# 9.Servlet

## 9.1.servlet概述

servlet是什么

Servlet是sun公司提供的一门用于开发动态web资源的技术。

按照这套规范写出来的Servlet可以放置到web应用中在Servlet容器中运行。

开发Servlet步骤

（1）写一个类, 实现Servlet接口, 并实现其中的方法

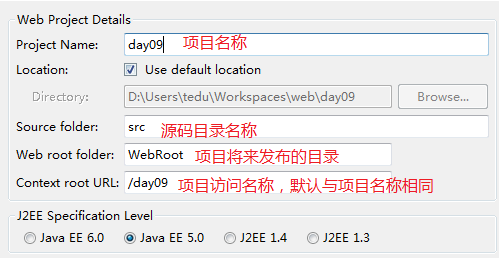
（2）在web.xml中为servlet配置对外访问路径。

## 9.2.使用myeclipse开发servlet

使用myeclipse开发Servlet程序时, 可以新建一个Servlet, 默认继承HttpServlet, 在Servlet内部会覆盖doGet和doPost方法, 分别来处理GET和POST请求。

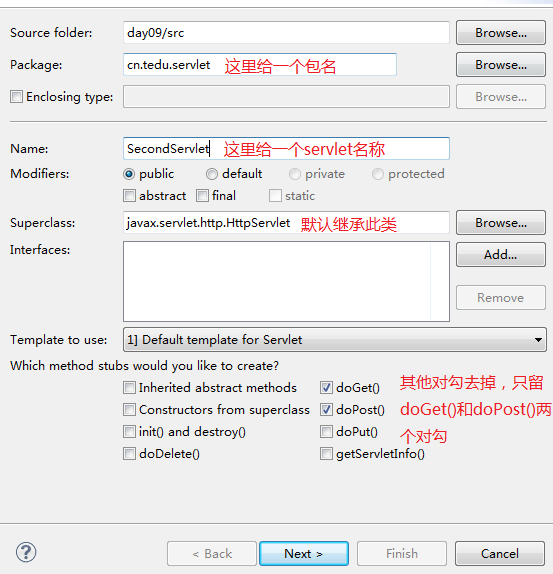
**新建项目**

新建一个Web Project，给一个项目名称，其他保持默认即可，然后finish，弹出的对话框选择yes即可。



**新建一个servlet**

新建一个servlet，如果new中找不到servlet，注意视图是否切换到MyEclipse中，给一个包名和类名，下面对勾只留doGet()和doPost()，其他去掉，下一步



这里将最后两行删掉，是一些xml中的提示，其他地方一般保持默认即可，也可以修改对外访问路径，其中name和URL会自动配置到xml中。最后finash。



将servlet中的注释以及方法中的默认实现全部删掉。

**为什么要继承HttpServlet?**

因为这是一个继承了GenericServlet的类，已经提供doGet()和doPost()方法，可以方便我们开发web项目。

而GenericServlet是一个基础的实现，如果要用此类，需要自己写doGet()和doPost()方法，并且需要在service方法中对请求进行判断，会比较麻烦。

**HttpServlet底层又是如何实现的？**

可以打开HttpServlet源码，发现HttpServlet也是继承自GenericServlet，同时，HttpServlet会写各种方法，比如doGet()和doPost()，然后在service方法中进行判断，不同的请求调用不同的方法。

**注意：复制一个servlet，web.xml中是不会自动生成配置信息的，所以一般不要复制，而是要新建！**

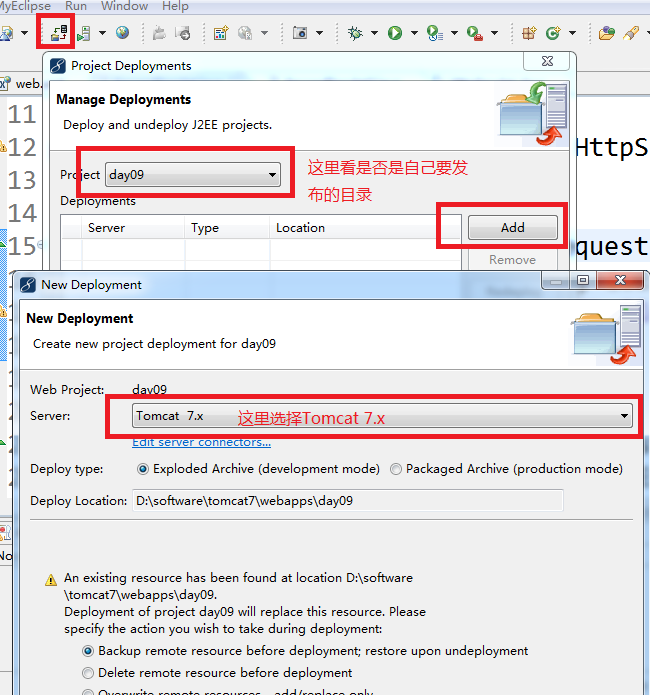
**将tomcat配置到myeclipse中**

**详细步骤见:/resource/在MyEclipse中配置自己安装的tomcat**

如果不配置，则每次都要手动将程序发布到tomcat中。

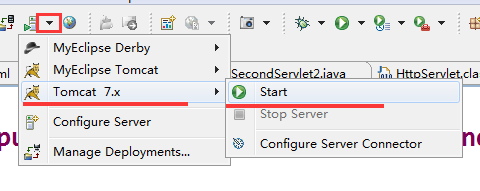
**将web应用发布到自己的tomcat中**

这里注意项目名称是否是自己要发布的项目，然后finish，最后ok。



也可以修改发布的项目名称，参见/resource/在myeclipse中修改web应用发布到Tomcat中的应用的名称.pdf

在MyEclipse中启动tomcat



启动后，可以通过浏览器访问servlet，比如：

http://localhost/day09/servlet/SecondServlet

可以查看servers中tomcat如果是debug模式，则可以修改方法中的代码而无需重启服务器，但是若要新建一个servlet，则需要重启服务器。

## 9.3.Servlet的继承结构

Servlet接口 – 提供了一个Servlet应该具有的最基本的功能

|

|-- GenericServlet类, 实现了Servlet接口, 并实现了其中大部分的方法, 但是service方法没有实现, 这个方法需要开发人员自己去实现

|

|-- HttpServlet类, 继承了GenericServlet, 并实现了service方法, 在service方法中是根据不同的请求方式, 调用不同的doXxx方法, 因 此 我们在开发中, 只需要写一个类, 继承HttpServlet, 并覆盖 doGet()和 doPost()方法分别来处理Get请求和POST请求即可!!

## 9.4.修改servlet模版

方式一：

将\resource\修改Servlet模板\com.genuitec.eclipse.wizards\_9.0.0.me201108091322.jar文件拷贝到：

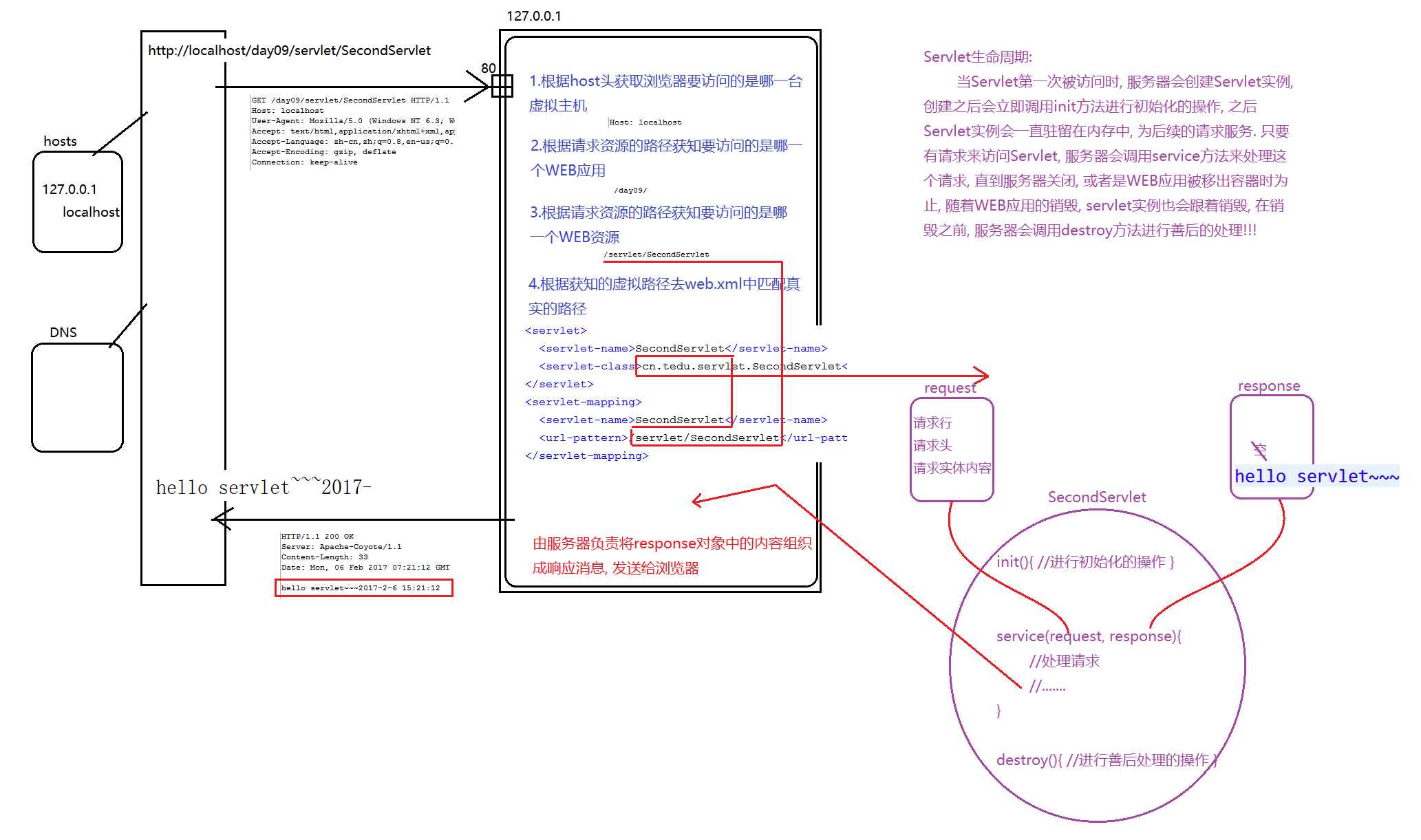
[Myeclipse安装目录]\Common\plugins目录下，会提示是否替换文件，确定替换即可，如果未出现提示，则看是否进对目录，或者是其他版本的MyEclipse

方式二：

1. 在[Myeclipse安装目录]\Common\plugins目录下找到文件：com.genuitec.eclipse.wizards.xxx.jar，在此文件中的Templates目录下可以看到Servlet.java源代码。
2. 打开源代码，将doGet()和doPost()两个方法的注释和方法中的内容删掉，在doPost()中调用doGet()方法即可。
3. 修改之后保存
4. 重新启动Myeclipse即可以使用新的模板代码了

## 9.5.Servlet调用过程

**调用过程**



**Servlet生命周期**

Servlet实例在第一次被访问时创建, 创建之后服务器会立即调用init方法进行初始化的操作, 从此以后该实例会一直驻留在服务器的内存中, 为后续的请求服务, 只要有请求访问servlet, 服务器就会调用service方法来处理这个请求, 直到服务器关闭或者是web应用被移出容器时为止, 随着web应用的销毁, servlet实例也会跟着销毁, 在销毁之前, 服务器就调用destroy方法进行善后的处理.

## 9.6.servlet虚拟路径的配置

在web.xml中的servlet对外访问的虚拟路径的配置, 可以直接写一个路径, 或者通过 \* 号匹配符写一个路径.

方式一：直接写一个路径: /servlet/SecondServlet

方式二：通过\*号匹配符写一个路径:

（1）以 / 开头, 以 /\* 结尾, 如: /servlet/\* /a/\* /\*

（2）以 \*.后缀 的形式, 如: \*.html \*.servlet \*.do \*.action

使用\*号匹配符写路径, 路径的配置变得更加灵活, 但是也可能会造成, 一个url会被多个servlet Mapping所匹配

Url：<http://localhost/day09/servlet/SecondServlet>.do

Servlet1：Test1: /servlet/\*

Servlet2：Test2: \*.do

匹配规则:

**\*.后缀的优先级永远最低!!**

**哪一个更接近哪一个起作用!!**

示例：

Servlet1 映射到 /abc/\*

Servlet2 映射到 /\*

Servlet3 映射到 /abc

Servlet4 映射到 \*.do

当请求URL为“/abc/a.html”，“/abc/\*”和“/\*”都匹配，哪个servlet响应

Servlet引擎将调用Servlet1。

当请求URL为“/abc”时，“/abc/\*”和“/abc”都匹配，哪个servlet响应

Servlet引擎将调用Servlet3。

当请求URL为“/abc/a.do”时，“/abc/\*”和“\*.do”都匹配，哪个servlet响应

Servlet引擎将调用Servlet1。

当请求URL为“/a.do”时，“/\*”和“\*.do”都匹配，哪个servlet响应

Servlet引擎将调用Servlet2。

## 9.7.Request

代表http请求的对象

### 9.7.1.继承结构

ServletRequest – 提供一个request对象最基本的功能

|

|-- HttpServletRequest – 继承了ServletRequest接口, 并在其基础上添加了很多和Http协议相关的方法

### 9.7.2.request的功能

**获取客户端相关的信息**

getRequestURL方法 -- 返回客户端发出请求完整URL

如: http://localhost/day09/servlet/SecondServlet

getRequestURI方法 -- 返回请求行中的资源名部分

如: /day09/servlet/SecondServlet

getQueryString方法 -- 返回请求行中的参数部分

如: username=zhangfei&password=123

getRemoteAddr方法 -- 返回发出请求的客户机的IP地址

如: 127.0.0.1 //可能会出现0:0:0:0:0:0:0:1形式，是ipv6的表现形式。

getMethod -- 得到客户机请求方式

如: GET或POST

!!getContextPath -- 获得当前web应用虚拟目录名称

如: /day09

注意：在写路径时不要将web应用的虚拟路径的名称写死, 应该在需要写web应用的名称的地方通过getContextPath方法动态获取

**获取请求头信息**

getHeader(name)方法 --- String

getHeaders(String name)方法 --- Enumeration<String>

可以通过遍历枚举遍历每一个信息

例如：while (values.hasMoreElements()) {

String value = (String) values.nextElement();

System.out.println(value);

}

getHeaderNames方法 --- Enumeration<String>

getIntHeader(name)方法 --- int

getDateHeader(name)方法 --- long(日期对应毫秒)

**获取请求参数(!!!重要)**

getParameter(String name) --- String 通过name获得值

getParameterValues(String name) --- String[ ] 通过name获得多值 checkbox

getParameterMap() --- Map<String,String[ ]> key :name value: 多值

getParameterNames() --- Enumeration<String> 获得所有name

例如：创建servlet：RequestDemo2，并在webRoot下，将css那天的regist文件夹放进去，并修改form中的属性：

|  |
| --- |
| <!-- <form action="http://localhost/day09/servlet/RequestDemo2"> -->  <!-- 也可以简写为如下格式 -->  <form action="/day09/servlet/RequestDemo2"> |

RequestDemo2中代码如下：

|  |
| --- |
| //getParameter(String name)  String username = request.getParameter("username");  System.*out*.println("username:"+username);  //getParameterValues(String name)  //可以自定义一个网址拼接参数，如：  //http://localhost/day09/servlet/RequestDemo2?like=lanqiu&like=zuqiu&like=ppq  String[] likes = request.getParameterValues("like");  System.*out*.println("likes:"+Arrays.*toString*(likes));  //getParameterMap()获取所有请求参数组成的map集合  //可以自己指定map的类型  Map<String,String[]> map = request.getParameterMap();  **for** (Map.Entry<String,String[]> entry:map.entrySet()) {  String key = entry.getKey();  String[] values = entry.getValue();  System.*out*.println(key+" : "+Arrays.*toString*(values));  } |

请求参数中的乱码问题

乱码分析: 编码时和解码时使用的码表不一致造成的!!

编码: 是在浏览器进行的, 浏览器发送数据使用的是什么编码?

浏览器在打开当前页面时使用的是什么码表,也会使用相同的码表来发送数据.

打开页面使用utf-8, 所以发送数据也是用utf-8码表

解码: 是在服务器端进行的, 服务器端接收数据使用的又是什么编码?

如果不指定, 服务器会使用默认的码表来接收浏览器发送过来的数据, 默认的码表是iso8859-1.

解决方案:

乱码造成的原因是编码不一致, 所以应该让两端的编码保持一致!, 应该通知服务器使用utf-8来接受客户端发送过来的数据!!

request.setCharacterEncoding(“utf-8”);//用来通知服务器使用什么编码来接受请求实体内容中的数据, 如果使用的是POST提交, POST提交的请求参数就是在请求实体内容中!, 所有这个方法可以解决POST提交的乱码问题!!!

GET提交的请求参数由于不在请求实体内容中,而是在请求行中的请求资源路径后面拼接着, 所以这行代码对GET提交的参数乱码不起作用!!!

GET提交的参数乱码问题该如何解决???

GET提交的乱码问题可以通过手动编解码来解决!!

//>>username为乱码, 通过乱码反向编码得回二进制数组

byte[] bytes = username.getBytes("iso8859-1");

//>>通过二进制数组查询正确的码表, 得出正确的数据

username = new String(bytes, "utf-8");

**实现请求转发(!!!重要)**

请求重定向: 302状态码+location响应头

请求转发: 和请求重定向都可以实现资源的跳转, 但是区别是请求转发是服务器内部的并且是同一个WEB应用内部的资源跳转

请求转发的特点:

一次请求对应一次响应

地址栏地址不会发生变化

请求转发只能在同一个WEB应用内部资源之间进行跳转! 不能是不同的WEB应用或者不同的主机!

实现代码:

|  |
| --- |
| 创建servlet：RequestDemo3和RequestDemo4，RequestDemo3转发到RequestDemo4  在RequestDemo3中：  /\*  \* 在web阶段写路径时，除了请求转发和请求包含在写  \* 路径是不用包含web应用的虚拟路径，其他地方都需要  \* 加上web应用的虚拟路径。  \* 完整路径：http://localhost/day09/servlet/RequestDemo4  \*/  request.getRequestDispatcher("/servlet/RequestDemo4").forward(request, response);  在RequestDemo4中响应：  response.getWriter().write("1$"); |

request开发细节:

在转发之前, 如果response缓冲区被写入了数据但是还没有打给浏览器, 在转发时response缓冲区(数据)将会被清空!

例如：在RequestDemo3中添加代码：

response.getWriter().write("no money!");

发现浏览器中并未得到"no money!"这段字符串的响应。

在转发之前, 如果response缓冲区被写入了数据并且已经打给了浏览器, 转发将会失败!!

例如：在RequestDemo3中添加代码：

response.getWriter().write("no money!");

response.flushBuffer();

发现浏览器可以显示"no money!"，因为强制刷新了，但是转发就会报错，因为已经响应过了，一次请求对应一次响应。

在同一个Servlet中转发不能进行多次!!(A既转发B, 又转发给C)

但是可以进行多重转发(比如A转发给B, B再转发给C)

**作为域对象来使用(!!!重要)**

域对象:如果一个对象具有一个可以被看见的范围, 利用该对象上的map可以在整个范围内实现资源的共享!

域对象提供的方法(可以操作map中的数据):

setAttribute(String name, Object value); 用来存储一个对象，也可以称之为存储一个域属性

getAttribute(String name); 用来获取request中的数据

removeAttribute(String name); 用来移除request中的域属性

getAttributeNames(); 获取所有域属性的名称

生命周期:

一次请求开始时创建request对象, 一次请求结束时销毁request对象

作用范围:

整个请求链

主要功能:

在整个范围内共享数据

带数据到目的地

例如：在RequestDemo3中可以设置一些属性，比如是从数据库查询出来的数据，转发到RequestDemo4后，在RequestDemo4中从request域中获取数据，并打印到页面中。

**实现请求包含**

请求包含是服务器内部资源合并的现象

如果浏览器访问Servlet A, 但是A不能独立的处理这次请求, 需要另外一个Servlet B帮忙, 于是在A中可以将B包含进来, 包含的代码如下:

request.getRequestDispatcher(“B的路径”).include(request, response);

将B包含进来后, 将会由A和B共同来处理这次请求, 处理的结果也会合并在一起, 一起打给浏览器!

## 9.8.Response

### 9.8.1.response概述

代表http响应的对象

**继承结构**

ServletResponse -- 通用的接口, 定义了一个response对象应该具有的功能

|

|--HttpServletResponse 在ServletResponse基础上, 增加很多和Http协议相关的方法

### 9.8.2.response重要方法

Http响应消息:

一个状态行: HTTP/1.1 200 OK

若干响应头:

xxx

...

xxx

(一个空行)

响应实体内容: xxxxx

设置状态码的方法

**!!void setStatus(int sc)**

void setStatus(int sc, String sm)

设置响应头的方法

**!!void setHeader(String name, String value)**

void setDateHeader(String name, long date)

void setIntHeader(String name, int value)

void addHeader(String name, String value)

void addDateHeader(String name, long date)

void addIntHeader(String name, int value)

注：set为设置，当本来存在这个头时是修改，add只是添加。

设置响应实体内容的方法

**!!ServletOutputStream getOutputStream()**

**!!PrintWriter getWriter()**

### 9.8.3.response对象功能

**向客户端发送数据 (!!!重点, 特别是乱码问题)**

getOutputStream() //字节流

getWriter() //字符流

**（1）字节流发送数据的中文乱码问题？**

服务器端指定了用utf-8来发送数据, 浏览器在接受数据时, 如果不指定将使用默认的平台码GBK, 编解码不一致导致乱码.

**解决方案:**

response.setHeader("Content-Type", "text/html;charset=utf-8");// 通知浏览器使用utf-8打开服务器发送过去的数据

response.getOutputStream().write("中国".getBytes("utf-8"));// 指定编码为utf-8

response.setHeader("Content-Type", "text/html;charset=utf-8");

等价于 <=======>

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

**（2）字符流发送数据的中文乱码问题？**

利用字符流发送数据, 底层还是要转成字节. 服务器端如果不手动指定, 服务器默认会使用iso8859-1码表, 由于里面没有中文汉字, 所以服务器端发送给客户端就是一堆乱码, 客户端不管使用什么码表都无法转成正常的字符

\* 服务器会根据getCharacterEncoding()方法返回的编码来发送数据, 如果没有指定, 该方法默认返回iso8859-1

**解决方案：**

第一步, 需要指定服务器发送数据使用utf-8

response.setCharacterEncoding("utf-8"); //通知服务器使用utf-8来发送响应实体中数据

第二步: 需要指定浏览器在接收数据时也使用同一个编码来打开数据

response.setHeader("Content-Type", "text/html;charset=utf-8");

等价于<==>

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

需要注意的是：在通知浏览器使用什么编码接受服务器发送的数据时, 服务器很智能, 会使用相同的编码来发送数据, 所以指定服务器以什么编码发送数据的代码可以省略不写

**总结: 不管是字符流还是字节流, 解决乱码问题, 可以用一行代码搞定:**

response.setContentType("text/html;charset=xxx");

**response开发细节**

\*\*getOutputStream()和getWriter() 这两个方法是互斥的, 在一次请求当中调用了其中的一个, 就不能再调用另一个!!!

\*\*在调用完getOutputStream()或getWriter()方法之后, 不需要手动去关闭流, 服务器会自动帮我们去关闭!!!

\*\*这两个方法获取到的流并不是指向客户端的流, 而是指向response缓冲区的流, 通过流将数据写入response缓冲区, service方法执行结束, 请求回到服务器, 由服务器将数据组织成响应消息打给浏览器!!

**代码示例：**

|  |
| --- |
| **创建web项目day10，并创建servlet：ResponseDemo1，在doGet中添加如下代码：**  //getOutputStream() -- 字节流  //response.getOutputStream().write("hello response...".getBytes());  //getBytes()得到一个操作系统默认的编码格式的字节数组，默认为gbk，而浏览器默认解码也是gbk  //response.getOutputStream().write("中国".getBytes());  //如果明确指定utf-8格式的字节数组，则会在浏览器出现乱码  //response.getOutputStream().write("中国".getBytes("utf-8"));    //通知浏览器接收数据也用utf-8编码  //response.setHeader("Content-Type", "text/html;charset=utf-8");  //response.setContentType("text/html;charset=utf-8");    //getWriter() -- 字符流  //response.getWriter().write("hello response...");  //服务器发送数据用的是iso8859-1，而浏览器解析是gbk。  //需要通知服务器用utf-8来发送数据，但是这部可以省略，服务器会根据浏览器接收的编码发送数据  //response.setCharacterEncoding("utf-8");  //还需要通知浏览器用utf-8来接收数据  response.setContentType("text/html;charset=utf-8");  response.getWriter().write("中国"); |

### 9.8.4.实现重定向

可以通过302状态码加上location响应头实现请求重定向

response.setStatus(302);

response.setHeader(“location”, “/day10/index.jsp”);

上面两行代码等价于下面这一行<===>

response.sendRedirect(“/day10/index.jsp”);

重定向的特点:

两次请求, 两次响应

地址栏地址会发生变化

既可以实现在同一个WEB应用内部资源之间进行跳转, 也可以在不同的WEB应用或者是不同的服务器资源之间进行跳转

由于是两次请求，两次响应，无法通过request对象共享数据

**代码示例：**

|  |
| --- |
| **创建servlet：ResponseDemo2，在doGet中添加如下代码：**  //请求重定向  //第一种方式  //response.setStatus(302);  //重定向到index.jsp，路径为http://localhost/day10/index.jsp  //可以简写为/day10/index.jsp，其中day10动态获取  //response.setHeader("location", request.getContextPath()+"/index.jsp");    //第二种方式  //response.sendRedirect( request.getContextPath()+"/index.jsp");    //不同资源之间跳转  //response.sendRedirect("/day09/index.jsp");  response.sendRedirect("http://www.tmooc.cn"); |

### 9.8.5.定时刷新

与重定向类似，不同之处就是可以指定几秒之后跳转。

可以通过refresh头实现在多少秒之后跳转指定的资源.

代码: response.setHeader(“refresh”, “3;url=/day10/index.jsp”);

定时刷新的特点:

两次请求, 两次响应

地址栏地址会发生变化

既可以实现在同一个WEB应用内部资源之间进行跳转, 也可以在不同的WEB应用或者是不同的服务器资源之间进行跳转

**代码示例：**

|  |
| --- |
| **创建servlet：ResponseDemo3，在doGet中添加如下代码：**  //定时刷新  //解决乱码问题  response.setContentType("text/html;charset=utf-8");  //提示用户3秒之后跳转  response.getWriter().write("恭喜您注册成功，3秒之后跳转到首页。");    //通过定时刷新可以实现多少秒之后跳转到指定资源  response.setHeader("refresh", "3;url="+request.getContextPath()+"/index.jsp"); |

### 9.8.6.控制浏览器的缓存行为

由于不同的浏览器的缓存行为可能是不同的, 我们可以在服务器中通过设置响应头来控制浏览器的缓存行为.

控制浏览器不要缓存:

setDateHeader("Expires", -1);

setHeader("Cache-control", "no-cache");

setHeader("Pragma", "no-cache");

控制浏览器缓存:

setDateHeader("Expires", System.currentTimeMillis()+1000\*60\*60\*24);

或

setHeader("Cache-control", "max-age=60"); //优先级更高

**代码示例：**

|  |
| --- |
| **创建servlet：ResponseDemo4，在doGet中添加如下代码：**  //控制浏览器是否缓存    //控制浏览器缓存一个资源  //response.setDateHeader("Expires", System.currentTimeMillis()+10000);  //response.setHeader("cache-control", "max-age=5");    //控制浏览器不要缓存一个资源  response.setDateHeader("Expires", -1);  response.setHeader("cache-control", "no-cache");    response.getWriter().write("hello demo4..."+**new** Date().~~toLocaleString~~()); |

## 9.9.资源跳转总结

**请求转发/请求重定向/定时刷新都可以实现资源的跳转, 区别是什么呢?**

**请求转发:**

一次请求,一次响应 request对象是同一个

地址栏不会发生变化

只能用于服务器内部的资源跳转, 并且只能是同一应用中的不同资源上进行跳转, 不可用在不同应用和不同服务器中的资源跳转

**请求重定向:**

两次请求,两次响应 request对象不是同一个

地址栏会发生变化

可以用于服务器内部的资源跳转, 也可以用于不同应用和不同服务器之间的资源跳转

**定时刷新:**

两次请求,两次响应 request对象不是同一个

地址栏会发生变化

可以用于服务器内部的资源跳转, 也可以用于不同应用和不同服务器之间的资源跳转

和重定向不同的是, 定时刷新可以在刷新到新的地址之间设置一个时间, 在间隔的这段时间内可以输出文本到浏览器并维系一段时间

**那什么时候用哪种方式进行资源的跳转呢?**

**(1)如果是同一服务器中的同一应用内部的资源跳转:**

如果在跳转时, 需要通过request对象带数据到目的地, 只能用请求转发

如果在跳转时, 希望地址栏地址不要发生变化, 只能用请求转发

如果在跳转时, 希望地址栏地址发生变化, 只能使用重定向或定时刷新

如果没有什么具体需要, 三种方式都可以, 最好使用请求转发. 可以减少访问服务器的次数, 降低服务器的压力!

**(2)如果是不同服务器或不同WEB应用内部的资源跳转, 只能用重定向或者定时刷新:**

由于重定向是立即跳转, 而定时刷新是在多少秒之后再进行跳转. 并且可以在跳转之前的时间里输出文本数据到浏览器并维系一段时间.

如果跳转时, 需要指定多少秒或者是需要在跳转之前发送数据到浏览器, 只能使用定时刷新, 否则两种方式都可以!

## 9.10.作业练习

1. Servlet是什么？

答：Servlet是sun公司提供的一门用于开发动态web资源的技术。按照这套规范写出来的Servlet可以放置到web应用中在Servlet容器中运行。

1. 说明Servlet接口中方法及方法的作用？

答：public void init(ServletConfig config) -- servlet创建时执行，初始化方法

public ServletConfig getServletConfig() -- 获取ServletConfig对象的方法

public String getServletInfo() -- 提供servlet的描述信息，如作者、版本、版权

public void service(ServletRequest request,ServletResponse response) -- 是servlet的核心，被Servlet容器调用，可以接收请求和响应请求

public void destroy() -- servlet销毁时执行，进行善后操作

1. 简述Servlet生命周期？

答：Servlet实例在第一次被访问时创建, 创建之后服务器会立即调用init方法进行初始化的操作, 从此以后该实例会一直驻留在服务器的内存中, 为后续的请求服务, 只要有请求访问servlet, 服务器就会调用service方法来处理这个请求, 直到服务器关闭或者是web应用被移出容器时为止, 随着web应用的销毁, servlet实例也会跟着销毁, 在销毁之前, 服务器就调用destroy方法进行善后的处理.

1. 描述Servlet调用过程？

答：1）在浏览器输入地址，浏览器先去查找hosts文件，将主机名翻译为ip地址，如果找不到就再去查询dns服务器将主机名翻译成ip地址。

（2）浏览器根据ip地址和端口号访问服务器，组织http请求信息发送给服务器。

（3）服务器收到请求后首先根据Host请求头判断当前访问的是哪台虚拟主机。

（4）服务器根据http请求头中的请求URI判断当前访问是哪个web应用。

（5）服务器根据http请求头中的请求URI判断当前访问是哪个web应用中的哪个web。

（6）检查web应用的web.xml文件，如果根据路径找到具体的servlet处理类的全部路径交给该servlet处理。

（7）这个过程中浏览器只知道自己发出来http请求，不久就收到了http响应，浏览器不知道也不关心服务器内部是如何处理的。浏览器和服务器之间的关系是非常单纯的，只有HTTP协议。

（8）解析请求、封装Request、Response对象、创建Servlet、调用Servlet方法都是服务器自动进行的，开发人员只需要写好Servlet、调用Service方法都是服务器自动进行的，我们只需要写好Servlet配置进容器中即可。

5.request对象能获取哪些客户机相关的信息？

答：getRequestURL方法返回客户端发出请求完整URL

getRequestURI方法返回请求行中的资源名部分

getQueryString方法返回请求行中的参数部分

getRemoteAddr方法返回发出请求的客户机的IP地址

getMethod得到客户机请求方式

getContextPath获得当前web应用虚拟目录名称

6.请求乱码产生的原因？

答：浏览器在打开当前页面时使用的是什么码表,也会使用相同的码表来发送数据. 打开页面使用utf-8, 所以发送数据也是用utf-8码表。如果不指定, tomcat服务器会使用默认的码表来接收浏览器发送过来的数据, 默认的码表是iso8859-1.。

1. 如何来处理get请求产生的乱码?

答：需要手动编解码：str = new String(str.getBytes("iso8859-1"), "utf-8");

1. 怎样把GB2312的编码的字符串转成IOS8859-1编码的字符串？

答：str = new String(str.getBytes("GB2312"), "iso8859-1");

1. Request生命周期

答：一次请求开始时创建request对象, 一次请求结束时销毁request对象

1. 如何处理响应乱码？

答：不管是字符流还是字节流, 解决乱码问题, 可以用一行代码搞定:

response.setContentType("text/html;charset=xxx");

1. Response向外输出数据有哪些方法？

答：字节流：ServletOutputStream getOutputStream()

字符流：PrintWriter getWriter()

输出中文时都需要注意编码问题。

1. 转发与重定向的比较？

答：请求转发:

一次请求,一次响应，request对象是同一个，可以使用request域共享数据

地址栏不会发生变化

只能用于服务器内部的资源跳转, 并且只能是同一应用中的不同资源上进行跳转, 不可用在不同应用和不同服务器中的资源跳转

请求重定向:

两次请求,两次响应，request对象不是同一个，不能使用request域共享数据

地址栏会发生变化

可以用于服务器内部的资源跳转, 也可以用于不同应用和不同服务器之间的资源跳转

1. 简述forward()与sendRedirect()的区别？

答：sendRedirect()方法会创建一个新的请求，是重定向，而 forward()方法只是把请求转发到一个新的目标上，是转发。（参见上一个问题）