**请求对象和响应对象**

request and response

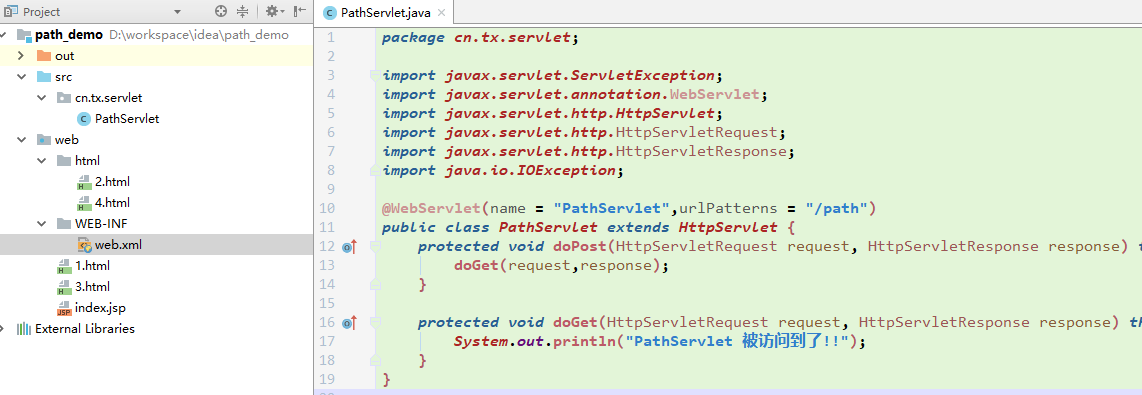
拓薪教育

刘洋

1. 相对路径和绝对路径(熟练)

Java中使用的路径，分为两种：绝对路径和相对路径。归根结底，Java本质上只能使用绝对路径来寻找资源。所有的相对路径寻找资源的方法，都不过是一些便利方法。不过是API在底层帮助我们构建了绝对路径，从而找到资源的！

我们以下面的项目路径来进行举例:



* 1. 相对路径

相对路径，根据当前资源路径 与 目标资源路径，寻找相对位置关系，通过 . (当前目录) 和 .. (上一级目录) 访问目标资源;

综上所述,我们发现相对路径总需要分析当前路径与目标路径对应关系，编写规则会根据当前路径不同而不同

* 1. 绝对路径

绝对路径是指目录下的绝对位置，直接到达目标位置

带有协议完整路径 （跨网站） http://localhost/day5/hello

以/ 开始路径 （同一个站点内） : /day5/hello

服务器端和客户端对于/ 的区别

客户端访问路径：/day5/hello

服务器内部路径：/hello, 多用于服务器跳转, 页面包含(jsp)

* 1. 测试结果

测试结果请参照: 相对路径绝对路径演示记录.xlsx

1. Tomcat请求响应

Web服务器收到客户端的http请求，会针对每一次请求，分别创建一个用于代表请求的request对象、和代表响应的response对象request和response对象即然代表请求和响应，那我们要获取客户机提交过来的数据，只需要找request对象就行了。要向客户机输出数据，只需要找response对象就行了

1. HttpServletRequest基本功能(了解)
   1. HttpServletRequest的功能

HttpServletRequest在JavaWeb中非常重要的一个类。它是Servlet的service()方法的参数之一！所以你必须必须要掌握它！

request的功能可以分为以下几种：

* 封装了请求头数据；
* 封装了请求正文数据，如果是GET请求，那么就没有正文；
* request是一个域对象，可以把它当成Map来添加获取数据；
* 做请求的转发
  1. request获取请求头数据

String value = request.getHeader(“请求头名称”);

request对象可以用来获取请求头数据，当然，这些请求数据都是Tomcat封装到request中去的。我们在service()方法中可以直接来获取！

request与请求头相关的方法有：

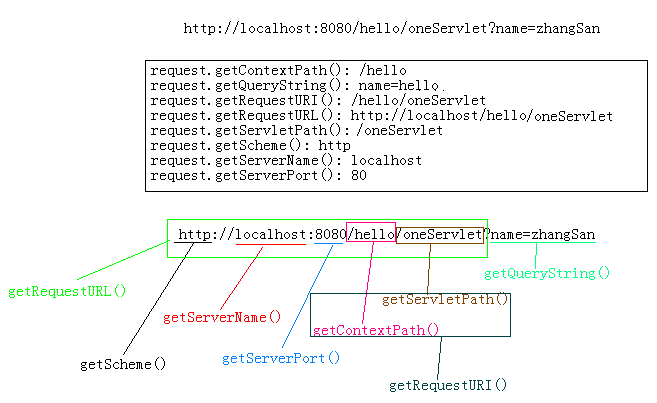
* String getHeader(String name)：获取指定名称的请求头；
* Enumeration getHeaderNames()：获取所有请求头名称；

|  |
| --- |
| response**.setContentType("text/html;charset=utf-8");** Enumeration ***names* =** request**.getHeaderNames(); *while*(*names*.hasMoreElements()) {  *String name* = (*String*)*names*.nextElement();  *String value* =** request**.getHeader(*name*);  *System*.*out*.println(*name* + ": " + *value*);** response**.getWriter().println(*name* + ": " + *value* + "<br/>"); }** |

* 1. request获取请求相关的其它方法

request中还提供了与请求相关的其他方法，有些方法是为了我们更加便捷的方法请求头数据而设计，有些是与请求URL相关的方法。

* int getContentLength()：获取请求正文的字节数，GET请求没有正文，没有正文返回-1；
* **String getContentType()：**获取请求类型，如果请求是GET，那么这个方法返回null；如果是POST请求，那么默认为application/x-www-form-urlencoded（理解为字符串类型），其它类型以后再学；
* String getMethod()：返回请求方法，例如：GET
* Locale getLocale()：返回当前客户端浏览器支持的Locale。java.util.Locale表示国家和言语，这个东西在国际化中很有用；
* **String getCharacterEncoding()：**获取请求编码，如果没有setCharacterEncoding()，那么返回null。表示使用ISO-8859-1编码；
* void setCharacterEncoding(String code)：设置请求编码，只对正文有效！注意，对于GET而言，没有正文！！！所以此方法只能对POST请求中的参数有效！
* http://localhost:8080/hello/oneServlet?name=zhangSan
* **String getContextPath()：**返回上下文路径，例如：/项目名称
* String getQueryString()：返回请求URL中的参数，例如：name=zhangSan
* String getRequestURI()：返回请求URI路径，例如：/hello/oneServlet
* StringBuffer getRequestURL()：返回请求URL路径，例如：http://localhost/hello/oneServlet，即返回除了参数以外的路径信息；
* String getServletPath()：返回Servlet路径，例如：/oneServlet
* String getRemoteAddr()：返回当前客户端的IP地址；
* String getRemoteHost()：返回当前客户端的主机名，但这个方法的实现还是获取IP地址；
* int getRemotePort()：返回客户端的端口号，每次请求都会变；
* String getSchema()：返回请求协议，例如：http；
* String getServerName()：返回主机名，例如：localhost
* int getServerPort()：返回服务器端口号，例如：80



|  |
| --- |
| ***System*.*out*.println("request.getContentLength(): " +** request**.getContentLength()); *System*.*out*.println("request.getContentType(): " +** request**.getContentType()); *System*.*out*.println("request.getContextPath(): " +** request**.getContextPath()); *System*.*out*.println("request.getMethod(): " +** request**.getMethod()); *System*.*out*.println("request.getLocale(): " +** request**.getLocale()); *System*.*out*.println("request.getQueryString(): " +** request**.getQueryString()); *System*.*out*.println("request.getRequestURI(): " +** request**.getRequestURI()); *System*.*out*.println("request.getRequestURL(): " +** request**.getRequestURL()); *System*.*out*.println("request.getServletPath(): " +** request**.getServletPath()); *System*.*out*.println("request.getRemoteAddr(): " +** request**.getRemoteAddr()); *System*.*out*.println("request.getRemoteHost(): " +** request**.getRemoteHost()); *System*.*out*.println("request.getRemotePort(): " +** request**.getRemotePort()); *System*.*out*.println("request.getScheme(): " +** request**.getScheme()); *System*.*out*.println("request.getServerName(): " +** request**.getServerName()); *System*.*out*.println("request.getServerPort(): " +** request**.getServerPort());** |

1. HttpServletRequest获取参数(精通)
   1. 传递参数的方式
      1. 传递参数的方式：GET和POST。

GET：

* 地址栏中直接给出参数：http://localhost/param/ParamServlet?p1=v1&p2=v2；
* 超链接中给出参数：

<a href=” <http://localhost/param/ParamServlet?p1=v1&p2=v2>”>点一下</a>

* 表单中给出参数：<form method=”GET” action=”ParamServlet”>…</form>
* Ajax暂不介绍

POST：

* 表单中给出参数：<form method=”POST” action=”ParamServlet”>…</form>
* Ajax暂不介绍
  1. HttpServletRequest获取参数方法

可以使用HttpServletRequest获取客户端的请求参数，相关方法如下：

* **String getParameter(String name)**：通过指定名称获取参数值；
* **String[] getParameterValues(String name)**：通过指定名称获取参数值数组，有可能一个名字对应多个值，例如表单中的多个复选框使用相同的name时；
* Enumeration getParameterNames()：获取所有参数的名字；
* Map getParameterMap():获取所有参数对应的Map，其中key为参数名，value为参数值。
  1. 单值参数的接收

单值参数包括单选，单值下拉框，文本，隐藏域

无论是GET还是POST，获取参数的方法是相同的。

String s1 = request.getParameter(“p1”);//返回v1

String s2 = request.getParameter(“p2”);//返回v2

前台页面:

|  |
| --- |
| **<*form* action="ParamServlet" method="post">  <*input* type="text" name="p1"/><*br*/>  <*input* type="text" name="p2"/><*br*/>  <*input* type="submit" value="提交"/><*br*/> </*form*>** |

|  |
| --- |
| **<*a* href="ParamServlet?p1=v1&p2=v2">**Param**</*a*>** |

后台接收:

|  |
| --- |
| ***String s1* =** request**.getParameter("p1"); *String s2* =** request**.getParameter("p2");** response**.getWriter().print("p1 = " + *s1* + "<br/>");** response**.getWriter().print("p2 = " + *s2* + "<br/>");** Enumeration ***names* =** request**.getParameterNames(); *while*(*names*.hasMoreElements()) {  *String name* = (*String*)*names*.nextElement();  *String value* =** request**.getParameter(*name*);  *System*.*out*.println(*name* + " = " + *value*); }**  Map**<*String*, *String*[]> *map* =** request**.getParameterMap();** Set**<*String*> *keySet* = *map*.keySet(); *for*(*String key* : *keySet*){  *String*[] *vals* = *map*.get(*key*);  *System*.*out*.print(*key*+" :");  *System*.*out*.println(*key*+" :" + *Arrays*.*toString*(*vals*));  *System*.*out*.println(); }** |

* 1. 多值参数接收

多值参数主要就是多选checkbox

例如在注册表单中，如果让用户填写爱好，那么爱好可能就是多个。那么hobby参数就会对应多个值：

|  |
| --- |
| **<*form* action="ParamServlet" method="post">** 上网：**<*input* type="checkbox" name="hobby" value="netplay" /><*br*/>** 踢球：**<*input* type="checkbox" name="hobby" value="football" /><*br*/>** 看书：**<*input* type="checkbox" name="hobby" value="read" /><*br*/>** 编程：**<*input* type="checkbox" name="hobby" value="programme" /><*br*/>  <*input* type="submit" value="提交"/><*br*/> </*form*>** |

|  |
| --- |
| ***String*[] *hobbies* =** request**.getParameterValues("hobby"); *System*.*out*.println(*Arrays*.*toString*(*hobbies*));** |

|  |
| --- |
| Map**<*String*, *String*[]> *map* =** request**.getParameterMap();** Set**<*String*> *keySet* = *map*.keySet(); *for*(*String key* : *keySet*){  *String*[] *vals* = *map*.get(*key*);  *System*.*out*.println(*key*+" :" + *Arrays*.*toString*(*vals*)); }** |

* 1. HttpServletRequest的获得中文乱码处理(熟练)

Request接收参数时有get和post两种请求方式，但是处理中文的编码却不一样，我们在做项目时会全站都采用统一的编码，最常用的就是UTF-8，在UTF-8编码的项目中的乱码处理方法如下:

* + 1. Post请求中文乱码

我们知道，请求信息中，只有POST存在正文，所谓POST参数编码就是就是请求正文的编码。默认情况下，使用getParameter()获取POST请求参数时，使用的是ISO-8859-1编码。

第一种方法,针对字符串本身进行手动转码:

|  |
| --- |
| ***String name* =** request**.getParameter("name"); *name* = *new* String(*name*.getBytes("ISO-8859-1"),"UTF-8"); *System*.*out*.println("name:"+*name*);** |
|  |

因为在获取参数时已经被错误的编码了，但因为我们知道，乱码的两个原因：本来是使用UTF-8编码的，还错误的使用了ISO-8859-1编码。所以我们可以先使用ISO-8859-1获取字节数组，然后再使用正确的UTF-8编码得到字符串，这样就没问题了。

request的setCharacterEncodng()可以设置编码，当然这必须在调用所有的getParameter()方法之前调用request的setCharacterEncodng()方法来设置编码，这样，就不会使用ISO解读字节串了，而是使用你给定的编码来解读。

第二种方法,直接设置request的编码格式:

|  |
| --- |
| request**.setCharacterEncoding("UTF-8"); *String name* =** request**.getParameter("name"); *System*.*out*.println("name:"+*name*);** |

对于每个请求，只需要调用request的setCharacterEncodng()一次，然后所有getParameter()都会使用这个编码来解读参数。但要注意，只对请求正文有效，即POST参数。

该方法 必须放在获取参数值之前

* + 1. Get请求中文乱码

Get请求是讲参数放在URL的queryString中进行提交,故此不存在请求体,所以上面处理post请求乱码的方法不再生效.

第一种方式,同样针对字符串进行单独解码:

|  |
| --- |
| // 第二种方法,直接设置request的编码格式首选 （一定要放第一行） request.setCharacterEncoding("utf-8"); String characterEncoding = request.getCharacterEncoding(); System.*out*.println("获取编码"+characterEncoding); //通过request获取到参数的键值对map集合 Map<String, String[]> parameterMap1 = request.getParameterMap(); //遍历map集合 获取map的全部key Set<String> keys1 = parameterMap1.keySet(); //遍历全部key for (String key : keys1) {  //通过key获取到全部valves  String[] Values = request.getParameterValues(key);  System.*out*.println(key +":"+ Arrays.*toString*(Values)); |

第二种方式,通过设置服务器配置来让get请求支持中文,例如我们使用tomcat服务器,则设置/conf/server.xml:

|  |
| --- |
| <Connector port="8080" protocol="HTTP/1.1"  connectionTimeout="20000"  redirectPort="8443" URIEncoding="UTF-8"/> |

一旦设置了这个属性，那么对于GET参数就直接是UTF-8编码的了。但是，<Connector>元素来说，对整个Tomcat都是有效的！

不建议使用get请求传递中文参数

说明:

**Tomcat8.5及以上版本,无需进行设置,即可直接支持URL传递中文;**

* 1. HttpServletRequest请求转发

在Servlet中请求转发是大量要使用的，因为当我们访问一个Servlet的时候通常会执行一些后台的业务逻辑，然后跳转到一个结果页面，那么跳转到结果页面的这个过程就是请求转发，举个例子我们做登录的功能，我们填写用户名密码然后提交到一个负责登录的Servlet，Servlet为我们做用户名和密码的校验，如果我们都正确的话，我们就要跳转到登录的提示页面，如果错误就要跳转到登录失败的页面。

Request的请求转发也可以叫做服务器端的跳转，虽然有页面的跳转但是我们会发现地址栏是不会有变化的

|  |
| --- |
| request**.getRequestDispatcher("/success.html").forward(**request**,** response**);** |

我们不但可以跳转到静态页面（后续主要讲解是动态页面我们通常会跳转到一个jsp（jsp在Servlet之后产生）的提示页面，因为我们要返回的是动态页面，所有html是不适合（后续讲解））。还可以跳转到Servlet，此时我们可以给request来设置当前域中的属性值，在该域之内（当前请求完成之前）都能获得到该属性值。

|  |
| --- |
| request**.setAttribute("name", "txjava");** request**.getAttribute("name");** |

* 1. request作用域

在上一讲我们提及过ServletContext的概念，它也是一个域的对象，它的范围非常大，是指定项目所有Servlet的公共的对象，随着服务器的启动而产生，服务器的停止而销毁，那么request的也是域对象，它的作用范围小的多，它的范围只在一次请求响应范围之内，每一个线程的请求都会新产生一个HttpServletRequest和HttpServletResponse的对象

1. HttpServletResponse(熟练)

request是请求对象，而response是响应对象。

response对象用于响应client请求，向客户输出信息。

他封装了JSP产生的响应，并发送到client以响应client请求。

* 1. HttpServletResponse功能介绍

response对象的功能分为以下四种：

* 设置响应头信息；addHeader(“reFresh”, “5;URL=xxxx”);
* 发送状态码；sendError(404);
* 设置响应正文；**getWriter().print(“fdsfdsa”)**;
* 重定向：**sendRedirect(“path”);**
  1. 设置状态码和其他方法
* **response.setContentType("text/html;charset=utf-8")**：设置响应类型为html，编码为utf-8，处理相应页面文本显示的乱码；
* **response.setCharacterEncoding(“utf-8”)**：如果响应类型为文本，那么就需要设置文本的编码类型，然后浏览器使用这个编码来解读文本。注意，如果没有设置contentType，那么浏览器会认为contentType为text/html，如果没设置编码，那么默认为ISO-8859-1编码。所以以上两点在使用response返回结果之前必须设置。
* response.setStatus(200)：设置状态码；
* response.sendError(404, “您要查找的资源不存在”)：当发送错误状态码时，Tomcat会跳转到固定的错误页面去，但可以显示错误信息。
  1. 设置响应头信息
* response.setHeader(“contentType”, “text/html;charset=utf-8”)：与setContentType()方法的功能相同。setContentType()方法属于便捷方法；
* 刷新（定时重定向）：
* response.setHeader("Refresh","5; URL=http://www.baidu.com")：5秒后自动跳转到百度主页。
  1. Response中文 乱码问题

当我们使用response向页面返回带有中文的值时,可能会出现中文乱码,需要进行如下设置:

|  |
| --- |
| response**.setContentType("text/html;charset=utf-8");** response**.setCharacterEncoding("UTF-8");** response**.getWriter().write("张三");** |

* 1. Response重定向

使用response对象的sendRedirect()方法能够将网页重定向到还有一个页面。重定向支持将地址重定向到不同的主机上，这一点与转发不同。在client浏览器上将会得到跳转后的地址，并又一次发送请求链接；用户能够从浏览器的地址栏中看到跳转后的地址；重定向操作后，request中的属性将会所有失效，并開始一个新的request对象

|  |
| --- |
| response**.sendRedirect("/param\_demo/success.html");** |

1. 附加信息
   1. 面试题:转发和重定向的区别
      1. 处理流程:

* 请求转发:
* 客户端发送请求，Servlet做出业务逻辑处理。
* Servlet调用forword()方法，服务器Servlet把目标资源返回给客户端浏览器。
* 重定向:
* 客户端发送请求，Servlet做出业务逻辑处理。
* Servlet调用response.sendReadirect()方法，把要访问的目标资源作为response响应头信息发给客户端浏览器。
* 客户端浏览器重新访问服务器资源xxx.jsp，服务器再次对客户端浏览器做出响应。
  + 1. 路径
* 使用相对路径在重定向和转发中没有区别
* 重定向和请求转发使用绝对路径时，根/路径代表了不同含义
  + 重定向response.sendRedirect("xxx")是服务器向客户端发送一个请求头信息，由客户端再请求一次服务器。/指的Tomcat的根目录,写绝对路径应该写成"/当前Web程序根名称/资源名" 。如"/WebModule/login.jsp","/bbs/servlet/LoginServlet"
  + 转发是在服务器内部进行的，写绝对路径/开头指的是当前的Web应用程序。绝对路径写法就是"/login.jsp"或"/servlet/LoginServlet"。
* 总结：以上要注意是区分是从服务器外的请求，还在是内部转发，从服务器外的请求，从Tomcat根写起(就是要包括当前Web的根)；是服务器内部的转发，很简单了，因为在当前服务器内，/写起指的就是当前Web的根目录。
  + 1. 其他区别
* request.getRequestDispatcher()是容器中控制权的转向，在客户端浏览器地址栏中不会显示出转向后的地址；服务器内部转发，整个过程处于同一个请求当中。
* response.sendRedirect()则是完全的跳转，浏览器将会得到跳转的地址，并重新发送请求链接。这样，从浏览器的地址栏中可以看到跳转后的链接地址。不在同一个请求。重定向，实际上客户端会向服务器端**发送两个请求**。
* 所以转发中数据的存取可以用request作用域：request.setAttribute(), request.getAttribute()，重定向是取不到request中的数据的。只能用session。
* forward()更加高效，在可以满足需要时，尽量使用RequestDispatcher.forward()方法。
* RequestDispatcher是通过调用HttpServletRequest对象的getRequestDispatcher()方法得到的，是属于请求对象的方法。
* sendRedirect()是HttpServletResponse对象的方法，即响应对象的方法，既然调用了响应对象的方法，那就表明整个请求过程已经结束了，服务器开始向客户端返回执行的结果。
* 重定向可以跨域访问，而转发是在web服务器内部进行的，不能跨域访问。

1. 学习目标

* 熟练掌握相对路径和绝对路径的使用,能够解决项目中的404问题;
* 能够使用request的API接收所有的前台传过来的参数(单值多值);
* 能够熟练处理页面各种中文乱码问题;
* 能够熟练的使用request完成请求转发,使用response完成请求重定向;
* 掌握请求转发和重定向的区别(面试题);