前篇说到了Response容器对象，这篇我们就来看一下Request容器对象，之前也说过了，这个两个容器对象是相对应的，每次用户请求服务器的时候web容器就会给创建这对容器对象，他们是共存亡的，当然Request除了有一个容器对象的角色，他还有一个角色就是Request域，我们之前在讲解Servlet的时候，说到一个ServletContext域，这个域的范围是整个web应用，这里的Request域的范围就小了，他只是一次用户的请求内，即用户发送一个请求的时候，Request创建，当用户关闭这次请求的时候Request就会消亡的。

下面就来看一下Request的相关方法：

****getContextPath()****：这个方法返回的是web应用映射的虚拟目录地址：如ServletDemo应用的虚拟目录是：/ServletDemo

****getCookies()****：这个方法返回的是一个Cookies[]，我们在response容器那篇文章中看一个方法是向response容器中存入一个cookie的，这个方法是从Request容器中拿取多个cookies,因为用户在请求的时候会携带很多的cookie，关于Cookie的相关知识，我们会在后面的文章中进行详解

****getHeader(String name)/getIntHeader(String name)/getDateHeader(String name)****：这些方法是获取请求头信息的，只是针对不同的类型的，有字符串类型的，时间类型，数值类型的

****getHeaderNames()****：这个方法是获取所有请求头的字段名称

****getHeaders(String name)****：这个方法是获取一个请求头字段的所有值，因为有时候可能会有相同请求头字段信息，不会覆盖的

****getMethod()****：这个方法是获取客户机的请求方法

****getQueryString()****：这个方法是获取用户请求时的查询参数的，即url后面携带的参数，如：http://localhost:8080/ServletDemo/ServletRequest?username=jiangwei&password=123456,那么getQueryString()方法返回来的值就是username=jiangwei&password=123456

****getRequestSessionId()****：这个方法是获取客户机在请求的时候携带的sessionid值，有关session的相关知识，后面会详解

****getRequestURL()****：这个方法是获取客户机请求的url

****getServletPath()****：这个方法返回的是请求的Servlet的映射路径,比如：ServletRequest映射的是是/ServletRequest

****getServerName()/getServerPort()****：这两个方法是获取服务器的名称和端口号，比如localhost,8080

****getSession()/getSession(boolean mode)****：这两个方法是获取一个session对象，相关之后在session篇会说到

****getAttribute(String name)****：这个方法是从Request域中获取值

****getAttributeNames()****：这个方法是获取Request域中获取所有的字段名称

****getParameter(String name)****：这个方法是获取用户使用get/post方式携带的参数值

****getParameterNames()****：这个方法是获取用户请求时携带的所有参数名称

****getParameterMap()****：这个方法是获取用户请求时携带的参数名称和参数值，并将其组装成一个Map对象

****getParameterValues()****：这个方法是获取用户请求携带的参数值，因为有时候一个参数名称可能对应多个值

****setAttribute(String name,Object value)****：这个方法是设置Request域中的属性值

****removeAttribute(String name)****：这个方法是删除Request域中的属性值

****getInputStream()/getReader()****：这个方法是获取用户请求的时候上传的输入流，比如我们在处理用户上传文件的时候。需要用到这个输入流

****setCharacterEncoding(String name)****：这个方法是设置Request容器的编码

****getRemoteAddr()/getRemoteHost()****：获取客户机的IP地址和主机名

****getProctocol()****：获取协议名称

****getRequestDispatcher(String path)****：获取一个转发对象RequestDispatcher，进行转发操作

下面就通过实例来看一下上面方法的使用：

****[java]**** [view plain](http://blog.csdn.net/jiangwei0910410003/article/details/22925915" \o "view plain) [copy](http://blog.csdn.net/jiangwei0910410003/article/details/22925915" \o "copy)

1. **public** **void** test1(HttpServletRequest request) **throws** Exception{
2. //有时候可能有多个name
3. String[] nameAry = request.getParameterValues("username");
4. //在获取用户的请求数据的时候先要进行判断数据的有效性，然后再去使用，提高应用的健壮性
5. **if**(nameAry != **null**){
6. System.out.println("getParameterValues方法");
7. System.out.println("---------------------");
8. System.out.println("参数名称:username");
9. **for**(**int** i=0;i<nameAry.length;i++){
10. System.out.println(nameAry[i]+",");
11. }
12. System.out.println();
13. }
15. System.out.println("getParameterMap方法");
16. System.out.println("------------------");
17. Map<String,String[]> map = request.getParameterMap();
18. **if**(map != **null**){
19. Set<Entry<String,String[]>> set = map.entrySet();
20. Iterator<Entry<String,String[]>> iterator = set.iterator();
21. **while**(iterator.hasNext()){
22. Entry<String,String[]> entry = iterator.next();
23. System.out.println("参数名:"+entry.getKey());
24. String[] values = entry.getValue();
25. **if**(values != **null**){
26. **for**(**int** i=0;i<values.length;i++){
27. System.out.print(values[i]+",");
28. }
29. System.out.println();
30. }
31. }
32. }
33. System.out.println();
35. System.out.println("getParameterNames()方法");
36. System.out.println("----------------------");
37. Enumeration names = request.getParameterNames();
38. **if**(names != **null**){
39. **while**(names.hasMoreElements()){
40. String name = (String) names.nextElement();
41. System.out.println("参数名:"+name);
42. System.out.println("参数值:"+request.getParameter(name));
43. }
44. }
46. }

public void test1(HttpServletRequest request) throws Exception{

*//有时候可能有多个name*

String[] nameAry = request.getParameterValues("username");

*//在获取用户的请求数据的时候先要进行判断数据的有效性，然后再去使用，提高应用的健壮性*

if(nameAry != null){

System.out.println("getParameterValues方法");

System.out.println("---------------------");

System.out.println("参数名称:username");

for(int i=0;i<nameAry.length;i++){

System.out.println(nameAry[i]+",");

}

System.out.println();

}

System.out.println("getParameterMap方法");

System.out.println("------------------");

Map<String,String[]> map = request.getParameterMap();

if(map != null){

Set<Entry<String,String[]>> set = map.entrySet();

Iterator<Entry<String,String[]>> iterator = set.iterator();

while(iterator.hasNext()){

Entry<String,String[]> entry = iterator.next();

System.out.println("参数名:"+entry.getKey());

String[] values = entry.getValue();

if(values != null){

for(int i=0;i<values.length;i++){

System.out.print(values[i]+",");

}

System.out.println();

}

}

}

System.out.println();

System.out.println("getParameterNames()方法");

System.out.println("----------------------");

Enumeration names = request.getParameterNames();

if(names != null){

while(names.hasMoreElements()){

String name = (String) names.nextElement();

System.out.println("参数名:"+name);

System.out.println("参数值:"+request.getParameter(name));

}

}

}

这里我还需要设计一个demo.jsp，在里面设计一个表单进行数据的上传：

****[html]**** [view plain](http://blog.csdn.net/jiangwei0910410003/article/details/22925915" \o "view plain) [copy](http://blog.csdn.net/jiangwei0910410003/article/details/22925915" \o "copy)

1. **<form** action="/ServletDemo/ServletRequest" method="post"**>**
2. 用户名1：**<input** type="text" name="username"**/><br/>**
3. 用户名2:**<input** type="text" name="username"**/><br/>**
4. 密码：**<input** type="text" name="password"**/><br/>**
5. **<input** type="submit" value="提交"**/>**
6. **</form>**

<form action="/ServletDemo/ServletRequest" method="post">

用户名1：<input type="text" name="username"/><br/>

用户名2:<input type="text" name="username"/><br/>

密码：<input type="text" name="password"/><br/>

<input type="submit" value="提交"/>

</form>

我们传递了两个参数名称为：username的字段，我们在浏览器中输入：http://localhost:8080/ServletDemo/demo.jsp,然后打印结果：

getParameterValues方法  
---------------------  
参数名称:username  
aaa,  
bbb,  
  
  
getParameterMap方法  
------------------  
参数名:username  
aaa,bbb,  
参数名:password  
123,  
  
  
getParameterNames()方法  
----------------------  
参数名:username  
参数值:aaa  
参数名:password  
参数值:123

下面我们再来看一下request的乱码问题：

我们还是直接使用demo.jsp中的方式传递参数，当我们在页面文本框中输入"中国"

在控制台中打印获取到的username的值，显示的是？？，这个是因为Request域中的采用的是ISO8859-1码表的，而我们的demo.jsp使用的是utf-8编码的，所以当我们点击提交的时候，浏览器会将"中国"使用utf-8码表编码，然后web容器创建一对request/response容器对象，数据传入到request容器中，因为request容器使用的是iso8859-1编码的，所以当我们在Servlet中从request容器中读取数据，使用的是iso8859-1进行解码的，所以会出现乱码了，所以我们只需要将request的容器码表设置成和我们页面显示的码表一样就可以了。这样我们在getParameter的时候得到正确的解码(utf-8)数据

****[java]**** [view plain](http://blog.csdn.net/jiangwei0910410003/article/details/22925915" \o "view plain) [copy](http://blog.csdn.net/jiangwei0910410003/article/details/22925915" \o "copy)

1. request.setCharacterEncoding("utf-8");
2. String name = request.getParameter("username");
3. System.out.println("username:"+name);

request.setCharacterEncoding("utf-8");

String name = request.getParameter("username");

System.out.println("username:"+name);

这时候就可以了，就能够正常显示了。

但是问题还没有结束，以上说到的乱码问题是在使用post方式传递的数据，下面我们在来看一下使用get方式传递数据的乱码问题，

昨天突然发现request.setCharacterEncoding("UTF-8")这句代码失效，前后台编码统一都是UTF-8，但通过request.getParameter("name")接收到的表单数据依然乱码,后来发现原因是表单的提交方式没有设置，也就是采用了默认的GET方式提交。   
那为什么GET方式会出现问题？难道request.setCharacterEncoding("UTF-8")这句代码只对POST方式提交数据才有效？   
做了一些测试之后总结出了一点规律：   
1、web浏览器对页面上通过GET方式提交的数据会进行URL编码，采用的编码方式通常由html页面上   
<meta http-equiv="content-type" content="text/html; charset=UTF-8">   
这个标签内的charset所指定的编码方式决定(前提是：没有自定义浏览器发送数据的编码设置)。比如现在charset="UTF-8",那么当采用get方式提交表单的时候，web浏览器默认会采用UTF-8的编码方式对数据进行编码。  
所以，当请求的URL如下时：   
http://localhost:8080/test/TestServlet?name=中国   
其实真实内容是这样：   
http://localhost:8080/test/TestServlet?name=%E4%B8%AD%E5%9B%BD   
其中%E4%B8%AD%E5%9B%BD是对‘中国’二字采用了UTF-8的URL编码之后产生的字符串。   
2、所以，当数据被发送到Web服务器上的时候（测试使用tomcat6），服务器要做的一件事就是解码%E4%B8%AD%E5%9B%BD这个字符串。那么如何解码这个字符串？   
 3、在JDK的java.net包下面有一个类叫做URLDecoder，该类即可对URL编码之后的字符串进行解码，如：

****[java]**** [view plain](http://blog.csdn.net/jiangwei0910410003/article/details/22925915" \o "view plain) [copy](http://blog.csdn.net/jiangwei0910410003/article/details/22925915" \o "copy)

1. URLDecoder.decode("%E4%B8%AD%E5%9B%BD","UTF-8");

URLDecoder.decode("%E4%B8%AD%E5%9B%BD","UTF-8");

返回解码之后的字符串，第二个参数是指采用何种字符集解码"%E4%B8%AD%E5%9B%BD"这个字符串。打印以上代码成功显示“中国”二字，说明解码正确！   
4、前面已经说过了：web服务器会自行解码%E4%B8%AD%E5%9B%BD这个字符串，但是我们通过request.getParameter("name")得到的却是乱码，所以问题一定出在web服务器在解码E4%B8%AD%E5%9B%BD这个字符串的时候采用的字符集不对。   
5、经过测试发现，web服务器（只测试了tomcat6）对GET方式的数据提交采用的解码字符集是"ISO-8859-1",所以web服务器其实是这样解码的：   
URLDecoder.decode("%E4%B8%AD%E5%9B%BD","ISO-8859-1");   
因此：明显服务器的解码方式是不对的，因为编码采用的是UTF-8，而解码却用的ISO-8859-1。   
6、所以，request.getParameter("name")返回的是用ISO-8859-1解码的字符串，那么必然是乱码了！   
那么如何获得正确编码的字符串？可以采用以下的方式：

****[java]**** [view plain](http://blog.csdn.net/jiangwei0910410003/article/details/22925915" \o "view plain) [copy](http://blog.csdn.net/jiangwei0910410003/article/details/22925915" \o "copy)

1. String a = **new** String(request.getParameter("name").getBytes("ISO-8859-1"),"UTF-8");

String a = new String(request.getParameter("name").getBytes("ISO-8859-1"),"UTF-8");

总结：因为GET方式提交数据会被浏览器进行URL编码，而tomcat服务器会采用了错误的解码方式进行解码，所以得的是乱码。而POST方式提交的数据不会被浏览器进行URL编码，所以服务器直接按照request.setCharacterEncoding("UTF-8")   
所指定的编码方式解析字符串，因此在POST方式下request.setCharacterEncoding("UTF-8")是好用的！

这里的解决方式有两种，

一种：是通过String类的getBytes方法进行编码转换，具体java代码是：

****[java]**** [view plain](http://blog.csdn.net/jiangwei0910410003/article/details/22925915" \o "view plain) [copy](http://blog.csdn.net/jiangwei0910410003/article/details/22925915" \o "copy)

1. **new** String(request.getParameter(“name”).getBytes(“iso-8859-1”),“客户端编码方式”)

new String(request.getParameter(“name”).getBytes(“iso-8859-1”),“客户端编码方式”)

这里的客户端编码方式就是页面的编码，比如demo.jsp中使用的是utf-8

第二种：在服务器server.xml代码中改配置信息：

****[html]**** [view plain](http://blog.csdn.net/jiangwei0910410003/article/details/22925915" \o "view plain) [copy](http://blog.csdn.net/jiangwei0910410003/article/details/22925915" \o "copy)

1. **<Connector** port="8080"protocol="HTTP/1.1"  maxThreads="150" connectionTimeout="20000"
2. **<span** style="white-space: pre;"**>**    **</span>**   redirectPort="8443"URIEncoding="客户端编码"**/>**

<Connector port="8080"protocol="HTTP/1.1" maxThreads="150" connectionTimeout="20000"

redirectPort="8443"URIEncoding="客户端编码"/>

这样我们就修改了Tomcat中的编码和解码字符集了。当tomcat获取客户机使用get方式带来的数据使用URL解码的字符集

当然我们之后修改server.xml之后需要重启服务器的，所以第二种方式是不赞同使用的。

上面我们就讲述了如何解决request的乱码问题

下面我们再来看一下请求转发的问题：

对于重定向和转发我们这里就不做太多的介绍了，之前不知道说了好多遍了，我们之前说过ServeltContext也是可以得到一个转发对象RequestDispatch的，其实Request也是可以得到一个RequestDispatch对象的，我们还是来做个例子，通过一个servlet转发到另一个servlet：

Servlet1代码：

****[java]**** [view plain](http://blog.csdn.net/jiangwei0910410003/article/details/22925915" \o "view plain) [copy](http://blog.csdn.net/jiangwei0910410003/article/details/22925915" \o "copy)

1. **public** **void** doGet(HttpServletRequest request, HttpServletResponse response)**throws** ServletException, IOException {
2. //在request域中存入一个属性值，然后转发到Servlet2中进行读取
3. request.setAttribute("data", "Hello Servlet2");
4. request.getRequestDispatcher("/Servlet2").forward(request, response);
5. }

public void doGet(HttpServletRequest request, HttpServletResponse response)throws ServletException, IOException {

*//在request域中存入一个属性值，然后转发到Servlet2中进行读取*

request.setAttribute("data", "Hello Servlet2");

request.getRequestDispatcher("/Servlet2").forward(request, response);

}

我们在request域中存入data属性，然后在Servlet2中读取出来进行显示：

Servlet2代码：

****[java]**** [view plain](http://blog.csdn.net/jiangwei0910410003/article/details/22925915" \o "view plain) [copy](http://blog.csdn.net/jiangwei0910410003/article/details/22925915" \o "copy)

1. **public** **void** doGet(HttpServletRequest request, HttpServletResponse response)**throws** ServletException, IOException {
2. //获取属性值
3. String data = (String) request.getAttribute("data");
4. //打印
5. response.getWriter().write(data);
6. }

public void doGet(HttpServletRequest request, HttpServletResponse response)throws ServletException, IOException {

*//获取属性值*

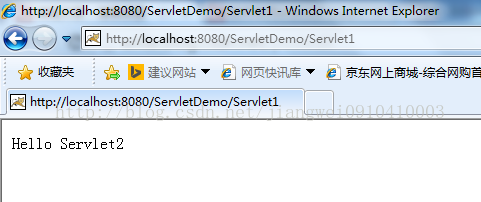
String data = (String) request.getAttribute("data");

*//打印*

response.getWriter().write(data);

}

运行结果：



我们取出来了属性data的值，显示成功了，但是我们记得在之前说的ServletContext域中存入了data属性值，那个也是可以读取的，但是那时候我们说过那种方式是不可靠的，因为ServletContext是全局的，整个web应用都是有效的，所以可能发生数据错乱的情况，比如现在一个人去请求servlet1,就在要进行转发到servlet2的时候，这时候又来一个人去请求servlet1,这时候他也去设置data属性值，因为ServletContext是全局的，所以第一个人在servlet2中读取的可能是第一个人在servlet1中存入的属性值，那样数据就乱了，所以我们只能使用request域了，而且转发都是在一个request域中的，当多个用户来访问servlet1的时候，是有多个request域的，所以互相是不会干扰的，这样数据也是不会乱的，所以说在使用转发技术的时候使用request域存数据是可靠的，而不是用ServletContext域。

下面在来看一下使用转发技术的时候我们需要注意的问题：

当我们在使用转发的时候不能将response流关闭了，不然会报错的，即在request.getRequestDispatcher("index.jsp").forward(request,response);这行代码前不能将response.getOutputStream流关闭。

****[java]**** [view plain](http://blog.csdn.net/jiangwei0910410003/article/details/22925915" \o "view plain) [copy](http://blog.csdn.net/jiangwei0910410003/article/details/22925915" \o "copy)

1. //关闭流
2. response.getOutputStream().close();
3. //进行转发
4. RequestDispatcher rd = request.getRequestDispatcher("/demo.jsp");//不是使用ServletContext的
5. rd.forward(request, response);

*//关闭流*

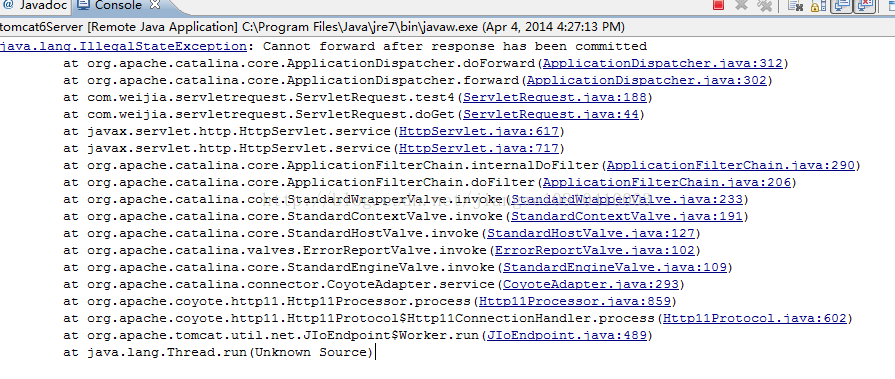
response.getOutputStream().close();

*//进行转发*

RequestDispatcher rd = request.getRequestDispatcher("/demo.jsp");*//不是使用ServletContext的*

rd.forward(request, response);

异常：



这个关闭流操作是很明显的，下面我们在看一下一个不明显的，也是最容易犯错的：

****[java]**** [view plain](http://blog.csdn.net/jiangwei0910410003/article/details/22925915" \o "view plain) [copy](http://blog.csdn.net/jiangwei0910410003/article/details/22925915" \o "copy)

1. request.getRequestDispatcher("/demo.jsp").forward(request, response);
2. //转发完之后再去转发
3. request.getRequestDispatcher("/index.jsp").forward(request, response);

request.getRequestDispatcher("/demo.jsp").forward(request, response);

*//转发完之后再去转发*

request.getRequestDispatcher("/index.jsp").forward(request, response);

从代码中我们可以看到是在一次转发之后，在通过一次转发，这个报错的原因和上面的是一样的，因为当一个转发之后，response就是想浏览器输出数据了，当输出完数据之后，response就会自动的关闭流，所以会报和上面的错误，这种错误是经常犯的，因为我们有时候会在不同的页面或者servlet中进行转发，这样就可能造成这种错了，所以我们解决这种问题就是：

****在每次调用转发代码之后一定要记得添加一句代码：return;****

****这样后续的代码就不会执行了，所以就不会有这种错误了，同样的前面说到的重定向也是这样的问题，为了我们在重定向之后，后续的代码不在执行，所以必须添加一句代码：return;****

关于转发还有一个问题就是，在使用转发技术的时候，会冲掉response中已写入的数据：实例代码如下：

****[java]**** [view plain](http://blog.csdn.net/jiangwei0910410003/article/details/22925915" \o "view plain) [copy](http://blog.csdn.net/jiangwei0910410003/article/details/22925915" \o "copy)

1. //在调用转发之前向response中写入的数据，会被冲掉
2. String data = "aaaaa";
3. response.getWriter().write(data);
4. request.getRequestDispatcher("/demo.jsp").forward(request, response);

*//在调用转发之前向response中写入的数据，会被冲掉*

String data = "aaaaaa";

response.getWriter().write(data);

request.getRequestDispatcher("/demo.jsp").forward(request, response);

这样我们只能看到demo.jsp页面，而看不到"aaaaaa"数据了，因为他被冲掉了。

最后我们在看一下在web应用中怎么书写各种路径：

规则：

写任何地址都是使用斜杠开头：如果是写给服务器用的这个"/"就代表当前web应用，如果是写给浏览器用的:这个"/"是指当前网站  
浏览器用的：客户机需要这个地址去请求服务器  
服务器用的：服务器本身用的

实例：

****[java]**** [view plain](http://blog.csdn.net/jiangwei0910410003/article/details/22925915" \o "view plain) [copy](http://blog.csdn.net/jiangwei0910410003/article/details/22925915" \o "copy)

1. //1:写给服务器用的
2. request.getRequestDispatcher("/form1.html").forward(request, response);
4. //2:写给浏览器的
5. response.sendRedirect("/ServletDemo/form1.html");
7. //3:写给服务器用的
8. **this**.getServletContext().getRealPath("/WEB-INF/form1.html");
10. //4:写给服务器用的
11. **this**.getServletContext().getResourceAsStream("/form1.html");
13. //5:写给浏览器用的
14. /\*\*
15. \* <a href="/ServletDemo/form1.html">点点</a>
16. \*/
18. //6:写给浏览器用的
19. /\*\*
20. \* <form action="/ServletDemo/form1.html">
21. \* </form>
22. \*/

*//1:写给服务器用的*

request.getRequestDispatcher("/form1.html").forward(request, response);

*//2:写给浏览器的*

response.sendRedirect("/ServletDemo/form1.html");

*//3:写给服务器用的*

this.getServletContext().getRealPath("/WEB-INF/form1.html");

*//4:写给服务器用的*

this.getServletContext().getResourceAsStream("/form1.html");

*//5:写给浏览器用的*

*/\*\**

*\* <a href="/ServletDemo/form1.html">点点</a>*

*\*/*

*//6:写给浏览器用的*

*/\*\**

*\* <form action="/ServletDemo/form1.html">*

*\* </form>*

*\*/*