

Servlet

Servlet

什么是Servlet

手动实现Servlet程序

Servlet生命周期

请求的分发处理

通过继承HttpServlet实现Servlet程序

编写一个类去继承HttpServlet类

Servlet继承体系

ServletConfig类使用介绍

ServletContext类

什么是ServletContext

ServletContext类的四个作用

什么是HTTP协议？

请求的HTTP协议格式

GET请求

POST请求

常用请求头说明

GET请求与POST请求的区分

GET请求

POST请求

响应的HTTP协议格式

常用的响应码说明

MIME类型说明

HttpServletRequest类

获取请求的参数值

解决post请求中文乱码问题

请求的转发

base标签的作用

HttpServletResponse类

HttpServletResponse类的作用

两个输出流的说明

如何往客户端回传数据

解决响应的中文乱码

法一

法二（推荐使用）

请求重定向

- 方案一
方案二（推荐使用）

什么是Servlet

1. Servlet是JavaEE的规范之一。是一种接口
2. Servlet是JavaWeb三大组件之一。三大组件是：Servlet程序，Filter过滤器、Listener监听器。
3. Servlet是运行在服务器的一个Java小程序，它可以接受客户端发送过来的请求，并响应数据给客户端

手动实现Servlet程序

在WEB-INF中的web.xml

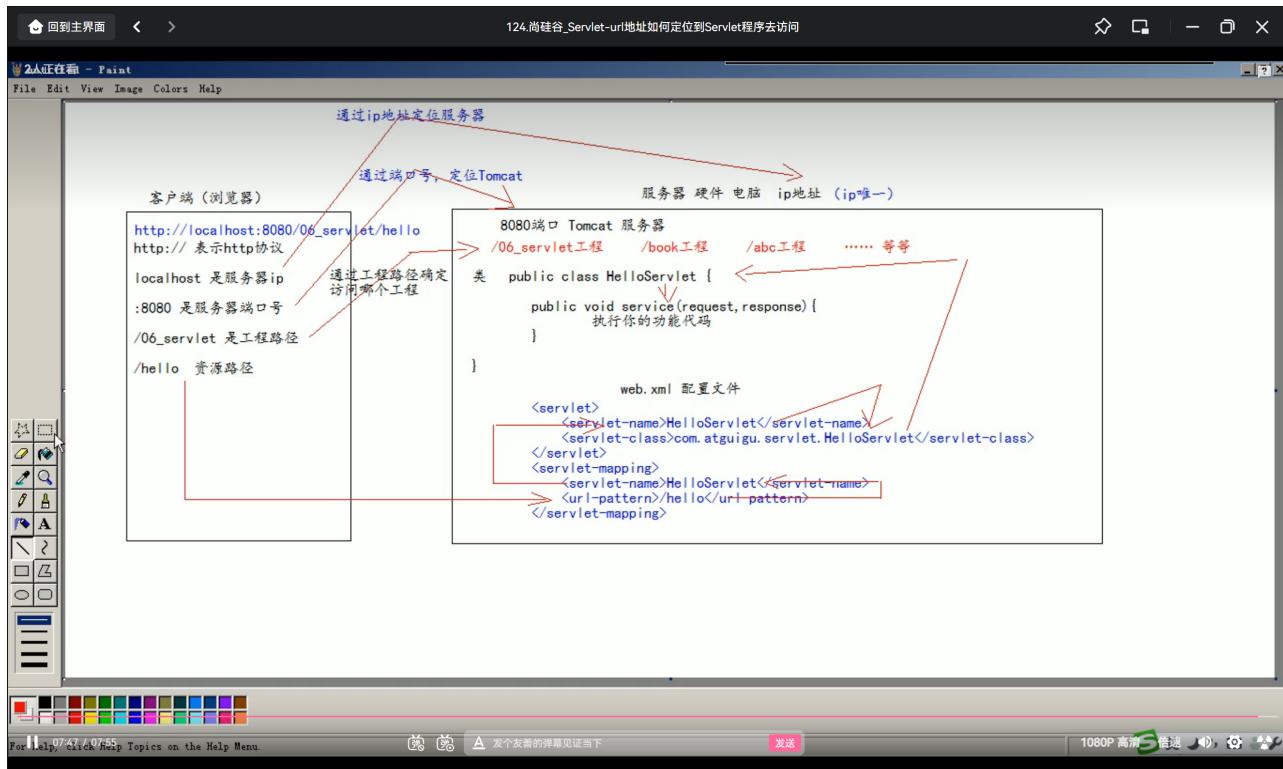
```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
3         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4         xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
5                         http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd"
6             version="4.0">
7
8     <servlet>
9         <!--
10            servlet-name标签 Servlet程序起一个别名（一般是类名）
11            servlet-class 是Servlet程序的全类名
12            -->
13         <servlet-name>HelloServlet</servlet-name>
14         <servlet-class>com.Servlet.HelloServlet</servlet-class>
15     </servlet>
16
17     <servlet-mapping>
18         <!--
19            servlet-name标签的作用是告诉服务器。当前配置的地址给哪个
20            Servlet程序使用
21            url-pattern标签配置访问地址
22            -->
```

```

21 <servlet-name>HelloServlet</servlet-name>
22 <url-pattern>/Hello</url-pattern>
23 </servlet-mapping>
24 </web-app>

```



Servlet生命周期

1. 执行Servlet构造器方法
2. 执行init初始化方法
3. 执行service方法
4. 执行destroy销毁方法

第一、二步是在第一次访问的时候创建Servlet程序会调用

第三步每次访问都会调用

第四步在Web工程停止的时候调用

请求的分发处理

```
1  public void service(ServletRequest servletRequest,
2      ServletResponse servletResponse) throws ServletException,
3      IOException {
4      //类型转换
5      HttpServletRequest httpServletRequest =
6      (HttpServletRequest) servletRequest;
7      //获取请求方式
8      String method = httpServletRequest.getMethod();
9      //打印请求
10     System.out.println(method);
11     //
12     if("GET".equals(method)){
13         doGet();
14     } else if ("POST".equals(method)) {
15         doPost();
16     }
17
18     public void doPost(){
19         //处理POST请求信息
20     }
21     public void doGet(){
22         //处理Get请求信息
23     }
```

通过继承**HttpServlet**实现**Servlet**程序

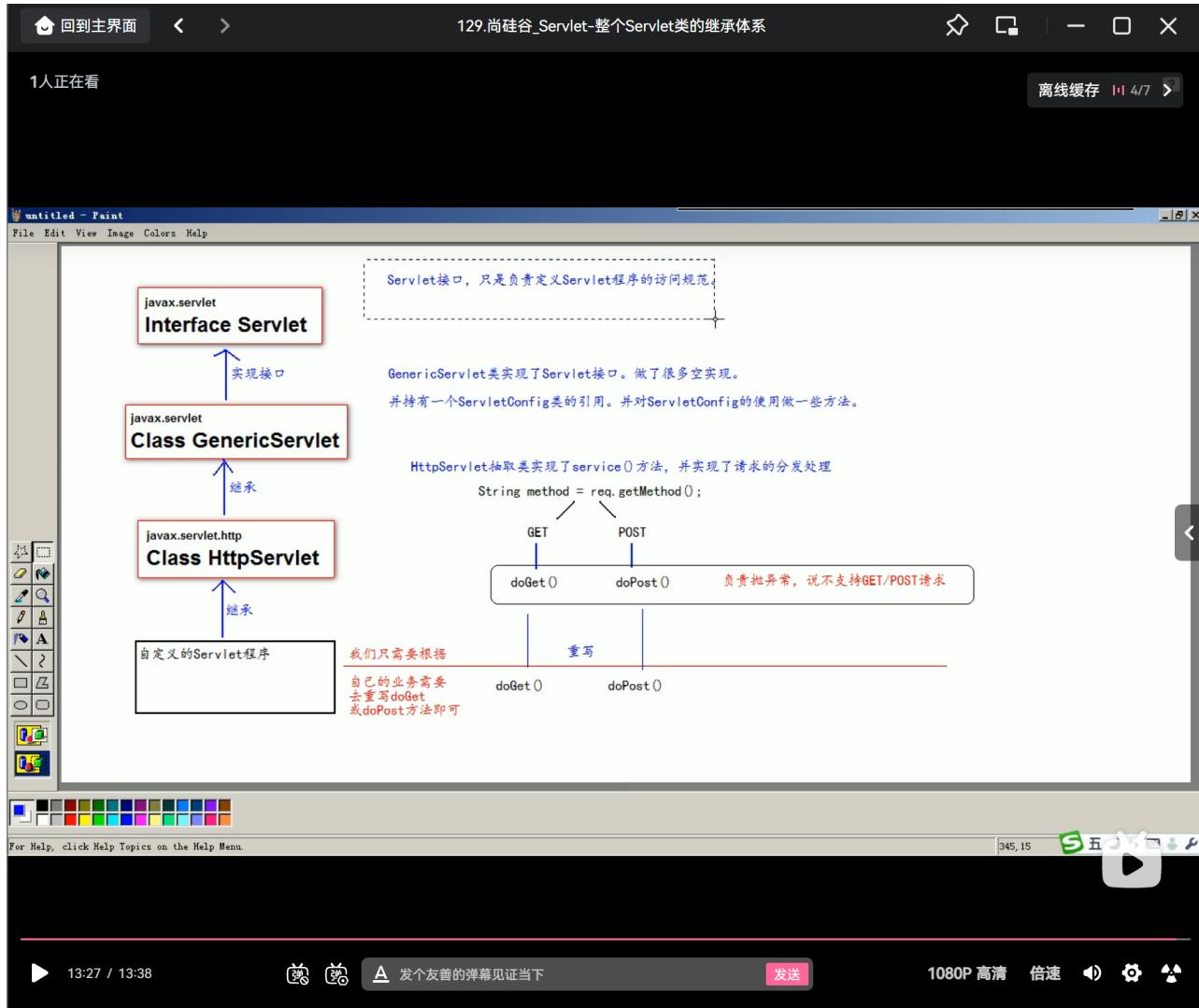
一般在实际项目开发中，都是使用继承**HttpServlet**实现**Servlet**程序。

编写一个类去继承**HttpServlet**类

```
1 public class HelloServlet extends HttpServlet {
2     @Override
3     protected void doGet(HttpServletRequest req,
4             HttpServletResponse resp) throws ServletException, IOException {
5         super.doGet(req, resp);
6     }
7     @Override
8     protected void doPost(HttpServletRequest req,
9             HttpServletResponse resp) throws ServletException, IOException {
10        super.doPost(req, resp);
11    }
12 }
```

1. 根据业务需要重写doGet或doPost方法
2. 到web.xml中的配置Servlet程序的访问地址

Servlet继承体系



ServletConfig类使用介绍

- ServletConfig类是Servlet程序的配置信息类
- Servlet程序和ServletConfig对象都是由Tomcat负责创建，由程序员负责使用
- Servlet程序默认是第一次访问的时候创建，ServletConfig是每个Servlet程序创建时，就创建一个对应的ServletConfig对象
- 其他class程序也可使用ServletConfig `ServletConfigif servletConfig = getServletConfig();` 其他程序不可获取其中的值
- 如果你要重写init()方法，要在方法第一行写上 `super.init(config);`

```

1  @Override
2      public void init(ServletConfig servletConfig) throws
3          ServletException {
4              //可以获取Servlet程序的别名servlet-name的值
5              System.out.println(servletConfig.getServletName());
6              //获取初始化init-param
7
8              System.out.println(servletConfig.getInitParameter("username"));
9
10         }

```

```

1 <init-param>
2     <param-name>username</param-name>
3     <param-value>root</param-value>
4 </init-param>

```

ServletContext类

什么是ServletContext

1. 是一个接口，表示Servlet上下文对象
2. 一个Web工程，只有一个ServletContext对象实例
3. ServletContext对象是一个域对象
4. ServletContext是在web工程部署启动的时候创建。在web工程停止的时候销毁。

域对象

域对象是可以像Map一样存取数据的对象，叫做域对象

这里的域指的是存取数据的操作范围为整个web工程

	存数据	取数据	删除数据
Map	put()	get()	remove()

存数据	取数据	删除数据
域对象	setAttribute()	getAttribute()

ServletContext类的四个作用

1. 获取web.xml中配置的上下文参数context-param

2. 获取当前的工程路径，格式：/工程名

3. 获取工程部署后在服务器硬盘上的绝对路径

4. 像Map一样存取数据

```

1 <!--context-param是上下文参数（整个web工程皆可用）-->
2 <context-param>
3   <param-name>username</param-name>
4   <param-value>context</param-value>
5 </context-param>
```

```

1 @Override
2   protected void doGet(HttpServletRequest req,
3   HttpServletResponse resp) throws ServletException, IOException {
4     //super.doGet(req, resp);
5     //super.doPost(req, resp);
6     //1. 获取web.xml中配置的上下文参数context-param
7     ServletContext context =
8       getServletConfig().getServletContext();
9
10    String username = context.getInitParameter("username_");
11    System.out.println("123456789");
12    System.out.println(username);
13    //2. 获取当前的工程路径，格式：/工程名
14    System.out.println(context.getContextPath());
15    //3. 获取工程部署后在服务器硬盘上的绝对路径
16    //"/" / " 被服务器解析地址为 http://ip:port:工程名
17    //映射到IDEA代码的web目录
18    System.out.println(context.getRealPath("/"));
19 }
```

```

1 protected void doGet(HttpServletRequest req, HttpServletResponse
  resp) throws ServletException, IOException {
2     //super.doGet(req, resp);
3     ServletContext servletContext = getServletContext();
4     servletContext.setAttribute("key", "value");
5     System.out.println("key 的值 = " +
6         servletContext.getAttribute("key"));

```

可在其他类中使用以下方法，使用以上的key值，总的来说**ServletContext**对象只能有一个且整个**Web**工程都可用

```

1
2 ServletContext context = getServletContext();
3         System.out.println(context.getAttribute("key"));

```

什么是**HTTP**协议？

是指客户端和服务器之间通信时，发送的数据，需要遵守的规则，称为**HTTP**协议。
HTTP协议中的数据又叫报文。

请求的**HTTP**协议格式

客户端给服务器发送数据叫请求。服务器给客户端回传数据叫响应。

请求分为**GET**请求和**POST**请求两种。

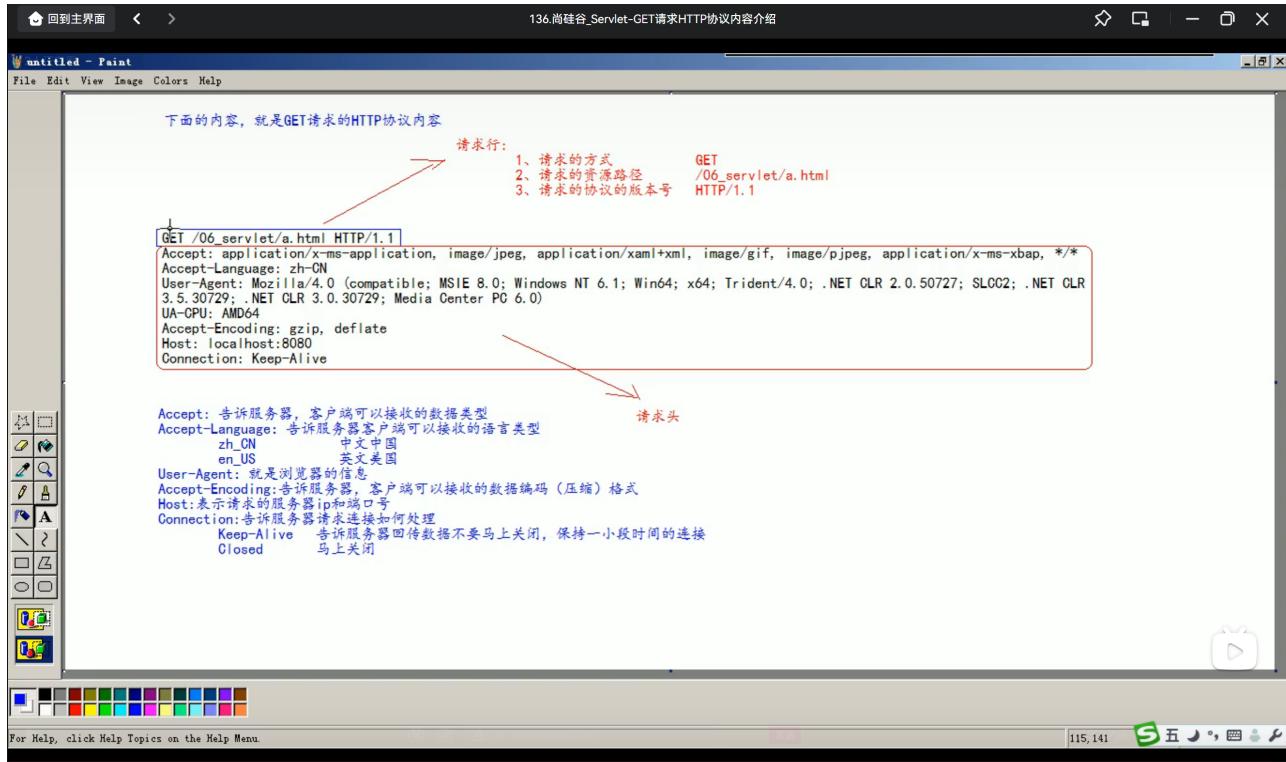
GET请求

1. 请求行

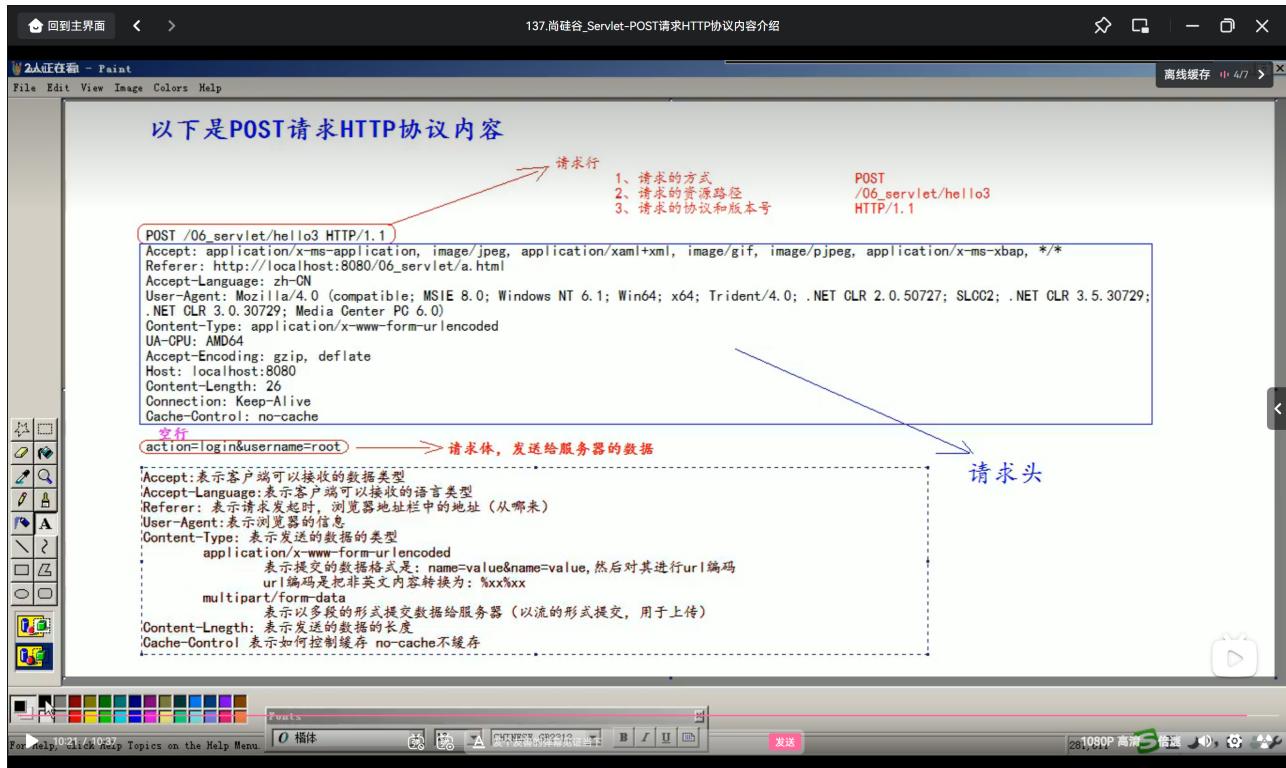
- a. 请求的方式
- b. 请求的资源路径
- c. 请求的协议的版本号

2. 请求头

- a. key:value



POST请求



常用请求头说明

如上二图

GET请求与POST请求的区分

GET请求

- from标签 `method = "get"`
- a标签
- link标签引入css
- Script标签引入JavaScript文件
- img标签引入图片
- iframe引入html页面
- 在浏览器地址栏中输入地址后回车

POST请求

- from标签 `method="post"`

响应的HTTP协议格式

1. 响应行

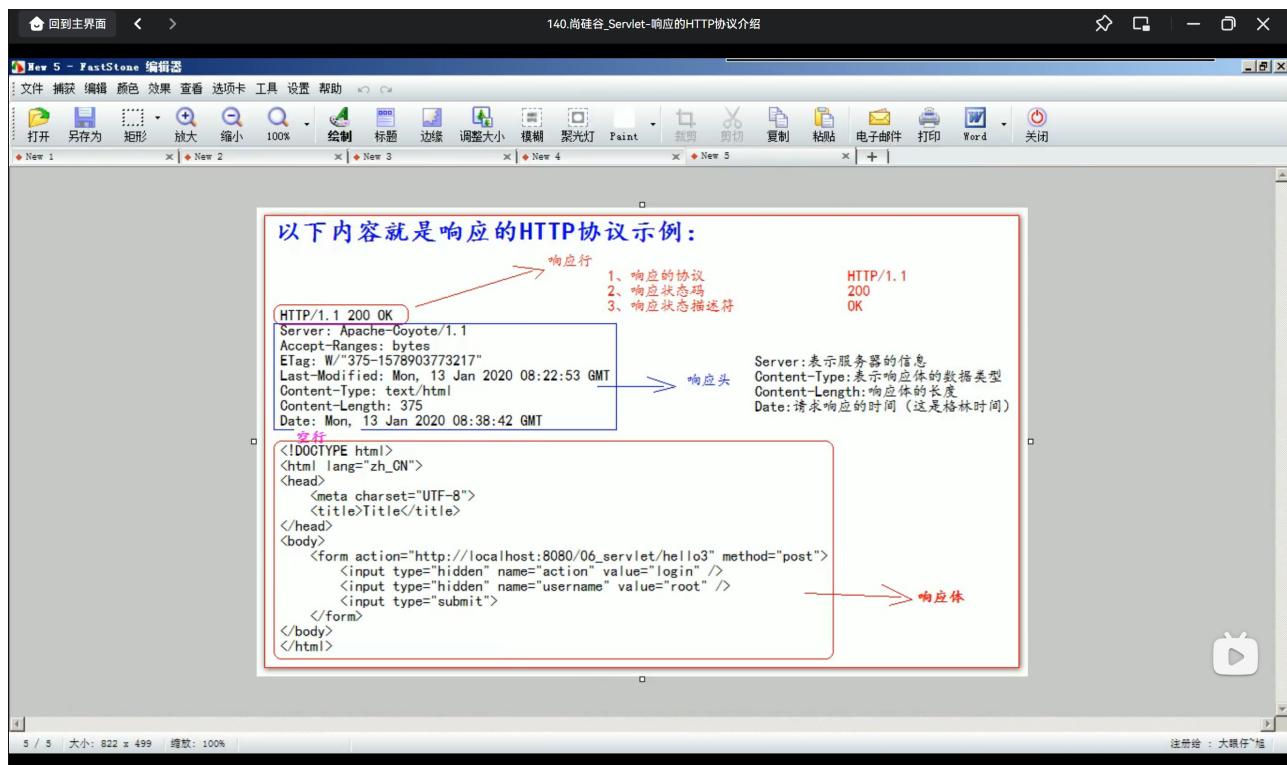
- a. 响应的协议和版本号
- b. 响应状态码
- c. 响应状态描述符

2. 响应头

- a. key:value 不同的响应头，有不同含义

3. 响应体

- a. 回传给客户端的数据



常用的响应码说明

200 表示请求成功

302 表示请求重定向

404 表示请求服务器已经收到了

500 表示服务器已经收到请求，但是服务器内部错误（代码错误）

MIME类型说明

MIME 是 HTTP 协议中数据类型。

MIME 的英文全称是 "Multipurpose Internet Mail Extensions" 多功能 Internet 邮件扩充服务。MIME 类型的格式是 “大类型/小类型”，并与某一种文件的扩展名相对应。

常见的 MIME 类型：



文件	MIME类型	
超文本标记语言文本	.html , .htm	text/html
普通文本	.txt	text/plain
RTF 文本	.rtf	application/rtf
GIF 图形	.gif	image/gif



JPEG 图形	.jpeg,.jpg	image/jpeg
au 声音文件	.au	audio/basic
MIDI 音乐文件	mid,.midi	audio/midi, audio/x-midi
RealAudio 音乐文件	.ra, .ram	audio/x-pn-realaudio
MPEG 文件	.mpg,.mpeg	video/mpeg
AVI 文件	.avi	video/x-msvideo
GZIP 文件	.gz	application/x-gzip
TAR 文件	.tar	application/x-tar

HttpServletRequest类

作用

每次只要有请求进入Tomcat服务器，Tomcat服务器就会把请求过来的HTTP协议信息解析好封装到Request对象中。然后传递到service方法（doGet和doPost）中给我们使用。我们可以通过HttpServletRequest对象，获取到所有请求的信息。

HttpServletRequest类常用方法

i. getRequestURI()	获取请求的资源路径
ii. getRequestURL()	获取请求的统一资源定位符（绝对路径）
iii. getRemoteHost()	获取客户端的 ip 地址
iv. getHeader()	获取请求头
v. getParameter()	获取请求的参数
vi. getParameterValues()	获取请求的参数（多个值的时候使用）
vii. getMethod()	获取请求的方式 GET 或 POST
viii. setAttribute(key, value);	设置域数据
ix. getAttribute(key);	获取域数据
x. getRequestDispatcher()	获取请求转发对象

```

1 public class RequestAPIServlet extends HttpServlet {
2     @Override

```

```

3   protected void doGet(HttpServletRequest req,
4       HttpServletResponse resp) throws ServletException, IOException {
5       System.out.println(req.getRequestURI());
6
7
8       //可得到访问客户端的访问者的IP地址
9       System.out.println(req.getRemoteHost());
10
11      System.out.println(req.getHeader("User-Agent"));
12
13      System.out.println(req.getMethod());
14  }
15 }
16

```

获取请求的参数值

```

1 String username = req.getParameter("username");
2 String password = req.getParameter("password");
3 String[] hobby = req.getParameterValues(Arrays.asList(hobby));

```

解决post请求中文乱码问题

```

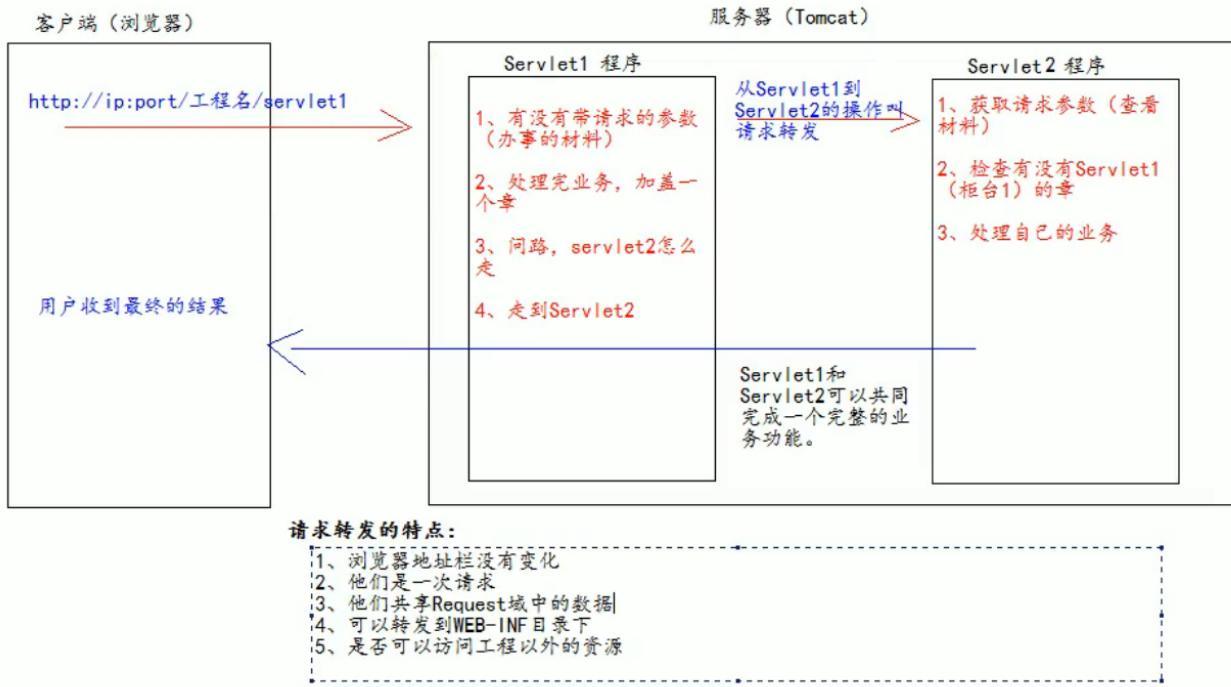
1     protected void doPost(HttpServletRequest req,
2         HttpServletResponse resp) throws ServletException, IOException
3     {//可解决post请求中文乱码问题
4         //在获取请求参数前调用才有效
5         req.setCharacterEncoding("UTF-8");
6     }

```

请求的转发

请求转发是指，服务器收到请求后，从一个资源跳转到另一个资源的操作叫请求转发

请求转发



Servlet1程序

```

1 package com.Servlet;
2
3 import javax.servlet.RequestDispatcher;
4 import javax.servlet.ServletException;
5 import javax.servlet.http.HttpServlet;
6 import javax.servlet.http.HttpServletRequest;
7 import javax.servlet.http.HttpServletResponse;
8 import java.io.IOException;
9
10 /**
11 * JunXing
12 * 2023/3/15 8:59
13 * IntelliJ IDEA
14 */
15 public class Servlet1 extends HttpServlet {
16     @Override
17     protected void doGet(HttpServletRequest req,
18             HttpServletResponse resp) throws ServletException, IOException {
19         //1. 获取请求的参数
20         String username = req.getParameter("username");

```

```

21     System.out.println("在Servlet1(柜台1)中查看参数(材料) " +
22         username);
23
24     //2.给材料盖一个章，并传递给Servlet2柜台查看
25     req.setAttribute("key1", "柜台1的章");
26
27     //3.问路：Servlet2柜台怎么走
28     //请求转发必须要以斜杆打头 / 表示地址为：http://ip:port/工程名/
29     //映射到IDEA代码的web目录
30     //RequestDispatcher requestDispatcher =
31     //req.getRequestDispatcher("/s2");
32
33
34
35
36     //走向Servlet2柜台2
37     requestDispatcher.forward(req, resp);
38 }
39 }
```

Servlet2程序

```

1 package com.Servlet;
2
3 import javax.servlet.ServletException;
4 import javax.servlet.http.HttpServlet;
5 import javax.servlet.http.HttpServletRequest;
6 import javax.servlet.http.HttpServletResponse;
7 import java.io.IOException;
8
9 /**
10  * JunXing
11  * 2023/3/15 8:59
12  * IntelliJ IDEA
13  */
14 public class Servlet2 extends HttpServlet {
```

```
15     @Override
16     protected void doGet(HttpServletRequest req,
17                           HttpServletResponse resp) throws ServletException, IOException {
18
19         //获取请求的参数（办事材料）查看
20         String username = req.getParameter("username");
21         System.out.println("在Servlet2（柜台2）中查看参数（材料）" +
22                           username);
23
24         //查看柜台1是否有章
25         Object key1 = req.getAttribute("key1");
26         System.out.println("柜台1是否有章" + key1);
27
28     }
29 }
```

base标签的作用

当我们点击a标签进行跳转的时候，浏览器地址栏中的地址是：http://localhost:8080/07_servlet/a/b/c.html

跳转回去的a标签路径是：[..../index.html](http://localhost:8080/07_servlet/index.html)

所有相对路径在工作时候都会参照当前浏览器地址栏中的地址来进行跳转。

那么参照后得到的地址是：

http://localhost:8080/07_servlet/index.html

正确的跳转路径

base标签可以设置当前页面中所有相对路径工作时，参照哪个路径来进行跳转。

当我们使用请求转发来进行跳转的时候，浏览器地址栏中的地址是

http://localhost:8080/07_servlet/forwardC

跳转回去的a标签路径是：[..../index.html](http://localhost:8080/index.html)

所有相对路径在工作时候都会参照当前浏览器地址栏中的地址来进行跳转。

那么参照后得到的地址是：

<http://localhost:8080/index.html>

错误的路径

```

1 <!--
2     base 标签设置页面相对路径工作时参照的地址
3     href 属性就是参数的地址值
4     资源路径中的资源名可省略，但资源名前的斜杆不可省略
5 -->
6 <base href="http://localhost:8080/工程名/资源路径">
```

Web中的相对路径和绝对路径

在 `javaWeb` 中，路径分为相对路径和绝对路径两种：

相对路径是：

.	表示当前目录
..	表示上一级目录
资源名	表示当前目录/资源名

绝对路径：

`http://ip:port/工程路径/资源路径`

Web中 / 斜杠的不同意义

在 web 中 / 斜杠 是一种绝对路径。

/ 斜杠 如果被浏览器解析，得到的地址是：<http://ip:port/>

```
<a href="/">斜杠</a>
```

/ 斜杠 如果被服务器解析，得到的地址是：<http://ip:port/工程路径>

```
1、<url-pattern>/servlet1</url-pattern>
2、servletContext.getRealPath("/");
3、request.getRequestDispatcher("/");
```

特殊情况： `response.sendRedirect("/");` 把斜杠发送给浏览器解析。得到 <http://ip:port/>

HttpServletResponse类

HttpServletResponse类的作用

`HttpServletResponse` 类和 `HttpServletRequest` 类一样。每次请求进来时，Tomcat 服务器都会创建一个 `Response` 对象传递给 `Servlet` 程序去使用。`HttpServletRequest` 表示请求过来的信息。`HttpServletResponse` 表示所有响应的信息，我们如果需要设置返回给客户端的信息，都可以通过 `HttpServletResponse` 对象来进行设置。

两个输出流的说明

字节流 `getOutputStream();`

- 常用于下载（传递二进制数据）

字符流 `getWriter();`

- 常用于回传字符串（常用）

两个流同时只能使用一个。使用了字节流，就不能再使用字符流，反之亦然，否则就会报错。

```

1 public class ResponseIOServlet extends HttpServlet {
2     @Override
3     protected void doGet(HttpServletRequest req,
4             HttpServletResponse resp) throws ServletException, IOException {
5         super.doGet(req, resp);
6         resp.getWriter();
7         resp.getOutputStream();
8         /**
9          * 会报错
10         * java.lang.IllegalStateException: getOutputStream() has
11         * already been called for this response
12     }

```

如何往客户端回传数据

```

1 public class ResponseIOServlet extends HttpServlet {
2     @Override
3     protected void doGet(HttpServletRequest req,
4             HttpServletResponse resp) throws ServletException, IOException {
5         PrintWriter writer = resp.getWriter();
6         writer.write("Response Content !");
7     }
8 }
```

解决响应的中文乱码

法一

```

1 public class ResponseIOServlet extends HttpServlet {
2     @Override
3     protected void doGet(HttpServletRequest req,
4             HttpServletResponse resp) throws ServletException, IOException {
5         //服务器默认字符集为: ISO-8859-1
6         System.out.println(resp.getCharacterEncoding());
7
8         //设置服务器字符集为UTF-8
9         resp.setCharacterEncoding("UTF-8");
10
11         //通过响应头, 设置浏览器也使用UTF-8字符集
12         resp.setHeader("Content-Type", "text/html; charset =
13             UTF-8");
14
15         PrintWriter writer = resp.getWriter();
16         writer.write("Response Content !");
17         writer.write("中文字符测试");
18     }
19 }
```

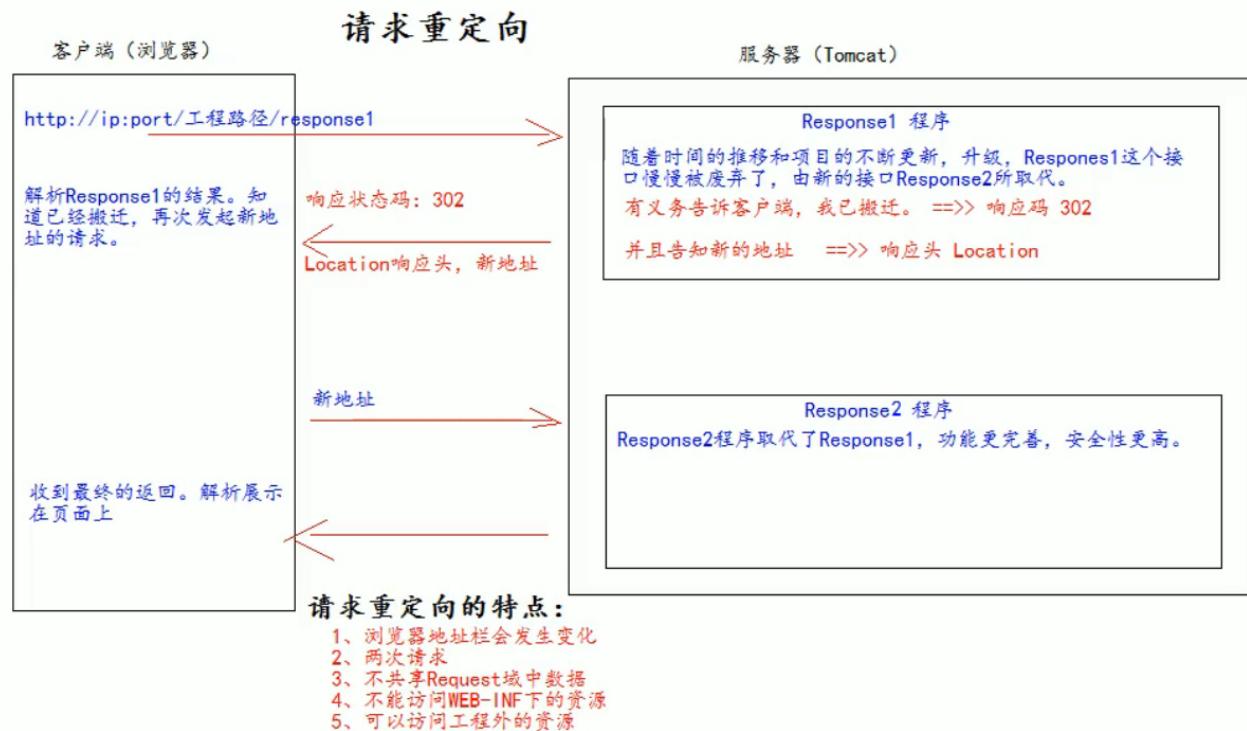
法二（推荐使用）

```

1  public class ResponseIOServlet extends HttpServlet {
2      @Override
3      protected void doGet(HttpServletRequest req,
4          HttpServletResponse resp) throws ServletException, IOException {
5          //它会同时设置服务器和客户端都使用UTF-8字符集，还设置响应头
6          //此方法一定要在获取流对象之前调用才有效
7          resp.setContentType("text/html; charset = UTF-8");
8
9          //获取流对象
10         PrintWriter writer = resp.getWriter();
11
12         writer.write("Response Content !");
13         writer.write("中文字符测试");
14     }
15 }
```

请求重定向

方案一



```

1 public class Response1 extends HttpServlet {
2     @Override
3     protected void doGet(HttpServletRequest req,
4             HttpServletResponse resp) throws ServletException, IOException {
5         System.out.println("重定向输出测试");
6
7         //设置响应状态码302, 表示重定向(已搬迁)
8         resp.setStatus(302);
9
10        resp.setHeader("Location",
11                "http://localhost:8080/Javaweb_war_exploded/res2");
12    }
13 }
```

```

1 public class Response2 extends HttpServlet {
2     @Override
3     protected void doGet(HttpServletRequest req,
4             HttpServletResponse resp) throws ServletException, IOException {
5         resp.getWriter().write("response2 result!");
6     }
6 }
```

方案二（推荐使用）

```

1 public class Response1 extends HttpServlet {
2     @Override
3     protected void doGet(HttpServletRequest req,
4             HttpServletResponse resp) throws ServletException, IOException {
5         //方案二
6
7         resp.sendRedirect("http://localhost:8080/Javaweb_war_exploded/re
8         s2");
9     }
10 }
```

