第 1 章 XML

1.1 XML 概述

- 什么是 XML？
  1) XML 指可扩展标记语言（EXtensible Markup Language）。
  2) XML 的设计宗旨是传输数据，而不是显示数据。
  3) XML 标签没有被预定义，标签由使用者自定义。
  4) XML 文件后缀为 .xml
- XML 的作用
  1) 用来保存数据，而且这些数据具有自我描述性
  2) 可以作为项目的配置文件
  3) 可以作为网络传输数据的格式

1.2 XML 语法

1.2.1 文档声明

表 1-1 XML 示例 books.xml 文件

```
<?xml version="1.0" encoding="UTF-8" ?>
<!--
    <?xml version="1.0" encoding="UTF-8" ?>
    以上内容就是 xml 文件的声明
    version: 表示 xml 的版本
    encoding：表示 xml 文件的编码
-->
<books><!-- 使用 books 标签来包裹多个图书信息 -->
    <book sn="SN1234132"> <!-- 一个 book 标签表示一本图书 sn 属性表示图书序列号 -->
        <name>活着</name> <!-- name 标签表示书名 -->
        <author>余华</author> <!-- author 标签表示作者 -->
        <price>19.9</price> <!-- price 标签表示价格 -->
    </book>
</books>
```

1.2.2 XML 注释

XML 和 HTML 的注释语法一样：<!—注释内容… -->

1.2.3 语法规则

1) 所有 XML 元素都须有关闭标签（也就是闭合）
2) XML 标签对大小写敏感
3) XML 必须正确地嵌套
4) XML 文档必须有根元素

5) XML 的属性值须加引号

6) 文本区域（CDATA 区）：CDATA 语法可以告诉 xml 解析器，我 CDATA 里的文本内容，只是纯文本，不需要 xml 语法解析，CDATA 格式：

<![CDATA[这里可以把你输入的字符原样显示，不会解析 xml]]>

表 1-2 示例

```
<?xml version="1.0" encoding="UTF-8" ?>
<!--
    <?xml version="1.0" encoding="UTF-8" ?>
    以上内容就是 xml 文件的声明
    version：表示 xml 的版本
    encoding：表示 xml 文件的编码
-->
<books><!-- 使用 books 标签来包裹多个图书信息 -->
    <book sn="SN1234132"> <!-- 一个 book 标签表示一本图书 sn 属性表示图书序列号 -->
        <name>活着</name> <!-- name 标签表示书名 -->
        <author>余华</author> <!-- author 标签表示作者 -->
        <price>19.9</price> <!-- price 标签表示价格 -->
    </book>
    <book sn="SN4313213" name="百年孤独" author="未知" price="29.9"></book>
    <book>
        <name>傲慢与偏见</name>
        <author>未知</author>
        <price>29.9</price>
        <desc>
            <![CDATA[
            呵呵哈哈哈
            ]]>
        </desc>
    </book>
</books>
```

1.3 XML 解析技术介绍

早期 JDK 为我们提供了两种 xml 解析技术 DOM 和 Sax 简介（已经过时，但我们需要知道这两种技术）

dom 解析技术是 W3C 组织制定的，而所有的编程语言都对这个解析技术使用了自己语言的特点进行实现。 Java 对 dom 技术解析标记也做了实现。

sun 公司在 JDK5 版本对 dom 解析技术进行升级：SAX（ Simple API for XML ） SAX 解析，它跟 W3C 制定的解析不太一样。它是以类似事件机制通过回调告诉用户当前正在解析的内容。 它是一行一行的读取 xml 文件进行解析的。不会创建大量的 dom 对象。 所以它在解析 xml 的时候，在内存的使用上。和性能上。都优于 Dom 解析。

第三方的解析： jdom 在 dom 基础上进行了封装，dom4j 又对 jdom 进行了封装。 pull 主要用在 Android 手机开发，是在跟 sax 非常类似都是事件机制解析 xml 文件。

Dom4j 它是第三方的解析技术。我们需要使用第三方给我们提供好的类库才可以解

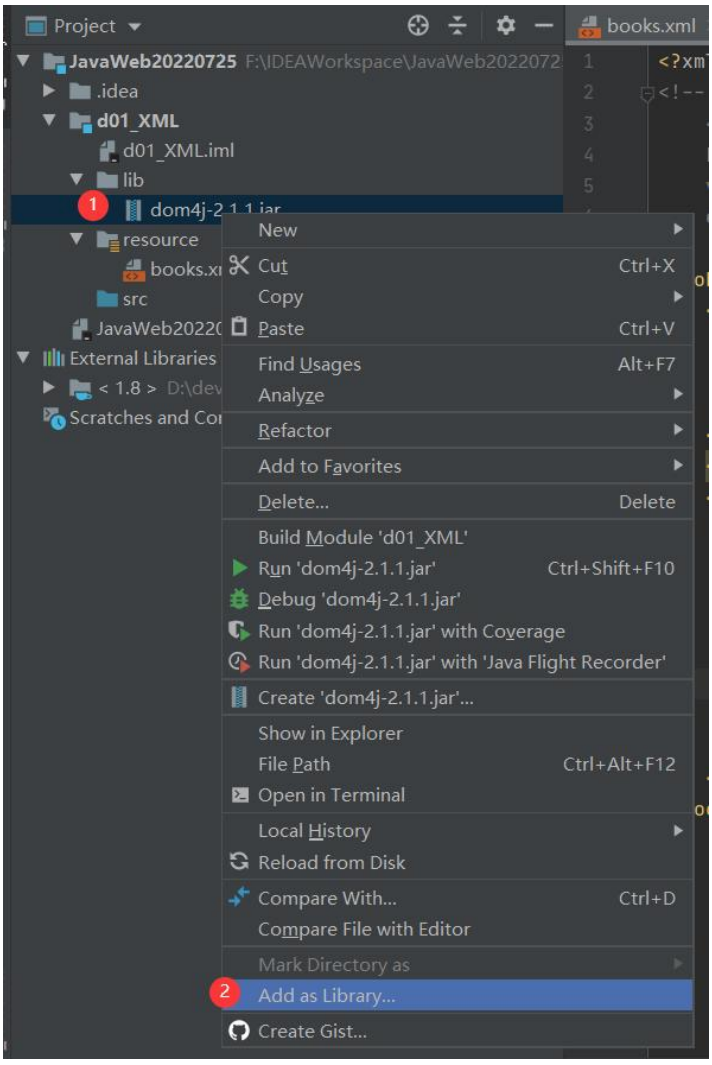
析 xml 文件。

1.4 dom4j 解析 XML

由于 dom4j 它不是 sun 公司的技术，而属于第三方公司的技术，我们需要使用 dom4j 就需要到 dom4j 官网下载 dom4j 的 jar 包。

- 第一步：下载 dom4j 的 jar 包，下载地址：https://mvnrepository.com/artifact/org.dom4j/dom4j
- 第二步：在项目中创建 lib 目录，然后将 dom4j.jar 赋值到该目录下，右击该 jar 包，选择 Add as library ，成功导入

图 1-1 导入 dom4j 包



- 第三步：创建一个 xml 文件以供解析

表 1-3 books.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!--
    <?xml version="1.0" encoding="UTF-8" ?>
    以上内容就是 xml 文件的声明
    version: 表示 xml 的版本
```

```
    encoding：表示 xml 文件的编码
-->
<books><!-- 使用 books 标签来包裹多个图书信息 -->
    <book sn="SN1234132"> <!-- 一个 book 标签表示一本图书 sn 属性表示图书序列号 -->
        <name>活着</name> <!-- name 标签表示书名 -->
        <author>余华</author> <!-- author 标签表示作者 -->
        <price>19.9</price> <!-- price 标签表示价格 -->
    </book>
<!--    <book sn="SN4313213" name="百年孤独" author="未知" price="29.9"></book>-->
    <book sn="SN424324">
        <name>傲慢与偏见</name>
        <author>未知</author>
        <price>29.9</price>
    </book>
</books>
```

- 第四步：解析，代码如下

表 1-4 dom4j 解析 xml 示例代码

```
import org.dom4j.Document;
import org.dom4j.DocumentException;
import org.dom4j.Element;
import org.dom4j.io.SAXReader;
import org.junit.Test;

import java.math.BigDecimal;
import java.util.ArrayList;
import java.util.List;

public class Dom4jDemo {

    @Test
    public void test() throws DocumentException {
        // 第一步：创建 SAXReader 对象来读取 XML 文件，并获得 Document 对象
        SAXReader reader = new SAXReader();
        Document document = reader.read("resource/books.xml");
        // 第二步：获取根元素对象
        Element rootElement = document.getRootElement();
//          System.out.println(rootElement);
        // asXML() 方法会将当前元素转换成 String 对象
//          System.out.println(rootElement.asXML());
        // 第三步：根据根元素对象获取所有的 book 对象
        List<Element> bookElements = rootElement.elements("book");
        // 第四步：遍历所有 book 标签，将 book 标签中的属性及标签封装到一个新的集合中
        List<Book> bookList = new ArrayList<>();
        for ( Element e: bookElements ) {
//              System.out.println("e.asXML() = " + e.asXML());
            // 拿到 book 中的 sn 属性
            String sn = e.attributeValue("sn");
//              System.out.println("sn = " + sn);
            // 拿到 book 下面的 name 标签的值
            String name = e.elementText("name");
//              System.out.println("name = " + name);
            String author = e.elementText("author");
            Element priceElement = e.element("price");
            String price = priceElement.getText();
            bookList.add(new Book(sn, author, name,
BigDecimal.valueOf(Double.parseDouble(price))));
```

```
            }

            for (Book book: bookList) {
                System.out.println(book);
            }
        }
}
```

第 2 章 JavaWeb

2.1 什么是 JavaWeb

JavaWeb 是指，所有通过 Java 语言编写可以通过浏览器访问的程序的总称，叫 JavaWeb。

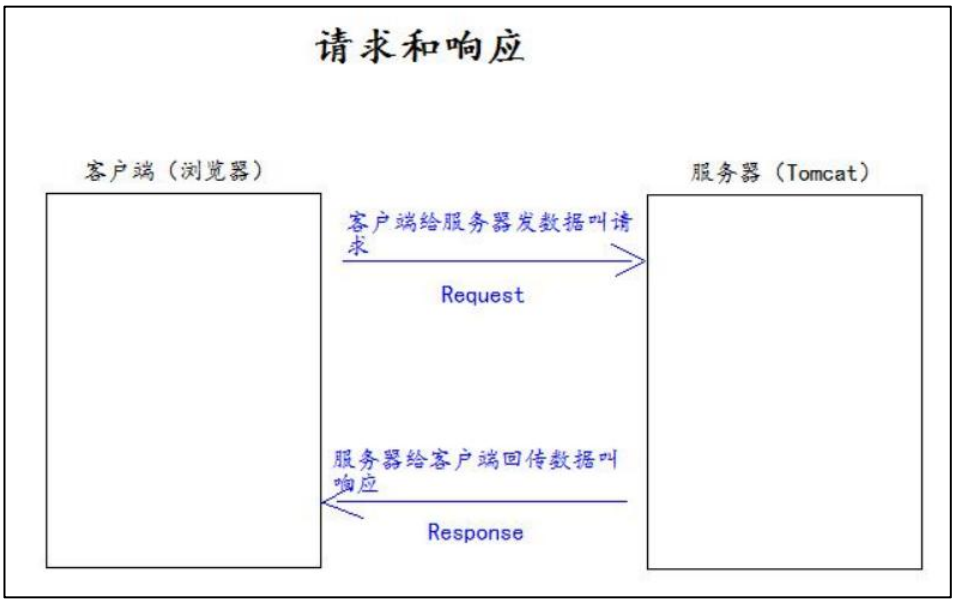
JavaWeb 是基于请求和响应来开发的。

请求(Request)：请求是指客户端给服务器发送数据，叫请求。

响应(Response)：响应是指服务器给客户端回传数据，叫响应。

请求和响应的关系：请求和响应是成对出现的，有请求就有响应。

图 2-1 请求和响应



2.2 Web 资源分类

web 资源按实现的技术和呈现的效果的不同，又分为静态资源和动态资源两种。

静态资源： html、css、js、txt、mp4 视频、jpg 图片等

动态资源： jsp 页面、Servlet 程序等

2.3 Web 服务器

1) Tomcat：由 Apache 组织提供的一种 Web 服务器，提供对 jsp 和 Servlet 的支持。它是一种轻量级的 javaWeb 容器（服务 器），也是当前应用最广的 JavaWeb 服务器（免费）。
2) Jboss：是一个遵从 JavaEE 规范的、开放源代码的、纯 Java 的 EJB 服务器，它支持所有的 JavaEE 规范（免费）。
3) GlassFish： 由 Oracle 公司开发的一款 JavaWeb 服务器，是一款强健的商业服务器，达到产品级质量（应用很少）。

4) Resin：是 CAUCHO 公司的产品，是一个非常流行的服务器，对 servlet 和 JSP 提供了良好的支持， 性能也比较优良，resin 自身采用 JAVA 语言开发（收费，应用比较多）。

5) WebLogic：是 Oracle 公司的产品，是目前应用最广泛的 Web 服务器，支持 Java EE 规范， 而且不断的完善以适应新的开发要求，适合大型项目（收费，用的不多，适合大公司）。

2.4 Tomcat 服务器和 Servlet 版本对应关系

当前企业常用的版本 7.*、8.*

Tomcat版本	Servlet/JSP版本	JavaEE版本	运行环境
4.1	2.3/1.2	1.3	JDK1.3
5.0	2.4/2.0	1.4	JDK1.4
5.5/6.0	2.5/2.1	5.0	JDK5.0
7.0	3.0/2.2	6.0	JDK6.0
8.0	3.1/2.3	7.0	JDK7.0

Servlet 程序从 2.5 版本是现在市面使用最多的版本（xml 配置）

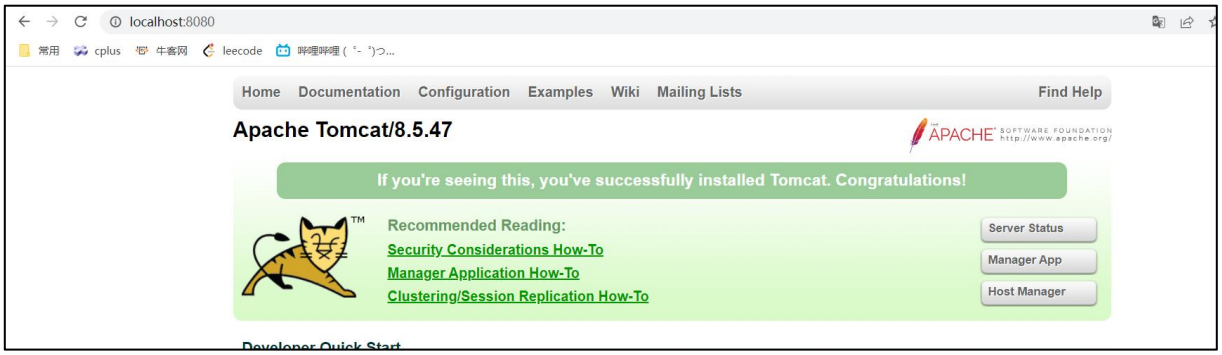
到了 Servlet3.0 之后就是注解版本的 Servlet 使用。

第 3 章 Tomcat 服务器

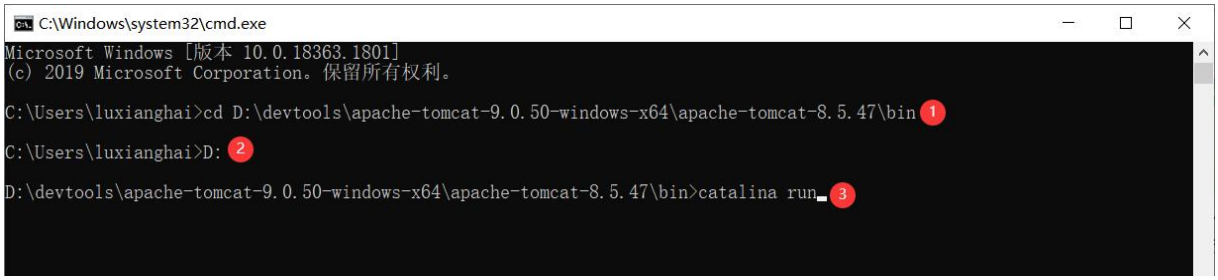
3.1 安装和启动

- 安装：找到你需要用的 Tomcat 版本对应的 zip 压缩包，解压到需要安装的目录即可。
- 启动：找到 Tomcat 目录下的 bin 目录下的 startup.bat 文件，双击，就可以启动 Tomcat 服务器。
- 如何测试 Tomcat 是否启动成功？- 打开浏览器，在浏览器地址栏中输入以下地址测试：
  1) http://localhost:8080
  2) http://127.0.0.1:8080
  3) http://本机 IP:8080

当出现如下界面，说明 Tomcat 服务器启动成功!!!



- 常见启动 Tomcat 失败的情况：双击 startup.bat 文件，就会出现一个小黑窗口一闪而来。这个时候，失败的原因基本上都是因为没有配置好 JAVA_HOME 环境变量。
- **启动 Tomcat 的第二种方式：打开命令行 → cd 到 你的 Tomcat 的 bin 目录下 → 敲入启动命令： catalina run**



- 停止 Tomcat：三种方式
  1) 点击 tomcat 服务器窗口的 x 关闭按钮
  2) 把 Tomcat 服务器窗口置为当前窗口，然后按快捷键 Ctrl+C
  3) 找到 Tomcat 的 bin 目录下的 shutdown.bat 双击，就可以停止 Tomcat 服务器

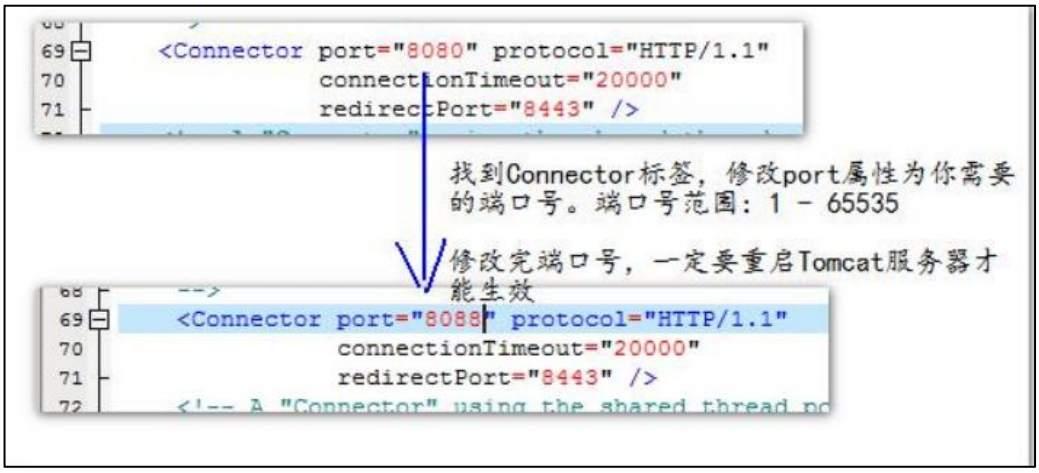
3.2 目录介绍

- bin：专门用来存放 Tomcat 服务器的可执行程序
- conf：专门用来存放 Tocmat 服务器的配置文件
- lib：专门用来存放 Tomcat 服务器的 jar 包
- logs：专门用来存放 Tomcat 服务器运行时输出的日记信息
- temp：专门用来存放 Tomcdat 运行时产生的临时数据
- webapps：专门用来存放部署的 Web 工程。
- work：是 Tomcat 工作时的目录，用来存放 Tomcat 运行时 jsp 翻译为 Servlet 的源码，和 Session 钝化的目录。

3.3 修改端口号

Tomcat 默认的端口号是：8080

找到 Tomcat 目录下的 conf 目录，找到 server.xml 配置文件。

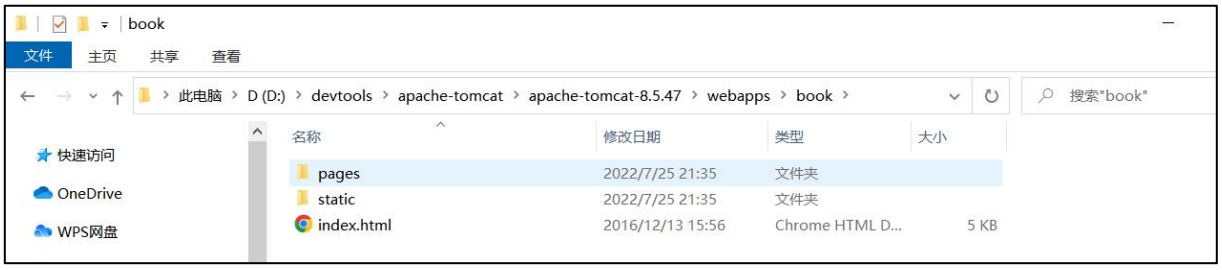


3.4 部署 web 工程到 Tomcat 中

3.4.1 方法一

第一种部署方法：只需要把 web 工程的目录拷贝到 Tomcat 的 webapps 目录下即可。

1. 在 webapps 目录下创建一个 book 工程，如何将静态页面放到 book 工程中



2. 重新启动 Tomcat

3. 访问

- http://ip:port/目录名/资源名
- 如：http://localhost:8080/book/index.html

3.4.2 方法二

找到 Tomcat 下的 conf 目录\Catalina\localhost\ 下,创建如下的配置文件：

demo.xml 配置文件内容如下：

表 3-1 demo.xml 文件内容

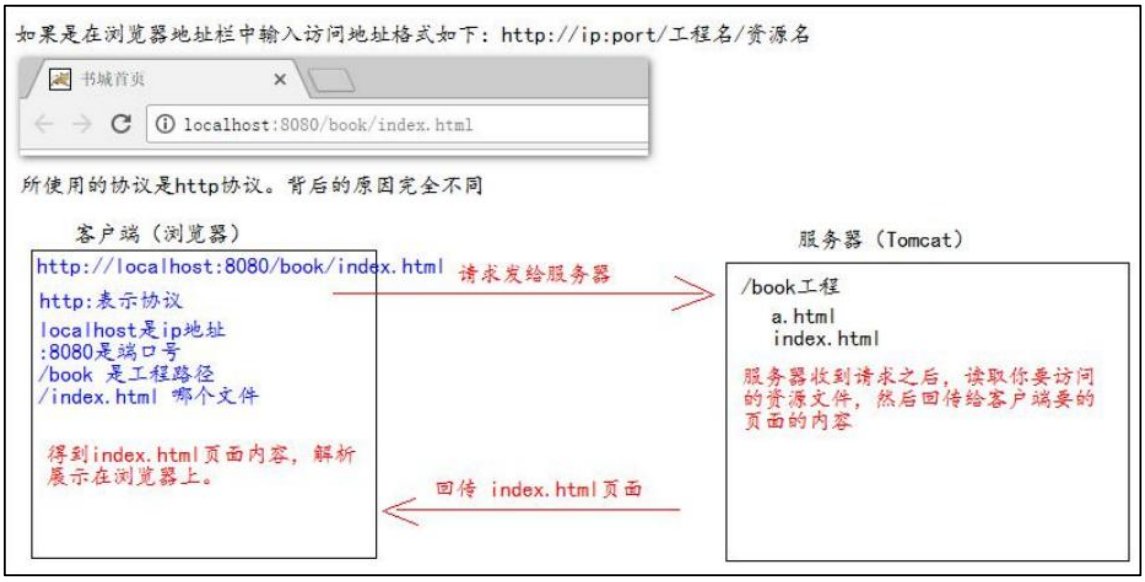
```
<!-- Context 表示一个工程上下文
        path 表示工程的访问路径
        docBase 表示工程文件夹所在目录
-->
<Context path="/demo" docBase="D:\book"/>
```

访问这个工程的路径如下：http://ip:port/demo/ 就表示访问 D:\book 目录

3.5 手拖 html 页面到浏览器和在浏览器中输入 http://ip:端 口号/工程名/访问的区别

手拖 html 页面的原理：

输入访问地址访问的原因：



3.6 ROOT 的工程的访问，以及 默认 index.html 页面的访问

当我们在浏览器地址栏中输入访问地址如下：

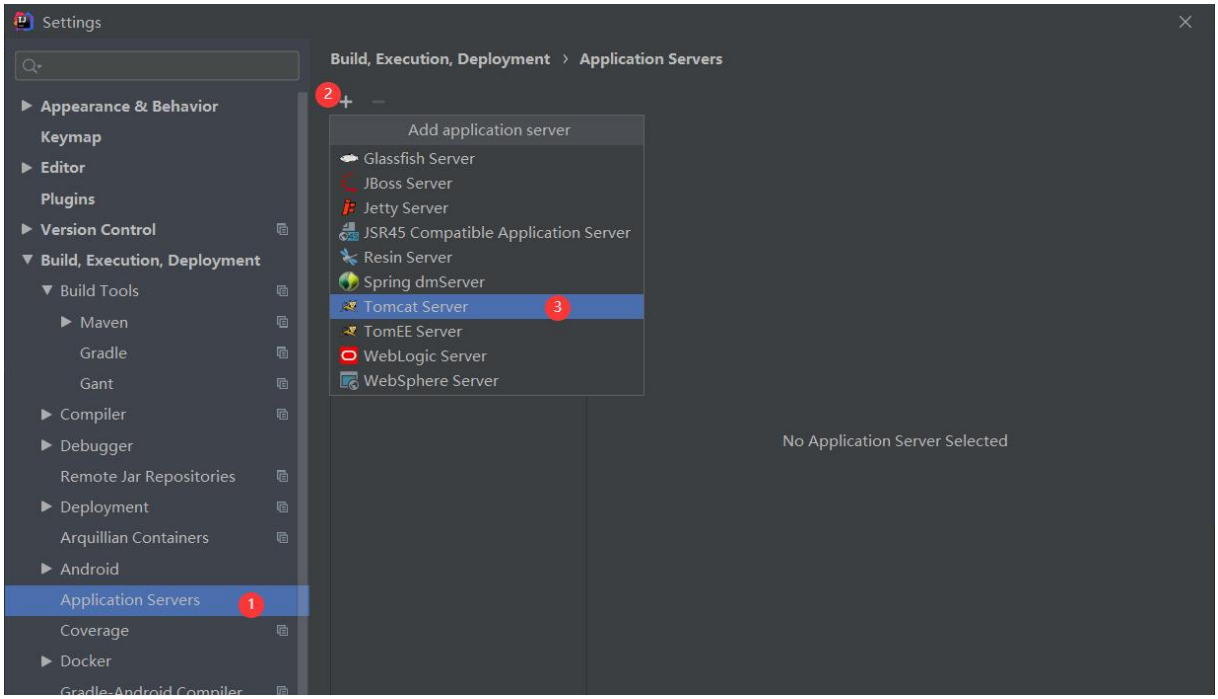
http://ip:port/ → 没有工程名的时候，默认访问的是 ROOT 工程。

当我们在浏览器地址栏中输入的访问地址如下：

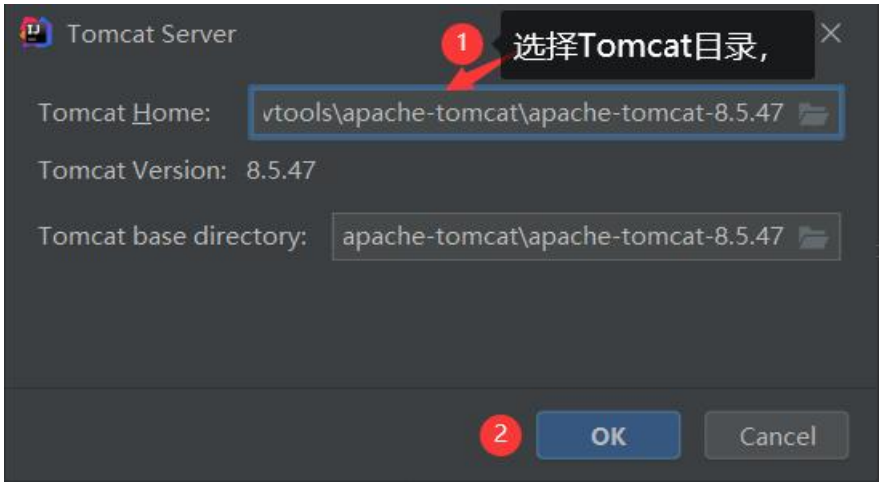
http://ip:port/工程名/ → 没有资源名，默认访问 index.html 页面

3.7 IDEA 整合 Tomcat 服务器

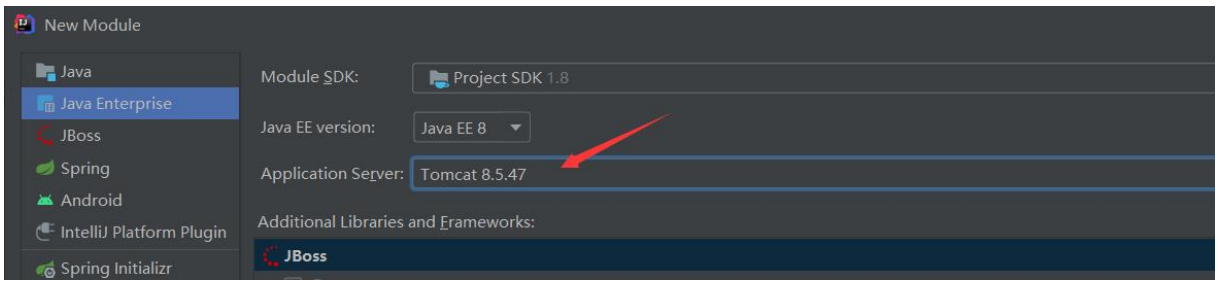
操作的菜单如下：File | Settings | Build, Execution, Deployment | Application Servers



配置 Tomcat 的安装目录



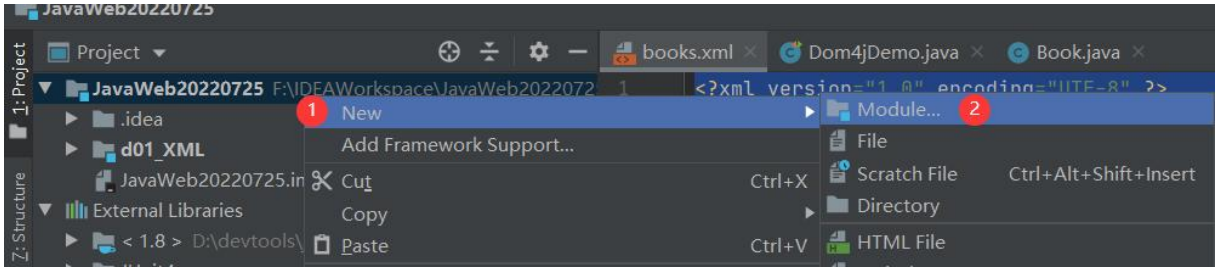
此时可以通过创建一个 Model 查看是不是配置成功!!!



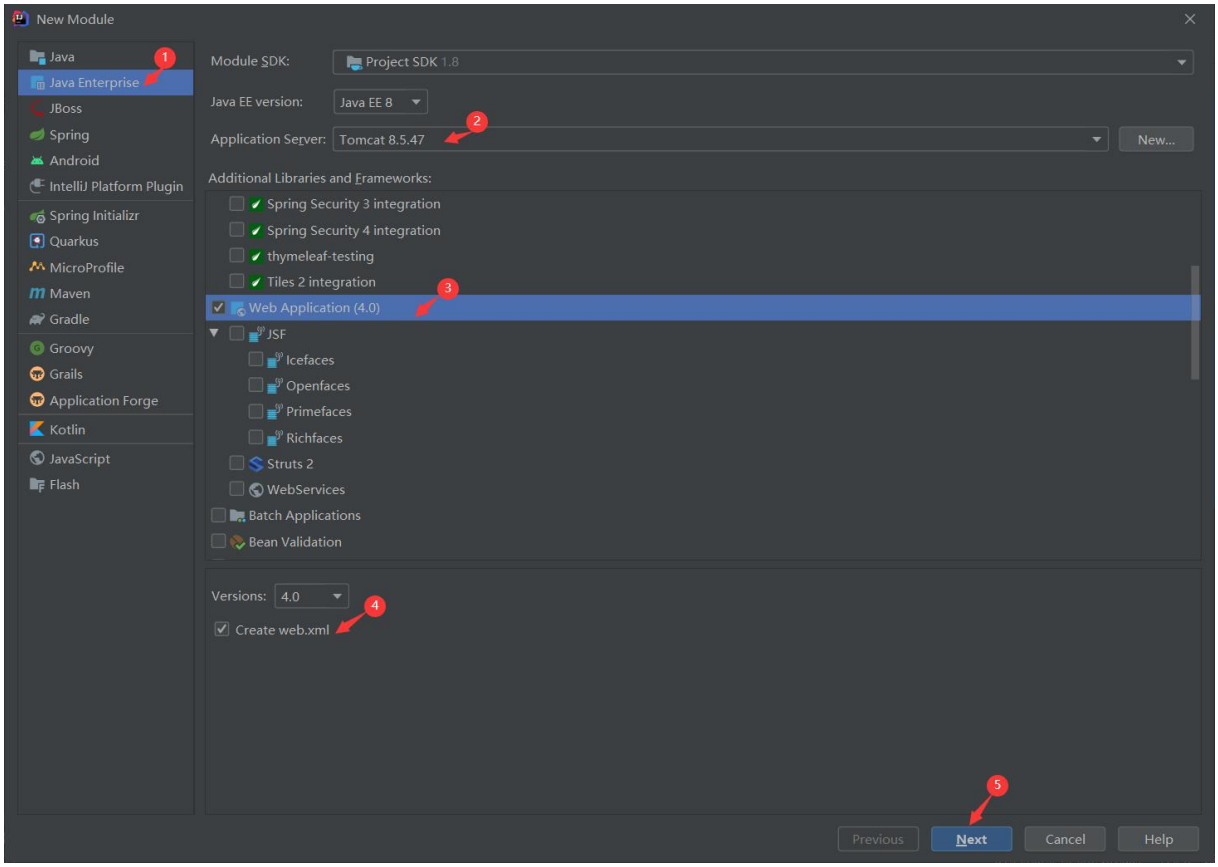
3.8 IDEA 中动态 web 工程的操作

3.8.1 创建动态 web 工程

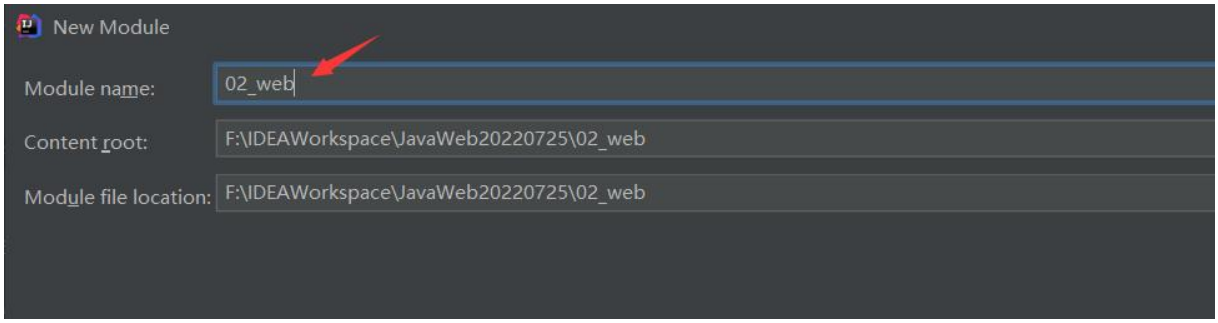
1、创建一个新模块：



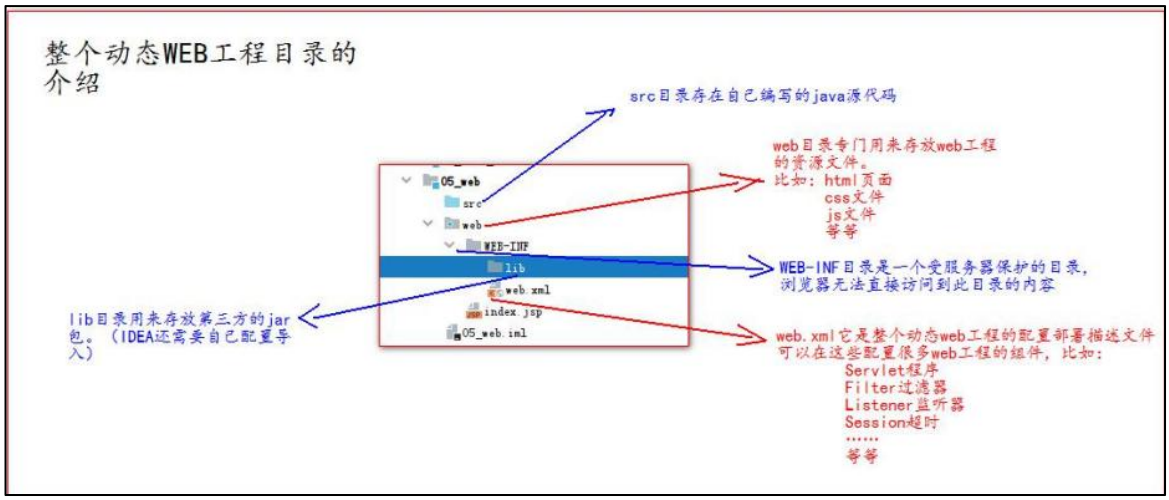
2、选择你要创建什么类型的模块



3、输入你的模块名，点击【Finish】完成创建。



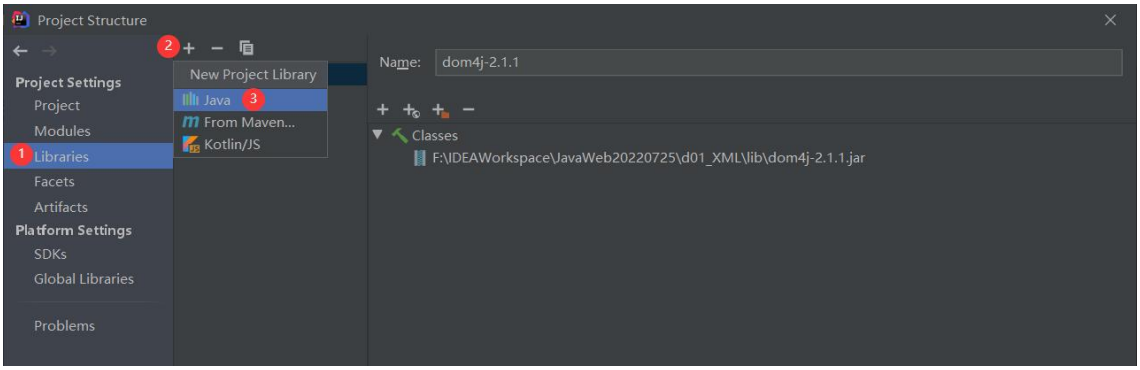
3.8.2 Web 工程的目录介绍



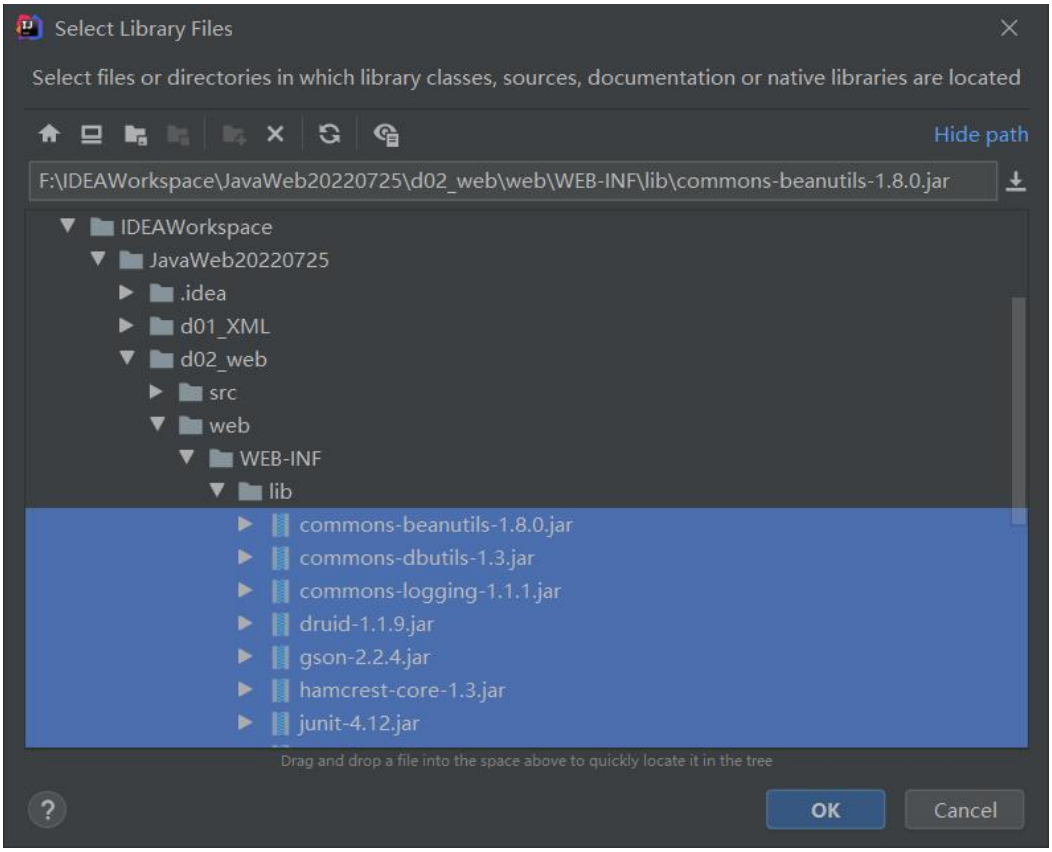
3.8.3 如何给动态 web 工程添加额外 jar 包

1、将 jar 包复制到 WEB-INF/lib 下

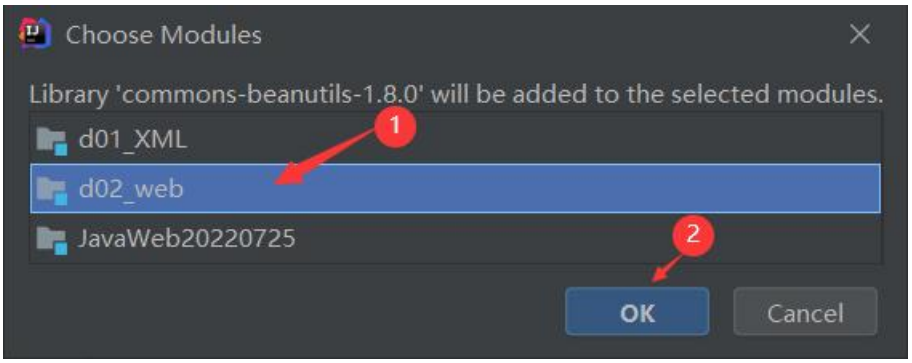
2、File | Project Structure



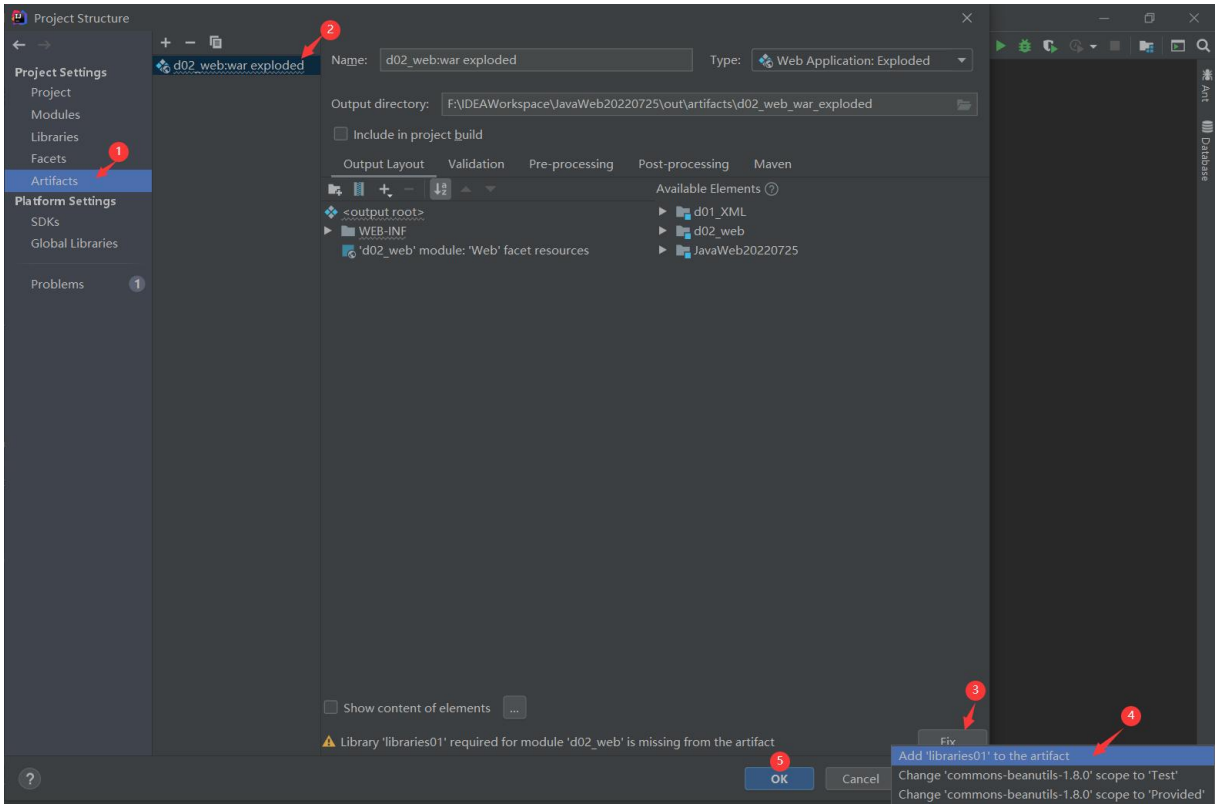
3、选择需要添加的 jar 包，点击【OK】



4、选择你添加的类库给哪个模块使用



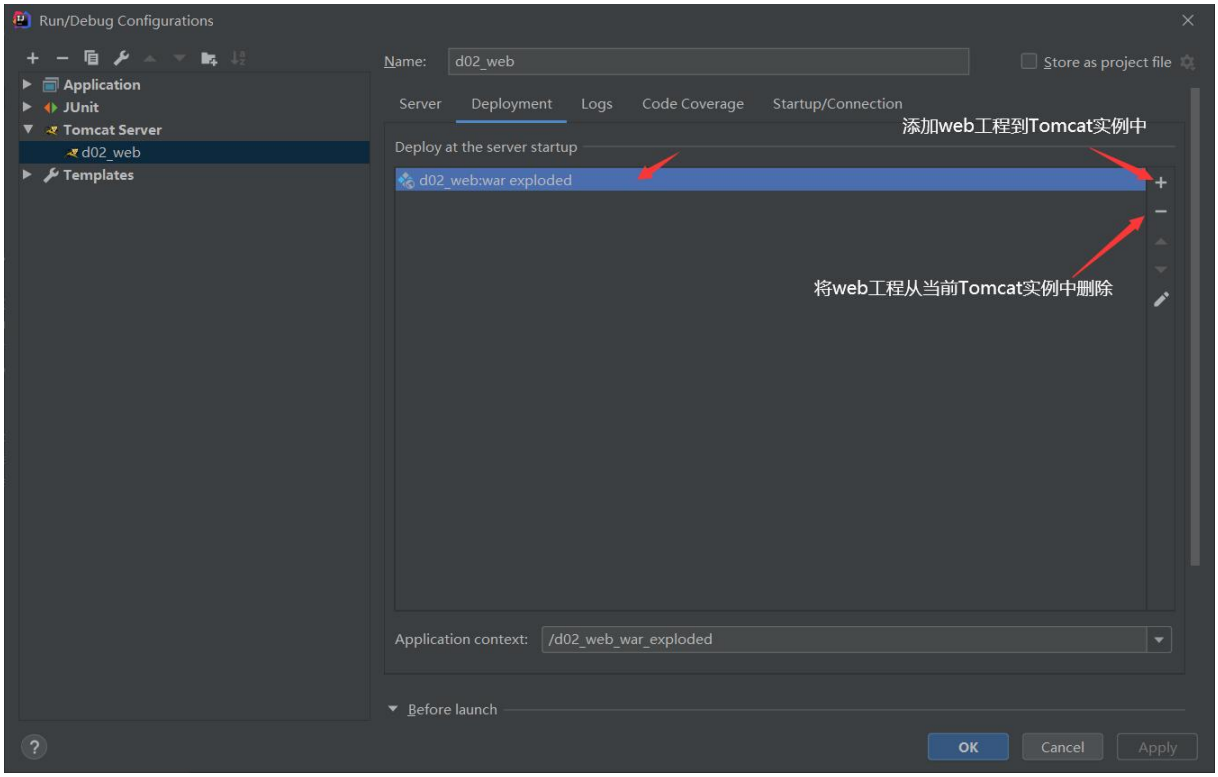
5、选择 Artifacts 选项，将类库添加到打包部署中



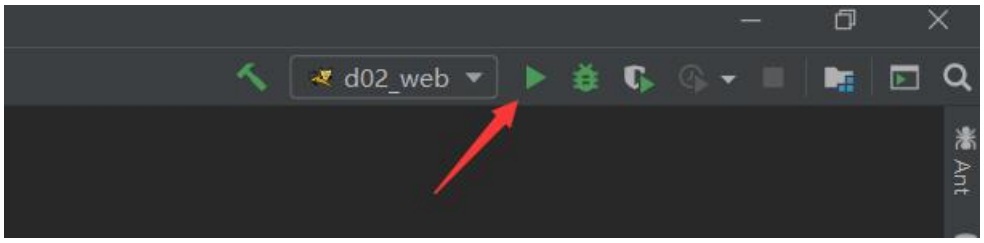
3.8.4 将 web 工程部署到 Tomcat 上运行

1、建议修改 web 工程对应的 Tomcat 运行实例名称：

2、确认你的 Tomcat 实例中有你要部署运行的 web 工程模块：


Run/Debug Configurations
Name:
d02_web
Store as project file
Application
JUnit
Tomcat Server
d02_web
Templates
Server
Deployment
Logs
Code Coverage
Startup/Connection
添加web工程到Tomcat实例中
Deploy at the server startup
d02_web:war exploded
将web工程从当前Tomcat实例中删除
Application context:
/d02_web_war_exploded
Before launch
OK
Cancel
Apply

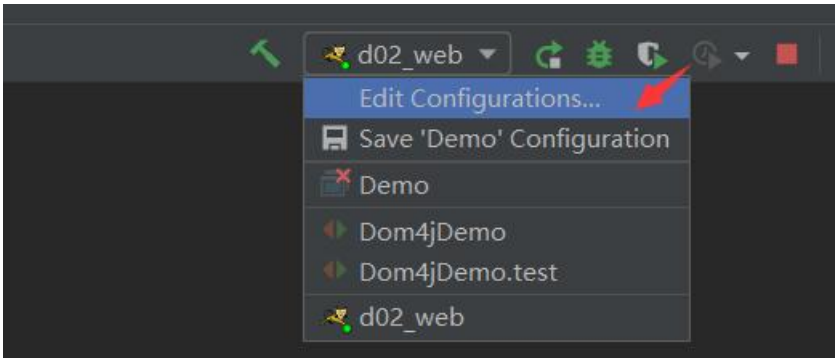
3、运行

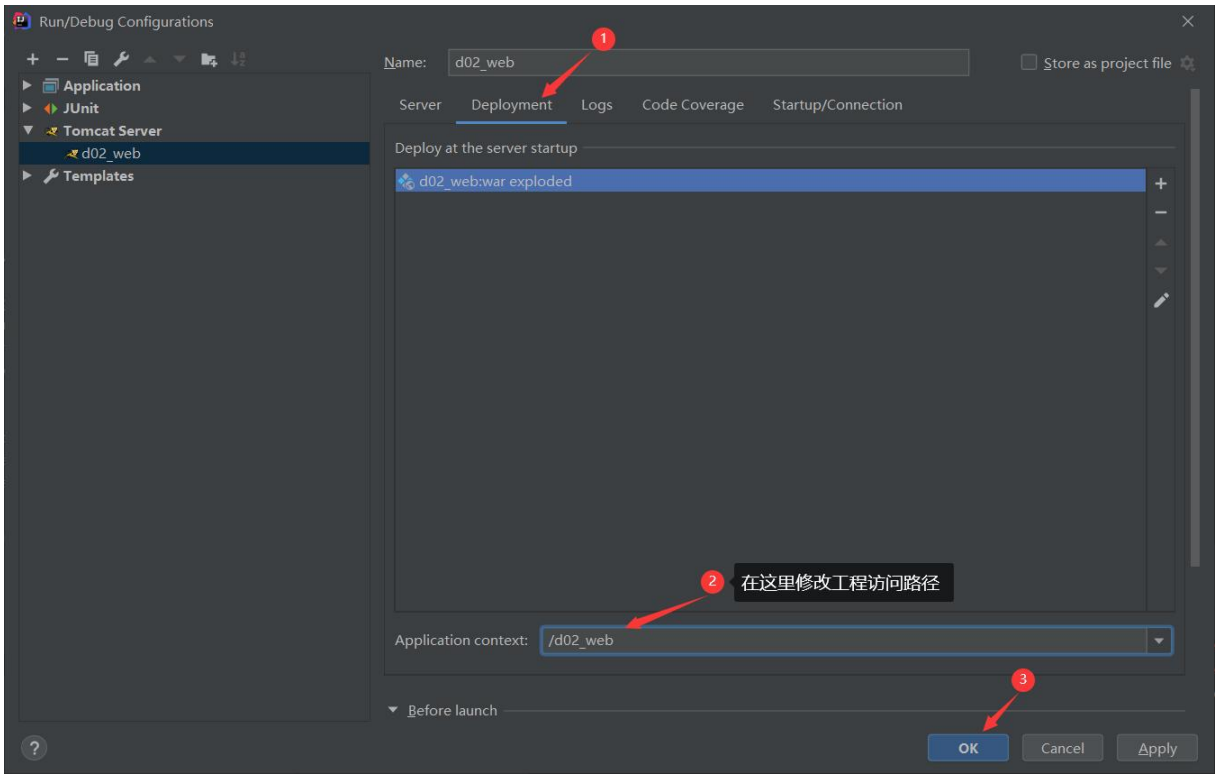

d02_web
Ant

4、运行成功


localhost:8080/d02_web_war_exploded/
常用
cplus
牛客网
leecode
哔哩哔哩 (°- °)つ...
END

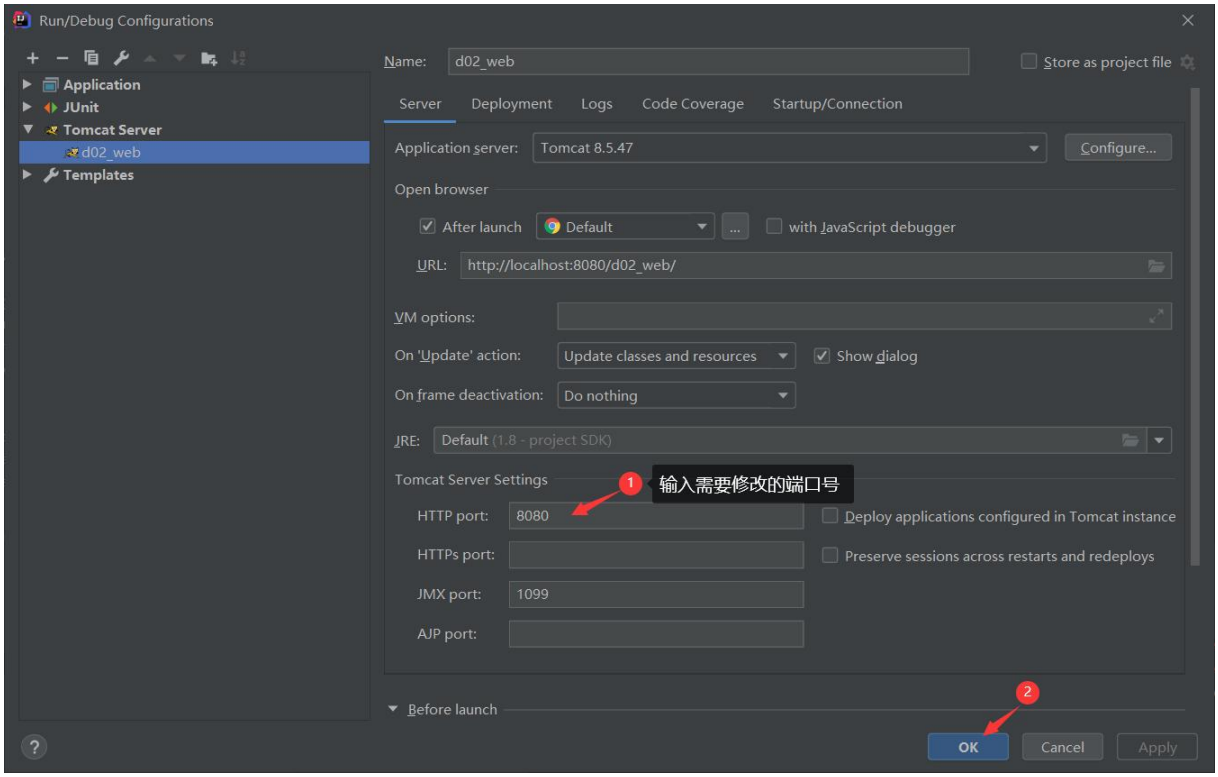
3.8.5 修改工程访问路径



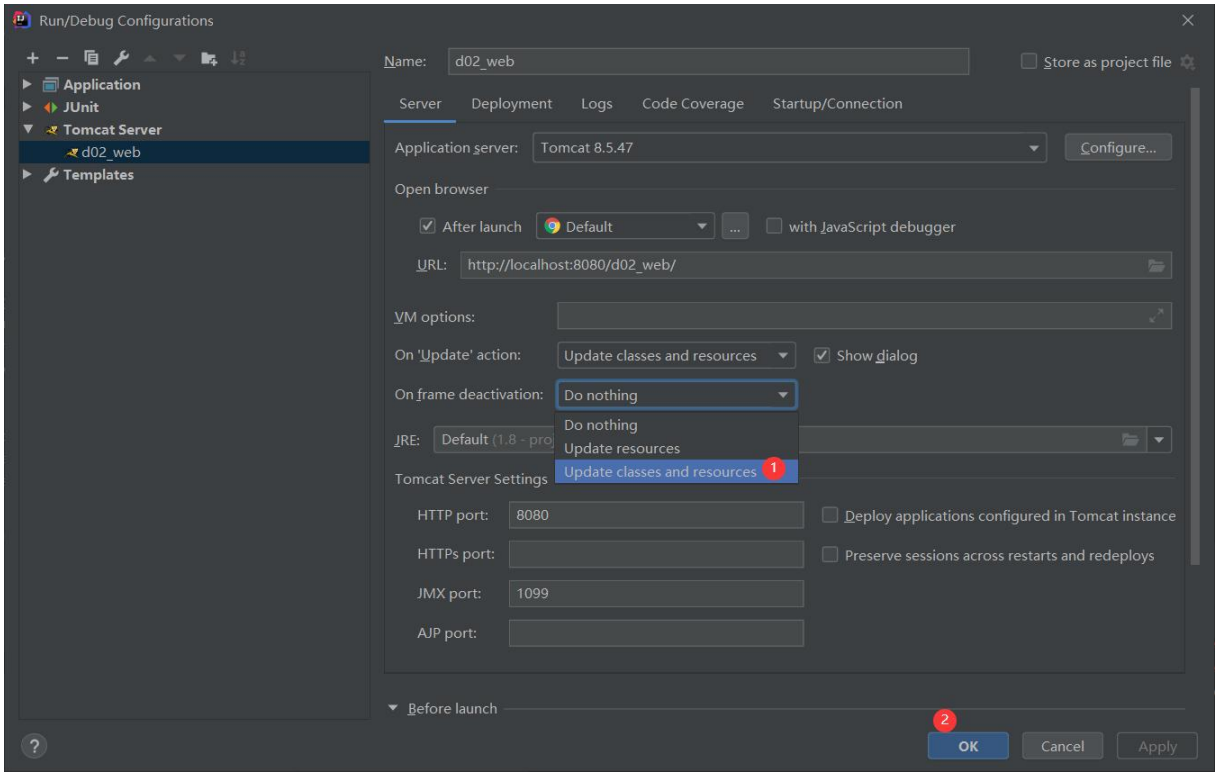


然后重启服务器即可

3.8.6 修改运行端口号



3.8.7 配置热部署



3.9 Tomcat 日志中文乱码

进入 Tomcat 安装目录 logs 下，修改 logging.properties 文件

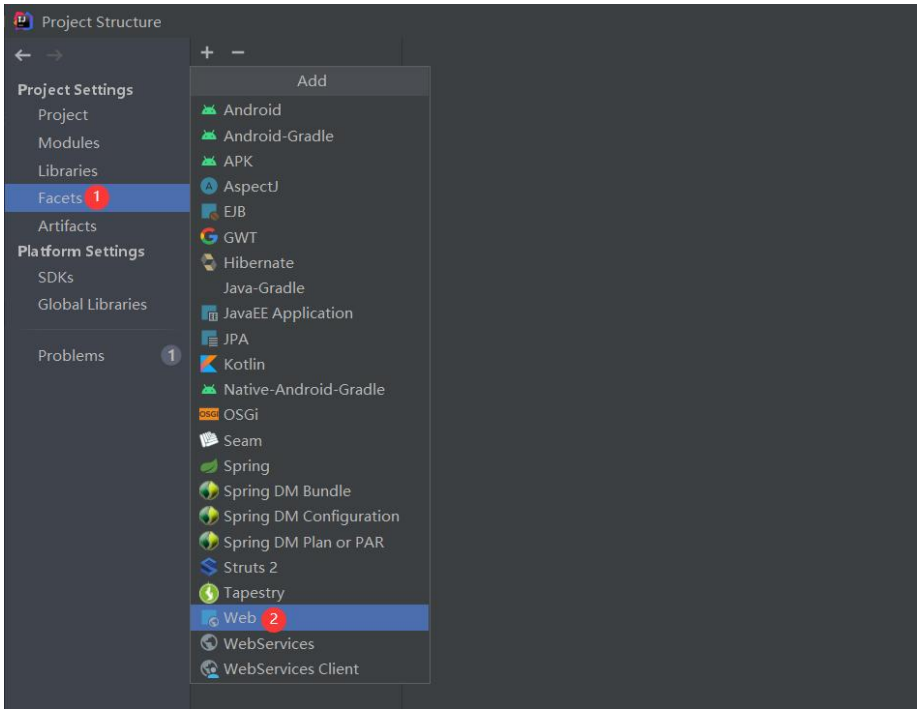
```
# Handler specific properties.
# Describes specific configuration info for Handlers.
############################################################

1catalina.org.apache.juli.AsyncFileHandler.level = FINE
1catalina.org.apache.juli.AsyncFileHandler.directory = ${catalina.base}/logs
1catalina.org.apache.juli.AsyncFileHandler.prefix = catalina.
1catalina.org.apache.juli.AsyncFileHandler.encoding = UTF-8

2localhost.org.apache.juli.AsyncFileHandler.level = FINE
2localhost.org.apache.juli.AsyncFileHandler.directory = ${catalina.base}/logs
2localhost.org.apache.juli.AsyncFileHandler.prefix = localhost.
2localhost.org.apache.juli.AsyncFileHandler.encoding = UTF-8

3manager.org.apache.juli.AsyncFileHandler.level = FINE
3manager.org.apache.juli.AsyncFileHandler.directory = ${catalina.base}/logs
3manager.org.apache.juli.AsyncFileHandler.prefix = manager.
3manager.org.apache.juli.AsyncFileHandler.encoding = UTF-8

4host-manager.org.apache.juli.AsyncFileHandler.level = FINE
4host-manager.org.apache.juli.AsyncFileHandler.directory = ${catalina.base}/logs
4host-manager.org.apache.juli.AsyncFileHandler.prefix = host-manager.
4host-manager.org.apache.juli.AsyncFileHandler.encoding = UTF-8

java.util.logging.ConsoleHandler.level = FINE
java.util.logging.ConsoleHandler.formatter = org.apache.juli.OneLineFormatter
java.util.logging.ConsoleHandler.encoding = GBK


############################################################
```


然后就可以了

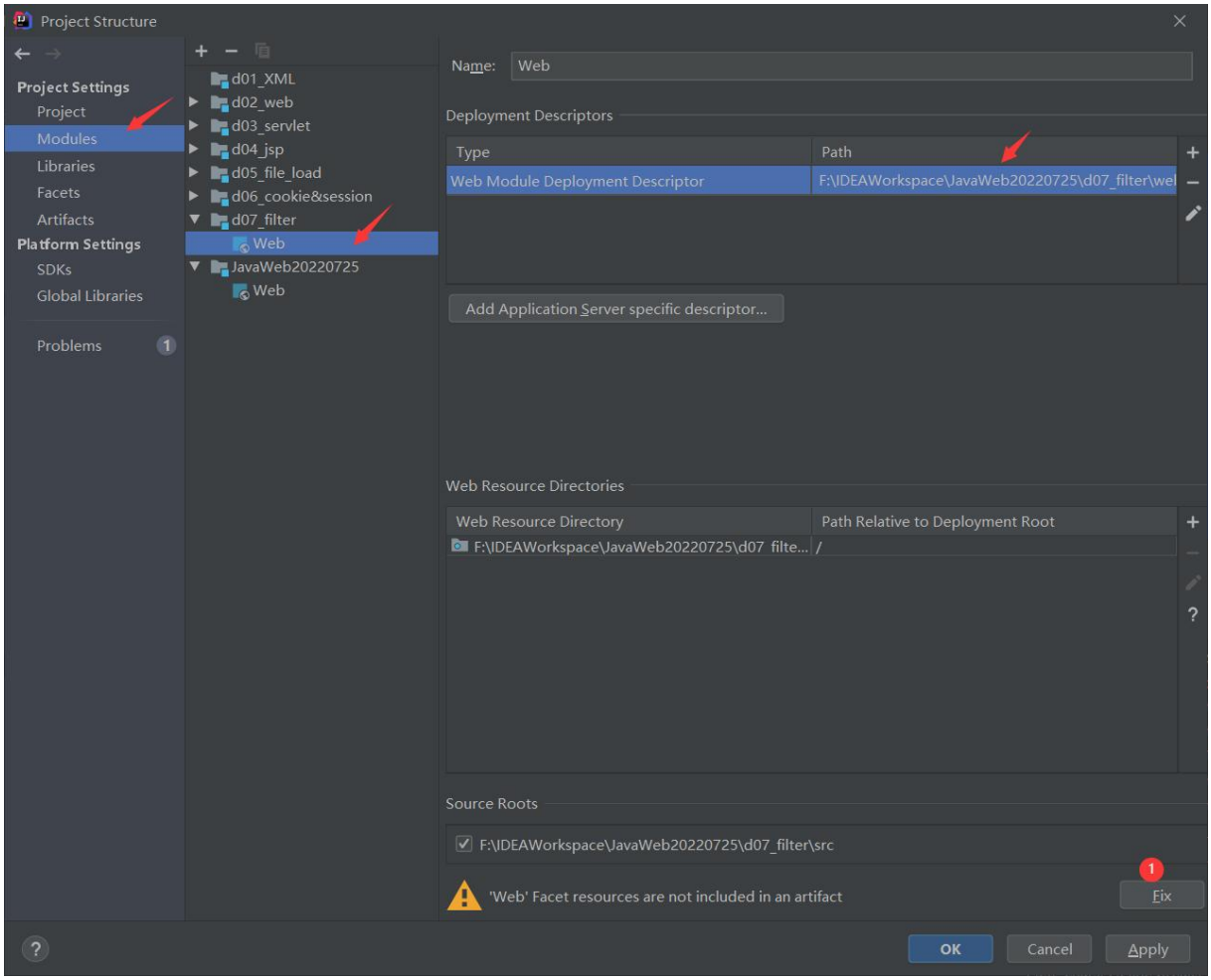
3.10 IDEA 将 java 项目变为 JavaWeb 项目

1、创建一个 Java 项目 – d07_filter

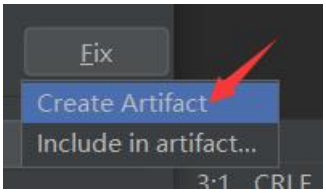
2、选中次 Java 项目 File | Project Structure | Facets



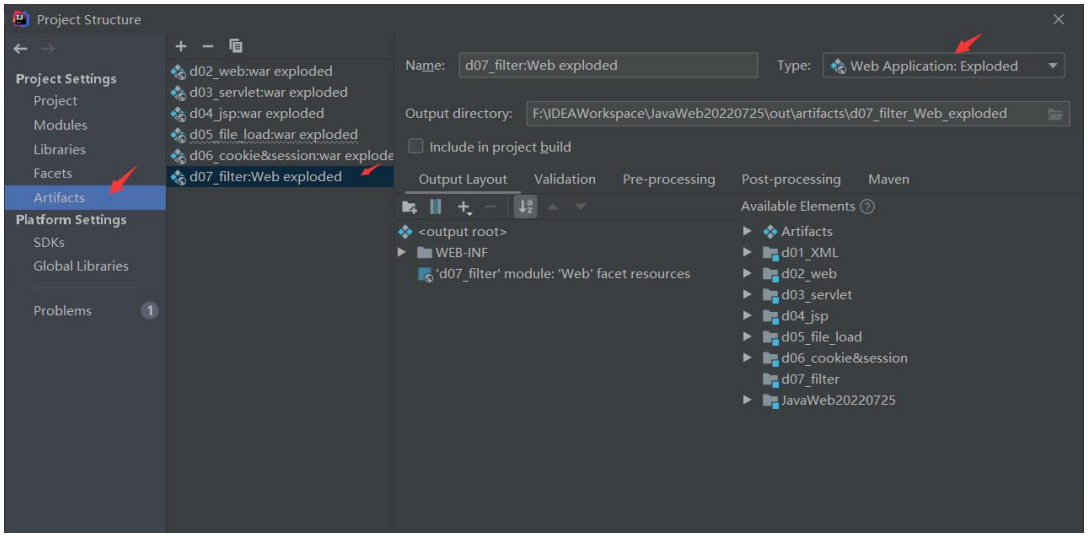
然后会弹出一个对话框，让我们选中添加要将 Web 目录添加到哪个工程中，我们选择好点击确定即可，点击确定后会跳转到 Module 中



点击 Fix，然后会



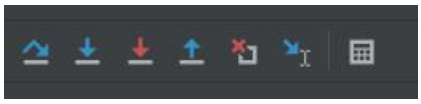
点击 Create Artifact



然后点击 OK 即可

3.11 IDEA 的 Debug 调试

3.11.1 调试工具类



 让代码往下执行一行

 可以进入当前方法体内（自己写的代码，非框架源码）

 强制进入当前方法体内

 跳出当前方法体外

 停在光标所在行（相当于临时断点）

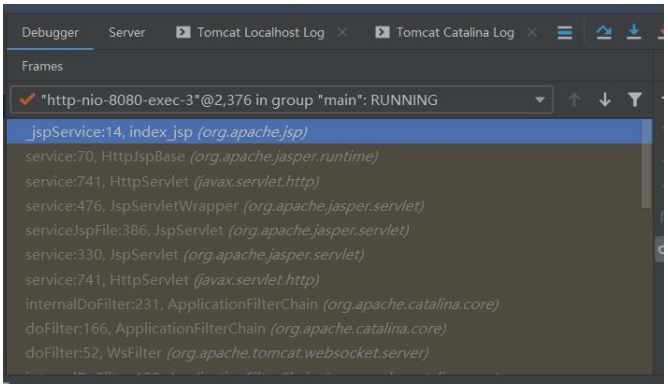
3.11.2 变量窗口

变量窗口：它可以查看当前方法范围内所有有效的变量。

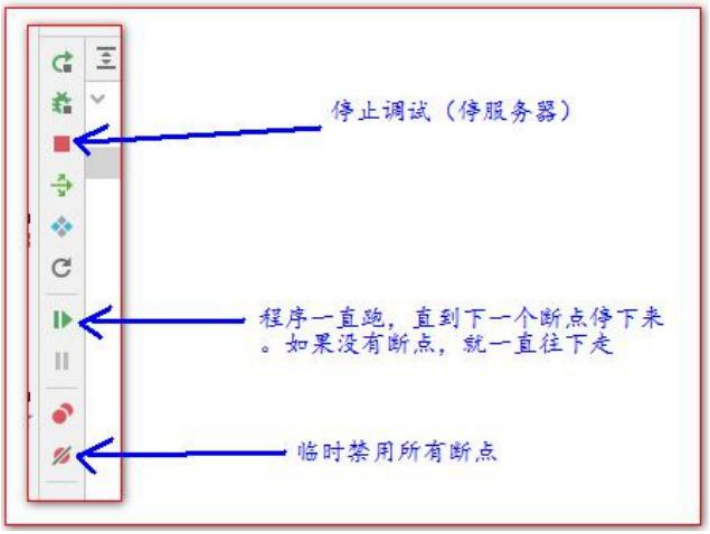
3.11.3 方法调用栈窗口

1、方法调用栈可以查看当前线程有哪些方法调用信息

2、下面的调用上一行的方法



3.11.4 其他常用调试相关按钮



第 4 章 Servlet

4.1 Servlet 快速入门

4.1.1 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。

4.1.2 第一个 Servlet 程序

1、在 IDEA 中创建一个新的 WEB Project
2、创建一个 HelloServlet 类，并实现 Servlet 接口，然后重写 service()方法，其他方法空实现即可

表 4-1 HelloServlet.java

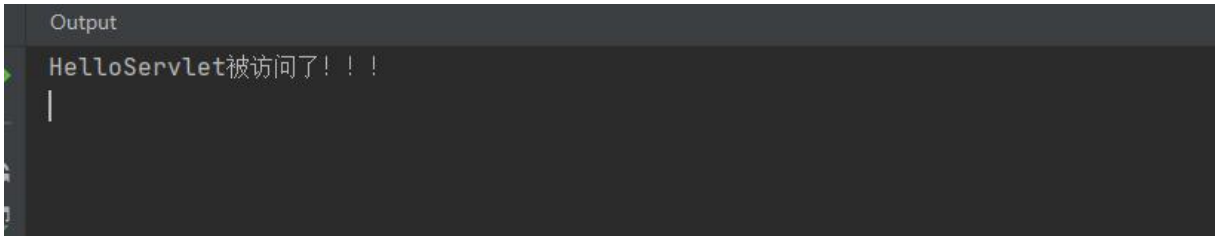
```
package com.lxh.servlet;

import javax.servlet.*;
import java.io.IOException;

public class HelloServlet implements Servlet {
    @Override
    public void init(ServletConfig servletConfig) throws ServletException {

    }

    @Override
    public ServletConfig getServletConfig() {
        return null;
    }

    @Override
    public void service(ServletRequest servletRequest, ServletResponse servletResponse) throws
ServletException, IOException {
        System.out.println("HelloServlet 被访问了！！！");
    }

    @Override
    public String getServletInfo() {
        return null;
    }

    @Override
    public void destroy() {

    }
}
```

3、到 web.xml 文件中配置 Servlet 程序的访问地址

表 4-2 web.xml

```
<?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 标签用于给 Tomcat 配置一个 Servlet 程序  -->
    <servlet>
        <!-- servlet-name 标签用于给 Servlet 其一个别名，一般是类名 -->
        <servlet-name>HelloServlet</servlet-name>
        <!-- servlet-class 用于指定 Servlet 程序的全类名 -->
        <servlet-class>com.lxh.servlet.HelloServlet</servlet-class>
    </servlet>
    <!-- servlet-mapping 标签用于给 Servlet 程序配置访问路径，一个 servlet 标签对应一个
servlet-mapping 标签，一个 servlet 可以对应多个 servlet-mapping，这样访问这些配置好的路径，
都会有同一个 servlet 程序来处理 -->
    <servlet-mapping>
        <!-- servlet-name 标签用于指定给哪个 Servlet 程序配置 -->
        <servlet-name>HelloServlet</servlet-name>
        <!--
            url-pattern 标签用于配置访问路径，可以配置多个访问路径，且必须以"/"开头
            "/" 在服务器解析的时候，表示地址为 http://ip:prot/工程路径/
            "/hello" 表示地址为 http://ip:port/工程路径/hello
        -->
        <url-pattern>/hello</url-pattern>
        <url-pattern>/first</url-pattern>
    </servlet-mapping>

</web-app>
```

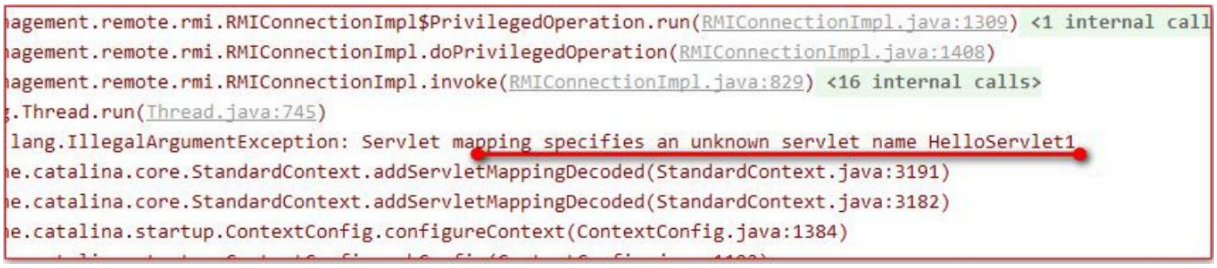
4、访问 HelloServlet 程序，访问地址：http://localhost:8080/d03_servlet/hello 或 http://localhost:8080/d03_servlet/first



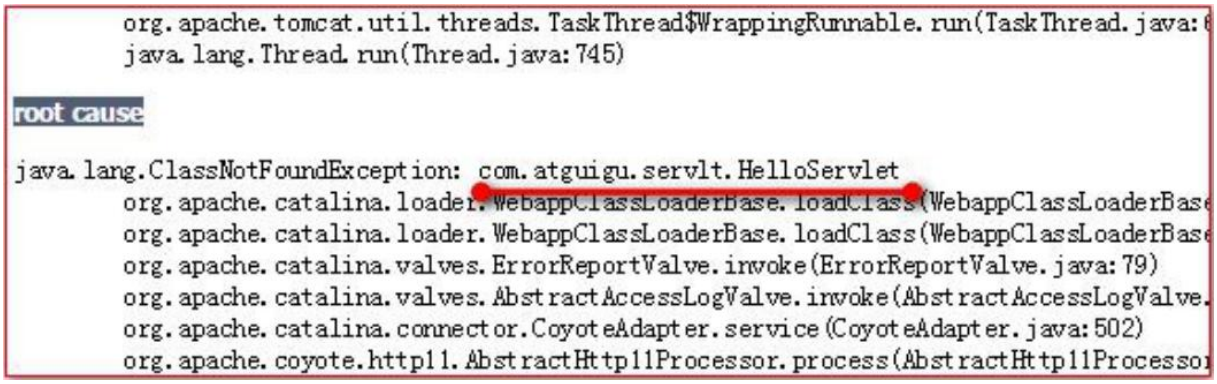
访问成功

常见错误 1： url-pattern 中配置的路径没有以斜杠打头。

常见错误 2：servlet-name 配置的值不存在：



常见错误 3：servlet-class 标签的全类名配置错误：



4.1.3 Servlet 的生命周期

生命周期：从出生到死亡的过程就是生命周期。对应 Servlet 中的三个方法：init(), service(),destroy()。

默认情况下：第一次接收请求时，这个 Servlet 会进行实例化(调用构造方法)、初始化(调用 init())、然后服务(调用 service())；从第二次请求开始，每一次都是服务

当容器关闭时，其中的所有的 servlet 实例会被销毁，调用销毁方法。

Servlet 实例 tomcat 只会创建一个，所有的请求都是这个实例去响应；默认情况下，第一次请求时，tomcat 才会去实例化、初始化 Servlet，然后再服务；这样的好处是什么？ 提高系统的启动速度 。 这样的缺点是什么？ 第一次请求时，耗时较长；如果需要提高系统的启动速度，当前默认情况就是这样。如果需要提高响应速度，我们应该设置 Servlet 的初始化时机。

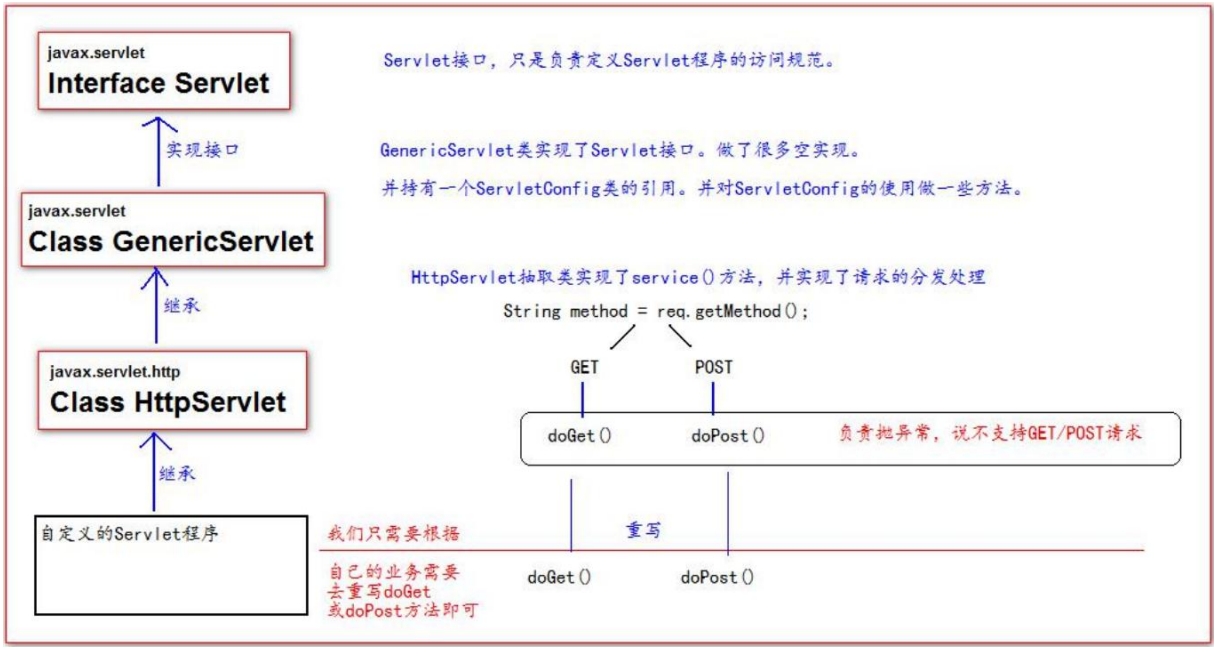
Servlet 的初始化时机：默认是第一次接收请求时，实例化，初始化；我们可以通过<load-on-startup>来设置 servlet 启动的先后顺序,数字越小，启动越靠前，最小值 0

Servlet 在容器中是：单例的、线程不安全的，所有的请求都是同一个实例去响应；一个线程需要根据这个实例中的某个成员变量值去做逻辑判断。但是在中间某个时机，另一个线程改变了这个成员变量的值，从而导致第一个线程的执行路径发生了变化

我们已经知道了 servlet 是线程不安全的，给我们的启发是： 尽量的不要在 servlet 中定义成员变量。如果不得不定义成员变量，那么不要去：①不要去修改成员变量的值 ②不要去根据成员变量的值做一些逻辑判断

4.1.4 Servlet 类的继承体系

图 4-1 Servlet 类的继承体系



相关方法：

javax.servlet.Servlet 接口：

- void init(ServletConfig config) – 初始化方法
- void service(ServletRequest request, ServletResponse response) – 服务方法
- void destroy(): 销毁方法

javax.servlet.GenericServlet 抽象类：

- public abstract void service(request, response) - 仍然是抽象的

javax.servlet.http.HttpServlet 抽象子类：

- void service(request,response) - 不是抽象的
- 1. String method = req.getMethod(); 获取请求的方式
  2. 各种 if 判断，根据请求方式不同，决定去调用不同的 do 方法

```
if (method.equals("GET")) {
    this.doGet(req,resp);
} else if (method.equals("HEAD")) {
    this.doHead(req, resp);
} else if (method.equals("POST")) {
    this.doPost(req, resp);
} else if (method.equals("PUT")) {
```

  3. 在 HttpServlet 这个抽象类中，do 方法都差不多:

```
protected void doGet(HttpServletRequest req, HttpServletResponse resp) thro
```

```
ws ServletException, IOException {
        String protocol = req.getProtocol();
        String msg = lStrings.getString("http.method_get_not_supported");
        if (protocol.endsWith("1.1")) {
            resp.sendError(405, msg);
        } else {
            resp.sendError(400, msg);
        }
    }
```

4.1.5 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 HelloServlet2 区继承 HttpServlet 类

2、根据业务需求重写 doGet()或 doPost()方法

3、到 web.xml 中配置 Servlet 的访问地址

表 4-3

```
package com.lxh.servlet;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;

public class HelloServlet2 extends HttpServlet {

    /**
     * doGet()方法在发生 get 请求时被调用
     * @param req
     * @param resp
     * @throws ServletException
     * @throws IOException
     */
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
        System.out.println("get 请求...");
    }

    /**
     * doPost()方法在发生 post 请求是被调用
     * @param req
     * @param resp
     * @throws ServletException
     * @throws IOException
     */
    @Override
```

```
    protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
        System.out.println("post 请求...");
    }
}
```

表 4-4 web.xml

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

</web-app>
```

- 在浏览器中输入访问地址：http://ip:port/工程路径/hello2 访问，例如我的：http://localhost:8080/d03_servlet/hello2

```
26-Jul-2022 15:42:36.508 警告 [RMI TCP Connection(3)-127.0.0.1] org.apache.tomcat.util.descriptor.web.WebXml.set\
[2022-07-26 03:42:36,770] Artifact d03_servlet:war exploded: Artifact is deployed successfully
[2022-07-26 03:42:36,770] Artifact d03_servlet:war exploded: Deploy took 618 milliseconds
get请求...
26-Jul-2022 15:42:45.710 信息 [localhost-startStop-1] org.apache.catalina.startup.HostConfig.deployDirectory 把we
26-Jul-2022 15:42:45.873 信息 [localhost-startStop-1] org.apache.catalina.startup.HostConfig.deployDirectory Depl
```

可以看到触发了 get 请求

- 在工程下的 web 目录下创建 demo.html 文件模拟 post 请求

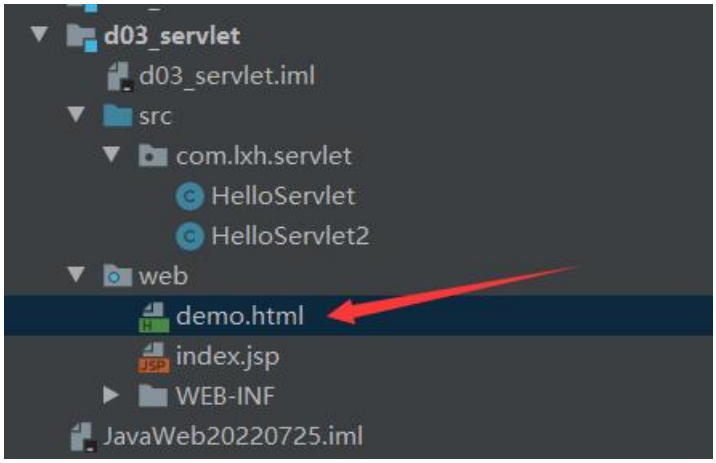


表 4-5 demo.html

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

```
    <title>Title</title>
</head>
<body>
<form action="http://localhost:8080/d03_servlet/hello2" method="post">
    <input type="submit">
</form>
</body>
</html>
```

访问：http://ip:port/工程路径/demo.html，如我的

点击提交

post 请求成功

4.1.6 使用注解开发 Servlet 程序

1、创建一个 Servlet 类并继承 HttpServlet 类，重写相应方法

2、在 Servlet 类上添加注解 @WetServlet，在注解上为该 Servlet 设置 servlet-name 和 url-pattern。

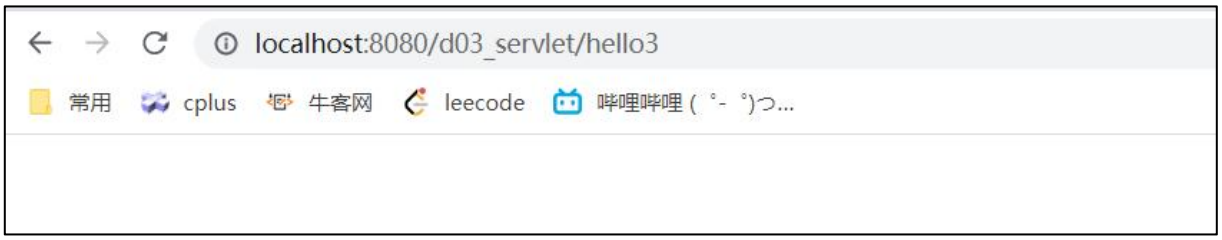
3、访问测试

表 4-6 HelloServlet3.java

```
package com.lxh.servlet;

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 java.io.IOException;

/**
 * @WebServlet(name = "HelloServlet3", value = {"/hello3"})
 * @WebServlet 注解用于配置 Servlet 程序
 * name 属性用于设置 Servlet 的别名，其作用相当于配置文件中的 servlet-name 标签
 * value 用于指定 Servlet 的访问路径，可以指定多个(使用逗号分隔),其中相当于配置文件中的
url-pattern，可以省略 name 属性，直接书写 value 的{}即可
 */
@WebServlet(name = "HelloServlet3", value = {"/hello3"})
```

```
public class HelloServlet3 extends HttpServlet {

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
        System.out.println("get 请求...");
    }
}
```

在浏览器中输入访问地址：http://ip:port/工程路径/hello3，例如我的：




```
Output
get请求...
```

访问成功

4.2 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 Servlet 对应一个 ServletConfig 对象

- **ServletConfig 的作用**

  1) 可以获取 Servlet 程序的别名 servlet-name 的值

  2) 可以获取 Servlet 初始化参数 init-param

  3) 可以获取 ServletContext 对象

- **使用实例：**

表 4-7 web.xml

```
<?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 标签用于给 Tomcat 配置一个 Servlet 程序  -->
    <servlet>
        <!-- servlet-name 标签用于给 Servlet 其一个别名，一般是类名 -->
        <servlet-name>HelloServlet</servlet-name>
        <!-- servlet-class 用于指定 Servlet 程序的全类名 -->
        <servlet-class>com.lxh.servlet.HelloServlet</servlet-class>
        <!--
            init-param 标签用于指定一个 Servlet 程序的默认参数,可以有多个，也可以没有
```

```
            参数形式为键值对形式
            子标签 param-name 用于指定键
            子标签 param-value 用于指定值
        -->
        <init-param>
            <!--
                param-name 标签
            -->
            <param-name>username</param-name>
            <param-value>luxianghai</param-value>
        </init-param>
        <init-param>
            <param-name>passwrod</param-name>
            <param-value>123456</param-value>
        </init-param>
    </servlet>
    <!-- servlet-mapping 标签用于给 Servlet 程序配置访问路径，一个 servlet 标签对应一个
servlet-mapping 标签 -->
    <servlet-mapping>
        <!-- servlet-name 标签用于指定给哪个 Servlet 程序配置 -->
        <servlet-name>HelloServlet</servlet-name>
        <!--
            url-pattern 标签用于配置访问路径，可以配置多个访问路径，且必须以"/"开头
            "/" 在服务器解析的时候，表示地址为 http://ip:prot/工程路径/
            "/hello" 表示地址为 http://ip:port/工程路径/hello
        -->
        <url-pattern>/hello</url-pattern>
        <url-pattern>/first</url-pattern>
    </servlet-mapping>

</web-app>
```

表 4-8 HelloServlet

```
package com.lxh.servlet;

import javax.servlet.*;
import java.io.IOException;

public class HelloServlet implements Servlet {
    @Override
    public void init(ServletConfig servletConfig) throws ServletException {
        System.out.println("HelloServlet 程序的别名 ==>> " + servletConfig.getServletName());
        System.out.println("username 参数值为： " + servletConfig.getInitParameter("username"));
        System.out.println("password 参数值为： " + servletConfig.getInitParameter("password"));

        ServletContext servletContext = servletConfig.getServletContext();
        System.out.println(servletContext);
    }


    @Override
    public void service(ServletRequest servletRequest, ServletResponse servletResponse) throws
ServletException, IOException {
        System.out.println("HelloServlet 被访问了！！！");

    }
```

```
    // 此处省略部分实现的方法，它们均为空实现
}
```

访问测试：

```
[2022-07-26 04:42:22,228] Artifact d03_servlet:war exploded: Deploy took 639 milliseconds
HelloServlet程序的别名 ===> HelloServlet
username参数值为：luxianghai
password参数值为：123456
org.apache.catalina.core.ApplicationContextFacade@3bc32436
HelloServlet被访问了！！！
```

如果我们的 Servlet 是继承自 HttpServlet 的话，可以通过 getServletConfig() 方法来获取 ServletConfig 对象。如果此时我们重写了 init(ServletConfig)方法，那么应该调用父类的 init(ServletConfig)方法，因为 ServletConfig 的赋值是通过 init()方法来完成的。

```
@Override
public void init(ServletConfig config) throws ServletException {
    super.init(config);
}

@Override
protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
    System.out.println("get请求...");
    ServletConfig servletConfig = getServletConfig();
    System.out.println(servletConfig);
}
```

4.3 ServletContext 类

- ServletContext 是什么？
  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。
     - 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。
  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。

表 4-9

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

- ServletContext 的作用
  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

- 代码演示：

表 4-10 ServletContext 类代码演示

```
package com.lxh.servlet;

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 java.io.IOException;
@WebServlet(name = "ContextServlet1", value = {"/context1"})
public class ContextServlet1 extends HttpServlet {

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
        System.out.println("ContextServlet1.doGet()");
        // 获取 web.xml 中通过 context-param 配置的全局参数
        String author = getServletContext().getInitParameter("author");
        System.out.println("author = " + author);
        // 获取当前工程的工程路径
        // "/"被服务器解析为 http://ip:port/工程路径/，映射到 web 目录下
        System.out.println("工程部署的路径为： " + getServletContext().getRealPath("/"));
        System.out.println("工程中 web 目录下的 css 目录的全路径： " +
getServletContext().getRealPath("/css"));
        // 设置域数据，作用域为整个工程
        getServletContext().setAttribute("id", "1001");
        getServletContext().setAttribute("name", "zs");
    }
}
```

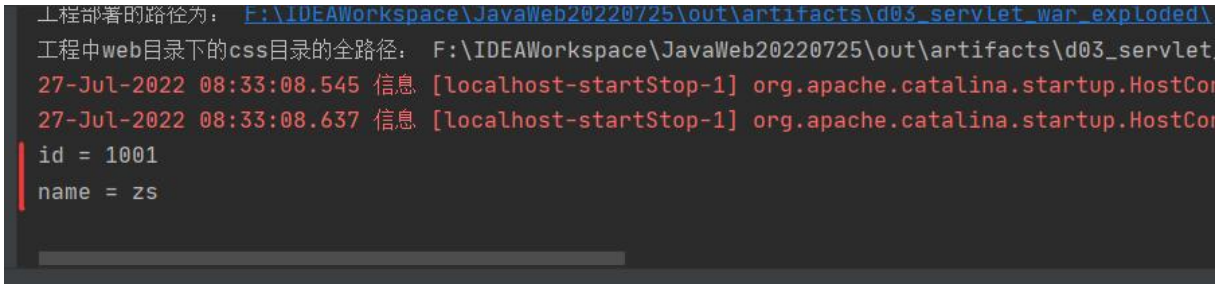
```
package com.lxh.servlet;

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 java.io.IOException;

@WebServlet(name = "ContextServlet2", value = {"/context2"})
public class ContextServlet2 extends HttpServlet {

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

        // 获取域数据
        Object id = getServletContext().getAttribute("id");
        Object name = getServletContext().getAttribute("name");
        System.out.println("id = " + id);
        System.out.println("name = " + name);
    }
}
```

先访问 ContextServlet1，再访问 ContextServlet2

图 4-2 访问 ContextServlet1 控制台输出结果

可以看到，我们已经成功获取到 web.xml 中配置的全局参数，也能够获取到工程部署的绝对路径

图 4-3 访问 ContextServlet2 控制台输出结果



可以看到，我们已经成功获取到了 id 和 name 的值，我们只有先访问 ContextServlet1，再访问 ContextServlet2，才能够获取到 id 和 name 的值，因为 id 和 name 的值是在 ContextServlet2 中设置的。

4.4 HTTP 协议

4.4.1 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。

4.4.2 HTTP 协议请求的格式

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。

1、GET 请求的格式（请求行+请求头）

1) 请求行（请求方式+资源路径+协议版本号）

i. 请求的方式 GET

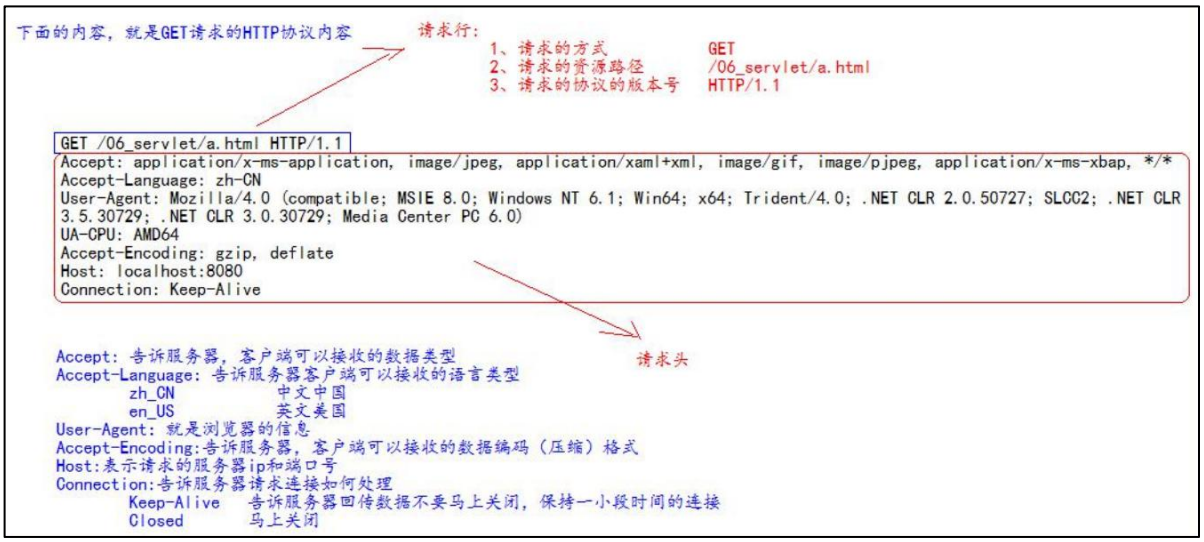
ii. 请求的资源路径[?请求参数]

iii. 请求的协议的版本号　　HTTP/1.1

2) 请求头

请求头由键值对组成，不同的键值对有不同的含义

图 4-4 GET 请求的格式示例



2、POST 请求的格式（请求行+请求头+空行+请求体）

1) 请求行（请求方式+资源路径+协议版本号）

i. 请求方式　　POST

ii. 请求资源的路径[?请求参数]

iii. 请求的协议和版本号　　HTTP/1.1

2) 请求头

请求头由键值对组成

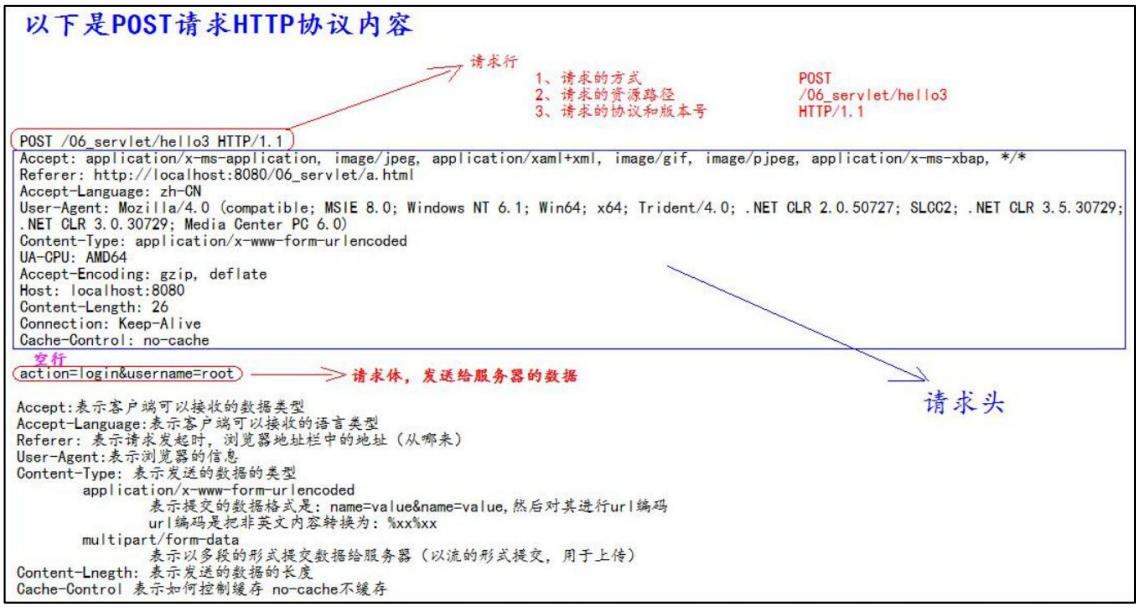
3) 空行

空行的目的是将请求头和请求体分隔开来

4) 请求体

请求体就是发送给服务器的数据

图 4-5 POST 请求示例



4.4.3 常用请求头的说明

- Accept: 表示客户端可以接收的数据类型
- Accept-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
- Referer：先前网页的地址，当前请求网页紧随其后,即来路

4.4.4 请求体

get 方式，没有请求体，但是有一个 queryString

post 方式，有请求体，form data

json 格式，有请求体，request payload

4.4.5 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：

1、form 标签 method=post

4.4.6 HTTP 协议响应的格式

HTTP 响应的格式为：响应行+响应头+空行+响应体

1、响应行（响应的协议及版本+状态码+状态描述符）

i. 响应的协议及其版本号

ii. 响应状态码

iii. 响应状态描述符

2、响应头

响应头有键值对组成

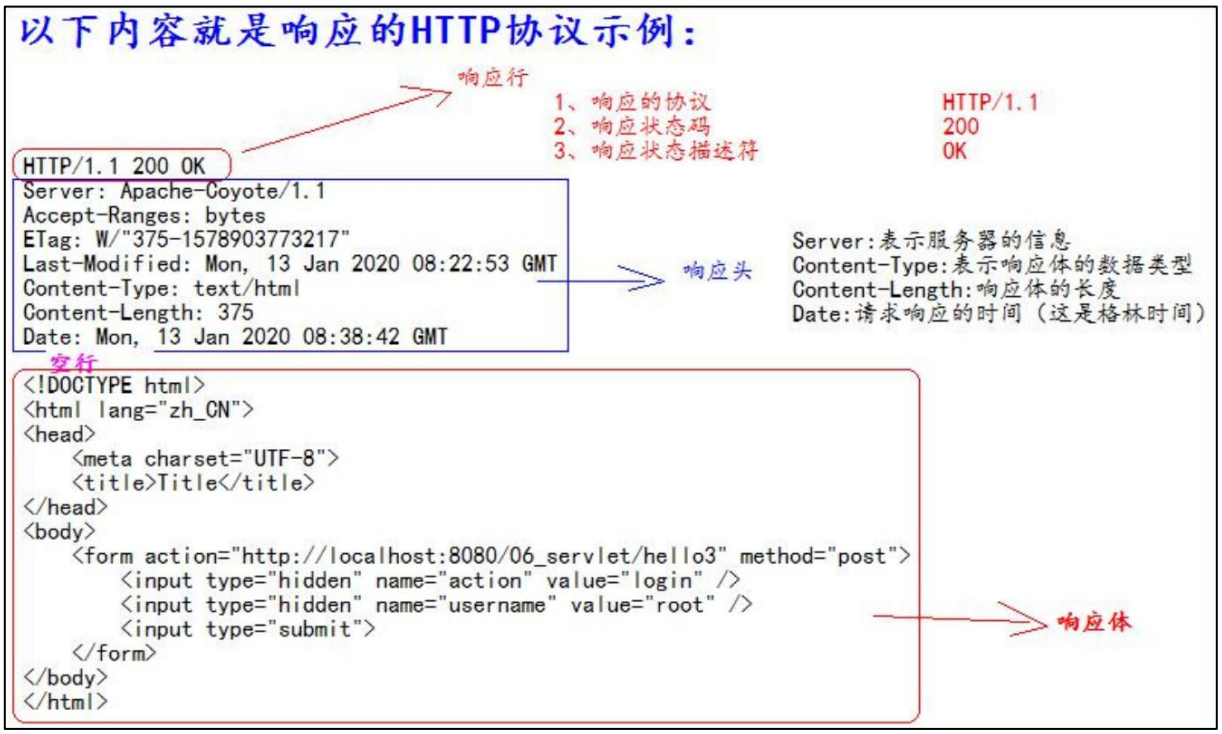
3、空行

空行的目的是将响应头和响应体分隔开来

4、响应体

就是服务器回传给客户端浏览器的数据

图 4-6 响应体示例



- 常见的响应状态码

表 4-11 常见响应状态码

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，但是你要的数据不存在（请求地址错误）
405	不支持当前请求方式（get/post）
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）

4.4.7 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：

表 4-12 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

4.5 HttpServletRequest 类

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。

4.5.1 HttpServletRequest 的常用方法

表 4-13 HttpServletRequest 类的常用方法

方法	说明

getRequestURI()	获取请求的资源路径
getRequestURL()	获取请求的统一资源定位符（绝对路径）
getRemoteHost()	获取客户端的 ip 地址
getHeader()	获取请求头
getParameter()	获取请求的参数
getParameterValues()	获取请求的参数（获取的键有多个值对应时使用）
getMethod()	获取请求的方式 GET 或 POST
setAttribute(key, value);	设置域数据
getAttribute(key);	获取域数据
getRequestDispatcher()	获取请求转发对象
getContextPath()	获取工程路径，格式为：/工程路径
getServletPath()	获取请求资源路径，如浏览器地址为：http://localhost:8080/d07_filter/demo.do，则调用此方法会返回 “/demo.do”

➢ 常用 API 代码示例

表 4-14

```
package com.lxh.request;

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 java.io.IOException;
@WebServlet(name = "RequestAPIServlet", value = "/request_api")
public class RequestAPIServlet extends HttpServlet {

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
        // 获取 URI 和 URL
        System.out.println("URI ==>>> " + req.getRequestURI()); // /d03_servlet/request_api
        System.out.println("URL ==>>> " + req.getRequestURL()); //
http://localhost:8080/d03_servlet/request_api
        // 获取访问服务器的客户端 IP 地址
        System.out.println("客户端 IP 地址 ==>>> " + req.getRemoteHost()); // 0:0:0:0:0:0:0:1
        // 获取指定的请求头
        System.out.println("请求头 User-Agent ==>>> " + req.getHeader("User-Agent"));
        // 获取请求方式
        System.out.println("请求方式 ==>>> " + req.getMethod()); // GET
    }
}
```

图 4-7 控制台输出结果

```
URI ==>>> /d03_servlet/request_api
URL ==>>> http://localhost:8080/d03_servlet/request_api
客户端IP地址 ==>>> 0:0:0:0:0:0:0:1
请求头User-Agent ==>>> Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/10
请求方式 ==>>> GET
27-Jul-2022 10:45:31.825 信息 [localhost-startStop-1] org.apache.catalina.startup.HostConfig.deployDirectory 把we
```

4.5.2 获取请求参数

如果获取的是 GET 请求的参数，则在 doGet()方法中获取。如果获取的是 POST 请求的参数，则在 doPost()方法中获取即可。

1、在 web 目录下创建 param.html 文件

2、创建 ParamServlet.java

表 4-15 param.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<form action="http://localhost:8080/ d03_servlet/req_param" method="get">
    用户名：<input type="text" name="username"><br>
    密码：<input type="text" name="password"><br>
    爱好：<input type="checkbox" name="hobbies" value="篮球">篮球
    <input type="checkbox" name="hobbies" value="羽毛球">羽毛球
    <input type="checkbox" name="hobbies" value="乒乓球">乒乓球<br>
    <input type="submit">
</form>
</body>
</html>
```

表 4-16 ParamServlet.java

```
package com.lxh.request;

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 java.io.IOException;
import java.util.Arrays;

@WebServlet(name = "ParamServlet", value = {"/req_param"})
public class ParamServlet extends HttpServlet {

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
        // 获取参数值是根据表单中的 name 值来获取的
        // getParameter() - 获取指定 key 的参数
        String username = req.getParameter("username");
        String password = req.getParameter("password");
        System.out.println("username = " + username);
        System.out.println("password = " + password);
        // 获取复选框的值
        String[] hobbies = req.getParameterValues("hobbies");
```

```
        System.out.println("hobbies = " + Arrays.asList(hobbies));
    }
}
```

访问：

图 4-8 控制台输出结果

```
27-Jul-2022 11:02:05.836 信息 [localhost-startStop-1] org.apache.catalina.startup.HostConfig.
username = zs
password = 123456
hobbies = [篮球, 羽毛球]
```

可以看到，我们已经成功的获取到了表单中的信息

4.5.3 解决获取请求参数时的中文乱码问题

将 para.html 中表单的提交方式改为 post 方式，因为 get 方式不会出现中文乱码，post 请求才会出现中文乱码

访问：

图 4-9 访问结果

```
27-Jul-2022 11:12:33.691 信息 [localhost-startStop-1] org.apache.catalina.startup.HostConfi
27-Jul-2022 11:12:33.735 信息 [localhost-startStop-1] org.apache.catalina.startup.HostConfi
username = ?°????
password = 123456
hobbies = [??????, ?????????]
```

可以看到，中文没有能够显示出来

解决：

```java
@Override
protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
    // 设置请求体的字符集为utf-8，从而解决中文乱码问题
    req.setCharacterEncoding("utf-8");
    // 获取参数值是根据表单中的 name 值来获取的
    // getParameter() - 获取指定key的参数
    String username = req.getParameter( s: "username");
    String password = req.getParameter( s: "password");
    System.out.println("username = " + username);
    System.out.println("password = " + password);
    // 获取复选框的值
    String[] hobbies = req.getParameterValues( s: "hobbies");
    System.out.println("hobbies = " + Arrays.asList(hobbies));
}
```

我们使用 setCharacterEncoding()方法来设置请求体的字符集为 utf-8 即可

再次访问：

```
[2022-07-27 11:14:51,322] Artifact d03_servlet:war exploded: Artifact is being deployed, please wait...
27-Jul-2022 11:14:51.538 警告 [RMI TCP Connection(5)-127.0.0.1] org.apache.tomcat.util.descriptor.web.WebXml.se
[2022-07-27 11:14:51,784] Artifact d03_servlet:war exploded: Artifact is deployed successfully
[2022-07-27 11:14:51,784] Artifact d03_servlet:war exploded: Deploy took 462 milliseconds
username = 小明
password = 123456
hobbies = [篮球, 乒乓球]
```

中文乱码问题已解决

以上是对于 Tomcat8 开始的中文乱码的解决方案

Tomcat8 之前的中文乱码解决方案如下：

表 4-17 Tomcat8 之前中文乱码解决方案

1)get 请求方式：

//get 方式目前不需要设置编码（基于 tomcat8）

//如果是 get 请求发送的中文数据，转码稍微有点麻烦（tomcat8 之前）

String fname = request.getParameter("fname");

//1.将字符串打散成字节数组

byte[] bytes = fname.getBytes("ISO-8859-1");

//2.将字节数组按照设定的编码重新组装成字符串

fname = new String(bytes,"UTF-8");

2)post 请求方式：

request.setCharacterEncoding("UTF-8");

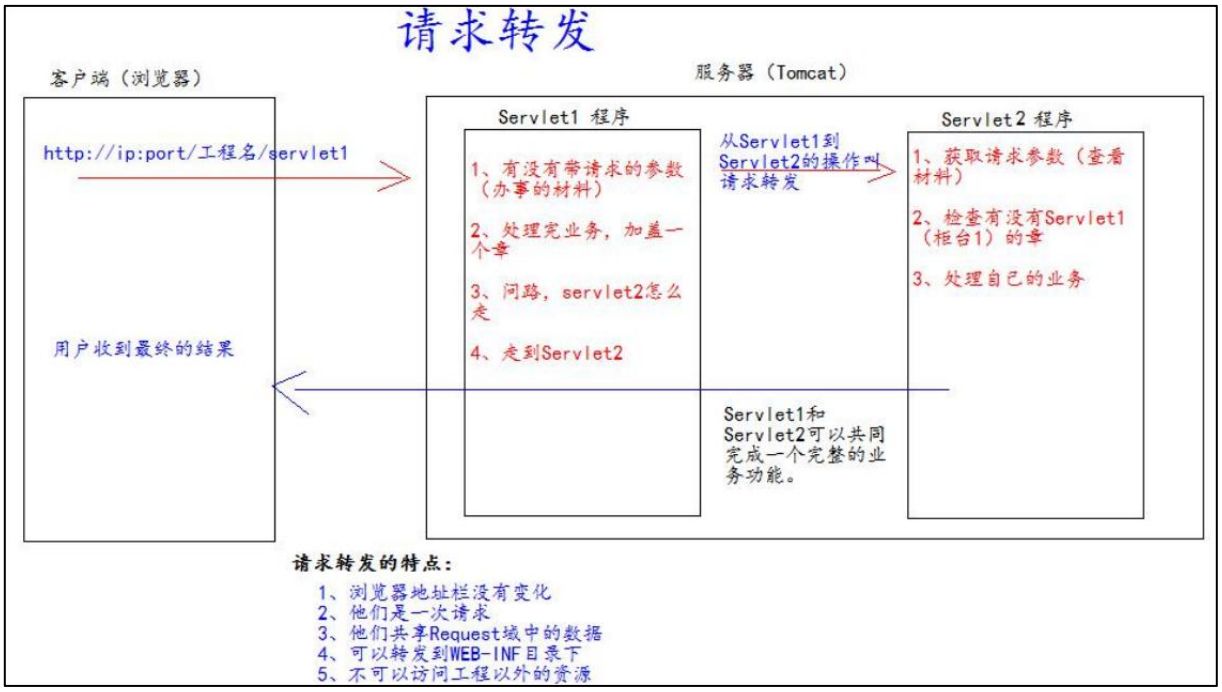
tomcat8 开始，设置编码，只需要针对 post 方式

request.setCharacterEncoding("UTF-8");

4.5.4 请求转发

请求转发是指，服务器收到请求后，从一次资源跳转到另一个资源的操作叫请求转发。

图 4-10 请求转发示例图



请求转发示例代码：

表 4-18 RequestServlet1.java

```
package com.lxh.request;

import javax.servlet.RequestDispatcher;
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 java.io.IOException;

@WebServlet(name = "RequestServlet1", value = "/request1")
public class RequestServlet1 extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
        System.out.println("我是  RequestServlet1 ...");
        /**
         * 请求转发
         * 必须以"/"打头，"/"表示地址为 http://ip:port/工程名/，映射到 web 目录下
         */
        RequestDispatcher requestDispatcher = req.getRequestDispatcher("/request2");
//          RequestDispatcher requestDispatcher = req.getRequestDispatcher("/param.html");
        requestDispatcher.forward(req, resp);
    }
}
```

表 4-19 RequestServlet2.java

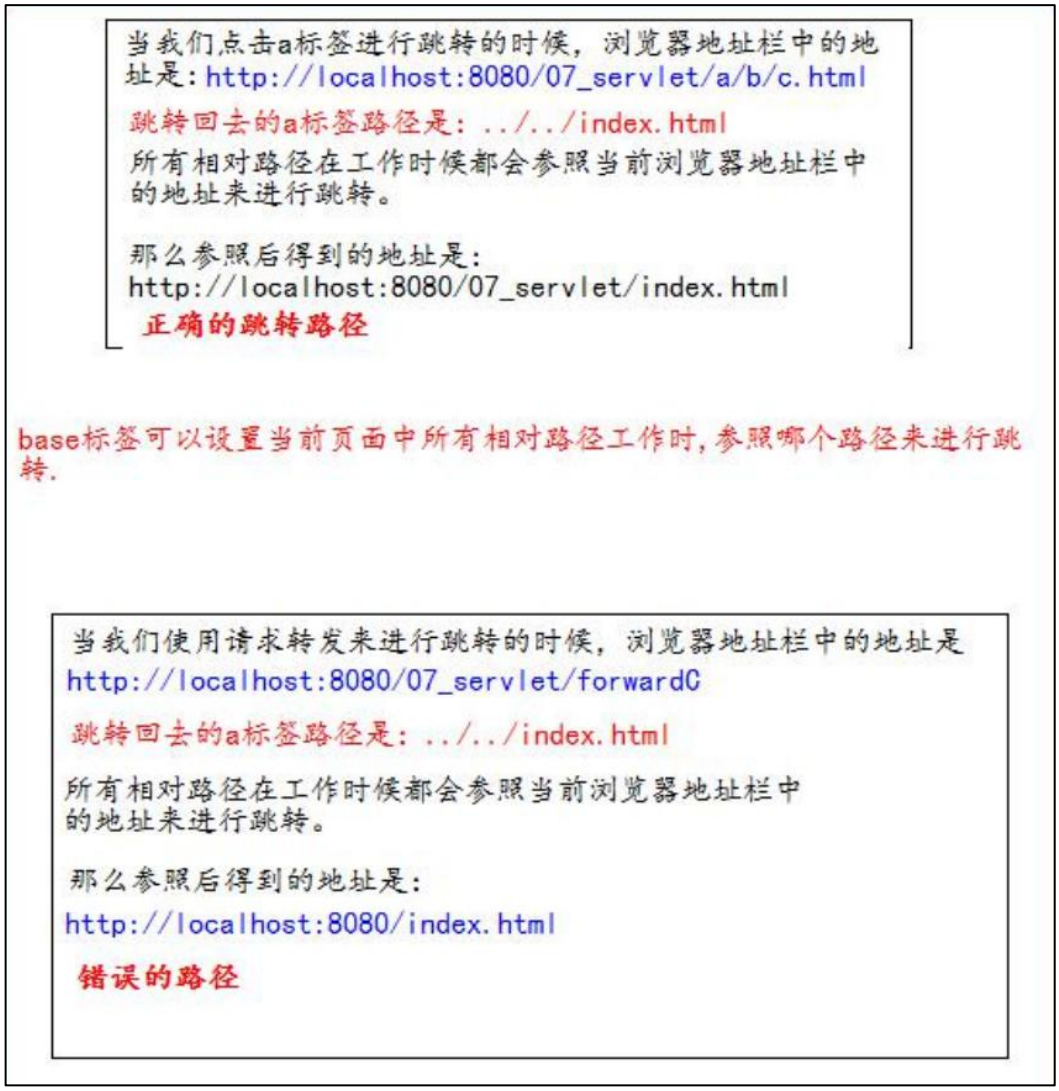
```
package com.lxh.request;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
```

```java
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;

@WebServlet(name = "RequestServlet2", value = "/request2")
public class RequestServlet2 extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
        System.out.println("我是  RequestServlet2 ...");

    }
}
```

我们直接访问 RequestServlet1 即可看出效果

4.5.5 HTML 中 base 标签

图 4-11 base 标签



4.5.6 Web 中“/”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/，如：斜杠

“/”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.sendRediect(“/”); 把斜杠发送给浏览器解析。得到 http://ip:port/

4.6 HttpServletResponse 类

HttpServletResponse 类和 HttpServletRequest 类一样。每次客户端请求进来，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

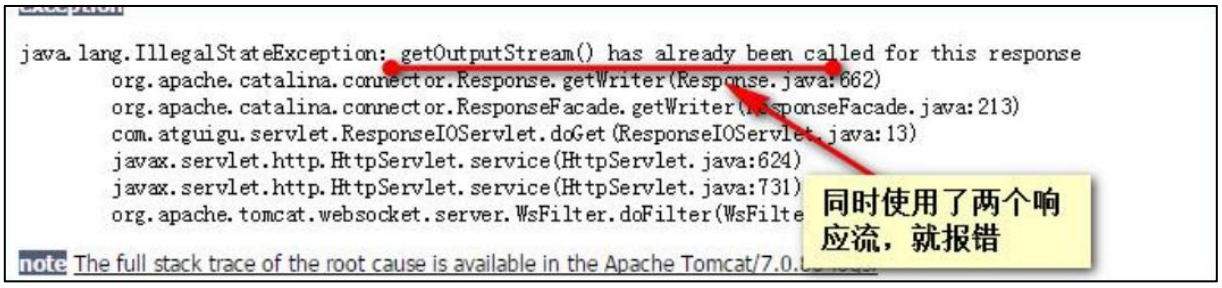
4.6.1 两个输出流的说明

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。



4.6.2 如何往客户端回传数据

要求：往客户端回传 字符串 数据。

表 4-20 ResponseIOServlet.java

```
package com.lxh.response;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
```

```java
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.io.PrintWriter;

@WebServlet(name = "ResponseIOServlet", value = {"/resp_io"})
public class ResponseIOServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
        PrintWriter writer = resp.getWriter();
        writer.write("hello response!!!");
    }
}
```

访问：

4.6.3 解决响应中文乱码问题

- 方案一：

表 4-21 解决响应中文乱码

```java
// 设置服务器字符集为 utf-8
resp.setCharacterEncoding("UTF-8");
// 通过响应头设置浏览器使用 utf-8 字符集
resp.setHeader("Content-Type", "text/html;charset=utf-8");
```

- 方案二（推荐）：

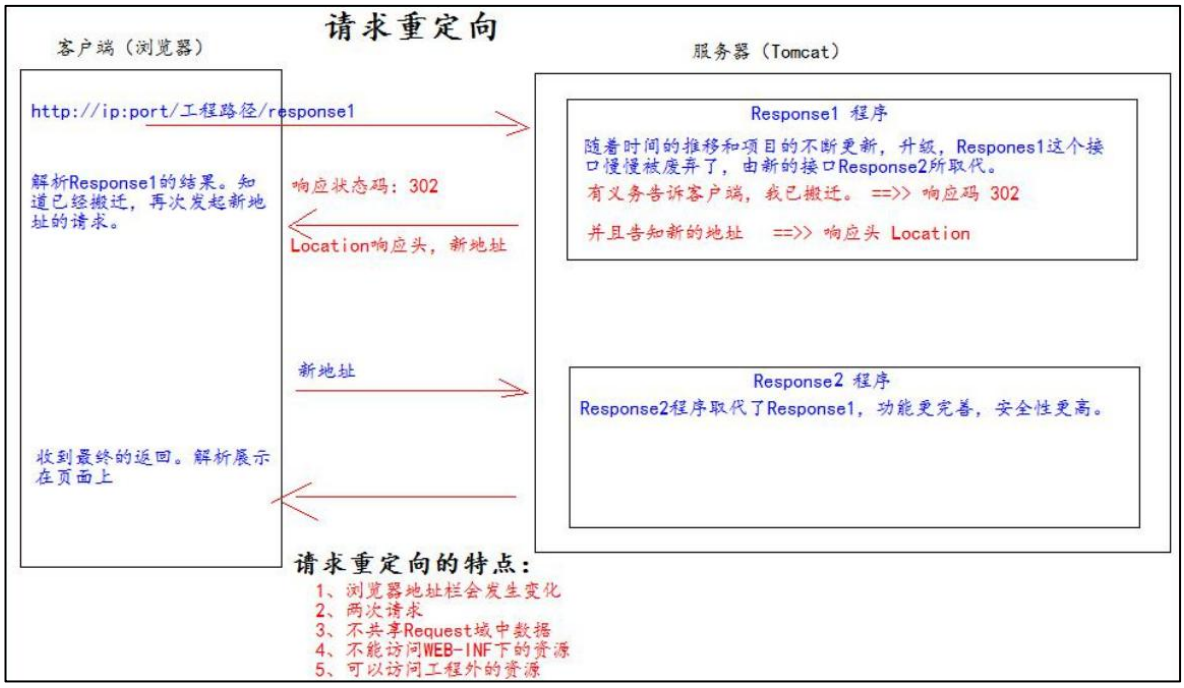
表 4-22 解决响应中文乱码

```java
// setContentType()方法会同时设置浏览器和服务器均使用 utf-8 编码
// 此方法一定要在获取流之前调用
resp.setContentType("text/html;charset=utf-8");
```

4.6.4 请求重定向

请求重定向，是指客户端给服务器发请求，然后服务器告诉客户端说。我给你一些地址。你去新地址访问。叫请求 重定向（因为之前的地址可能已经被废弃）。

图 4-12 请求重定向示例图



- 方式一：

表 4-23 请求重定向代码

```
// 设置状态码，302 表示重定向
resp.setStatus(302);
// 设置响应头，说明新的地址在哪里
resp.setHeader("location","http://localhost:8080/d03_servlet");
```

- 方式二：

表 4-24 请求重定向

```
resp.sendRedirect("/d03_servlet");
```

这里的”/”表示 http://ip:port/

“/d03_servlet”则表示 http://ip:port/d03_servlet

4.7 使用 BaseServlet 优化 Servlet 程序

通过之前的使用我们可以看到，每一个功能都会对应一个 Servlet 程序，通常一个工程中的功能可能会有上百个，如果我们均要为每个功能创建一个 Servlet 程序与之对应，显然不是很优雅。

实际开发中我们会根据一个模型创建一个对应的 Servlet，一个 Servlet 就要完成对该模型的功能。比如 User 模型，我们需要对其进行实现 getOne()、getList()、save()、delete()，我们就将其封装到 UserServlet 程序中即可。

所以我们需要抽取处一个 BaseServlet 程序来将请求进行分发处理。

BaseServlet.java 代码如下：

表 4-25

```
package com.lxh;
```

```
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.lang.reflect.Method;

public class BaseServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
        action(req, resp);
    }

    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
        this.doGet(req, resp);
    }

    protected void action(HttpServletRequest req, HttpServletResponse resp) {
        try {
            // action 参数写就是我们要调用的方法名
            String action = req.getParameter("action");
            /*
                获取要调用的方法，
                参数 1 为方法名
                参数 2 为形参列表
             */
            Method method = this.getClass().getDeclaredMethod(action, HttpServletRequest.class,
HttpServletResponse.class);
            //  调用方法，参数 1 为方法的调用者，参数 2 为形参列表
            method.invoke(this, req, resp);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

使用示例

表 4-26

```
import com.lxh.BaseServlet;
import com.lxh.util.CookieUtils;

import javax.servlet.annotation.WebServlet;
import javax.servlet.http.Cookie;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.io.PrintWriter;

@WebServlet({"/test_servlet"})
public class TestServlet extends BaseServlet {


    public void add(HttpServletRequest req, HttpServletResponse resp) {
        System.out.println("add...");
```

```
    }

    public void delete(HttpServletRequest req, HttpServletResponse resp) {
        System.out.println("delete...");
    }

    public void getOne(HttpServletRequest req, HttpServletResponse resp) {
        System.out.println("getOne...");
    }
}
```

调用 add() 方法：http://ip:port/工程名/test_servlet?action=add

调用 delete() 方法：http://ip:port/工程名/test_servlet?action=delete

调用 getOne() 方法：http://ip:port/工程名/test_servlet?action=getOne

第 5 章 文件上传和下载

5.1 文件上传

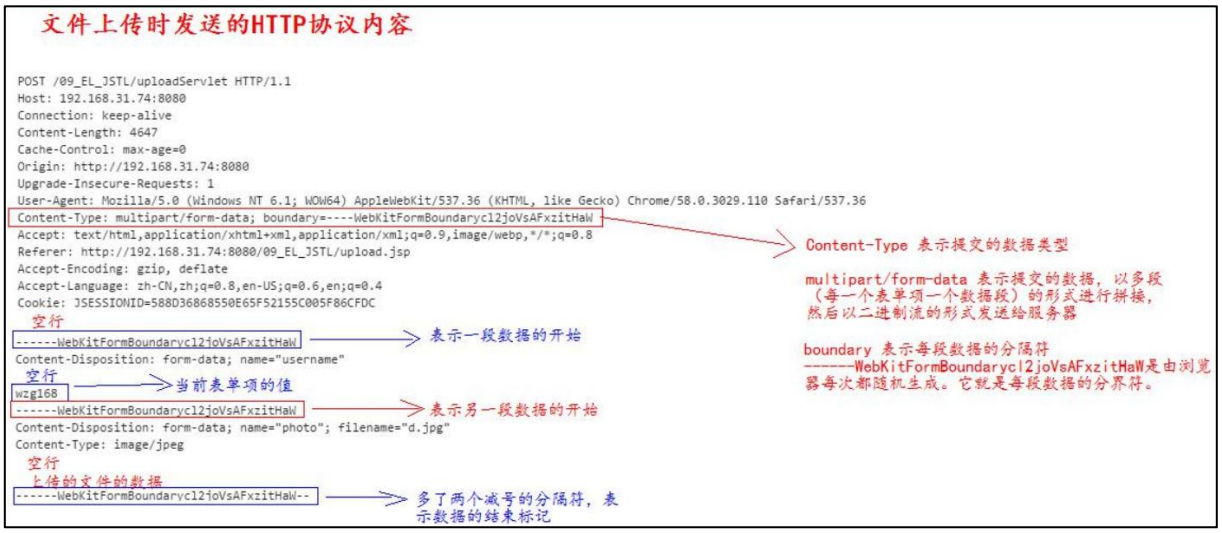
5.1.1 编写文件上传功能步骤：

1. 要有一个 form 标签，method=post 请求
2. form 标签的 encType 属性值必须为 multipart/form-data 值
3. 在 form 标签中使用 input type=file 添加上传的文件
4. 编写服务器代码（Servlet 程序）接收，处理上传的数

encType=multipart/form-data 表示提交的数据，以多段（每一个表单项一个数据段）的形式进行拼 接，然后以二进制流的形式发送给服务器

5.1.2 文件上传时的 HTTP 协议说明

图 5-1 文件上传时的 HTTP 协议说明



5.1.3 commons-fileupload.jar 常用 API 介绍

commons-fileupload.jar 用于已经封装好了文件上传和下载的一些常用方法，我们只需要调用其中的方法即可轻松实现文件的上传和下载。

commons-fileupload.jar 需要依赖 commons-io.jar 这个包，所以两个包我们都要引入。

commons-fileupload.jar 和 commons-io.jar 常用类：

1. ServletFileUpload 类，用于解析上传的数据。
2. FileItem 类，表示每一个表单项。

- 常用 API：

- boolean ServletFileUpload.isMultipartContent(HttpServletRequest request);
  判断当前上传的数据格式是否是多段的格式
- public List<FileItem> parseRequest(HttpServletRequest request)
  解析上传的数据，将其放到 List 集合中
- boolean FileItem.isFormField()
  判断当前这个表单项，是否是普通的表单项，还是文件类型。
  true 表示普通类型的表单项
  false 表示上文件类型
- String FileItem.getFieldName()
  获取表单项的 name 属性值
- String FileItem.getString()
  获取当前表单项的值
- void FileItem.write(file);
  将上传的文件写到 参数 file 所指向抽硬盘位置
- long getSize();
  获取文件大小，单位为字节
- String getContentType()
  获取文件类型，如 image/jpeg
- String getName()
  获取文件名

5.1.4 文件上传实现

表 5-1 upload.jsp

```
<%--
  Created by IntelliJ IDEA.
  User: luxianghai
  Date: 2022/7/28
  Time: 11:53
  To change this template use File | Settings | File Templates.
--%>
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>Title</title>
</head>
<body>
    <form action="/d05_file_load/upload" enctype="multipart/form-data" method="post">
        用户名：<input type="text" name="username"><br>
        头像：<input type="file" name="photo">
        <input type="submit" value="提交">
    </form>
</body>
```

```
</html>
```

表 5-2 UploadServlet.java

```
package com.lxh.file;

import org.apache.commons.fileupload.FileItem;
import org.apache.commons.fileupload.FileItemFactory;
import org.apache.commons.fileupload.FileUploadException;
import org.apache.commons.fileupload.FileUploadException;
import org.apache.commons.fileupload.disk.DiskFileItemFactory;
import org.apache.commons.fileupload.servlet.ServletFileUpload;

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 java.io.File;
import java.io.IOException;
import java.util.List;

@WebServlet(name = "UploadServlet", value = {"/upload"})
public class UploadServlet extends HttpServlet {

    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
        req.setCharacterEncoding("UTF-8");
        resp.setContentType("text/html;charset=utf-8");
        // 1.判断是否是多段数据上传的表单（多段数据才可能是文件上传）
        if (ServletFileUpload.isMultipartContent(req)) {
            // 创建 FileItemFactory 工厂实现类
            FileItemFactory fileItemFactory = new DiskFileItemFactory();
            // 创建用于解析上传数据的工具类 ServletFileUpload
            ServletFileUpload servletFileUpload = new ServletFileUpload(fileItemFactory);
            try {
                // 将上传的数据解析到 List 集合中
                List<FileItem> list = servletFileUpload.parseRequest(req);
                for (FileItem item: list) {
                    if (item.isFormField()) {
                        // 普通表单项
                        // 获取表单项的 name 值
                        String fieldName = item.getFieldName();
                        // 获取表单项的 value 值,指定编码为 utf-8
                        String value = item.getString("UTF-8");
                        System.out.println(fieldName + " = " + value);
                    } else {
                        // 文件
                        System.out.println("表单项的 name 值： " + item.getFieldName());
                        System.out.println("上传的文件名： " + item.getName());
                        // 获取工程部署路径 "/"映射到 http://ip:port/工程路径/
                        String realPath = getServletContext().getRealPath("/");
//                        System.out.println("realPath = " + realPath);
//                        long size = item.getSize();
//                        System.out.println("size = " + size);
//                        String contentType = item.getContentType();
//                        System.out.println("contentType = " + contentType);
                        // 限制文件类型只能为 .jpg 和 .jpeg
```

```
                                if (!item.getName().endsWith(".jpg")
&& !item.getName().endsWith(".jpeg")) {
                                    throw new RuntimeException("文件类型不符");
                                }
                                // 限制文件大小最大为 200KB：102400Byte = 1024Byte*100 = 100KB
--》 *2=200KB
                                if (item.getSize() > 102400*2) {
                                    throw new RuntimeException("文件太大，超过 200KB 了");
                                }
                                item.write(new File(realPath + "\\file\\" + item.getName()));
                            }
                        }
                    } catch (Exception e) {
                        e.printStackTrace();
                    }
                }

            }
}
```

5.2 文件下载

5.2.1 编写文件下载功能的步骤：

1. 获取要下载文件的路径和文件名
2. 根据文件名获取文件对应的 MIME 类型
3. 设置响应数据类型，设置为文件对应的 MIME 类型即可
4. 通过响应头设置回传到客户端的文件的处理处理方式。【可选】
5. 根据文件获取其对应的输入流
6. 获取响应输出流
7. 将文件输入流复制到响应输出流中

5.2.2 实现文件下载

表 5-3 文件下载代码如下 DownloadServlet.java

```
package com.lxh.file;

import org.apache.commons.io.IOUtils;

import javax.servlet.ServletException;
import javax.servlet.ServletOutputStream;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.io.InputStream;
import java.net.URLEncoder;

@WebServlet(name = "DownloadServlet", value = {"/download"})
```

```
public class DownloadServlet extends HttpServlet {

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
        // 要下载的目标文件, targetFile 中的第一个"/"表示 http://ip:port/工程路径/
        // "/file/" 表示地址为 http://ip:port/工程路径/file/
        String fileName = "无标题.jpg";
        String targetFile = "/file/" + fileName;
        // 获取目标文件的 MimeType 类型
        String mimeType = getServletContext().getMimeType(targetFile);
        //System.out.println("mimeType = " + mimeType);
        // 设置响应给客户端的文件是什么样的格式
        resp.setContentType(mimeType);
        // Content-Disposition 响应头表示客户端接收到的数据要怎么处理
        // attachment 表示附件，表示下载使用
        String str = "attachment; filename=" + URLEncoder.encode(fileName, "UTF-8");
        resp.setHeader("Content-Disposition", str);

        // 获取输入流
        InputStream resourceAsStream = getServletContext().getResourceAsStream(targetFile);
        // 获取响应输出流
        ServletOutputStream outputStream = resp.getOutputStream();
        // 将输入流的数据复制到响应输出流中，完成将数据传递到客户端
        IOUtils.copy(resourceAsStream, outputStream);
    }
}
```

5.2.3 解决 Chrome 和 IE 附件中文名乱码问题

如果客户端浏览器是 IE 浏览器 或者 是谷歌浏览器。我们需要使用 URLEncoder 类先对中文名进行 UTF-8 的编码 操作。

因为 IE 浏览器和谷歌浏览器收到含有编码后的字符串后会以 UTF-8 字符集进行解码显示。

代码如下：

表 5-4 使用 URL 编码器解决中文乱码问题

```
String str = "attachment; filename=" + URLEncoder.encode(fileName, "UTF-8");
resp.setHeader("Content-Disposition", str);
```

5.2.4 解决早期 FireFox 附件中文乱名码问题

如果客户端浏览器是早期的火狐浏览器。 那么我们需要对中文名进行 BASE64 的编码操作。如果是新版的则用 URL 编码器解决。

这时候需要把请求头 Content-Disposition: attachment; filename=中文名 编码成为：Content-Disposition: attachment; filename==?charset?B?xxxxx?=

=?charset?B?xxxxx?= 现在我们对这段内容进行一下说明。

=? 表示编码开始

charset	表示编码集，一般使用 utf-8
B	大写的 B 表示 Base64
xxxx	进行 Base64 编码后的中文名
?=	编码结束

因为早期火狐使用的是 BASE64 的编解码方式还原响应中的汉字。所以需要使用 BASE64Encoder 类进行编码操作。代码如下：

表 5-5 Base64 解决早期火狐附件中文名乱码问题

```
// 使用下面的格式进行 BASE64 编码后
String str = "attachment; fileName=" + "=?utf-8?B?"
+ new BASE64Encoder().encode("中文.jpg".getBytes("utf-8")) + "?=";
// 设置到响应头中
response.setHeader("Content-Disposition", str);
```

5.2.5 动态处理 FireFox 和 Chrome 中文名乱码问题

代码如下：

表 5-6 使用 User-Agent 请求头动态处理中文名乱码问题

```
String ua = request.getHeader("User-Agent");
// 判断是否是火狐浏览器
if (ua.contains("Firefox")) {
// 使用下面的格式进行 BASE64 编码后
String str = "attachment; fileName=" + "=?utf-8?B?"
+ new BASE64Encoder().encode("中文.jpg".getBytes("utf-8")) + "?=";
// 设置到响应头中
response.setHeader("Content-Disposition", str);
} else {
// 把中文名进行 UTF-8 编码操作。
String str = "attachment; fileName=" + URLEncoder.encode("中文.jpg",
"UTF-8");
// 然后把编码后的字符串设置到响应头中
response.setHeader("Content-Disposition", str);
}
```

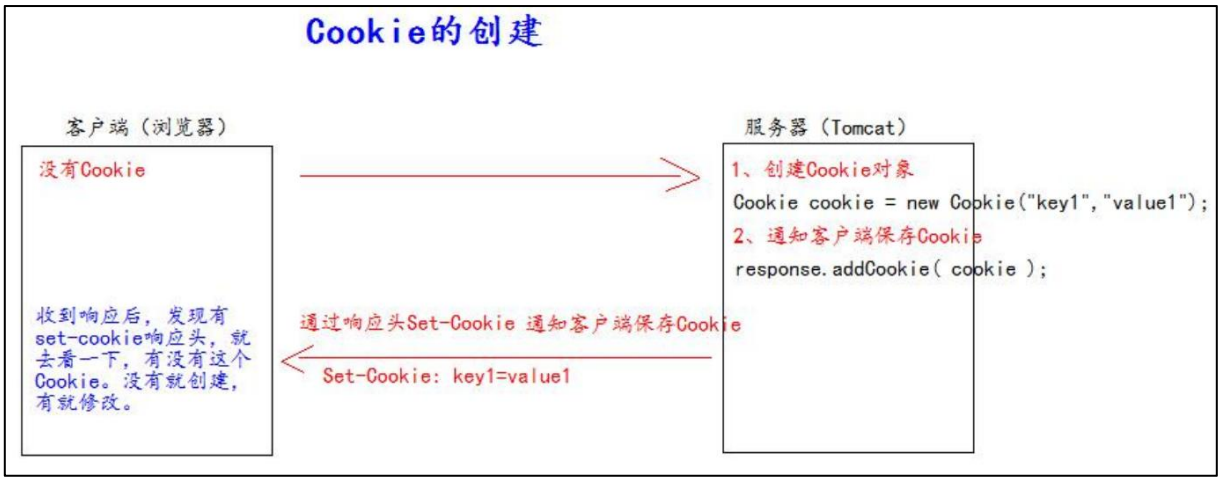
第 6 章 Cookie

6.1 Cookie 基本介绍

1. Cookie 是服务器通知客户端保存键值对的一种技术。
2. 客户端有了 Cookie 后，每次请求都会将其发送给服务器。
3. 每个 Cookie 的大小不能超过 4kb。

6.2 创建 Cookie

图 6-1 Cookie 的创建



示例代码：

表 6-1 创建 Cookie 示例代码

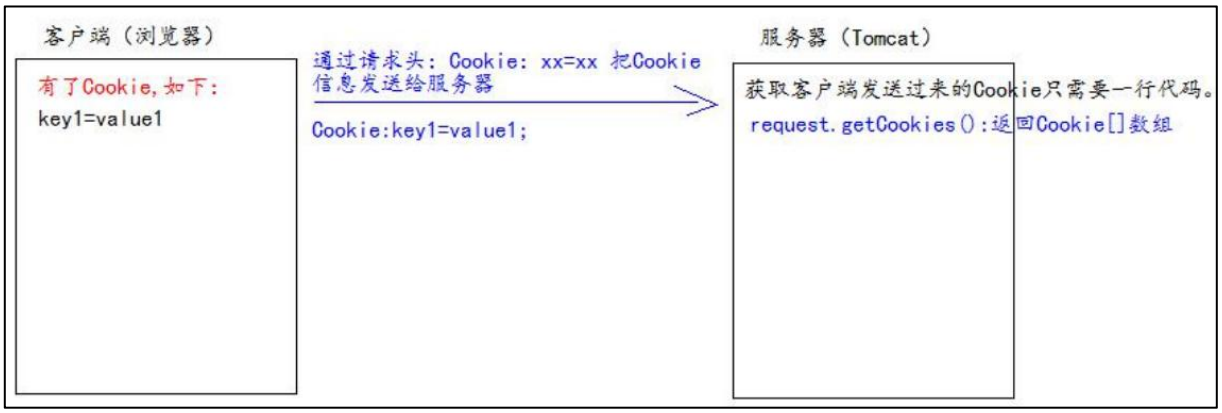
```
package com.lxh.cookie;

import com.lxh.BaseServlet;

import javax.servlet.annotation.WebServlet;
import javax.servlet.http.Cookie;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.io.PrintWriter;

@WebServlet(name = "CookieServlet", value = {"/cookie_servlet"})
public class CookieServlet extends BaseServlet {

    // 创建 cookie
    public void createCookie(HttpServletRequest req, HttpServletResponse resp) {
        Cookie cookie = new Cookie("username", "lxh");
        resp.addCookie(cookie);
    }

}
```

6.3 获取 Cookie

服务器获取客户端的 Cookie 只需要一行代码：request.getCookies():Cookie[]

注意，getCookies()方法返回的是 cookie 数组，我们要想拿到指定 key 的 cookie，就得逐个遍历 cookie 然后获取其 key 与目标 key 比较

图 6-2



示例代码：

表 6-2 CookieUtils.java

```
package com.lxh.cookie;

import com.lxh.BaseServlet;
import com.lxh.util.CookieUtils;

import javax.servlet.annotation.WebServlet;
import javax.servlet.http.Cookie;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.io.PrintWriter;

@WebServlet(name = "CookieServlet", value = {"/cookie_servlet"})
public class CookieServlet extends BaseServlet {

    // 创建 cookie
    public void createCookie(HttpServletRequest req, HttpServletResponse resp) {
        Cookie cookie = new Cookie("username", "lxh");
        resp.addCookie(cookie);
    }

    // 获取 cookie
    public void getCookie(HttpServletRequest req, HttpServletResponse resp) throws IOException {
        resp.setContentType("text/html; charset=utf-8");
        PrintWriter writer = resp.getWriter();
        Cookie[] cookies = req.getCookies();
        Cookie cookie = CookieUtils.getCookie("username", cookies);
        if (null == cookie) {
            System.out.println("cookie 不存在");
        } else {
            System.out.println(cookie.getName() + " - " +    cookie.getValue());
        }
    }

}
```

表 6-3 获取指定 key 的 cookie 示例代码

```
package com.lxh.cookie;

import com.lxh.BaseServlet;
import com.lxh.util.CookieUtils;

import javax.servlet.annotation.WebServlet;
import javax.servlet.http.Cookie;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.io.PrintWriter;

@WebServlet(name = "CookieServlet", value = {"/cookie_servlet"})
public class CookieServlet extends BaseServlet {

    // 获取 cookie
    public void getCookie(HttpServletRequest req, HttpServletResponse resp) throws IOException {
        resp.setContentType("text/html; charset=utf-8");
        PrintWriter writer = resp.getWriter();
        Cookie[] cookies = req.getCookies();
        Cookie cookie = CookieUtils.getCookie("username", cookies);
        if (null == cookie) {
            System.out.println("cookie 不存在");
        } else {
            System.out.println(cookie.getName() + " - " +    cookie.getValue());
        }
    }

}
```

6.4 修改 Cookie 值

方案一：

1、先创建一个要修改的同名（指的就是 key）的 Cookie 对象

2、在构造器，同时赋于新的 Cookie 值。

3、调用 response.addCookie(Cookie);

表 6-4 方案一修改 cookie 值核心代码

```
Cookie cookie = new Cookie("username", "zs");
resp.addCookie(cookie);
```

方案二：

1、先查找到需要修改的 Cookie 对象

2、调用 setValue()方法赋于新的 Cookie 值。

3、调用 response.addCookie()通知客户端保存修改

表 6-5 方案二修改 cookie 值核心代码

```
Cookie cookie = CookieUtils.getCookie("username", req.getCookies());
cookie.setValue("lisi");
resp.addCookie(cookie);
```

6.5 Cookie 的生命控制

通过 setMaxAge(int expiry)方法来设置 cookie 的存活时间

setMaxAge(int expiry)

- 正数，表示在指定的秒数后过期
- 负数，表示浏览器一关，Cookie 就会被删除（默认值是-1）
- 零，表示马上删除 Cookie

表 6-6 修改 cookie 存活时间示例代码

```
package com.lxh.cookie;

import com.lxh.BaseServlet;
import com.lxh.util.CookieUtils;

import javax.servlet.annotation.WebServlet;
import javax.servlet.http.Cookie;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.io.PrintWriter;

@WebServlet(name = "CookieServlet", value = {"/cookie_servlet"})
public class CookieServlet extends BaseServlet {

        // 创建一个指定存活时间的 cookie
    public void cookieLife3600(HttpServletRequest req, HttpServletResponse resp) {
        Cookie cookie = new Cookie("password", "123456");
        // 指定 cookie 存活时间为 3600S
        cookie.setMaxAge(3600);
        resp.addCookie(cookie);
    }

    // 创建一个默认存活时间的 cookie
    public void cookieLifeDefault(HttpServletRequest req, HttpServletResponse resp) {
        Cookie cookie = new Cookie("nick", "一直沉默的猪");
        resp.addCookie(cookie);
    }

    // 立即删除一个指定的 cookie
    public void deleteNow(HttpServletRequest req, HttpServletResponse resp) {
        Cookie cookie = CookieUtils.getCookie("password", req.getCookies());
        if (null != cookie) {
            cookie.setMaxAge(0);
            resp.addCookie(cookie);
            System.out.println("成功删除 cookie！ ");
        }
    }
}
```

6.6 Cookie 的有效路径 Path 设置

Cookie 的 path 属性可以有效的过滤哪些 Cookie 可以发送给服务器。哪些不发。path 属性是通过请求的地址来进行有效的过滤。

例：CookieA path=/工程路径 CookieB path=/工程路径/abc

1) 若请求地址为 http://ip:port/工程路径/a.html

CookieA 发送，CookieB 不发送

2) 若请求地址为 http://ip:port/工程路径/abc/a.html

CookieA 和 CookieB 均会发送

表 6-7 path 属性示例代码

```
package com.lxh.cookie;

import com.lxh.BaseServlet;
import com.lxh.util.CookieUtils;

import javax.servlet.annotation.WebServlet;
import javax.servlet.http.Cookie;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.io.PrintWriter;

@WebServlet(name = "CookieServlet", value = {"/cookie_servlet"})
public class CookieServlet extends BaseServlet {

    // 测试 path 属性
    public void path(HttpServletRequest req, HttpServletResponse resp) {
        Cookie cookie = new Cookie("path", "test_path");
        // 设置 cookie 的 path 属性  -  /工程路径/abc
        cookie.setPath(req.getContextPath() + "/abc");
        resp.addCookie(cookie);
    }

}
```

6.7 Cookie 实现免用户名登录

图 6-3 流程图

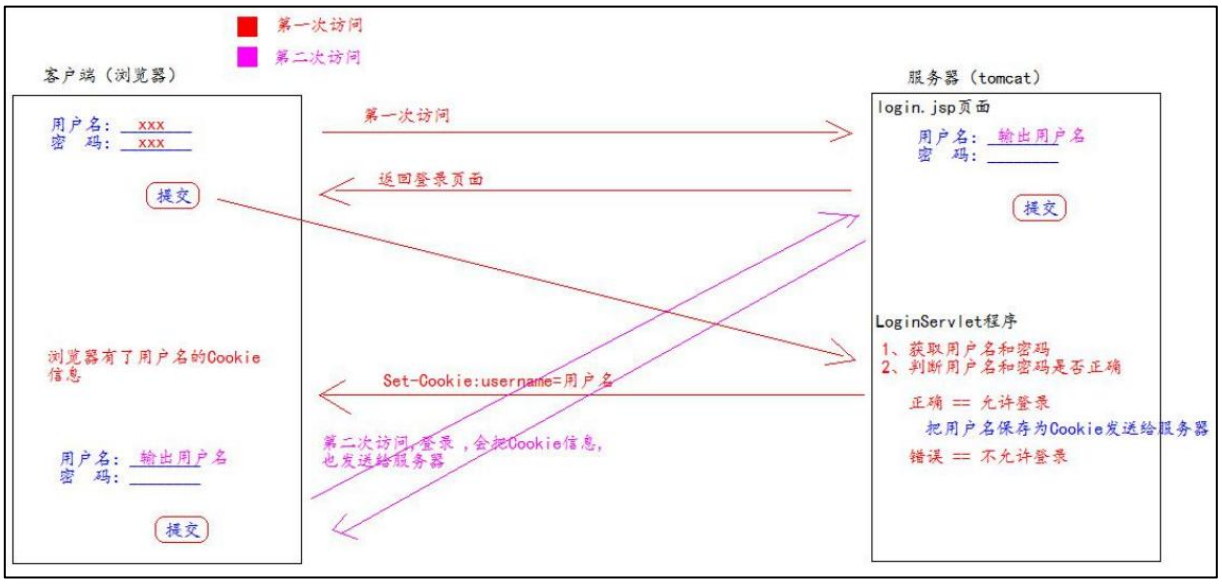


表 6-8 login.jsp

```jsp
<%--
  Created by IntelliJ IDEA.
  User: Administrator
  Date: 2020/2/10
  Time: 11:34
  To change this template use File | Settings | File Templates.
--%>
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>Title</title>
</head>
<body>
    <form action="/d06_cookie_session/login_servlet" method="post">
        <input type="hidden" name="action" value="login">
        用户名：<input type="text" name="username" value="${cookie.username.value}"> <br>
        密码：<input type="password" name="password"> <br>
        <input type="submit" value="登录">
    </form>
</body>
</html>
```

表 6-9 LoginServlet.java

```java
package com.lxh.cookie;

import com.lxh.BaseServlet;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.Cookie;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
@WebServlet(name = "LoginServlet", value = {"/login_servlet"})
public class LoginServlet extends BaseServlet {

    public void login(HttpServletRequest req, HttpServletResponse resp) throws ServletException,
IOException {
        String username = req.getParameter("username");
        String password = req.getParameter("password");
        if ("lxh".equals(username) && "123456".equals(password)) {
            // 登录成功
            Cookie cookie = new Cookie("username", username);
            // 设置存活时间为一星期
            cookie.setMaxAge(60 * 60 * 24 * 7);
            resp.addCookie(cookie);
        }
    }
}
```

第 7 章 Session

7.1 Session 基本介绍

1. Session 就一个接口（HttpSession）。
2. Session 就是会话。它是用来维护一个客户端和服务器之间关联的一种技术。
3. 每个客户端都有自己的一个 Session 会话。
4. Session 会话中，我们经常用来保存用户登录之后的信息。

Session 的存取和 Cookie 雷同

7.2 为什么需要 Session

Http 是无状态的， 无状态指的是：服务器无法判断这两次请求是同一个客户端发过来的，还是不同的客户端发过来的；无状态带来的现实问题：第一次请求是添加商品到购物车，第二次请求是结账；如果这两次请求服务器无法区分是同一个用户的，那么就会导致混乱。

所以我们需要通过会话跟踪技术来解决无状态的问题。

7.3 创建和获取 Session

request.getSession()

1) 第一次调用是创建 Session 会话
2) 之后调用都是获取前面创建好的 Session 会话对象。
3) request.getSession(true) -> 效果和不带参数相同
4) request.getSession(false) -> 获取当前会话，没有则返回 null，不会创建新的

isNew() 判断 Session 是不是刚创建出来的（新的）

1) true 表示刚创建
2) false 表示获取之前创建

每个会话都有一个身份证号。也就是 ID 值。而且这个 ID 是唯一的。

getId() 得到 Session 的会话 id 值。

7.4 Session 域存取数据

void session.setAttribute(k,v)

Object session.getAttribute(k)

void removeAttribute(k)

表 7-1 Session 域存储数据示例代码

```
package com.lxh.session;

import com.lxh.BaseServlet;

import javax.servlet.annotation.WebServlet;
```

```
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
import java.io.IOException;

@WebServlet(name = "SessionServlet", value = {"/session_servlet"})
public class SessionServlet extends BaseServlet {

    // 获取或创建 session
    public void createOrGetSession(HttpServletRequest request, HttpServletResponse response)
throws IOException {
        response.setContentType("text/html; charset=utf-8");
        HttpSession session = request.getSession();
        response.getWriter().write("SessionId ===>" + session.getId());
        response.getWriter().write("\n 是否是新创建的： " + session.isNew());
    }

    // 保存数据到 session 域
    public void setSessionData(HttpServletRequest request, HttpServletResponse response) throws
IOException {
        response.setContentType("text/html; charset=utf-8");
        request.getSession().setAttribute("nick", "兴之所至");
        response.getWriter().write("session 保存了数据~~");
    }

    // 获取 session 域中的数据
    public void getSessionData(HttpServletRequest request, HttpServletResponse response) throws
IOException {
        response.setContentType("text/html; charset=utf-8");
        Object nick = request.getSession().getAttribute("nick");
        response.getWriter().write("session 域数据： " + nick);
    }
}
```

7.5 Session 的生命控制

public void setMaxInactiveInterval(int interval) 设置 Session 的超时时间（以秒为单位），超过指定的时长，Session 就会被销毁。

- 值为正数的时候，设定 Session 的超时时长。
- 负数表示永不超时（极少使用）

public int getMaxInactiveInterval()获取 Session 的超时时间

public void invalidate() 让当前 Session 会话马上超时无效。

Session 默认的超时时间长为 30 分钟。 因为在 Tomcat 服务器的配置文件 web.xml 中默认有以下的配置，它就表示配置了当前 Tomcat 服务器下所有的 Session 超时配置默认时长为：30 分钟

如果说。你希望你的 web 工程，默认的 Session 的超时时长为其他时长。你可以在你自己的 web.xml 配置文件中做 以上相同的配置。就可以修改你的 web 工程所有 Sees sion 的默认超时时长。

```
<session-config>
```

```
    <session-timeout>20</session-timeout>
</session-config>
```

如果你想只修改个别 Session 的超时时长。使用 setMaxInactiveInterval(int interval)来进行单独的设置。

Session 的超时指的是客户端两次请求的最大时间间隔。

7.6 浏览器和 Session 之间关联的技术内幕

第 8 章 Listener 监听器

第 9 章 Filter 过滤器

9.1 Filter 基本介绍

1. Filter 过滤器它是 JavaWeb 的三大组件之一。三大组件分别是：Servlet 程序、Listener 监听器、Filter 过滤器
2. Filter 过滤器它是 JavaEE 的规范。也就是接口。
3. Filter 过滤器它的作用是：拦截请求，过滤响应。

拦截请求常见的应用场景有：权限检查、日记操作、事务管理等…

9.2 Filter 初体验

表 9-1 DemoServlet.java

```
package com.lxh.servlet;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;

public class DemoServlet extends HttpServlet {

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
        System.out.println("DemoServlet...");
    }
}
```

表 9-2 DemoFilter.java

```
package com.lxh.filter;

import javax.servlet.*;
import java.io.IOException;

public class DemoFilter implements Filter {

    @Override
    public void init(FilterConfig filterConfig) throws ServletException {

    }

    @Override
    public void doFilter(ServletRequest servletRequest, ServletResponse servletResponse, FilterChain
filterChain) throws IOException, ServletException {
        System.out.println("DemoFilter...");
        // 放行
        filterChain.doFilter(servletRequest, servletResponse);
    }

    @Override
    public void destroy() {

    }
```

```
}
```

表 9-3 web.xml

```
<?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>
        <servlet-name>DemoServlet</servlet-name>
        <servlet-class>com.lxh.servlet.DemoServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>DemoServlet</servlet-name>
        <url-pattern>/demo.do</url-pattern>
    </servlet-mapping>

    <!-- 配置一个 filter -->
    <filter>
        <filter-name>DemoFilter</filter-name>
        <filter-class>com.lxh.filter.DemoFilter</filter-class>
    </filter>
    <filter-mapping>
        <filter-name>DemoFilter</filter-name>
        <url-pattern>*.do</url-pattern>
    </filter-mapping>

</web-app>
```

此时我们访问 http://ip:port/工程名/demo.do，会发现控制台输出：

```
DemoFilter...
DemoServlet...
```

由控制台输出结果可以看出，程序是先执行了 DemoFilter 中的 doFilter()方法，然后才执行 DemoServlet 中的 doGet()方法的，所以过滤器就已经配置成功了。

Filter 过滤器的使用步骤：

1、编写一个类去实现 Filter 接口

2、实现过滤方法 doFilter()

3、到 web.xml 中去配置 Filter 的拦截路径

9.3 Filter 的生命周期

Filter 的生命周期包含几个方法：

1、构造器方法

2、init() 初始化方法

第 1，2 步，在 web 工程启动的时候执行（Filter 已经创建）

3、doFilter(request, response, chain) 过滤方法

第 3 步，每次拦截到请求，就会执行

4、destroy() 销毁

第 4 步，停止 web 工程的时候，就会执行（停止 web 工程，也会销毁 Filter 过滤器）

9.4 Filter 使用案例-用户登录验证

现在我们有这样一个需求：用户如果登录了系统则正常使用，用户如果没有登录而去访问其他页面，则会跳转到登录页面，要求用户进行登录。

表 9-4 login.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<form action="/d07_filter/login.do" method="post">
    用户名：<input type="text" name="username"><br>
    密码：<input type="text" name="password"><br>
    <input type="submit">
</form>
</body>
</html>
```

表 9-5 index.html

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

表 9-6 LoginServlet.java

```
package com.lxh.servlet;

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 java.io.IOException;

@WebServlet({"/login.do"})
public class LoginServlet extends HttpServlet {

    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
        // 获取用户名和密码
        String username = req.getParameter("username").toString();
        String password = req.getParameter("password").toString();
```

```
            // 判断用户名密码是否正确
            if ("lxh".equals(username) && "123456".equals(password)) {
                // 登录成功
                System.out.println("登录成功！！！");
                // 将用户名保存到 session 中
                req.getSession().setAttribute("username", username);
                // 重定向到 index.html
                resp.sendRedirect(req.getContextPath() + "/index.html");
            } else {
                // 登录失败
                System.out.println("登录失败！！！");
                resp.sendRedirect(req.getContextPath() + "/login.html");
            }
        }
}
```

LoginServlet 程序用于处理登录功能

表 9-7 LoginFilter

```
package com.lxh.filter;

import javax.servlet.*;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.util.ArrayList;

public class LoginFilter implements Filter {
    @Override
    public void init(FilterConfig filterConfig) throws ServletException {

    }

    @Override
    public void doFilter(ServletRequest servletRequest, ServletResponse servletResponse, FilterChain
filterChain) throws IOException, ServletException {
        HttpServletRequest request = (HttpServletRequest) servletRequest;
        HttpServletResponse response = (HttpServletResponse) servletResponse;
        Object username = request.getSession().getAttribute("username");

        // 获取请求资源路径
        String url = request.getServletPath();
        System.out.println("url = " + url);

        // 允许为登录也能访问的资源 - 共享资源
        ArrayList<String> allowList = new ArrayList<>(10);
        allowList.add("/login.html");
        allowList.add("/login.do");
        allowList.add(".css");
        allowList.add(".js");

        if (allowList.contains(url)) {
            // 访问共享资源
            // 放行
            filterChain.doFilter(servletRequest, servletResponse);
        } else if (null != username) {
            // 已登录
            // 发行
```

```
                filterChain.doFilter(servletRequest,servletResponse);
            } else {
                // 未登录
                response.sendRedirect(request.getContextPath() + "/login.html");
            }
        }

        @Override
        public void destroy() {

        }
}
```

LoginFilter 程序用于过滤用户是否登录，如果登录了，则正常使用；如果用户访问的是 css 文件、js 文件或者是登录页面 login.html 则放行资源；如果用户未登录则重定向到登录页 login.html

表 9-8 web.xml 配置文件

```
<?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">

    <!-- 配置一个 filter -->
    <filter>
        <filter-name>LoginFilter</filter-name>
        <filter-class>com.lxh.filter.LoginFilter</filter-class>
    </filter>
    <!-- 配置 filter 的拦截路径 -->
    <filter-mapping>
        <filter-name>LoginFilter</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>

</web-app>
```

运行程序发现，如果我们访问其他 html 页面则会自动跳转到 login.html 页面。

9.5 FilterConfig 类

FilterConfig 类见名知义，它是 Filter 过滤器的配置文件类。Tomcat 每次创建 Filter 的时候，也会同时创建一个 FilterConfig 类，这里包含了 Filter 配置文件的配置信息。

FilterConfig 类的作用是获取 filter 过滤器的配置内容：

1、 获取 Filter 的名称 filter-name 的内容

2、 获取在 Filter 中配置的 init-param 初始化参数

3、 获取 ServletContext 对象

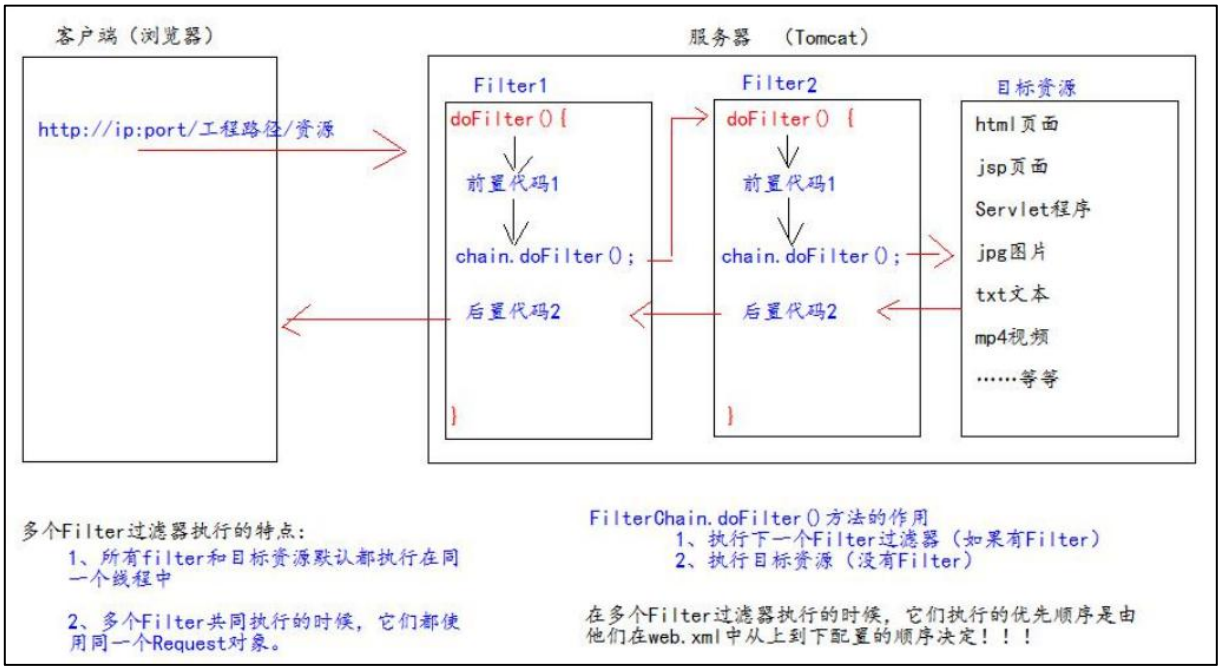
表 9-9 init-param 配置默认参数

```
<filter>
    <filter-name>LoginFilter</filter-name>
```

```
    <filter-class>com.lxh.filter.LoginFilter</filter-class>
    <init-param>
        <param-name>user</param-name>
        <param-value>root</param-value>
    </init-param>
</filter>
```

9.6 Filter 过滤器链

FilterChain 就是过滤器链（多个过滤器如何一起工作）



9.7 Filter 的拦截路径

- 精确匹配

  例：<url-pattern>/target.html</url-pattern>：表示只拦截请求路径为 http://ip:port/target.html 的资源

- 目录匹配

  例：<url-pattern>/admin/*</url-pattern>：表示拦截请求路径为 http://ip:port/admin/* 的资源

- 后缀名匹配

  例：<url-pattern>*.html</url-pattern>：表示请求地址必须以.html 结尾才会拦截到

  例：<url-pattern>*.do</url-pattern>：表示请求地址必须以.do 结尾才会拦截到

  例：<url-pattern>*.action</url-pattern>：表示请求地址必须以.action 结尾才会拦截到

注：Filter 过滤器它只关心请求的地址是否匹配，不关心请求的资源是否存在!!!

9.8 使用 Filter 完成统一异常处理

表 9-10 TransactionFilter.java

```
package com.lxh.filter;
```

```
import javax.servlet.*;
import java.io.IOException;

public class TransactionFilter implements Filter {

    @Override
    public void init(FilterConfig filterConfig) throws ServletException {

    }

    @Override
    public void doFilter(ServletRequest servletRequest, ServletResponse servletResponse, FilterChain
filterChain) throws IOException, ServletException {

        try {
            System.out.println("TransactionFilter...");
            // 事务处理
            // 如果没有发生异常信息则放行然后提交事务
            filterChain.doFilter(servletRequest, servletResponse);
            // 提交事务操作
        } catch(Exception e) {
            // 如果有异常则将异常抛出去给 Tomcat 处理
            e.printStackTrace();
            throw new RuntimeException(e);
            // 事务回滚
        }
    }

    @Override
    public void destroy() {

    }
}
```

此过滤器就用于完成统一异常处理

注：其他地方 try…catch 的时候，一定要再把异常 throw 出去，资源该过滤器才能知晓有异常发生。

表 9-11 error404.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
    <p>error 404</p>
</body>
</html>
```

表 9-12 error500.html

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

```
    <title>Title</title>
</head>
<body>
    <p>error 500</p>
</body>
</html>
```

表 9-13 web.xml 配置文件

```
<?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">

    <!-- 配置一个 filter -->
    <filter>
        <filter-name>LoginFilter</filter-name>
        <filter-class>com.lxh.filter.LoginFilter</filter-class>
        <init-param>
            <param-name>user</param-name>
            <param-value>root</param-value>
        </init-param>
    </filter>
    <!-- 配置 filter 的拦截路径 -->
    <filter-mapping>
        <filter-name>LoginFilter</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>

    <filter>
        <filter-name>TransactionFilter</filter-name>
        <filter-class>com.lxh.filter.TransactionFilter</filter-class>
    </filter>
    <filter-mapping>
        <filter-name>TransactionFilter</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>

    <!-- error-page 标签用于配置单服务器内部发生错误后自动跳转的页面 -->
    <error-page>
        <!-- error-code 是错误类型 -->
        <error-code>404</error-code>
        <!-- location 为发生错误时要跳转为的页面-->
        <location>/error404.html</location>
    </error-page>
    <error-page>
        <error-code>500</error-code>
        <location>/error500.html</location>
    </error-page>
</web-app>
```

表 9-14 LoginServlet.java

```
package com.lxh.servlet;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
```

```java
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;

@WebServlet({"/login.do"})
public class LoginServlet extends HttpServlet {

    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
        try {
            // 获取用户名和密码
            String username = req.getParameter("username").toString();
            String password = req.getParameter("password").toString();
            // 判断用户名密码是否正确
            if ("lxh".equals(username) && "123456".equals(password)) {
                // 登录成功
                System.out.println("登录成功！！！");
                // 将用户名保存到 session 中
                req.getSession().setAttribute("username", username);
                // 重定向到 index.html
                resp.sendRedirect(req.getContextPath() + "/index.html");
            } else {
                // 登录失败
                System.out.println("登录失败！！！");
                resp.sendRedirect(req.getContextPath() + "/login.html");
            }
            int i = 5/0;
        } catch (IOException e) {
            e.printStackTrace();
            throw new RuntimeException(e);
        }
    }
}
```

我在 LoginServlet 程序中加入了除 0 操作，发生 500 错误，所以程序会跳转到 error 500.html 页面