# Request对象

代表Http请求的对象

浏览器向服务器发送一个请求, 服务器在调用Servlet的service方法处理请求之前, 会创建代表Http请求的request对象, 将所有的请求信息封装在request对象中.

## Request的继承结构

ServletRequest – 通用的Request接口, 提供了一个Request对象应该具有的功能

|

|-- HttpServletRequest接口 – 继承了ServletRequest接口, 在父接口的基础上, HttpServletRequest增加了一些和Http协议相关的方法.

## Request对象提供的功能

### 获取客户端基本信息、获取请求头信息

1.**getRequestURL**方法 – 用于获取发送请求的完整url



2.**getMethod**方法 – 用于获取发送请求时使用的请求方式, 这里只用GET和POST提交

3.**getContextPath**方法 – 用于获取WEB应用对外访问的虚拟路径, 这个方法非常重要

在开发中, 当需要书写WEB应用的虚拟路径时, 最好不要写死(比如:/day14), 而是通过getContextPath方法来动态获取.

4.**getHeader**(String name)

host: localhost

### 获取请求参数(!!!)

getParameter(String name) – 通过请求参数的名字获取对应的参数值

getParameterValues(String name);

getParameterMap();

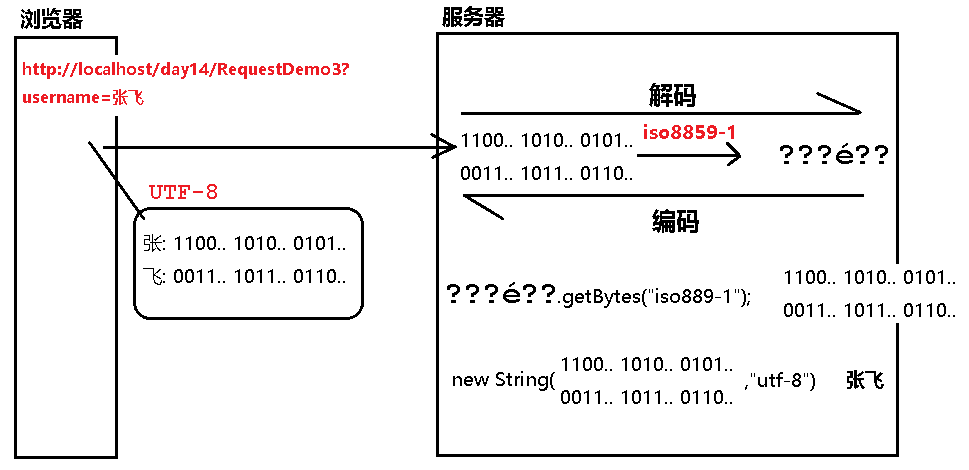
getParameterNames();

* **请求参数的乱码问题?**

问题描述: 在通过request对象获取请求参数时, 如果参数中有中文数据, 在获取时就会出现乱码, 其原因是由于浏览器发送参数时(编码时)使用的码表 和 服务器在结束数据时(解码时)使用的码表不一致造成的!

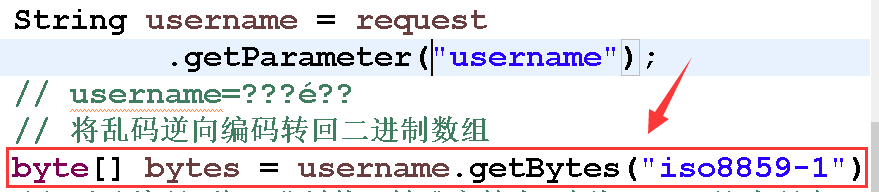
1. 浏览器发送数据时使用的码表: 浏览器在发送数据时使用的码表取决于打开页面时使用的码表, 即浏览器使用什么编码打开页面, 就会使用相同的编码来发送数据. 这里使用的是utf-8这个码表!
2. 服务器接收数据时使用的码表: 服务器在接收数据时默认使用iso8859-1来接收请求参数

解决方案:让编码时使用的码表和解码时使用的码表保持一致即可!!!

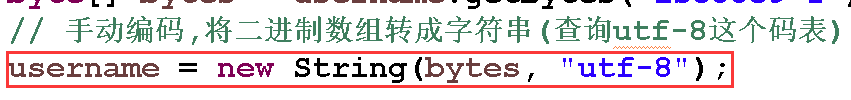


可以通过手动编解码来解决乱码!!

1. 根据获取到的乱码字符串, 反向编码得回正确的二进制数据



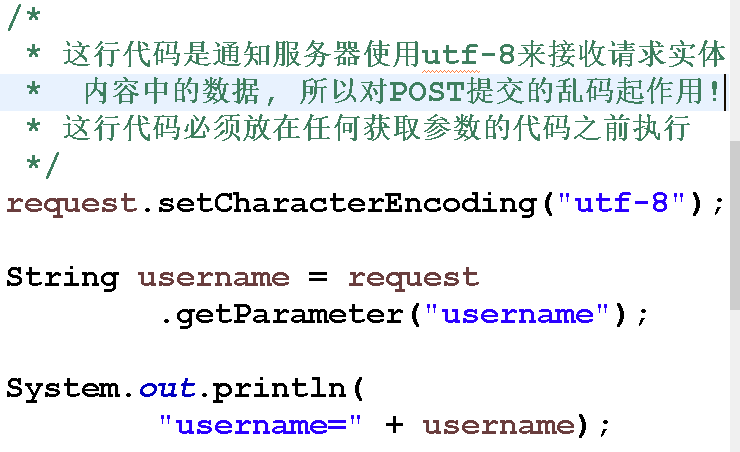
1. 将二进制数组查询utf-8码表转成字符串, 得回正确的数据



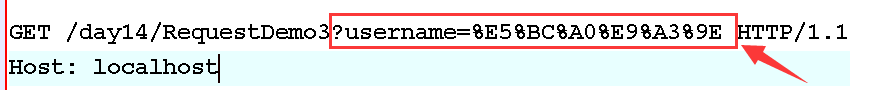
如果发送请求的方式**是POST提交**, 还可以通过request提供的方法来解决乱码:

request.setCharacterEncoding(“utf-8”); 这行代码是通知服务器使用utf-8来**接收请求实体内容**中的数据.

由于POST提交的参数刚好在请求实体内容中, 所以这行代码可以解决POST提交时产生乱码问题.



但是由于GET提交的参数在请求行中的请求资源路径后面拼接着, 不在请求实体内容中, 所有这行代码对GET提交产生的乱码不会起作用!



### 实现请求转发(!!!)

请求重定向: 实现资源的跳转

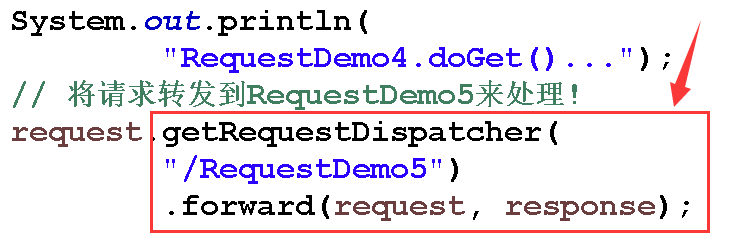
请求转发: 实现资源的跳转, 并且是服务器内部的同一个WEB应用内部资源的跳转

实现请求转发:

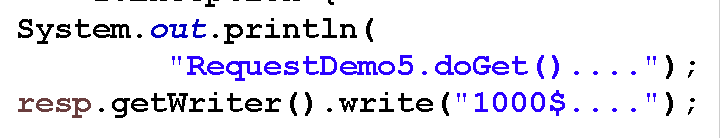
request.getRequestDispatcher(“转发到资源的url地址”).forward(request, response);

* **请求转发示例**

1. 创建一个RequestDemo4 servlet, 接收客户端浏览器的请求, 将请求转发给RequestDemo5来处理

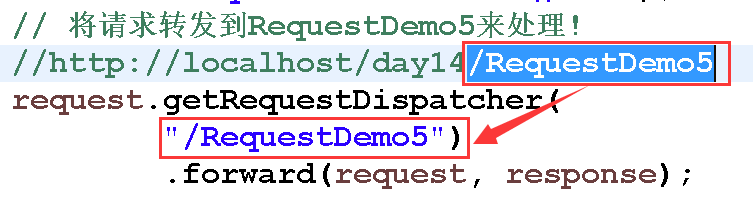


1. RequestDemo5处理请求, 将结果响应给浏览器.

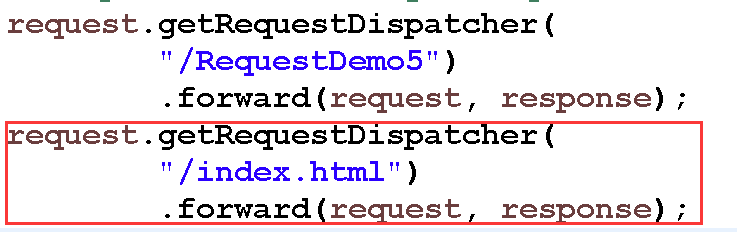


* **请求转发的细节问题:**

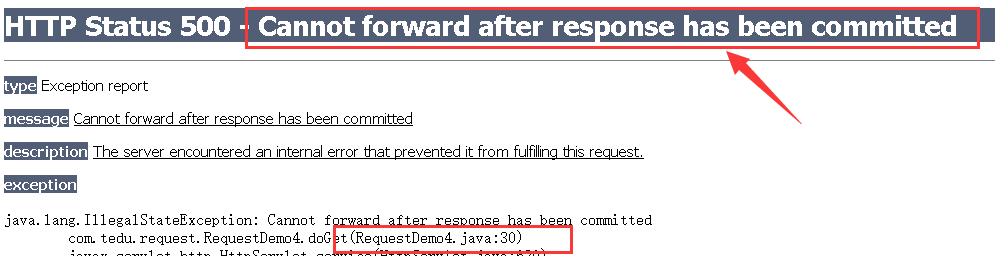
1. 在转发时, 转发时书写资源路径是WEB应用的虚拟路径后面的路径



1. 在转发时, 需要将代表请求的request对象和代表响应的response对象作为forward方法的参数传递给被转发的资源. 原因是不管转发多少次, 这些资源处理的请求始终是同一个, 服务器针对同一个请求, 只会创建一个request对象和一个response对象, 后面的资源在处理请求时又需要这两个对象, 所有需要不断的传递这两个对象.
2. 转发可以实现资源的跳转, 但是在同一个Servlet中, 不能转发多次

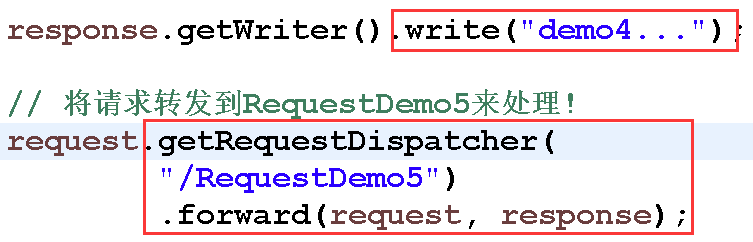


如果转发了多次, 将会报如下错误:

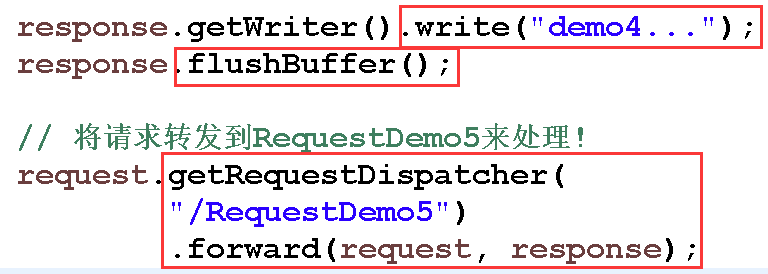


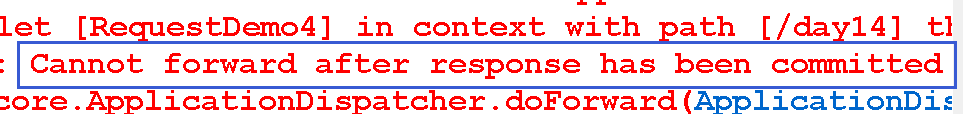
转发不能进行多次, 但是可以多重转发(比如: Demo4转发给Demo5, Demo5处理不了转发给Demo6, Demo6转发给Demo7….)

1. 如果在转发之前, 向response缓冲区写入了数据, 在转发时response中的数据会被清空(即下图中的”demo4…”内容将不会发送给浏览器)



1. 如果在转发之前, 向response缓冲区写入了数据并且强制性发送给了浏览器, 这时转发将会失败!!





### 作为域对象使用(!!!)

域对象: 如果一个对象, 具有一个可以被看见的范围,利用这个对象上的map集合, 可以在这个范围内, 实现数据的共享.

request对象就是一个域对象, request上提供了很多操作自身map集合的方法

request.setAttribute(String name, Object value);//存入一个属性/修改已有属性的值

request.getAttribute(String name); //根据属性名获取对应的属性值

request.removeAttribute(String name);//根据属性名删除对应的属性

request.getAttributeNames();//获取域对象中所有属性的名字组成的枚举.

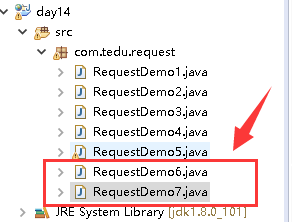
生命周期: 一次请求开始创建request对象, 响应结束request对象就销毁了

作用范围: 整个请求链中

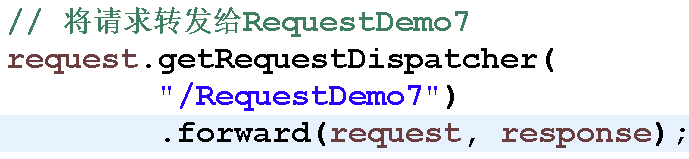
主要功能: 在整个范围内实现数据的共享(带数据到目的地)

需求: 创建两个Servlet: 分别是RequestDemo6和RequestDemo7, 在RequestDemo6中将请求转发给RequestDemo7, 在转发时负责带参数到RequestDemo7中并接收打印在控制台上.

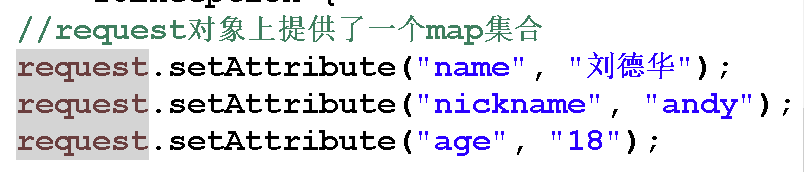
1. 创建两个Servlet



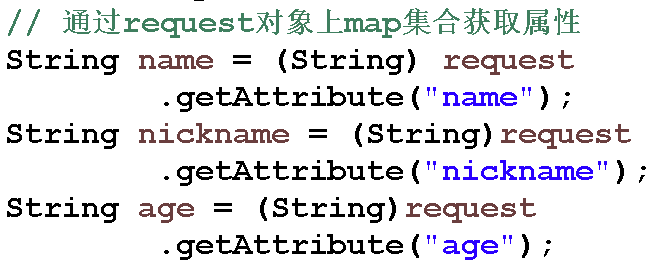
1. 在RequestDemo6中将请求转发给RequestDemo7



1. 在RequestDemo6负责将参数带给RequestDemo7\



1. 在RequestDemo7获取参数并打印在控制台上



### 实现请求包含

请求包含是服务器内部资源合并的效果

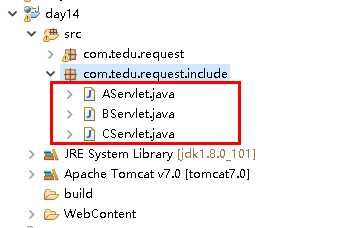
假设浏览器发送请求访问服务器中的 Aservlet, Aservlet在处理请求的期间, 发现自己不能够独立的处理这次请求, 需要另外一个Servlet过来帮忙, 这时可以在Aservlet中将Bservlet包含进来, 然后由Aservlet和Bservlet共同来处理这次请求, Aservlet处理的结果将会和Bservlet处理的结果合并在一起, 一起响应给浏览器.

实现请求包含:

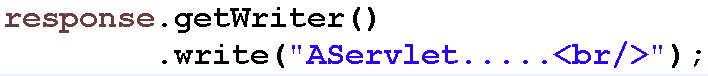
request.getRequestDispatcher(“/Bservlet”).include(request, response);

示例: 编写三个Servlet, 分别为Aservlet, Bservlet, Cservlet, 实现Servlet中的doGet方法来处理请求, 在Bservlet中将Aservlet和Cservlet包含进来, 共同来处理请求.

1. 创建三个Servlet



1. 编写三个Servlet中的doGet方法, 处理请求



1. 在Bservlet中将Aservlet和Cservlet包含进来.



1. 测试: 访问Bservlet, 查询效果

