06-Servlet

讲师：王振国

今日任务

1.Servlet 技术

a)什么是 Servlet

1、Servlet 是JavaEE 规范之一。规范就是接口
2、Servlet 就JavaWeb 三大组件之一。三大组件分别是：Servlet 程序、Filter 过滤器、Listener 监听器。
3、Servlet 是运行在服务器上的一个 java 小程序，它可以接收客户端发送过来的请求，并响应数据给客户端。

b)手动实现 Servlet 程序

1、编写一个类去实现 Servlet 接口
2、实现 service 方法，处理请求，并响应数据
3、到 web.xml 中去配置 servlet 程序的访问地址

自定义的Servlet类，service()方法的作用是处理服务器的请求，并发送响应。servlet的配置在web.xml中，需要配置访问该Servlet的访问地址。

Servlet 程序的示例代码：

```
public class HelloServlet implements Servlet {
    /**
     * service 方法是专门用来处理请求和响应的
     * @param servletRequest
     * @param servletResponse
     * @throws ServletException
     * @throws IOException
     */
    @Override
    public void service(ServletRequest servletRequest, ServletResponse servletResponse) throws
ServletException, IOException {
        System.out.println("Hello Servlet 被访问了");
    }

  }
```

web.xml 中的配置：

servlet在web.xml中的配置

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd"
         version="4.0">
    <!-- servlet 标签给 Tomcat 配置 Servlet 程序 -->
    <servlet>
        <!--servlet-name 标签 Servlet 程序起一个别名（一般是类名）-->
        <servlet-name>HelloServlet</servlet-name>
        <!--servlet-class 是 Servlet 程序的全类名-->
        <servlet-class>com.atguigu.servlet.HelloServlet</servlet-class>
    </servlet>
    <!--servlet-mapping 标签给 servlet 程序配置访问地址-->
    <servlet-mapping>
        <!--servlet-name 标签的作用是告诉服务器，我当前配置的地址给哪个 Servlet 程序使用-->
        <servlet-name>HelloServlet</servlet-name>
        <!--url-pattern 标签配置访问地址 <br/>
              / 斜杠在服务器解析的时候，表示地址为：http://ip:port/工程路径          <br/>
             /hello 表示地址为：http://ip:port/工程路径/hello          <br/>
        -->
        <url-pattern>/hello</url-pattern>
    </servlet-mapping>
</web-app>
```

常见的错误 1：url-pattern 中配置的路径没有以斜杠打头。

```
javax.management.remote.rmi.RMIConnectionImpl.invoke(RMIConnectionImpl.java:829) <16 internal calls>
java.lang.Thread.run(Thread.java:745)
by: java.lang.IllegalArgumentException: Invalid <url-pattern> hello in servlet mapping
org.apache.catalina.core.StandardContext.addServletMappingDecoded(StandardContext.java:3195)
org.apache.catalina.core.StandardContext.addServletMappingDecoded(StandardContext.java:3182)
org.apache.catalina.startup.ContextConfig.configureContext(ContextConfig.java:1384)
```

常见错误 2：servlet-name 配置的值不存在：

```
nagement.remote.rmi.RMIConnectionImpl$PrivilegedOperation.run(RMIConnectionImpl.java:1309) <1 internal call
nagement.remote.rmi.RMIConnectionImpl.doPrivilegedOperation(RMIConnectionImpl.java:1408)
nagement.remote.rmi.RMIConnectionImpl.invoke(RMIConnectionImpl.java:829) <16 internal calls>
g.Thread.run(Thread.java:745)
lang.IllegalArgumentException: Servlet mapping specifies an unknown servlet name HelloServlet1
he.catalina.core.StandardContext.addServletMappingDecoded(StandardContext.java:3191)
he.catalina.core.StandardContext.addServletMappingDecoded(StandardContext.java:3182)
he.catalina.startup.ContextConfig.configureContext(ContextConfig.java:1384)
```

常见错误 3：servlet-class 标签的全类名配置错误：

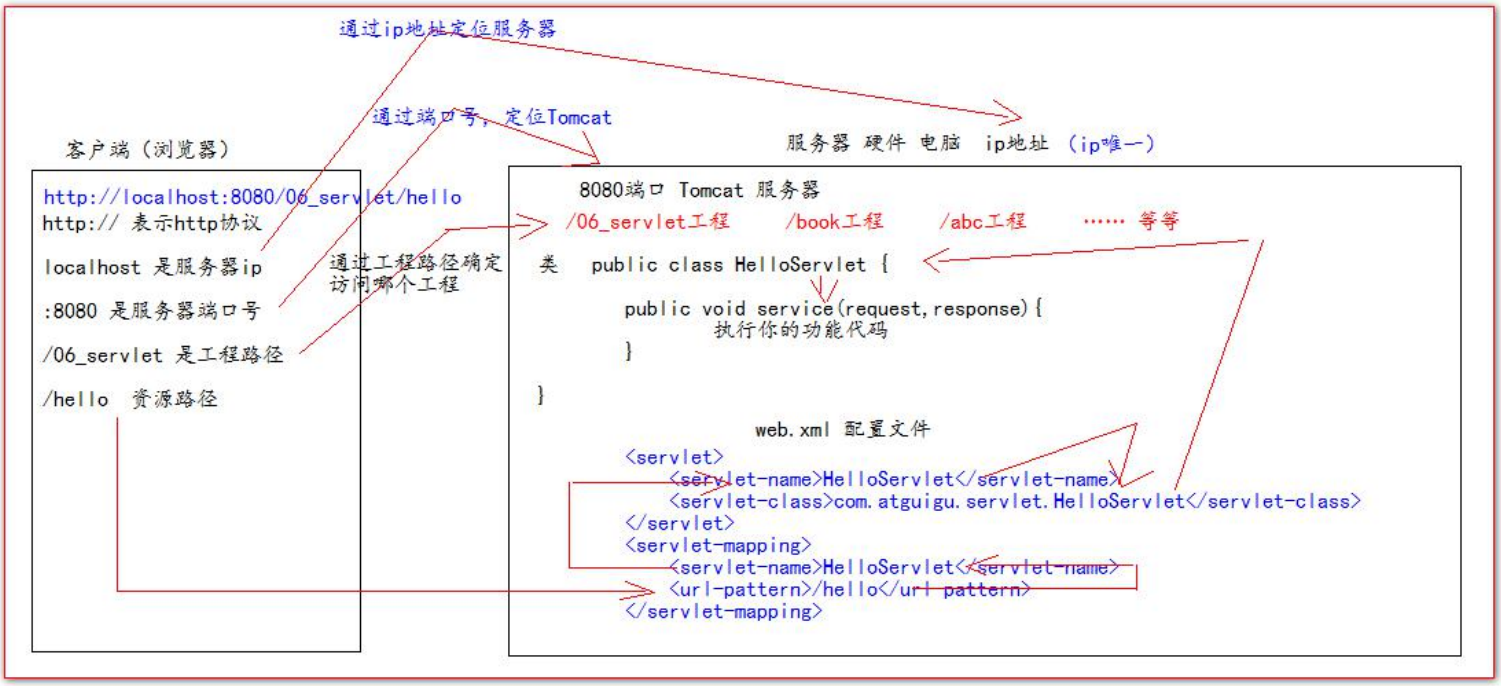
```
        org.apache.tomcat.util.threads.TaskThread$WrappingRunnable.run(TaskThread.java:6
        java.lang.Thread.run(Thread.java:745)

root cause

java.lang.ClassNotFoundException: com.atguigu.servlt.HelloServlet
        org.apache.catalina.loader.WebappClassLoaderBase.loadClass(WebappClassLoaderBase
        org.apache.catalina.loader.WebappClassLoaderBase.loadClass(WebappClassLoaderBase
        org.apache.catalina.valves.ErrorReportValve.invoke(ErrorReportValve.java:79)
        org.apache.catalina.valves.AbstractAccessLogValve.invoke(AbstractAccessLogValve.
        org.apache.catalina.connector.CoyoteAdapter.service(CoyoteAdapter.java:502)
        org.apache.coyote.http11.AbstractHttp11Processor.process(AbstractHttp11Processor
```

c)url 地址到 Servlet 程序的访问

web.xml中的servlet配置能够确定资源地址/hello要访问的Servlet类，它会默认访问service()方法



d)Servlet 的生命周期

1、执行 Servlet 构造器方法
2、执行 init 初始化方法

第一、二步，是在第一次访问，的时候创建 Servlet 程序会调用。

3、执行 service 方法
第三步，每次访问都会调用。

4、执行 destroy 销毁方法
第四步，在 web 工程停止的时候调用。

e)GET 和 POST 请求的分发处理

```
public class HelloServlet implements Servlet {

    /**
     * service 方法是专门用来处理请求和响应的
     * @param servletRequest
     * @param servletResponse
     * @throws ServletException
     * @throws IOException
     */
    @Override
    public void service(ServletRequest servletRequest, ServletResponse servletResponse) throws
ServletException, IOException {
        System.out.println("3 service === Hello Servlet 被访问了");
        // 类型转换（因为它有getMethod()方法）
        HttpServletRequest httpServletRequest = (HttpServletRequest) servletRequest;
        // 获取请求的方式
        String method = httpServletRequest.getMethod();

        if ("GET".equals(method)) {
            doGet();
        } else if ("POST".equals(method)) {
           doPost();
        }

    }

    /**
     * 做get 请求的操作
     */
    public void doGet(){
        System.out.println("get 请求");
        System.out.println("get 请求");
    }
    /**
     * 做post 请求的操作
     */
    public void doPost(){
        System.out.println("post 请求");
        System.out.println("post 请求");
    }

}
```

通过获取请求方法的类型为post或者get来处理不同的请求

f) 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 类的方式去实现 Servlet 程序。

1、编写一个类去继承 HttpServlet 类
2、根据业务需要重写 doGet 或 doPost 方法
3、到 web.xml 中的配置 Servlet 程序的访问地址

Servlet 类的代码：

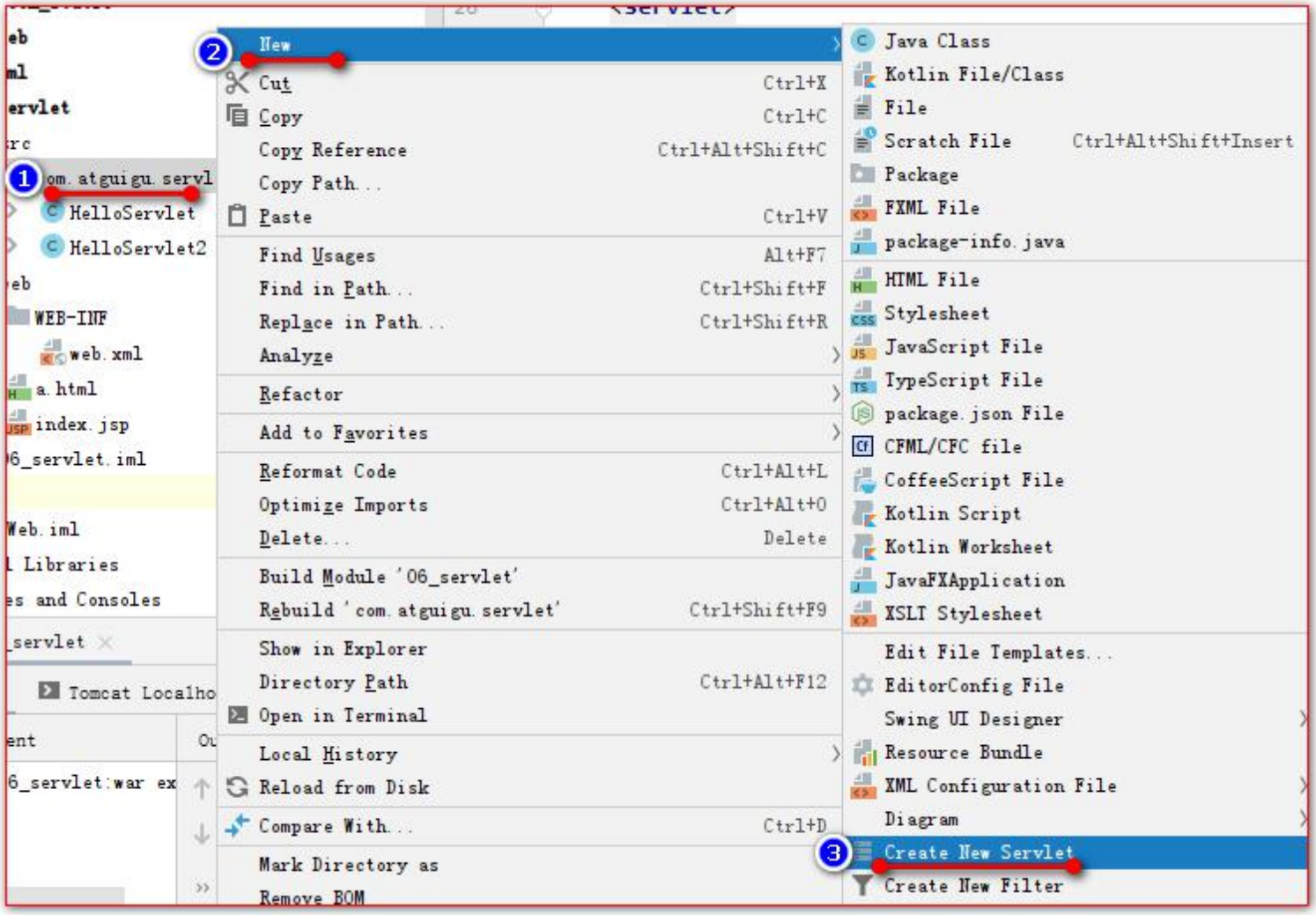
```
public class HelloServlet2 extends HttpServlet {
    /**
     * doGet（）在 get 请求的时候调用
     * @param req
     * @param resp
     * @throws ServletException
     * @throws IOException
     */
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException,
IOException {
        System.out.println("HelloServlet2 的 doGet 方法");
    }
    /**
     * doPost（）在 post 请求的时候调用
     * @param req
     * @param resp
     * @throws ServletException
     * @throws IOException
     */
    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws ServletException,
IOException {
        System.out.println("HelloServlet2 的 doPost 方法");
    }
}
```

web.xml 中的配置：

```
<servlet>
    <servlet-name>HelloServlet2</servlet-name>
    <servlet-class>com.atguigu.servlet.HelloServlet2</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>HelloServlet2</servlet-name>
    <url-pattern>/hello2</url-pattern>
</servlet-mapping>
```

g)使用 IDEA 创建 Servlet 程序

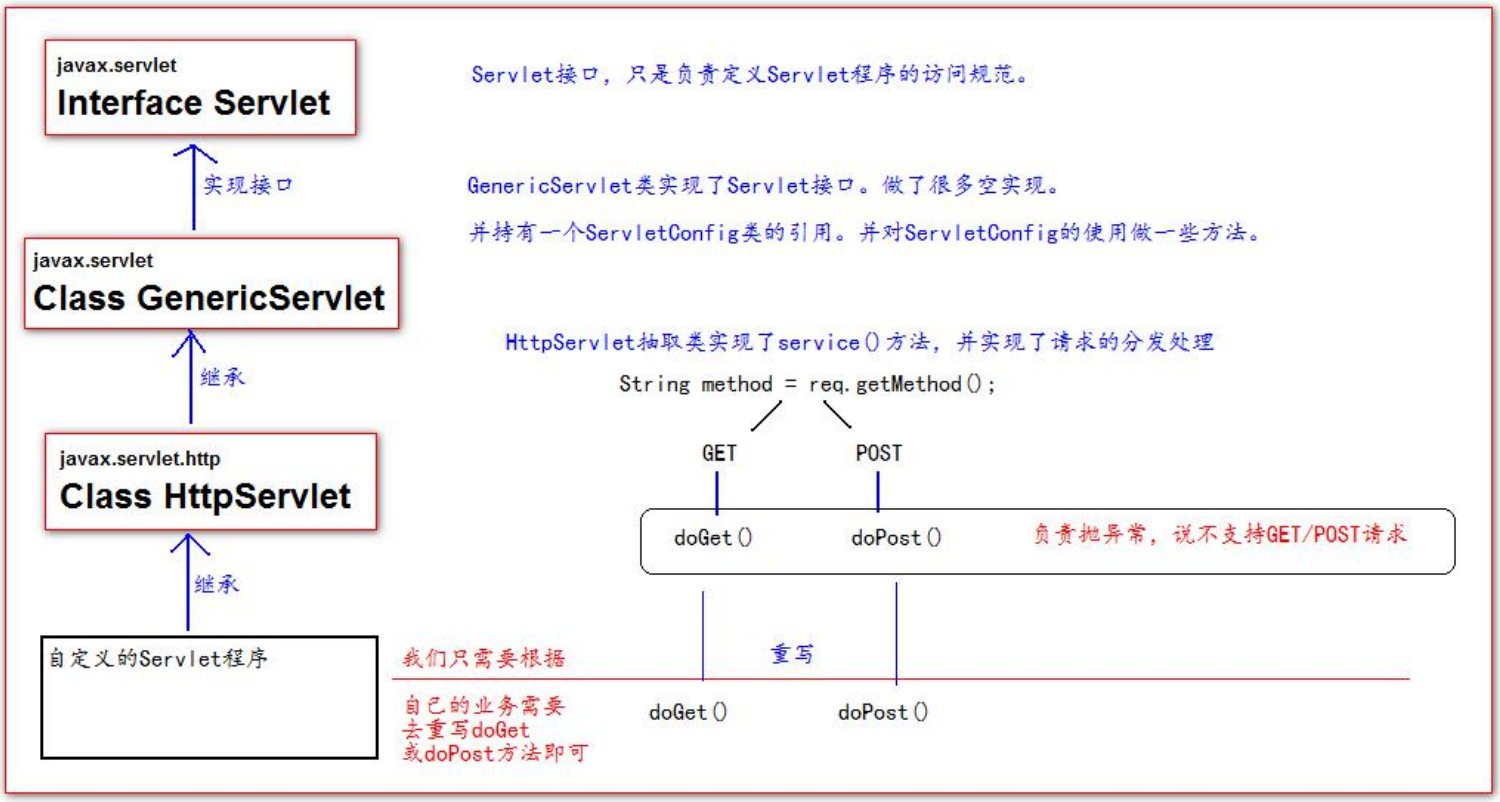
菜单：new ->Servlet 程序



配置 Servlet 的信息：

自动创建Servlet类，自动在web.xml中导入配置，自动重写doGet()，doPost()方法

h)Servlet 类的继承体系



2.ServletConfig 类

ServletConfig 类从类名上来看，就知道是 Servlet 程序的配置信息类。

Servlet 程序和 ServletConfig 对象都是由 Tomcat 负责创建，我们负责使用。

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。

一个ServletConfig对应一个Servlet程序

a)ServletConfig 类的三大作用

1、可以获取 Servlet 程序的别名 servlet-name 的值
2、获取初始化参数 init-param
3、获取 ServletContext 对象

也是在web.xml中配置init-param初始化参数，可以设置一个Servlet中共有的参数

可以直接通过ServletConfig来获取这些参数

web.xml 中的配置：

getServletConfig()可获得该Servlet的ServletConfig对象

```
<!-- servlet 标签给 Tomcat 配置 Servlet 程序 -->
<servlet>
    <!--servlet-name 标签 Servlet 程序起一个别名（一般是类名） -->
    <servlet-name>HelloServlet</servlet-name>
```

```
    <!--servlet-class 是 Servlet 程序的全类名-->
    <servlet-class>com.atguigu.servlet.HelloServlet</servlet-class>
    <!--init-param 是初始化参数-->
    <init-param>
        <!--是参数名-->
        <param-name>username</param-name>
        <!--是参数值-->
        <param-value>root</param-value>
    </init-param>
    <!--init-param 是初始化参数-->
    <init-param>
        <!--是参数名-->
        <param-name>url</param-name>
        <!--是参数值-->
        <param-value>jdbc:mysql://localhost:3306/test</param-value>
    </init-param>
</servlet>
<!--servlet-mapping 标签给 servlet 程序配置访问地址-->
<servlet-mapping>
    <!--servlet-name 标签的作用是告诉服务器，我当前配置的地址给哪个 Servlet 程序使用-->
    <servlet-name>HelloServlet</servlet-name>
    <!--
        url-pattern 标签配置访问地址                                    <br/>
          / 斜杠在服务器解析的时候，表示地址为：http://ip:port/工程路径          <br/>
          /hello 表示地址为：http://ip:port/工程路径/hello              <br/>
    -->
    <url-pattern>/hello</url-pattern>
</servlet-mapping>
```

Servlet 中的代码：

```
    @Override
    public void init(ServletConfig servletConfig) throws ServletException {
        System.out.println("2 init 初始化方法");

//        1、可以获取 Servlet 程序的别名 servlet-name 的值
        System.out.println("HelloServlet 程序的别名是:" + servletConfig.getServletName());
//        2、获取初始化参数 init-param
        System.out.println("初始化参数 username 的值是;" + servletConfig.getInitParameter("username"));
        System.out.println("初始化参数 url 的值是;" + servletConfig.getInitParameter("url"));
//        3、获取 ServletContext 对象
        System.out.println(servletConfig.getServletContext());
    }
```

注意点：

```
@Override
public void init(ServletConfig config) throws Se
    super.init(config);
    System.out.println("重写了init初始化方法,做了一些工作");
}
```

重写init方法里面一定要调用父类的init(ServletConfig)操作

因为getServletConfig()是抽象父类GenericServlet中的方法，获取的是父类中的保存的ServletConfig对象，因此需要调用父类的init方法来保存config对象，不然会有空指针异常。

3.ServletContext 类

a)什么是 ServletContext?

1、ServletContext 是一个接口，它表示 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。

什么是域对象?
域对象，是可以像 Map 一样存取数据的对象，叫域对象。
这里的域指的是存取数据的操作范围，整个 web 工程。

	存数据	取数据	删除 数据
Map	put()	get()	remove()
域对象	setAttribute()	getAttribute()	removeAttribute();

b)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: /工程路径
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

域参数在web.xml中配置，整个工程使用

在整个工程之间从传递数据

工程在服务器中的绝对路径有可能采用Catalina启动的方式，不在webapps中。且该路径对应的是工程中的/web目录，其中还保存着源码编译后的class文件

ServletContext 演示代码：

```
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
//        1、获取web.xml 中配置的上下文参数 context-param
        ServletContext context = getServletConfig().getServletContext();

        String username = context.getInitParameter("username");
        System.out.println("context-param 参数 username 的值是:" + username);
        System.out.println("context-param 参数 password 的值是:" +
context.getInitParameter("password"));
//        2、获取当前的工程路径，格式: /工程路径
```

```
        System.out.println( "当前工程路径:" + context.getContextPath() );
//        3、获取工程部署后在服务器硬盘上的绝对路径
        /**
         *  / 斜杠被服务器解析地址为:http://ip:port/工程名/  映射到 IDEA 代码的web 目录<br/>
         */
        System.out.println("工程部署的路径是:" + context.getRealPath("/"));
        System.out.println("工程下 css 目录的绝对路径是:" + context.getRealPath("/css"));
        System.out.println("工程下 imgs 目录 1.jpg 的绝对路径是:" + context.getRealPath("/imgs/1.jpg"));
    }
```

"/"表示web目录

web.xml 中的配置：

```
<!--context-param 是上下文参数(它属于整个 web 工程)-->
<context-param>
    <param-name>username</param-name>
    <param-value>context</param-value>
</context-param>
  <!--context-param 是上下文参数(它属于整个 web 工程)-->
<context-param>
    <param-name>password</param-name>
    <param-value>root</param-value>
</context-param>
```

ServletContext 像 Map 一样存取数据：

ContextServlet1 代码：

```
public class ContextServlet1 extends HttpServlet {

    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        // 获取 ServletContext 对象
        ServletContext context = getServletContext();
        System.out.println(context);
        System.out.println("保存之前: Context1 获取 key1 的值是:"+ context.getAttribute("key1"));

        context.setAttribute("key1", "value1");

        System.out.println("Context1 中获取域数据 key1 的值是:"+ context.getAttribute("key1"));
    }
}
```

ContextServlet2 代码：

```
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException,
IOException {
    ServletContext context = getServletContext();
```

```
    System.out.println(context);
    System.out.println("Context2 中获取域数据 key1 的值是:"+ context.getAttribute("key1"));
}
```

4.HTTP 协议

a)什么是 HTTP 协议

什么是协议?

协议是指双方，或多方，相互约定好，大家都需要遵守的规则，叫协议。

所谓 HTTP 协议，就是指，客户端和服务器之间通信时，发送的数据，需要遵守的规则，叫 HTTP 协议。

HTTP 协议中的数据又叫报文。

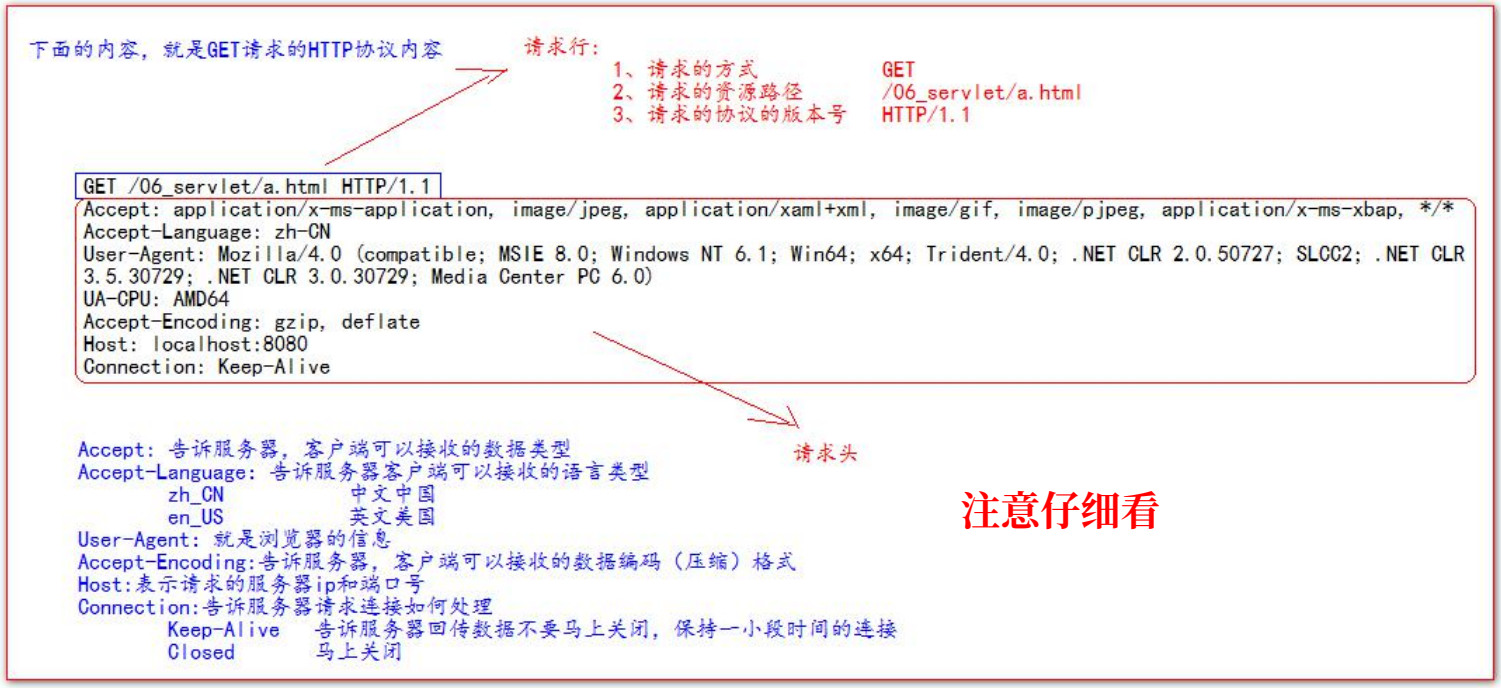
b)请求的 HTTP 协议格式

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。

请求又分为 GET 请求，和 POST 请求两种

i. GET 请求

1、请求行
   (1) 请求的方式 GET
   (2) 请求的资源路径[+?+请求参数]
   (3) 请求的协议的版本号 HTTP/1.1
2、请求头
   key : value 组成 不同的键值对，表示不同的含义。



ii. POST 请求

1、请求行

(1) 请求的方式 POST

(2) 请求的资源路径[+?+请求参数]

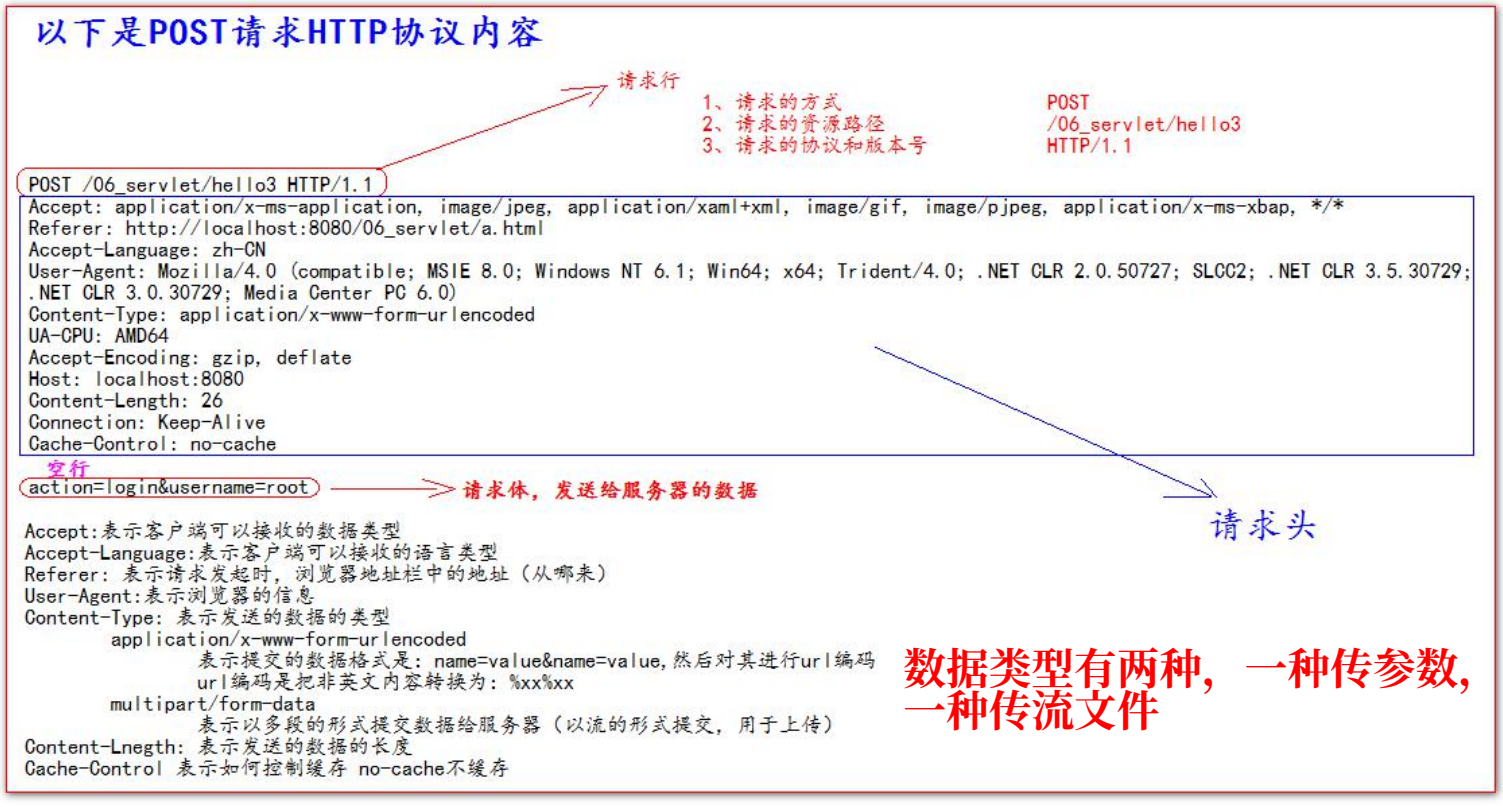
(3) 请求的协议的版本号 HTTP/1.1

2、请求头

1) key : value 不同的请求头，有不同的含义

空行

3、请求体 ===>>> 就是发送给服务器的数据



iii. 常用请求头的说明

Accept: 表示客户端可以接收的数据类型
Accpet-Languege: 表示客户端可以接收的语言类型
User-Agent: 表示客户端浏览器的信息
Host： 表示请求时的服务器 ip 和端口号

iv. 哪些是 GET 请求，哪些是 POST 请求

GET 请求有哪些：

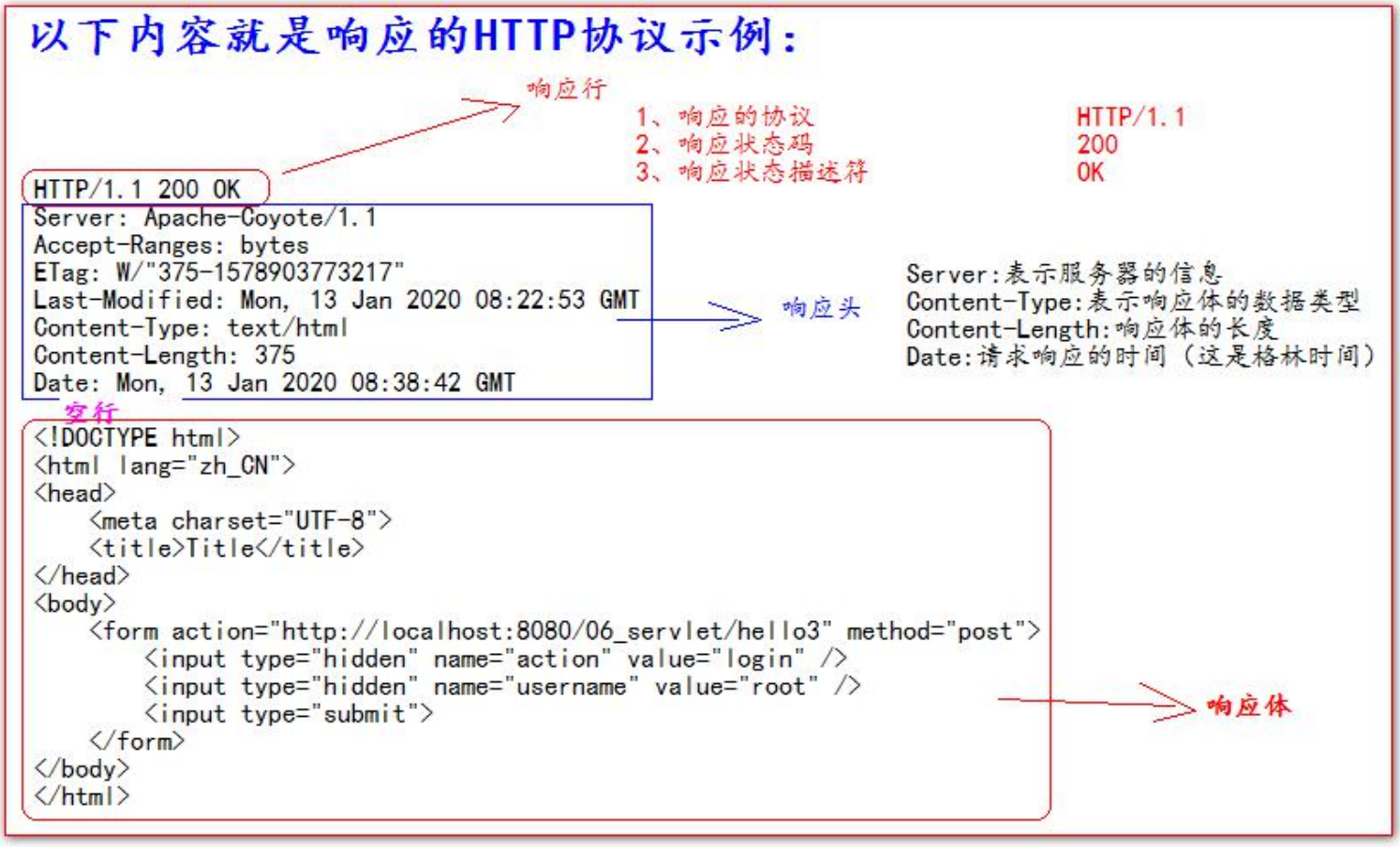
1、form 标签 method=get
2、a 标签
3、link 标签引入 css
4、Script 标签引入 js 文件
5、img 标签引入图片
6、iframe 引入 html 页面
7、在浏览器地址栏中输入地址后敲回车

POST 请求有哪些：

8、form 标签 method=post

c)响应的 HTTP 协议格式

1、响应行
(1) 响应的协议和版本号
(2) 响应状态码
(3) 响应状态描述符
2、响应头
(1) key : value 不同的响应头，有其不同含义
空行
3、响应体 --->>> 就是回传给客户端的数据



d)常用的响应码说明

200	表示请求成功
302	表示请求重定向（明天讲）
404	表示请求服务器已经收到了，但是你要的数据不存在（请求地址错误）
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）

e)MIME 类型说明

MIME 是HTTP 协议中数据类型。

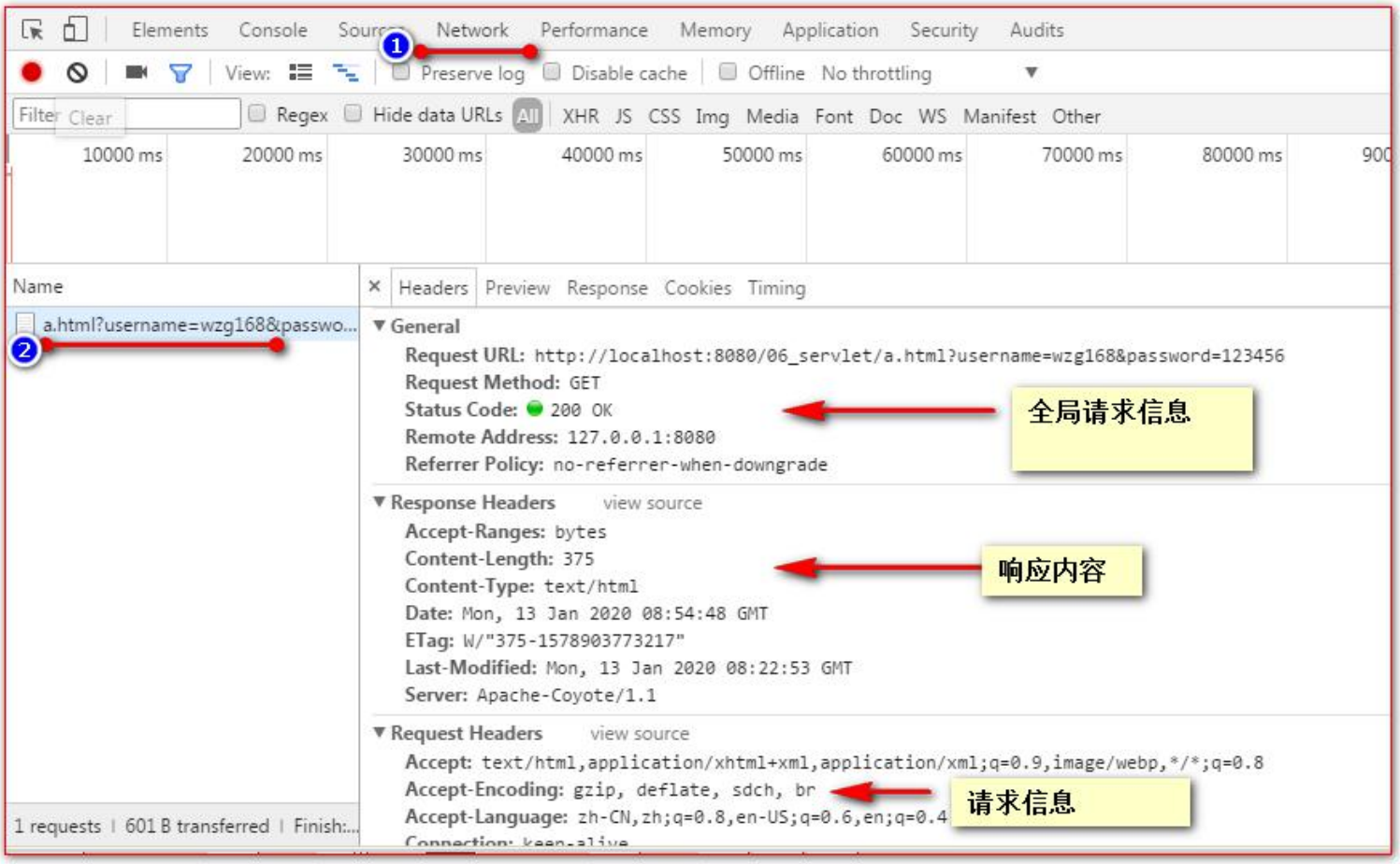
MIME 的英文全称是"Multipurpose Internet Mail Extensions" 多功能 Internet 邮件扩充服务。MIME 类型的格式是“大类型/小类型”，并与某一种文件的扩展名相对应。

常见的 MIME 类型：

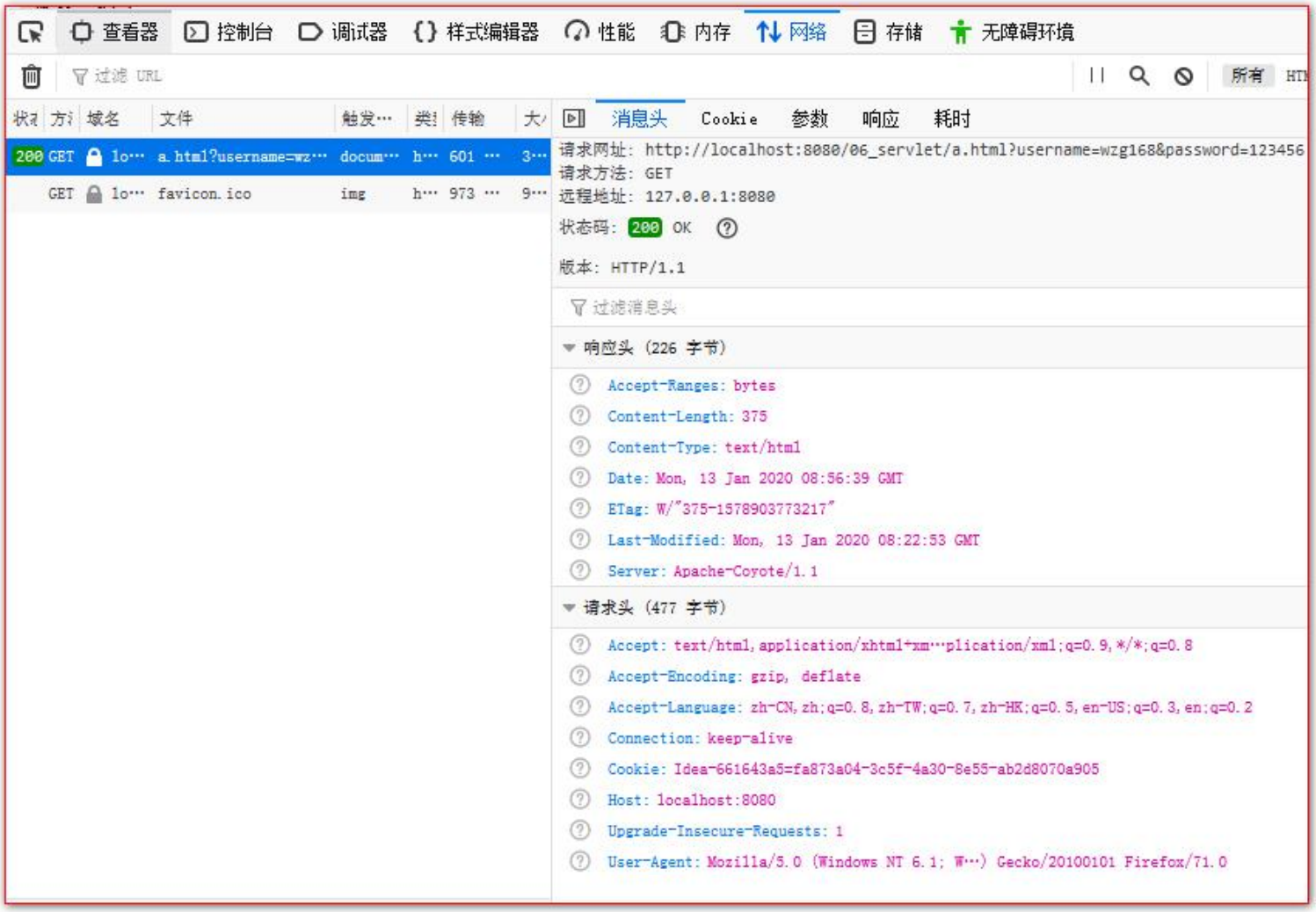
文件	MIME 类型	
超文本标记语言文本	.html , .htm	text/html
普通文本	.txt	text/plain
RTF 文本	.rtf	application/rtf
GIF 图形	.gif	image/gif
JPEG 图形	.jpeg,.jpg	image/jpeg
au 声音文件	.au	audio/basic

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

谷歌浏览器如何查看 HTTP 协议：



火狐浏览器如何查看 HTTP 协议：


查看器
控制台
调试器
样式编辑器
性能
内存
网络
存储
无障碍环境
过滤 URL
所有
状态 方法 域名 文件 触发… 类型 传输 大小
200 GET lo… a.html?username=wz… docum… h… 601 … 3…
GET lo… favicon.ico img h… 973 … 9…
消息头 Cookie 参数 响应 耗时
请求网址: http://localhost:8080/06_servlet/a.html?username=wzg168&password=123456
请求方法: GET
远程地址: 127.0.0.1:8080
状态码: 200 OK
版本: HTTP/1.1
过滤消息头
响应头 (226 字节)
Accept-Ranges: bytes
Content-Length: 375
Content-Type: text/html
Date: Mon, 13 Jan 2020 08:56:39 GMT
ETag: W/"375-1578903773217"
Last-Modified: Mon, 13 Jan 2020 08:22:53 GMT
Server: Apache-Coyote/1.1
请求头 (477 字节)
Accept: text/html,application/xhtml+xm…plication/xml;q=0.9,*/*;q=0.8
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2
Connection: keep-alive
Cookie: Idea-661643a5=fa873a04-3c5f-4a30-8e55-ab2d8070a905
Host: localhost:8080
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 6.1; W…) Gecko/20100101 Firefox/71.0