request概述与功能一---API操作

2019年5月17日 星期五 9:05

1. 回顾：HTTP协议
   a. HTTP数据交互模型：
      请求响应模型。
      一次请求对应一次响应。
      请求只能由浏览器发出，服务器可以根据请求做出响应。
2. HTTP请求
   a. 继承结构
      ServletRequest 提供servletrequest对象必要的方法。
      |
      |---HttpServletRequest 包含和HTTP协议操作的一些API，可以通过这些API来开发浏览器和服务器之间的功能。
   b. HTTP请求的组成
      一个请求行 请求方式 请求资源的路径 HTTP协议版本
      多个请求头 Host：虚拟主机名称
      一个空行
      请求实体内容 用户发送请求时携带的请求参数
3. Request当中的API操作：
   - 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

     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 -- 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>

     getHeaderNames方法 --- Enumeration<String>

     //快捷键获取返回值：Alt+Shift+L; 注意：光标放在最后。

     getIntHeader(name)方法 --- int

     getDateHeader(name)方法 --- long(日期对应毫秒)
   - 获取请求参数

     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 将查询的参数保存在一个Map中

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

i. 获取请求参数乱码问题：

乱码出现的原因：编解码不一致。

ii. 浏览器端发送的数据字符集使用的UTF-8，原因是在浏览器打开的时候使用的是什么字符集发送数据的时候就会使用什么字符集进行编码操作。

iii. tomcat服务器接收数据字符集ISO8859-1，因为服务器的默认字符集是ISO8859-1，所以在收到浏览器发送的以UTF-8编码的数据时解码过程会出现乱码。

iv. 统一浏览器端和服务器的字符集为UTF-8.

1） 修改服务器接收数据时使用的字符集。

a） request.setCharacterEncoding("utf-8")；
当前语句仅对post请求生效，因为用户使用post提交时，请求参数会存储在请求实体内容中。语句可以将请求实体内容中的中文参数乱码做出处理。

b） 由于get提交参数在地址栏中拼接传输，没有进入请求实体内容，所以此处无法处理get提交的乱码。

c） 注意：post乱码处理一定要放在获取参数的代码之前。

2） get提交参数乱码处理

a） 先对获取到的中文进行编码操作，使用字符集iso8859-1，获取到真正的二进制数据，将二进制数据使用正确是utf-8字符集重新解码。

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

b） 使用utf-8字符集解码二进制数据。

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

c) 这种处理方式其实就是讲获取到的参数重新编码、解码各一次，所以不管get提交还是post提交参数都会重新编解码，所以它既能解决get提交乱码也可以解决post提交乱码。

```
username = new String(????.getBytes("iso8859-1"),"utf-8");
```

request功能二--请求转发

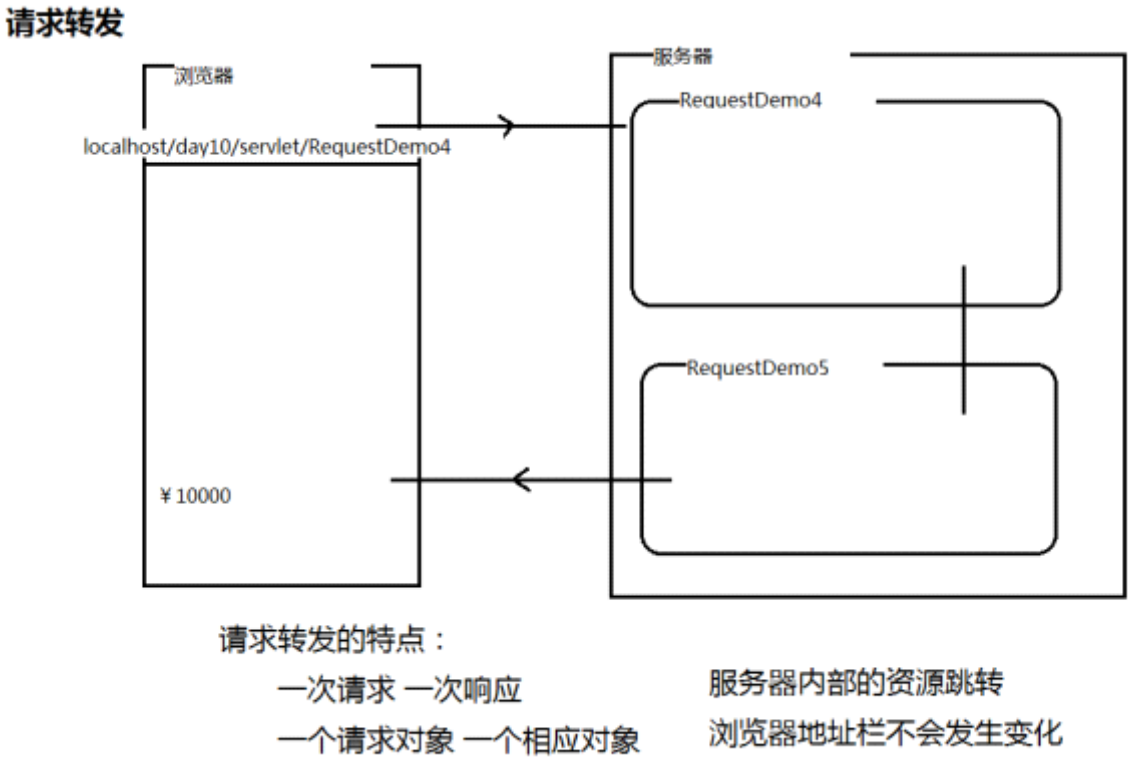
2019年5月17日 星期五 14:02

1. 什么是请求转发：

   是一种资源跳转方式。

   由一个动态资源跳转到另外一个动态资源，让第二个动态资源对浏览器做出响应的操作，这个过程就称之为请求转发。

2. 请求转发的特点：



   一次请求一次响应

   一个请求对象和一个响应对象

   服务器内部的资源跳转

   浏览器地址栏不会发生变化。

   注意：请求转发是一个服务器内部的资源跳转，浏览器不会察觉到变化，最明显的反馈就是浏览器地址栏不发生变化。

3. 请求转发实现：

   创建一个调度器，利用调度器实现请求转发，代码如下：

```
//获取一个调度器
RequestDispatcher rd = request.getRequestDispatcher("/servlet/RequestDemo5");
//利用调度器实现请求转发
rd.forward(request, response);
```

   a. 请求转发的细节：

      i. 请求转发之前如果response缓冲区中有数据,在请求转发的时候会将response缓冲区中的内容清空.

      ii. 如果冲刷response缓冲区，则会造成一次响应操作，在请求转发过程中也会发生一次响应操作，两个响应操作不满足一次请求对应一次响应的模型，所以会

发生异常。

iii. 请求转发之前和之后的语句依然会正常执行，执行的顺序就是请求转发的顺序。

iv. 请求转发可以多重转发，但是不能再同一个servlet中多次装发。如果多次转发会造成多次响应必然不符合一次请求一次响应的模型，发生异常。

4. 注意：请求转发是web应用内部的资源跳转，不可以在web应用之间跳转。

request功能三--作为域对象使用

2019年5月17日 星期五 15:12

1. 域对象概述

   域对象就是一个对象有一个可以被看见的范围，在这个范围内利用对象身上的map实现数据、资源的共享，像这样的一个对象称之为域对象。

2. 域对象的使用方式：

   a. setAttribute() 向域中添加数据

   b. getAttribute(String name) 获取指定名称的域属性

   c. removeAttribute(String name) 删除指定名称的域属性

   d. getAttributeNames() 获取全部域属性的名称

request. getParameter("name")	---- 请求参数中包含的参数
request.getAttribute("name")	----域中包含的属性

3. request域对象的特点：

   a. 生命周期：

      在请求链开始的时候request对象创建，在请求链结束的时候request对象销毁。

   b. 作用范围：

      请求链的范围即为request域对象的作用范围。（整个请求链。）

   c. 主要功能：

      在请求链中共享数据。

4. 案例：从servlet向首页输出数据

   a. 使用request域对象实现。

   b. 代码如下：

   RequestDemo11：

```
package cn.tedu.request;

import java.io.IOException;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
/**
 * 作为域对象使用：
 *              向域中添加数据
 *              请求转发给首页展示。
 * @author Administrator
 *
 */
public class RequestDemo11 extends HttpServlet {

    public void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
            String username = "lishuai";
            int age = 19;
            String addr = "bj";
            String gender = "男";
            //第一步添加数据到request域中
            request.setAttribute("username", username);
            request.setAttribute("age", age);
```

```
            request.setAttribute("addr", addr);
            request.setAttribute("gender", gender);
            //添加完数据后，请求转发到首页
            request.getRequestDispatcher("/index.jsp").forward(request, response);
        }

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

        }

}
```

index. jsp页面：

```
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
 <head>
  <title>My JSP 'index.jsp' starting page</title>

 </head>

 <body>
        <!-- 在首页中获取RequestDemo11中设置的域属性对应的值 -->
        姓名：<%=request.getAttribute("username") %>
        年龄：<%=request.getAttribute("age") %>
        性别：<%=request.getAttribute("gender") %>
        地址：<%=request.getAttribute("addr") %>
  </body>
</html>
```

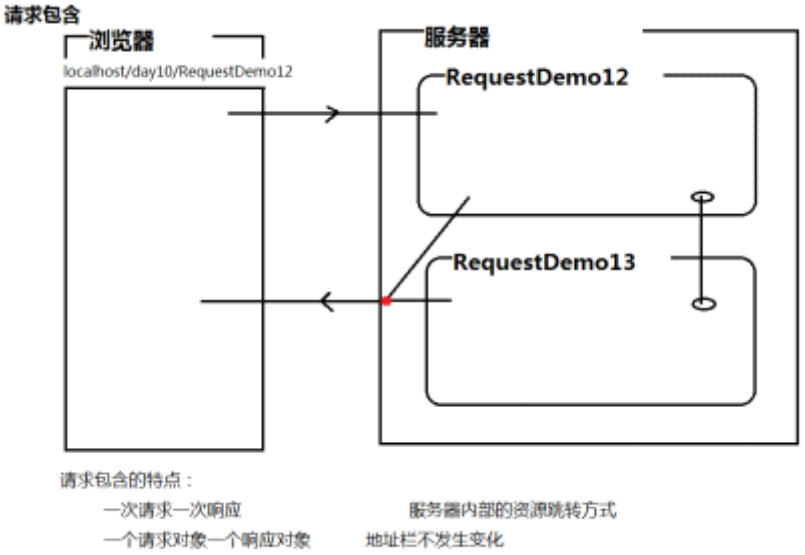
request功能四--请求包含

2019年5月17日 星期五 15:41

1. 请求包含概念

   一个动态资源和另一个动态资源一起处理一个请求操作，并将两个Servlet的结果合并输出给浏览器，像这样的过程称作请求包含。

2. 请求包含的特点：



   一次请求一次响应
   一个请求对象一个响应对象
   服务器内部的资源跳转
   地址栏不会发生变化

3. 请求包含实现：

   代码如下：

```
package cn.tedu.request;

import java.io.IOException;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
/**
 * 请求包含实现
 * 与Demo13一组
 * @author Administrator
 *
 */
public class RequestDemo12 extends HttpServlet {

	public void doGet(HttpServletRequest request, HttpServletResponse response)
			throws ServletException, IOException {
			response.getWriter().write("aaa");
			//获取一个调度器
			//利用调度器实现请求包含
			request.getRequestDispatcher("/servlet/RequestDemo13").include(request, response);
	}

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

```
            doGet(request, response);

        }

}
```

response对象

2019年5月17日 星期五 15:57

1. response对象
   a. 继承结构

      ServletResponse
      |
      |---HttpServletResponse 包含HTTP协议相关的API操作，跟适合用户HTTP协议相关的开发。

   b. HTTP响应的组成

      一个状态行 200成功 302+location请求重定向 304、307缓存 404资源找不到 500服务出错
      多个响应头 refresh 定时刷新 location地址 Expires Pragma Cache-Cotrol 控制缓存
      一个空行
      响应实体内容 response缓冲区

   c. response对象的操作：

   - *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)

   - *设置响应内容的方法*

     ServletOutputStream getOutputStream()

     PrintWriter getWriter()

response参数乱码处理

2019年5月17日 星期五 16:21

1. response参数乱码处理
   a. 字节流乱码处理

```
response.getOutputStream().write("中文".getBytes());
```

   i. 当前语句执行不发生乱码。由于浏览器接收数据采用系统默认的GBK字符集，服务器发送数据采用servlet默认的GBK字符集，编解码一致所以不会出现乱码。

```
response.getOutputStream().write("中文".getBytes("utf-8"));
```

   i. 在服务器发送数据的时候如果使用utf-8，浏览器也必须以utf-8字符集来接收数据，才能避免乱码的出现。所以应该通知浏览器使用utf-8字符集。
   ii. 通知的方式如下：

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

   此句会通知浏览器使用utf-8来接收数据。

   b. 字符流乱码处理

```
response.getWriter().write("中文");
```

   服务器以字符流发送数据本质上仍然是用字节流发送数据。在服务器发送数据的时候会采用tomcat默认的字符集进行编码iso8859-1操作，在这个字符集中没有中文，自然就会出现乱码。

   c. 必须通知服务器和浏览器两方面都是用utf-8字符集才能避免乱码。
   i. 通知服务器使用utf-8字符集。

```
response.setCharacterEncoding("utf-8");
```

   ii. 通知浏览器使用utf-8字符集。

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

   d. 在通知浏览器使用utf-8字符集是，服务器也自动采用所通知的字符集来发送数据。
   e. response.setHeader("Content-Type","text/html;charset=utf-8");可以简写为：

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

   f. 总结：
   i. 在面对字节流和字符流的时候都是用通知浏览器的方式来修改字符集。
   ii. 字节流和字符流是不可以同时使用的。会在第二个流身上发生异常。
   iii. 使用流之后不需要关闭，浏览器会自动帮助我们关闭流。千万不要手动关闭可能会发生异常。
   iv. 将通知服务器与浏览器的使用字符集的语句书写在所有代码之前。

response功能一 --- 请求重定向

2019年5月17日 星期五 16:47

1. 请求重定向

   是一种资源跳转方式，可以在不同的服务器之间跳转。

2. 请求重定向的特点：

   两次请求两次响应
   两个请求对象两个响应对象
   服务器之间的资源跳转
   浏览器地址栏会发生变化

3. 请求重定向的实现：
   a. 设置状态码302
   b. 设置响应头location

      代码如下：

```
//1.设置302状态码
//        response.setStatus(302);
        //2.设置location响应头
//        response.setHeader("location", "http://www.tmooc.cn");
//        response.setHeader("location", request.getContextPath()+"/servlet/ResponseDemo1");
        response.sendRedirect("http://www.baidu.com");
```

   c. 也可以通过

      response.sendRedirect(location);实现重定向。

response功能--定时刷新

2019年5月17日 星期五 17:03

1. 定时刷新概念

   是一种资源跳转方式，可以在不同服务器之间跳转。

2. 定时刷新的特点：

   两次请求两次响应
   两个请求对象两个响应对象
   服务器之间的资源跳转
   浏览器地址栏会发生变化

   注意：与请求重定向不同是定时刷新会等待指定时候之后才跳转。

3. 定时刷新的实现：

   只需设置一个refresh响应头即可。

   代码如下：

```
response.setContentType("text/html;charset=utf-8");
        response.getWriter().write("<h1 align='center'><font color='red'>3秒钟之后跳转！</font></h1>");
        //设置响应头
        response.setHeader("refresh","3;url=http://www.tmooc.cn");
```

response对象--控制浏览器缓存

2019年5月17日 星期五 17:11

1. 控制缓存

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!!

a. 控制浏览器不要缓存:

```
response.setDateHeader("Expires", -1);
response.setHeader("Pragma", "no-cache");
response.setHeader("Cache-control", "no-cache");
```

b. 控制浏览器缓存:

```
response.setDateHeader("Expires", System.currentTimeMillis()+1000*60*60*24); //24小时
response.setHeader("Cache-control", "max-age=60");  //60秒
```