

** 第十讲** 服务器与Servlet入门

课程内容

Request&Response

一、Request对象

1) Request理解

1个完整的request对象，由3部分组成

请求行：包含请求地址和请求方式

Request URL: http://localhost:8080/day14/request1?name=admin

Request Method: GET

请求头：包含了请求的详细信息 以键值对的形式存在

▼ Request HeadersView parsed

POST /day14/request1 HTTP/1.1
Host: localhost:8080
Connection: keep-alive
Content-Length: 5
Pragma: no-cache
Cache-Control: no-cache
sec-ch-ua: "Chromium";v="94", "Google Chrome";v="94", ";Not A Brand";v="99"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Windows"

==请求体：包含了用户输入的数据，只有post有请求体==

▼ Form Dataview sourceview URL-encoded

name: admin

Get请求方式是以Query String方式传递数据，

name: admin

request是由tomcat创建的
request对象是用来获取请求信息的
例如请求标头、请求方法、请求参数、客户端IP，客户端浏览器等信息。

2) Request对象的体系

ServletRequest 接口
 ↑继承
HttpServletRequest 子接口
 ↑实现
org.apache.catalina.connector.RequestFacade 实现类

3) Request的作用

接收客户端的请求，获取请求中的信息。除了可以获取请求中携带的数据之外，还可以获取比如主机地址、端口、请求方式、项目名称等一系列信息。
请求分类：
请求行、请求头、请求体。

1. 获取请求行数据

请求行中，我们可以通过request对象的相应方法获取到比如协议名、服务名、端口号、项目名称、请求方式、参数列表等信息。

案例实现:

```
//通过req对象获取很多请求信息 请求地址，主机，端口号，请求方式等
System.out.println(req.getRequestURL()); //完整的请求路径
System.out.println(req.getScheme()); //协议
System.out.println(req.getServerName()); //主机名
System.out.println(req.getServerPort()); //主机端口号
System.out.println(req.getRequestURI()); //只包括项目地址
System.out.println(req.getQueryString()); //获取请求参数的字符串 get请求
```

实现效果:

```
http://localhost:8080/day14/request1
http
localhost
8080
/day14/request1
name=admin&file=
```

2. 获取请求头数据

请求头是当前对用户发送的数据的描述信息。

请求头信息在请求的时候不需要程序员手动添加，是浏览器发送的时候已经处理好的。

如果想查看请求头信息，也可以在Servlet中通过getHeader方法获取。

获取请求头数据的方法:

方法名	描述
String getHeader(String name)	根据请求头的名称获取请求头信息
Enumeration getHeaderNames()	返回此请求包含的所有头名称的枚举

案例实现:

```
//2 获取请求头信息
System.out.println(req.getHeader("Host"));
System.out.println(req.getHeader("Connection"));
System.out.println(req.getHeader("Accept"));
System.out.println("-----");
Enumeration<String> headerNames = req.getHeaderNames();
while (headerNames.hasMoreElements()){
    String key = headerNames.nextElement();
    System.out.println(key + ":" + req.getHeader(key));
}
```

实现效果:

```
host:localhost:8080
connection:keep-alive
pragma:no-cache
cache-control:no-cache
sec-ch-ua:"Chromium";v="94", "Google Chrome";v="94", ";Not A Brand";v="99"
sec-ch-ua-mobile:?0
sec-ch-ua-platform:"Windows"
upgrade-insecure-requests:1
user-agent:Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/94.0.4606.81 Safari/537.36
accept:text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8
sec-fetch-site:same-origin
sec-fetch-mode:navigate
sec-fetch-user:?1
```

3. 获取请求正文数据 请求体 请求报文 form data

① 获取流对象

只有POST请求方式，才有请求体，在请求体中封装了POST请求的请求参数
步骤: 1.获取流对象(字符流/字节流) 2.再从流对象中拿数据

获取流对象的方式:

1. `BufferedReader request.getReader()` 获取字符输入流
2. `ServletInputStream request.getInputStream()` 获取字节输入流

```
//3:获取请求体信息 数据 post
//获取字节输入流对象
// ServletInputStream is = req.getInputStream();
// int num = -1;
// while((num = is.read())!=-1){
//     System.out.print((char)num);
// }
// //关流
// is.close();

System.out.println( "===== ");
//获取字符输入流对象
BufferedReader reader = req.getReader();
String str = null;
while((str = reader.readLine())!=null){
    System.out.println(str);
}
reader.close();
```

② 获取请求参数

请求体就是请求中携带的数据，也就是我们需要获取的参数。 GET/POST方式都可以使用

获取请求参数的方法：

方法名	描述
String getParameter(String name)	根据参数名获取参数值
String[] getParameterValues(String name)	根据参数名获取参数值(可以是多个值)
Enumeration getParameterNames()	获取所有的参数名
Map<String,String[]> getParameterMap()	获取所有参数的map集合

页面实现:

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
  <form action="requestBody" method="post">
    <p>
      用户名:
      <input name="username" type="text">
    </p>
    <p>
      密码:
      <input name="password" type="password">
    </p>
    <p>
      性别:
      <input name="sex" type="radio" value="男" checked="checked">男
      <input name="sex" type="radio" value="女">女
    </p>
    <p>
      爱好:
      <input name="hobby" type="checkbox" value="唱歌">唱歌
      <input name="hobby" type="checkbox" value="跳舞">跳舞
      <input name="hobby" type="checkbox" value="上网">上网
      <input name="hobby" type="checkbox" value="跑步">跑步
    </p>
    <p>
      地址:
      <select name="address">
        <option value="北京">北京</option>
        <option value="天津">天津</option>
        <option value="上海">上海</option>
      </select>
    </p>
  </form>
</body>
</html>
```

```

<p>
    个人简介: <br>
    <textarea name="introduce" rows="4" cols="40"></textarea>
</p>
<p>
    <input type="submit" value="提交">
    <input type="reset" value="取消">
</p>
</form>
</body>
</html>

```

案例实现1:

```

package com.offcn.servlet;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.util.Arrays;
import java.util.Enumeration;
import java.util.Map;
import java.util.Set;

/**
 * CompanyName: IT优就业<br/>
 * Datetime: 2022/1/26 20:51<br/>
 *
 * @author TanJian
 */
public class Request2Servlet extends HttpServlet {
    @Override
    protected void service(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
        //获取数据方式1 getParameter 一次获取一个数据
        // String username = req.getParameter("username");
        // String password = req.getParameter("password");
        // String sex = req.getParameter("sex");
        // String address = req.getParameter("address");
        // String introduce = req.getParameter("introduce");
        //
        // //获取数据方式2 数组形式获取, 主要获取复选框的数据
        // String[] hobbies = req.getParameterValues("hobby");
        // System.out.println(Arrays.toString(hobbies));
        //
        // //获取数据方式3 获取所有的name/属性名
        // Enumeration<String> parameterNames = req.getParameterNames();
        // while(parameterNames.hasMoreElements()){
        //     String key = parameterNames.nextElement();
        //     System.out.println(key + ":" + req.getParameter(key));
        // }
    }
}

```

```

//获取数据方式4，获取Map集合
Map<String, String[]> parameterMap = req.getParameterMap();
Set<Map.Entry<String, String[]>> entries = parameterMap.entrySet();
for(Map.Entry<String, String[]> map:entries){
    String key = map.getKey();
    String[] value = map.getValue();
    System.out.println(key + ":" + Arrays.toString(value));
}

}

}

```

③ 中文乱码

Get:

默认编码类型是：application/x-www-form-urlencoded；Tomcat8之前的版本，默认编码格式是iso-8859-1，从Tomcat8版本之后默认编码改为UTF-8,所以如果是Tomcat8及以上版本就不需要进行转码处理，如果是Tomcat7及之前版本可以使用以下方法进行转码：

```

String name = request.getParameter("name");
String encodingName=new String(name.getBytes("iso-8859-1"), "utf-8");

```

Post:

支持多种编码类型，application/x-www-form-urlencoded 或 multipart/form-data。可以使用以下方法进行转码：

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

在这里顺便说一下响应的乱码处理方法，后面我们还会详细讲解

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

4. Servlet域对象：共享数据

域对象：有作用域的对象就是域对象。域对象可以用来存值并在不同组件之间进行传递，域对象限制了数据的访问范围，其值会随着对象的消失而消失。

Servlet中提供了3个域对象：

1. request 作用域的值是在一次请求范围内有效（周期短）。
- 2: session 作用范围是一次会话（包含多次请求）（浏览器开关之间，默认时间30分钟）
2. ServletContext是一个全局作用域对象，在整个Web应用内都有效（周期长）。关闭服务器或者重启服务器，数据失效 上下文对象

常用方法:

方法名	描述
void setAttribute(String name,Object obj)	向作用域中存储数据
Object getAttribute(String name)	从作用域中获取数据
void removeAttribute(String name)	从作用域中移除数据

6. 跳转方式

1: 浏览器端的跳转

1.1: 表单的submit提交跳转 action = "url" method = GET/POST

1.2: 超链接的跳转方式 [跳转](#) GET

1.3: JS的跳转方式 location.href = "url?name=value&name1=value1" GET

2: 服务器端的跳转

2.1:请求转发 req.getRequestDispatcher("success").forward(req,resp);

2.2:重定向 resp.sendRedirect("success");

面试题：请求转发和重定向的区别？

二、Response对象

1) Response理解

response是由tomcat创建的
response对象是用来设置响应信息

2) Response对象的体系

ServletResponse 接口
↑继承
HttpServletResponse 接口
↑实现
org.apache.catalina.connector.ResponseFacade 实现类

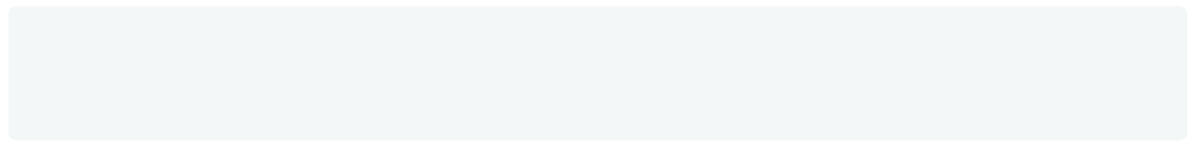
3) Response的作用

针对页面发送的请求做出数据响应，向页面输出信息，包括文本、图片、视频等。
响应分类：
响应行、响应头、响应体。

1. 设置响应行

响应行中包含的信息：可以通过开发者模式F12-Network查看

① HTTP协议



② 状态码



2. 设置响应头

当我们在浏览器中打开network工具时，每一次的请求响应数据，都可以被捕捉到，而在内容中 Response Headers 中的内容就是当前这一次请求响应的响应头信息。

设置响应头信息可以通过以下两种方法：

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

二者的区别：

`response.setHeader(String name, String value)`;一个关键字对应一个值，如果设置了多个值，则会覆盖。

`response.addHeader(String name, String value)`;一个关键字可以对应多个值
在实际开发中，同一个响应头信息只会对应一个值，所以在使用时一般没什么区别。

servlet的默认响应格式：text/html格式，但是我们为了区分编码集，需要重写

案例实现 response的默认的响应格式：文本格式 tomcat/conf/web.xml 查看格式

查看servlet支持的响应格式

3. 设置响应体

响应的数据就是响应体。响应对象response在返回数据、响应数据的时候，会将一些HTML、text、流数据等信息通过响应主体返回给页面，而响应体绝大多数都是文本类型。

响应数据需要通过流来进行数据传输，而response自带的流有两个：

`response.getWriter()` ==> `PrintWriter` 输出文本信息

`response.getOutputStream()` ==> `ServletOutputStream` 输出字节信息，比如图片、音频、视频

需要注意：

这两个流不能同时存在。

案例: 响应表格

案例实现:

4) 案例：文件下载

文件下载分为两种下载方法:

1. 超链接下载: 使用a标签可以下载文件, 但具有局限性, 如果能被浏览器识别的文件格式会直接打开显示, 如果浏览器无法识别, 则进行下载。
2. 后台下载

① 超链接下载

案例实现:

```
<a href="img/1.png">响应图片</a> <br>
```

② 后台下载

页面实现:

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>Title</title>
</head>
<body>

<a href="resp3?filename=1.png">下载图片</a>

</body>
</html>
```

案例实现:

```
package com.offcn.servlet;

import sun.misc.IOUtils;

import javax.servlet.ServletContext;
import javax.servlet.ServletException;
import javax.servlet.ServletOutputStream;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.FileInputStream;
import java.io.IOException;
import java.net.URLEncoder;

/**
 * CompanyName: IT优就业<br/>
 * Datetime: 2021/6/17 17:08<br/>
```

```

*
* @author TanJian
*/
public class Resp3Servlet extends HttpServlet {

    @Override
    protected void service(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
        //1:设置乱码
        req.setCharacterEncoding("utf-8");
        resp.setContentType("text/html;charset=utf-8");

        //2:接受文件的名称
        String filename = req.getParameter("filename");

        //告诉浏览器以图片的格式输出
        //告知浏览器要以下载的方式下载
        resp.setHeader("content-disposition", "attachment;filename="+
URLLEncoder.encode(filename, "utf-8"));

        //3:给出响应
        //1.获取一个输入流 服务器不能使用使用相对路径
        //获取文件的真实路径
        ServletContext servletContext = req.getServletContext();
        //把相对路径改为绝对路径
        String path = servletContext.getRealPath("img/"+filename);
        FileInputStream fis = new FileInputStream(path);

        System.out.println();

        //获取一个输出流
        ServletOutputStream os = resp.getOutputStream();
        int num = -1;
        while((num = fis.read())!=-1){
            os.write(num);
        }

        //4:关闭资源
        fis.close();
        os.close();

    }
}

```

三、Servlet3.0注解模式

简化servlet的配置，不在使用web.xml文件进行配置。

语法:

常用语法格式: @WebServlet("/路径")

完整版注解格式

```
@WebServlet(urlPatterns = {"/test","/a"},loadOnStartup = 1,initParams =
{@WebInitParam(name = "encoding",value = "utf-8")})
```

四、验证码

```
package com.offcn.util;
import java.awt.Color;
import java.awt.Font;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.RenderingHints;
import java.awt.geom.AffineTransform;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.OutputStream;
import java.util.Arrays;
import java.util.Random;
import javax.imageio.ImageIO;
import javax.servlet.ServletContext;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
@WebServlet("/code")
public class VerifyCodeUtils extends HttpServlet{

    @Override
    protected void service(HttpServletRequest req, HttpServletResponse resp) throws
    ServletException, IOException {

        //生成随机字符串
        String verifyCode = VerifyCodeUtils.generateVerifyCode(4);
        //存入会话session
        HttpSession session = req.getSession();
        session.setAttribute("verCode", verifyCode.toLowerCase());

        //生成图片
        int w = 100, h = 30;
        VerifyCodeUtils.outputImage(w, h, resp.getOutputStream(), verifyCode);

    }

    //使用到Algerian字体，系统里没有的话需要安装字体，字体只显示大写，去掉了1,0,i,o几个容易混
    淆的字符
```

```

public static final String VERIFY_CODES = "23456789ABCDEFGHJKLMNPQRSTUVWXYZ";
private static Random random = new Random();

/**
 * 使用系统默认字符源生成验证码
 * @param verifySize 验证码长度
 * @return
 */
public static String generateVerifyCode(int verifySize){
    return generateVerifyCode(verifySize, VERIFY_CODES);
}

/**
 * 使用指定源生成验证码
 * @param verifySize 验证码长度
 * @param sources 验证码字符源
 * @return
 */
public static String generateVerifyCode(int verifySize, String sources){
    if(sources == null || sources.length() == 0){
        sources = VERIFY_CODES;
    }
    int codesLen = sources.length();
    Random rand = new Random(System.currentTimeMillis());
    StringBuilder verifyCode = new StringBuilder(verifySize);
    for(int i = 0; i < verifySize; i++){
        verifyCode.append(sources.charAt(rand.nextInt(codesLen-1)));
    }
    return verifyCode.toString();
}

/**
 * 生成随机验证码文件,并返回验证码值
 * @param w
 * @param h
 * @param outputFile
 * @param verifySize
 * @return
 * @throws IOException
 */
public static String outputVerifyImage(int w, int h, File outputFile, int verifySize) throws
IOException{
    String verifyCode = generateVerifyCode(verifySize);
    outputImage(w, h, outputFile, verifyCode);
    return verifyCode;
}

/**
 * 输出随机验证码图片流,并返回验证码值
 * @param w
 * @param h
 * @param os
 * @param verifySize
 * @return
 * @throws IOException

```

```

    */
    public static String outputVerifyImage(int w, int h, OutputStream os, int verifySize) throws
IOException{
        String verifyCode = generateVerifyCode(verifySize);
        outputImage(w, h, os, verifyCode);
        return verifyCode;
    }

    /**
     * 生成指定验证码图像文件
     * @param w
     * @param h
     * @param outputFile
     * @param code
     * @throws IOException
     */
    public static void outputImage(int w, int h, File outputFile, String code) throws
IOException{
        if(outputFile == null){
            return;
        }
        File dir = outputFile.getParentFile();
        if(!dir.exists()){
            dir.mkdirs();
        }
        try{
            outputFile.createNewFile();
            FileOutputStream fos = new FileOutputStream(outputFile);
            outputImage(w, h, fos, code);
            fos.close();
        } catch(IOException e){
            throw e;
        }
    }

    /**
     * 输出指定验证码图片流
     * @param w
     * @param h
     * @param os
     * @param code
     * @throws IOException
     */
    public static void outputImage(int w, int h, OutputStream os, String code) throws
IOException{
        int verifySize = code.length();
        BufferedImage image = new BufferedImage(w, h, BufferedImage.TYPE_INT_RGB);
        Random rand = new Random();
        Graphics2D g2 = image.createGraphics();

        g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,RenderingHints.VALUE_ANTIALIAS_
ON);
        Color[] colors = new Color[5];
        Color[] colorSpaces = new Color[] { Color.WHITE, Color.CYAN,
            Color.GRAY, Color.LIGHT_GRAY, Color.MAGENTA, Color.ORANGE,

```

```

        Color.PINK, Color.YELLOW };
float[] fractions = new float[colors.length];
for(int i = 0; i < colors.length; i++){
    colors[i] = colorSpaces[rand.nextInt(colorSpaces.length)];
    fractions[i] = rand.nextFloat();
}
Arrays.sort(fractions);

g2.setColor(Color.GRAY); // 设置边框色
g2.fillRect(0, 0, w, h);

Color c = getRandColor(200, 250);
g2.setColor(c); // 设置背景色
g2.fillRect(0, 2, w, h-4);

// 绘制干扰线
Random random = new Random();
g2.setColor(getRandColor(160, 200)); // 设置线条的颜色
for (int i = 0; i < 20; i++) {
    int x = random.nextInt(w - 1);
    int y = random.nextInt(h - 1);
    int xl = random.nextInt(6) + 1;
    int yl = random.nextInt(12) + 1;
    g2.drawLine(x, y, x + xl + 40, y + yl + 20);
}

// 添加噪点
float yawpRate = 0.05f; // 噪声率
int area = (int) (yawpRate * w * h);
for (int i = 0; i < area; i++) {
    int x = random.nextInt(w);
    int y = random.nextInt(h);
    int rgb = getRandomIntColor();
    image.setRGB(x, y, rgb);
}

shear(g2, w, h, c); // 使图片扭曲

g2.setColor(getRandColor(100, 160));
int fontSize = h-4;
Font font = new Font("Algerian", Font.ITALIC, fontSize);
g2.setFont(font);
char[] chars = code.toCharArray();
for(int i = 0; i < verifySize; i++){
    AffineTransform affine = new AffineTransform();
    affine.setToRotation(Math.PI / 4 * rand.nextDouble() * (rand.nextBoolean() ? 1 : -1), (w
/ verifySize) * i + fontSize/2, h/2);
    g2.setTransform(affine);
    g2.drawChars(chars, i, 1, ((w-10) / verifySize) * i + 5, h/2 + fontSize/2 - 10);
}

g2.dispose();
ImageIO.write(image, "jpg", os);
}

```



```

private static Color getRandColor(int fc, int bc) {
    if (fc > 255) {
        fc = 255;
    }
    if (bc > 255) {
        bc = 255;
    }
    int r = fc + random.nextInt(bc - fc);
    int g = fc + random.nextInt(bc - fc);
    int b = fc + random.nextInt(bc - fc);
    return new Color(r, g, b);
}

private static int getRandomIntColor() {
    int[] rgb = getRandomRgb();
    int color = 0;
    for (int c : rgb) {
        color = color << 8;
        color = color | c;
    }
    return color;
}

private static int[] getRandomRgb() {
    int[] rgb = new int[3];
    for (int i = 0; i < 3; i++) {
        rgb[i] = random.nextInt(255);
    }
    return rgb;
}

private static void shear(Graphics g, int w1, int h1, Color color) {
    shearX(g, w1, h1, color);
    shearY(g, w1, h1, color);
}

private static void shearX(Graphics g, int w1, int h1, Color color) {

    int period = random.nextInt(2);

    boolean borderGap = true;
    int frames = 1;
    int phase = random.nextInt(2);

    for (int i = 0; i < h1; i++) {
        double d = (double) (period >> 1)
            * Math.sin((double) i / (double) period
                + (6.2831853071795862D * (double) phase)
                / (double) frames);
        g.copyArea(0, i, w1, 1, (int) d, 0);
        if (borderGap) {
            g.setColor(color);
            g.drawLine((int) d, i, 0, i);
            g.drawLine((int) d + w1, i, w1, i);
        }
    }
}

```

```

    }

}

private static void shearY(Graphics g, int w1, int h1, Color color) {

    int period = random.nextInt(40) + 10; // 50;

    boolean borderGap = true;
    int frames = 20;
    int phase = 7;
    for (int i = 0; i < w1; i++) {
        double d = (double) (period >> 1)
            * Math.sin((double) i / (double) period
                + (6.2831853071795862D * (double) phase)
                / (double) frames);
        g.copyArea(i, 0, 1, h1, 0, (int) d);
        if (borderGap) {
            g.setColor(color);
            g.drawLine(i, (int) d, i, 0);
            g.drawLine(i, (int) d + h1, i, h1);
        }

    }


}

}

public static void main(String[] args) throws IOException{
    String verifyCode = generateVerifyCode(4);
    System.out.println(verifyCode);
}
}

```

实现效果(验证码是随机的):

 image-20210205140523368