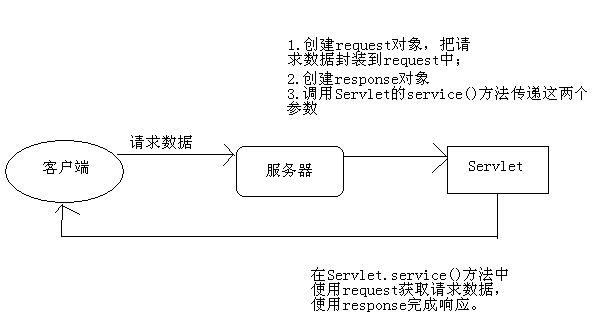
# Day03

## 请求响应流程图



## response

### 1 response概述

response是Servlet.service方法的一个参数，类型为javax.servlet.http.HttpServletResponse。在客户端发出每个请求时，服务器都会创建一个response对象，并传入给Servlet.service()方法。response对象是用来对客户端进行响应的，这说明在service()方法中使用response对象可以完成对客户端的响应工作。

response对象的功能分为以下四种：

* 设置响应头信息；
* 发送状态码；
* 设置响应正文；
* 重定向；

### 2　response响应正文

response是响应对象，向客户端输出响应正文（响应体）可以使用response的响应流，repsonse一共提供了两个响应流对象：

* PrintWriter out = response.getWriter()：获取字符流；
* ServletOutputStream out = response.getOutputStream()：获取字节流；

当然，如果响应正文内容为字符，那么使用response.getWriter()，如果响应内容是字节，例如下载时，那么可以使用response.getOutputStream()。

注意，在一个请求中，不能同时使用这两个流！也就是说，要么你使用repsonse.getWriter()，要么使用response.getOutputStream()，但不能同时使用这两个流。不然会抛出[IllegalStateException](http://java.sun.com/j2se/1.5/docs/api/java/lang/IllegalStateException.html" \o "class or interface in java.lang)异常。

#### 2.1　字符响应流

* 字符编码

在使用response.getWriter()时需要注意默认字符编码为ISO-8859-1，如果希望设置字符流的字符编码为utf-8，可以使用response.setCharaceterEncoding(“utf-8”)来设置。这样可以保证输出给客户端的字符都是使用UTF-8编码的！

但客户端浏览器并不知道响应数据是什么编码的！如果希望通知客户端使用UTF-8来解读响应数据，那么还是使用response.setContentType("text/html;charset=utf-8")方法比较好，因为这个方法不只会调用response.setCharaceterEncoding(“utf-8”)，还会设置content-type响应头，客户端浏览器会使用content-type头来解读响应数据。

* 缓冲区

response.getWriter()是PrintWriter类型，所以它有缓冲区，缓冲区的默认大小为8KB。也就是说，在响应数据没有输出8KB之前，数据都是存放在缓冲区中，而不会立刻发送到客户端。当Servlet执行结束后，服务器才会去刷新流，使缓冲区中的数据发送到客户端。

如果希望响应数据马上发送给客户端：

* 向流中写入大于8KB的数据；
* 调用response.flushBuffer()方法来手动刷新缓冲区；

### 3　设置响应头信息

　　可以使用response对象的setHeader()方法来设置响应头！使用该方法设置的响应头最终会发送给客户端浏览器！

* response.setHeader(“content-type”, “text/html;charset=utf-8”)：设置content-type响应头，该头的作用是告诉浏览器响应内容为html类型，编码为utf-8。而且同时会设置response的字符流编码为utf-8，即response.setCharaceterEncoding(“utf-8”)；
* response.setHeader("Refresh","5; URL=http://www.baidu.com")：5秒后自动跳转到百度主页。

### 4　设置状态码及其他方法

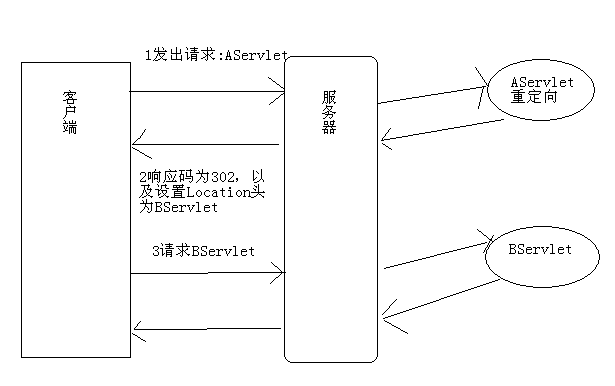
* response.setContentType("text/html;charset=utf-8")：等同与调用response.setHeader(“content-type”, “text/html;charset=utf-8”)；
* response.setCharacterEncoding(“utf-8”)：设置字符响应流的字符编码为utf-8；
* response.setStatus(200)：设置状态码；
* response.sendError(404, “您要查找的资源不存在”)：当发送错误状态码时，Tomcat会跳转到固定的错误页面去，但可以显示错误信息。

### 5　重定向

#### 5.1　什么是重定向

当你访问http://www.sun.com时，你会发现浏览器地址栏中的URL会变成http://www.oracle.com/us/sun/index.htm，这就是重定向了。

重定向是服务器通知浏览器去访问另一个地址，即再发出另一个请求。



#### 5.2　完成重定向

响应码为200表示响应成功，而响应码为302表示重定向。所以完成重定向的第一步就是设置响应码为302。

因为重定向是通知浏览器再第二个请求，所以浏览器需要知道第二个请求的URL，所以完成重定向的第二步是设置Location头，指定第二个请求的URL地址。

|  |
| --- |
| **public** **class** AServlet **extends** HttpServlet {  **public** **void** doGet(HttpServletRequest request, HttpServletResponse response)  **throws** ServletException, IOException {  response.setStatus(302);  response.setHeader("Location", "http://www.baidu.com");  }  } |

　　上面代码的作用是：当访问AServlet后，会通知浏览器重定向到百度主页。客户端浏览器解析到响应码为302后，就知道服务器让它重定向，所以它会马上获取响应头Location，然发出第二个请求。

#### 5.3　便捷的重定向方式

|  |
| --- |
| **public** **class** AServlet **extends** HttpServlet {  **public** **void** doGet(HttpServletRequest request, HttpServletResponse response)  **throws** ServletException, IOException {  response.sendRedirect("http://www.baidu.com");  }  } |

response.sendRedirect()方法会设置响应头为302，以设置Location响应头。

如果要重定向的URL是在同一个服务器内，那么可以使用相对路径，例如：

|  |
| --- |
| **public** **class** AServlet **extends** HttpServlet {  **public** **void** doGet(HttpServletRequest request, HttpServletResponse response)  **throws** ServletException, IOException {  response.sendRedirect("/hello/BServlet");  }  } |

重定向的URL地址为：http://localhost:8080/hello/BServlet。

#### 5.4　重定向小结

重定向使用场景：1.跨域访问 2.希望修改浏览器地址栏的路径

* 重定向是两次请求；====增加访问压力
* 重定向的URL可以是其他应用，不局限于当前应用；
* 重定向的响应头为302，并且必须要有Location响应头；
* 重定向就不要再使用response.getWriter()或response.getOutputStream()输出数据，不然可能会出现异常；

### 6 案例

#### 6-1 响应流输出中文

参考代码

|  |
| --- |
| String data="中国人";  //写中文方式一：通知浏览器用程序指定的编码解析响应的内容  response.setHeader("content-type", "text/html;charset=utf-8");  response.getOutputStream().write(data.getBytes());//默认使用utf-8编码  //写中文方式二  response.setHeader("content-type", "text/html;charset=utf-8");  response.setCharacterEncoding("utf-8");  response.getWriter().write(data);  //写中文方式三  response.setContentType("text/html;charset=utf-8");//可以一行代码实现上面两种方式的输出  response.getWriter().write(data);  response.getOutputStream().write(data.getBytes()); |

#### 6-2 文件下载及中文文件名的解决方案

##### 6-2-1 非中文文件名的文件下载 后面讲

|  |
| --- |
| InputStream in = getServletContext().getResourceAsStream("/img/8.jpg");  OutputStream out = response.getOutputStream();  response.setContentType("text/html;charset=utf-8");  response.setHeader("content-disposition", "attachment;filename=8.jpg");  IOUtils.copy(in, out); |

##### 6-2-2 中文文件名的文件下载

|  |
| --- |
| /\*\*  \* 实现下载思路  \* 1.获取要下载的文件名称----自动生成唯一不重复的名字  \* 2.获取图片文件对应字节流  \* 3.设置下载响应头，实现下载  \*/  String fileName="八.jpg";//数据库拿 File指定目录所有文件  InputStream in = getServletContext().getResourceAsStream("/download/"+fileName);  OutputStream out = response.getOutputStream();  //获取浏览器类型，封装请求协议 request  String agent=request.getHeader("User-Agent");    //解决中文名称下载问题，火狐不识别  if(agent.toLowerCase().contains("firefox")){  //解决火狐的识别问题 IE识别  fileName=new String(fileName.getBytes("utf-8"),"iso8859-1");  }else{  fileName=URLEncoder.encode(fileName,"utf-8");  // System.out.println(fileName);  }  //设置头，告诉浏览器要以下载形式保存文件  response.setHeader("content-disposition","attachment;filename="+fileName);  //IOUtils  IOUtils.copy(in, out); |

#### 6-3 输出随机图片—验证码 【绘制图片】

验证码：登录 注册 图片

作用：削弱程序暴力破解的次数。

原因：12306购买火车票的网站------Web应用

黄牛：租几台性能特别电脑：暴力登录

理解 while(true){

Send 手机号和密码

}

暴力注册：

热力网==》代理IP地址注册信息0.~~~~~

参考代码参见案例中的-VC.java

#### 6-4 定时刷新

**自动刷新**响应头，浏览器会在3秒之后请求http://www.baidu.com：

* Refresh: 3;url=http://www.baidu.com

参考代码

|  |
| --- |
| response.setHeader("refresh", "3"); |

#### 6-5 控制浏览器不缓存当前文档内容

告诉浏览器不要缓存的响应头：

* Expires: -1；//正数
* Cache-Control: no-cache；
* Pragma: no-cache；

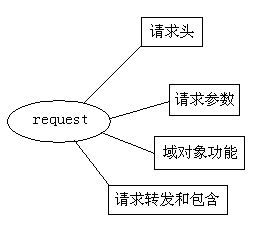
参考代码

|  |
| --- |
| package day03.servlet;  import java.awt.Color;  import java.awt.Font;  import java.awt.Graphics;  import java.awt.Graphics2D;  import java.awt.image.BufferedImage;  import java.io.IOException;  import java.util.Random;  import javax.imageio.ImageIO;  import javax.servlet.ServletException;  import javax.servlet.ServletOutputStream;  import javax.servlet.http.HttpServlet;  import javax.servlet.http.HttpServletRequest;  import javax.servlet.http.HttpServletResponse;  import sun.java2d.loops.DrawLine;  /\*\*  \* 验证码本质：一张图片  \* 动态生成一张图  \*/  public class VCServlet extends HttpServlet {  private static final long serialVersionUID = 1L;  private static final int WIDTH=120;  private static final int HEIGHT=25;  protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  //动态生成一个新图  BufferedImage image=new BufferedImage(WIDTH , HEIGHT, BufferedImage.TYPE\_INT\_RGB);  //创建画笔  Graphics g=image.getGraphics();  //填充背景颜色  fillBgColor(g);  //绘制边框  drawBorld(g);  //绘制干扰线  drawRandomLine(g);  //绘制验证码的字符  drawRandomString(g);    //图片输出浏览器上  //清除缓存  response.setHeader("cache-control", "no-cache");  response.setHeader("pragam", "no-cache");  response.setDateHeader("expires", -1);  response.setContentType("Image/jpeg");  ServletOutputStream os = response.getOutputStream();  ImageIO.write(image, "jpg", os);  }  private void fillBgColor(Graphics g) {  g.setColor(Color.WHITE);  g.fillRect(0, 0, WIDTH, HEIGHT);  }  private void drawBorld(Graphics g) {  g.setColor(Color.BLUE);  g.drawRect(1, 1, WIDTH-2, HEIGHT-2);  }  private void drawRandomLine(Graphics g) {  g.setColor(Color.RED);  for (int i = 0; i <5; i++) {  int x1 = new Random().nextInt(WIDTH);  int y1 = new Random().nextInt(HEIGHT);  int x2 = new Random().nextInt(WIDTH);  int y2 = new Random().nextInt(HEIGHT);  g.drawLine(x1, y1, x2, y2);  }  }  private void drawRandomString(Graphics g) {  Graphics2D g2=(Graphics2D)g;  g.setColor(Color.BLACK);  g.setFont(new Font("宋体", Font.BOLD, 20));  String[] content={"A","B","C","0","1","2","3","尼","额","妾","墨","啧"};  int x=10;  for (int i = 0; i <4; i++) {  //旋转  int degree=new Random().nextInt()%30;  g2.rotate(degree\*Math.PI/180,x,20);//旋转  g2.drawString(content[new Random().nextInt(content.length)], x, 20);  g2.rotate(-degree\*Math.PI/180, x, 20);  x+=32;  }  }  /\*\*  \* @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)  \*/  protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  // TODO Auto-generated method stub  doGet(request, response);  }  } |

## request

### 1　request概述

request是Servlet.service()方法的一个参数，类型为javax.servlet.http.HttpServletRequest。在客户端发出每个请求时，服务器都会创建一个request对象，并把请求数据封装到request中，然后在调用Servlet.service()方法时传递给service()方法，这说明在service()方法中可以通过request对象来获取请求数据。



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；
* request提供了请求转发和请求包含功能。

### 2　request域方法

request是域对象！在JavaWeb中一共四个域对象，其中ServletContext就是域对象，它在整个应用中只创建一个ServletContext对象。request其中一个，request可以在一个请求中共享数据。

一个请求会创建一个request对象，如果在一个请求中经历了多个Servlet，那么多个Servlet就可以使用request来共享数据。现在我们还不知道如何在一个请求中经历了几个Servlet，后面在学习请求转发和请求包含后就知道了。

下面是request的域方法：

* void setAttribute(String name, Object value)：用来存储一个对象，也可以称之为存储一个域属性，例如：servletContext.setAttribute(“xxx”, “XXX”)，在request中保存了一个域属性，域属性名称为xxx，域属性的值为XXX。请注意，如果多次调用该方法，并且使用相同的name，那么会覆盖上一次的值，这一特性与Map相同；
* Object getAttribute(String name)：用来获取request中的数据，当前在获取之前需要先去存储才行，例如：String value = (String)request.getAttribute(“xxx”);，获取名为xxx的域属性；
* void removeAttribute(String name)：用来移除request中的域属性，如果参数name指定的域属性不存在，那么本方法什么都不做；
* Enumeration getAttributeNames()：获取所有域属性的名称；

### 3　request获取请求头数据

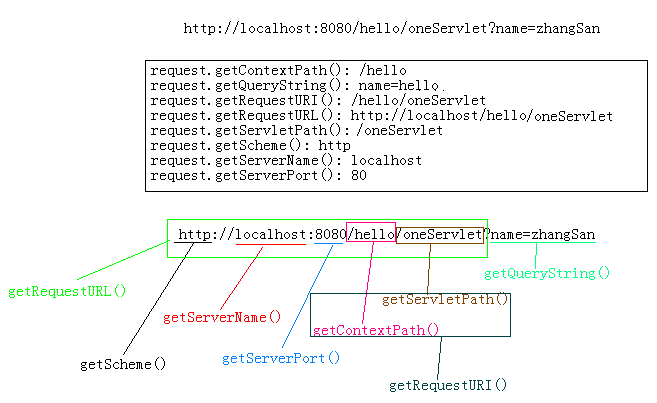
request与请求头相关的方法有：

* 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；
* int getIntHeader(String name)：获取值为int类型的请求头。

### 4　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()：获取请求体的字节数，GET请求没有请求体，没有请求体返回-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，表示请求体内容使用了URL编码；
* String getMethod()：返回请求方法，例如：GET
* Locale getLocale()：返回当前客户端浏览器的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，只对请求体有效！注意，对于GET而言，没有请求体！！！所以此方法只能对POST请求中的参数有效！
* String getContextPath()：返回上下文路径，例如：/hello 配置项目访问路径/day01
* 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；
* String getScheme()：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：8080



|  |
| --- |
| 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()); |

#### 4.1　案例：request.getRemoteAddr()：封IP

　　可以使用request.getRemoteAddr()方法获取客户端的IP地址，然后判断IP是否为禁用IP。

|  |
| --- |
| String ip = request.getRemoteAddr();  System.*out*.println(ip);  **if**(ip.equals("127.0.0.1")) {  response. getWriter().print("您的IP已被禁止！");  } **else** {  response.getWriter().print("Hello!");  } |

### 5　request获取请求参数[重点]

最为常见的客户端传递参数方式有两种：

* 浏览器地址栏直接输入：一定是GET请求；
* 超链接：一定是GET请求；
* 表单：可以是GET，也可以是POST，这取决与<form>的method属性值；

GET请求和POST请求的区别：

* GET请求：
* 请求参数会在浏览器的地址栏中显示，所以不安全；
* 请求参数长度限制长度在1K之内；
* GET请求没有请求体，无法通过request.setCharacterEncoding()来设置参数的编码；
* POST请求：
* 请求参数不会显示浏览器的地址栏，相对安全；
* 请求参数长度没有限制；

|  |
| --- |
| <a href=*"/hello/ParamServlet?p1=v1&p2=v2"*>超链接</a>  <hr/>  <form action=*"/hello/ParamServlet"* method=*"post"*>  参数1：<input type=*"text"* name=*"p1"*/><br/>  参数2：<input type=*"text"* name=*"p2"*/><br/>  <input type=*"submit"* value=*"提交"*/>  </form> |
|  |

下面是使用request获取请求参数的API：

* String getParameter(String name)：通过指定名称获取参数值；

单个数据获取：text password textarea hidden 单选按钮和select

|  |
| --- |
| **public** **void** doGet(HttpServletRequest request, HttpServletResponse response)  **throws** ServletException, IOException {  String v1 = request.getParameter("p1");  String v2 = request.getParameter("p2");  System.*out*.println("p1=" + v1);  System.*out*.println("p2=" + v2);  }    **public** **void** doPost(HttpServletRequest request, HttpServletResponse response)  **throws** ServletException, IOException {  String v1 = request.getParameter("p1");  String v2 = request.getParameter("p2");  System.*out*.println("p1=" + v1);  System.*out*.println("p2=" + v2);  } |

* String[] getParameterValues(String name)：当多个参数名称相同时，可以使用方法来获取；

获取同一个name对应的多个value，String[]

|  |
| --- |
| <a href=*"/hello/ParamServlet?name=zhangSan&name=liSi"*>超链接</a> |
| **public** **void** doGet(HttpServletRequest request, HttpServletResponse response)  **throws** ServletException, IOException {  String[] names = request.getParameterValues("name");  System.*out*.println(Arrays.*toString*(names));  } |

* Enumeration getParameterNames()：获取所有参数的名字；

|  |
| --- |
| <form action=*"/hello/ParamServlet"* method=*"post"*>  参数1：<input type=*"text"* name=*"p1"*/><br/>  参数2：<input type=*"text"* name=*"p2"*/><br/>  <input type=*"submit"* value=*"提交"*/>  </form> |
| **public** **void** doPost(HttpServletRequest request, HttpServletResponse response)  **throws** ServletException, IOException {  Enumeration names = request.getParameterNames();  **while**(names.hasMoreElements()) {  System.*out*.println(names.nextElement());  }  } |

* Map<String,String[]> getParameterMap()：获取所有参数封装到Map中，其中key为参数名，value为参数值，因为一个参数名称可能有多个值，所以参数值是String[]，而不是String。

封装formUtils一键封装javaBean对象，简化创建传递对象

|  |
| --- |
| <a href=*"/day05\_1/ParamServlet?p1=v1&p1=vv1&p2=v2&p2=vv2"*>超链接</a> |
| Map<String,String[]> paramMap = request.getParameterMap();  **for**(String name : paramMap.keySet()) {  String[] values = paramMap.get(name);  System.*out*.println(name + ": " + Arrays.*toString*(values));  } |
| p2: [v2, vv2]  p1: [v1, vv1] |

使用BeanUtils的populate方法实现快速的对象属性值填充

|  |
| --- |
| try {  User user=new User();  //快速实现页面与javabean对象的创建  //前提：表-javabean 属性和类型一致 javabean属性与html中表单项name值一致  BeanUtils.populate(user, request.getParameterMap());  System.out.println(user);  } catch (Exception e) {  throw new RuntimeException(e);  } |

Date日期格式不能自动填充。解决方案如下：

|  |
| --- |
| //处理时间格式  DateConverter dateConverter = new DateConverter();  //设置日期格式  dateConverter.setPatterns(new String[]{"yyyy-MM-dd","yyyy-MM-dd HH:mm:ss"});  //注册格式  ConvertUtils.register(dateConverter, Date.class); |

### 6　请求转发和请求包含

转发作用：

实现两个动态资源之间互相调整，带着数据一起转发过去

无论是请求转发还是请求包含，都表示由多个Servlet共同来处理一个请求。例如Servlet1来处理请求，然后Servlet1又转发给Servlet2来继续处理这个请求。

#### 6.1　请求转发

在AServlet中，把请求转发到BServlet：

|  |
| --- |
| **public** **class** AServlet **extends** HttpServlet {  **public** **void** doGet(HttpServletRequest request, HttpServletResponse response)  **throws** ServletException, IOException {  System.*out*.println("AServlet");  RequestDispatcher rd = request.getRequestDispatcher("/BServlet");  rd.forward(request, response);  }  } |
| **public** **class** BServlet **extends** HttpServlet {  **public** **void** doGet(HttpServletRequest request, HttpServletResponse response)  **throws** ServletException, IOException {  System.*out*.println("BServlet");  }  } |
| Aservlet  BServlet |

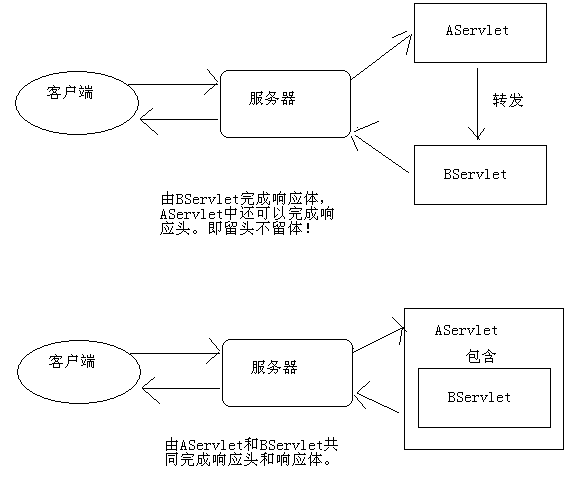
#### 6.2　请求包含

在AServlet中，把请求包含到BServlet：

|  |
| --- |
| **public** **class** AServlet **extends** HttpServlet {  **public** **void** doGet(HttpServletRequest request, HttpServletResponse response)  **throws** ServletException, IOException {  System.*out*.println("AServlet");  RequestDispatcher rd = request.getRequestDispatcher("/BServlet");  rd.include(request, response);  }  } |
| **public** **class** BServlet **extends** HttpServlet {  **public** **void** doGet(HttpServletRequest request, HttpServletResponse response)  **throws** ServletException, IOException {  System.*out*.println("BServlet");  }  } |
| Aservlet  BServlet |

#### 6.3　请求转发与请求包含比较

* 如果在AServlet中请求转发到BServlet，那么在AServlet中就不允许再输出响应体，即不能再使用response.getWriter()和response.getOutputStream()向客户端输出，这一工作应该由BServlet来完成；如果是使用请求包含，那么没有这个限制；
* 请求转发虽然不能输出响应体，但还是可以设置响应头的，例如：response.setContentType(”text/html;charset=utf-8”);
* 请求包含大多是应用在JSP页面中，完成多页面的合并；请求包含了解
* 请求转发大多是应用在Servlet中，转发目标大多是JSP页面；



#### 6.4　请求转发与重定向比较【面试喜欢问】

* 请求转发是一个请求，而重定向是两个请求；
* 请求转发后浏览器地址栏不会有变化，而重定向会有变化，因为重定向是两个请求；
* 请求转发的目标只能是本应用中的资源，重定向的目标可以是其他应用；
  + 转发不能转发其他的服务器呢？？？转发服务器内部行为，浏览器不知道的。
    - 本web应用大部分情况都是转发 MVC
  + 重定向通过地址栏修改实现请求，可以提供任意的url让浏览器重新定向
    - 适用场景：如果要请求其他服务器的资源
* 请求转发对AServlet和BServlet的请求方法是相同的，即要么都是GET，要么都是POST，因为请求转发是一个请求；
* 重定向的第二个请求一定是GET；修改浏览器的地址栏实现重定向

## 编码

### 1.1　在页面中发出请求

通常向服务器发送请求数据都需要先请求一个页面，然后用户在页面中输入数据。页面中有超链接和表单，通过超链接和表单就可以向服务器发送数据了。

因为页面是服务器发送到客户端浏览器的，所以这个页面本身的编码由服务器决定。而用户在页面中输入的数据也是由页面本身的编码决定的。

index.html

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <title>index.html</title>  <meta http-equiv=*"content-type"* content=*"text/html; charset=UTF-8"*>  </head>    <body>  <form action=*"/hello/servlet/AServlet"*>  名称:<input type=*"text"* name=*"name"*/>  <input type=*"submit"* value=*"提交"*/>  </form>  <a href=*"/hello/servlet/AServlet?name=中国"*>链接</a>  </body>  </html> |

当用户在index.html页面中输入数据时，都是UTF-8列表的。因为这个页面本身就是UTF-8编码的！

**页面的编译就是页面中输入数据的编码。**

#### 1.3　GET请求解读编码

当客户端通过GET请求发送数据给服务器时，使用request.getParameter()获取的数据是被服务器误认为ISO-8859-1编码的，也就是说客户端发送过来的数据无论是UTF-8还是GBK，服务器都认为是ISO-8859-1，这就说明我们需要在使用request.getParameter()获取数据后，再转发成正确的编码。

例如客户端以UTF-8发送的数据，使用如下转码方式：

String name = request.getParameter(“name”);

name = new String(name.getBytes(“iso-8859-1”), “utf-8”);

#### 1.4　POST请求解读编码

　　当客户端通过POST请求发送数据给服务器时，可以在使用request.getParameter()获取请求参数之前先通过request.setCharacterEncoding()来指定编码，然后再使用reuqest.getParameter()方法来获取请求参数，那么就是用指定的编码来读取了。

也就是说，如果是POST请求，服务器可以指定编码！但如果没有指定编码，那么默认还是使用ISO-8859-1来解读。

request.setCharacterEncoding(“utf-8”);

String name = request.getParameter(“name”);

### 2　响应编码

响应：服务器发送给客户端数据！响应是由response对象来完成，如果响应的数据不是字符数据，那么就无需去考虑编码问题。当然，如果响应的数据是字符数据，那么就一定要考虑编码的问题了。

response.getWriter().print(“中国”);

上面代码因为没有设置repsonse.getWriter()字符流的编码，所以服务器使用默认的编码（ISO-8859-1）来处理，因为ISO-8859-1不支持中文，所以一定会出现编码的。

所以在使用response.getWriter()发送数据之前，一定要设置response.getWriter()的编码，这需要使用response.setCharacterEncoding()方法：

response.setCharacterEncoding(“utf-8”);

response.getWriter().print(“中国”);

上面代码因为在使用response.getWriter()输出之前已经设置了编码，所以输出的数据为utf-8编码。但是，因为没有告诉浏览器使用什么编码来读取响应数据，所以很可能浏览器会出现错误的解读，那么还是会出现乱码的。当然，通常浏览器都支持来设置当前页面的编码，如果用户在看到编码时，去设置浏览器的编码，如果设置的正确那么乱码就会消失。但是我们不能让用户总去自己设置编码，而且应该直接通知浏览器，服务器发送过来的数据是什么编码，这样浏览器就直接使用服务器告诉他的编码来解读！这需要使用content-type响应头。

response.setContentType(“text/html;charset=utf-8”);

response.getWriter().print(“中国”);

　　上面代码使用setContentType()方法设置了响应头content-type编码为utf-8，这不只是在响应中添加了响应头，还等于调用了一次response.setCharacterEncoding(“utf-8”)，也就是说，通过我们只需要调用一次response.setContentType(“text/html;charset=utf-8”)即可，而无需再去调用response.setCharacterEncoding(“utf-8”)了。

在静态页面中，使用<meta>来设置content-type响应头，例如：

<meta http-equiv="content-type" content="text/html; charset=UTF-8">

## 路径的写法

<a href=””>超链接</a>

response.sendRedirect(“”);

<form action=””></form>

<img src=””/>

request.getRequestDispatcher(“”).forward(request,response);

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

预习：

Cookie

Session===============预习

Jsp简单 其实Servlet

作业：

Index.html login.html reg.html

loginServlet regServlet

注册

登录

需求：登录成功，重定向网站首页

登录失败，转发到登录页面重新登录