Web核心

什么是 JavaWeb ?

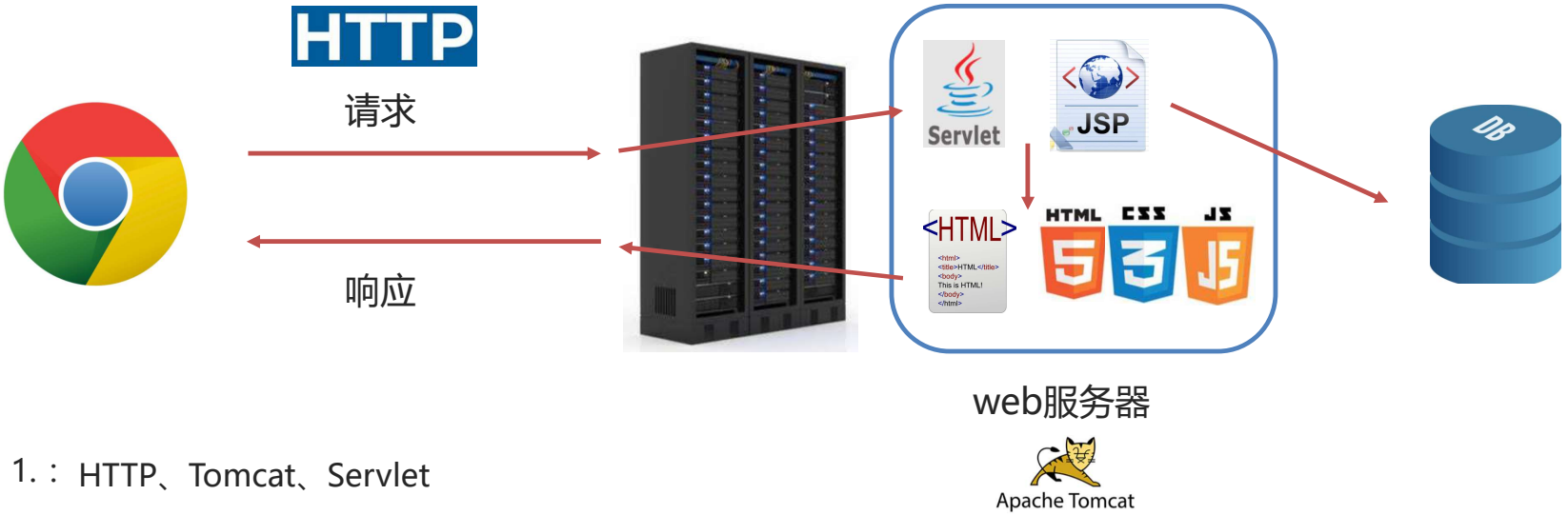
- Web：全球广域网，也称为万维网(www)，能够通过浏览器访问的网站
- JavaWeb： 是用 Java技术来解决相关web互联网领域的技术栈

JavaWeb 技术栈

- B/S 架构：Browser/Server，浏览器/服务器 架构模式，它的特点是，客户端只需要浏览器，应用程序的逻辑和数据都存储在服务器端。浏览器只需要请求服务器，获取Web资源，服务器把Web资源发送给浏览器即可
  - 好处：易于维护升级：服务器端升级后，客户端无需任何部署就可以使用到新的版本

- 静态资源：HTML、CSS、JavaScript、图片等。负责页面展现
- 动态资源：Servlet、JSP 等。负责逻辑处理
- 数据库：负责存储数据
- HTTP协议：定义通信规则
- Web服务器：负责解析 HTTP 协议，解析请求数据，并发送响应数据

课程安排



- 1.： HTTP、Tomcat、Servlet
- 2.： Request(请求)、Response(响应)
- 3.： JSP、会话技术(Cookie、Session)
- 4.： Filter（过滤器）、Listener（监听器）
- 5.： Ajax、Vue、ElementUI
- 6.： 综合案例

目录

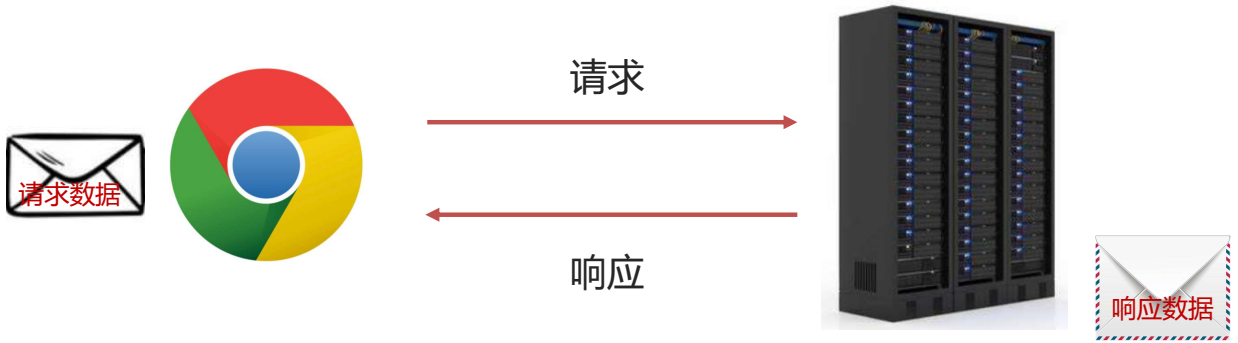
Contents

目录

Contents

HTTP

- 概念：HyperText Transfer Protocol，超文本传输协议，规定了浏览器和服务器之间数据传输的规则



- HTTP 协议特点：
  1. 基于TCP协议：面向连接，安全
  2. 基于请求-响应模型的：一次请求对应一次响应
  3. HTTP协议是无状态的协议：对于事务处理没有记忆能力。每次请求-响应都是独立的。
     - 缺点：多次请求间不能共享数据。Java中使用会话技术（Cookie、Session）来解决这个问题
     - 优点：速度快

HTTP-请求数据格式

- 请求数据分为3部分：
  1. 请求行：请求数据的第一行。其中GET表示请求方式，/ 表示请求资源路径，HTTP/1.1表示协议版本
  2. 请求头：第二行开始，格式为key：value形式。
  3. 请求体： POST请求的最后一部分，存放请求参数

```
GET / HTTP/1.1
Host: www.itcast.cn
Connection: keep-alive
User-Agent: Mozilla/5.0 Chrome/91.0.4472.106
...
```

- 常见的HTTP 请求头：
  - Host: 表示请求的主机名
  - User-Agent: 浏览器版本，例如Chrome浏览器的标识类似Mozilla/5.0 ... Chrome/79，IE浏览器的标识类似Mozilla/5.0 (Windows NT ...) like Gecko；
  - Accept：表示浏览器能接收的资源类型，如text/*，image/*或者*/*表示所有；
  - Accept-Language：表示浏览器偏好的语言，服务器可以据此返回不同语言的网页；
  - Accept-Encoding：表示浏览器可以支持的压缩类型，例如gzip, deflate等。

- GET请求和 POST请求区别：
  1. GET请求请求参数在请求行中，没有请求体。POST请求请求参数在请求体中
  2. GET请求请求参数大小有限制，POST没有

```
POST / HTTP/1.1
Host: www.itcast.cn
Connection: keep-alive
Cache-Control: max-age=0 Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 Chrome/91.0.4472.106

username=superbaby&password=123456
```

HTTP

- 概念：HyperText Transfer Protocol，超文本传输协议，规定了浏览器和服务器之间数据传输的规则。

- HTTP 协议特点：
  1. 基于TCP协议：面向连接，安全
  2. 基于请求-响应模型的：一次请求对应一次响应
  3. HTTP协议是无状态的协议：对于事务处理没有记忆能力。每次请求响应都是独立的。
     - 缺点：多次请求间不能共享数据。Java中使用会话技术（Cookie、Session）来解决这个问题
     - 优点：速度快

HTTP-响应数据格式

- 响应数据分为3部分：
  1. 响应行：响应数据的第一行。其中HTTP/1.1表示协议版本，200表示响应状态码，OK表示状态码描述
  2. 响应头：第二行开始，格式为key：value形式
  3. 响应体： 最后一部分。存放响应数据

状态码分类	说明
1xx	**响应中**——临时状态码，表示请求已经接受，告诉客户端应该继续请求或者如果它已经完成则忽略它
2xx	**成功**——表示请求已经被成功接收，处理已完成
3xx	**重定向**——重定向到其它地方：它让客户端再发起一个请求以完成整个处理。
4xx	**客户端错误**——处理发生错误，责任在客户端，如：客户端的请求一个不存在的资源，客户端未被授权，禁止访问等
5xx	**服务器端错误**——处理发生错误，责任在服务端，如：服务端抛出异常，路由出错，HTTP版本不支持等

```
HTTP/1.1 200 OK
Server: Tengine
Content-Type: text/html
Transfer-Encoding: chunked…

<html>
<head>
   <title></title>
</head>
<body></body>
</html>
```

- 常见的HTTP 响应头：
  - Content-Type：表示该响应内容的类型，例如text/html，image/jpeg；
  - Content-Length：表示该响应内容的长度（字节数）；
  - Content-Encoding：表示该响应压缩算法，例如gzip；
  - Cache-Control：指示客户端应如何缓存，例如max-age=300表示可以最多缓存300秒

HTTP

- 概念：HyperText Transfer Protocol，超文本传输协议，规定了浏览器和服务器之间数据传输的规则。

目录

Contents

Web 服务器

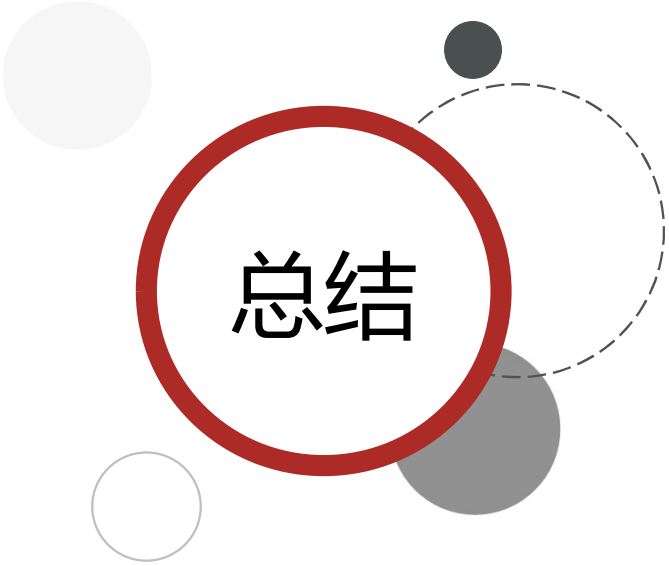
- Web服务器是一个应用程序（软件），对 HTTP协议的操作进行封装，使得程序员不必直接对协议进行操作，让Web开发更加便捷。主要功能是“提供网上信息浏览服务”

Tomcat

- 简介
- 基本使用：下载、安装、卸载、启动、关闭、配置、部署项目
- IDEA中创建 Maven Web项目
- IDEA中使用 Tomcat

Tomcat

- 概念： Tomcat是Apache 软件基金会一个核心项目，是一个开源免费的轻量级Web服务器，支持Servlet/JSP少量JavaEE规范。
- JavaEE：Java Enterprise Edition，Java企业版。指Java企业级开发的技术规范总和。包含13项技术规范：JDBC、JNDI、EJB、RMI、JSP、Servlet、XML、JMS、Java IDL、JTS、JTA、JavaMail、JAF
- Tomcat 也被称为 Web容器、Servlet容器。Servlet 需要依赖于 Tomcat才能运行
- 官网：https://tomcat.apache.org/



1. Web 服务器作用?
   - 封装HTTP协议操作，简化开发
   - 可以将web项目部署到服务器中，对外提供网上浏览服务
2. Tomcat是一个轻量级的Web服务器，支持Servlet/JSP少量JavaEE规范，也称为Web容器，Servlet容器

Tomcat

- 简介
- 基本使用：下载、安装、卸载、启动、关闭、配置、部署项目
- IDEA中创建 Maven Web项目
- IDEA中使用 Tomcat

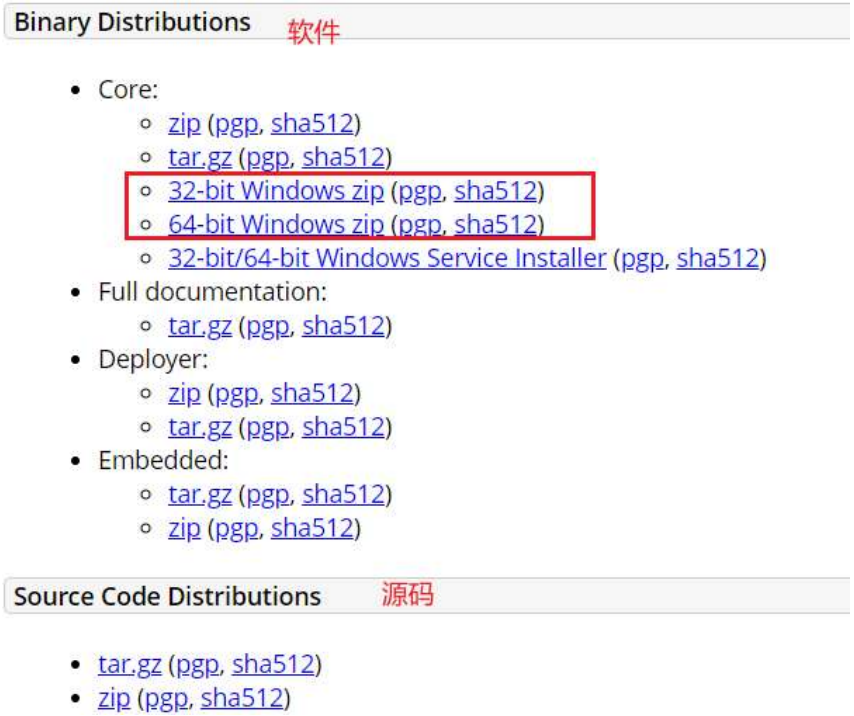
Tomcat – 基本使用

- 下载：官网下载
- 安装：绿色版，直接解压即可
- 卸载：直接删除目录即可
- 启动：双击：bin\startup.bat

- 控制台中文乱码：修改conf/ logging.properties

```
java.util.logging.ConsoleHandler.encoding = UTF-8 GBK
```

- 关闭：
  1. 直接×掉运行窗口：强制关闭
  2. bin\shutdown.bat：正常关闭
  3. Ctrl+C：正常关闭



Tomcat

- 简介
- 基本使用：下载、安装、卸载、启动、关闭、配置、部署项目
- IDEA中创建 Maven Web项目
- IDEA中使用 Tomcat

Tomcat – 基本使用

- 配置：
  1. 修改启动端口号：conf/server.xml

```
<Connector port="8080" protocol="HTTP/1.1"
           connectionTimeout="20000"
           redirectPort="8443" />
```

  - 注：HTTP协议默认端口号为80，如果将Tomcat端口号改为80，则将来访问Tomcat时，将不用输入端口号

- 启动时可能出现的问题：
  1. 端口号冲突：找到对应程序，将其关闭掉

```
        at org.apache.catalina.startup.Bootstrap.load(Bootstrap.java:
        at org.apache.catalina.startup.Bootstrap.main(Bootstrap.java:
Caused by: java.net.BindException: Address already in use: bind
        at sun.nio.ch.Net.bind0(Native Method)
        at sun.nio.ch.Net.bind(Net.java:433)
```

  2. 启动窗口一闪而过：检查JAVA_HOME环境变量是否正确配置

Tomcat – 部署项目

- Tomcat 部署项目：
  - 将项目放置到 webapps 目录下， 即部署完成

- 一般 JavaWeb项目会被打成war包，然后将 war包放到 webapps目录下，Tomcat会自动解压缩 war文件

Tomcat

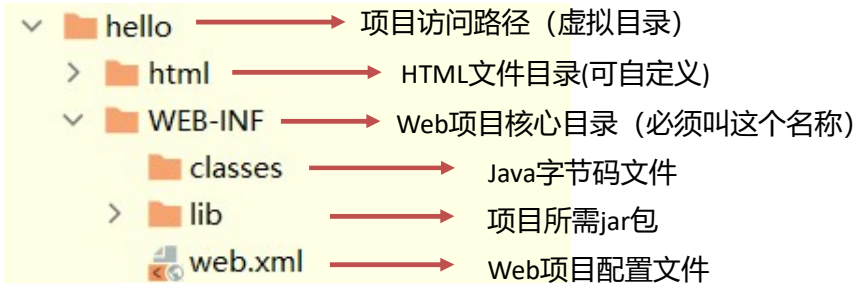
- 简介
- 基本使用：安装、卸载、启动、关闭、配置、部署项目
- IDEA中创建 Maven Web项目
- IDEA中使用 Tomcat

IDEA中创建 Maven Web项目

- Web项目结构：
  - Maven Web项目结构：开发中的项目

package

  - 部署的JavaWeb项目结构：开发完成，可以部署的项目

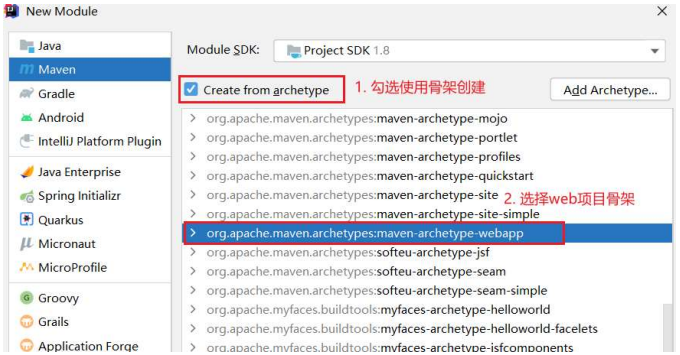


- 编译后的Java字节码文件和resources的资源文件，放到WEB-INF下的classes目录下
- pom.xml中依赖坐标对应的jar包，放入WEB-INF下的lib目录下

IDEA中创建 Maven Web项目

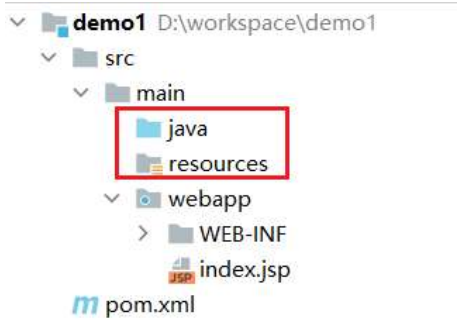
- 使用骨架
  - 骨架：项目模板

1. 选择web项目骨架，创建项目

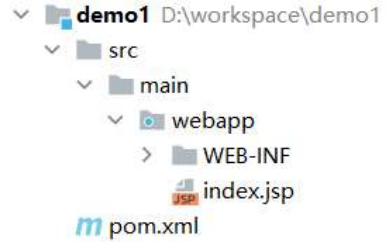


2. 删除pom.xml中多余的坐标

```
<build>
  <finalName>demo1</finalName>
  <pluginManagement><!-- lock down pl
    <plugins>
      <plugin>
```



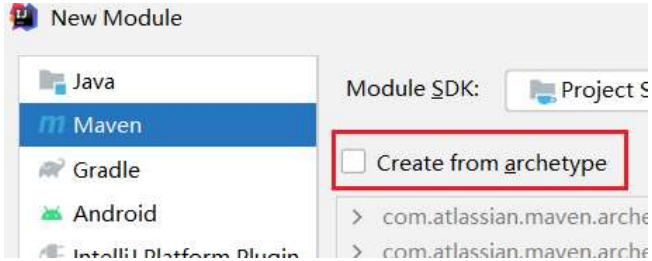
3. 补齐缺失的目录结构



IDEA中创建 Maven Web项目

- 不使用骨架

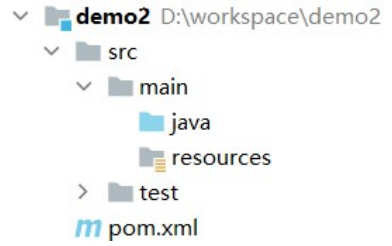
1. 选择web项目骨架，创建项目

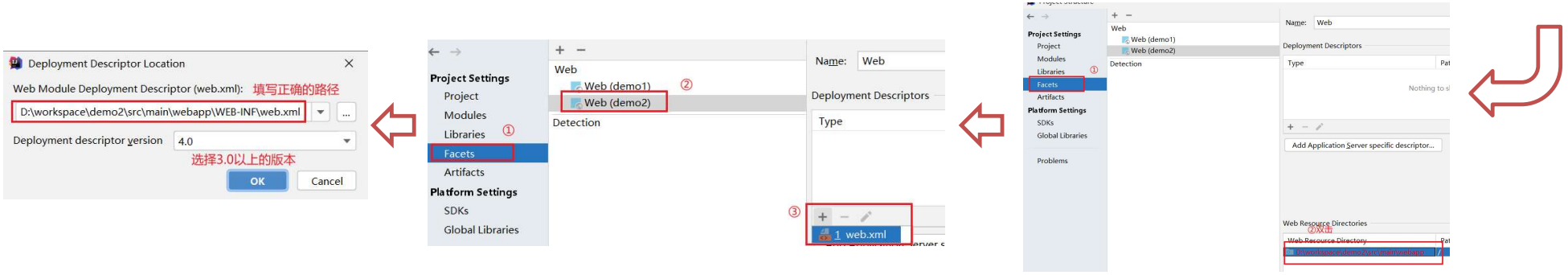


2. pom.xml中添加打包方式为war

```
<groupId>com.itheima</groupId>
<artifactId>demo2</artifactId>
<version>1.0-SNAPSHOT</version>
<packaging>war</packaging>
```

3. 补齐缺失的目录结构：webapp



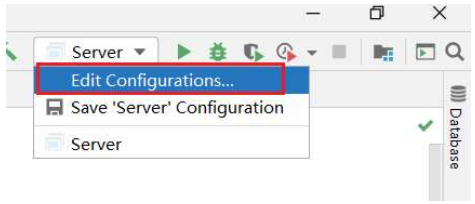


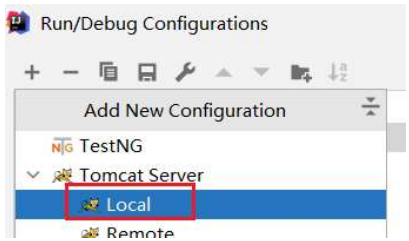
Tomcat

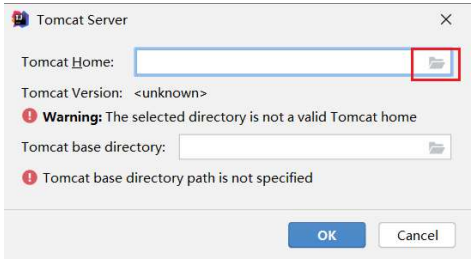
- 简介
- 基本使用：安装、卸载、启动、关闭、配置、部署项目
- IDEA中创建 Maven Web项目
- IDEA中使用 Tomcat

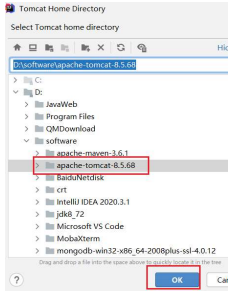
IDEA中使用 Tomcat – 集成本地 Tomcat

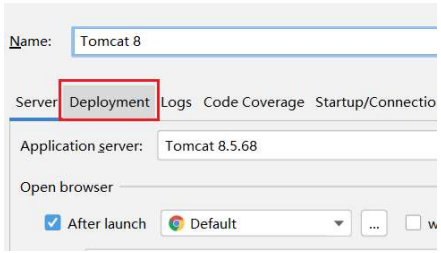
- 将本地Tomcat 集成到Idea中，然后进行项目部署即可

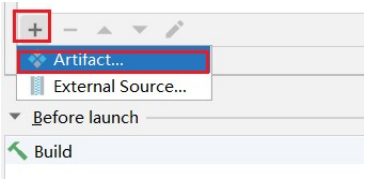


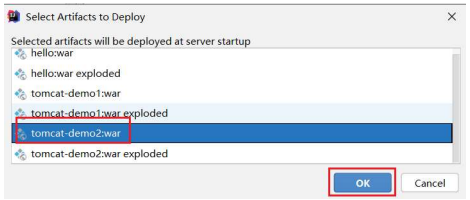


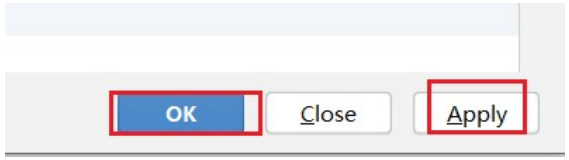


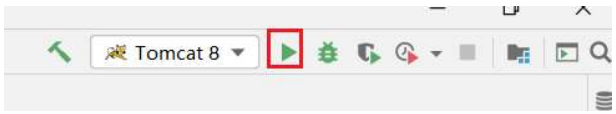












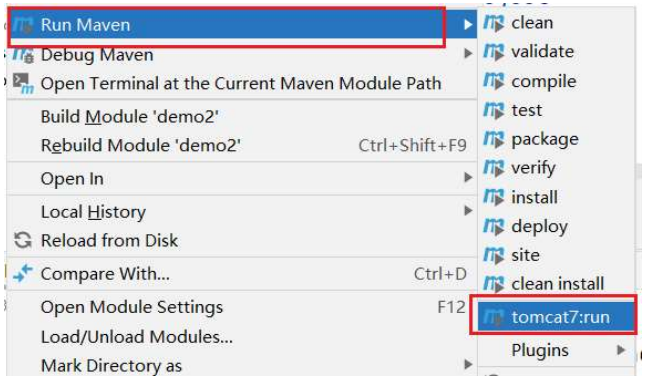
IDEA中使用 Tomcat – Tomcat Maven 插件

1. pom.xml 添加 Tomcat插件

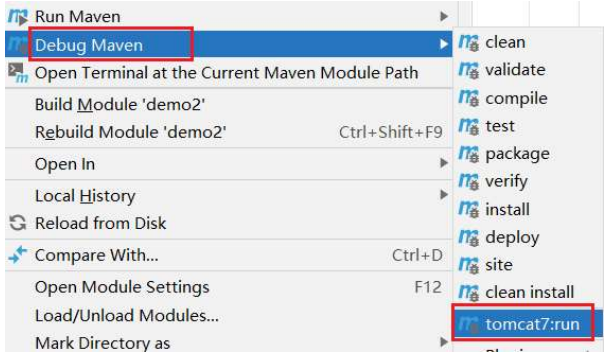
```
<build>
    <plugins>
        <!--Tomcat 插件-->
        <plugin>
            <groupId>org.apache.tomcat.maven</groupId>
            <artifactId>tomcat7-maven-plugin</artifactId>
            <version>2.2</version>
        </plugin>
    </plugins>
</build>
```

```
<plugin>
    <groupId>org.apache.tomcat.maven</groupId>
    <artifactId>tomcat7-maven-plugin</artifactId>
    <version>2.2</version>
    <configuration>
        <port>80</port><!--访问端口号-->
        <path>/</path><!--项目访问路径-->
    </configuration>
</plugin>
```

2. 使用Maven Helper 插件快速启动项目，选中项目，右键 --> Run Maven --> tomcat7:run



如果需要断点调试，选择 Debug Maven

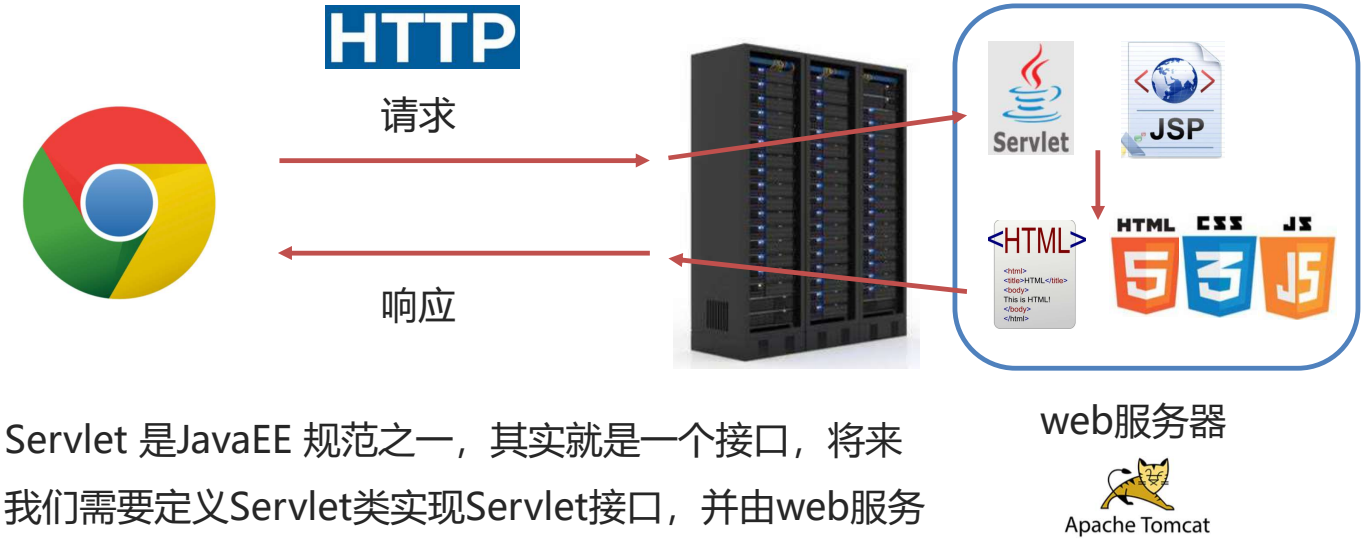


目录

Contents

Servlet

- Servlet 是 Java提供的一门动态web资源开发技术



- Servlet 是JavaEE 规范之一，其实就是一个接口，将来我们需要定义Servlet类实现Servlet接口，并由web服务器运行Servlet

```
public interface Servlet
```

Defines methods that all servlets must implement.

A servlet is a small Java program that runs within a Web server.

Servlet

- 快速入门
- Servlet 执行流程
- Servlet 生命周期
- Servlet 体系结构
- Servlet urlPattern配置
- XML 配置方式编写 Servlet

Servlet

- 快速入门
- Servlet 执行流程
- Servlet 生命周期
- Servlet 体系结构
- Servlet urlPattern配置
- XML 配置方式编写 Servlet

步骤

Servlet 快速入门

1. 创建 web项目，导入 Servlet依赖坐标

```
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>javax.servlet-api</artifactId>
    <version>3.1.0</version>
    <scope>provided</scope>
</dependency>
```

2. 创建：定义一个类，实现 Servlet接口，并重写接口中所有方法，并在 service方法中输入一句话

```
public class ServletDemo1 implements Servlet {
    public void service(){}
}
```

3. 配置：在类上使用@WebServlet 注解，配置该 Servlet的访问路径

```
@WebServlet("/demo1")
public class ServletDemo1 implements Servlet {
```

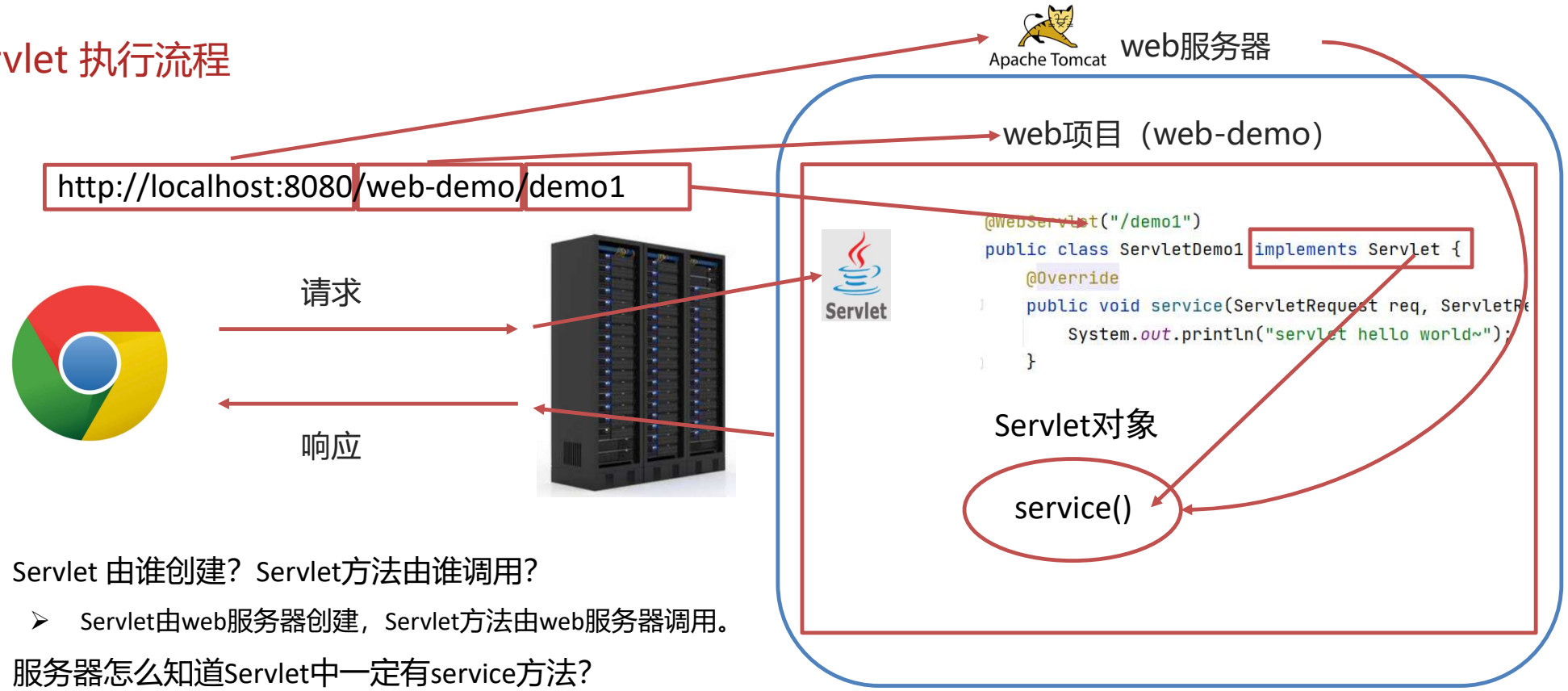
4. 访问：启动 Tomcat，浏览器输入URL 访问该Servlet

```
http://localhost:8080/web-demo/demo1
```

Servlet

- 快速入门
- Servlet 执行流程
- Servlet 生命周期
- Servlet 体系结构
- Servlet urlPattern配置
- XML 配置方式编写 Servlet

Servlet 执行流程



1. Servlet 由谁创建？Servlet方法由谁调用？
   - Servlet由web服务器创建，Servlet方法由web服务器调用。
2. 服务器怎么知道Servlet中一定有service方法？
   - 因为我们自定义的Servlet，必须实现Servlet接口并复写其方法，而Servlet接口中有service方法

Servlet

- 快速入门
- Servlet 执行流程
- Servlet 生命周期
- Servlet 体系结构
- Servlet urlPattern配置
- XML 配置方式编写 Servlet

Servlet 生命周期

- 对象的生命周期指一个对象从被创建到被销毁的整个过程

- Servlet运行在Servlet容器(web服务器)中，其生命周期由容器来管理，分为4个阶段：
  1. 加载和实例化：默认情况下，当Servlet第一次被访问时，由容器创建Servlet对象
  2. 初始化：在Servlet实例化之后，容器将调用Servlet的init()方法初始化这个对象，完成一些如加载配置文件、创建连接等初始化的工作。该方法只调用一次
  3. 请求处理：每次请求Servlet时，Servlet容器都会调用Servlet的service()方法对请求进行处理。
  4. 服务终止：当需要释放内存或者容器关闭时，容器就会调用Servlet实例的destroy()方法完成资源的释放。在destroy()方法调用之后，容器会释放这个Servlet实例，该实例随后会被Java的垃圾收集器所回收

```
@WebServlet(urlPatterns = "/demo",
        loadOnStartup = 1)
```

① 负整数：第一次被访问时创建Servlet对象

② 0或正整数：服务器启动时创建Servlet对象，数字越小优先级越高

Servlet 方法介绍

- 初始化方法，在Servlet被创建时执行，只执行一次

```
void init(ServletConfig config)
```

- 提供服务方法， 每次Servlet被访问，都会调用该方法

```
void service(ServletRequest req, ServletResponse res)
```

- 销毁方法，当Servlet被销毁时，调用该方法。在内存释放或服务器关闭时销毁Servlet

```
void destroy()
```

- 获取ServletConfig对象

```
ServletConfig getServletConfig()
```

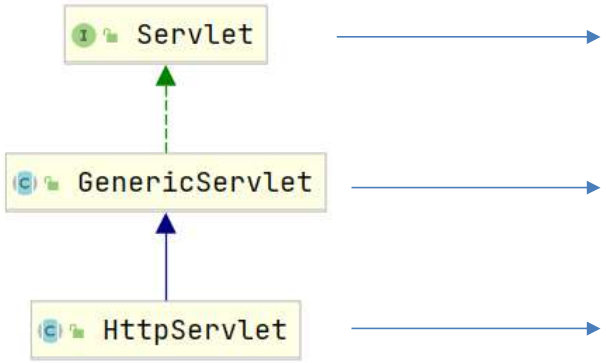
- 获取Servlet信息

```
String getServletInfo()
```

Servlet

- 快速入门
- Servlet 执行流程
- Servlet 生命周期
- Servlet 体系结构
- Servlet urlPattern配置
- XML 配置方式编写 Servlet

Servlet 体系结构



Servlet体系根接口

Servlet抽象实现类

对HTTP协议封装的Servlet实现类

我们将来开发B/S架构的web项目，都是针对HTTP协议，所以我们自定义Servlet，会继承HttpServlet

```
@WebServlet("/demo4")
public class ServletDemo4 extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
        //TODO Get 请求方式处理逻辑
    }
    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse resp)
        //TODO Post 请求方式处理逻辑
    }
}
```

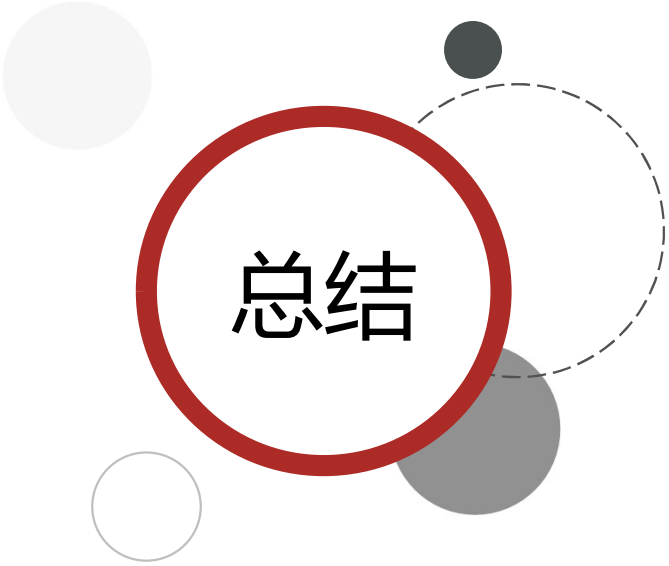
1. HttpServlet中为什么要根据请求方式的不同，调用不同方法?
2. 如何调用?

Servlet 体系结构

HttpServlet 原理

- HTTP 协议中，GET 和 POST 请求方式的数据格式不一样，将来要想在Servlet中处理请求参数，得在service方法中判断请求方式，并且根据请求方式的不同，分别进行处理：

```
//获取请求方式
String method = req.getMethod();
// 判断请求参数，不同请求方式，进行不一样的处理逻辑
if("GET".equals(method)){
    // 执行 GET请求方式的处理逻辑
    doGet(req,resp);
}else if("POST".equals(method)){
    //执行 POST请求方式的处理逻辑
    doPost(req,resp);
}
```



1. HttpServlet 使用步骤

   ① 继承HttpServlet

   ② 重写doGet和doPost方法

2. HttpServlet原理

   获取请求方式，并根据不同的请求方式，调用不同的doXxx方法

Servlet

- 快速入门
- Servlet 执行流程
- Servlet 生命周期
- Servlet 体系结构
- Servlet urlPattern配置
- XML 配置方式编写 Servlet

Servlet urlPattern配置

- Servlet 要想被访问，必须配置其访问路径（urlPattern）

1. 一个Servlet，可以配置多个 urlPattern

```
@WebServlet(urlPatterns = {"/demo1","/demo2"})
```

2. urlPattern 配置规则

① 精确匹配

② 目录匹配

③ 扩展名匹配

④ 任意匹配

Servlet urlPattern配置

2. urlPattern配置规则：

① 精确匹配：

- 配置路径：`@WebServlet("/user/select")`
- 访问路径：localhost:8080/web-demo/user/select

② 目录匹配：

- 配置路径：`@WebServlet("/user/*")`
- 访问路径：localhost:8080/web-demo/user/aaa
  localhost:8080/web-demo/user/bbb

③ 扩展名匹配：

- 配置路径：`@WebServlet("*.do")`
- 访问路径：localhost:8080/web-demo/aaa.do
  localhost:8080/web-demo/bbb.do

④ 任意匹配：

- 配置路径：`@WebServlet("/")`
  `@WebServlet("/*")`
- 访问路径：localhost:8080/web-demo/hehe
  localhost:8080/web-demo/haha
- / 和 /* 区别：
  - 当我们的项目中的Servlet配置了“/”，会覆盖掉tomcat中的DefaultServlet，当其他的 url-pattern都匹配不上时都会走这个Servlet
  - 当我们的项目中配置了“/*”，意味着匹配任意访问路径
- 优先级：

  精确路径 > 目录路径 > 扩展名路径 > /* > /

Servlet

- 快速入门
- Servlet 执行流程
- Servlet 生命周期
- Servlet 体系结构
- Servlet urlPattern配置
- XML 配置方式编写 Servlet

XML 配置方式编写 Servlet

- Servlet 从3.0版本后开始支持使用注解配置，3.0版本前只支持 XML 配置文件的配置方式
- 步骤：
  1. 编写 Servlet类
  2. 在 web.xml中配置该Servlet

```
<servlet>
    <servlet-name>demo5</servlet-name>
    <servlet-class>com.itheima.web.servlet.ServletDemo5</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>demo5</servlet-name>
    <url-pattern>/demo5</url-pattern>
</servlet-mapping>
```


黑马程序员
www.itheima.com
传智教育旗下高端IT教育品牌