# Response

代表Http响应的对象

## Response继承关系

ServletResponse接口 – 提供了一个Response对象应该具有的功能

|

|-- HttpServletResponse接口 – 继承了ServletResponse, 并且在此基础上添加了很多和Http协议相关的方法.

## Response常用方法

状态行

HTTP/1.1 200 OK

若干响应头

xxx: xxx

xxx: xxx

...

响应实体内容

xxxxx

设置状态码的方法:

setStatus()

设置响应头的方法

setHeader()

设置响应实体内容的方法

getOutputStream()

getWriter()

## Response对象的功能

### 向客户端浏览器发送数据

getOutputStream

getWriter

**1、getOutputStream(字节流)**

//>>向外发送数据（英文）

response.getOutputStream().write( "Hello response!".getBytes());

//>>向外发送中文数据

//getBytes方法默认使用GBK来发送数据。

response.getOutputStream().write(

"哈喽 response!".getBytes());

字节流在发送中文数据的乱码问题：

原因：(1) 在服务端通过getBytes()方法将字符串转换成二进制数据再发送给浏览器，此时默认的编码是系统平台码，也就是GBK。

(2)浏览器在接收数据时，如果使用编码不是GBK，两端编码不一致就会出现乱码问题。

解决：保证编码时和解码时使用的码表是一致的！由于在开发中我们使用最多的码表是utf-8, 因此我们应该保证使用utf-8来发送数据， 也保证浏览器使用utf-8来接收数据！

//指定浏览器在接收数据时也使用utf-8

response.setContentType(

"text/html;charset=utf-8");

//指定在服务端使用utf-8来发送数据（getBytes方法默认使用GBK来发送数据。）

response.getOutputStream().write(

"哈喽 response!".getBytes("utf-8"));

**2、getWriter(字节流)**

在通过字符流发送数据时，底层也是要转成字节再发送，如果不指定，服务器默认使用的编码为iso8859-1, 这个码表中没有中文数据，所以在发送中文数据时，不管浏览器使用什么编码来接收，都会出现乱码问题。

解决：(1)指定服务器端在发送数据时使用utf-8编码，再指定浏览器在接收时也使用相同的码表即可！

//(1)指定服务端发送数据使用utf-8

response.setCharacterEncoding("utf-8");

//(2)指定浏览器接收数据也是用utf-8

response.setContentType(

"text/html;charset=utf-8");

response.getWriter().write("哈喽 reponse!");

在实际开发中，我们只需要通过response.setContentType("text/html;charset=utf-8")这行代码就可以解决字符流在发送中文数据时的乱码问题。

/\* 这行代码既会通知服务器使用utf-8发送数据，

\* 也会通知浏览器使用utf-8来接收数据 \*/

response.setContentType(

"text/html;charset=utf-8");

response.getWriter().write("哈喽 reponse!");

总结：不管是字节流还是字符流，在发送数据时如果出现了乱码问题，都可以使用response.setContentType("text/html;charset=utf-8")这行代码来解决：

字节流：

response.setContentType(

"text/html;charset=utf-8");

response.getOutputStream().write("哈喽 reponse!".getBytes("utf-8"));

字符流：

response.setContentType(

"text/html;charset=utf-8");

response.getWriter().write("哈喽 reponse!");

开发细节：

1. getOutputStream和getWriter这两个方法不能同时使用(出现), 因为这两个方法是互斥的!
2. getOutputStream和getWriter这两个方法可以获取能够像浏览器发送数据的流, 这两个流不是指向浏览器客户端,而是指向response缓冲区. 等请求结束后, 请求会再次回到服务器,服务器会从response对象获取到数据, 组织成响应消息在发送给浏览器.
3. getOutputStream和getWriter这两个方法获取到的流, 使用完之后不需要关闭,因为浏览器会帮我们关闭!

### 实现请求重定向

可以实现资源的跳转, 既可以是服务器内部同一个WEB应用内的资源之间的跳转, 也可以是不同的WEB应用或者是不同的服务器之间的资源跳转.

实现代码:

response.setStatus(302);

response.setHeader("location", "url地址");

上面的两行代码可以简化为:

response.sendRedirect("uri地址");

重定向的特点:

1. 两次请求,两次响应
2. 地址栏地址会发生变化
3. request对象不是同一个
4. 既可以实现在同一个WEB应用内部的资源之间进行跳转, 也可以实现在不同的WEB应用或者是不同的服务器内部的资源之间进行跳转

例如:

response.sendRedirect("/day14/param.html");

response.sendRedirect("http://tmooc.cn");

### 实现定时刷新

通过refresh响应头可以实现在多少秒之后跳转到指定的资源.

代码实现:

//提示用户注册成功, 3秒之后将会跳转到主页

//>>处理响应正文乱码

response.setContentType(

"text/html;charset=utf-8");

response.getWriter().write("恭喜您注册成"

+ "功! 3秒之后将会跳转到主页...");

//实现3秒之后刷新到主页

response.setHeader("refresh",

"3;url=/day15/index.html");

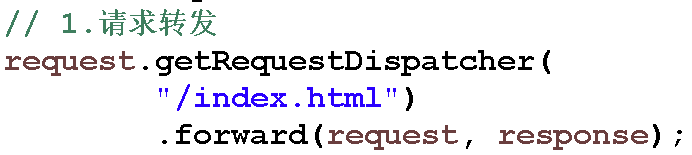
**定时刷新的特点:**

1. 两次请求, 两次响应
2. 地址栏地址会发生变化
3. request对象不是同一个
4. 既可以实现在不同的WEB应用或者不同的主机之间进行跳转, 也可以实现在同一个WEB应用内的资源之间进行跳转
5. 和重定向不同的是, 定时刷新可以实现在多少秒之后再进行跳转, 而重定向是立即跳转!

## 三种资源跳转方式的区别和使用场景

### 三种资源跳转方式的区别

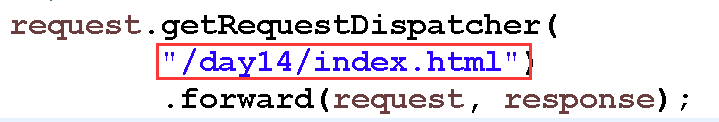
**1、请求转发**

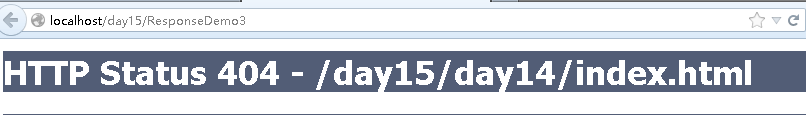


（1）一次请求，一次响应（request对象是同一个）

（2）地址栏地址不会发生变化

（3）转发只能在同一个WEB应用内部的资源之间进行跳转，不能在不同的WEB应用或者是不同的虚拟主机之间进行跳转。否则将会提示找不到资源.



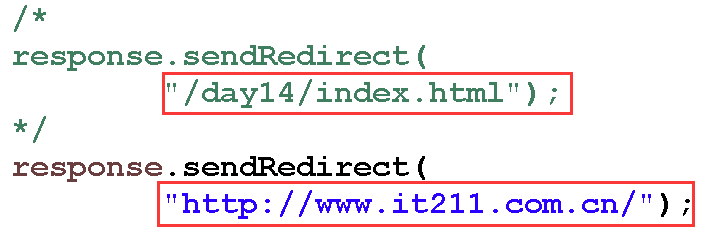


**2、重定向**

（1）两次请求，两次响应(request对象不是同一个)

（2）地址栏地址会发生变化

（3）既可以在同一个WEB应用内部的资源之间进行跳转，也可以在不同的WEB应用或者是不同的虚拟主机之间进行跳转，如下:

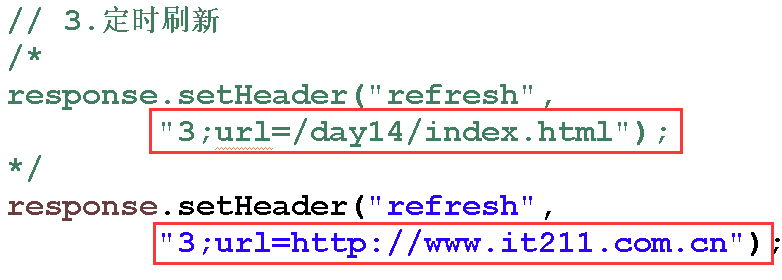


**3、定时刷新**

（1）两次请求，两次响应(request对象不是同一个)

（2）地址栏地址会发生变化

（3）既可以在同一个WEB应用内部的资源之间进行跳转，也可以在不同的WEB应用或者是不同的虚拟主机之间进行跳转，如下:



（4）定时刷新和重定向主要的区别在于，重定向是立即跳转,而定时刷新是在指定多少秒之后立即跳转，并且在跳转之前,可以向浏览器发送响应数据并维系一段时间.

### 三种资源跳转方式的使用场景

**1、如果是同一个WEB应用内部资源的跳转，三种方式都可以，具体使用哪一种:**

（1）如果希望在跳转之后地址栏地址不会发生变化，这里只能使用请求转发

（2）如果希望在跳转之后地址栏地址会发生变化，这里可以使用重定向或定时刷新

（3）如果在跳转的过程中，希望带数据到目的地，这里只能使用请求转发

（4）如果没有什么特别的需求，仅仅是做一个跳转，推荐使用转发. 因为转发是一次请求，可以减少访问服务器的次数，减少服务器的压力!

**2、如果是不同的WEB应用之间资源的跳转，只能使用重定向和定时刷新，具体使用哪一种:**

重定向是立即跳转，中间没有间隔，而定时刷新可以指定多少秒之后再进行跳转，在跳转之前，还可以发送响应到客户端，并维系一段时间.

（1）因此如果需要指定多少时间之后再跳转可以使用定时刷新，如果在跳转之前，需要提示用户，可以使用定时刷新.

（2）如果没有什么特殊需要，两种方式都可以.

# ServletContext

## 概述

### 什么ServletContext对象

代表整个WEB应用的对象

服务器启动并加载WEB应用之后，立即创建代表当前WEB应用的ServletContext对象。创建之后该对象会一直驻留在服务器的内存中，唯一的代表当前WEB应用。直到服务器关闭，或者是WEB应用被移出容器，随着WEB应用的销毁，ServletContext对象也会跟着销毁。

## 获取ServletContext对象

在servlet中提供了一个方法，getServletContet()，可以直接获取ServletContext对象。

ServletContext context = this.getServletContext();

## 作为域对象使用

### 作为域对象使用

1、域对象提供的方法

setAttribute(String name, Object value) -- 往ServletContext域中添加一个域属性.

getAttribute(String name) -- 通过属性名称获取指定名称的属性

removeAttribute(String name) -- 通过属性名称删除指定名称的属性

getAttributeNames() -- 获取所有属性名称组成的枚举

2、域对象的特征：

(1)生命周期：WEB应用被加载之后，服务器会立即创建代表当前WEB应用的ServletContext对象，服务器关闭或者WEB应用被移出容器，ServletContext对象将会销毁.

(2)作用范围：在整个WEB应用范围内

(3)主要功能：在整个 WEB应用范围内实现资源的共享

示例：

//获取ServletContext对象

ServletContext context = **this**.getServletContext();

context.setAttribute("name", "尼古拉斯赵四");

context.setAttribute("age", 38);

context.setAttribute("addr", "东北");

System.***out***.println("demo1...");

//获取ServletContext对象

ServletContext context = **this**.getServletContext();

//获取ServletContext域中的数据

String name = (String) context.getAttribute("name");

**int** age = (Integer) context.getAttribute("age");

String addr = (String) context.getAttribute("addr");

System.***out***.println(name+" : "+age+" : "+addr);

### 案例：统计网站访问人次

//1.获取ServletContext对象

ServletContext context = **this**

.getServletContext();

/\* 2.判断ServletContext域中是否包含统计

\* 访问次数的计数器

\*/

**int** count = 0;

**if**(context.getAttribute("count") == **null**){

//>>如果是第一次访问, 将计数器存入域中

context.setAttribute("count", ++count);

}**else**{

//>>如果不是第一次访问, 取出计数器+1再存入域中

count = (**int**) context.getAttribute("count");

context.setAttribute("count", ++count);

}

//3.响应

response.setContentType(

"text/html;charset=utf-8");

response.getWriter().write(

"您是第"+count

+"位访问本网站的用户...");