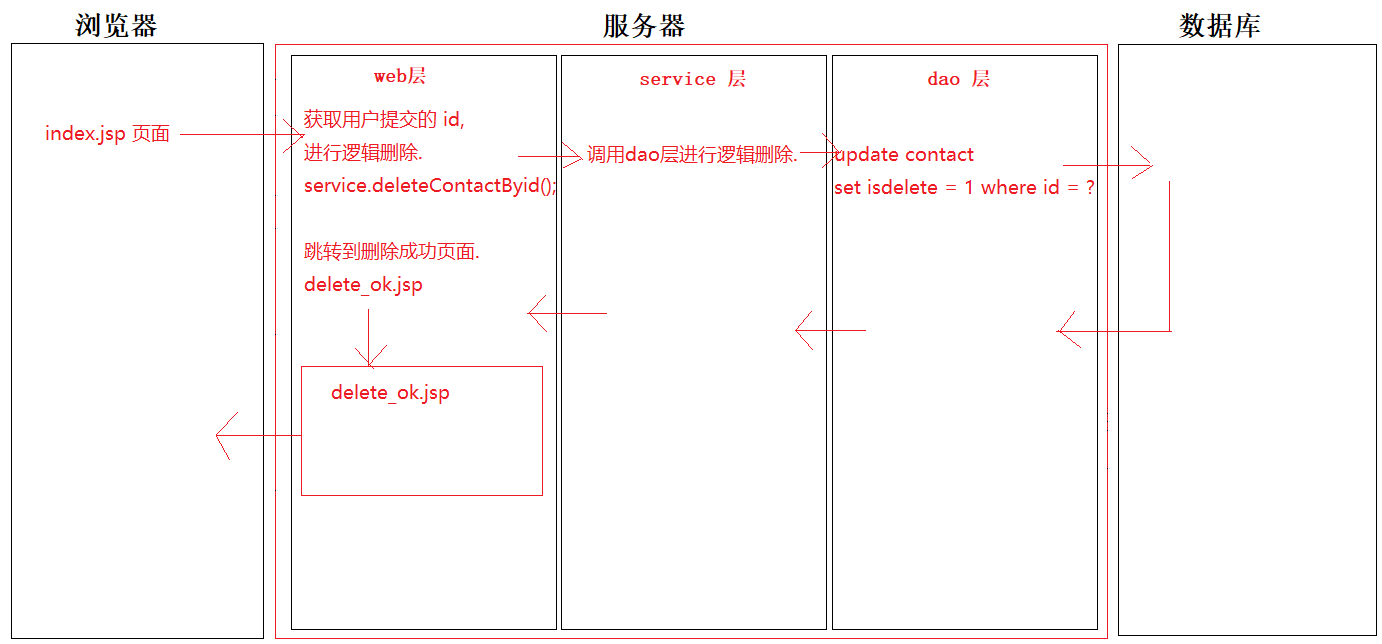
<body>  
<div class="container text-left">  
 <center><h3>添加联系人页面</h3></center>  
 <form action="/addUser" method="post" class="form-horizontal">  
 <div class="form-group">  
 <label class="col-lg-2 control-label" for="name">姓名：</label>  
 <div class="col-lg-8">  
 <input type="text" class="form-control" id="name" name="name" placeholder="请输入姓名" value="${contact.name }">  
 </div>  
 <label class="col-lg-2 control-label" id="nameMsg"></label>  
 </div>  
  
 <div class="form-group">  
 <label class="col-lg-2 control-label">性别：</label>  
 <div class="col-lg-3">  
 <input type="radio" name="sex" value="男" checked="checked" />男  
 </div>  
 <div class="col-lg-3">  
 <input type="radio" name="sex" value="女"/>女  
 </div>  
 <label class="col-lg-2 control-label" id="sexMsg"></label>  
 </div>  
  
 <div class="form-group">  
 <label for="age" class="control-label col-lg-2">年龄：</label>  
 <div class="col-lg-8">  
 <input type="text" class="form-control" id="age" name="age" placeholder="请输入年龄" value="${contact.age }">  
 </div>  
 </div>  
  
 <div class="form-group">  
 <label for="jiguan" class="col-lg-2 control-label">籍贯：</label>  
 <div class="col-lg-8">  
 <select name="address" class="form-control" id="jiguan">  
 <option value="广东">广东</option>  
 <option value="广西">广西</option>  
 <option value="湖南">湖南</option>  
 </select>  
 </div>  
 </div>  
  
 <div class="form-group">  
 <label for="qq" class="control-label col-lg-2">QQ：</label>  
 <div class="col-lg-8">  
 <input type="text" class="form-control" id="qq" name="qq" placeholder="请输入QQ号码" value="${contact.qq }"/>  
 </div>  
 </div>  
  
 <div class="form-group">  
 <label for="email" class="col-lg-2 control-label">Email：</label>  
 <div class="col-lg-8">  
 <span style="color: red;">${msg }</span>  
 <input type="text" class="form-control" id="email" name="email" placeholder="请输入邮箱地址" value="${contact.email }"/>  
 </div>  
 </div>  
  
 <div class="form-group" style="text-align: center">  
 <input class="btn btn-primary" type="submit" value="提交" />  
 <input class="btn btn-default" type="reset" value="重置" />  
 <input class="btn btn-default" type="button" value="返回" />  
 </div>  
 </form>  
</div>  
</body>

## 逻辑删除思路分析 :



**list.jsp 页面 :**

<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>  
<%@ page contentType="text/html;charset=UTF-8" language="java" %>  
<!DOCTYPE html>  
*<!-- 网页使用的语言 -->*<html lang="zh-CN">  
<head>  
 *<!-- 指定字符集 -->* <meta charset="utf-8">  
 *<!-- 使用Edge最新的浏览器的渲染方式 -->* <meta http-equiv="X-UA-Compatible" content="IE=edge">  
 *<!-- viewport视口：网页可以根据设置的宽度自动进行适配，在浏览器的内部虚拟一个容器，容器的宽度与设备的宽度相同。  
 width: 默认宽度与设备的宽度相同  
 initial-scale: 初始的缩放比，为1:1 -->* <meta name="viewport" content="width=device-width, initial-scale=1">  
 *<!-- 上述3个meta标签\*必须\*放在最前面，任何其他内容都\*必须\*跟随其后！ -->* <title>用户信息管理系统</title>  
  
 *<!-- 1. 导入CSS的全局样式 -->* <link href="css/bootstrap.min.css" rel="stylesheet">  
 *<!-- 2. jQuery导入，建议使用1.9以上的版本 -->* <script src="js/jquery-2.1.0.min.js"></script>  
 *<!-- 3. 导入bootstrap的js文件 -->* <script src="js/bootstrap.min.js"></script>  
 <style type="text/css">  
 td, th {  
 text-align: center;  
 }  
 </style>  
  
 <script>  
 *// 根据 id 删除联系人* function *deleteContactByID*(id) {  
 *// 弹框, 判断用户是否真的要删除* var result = *window*.confirm("您确定真的要删除吗?")  
 if (result) {  
 *// 用户点击了确定按钮, 实现逻辑删除  
 location*.href = "/deleteContactByID?id=" + id;  
 }  
 }  
 </script>  
</head>  
<body>  
<div class="container">  
 <h3 style="text-align: center">用户信息列表</h3>  
 <table border="1" class="table table-bordered table-hover">  
 <tr class="success">  
 <th>编号</th>  
 <th>姓名</th>  
 <th>性别</th>  
 <th>年龄</th>  
 <th>籍贯</th>  
 <th>QQ</th>  
 <th>邮箱</th>  
 <th>操作</th>  
 </tr>  
 <c:forEach items="${contactList }" var="contact">  
 <tr>  
 <td>${contact.id}</td>  
 <td>${contact.name}</td>  
 <td>${contact.sex}</td>  
 <td>${contact.age}</td>  
 <td>${contact.address}</td>  
 <td>${contact.qq}</td>  
 <td>${contact.email}</td>  
 <td><a class="btn btn-default btn-sm" href="update.jsp">修改</a>&nbsp;  
 <a class="btn btn-default btn-sm" onclick="*deleteContactByID*(${contact.id });">删除</a>  
 </td>  
 </tr>  
 </c:forEach>  
 <tr>  
 <td colspan="8" align="center"><a class="btn btn-primary" href="add.jsp">添加联系人</a></td>  
 </tr>  
 </table>  
</div>  
</body>  
</html>



## 逻辑删除代码实现 :

@WebServlet("/deleteContactByID")  
public class DeleteContactByIDServlet extends HttpServlet {  
 protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 doGet(request, response);  
 }  
  
 protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 *// 1. 获取 id* int id = Integer.*parseInt*(request.getParameter("id"));  
 *// 2. 根据 id 实现逻辑删除* ContactService contactService = new ContactService();  
 contactService.deleteContactByID(id);  
 *// 3. 删除成功后, 跳转到 delete\_ok.jsp 页面* response.sendRedirect("/delete\_ok.jsp");  
 }  
}

public class ContactService {  
  
 ContactDao contactDao = new ContactDao();  
  
 public List<Contact> findAll() {  
 List<Contact> list = contactDao.findAll();  
 return list;  
 }  
  
 *// 添加用户信息, 先查(邮箱是否重复) 后插* public boolean addContact(Contact contact) {  
 *// 1. 查询邮箱是否重复* List<Contact> list = contactDao.queryContactByEmail(contact.getEmail());  
 *// 2. 对集合大小实现判断* if (list == null || list.size() == 0) {  
 *// 2.1 数据库中没有重复邮箱地址, 插入数据* contactDao.addContact(contact);  
 return true;  
 } else {  
 *// 2.2 有重复邮箱, 插入失败* return false;  
 }  
 }  
  
 *// 根据 id 实现逻辑删除* public void deleteContactByID(int id) {  
 contactDao.deleteContactByID(id);  
 }  
}

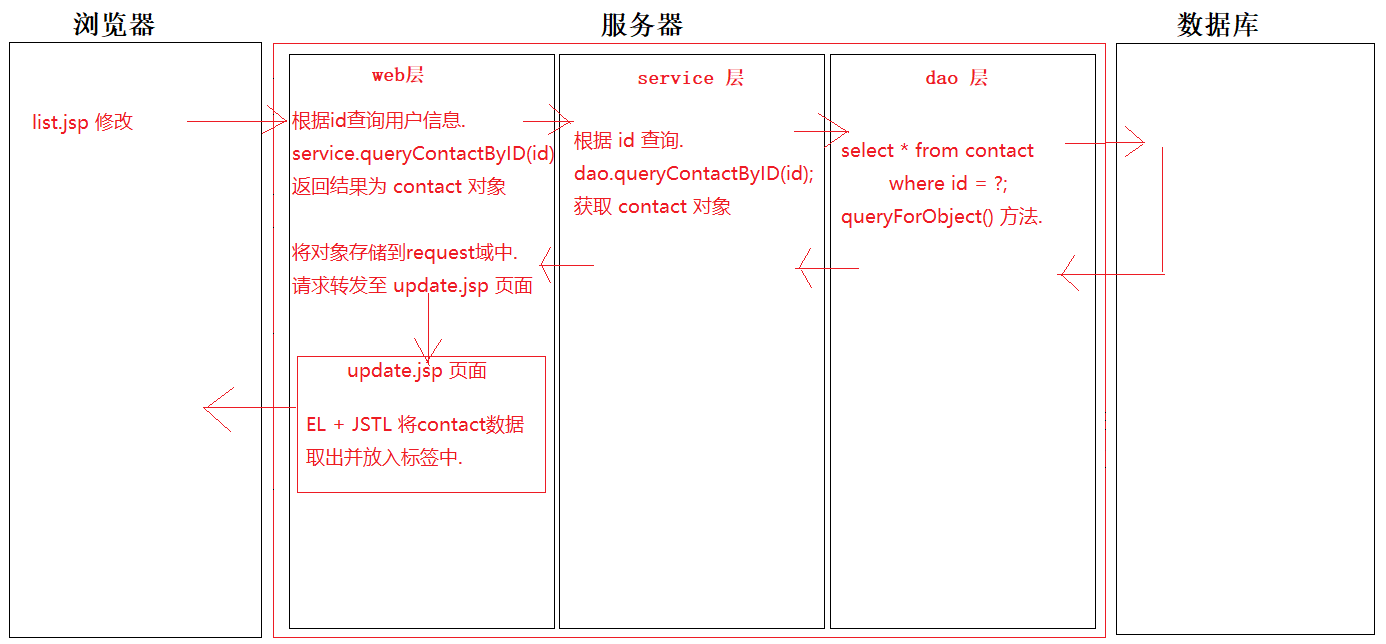
public class ContactDao {  
  
 JdbcTemplate jdbcTemplate = new JdbcTemplate(JDBCUtils.*getDataSource*());  
  
 public List<Contact> findAll() {  
 String sql = "select *\** from contact where isdelete = 0;";  
 List<Contact> list = jdbcTemplate.query(sql, new BeanPropertyRowMapper<>(Contact.class));  
 return list;  
 }  
  
 *// 根据邮箱查询用户信息* public List<Contact> queryContactByEmail(String email) {  
 String sql = "select *\** from contact where email = ? and isdelete = 0;";  
 List<Contact> list = jdbcTemplate.query(sql, new BeanPropertyRowMapper<>(Contact.class), email);  
 return list;  
 }  
  
 *// 添加用户* public void addContact(Contact contact) {  
 String sql = "insert into contact values(null, ?, ?, ?, ?, ?, ?, 0);";  
 Object[] params = { contact.getName(), contact.getSex(), contact.getAge(), contact.getAddress(), contact.getQq(), contact.getEmail() };  
 jdbcTemplate.update(sql, params);  
 }  
  
 public void deleteContactByID(int id) {  
 *// 逻辑删除* String sql = "update contact set isdelete = 1 where id = ?";  
 *// 物理删除 delete from contact where id = ?;* jdbcTemplate.update(sql, id);  
 }  
}

**delete\_ok.jsp 页面**

<%@ page contentType="text/html;charset=UTF-8" language="java" %>  
<html>  
<head>  
 <title>添加成功</title>  
</head>  
<body>  
 <a href="/findAll">查询删除结果</a> <br>  
 <a href="index.jsp">主页面</a>  
</body>  
</html>

## 修改回显思路分析 :





list.jsp 页面 :

<body>  
<div class="container">  
 <h3 style="text-align: center">用户信息列表</h3>  
 <table border="1" class="table table-bordered table-hover">  
 <tr class="success">  
 <th>编号</th>  
 <th>姓名</th>  
 <th>性别</th>  
 <th>年龄</th>  
 <th>籍贯</th>  
 <th>QQ</th>  
 <th>邮箱</th>  
 <th>操作</th>  
 </tr>  
 <c:forEach items="${contactList }" var="contact">  
 <tr>  
 <td>${contact.id}</td>  
 <td>${contact.name}</td>  
 <td>${contact.sex}</td>  
 <td>${contact.age}</td>  
 <td>${contact.address}</td>  
 <td>${contact.qq}</td>  
 <td>${contact.email}</td>  
 <td><a class="btn btn-default btn-sm" href="/queryContactByID?id=${contact.id }">修改</a>&nbsp;  
 <a class="btn btn-default btn-sm" onclick="*deleteContactByID*(${contact.id });">删除</a>  
 </td>  
 </tr>  
 </c:forEach>  
 <tr>  
 <td colspan="8" align="center"><a class="btn btn-primary" href="add.jsp">添加联系人</a></td>  
 </tr>  
 </table>  
</div>  
</body>

@WebServlet("/queryContactByID")  
public class QueryContactByIDServlet extends HttpServlet {  
 protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 doGet(request, response);  
 }  
  
 protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 *// 1. 获取用户 id* int id = Integer.*parseInt*(request.getParameter("id"));  
 *// 2. 调用 service 层方法查询业务逻辑* ContactService contactService = new ContactService();  
 Contact contact = contactService.queryContactByID(id);  
 *// 3. 将 contact 对象存储到 request 域中* request.setAttribute("contact", contact);  
 *// 4. 请求转发到 update.jsp 页面* request.getRequestDispatcher("/update.jsp").forward(request, response);  
 }  
}

*// service 层 : 根据 id查询用户信息*public Contact queryContactByID(int id) {  
 Contact contact = contactDao.queryContactByID(id);  
 return contact;  
}

public Contact queryContactByID(int id) {  
 String sql = "select *\** from contact where id = ?;";  
 Contact contact = jdbcTemplate.queryForObject(sql, new BeanPropertyRowMapper<>(Contact.class), id);  
 return contact;  
}

**update.jsp 页面 :**

<%@ page language="java" import="java.util.\*" pageEncoding="UTF-8"%>  
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>  
<%  
String path = request.getContextPath();  
String basePath = request.getScheme()+"://"+request.getServerName()+":"+request.getServerPort()+path+"/";  
%>  
<!DOCTYPE html>  
*<!-- 网页使用的语言 -->*<html lang="zh-CN">  
 <head>  
 <base href="<%=basePath%>"/>  
 *<!-- 指定字符集 -->* <meta charset="utf-8">  
 <meta http-equiv="X-UA-Compatible" content="IE=edge">  
 <meta name="viewport" content="width=device-width, initial-scale=1">  
 <title>修改用户</title>  
  
 <link href="css/bootstrap.min.css" rel="stylesheet">  
 <script src="js/jquery-2.1.0.min.js"></script>  
 <script src="js/bootstrap.min.js"></script>  
   
 </head>  
 <body>  
 <div class="container" style="width: 400px;">  
 <h3 style="text-align: center;">修改联系人</h3>  
 <form action="" method="post">  
 <div class="form-group">  
 <label for="name">姓名：</label>  
 <input type="text" class="form-control" id="name" name="name" placeholder="请输入姓名" value="${contact.name }" />  
 </div>  
  
 <div class="form-group">  
 <label>性别：</label>  
 <input type="radio" name="sex" value="男"  
 <c:if test="${contact.sex == '男'}">checked="checked"</c:if>  
 />男  
 <input type="radio" name="sex" value="女"  
 <c:if test="${contact.sex == '女'}">checked="checked"</c:if>  
 />女  
 </div>  
  
 <div class="form-group">  
 <label for="age">年龄：</label>  
 <input type="text" class="form-control" id="age" name="age" placeholder="请输入年龄" value="${contact.age }"/>  
 </div>  
  
 <div class="form-group">  
 <label>籍贯：</label>  
 <select name="address" class="form-control" >  
 <option value="广东"  
 <c:if test="${contact.address == '广东'}">selected="selected"</c:if>  
 >广东</option>  
 <option value="广西"  
 <c:if test="${contact.address == '广西'}">selected="selected"</c:if>  
 >广西</option>  
 <option value="湖南"  
 <c:if test="${contact.address == '湖南'}">selected="selected"</c:if>  
 >湖南</option>  
 </select>  
 </div>  
  
 <div class="form-group">  
 <label>QQ：</label>  
 <input type="text" class="form-control" name="qq" placeholder="请输入QQ号码" value="${contact.qq }"/>  
 </div>  
  
 <div class="form-group">  
 <label>Email：</label>  
 <input type="text" class="form-control" name="email" placeholder="请输入邮箱地址" value="${contact.email }"/>  
 </div>  
  
 <div class="form-group" style="text-align: center">  
 <input class="btn btn-primary" type="submit" value="提交" />  
 <input class="btn btn-default" type="reset" value="重置" />  
 <input class="btn btn-default" type="button" value="返回"/>  
 </div>  
 </form>  
 </div>  
 </body>  
</html>

## 更新联系人信息 :





**update.jsp 页面 :**

<body>  
 <div class="container" style="width: 400px;">  
 <h3 style="text-align: center;">修改联系人</h3>  
 <form action="/updateContactByID" method="post">  
 *<%--提交数据方式二 : 隐藏域 (注意:post请求必须在请求体中提交参数) --%>* <input type="hidden" name="id" value="${contact.id }">  
 <div class="form-group">  
 <label for="name">姓名：</label>  
 <input type="text" class="form-control" id="name" name="name" placeholder="请输入姓名" value="${contact.name }" />  
 </div>  
  
 <div class="form-group">  
 <label>性别：</label>  
 <input type="radio" name="sex" value="男"  
 <c:if test="${contact.sex == '男'}">checked="checked"</c:if>  
 />男  
 <input type="radio" name="sex" value="女"  
 <c:if test="${contact.sex == '女'}">checked="checked"</c:if>  
 />女  
 </div>  
  
 <div class="form-group">  
 <label for="age">年龄：</label>  
 <input type="text" class="form-control" id="age" name="age" placeholder="请输入年龄" value="${contact.age }"/>  
 </div>  
  
 <div class="form-group">  
 <label>籍贯：</label>  
 <select name="address" class="form-control" >  
 <option value="广东"  
 <c:if test="${contact.address == '广东'}">selected="selected"</c:if>  
 >广东</option>  
 <option value="广西"  
 <c:if test="${contact.address == '广西'}">selected="selected"</c:if>  
 >广西</option>  
 <option value="湖南"  
 <c:if test="${contact.address == '湖南'}">selected="selected"</c:if>  
 >湖南</option>  
 </select>  
 </div>  
  
 <div class="form-group">  
 <label>QQ：</label>  
 <input type="text" class="form-control" name="qq" placeholder="请输入QQ号码" value="${contact.qq }"/>  
 </div>  
  
 <div class="form-group">  
 <label>Email：</label>  
 <input type="text" class="form-control" name="email" placeholder="请输入邮箱地址" value="${contact.email }" readonly="readonly"/>  
 </div>  
  
 <div class="form-group" style="text-align: center">  
 <input class="btn btn-primary" type="submit" value="提交" />  
 <input class="btn btn-default" type="reset" value="重置" />  
 <input class="btn btn-default" type="button" value="返回"/>  
 </div>  
 </form>  
 </div>  
</body>

@WebServlet("/updateContactByID")  
public class UpdateContactByIDServlet extends HttpServlet {  
 protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 doGet(request, response);  
 }  
  
 protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 *// 1. 解决乱码* request.setCharacterEncoding("utf-8");  
 *// 2. 获取用户提交的数据 (BeanUtils工具类)* Contact contact = new Contact();  
 try {  
 BeanUtils.*populate*(contact, request.getParameterMap());  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 *// 3. 实现业务层更新操作* ContactService contactService = new ContactService();  
 contactService.updateContactByID(contact);  
 *// 4. 更新完成后, 跳转到 update\_ok.jsp 页面* response.sendRedirect("/update\_ok.jsp");  
 }  
}

*// 根据 id 修改用户 (service层)*public void updateContactByID(Contact contact) {  
 contactDao.updateContactByID(contact);  
}

*// 根据 id 实现联系人信息更新*public void updateContactByID(Contact contact) {  
 String sql = "update contact set name = ?, sex = ?, age = ?, address = ?, qq = ? where id = ?;";  
 Object[] params = { contact.getName(), contact.getSex(), contact.getAge(), contact.getAddress(), contact.getQq(), contact.getId() };  
 jdbcTemplate.update(sql, params);  
}

**update\_ok.jsp 页面 :**

<%@ page contentType="text/html;charset=UTF-8" language="java" %>  
<html>  
<head>  
 <title>修改成功</title>  
</head>  
<body>  
 <a href="/findAll">查询修改结果</a> <br>  
 <a href="index.jsp">主页面</a>  
</body>  
</html>

## 分页查询 :



**定义 PageBean 实体类 :**

public class PageBean {  
 *// 每页显示数据* private List<Contact> data;  
 *// 每一页显示的数据个数* private int pageSize;  
 *// 数据库数据总条数* private int count;  
 *// 当前页数* private int currentPage;  
 *// 总的页数* private int totalPage;  
  
 *// 获取开始索引* public int getStartIndex() {  
 return (this.currentPage - 1) \* this.pageSize;  
 }  
  
 @Override  
 public String toString() {  
 return "PageBean{" +  
 "data=" + data +  
 ", pageSize=" + pageSize +  
 ", count=" + count +  
 ", currentPage=" + currentPage +  
 ", totalPage=" + totalPage +  
 '}';  
 }  
  
 public List<Contact> getData() {  
 return data;  
 }  
  
 public void setData(List<Contact> data) {  
 this.data = data;  
 }  
  
 public int getPageSize() {  
 return pageSize;  
 }  
  
 public void setPageSize(int pageSize) {  
 this.pageSize = pageSize;  
 }  
  
 public int getCount() {  
 return count;  
 }  
  
 public void setCount(int count) {  
 this.count = count;  
 }  
  
 public int getCurrentPage() {  
 return currentPage;  
 }  
  
 public void setCurrentPage(int currentPage) {  
 this.currentPage = currentPage;  
 }  
  
 *// 计算获取总页码数* public int getTotalPage() {  
 return (int) Math.*ceil*(this.count \* 1.0 / this.pageSize);  
 }  
  
 public void setTotalPage(int totalPage) {  
 this.totalPage = totalPage;  
 }  
}

**编写 PageBeanServlet 类 :**

@WebServlet("/pageBean")  
public class PageBeanServlet extends HttpServlet {  
 protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 doGet(request, response);  
 }  
  
 protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 *// 1. 获取当前页码* int currentPage = Integer.*parseInt*(request.getParameter("currentPage"));  
 *// 2. 定义每页显示的条数* int pageSize = 2;  
 *// 3. 使用 service 层查询数据, 将数据封装到 pageBean 对象中* ContactService contactService = new ContactService();  
 PageBean pageBean = contactService.queryPageBean(currentPage, pageSize);  
 *// 4. 将 pageBean 对象存储到 request 域中* request.setAttribute("pageBean", pageBean);  
 *// 5. 请求转发至 page.jsp 页面* request.getRequestDispatcher("/page.jsp").forward(request, response);  
 }  
}

*// 将数据封装到 pageBean 对象中 service层*public PageBean queryPageBean(int currentPage, int pageSize) {  
 *// 1. 创建 PageBean 对象* PageBean pageBean = new PageBean();  
 pageBean.setCurrentPage(currentPage);  
 pageBean.setPageSize(pageSize);  
 *// 2. 每一页的显示数据* List<Contact> data = contactDao.queryData(pageBean.getStartIndex(), pageBean.getPageSize());  
 pageBean.setData(data);  
 *// 3. 查询数据库中记录数* int count = contactDao.queryCount();  
 pageBean.setCount(count);  
 *// 4. 返回封装完毕后的 pageBean 对象* return pageBean;  
}

*// 查询每一页显示的数据集合*public List<Contact> queryData(int startIndex, int pageSize) {  
 String sql = "select *\** from contact where isdelete = 0 limit ?, ?;";  
 List<Contact> data = jdbcTemplate.query(sql, new BeanPropertyRowMapper<>(Contact.class), startIndex, pageSize);  
 return data;  
}  
  
*// 查询数据库中的总记录数*public int queryCount() {  
 String sql = "select *count*(*\**) from contact where isdelete = 0;";  
 Integer count = jdbcTemplate.queryForObject(sql, int.class);  
 return count;  
}

**复制 list.jsp页面, 将名称修改为 page.jsp :**

分页组件复制代码 :

<nav>  
 <ul class="pagination">  
 *<%-- 上一页 --%>* <li class="disabled">  
 <a href="#" >  
 <span>&laquo;</span>  
 </a>  
 </li>  
  
 *<%-- 页码显示区 --%>* <li class="active"><a href="#">1</a></li>  
 <li><a href="#">2</a></li>  
 <li><a href="#">3</a></li>  
 <li><a href="#">4</a></li>  
  
 *<%-- 下一页 --%>* <li>  
 <a href="#">  
 <span>&raquo;</span>  
 </a>  
 </li>  
 </ul>  
</nav>

<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>  
<%@ page contentType="text/html;charset=UTF-8" language="java" %>  
<!DOCTYPE html>  
*<!-- 网页使用的语言 -->*<html lang="zh-CN">  
<head>  
 *<!-- 指定字符集 -->* <meta charset="utf-8">  
 *<!-- 使用Edge最新的浏览器的渲染方式 -->* <meta http-equiv="X-UA-Compatible" content="IE=edge">  
 *<!-- viewport视口：网页可以根据设置的宽度自动进行适配，在浏览器的内部虚拟一个容器，容器的宽度与设备的宽度相同。  
 width: 默认宽度与设备的宽度相同  
 initial-scale: 初始的缩放比，为1:1 -->* <meta name="viewport" content="width=device-width, initial-scale=1">  
 *<!-- 上述3个meta标签\*必须\*放在最前面，任何其他内容都\*必须\*跟随其后！ -->* <title>用户信息管理系统</title>  
  
 *<!-- 1. 导入CSS的全局样式 -->* <link href="css/bootstrap.min.css" rel="stylesheet">  
 *<!-- 2. jQuery导入，建议使用1.9以上的版本 -->* <script src="js/jquery-2.1.0.min.js"></script>  
 *<!-- 3. 导入bootstrap的js文件 -->* <script src="js/bootstrap.min.js"></script>  
 <style type="text/css">  
 td, th {  
 text-align: center;  
 }  
 </style>  
  
 <script>  
 *// 根据 id 删除联系人* function *deleteContactByID*(id) {  
 *// 弹框, 判断用户是否真的要删除* var result = *window*.confirm("您确定真的要删除吗?")  
 if (result) {  
 *// 用户点击了确定按钮, 实现逻辑删除  
 location*.href = "/deleteContactByID?id=" + id;  
 }  
 }  
 </script>  
</head>  
<body>  
<div class="container">  
 <h3 style="text-align: center">用户信息列表</h3>  
 <table border="1" class="table table-bordered table-hover">  
 <tr class="success">  
 <th>编号</th>  
 <th>姓名</th>  
 <th>性别</th>  
 <th>年龄</th>  
 <th>籍贯</th>  
 <th>QQ</th>  
 <th>邮箱</th>  
 <th>操作</th>  
 </tr>  
 <c:forEach items="${pageBean.data }" var="contact">  
 <tr>  
 <td>${contact.id}</td>  
 <td>${contact.name}</td>  
 <td>${contact.sex}</td>  
 <td>${contact.age}</td>  
 <td>${contact.address}</td>  
 <td>${contact.qq}</td>  
 <td>${contact.email}</td>  
 <td><a class="btn btn-default btn-sm" href="/queryContactByID?id=${contact.id }">修改</a>&nbsp;  
 <a class="btn btn-default btn-sm" onclick="*deleteContactByID*(${contact.id });">删除</a>  
 </td>  
 </tr>  
 </c:forEach>  
 <tr>  
 <td colspan="8" align="center"><a class="btn btn-primary" href="add.jsp">添加联系人</a></td>  
 </tr>  
 </table>  
  
 <div style="width: 100%; text-align: center">  
 <nav>  
 <ul class="pagination">  
 *<%-- 上一页 --%>* <c:if test="${pageBean.currentPage == 1 }">  
 <li class="disabled">  
 <a href="#" >  
 <span>&laquo;</span>  
 </a>  
 </li>  
 </c:if>  
 <c:if test="${pageBean.currentPage != 1 }">  
 <li>  
 <a href="/pageBean?currentPage=${pageBean.currentPage - 1 }" >  
 <span>&laquo;</span>  
 </a>  
 </li>  
 </c:if>  
  
 *<%-- 页码显示区 --%>* <c:forEach begin="1" end="${pageBean.totalPage }" var="i">  
 <c:if test="${pageBean.currentPage == i}">  
 <li class="active"><span style="color: red;">${i }</span></li>  
 </c:if>  
 <c:if test="${pageBean.currentPage != i}">  
 <li class="active"><li><a href="/pageBean?currentPage=${i }">${i }</a></li></li>  
 </c:if>  
 </c:forEach>  
  
 *<%-- 下一页 --%>* <c:if test="${pageBean.currentPage == pageBean.totalPage }">  
 <li class="disabled">  
 <a href="#">  
 <span>&raquo;</span>  
 </a>  
 </li>  
 </c:if>  
 <c:if test="${pageBean.currentPage != pageBean.totalPage }">  
 <li>  
 <a href="/pageBean?currentPage=${pageBean.currentPage + 1 }">  
 <span>&raquo;</span>  
 </a>  
 </li>  
 </c:if>  
 </ul>  
 </nav>  
 </div>  
  
</div>  
</body>  
</html>

**index.jsp 页面 :**

<%@ page language="java" import="java.util.\*" pageEncoding="UTF-8"%>  
<%  
String path = request.getContextPath();  
String basePath = request.getScheme()+"://"+request.getServerName()+":"+request.getServerPort()+path+"/";  
%>  
  
<!DOCTYPE html>   
<html lang="zh-CN">  
 <head>  
 <base href="<%=basePath%>"/>  
 <meta charset="utf-8"/>  
 <meta http-equiv="X-UA-Compatible" content="IE=edge"/>  
 <meta name="viewport" content="width=device-width, initial-scale=1"/>  
 <title>首页</title>  
  
 *<!-- 1. 导入CSS的全局样式 -->* <link href="css/bootstrap.min.css" rel="stylesheet">  
 *<!-- 2. jQuery导入，建议使用1.9以上的版本 -->* <script src="js/jquery-2.1.0.min.js"></script>  
 *<!-- 3. 导入bootstrap的js文件 -->* <script src="js/bootstrap.min.js"></script>  
 <script type="text/javascript">  
 </script>  
 </head>  
 <body>  
 <div align="center">  
 <a  
 href="/pageBean?currentPage=1" style="text-decoration:none;font-size:33px">查询所有用户信息  
 </a>  
 </div>  
 </body>  
</html>



# 知识点 :

**目标 :**

**步骤 :**

**实现 :**

**小结 :**