**day10-Servlet的使用和跳转**

教学目标

1、掌握B/S和C/S的区别

2、掌握Tomcat的搭建

3、掌握Web项目的结构

4、掌握Servlet获取请求参数

5、掌握Servlet实现页面跳转

6、掌握Servlet的生命周期

7、掌握Servlet的三种创建方式

8、掌握Servlet的两种配置方式

**第一章 Web应用的演变**

1.1 B/S 与C/S模式

|  |  |
| --- | --- |
| B/S:Browser/Server或浏览器/服务器模式 | C/S:Client/Server或客户端/服务器模式 |
| 优点是用户使用简单只要有浏览器和网络即可 | 常见的CS程序：LOL、穿越火线、QQ  缺点是对用户的电脑配置要求较高 |
| 常见的程序：淘宝网、京东网等 | 常见的CS程序：LOL、穿越火线、QQ |

1.2 Web服务器简介

Web服务器是运行及发布Web应用的容器，只有将开发的Web项目放置到该容器中，才能使网络中的所有用户通过浏览器进行访问。开发Java Web应用所采用的服务器主要是与JSP/Servlet兼容的Web服务器，比较常用的有Tomcat、Resin、JBoss、WebSphere 和 WebLogic 等

|  |  |
| --- | --- |
| **Tomcat 服务器** | 目前最为流行的Tomcat服务器是Apache-Jarkarta开源项目中的一个子项目，是一个小型、轻量级的支持JSP和Servlet 技术的Web服务器，也是初学者学习开发JSP应用的首选 |
| **Resin 服务器** | Resin是Caucho公司的产品，是一个非常流行的支持Servlet和JSP的服务器，速度非常快。Resin本身包含了一个支持HTML的Web服务器，这使它不仅可以显示动态内容，而且显示静态内容的能力也毫不逊色，因此许多网站都是使用Resin服务器构建 |
| **JBoss服务器** | JBoss是一个种遵从JavaEE规范的、开放源代码的、纯Java的EJB服务器，对于J2EE有很好的支持。JBoss采用JML API实现软件模块的集成与管理，其核心服务又是提供EJB服务器，不包含Servlet和JSP的Web容器，不过它可以和Tomcat完美结合 |
| **WebSphere 服务器** | WebSphere是IBM公司的产品，可进一步细分为 WebSphere Performance Pack、Cache Manager 和WebSphere Application Server等系列，其中WebSphere Application Server 是基于Java 的应用环境，可以运行于 Sun Solaris、Windows NT 等多种操作系统平台，用于建立、部署和管理Internet和Intranet Web应用程序 |
| **WebLogic 服务器** | WebLogic 是BEA公司的产品（现在已经被Oracle收购），可进一步细分为 WebLogic Server、WebLogic Enterprise 和 WebLogic Portal 等系列，其中 WebLogic Server 的功能特别强大。WebLogic 支持企业级的、多层次的和完全分布式的Web应用，并且服务器的配置简单、界面友好。对于那些正在寻求能够提供Java平台所拥有的一切应用服务器的用户来说，WebLogic是一个十分理想的选择 |

**第二章 Tomcat服务器搭建**

2.1 Tomcat服务器介绍

Tomcat是Apache 软件基金会（Apache Software Foundation）的Jakarta 项目中的一个核心项目，由[Apache](https://baike.baidu.com/item/Apache/6265)、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，Tomcat 5支持最新的Servlet 2.4 和JSP 2.0 规范。

2.3 Tomcat目录的介绍

|  |  |
| --- | --- |
| **bin目录** | 存放的是二进制可执行文件，如果是安装版，那么这个目录下会有两个exe文件：tomcat9.exe、tomcat9w.exe，前者是在控制台下启动Tomcat，后者是弹出UGI窗口启动Tomcat；如果是解压版，那么会有startup.bat和shutdown.bat文件，startup.bat用来启动Tomcat，但需要JDK的配置，shutdawn.bat用来停止Tomcat； |
| **conf目录** | 这是一个非常非常重要的目录，这个目录下有四个最为重要的文件  server.xml：配置整个服务器信息。例如修改端口号，添加虚拟主机等  tomcatusers.xml：存储tomcat用户的文件，这里保存的是tomcat的用户名及密码，以及用户的角色信息。可以按着该文件中的注释信息添加tomcat用户，然后就可以在Tomcat主页中进入Tomcat Manager页面了  web.xml：部署描述符文件，这个文件中注册了很多MIME类型，即文档类型  context.xml：对所有应用的统一配置，通常我们不会去配置它 |
| **lib目录** | Tomcat的类库，里面是一大堆jar文件。如果需要添加Tomcat依赖的jar文件，可以把它放到这个目录中，当然也可以把应用依赖的jar文件放到这个目录中，这个目录中的jar所有项目都可以共享之，但这样你的应用放到其他Tomcat下时就不能再共享这个目录下的Jar包了，所以建议只把Tomcat需要的Jar包放到这个目录下 |
| **webapps目录** | 存放web项目的目录，其中每个文件夹都是一个项目；如果这个目录下已经存在了目录，那么都是tomcat自带的项目。其中ROOT是一个特殊的项目，在地址栏中没有给出项目目录时，对应的就是ROOT项目 |
| **work目录** | 运行时生成的文件，最终运行的文件都在这里。通过webapps中的项目生成的！可以把这个目录下的内容删除，再次运行时会生再次生成work目录。当客户端用户访问一个JSP文件时，Tomcat会通过JSP生成Java文件，然后再编译Java文件生成class文件，生成的java和class文件都会存放到这个目录下 |
| **temp目录** | 存放Tomcat的临时文件，这个目录下的东西可以在停止Tomcat后删除 |

2.4 启动tomcat服务器

​ tomcat安装文件/bin/startup.bat 启动程序

\*\*测试\*\*

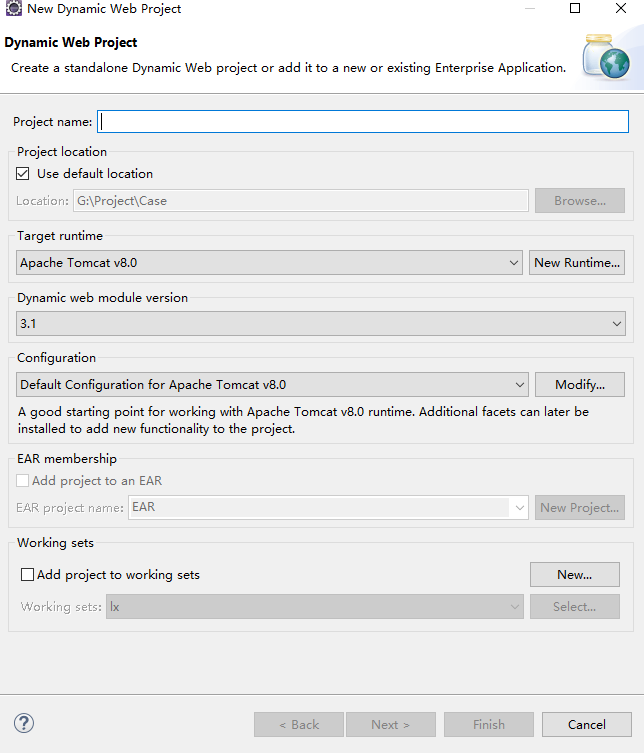
​ 打开浏览器!输入 http://localhost:8080

如果出现以下界面证明成功!（图略）

2.4 在Eclipse中集成Tomcat服务器（图略）

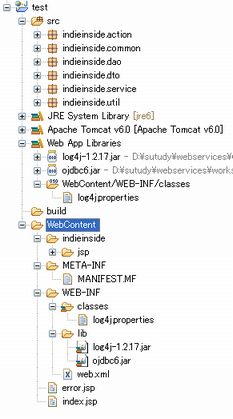
**第三章 JavaWEB项目创建及结构**

3.1 javaWeb项目创建



输入项目名称点击Finsh即可

3.2 javaweb项目结构讲解



|  |  |
| --- | --- |
| **src目录** | Java源码 |
| **Webcontent目录** | 发布到服务器中的内容 |
| **WEB-INF中的lib** | 项目中用到的所有jar |
| **WEB-INF中的classes** | Class文件 |

**第四章 HTTP协议回顾**

4.1 HTTP协议简介

​ HTTP是一个属于应用层的面向对象的协议，由于其简捷、快速的方式，适用于分布式超媒体信息系统。

HTTP协议的主要特点可概括如下：

1.支持客户/服务器模式。

2.简单快速：客户向服务器请求服务时，只需传送请求方法和路径。请求方法常用的有GET、HEAD、POST。每种方法规定了客户与服务器联系的类型不同。由于HTTP协议简单，使得HTTP服务器的程序规模小，因而通信速度很快。

3.灵活：HTTP允许传输任意类型的数据对象。正在传输的类型由Content-Type加以标记。

4.无连接：无连接的含义是限制每次连接只处理一个请求。服务器处理完客户的请求，并收到客户的应答后，即断开连接。采用这种方式可以节省传输时间。

**5.无状态：**HTTP协议是无状态协议。无状态是指协议对于事务处理没有记忆能力。缺少状态意味着如果后续处理需要前面的信息，则它必须重传，这样可能导致每次连接传送的数据量增大。另一方面，在服务器不需要先前信息时它的应答就较快。

描述客户端和服务器端的数据标准，该协议由W3C维护和管理。

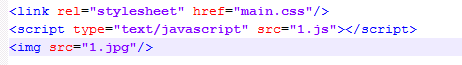
版本：

HTTP1.0:每次发出请求都需要建立网络连接

HTTP1.1:(主流)在一次网络连接上发出多次请求和得到多次响应。多了一些头。

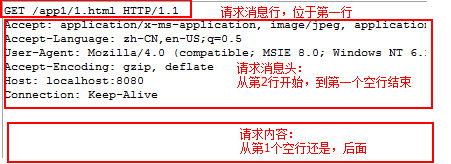
明确：

浏览器遇到以下标记时，会自动发出请求



4.2 HTTP协议详解之请求篇

http请求由三部分组成，分别是：请求行、消息报头、请求正文



1）请求行：

GET /app1/1.html HTTP/1.1

GET：请求方式。

常用的请求方式：GET(默认的)、POST、HEAD、OPTIONS等

GET：默认的

<http://localhost:8080/app1/1.html?username=abc&password=123>

提交的数据显示出来了，相对不安全，协议的第一行有长度限制，<1kb.

(有请求数据时推荐)POST：可以通过<form method=”post”/>

username=abc&password=123

提交的数据在请求正文中的，相对安全，长度没有限制。

2） 请求消息头

作用：向服务器端传递附加信息（暗号指令）

* Accept:告知服务器，客户端可以接受的数据类型（MIME类型）

文件系统：通过文件的扩展名区分不同的文件的。txt jpeg

MIME类型：大类型/小类型。 txt--->text/plain html---->text/html js---->text/javascript (具体对应关系：Tomcat\conf\web.xml)

* Accept-Encoding：告知服务器，客户端可以接受的压缩编码。比如gzip
* Accept-Language：告知服务器，客户端支持的语言。
* Referer：告知服务器，从哪个页面过来的。
* Content-Type：告知服务器，请求正文的MIME类型

默认类型：application/x-www-form-urlencoded(表单enctype属性的默认取值)

具体体现：username=abc&password=123

其他类型：multipart/form-data(文件上传时用的)

* User-Agent:告知服务器，浏览器的类型
* Content-Length：请求正文的数据长度

3）请求正文：

数据能提交到服务器，表单中的输入域必须有name属性值

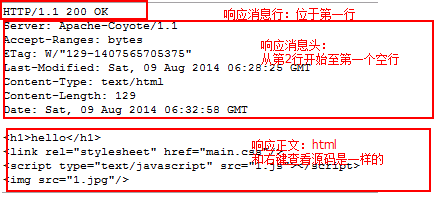
POST请求是才有正文

username=abc&password=123

4.4 HTTP协议详解之响应篇

在接收和解释请求消息后，服务器返回一个HTTP响应消息。

HTTP响应也是由三个部分组成，分别是：状态行、消息报头、响应正文



1）响应行：

HTTP/1.1 200 OK

HTTP/1.1 ：说明服务器端用的协议版本

200 ：响应状态码

1XX 2XX 3XX 4XX 5XX

2XX:处理完毕

4XX：请求有误

5XX：服务器有误

记住常用的响应码：

200：一切正常

302/307:请求重定向

304：服务器上的资源没有发生改变

404：访问的地址不存在

500：服务器端错误

OK：响应吗描述

2）响应消息头：

作用：服务器端向客户端传递的附加信息（暗号指令）

* Location：告知客户端，你去访问的地址。

和302/307实现请求重定向

* Content-Encoding：告知客户端，响应正文使用的压缩编码（gzip）
* Content-Length:告知客户端，响应正文的长度
* Content-Type：告知客户端，响应文正的MIME类型。默认text/html
* Refresh:告知客户端，定时刷新
* Content-Disposition：告知客户端，用下载的方式打开

attachment;filename=23.jpg

3）响应正文：

浏览器解析的正文内容，右键查看源码一样的。

**第五章 Servlet概述**

5.1 什么是servlet

Servlet（Server Applet）是Java Servlet的简称，称为小服务程序或服务连接器，用Java编写的服务器端程序，主要功能在于交互式地浏览和修改数据，生成动态Web内容

Servlet运行于支持Java的应用服务器中。从实现上讲，Servlet可以响应任何类型的请求，但绝大多数情况下Servlet只用来扩展基于HTTP协议的Web服务器。

**Servlet是一个运行在服务器端的Java小程序，通过HTTP协议用于接收来自客户端请求，并发出响应。**

Servlet编程需要使用到javax.servlet 和 javax.servlet.http两个包下面的类和接口，在所有的类和接口中，javax.servlet.servlet 接口最为重要。所有的servlet程序都必须实现该接口或者继承实现了该接口的类。javax.servlet.ServletConfig;

javax.servlet.ServletException;

javax.servlet.http.HttpServlet;

javax.servlet.http.HttpServletRequest;

javax.servlet.http.HttpServletResponse;

javax.servlet.http.HttpSession;

javax.servlet.http.Cookie;

5.2 Servlet的基本使用

1. 在src创建package

2. 选中刚刚创建的包，右键-->New-->Servlet

3. 第一个Servlet代码如下：

|  |
| --- |
| @WebServlet("/HelloServlet")  public class HelloServlet extends HttpServlet {  private static final long serialVersionUID = 1L;    public HelloServlet() {  super();  // TODO Auto-generated constructor stub  }  protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  // 响应内容到浏览器  response.getWriter().print("Hello Word");;  }  protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  // TODO Auto-generated method stub  doGet(request, response);  }  } |

1. 发布右键-->Run As-->Run On Server

2. 浏览器输入网址访问：http://localhost:8080/项目名称/HelloServlet

5.3 常见错误解决方法

HTTP Status 404 找不到

解决方法：查看Tomcat的webapps目录下找到当前项目在WEB-INF下的classes内能否找到刚刚的class文件

如果有，重新启动Tomcat

如果没有，在Eclipse中选择Project-->clean让Eclipse清空缓存并重新构建项目，再次运行

**第六章 Servlet使用**

6.1 Servlet核心接口和类

1）Servlet接口

在ServletAPI中最重要的是Servlet接口，所有Servlet都会直接或间接的与该接口发生联系，或是直接实现该接口，或间接继承自实现了该接口的类。

该接口包括以下五个方法：

init(ServletConfig config)

ServletConfig getServletConfig()

service(ServletRequest req,ServletResponse res)

String getServletInfo()

destroy( )

2）GenericServlet抽象类

它对Servlet接口中的部分方法（init和destroy）添加了实现，使得开发时只需要考虑针对service方法的业务实现即可。

3）HttpServlet类

是继承GenericServlet的基础上进一步的扩展，

6.2 Servlet的三种创建方式

Servlet的第一种创建方式：继承HttpServlet

|  |
| --- |
| /\*\*  \* Servlet implementation class HelloServlet  \* 演示Servlet的第一种创建方式，继承HttpServlet.也是开发中推荐的  \*  \*/  @WebServlet("/hs1")  public class HelloServlet extends HttpServlet {  private static final long serialVersionUID = 1L;    /\*\*  \* @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)  \*/  protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {    response.getWriter().print("我是Servlet创建的第一种方式");  }  /\*\*  \* @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)  \*/  protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  // TODO Auto-generated method stub  doGet(request, response);  }  } |

Servlet创建的第二种方式：实现接口Servlet

|  |
| --- |
| @WebServlet("/hs2")  public class HelloServlet2 implements Servlet{  @Override  public void destroy() {  // TODO Auto-generated method stub  }  @Override  public ServletConfig getServletConfig() {  // TODO Auto-generated method stub  return null;  }  @Override  public String getServletInfo() {  // TODO Auto-generated method stub  return null;  }  @Override  public void init(ServletConfig arg0) throws ServletException {  // TODO Auto-generated method stub  }  @Override  public void service(ServletRequest request, ServletResponse response) throws ServletException, IOException {  // TODO Auto-generated method stub  System.out.println("OK");  response.setContentType("text/html;charset=UTF-8");  response.getWriter().println("我是第二种创建方式");  }  } |

Servlet的第三种创建方式：继承GenericServlet类

|  |
| --- |
| @WebServlet(value="/hs3")  public class HelloServlet3 extends GenericServlet{  @Override  public void service(ServletRequest request, ServletResponse response) throws ServletException, IOException {  // TODO Auto-generated method stub  System.out.println("OK");  response.setContentType("text/html;charset=UTF-8");  response.getWriter().println("我是第三种创建方式");  }  } |

6.3 Servlet的2种配置方式

第一种注解式配置 Servlet3.0及以后 ：

|  |
| --- |
| /\*\*  \* Servlet implementation class HelloServlet  \* 演示Servlet注解式配置  \*/  @WebServlet(value="/hello",loadOnStartup=1)  public class HelloServlet extends HttpServlet {  private static final long serialVersionUID = 1L;  /\*\*  \* @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)  \*/  protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {    response.getWriter().print("OK");  }  /\*\*  \* @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)  \*/  protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  // TODO Auto-generated method stub  doGet(request, response);  }  } |

第二种web.xml配置 Servlet所有版本都支持：

|  |
| --- |
| /\*\*  \* Servlet implementation class HelloServlet  \* 演示Servlet的web.xml配置  \*/  public class HelloServlet extends HttpServlet {  private static final long serialVersionUID = 1L;  /\*\*  \* @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)  \*/  protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {    response.getWriter().print("OK");  }  /\*\*  \* @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)  \*/  protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  // TODO Auto-generated method stub  doGet(request, response);  }  }  ```  web.xml  ```xml  <?xml version="1.0" encoding="UTF-8"?>  <web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://xmlns.jcp.org/xml/ns/javaee" xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app\_3\_1.xsd" version="3.1">  <display-name>Web\_Day11</display-name>  <!--Servlet的第二种配置 -->  <!--Servlet配置 -->  <servlet>  <!--名称 -->  <servlet-name>hello2</servlet-name>  <!--Servlet的全称类名 -->  <servlet-class>com.qf.web.servlet.HelloServlet</servlet-class>  <!--启动的优先级，数字越小越先起作用 -->  <load-on-startup>1</load-on-startup>  </servlet>  <!--映射配置 -->  <servlet-mapping>  <!--名称 -->  <servlet-name>hello2</servlet-name>  <!--资源的匹配规则：精确匹配 -->  <url-pattern>/hello2</url-pattern>  </servlet-mapping>  <welcome-file-list>  <welcome-file>login.html</welcome-file>  </welcome-file-list>  </web-app> |

6.4 Servlet生命周期

6.4.1 生命周期的四个阶段

1、生命周期（必须清晰）：诞生--活着--死亡。人有这个过程，内存中的对象也有。

2、Servlet对象的生命周期

* 诞生：用户第一次访问时，由容器创建他的实例。
* 活着：一旦创建就驻留内存（单例）。每次用户的访问，容器都会调用sevice方法发出响应（多线程）
* 死亡：应用被卸载或者Tomcat关闭了

Servlet代码如下：

|  |
| --- |
| /\*\*  \* Servlet implementation class LifeServlet  \* 演示Servlet的生命周期：  \* 1、实例化  \* 2、init:初始化  \* 3、service：服务  \* 4、destory：销毁  \*/  @WebServlet("/LifeServlet")  public class LifeServlet extends HttpServlet {  private static final long serialVersionUID = 1L;    /\*\*  \* @see HttpServlet#HttpServlet()  \*/  public LifeServlet() {  super();  System.out.println("1、完成了实例化");  // TODO Auto-generated constructor stub  }  @Override  public void init() throws ServletException {  // TODO Auto-generated method stub  super.init();  System.out.println("2、完成了初始化");  }  /\*\*  \* @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)  \*/  protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  // TODO Auto-generated method stub  System.out.println("3、就绪中");  response.getWriter().append("Served at: ").append(request.getContextPath());  }  /\*\*  \* @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)  \*/  protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  // TODO Auto-generated method stub  doGet(request, response);  }    @Override  public void destroy() {  // TODO Auto-generated method stub  super.destroy();  System.out.println("4、销毁了");  }  } |

**第七章 请求对象与响应对象**

7.1 Servlet的Request

HttpRequest 是Servlet中获取请求信息的类，提供了一些方法可以获取请求中的相关内容。

1、获取请求参数的4种方法

a.当表单中只有一个值,根据HTML表单中的name属性来获取

|  |
| --- |
| request.getParameter("username"); |

b.一个名字对应多个参数值,例如checkbox

|  |
| --- |
| String [] cbs = request.getParameterValues("checkbox"); |

c.如果servlet中不知道有哪几个参数名称

|  |
| --- |
| //枚举器  Enumeration e =request.getParameterNames() //username,hobby  String parameterName = null;  while(e.hasMoreElements()){  parameterNames = (String) e.nextElement();  String values[] = request.getParameterValues(parameterName);  } |

d.同c,方法不同

|  |
| --- |
| Map map =request.getParameterMap() // name : value  Set set =map.keySet(); //set为所有参数名称  Iterator it = set.iterator();  String parameterName = null;  while(it.hasNext()){  parameterName=(String)it.next();  String values[] = request.getParameterValues(parameterName);  } |

2、请求方式 request.getMethod()

1)GET请求

get是明文的，上传一些不需保密的数据，不安全，效率高，大小是有限制1024个字节。

2) POST请求

文件上传、用户名和密码等保密性的东西，打包，安全，效率低，对大小没有限制

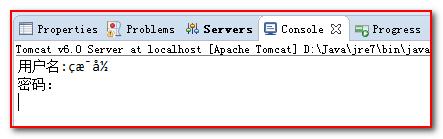
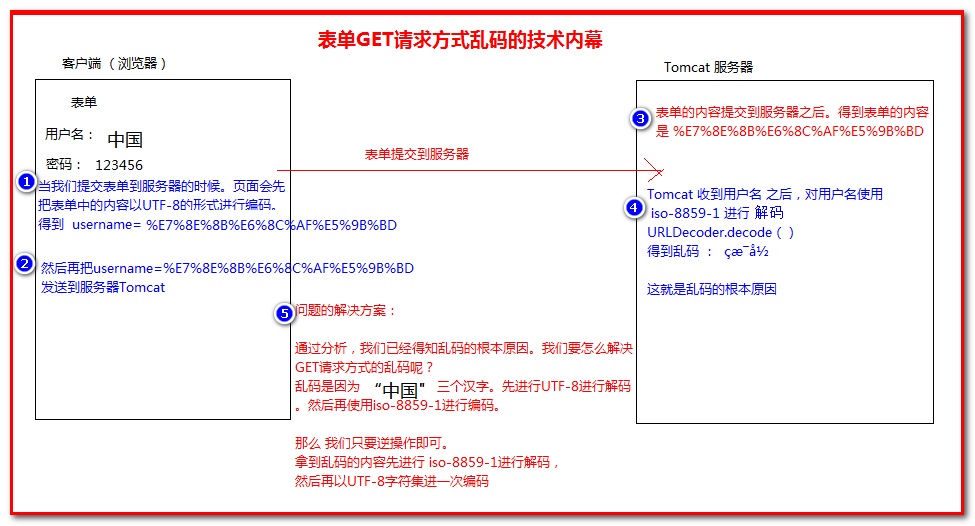
3. 如何处理中文参数

a、Get方式的中文乱码

1) 使用如下页面表单内容：

|  |
| --- |
| <form action="http://127.0.0.1:8080/day07/params" method="get">  用户名：<input name="username" type="text" /><br/>  密　码：<input name="password" type="password" /><br/>  <input type="submit" />  </form> |

2) 获取表单内容代码：  


3) 控制台打印乱码内容：  
   
4) 乱码的根本原因是什么呢？   
  
解决乱码的核心代码：

解决乱码的核心思路，就是把得到的乱码按照原来乱码的步骤逆序操作。

1、先以iso-8895-1进行解码

2、然后再以utf-8进行编码

1) 第一种方式 使用URLEncoder 和 URLDecoder 两个类编解码

|  |
| --- |
| //获取客户端传递过来的用户名参数值  String username = request.getParameter("username");  System.out.println("用户名:" + username);  // 先对用户名进行解码得到%E7%8E%8B%E6%8C%AF%E5%9B%BD 这样的形式  username = URLEncoder.encode(username, "ISO-8859-1");  // 再进行utf-8编码 一次得到页面上输入的文本内容  username = URLDecoder.decode(username, "UTF-8");  System.out.println("乱码解决后用户名：" + username); |

2) 第二种方式使用String类的方法进行编解码

|  |
| --- |
| username = new String(username.getBytes("ISO-8859-1"), "UTF-8");  System.out.println("乱码解决后用户名：" + username); |

Tomcat8的版本中GET基本就不会乱码了，因为服务器对url的编码格式可以进行自动转换

7.2 Servlet的Response

1) 响应内容给客户端

|  |
| --- |
| // 通过response响应对象获取到字符输出流  Writer writer = response.getWriter();  // 往客户端输出数据。  writer.write("this is response content!"); |

2）输出中文到客户端的乱码解决方法

如果拿到writer字符输出流。直接输出中文内容返回到客户端。会得到乱码。



主要是因为服务器输出的字符串的编码和客户端显示字符串的编码不一致。导致乱码问题。

设置服务器的字符串编码

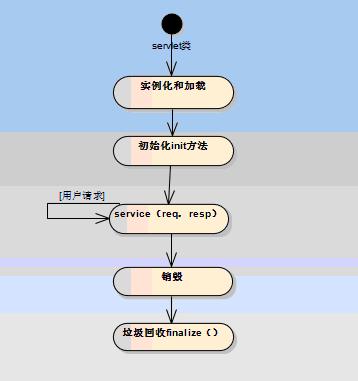
|  |
| --- |
| //设置服务器输出的编码为UTF-8  response.setCharacterEncoding("UTF-8"); |

设置客户端的字符串显示编码。

|  |
| --- |
| //告诉浏览器输出的内容是html,并且以utf-8的编码来查看这个内容。  response.setContentType("text/html;charset=utf-8"); |

这两行语句要在获取输出流之前执行。才会生效。

7.3 Servlet运行步骤



1. 当客户端浏览器输入URL后，请求到了Tomcat服务器容器
2. Tomcat根据web.xml的配置文件的路径，找到了响应的Servlet类的路径
3. Servlet引擎(Tomcat) 加载我们的Servlet类，创建Servlet类的对象，始终只有一个对象
4. Servlet引擎初始化对象 init方法,只会被执行一次
5. 根据客户端浏览器的刷新或提交，反复执行service(request,response)方法,其实服务器是为每个客户端的响应开辟了线程

当Tomcat把该Servlet从容器中移除，就会调用destroy()方法，释放Servlet对象空间，启动了垃圾回收机制

7.4 Servlet路径处理和特性

1. 重定向

什么是重定向

重定向原理和特点

servlet请求转发与重定向的区别：

|  |
| --- |
| //跳转  request.setAttribute("test","hello");  request.getRequestDispacther("/test.jsp").forword(request,response);  //重定向  response.sendRedirect("test.jsp"); |

1. 当用request.getRequestDispacther("/test.jsp").forword(request,response);

请求转发后，结果页面输出：hello

1. 当用response.sendRedirect("test.jsp");重定向后，结果页面输出：null ，重定向到新的Servlet后数据是跟不过去的



从URL地址看：redirect显示重定向的地址，forward显示的是第一次访问的路径

从对象看:response.redirect(url) request.getDispatcher(url).forward(req,resp)

7.8 ServletContext

ServletContext:Servlet上下文

7.8.1 什么是Servlet上下文

Servlet上下文又叫做：ServletContext。

当WEB服务器启动时，会为每一个WEB应用程序(webapps下的每个目录就是一个应用程序)创建一块共享的存储区域

ServletContext也叫做“公共区域”，也就是同一个WEB应用程序中，所有的Servlet和JSP都可以共享同一个区域。

ServletContext在WEB服务器启动时创建，服务器关闭时销毁

容器在启动的时候，会为每一个web应用创建唯一的一个符合ServletContext接口的对象，该对象一般称之为"servlet上下文"。

7.8.2 如何获得Servlet上下文

方式一:GenericServlet提供了getServletContext()方法。

方式二:ServletConfig提供了getServletContext()方法。

方式三:HttpSession提供了getServletContext()方法。

方式四:HttpServletRequest提供了getServletContext()方法。

7.8.3 Servlet上下文的作用及特点

作用：

1、获取真实路径

获取当前项目的发布路径

request.getSelectContext().getRealPath("/");

2、获取容器的附加信息

System.out.println(request.getServletContext().getServerInfo());

System.out.println(request.getServletContext().getContextPath());

3、全局容器

//设置信息到全局容器中

request.getServletContext().setAttribute("msg", "共享信息");

//获取数据

System.out.println(request.getServletContext().getAttribute("msg"));

//移除数据

request.getServletContext().removeAttribute("msg");

特点：

唯一性: 一个应用对应一个servlet上下文。

一直存在: 只要容器不关闭或者应用不卸载，servlet上下文就一直存在。

7.9 Servlet的路径

绝对路径：绝对路径就是你的主页上的文件或目录在硬盘上真正的路径，(URL和物理路径)例如：

C:/xyz/test.txt 代表了test.txt文件的绝对路径。http://www.sun.com/index.htm也代表了一个

URL绝对路径

相对路径：相对与某个基准目录的路径。包含Web的相对路径（HTML中的相对目录），例如：在

Servlet中，"/"代表Web应用的跟目录。和物理路径的相对表示。例如："./" 代表当前目录,

"../"代表上级目录。这种类似的表示，也是属于相对路径

服务器端的相对地址指的是相对于你的web应用的地址，这个地址是在服务器端解析的

（不同于html和javascript中的相对地址，他们是由客户端浏览器解析的）也就是说这时候

在jsp和servlet中的相对地址应该是相对于你的web应用，即相对于http://127.0.0.1:8080/test/的