

# 初识 Servlet

## 本节目标

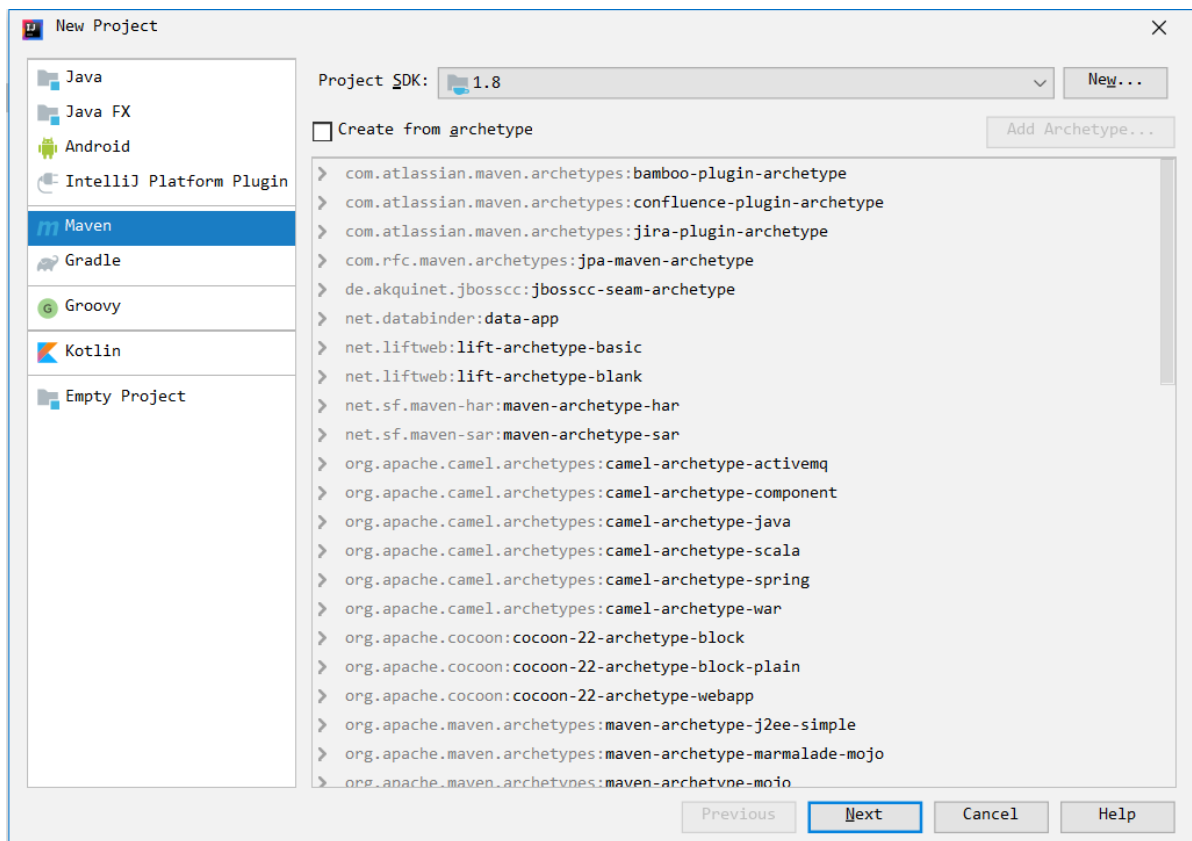
- 学会 Tomcat 的安装与基本配置
- 可以独立完成一个演示级别的 Servlet 项目
- 理解 Tomcat 的作用和基本工作原理
- 理解 Servlet 的作用
- 理解 Servlet 对象的生命周期
- 理解 Servlet 和多线程的关系

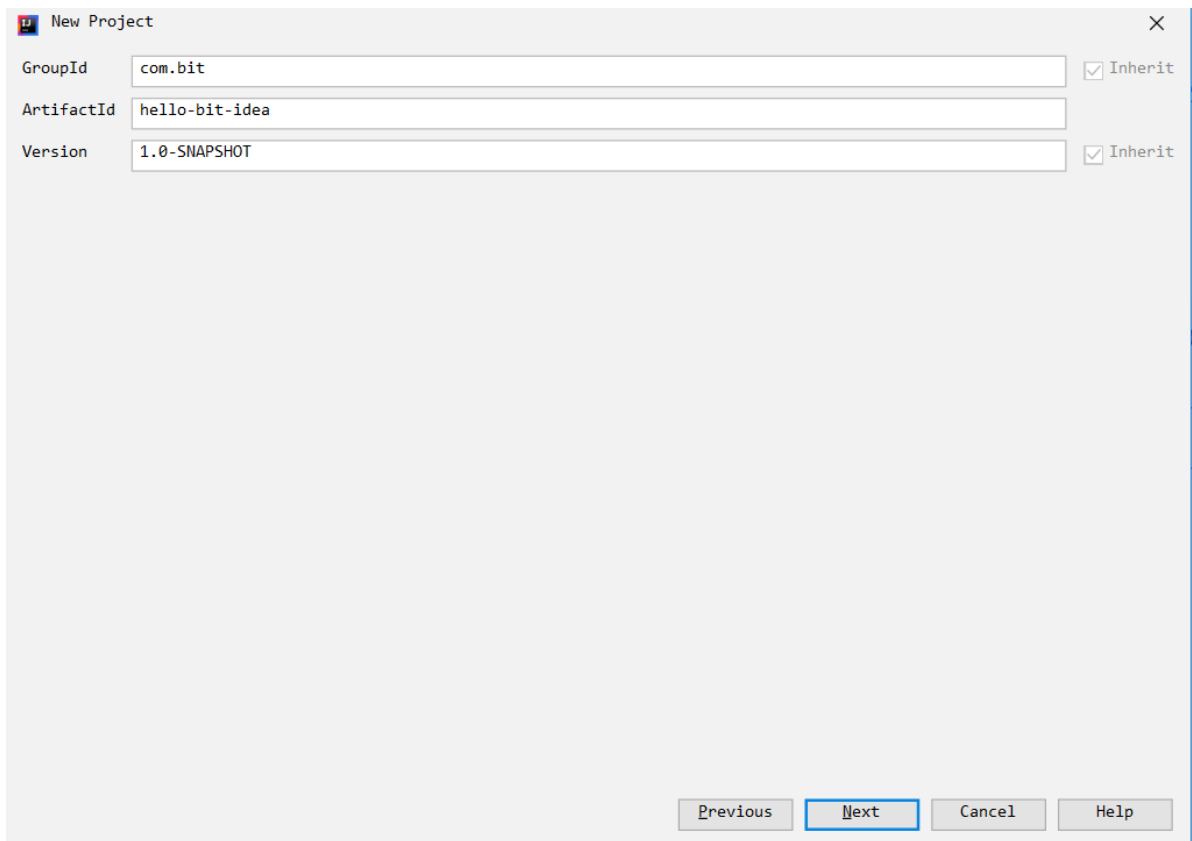
## 1. Servlet 简介

Servlet 的 JavaEE 中指定了一套标准，目前主要的用途就是写 web 应用（web application）。

## 2. 演示: Servlet 的 Hello World —— 使用 IDEA

使用 IDEA 创建 maven 项目





New Project

GroupId: com.bit ☒ Inherit

ArtifactId: hello-bit-idea

Version: 1.0-SNAPSHOT ☒ Inherit

Previous Next Cancel Help

修改 pom.xml 如下，添加项的意义见注释

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.bit</groupId>
  <artifactId>hello-bit-idea</artifactId>
  <version>1.0-SNAPSHOT</version>
  <!-- 打包方式是 war 包，一种用于 web 应用的包，原理类似 jar 包 -->
  <packaging>war</packaging>

  <dependencies>
    <!-- 加入 servlet 依赖 -->
    <dependency>
      <groupId>javax.servlet</groupId>
      <artifactId>javax.servlet-api</artifactId>
      <!-- servlet 版本和 tomcat 版本有对应关系，切记 -->
      <version>3.1.0</version>
      <!-- 这个意思是我们只在开发阶段需要这个依赖，部署到 tomcat 上时就不需要了 -->
      <scope>provided</scope>
    </dependency>
  </dependencies>
</project>
```

目前的文件夹结构应该是

```
src\  
  main\  
    java\      这里是我们 java 代码的根路径  
    resources\ 这里是我们项目中用到的资源文件的根路径  
  test\  
    java\      这里用来存放单元测试代码用的，学过测试的同学应该了解  
pom.xml
```

我们需要在 src\main 下新建 webapp\WEB-INF 文件夹，

然后在 src\main\webapp 下新建 index.html 文件，

在 src\main\webapp\WEB-INF 下新建 web.xml 文件

在 src\main\java 下新建 HelloServlet.java 文件

最终效果是

```
src\  
  main\  
    java\      这里是我们 java 代码的根路径  
      HelloServlet.java 我们要写 Servlet 的地方  
    resources\ 这里是我们项目中用到的资源文件的根路径  
  webapp\  
    index.html 静态文件  
    WEB-INF\  
      web.xml 相关配置  
  test\  
    java\      这里用来存放单元测试代码用的，学过测试的同学应该了解  
pom.xml
```

其中 index.html 内容为

```
<html>  
  <head>  
    <meta charset="utf-8">  
  </head>  
  <body>  
    <h1>这是一个静态页面</h1>  
  </body>  
</html>
```

web.xml 内容为，注意不要修改文件上边的信息，这个也是和版本有关的。

```
<?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_3_1.xsd"  
  version="3.1"  
  metadata-complete="true">  
  <servlet>  
    <servlet-name>Hello</servlet-name>  
    <servlet-class>HelloServlet</servlet-class>  
  </servlet>  
  <servlet-mapping>
```

```
<servlet-name>Hello</servlet-name>
  <url-pattern>/hello-servlet</url-pattern>
</servlet-mapping>
</web-app>
```

HelloServlet.java 内容为

```
import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class HelloServlet extends HttpServlet {
    @Override
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws IOException, ServletException {
        response.setContentType("text/html");
        response.setCharacterEncoding("UTF-8");
        PrintWriter out = response.getWriter();

        out.println("<h1>这是动态的内容</h1>");
    }
}
```

同学应该可以发现，这些文件的内容和我们之前通过 cmd 方式创建时的内容时基本一致的。

然后使用我们的 maven package

将 target 中的 hello-bit-idea.war 文件复制到 tomcat 的 webapps 路径下，最好提前把原来的 hello-bit-idea.war 和 hello-bit-idea 文件夹删除，以免相互影响。

不用特意重启 tomcat，再次请求

<http://127.0.0.1:8080/hello-bit-idea>

<http://127.0.0.1:8080/hello-bit-idea/hello-servlet>

可以看到同样的效果

然后可以观察到 tomcat 的 webapps 文件夹下，我们的 war 包已经被自动解压了，实际内部的结构和我们 cmd 上做的完全一致

## 3. Tomcat

### 3.1 相关概念

**container** 我们在 java 中，一般把可以承担应用服务器 + Servlet 标准的一套软件叫做 web 容器 (container)。

**context** 在 tomcat 的语境下，通常 context 就是指的是一个 web 应用。

## 3.2 HttpServlet 和 Servlet 的关系

```
interface Servlet {  
    void init(ServletConfig var1) throws ServletException;  
    void service(ServletRequest var1, ServletResponse var2) throws  
ServletException, IOException;  
    void destroy();  
    // 省略了部分我们不关心的方法  
}
```

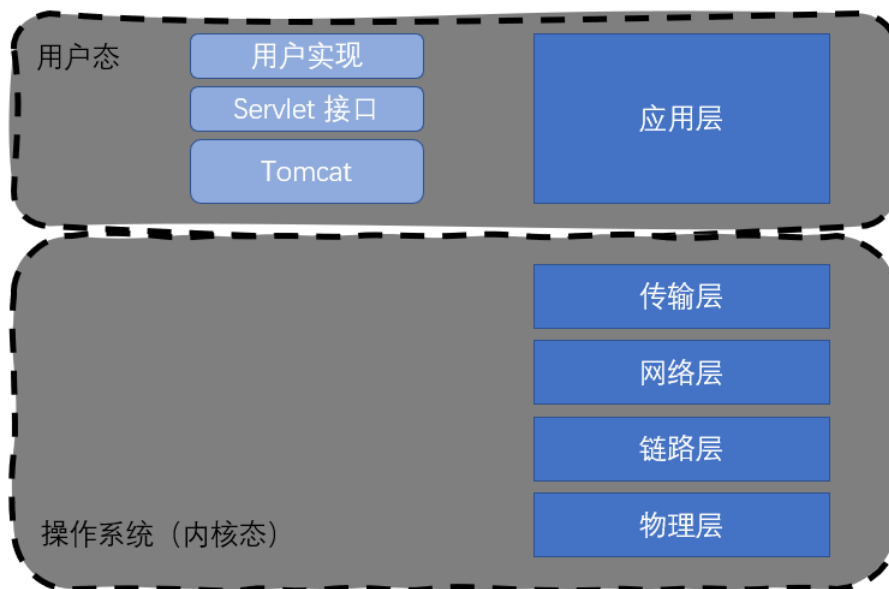
Servlet 在这个语境下只是一个 java 中的普通接口

```
abstract class HttpServlet implements Servlet {  
    @Override  
    protected void service(HttpServletRequest req, HttpServletResponse resp)  
throws ServletException, IOException {  
        // 根据 request 的 method 不同, 调用不同的方法  
    }  
  
    // GET 时调用  
    protected void doGet(protected void service(HttpServletRequest req,  
HttpServletResponse resp) throws ServletException, IOException {  
        ...  
    }  
  
    // POST 时调用  
    protected void doPost(protected void service(HttpServletRequest req,  
HttpServletResponse resp) throws ServletException, IOException {  
        ...  
    }  
  
    // 省略了 HTTP 协议支持的其他方法  
}
```

**总结：**HttpServlet 是 Servlet 接口的一个实现类，但它本身是个抽象类，所以我们的 HelloServlet 继承了 HttpServlet，主要负责的是 HTTP 请求的处理，根据我们要支持的方法，选择覆写 doGet 或者 doPost 或者其他合适的方法即可。

## 3.3 Tomcat 的作用及定位

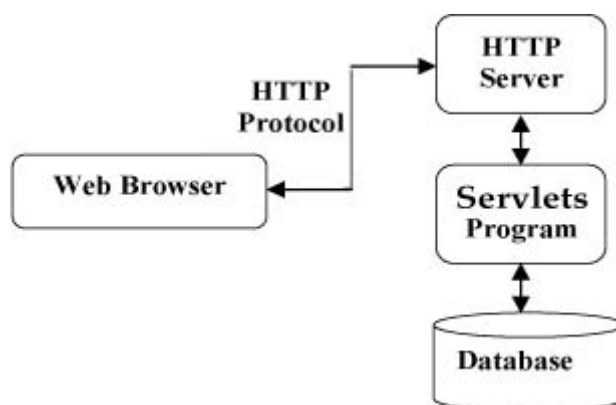
**Tomcat 在操作系统和网络协议栈中的位置**



Tomcat 实现的主要是 HTTP 协议，也就是应用层，同时在操作系统的视角里，是属于用户态程序。

我们自己的实现是在 Tomcat 基础上运行的。

我们写的 Servlet 主要的位置



### 3.4 伪码讲解 tomcat 基本工作原理——只考虑单 web application 的情况

总体流程

```
class Tomcat {
    // 用来存储所有的 Servlet 对象
    private List<Servlet> instanceList = new ArrayList<>();

    public void start() {
        // 根据约定，读取 WEB-INF/web.xml 配置文件；

        // 假定是我们从配置文件中解析出的所有的 Servlet 子类（也就是我们的业务类）
        Class<Servlet>[] allServletClasses = ...;

        // 这里要做的是实例化出所有的 Servlet 对象出来；
        for (Class<Servlet> cls : allServletClasses) {
            // 这里是利用 java 中的反射特性做的
            // 实际上还得涉及一个类的加载问题，因为我们的类字节码文件，是按照约定的
            // 方式（全部在 WEB-INF/classes 文件夹下）存放的，所以 tomcat 内部是
            // 实现了一个自定义的类加载器(ClassLoader)用来负责这部分工作。
        }
    }
}
```

```

        servlet ins = cls.newInstance();
        instanceList.add(ins);
    }

    // 调用每个 Servlet 对象的 init() 方法，这个方法在对象的生命中只会被调用这一次；
    for (Servlet ins : instanceList) {
        ins.init();
    }

    // 利用我们之前学过的知识，启动一个 HTTP 服务器
    // 并用线程池的方式分别处理每一个 Request
    ServerSocket serverSocket = new ServerSocket(8080);
    // 实际上 tomcat 不是用的固定线程池，这里只是为了说明情况
    ExecutorService pool = Executors.newFixedThreadPool(100);

    while (true) {
        Socket socket = ServerSocket.accept();
        // 每个请求都是用一个线程独立支持，这里体现了我们 Servlet 是运行在多线程环境下的
        pool.execute(new Runnable() {
            doHttpRequest(socket);
        });
    }

    // 调用每个 Servlet 对象的 destroy() 方法，这个方法在对象的生命中只会被调用这一次；
    for (Servlet ins : instanceList) {
        ins.destroy();
    }
}

public static void main(String[] args) {
    new Tomcat().start();
}
}

```

## 解析 web.xml 的过程

```

class Tomcat {
    // 根据 servlet-name 找到合适的 Servlet 对象，这个 map 可以在合适的位置填充
    private Map<String, Servlet> namedInstanceMap;

    // 根据 URL 找到合适的 servlet-name
    // 实际上，因为是利用正则做的，所以不一定是 map
    private Map<String, String> urlMapping;

    private void parsewebXML() {
        // 在 WEB-INF 文件夹下找到 web.xml 文件，根据 xml 上的版本，选择合适的格式去解析
        // 主要目的有以下几个：

        // 1. 每一个 servlet 标签，就代表着一个 Servlet 子类
        //     标签中明确给出的 servlet-name 用来查找定位该类的 key
        //     标签中明确给出的 servlet-class 用来定位类字节码文件在硬盘的位置
        //     最终的类字节码文件在 WEB-INF/classes + <servlet-class> 标签中的路径

        // 2. 每一个 servlet-mapping 标签，代表着如何通过用户请求的 URL，决定是交给哪个
        //     Servlet 对象去处理
        //     定位流程是这样的
    }
}

```

```

// 如果 URL 可以匹配 servlet-mapping 中的 url-pattern 子标签（这里用的是正则）
// 则，使用 servlet-name 为 key 的 Servlet 对象去处理这个请求
// 再根据 servlet-name 去创建好的对象 map 中，找到 Servlet 对象即可
//
// 如果没有找到合适 Servlet 类能匹配这次 URL，则返回 tomcat 中的 404 页面
}
}

```

在线程中调用的每次请求的处理流程

```

class Tomcat {
    void doHttpRequest(Socket socket) {
        // 参照我们之前学习的 HTTP 服务器类似的原理，进行 HTTP 协议的请求解析，和响应构建
        HttpServletRequest req = HttpServletRequest.parse(socket);
        HttpServletRequest resp = HttpServletRequest.build(socket);

        // 判断 URL 对应的文件是否可以直接在我们的根路径上找到对应的文件，如果找到，就是静态内容
        // 直接使用我们学习过的 IO 进行内容输出
        if (file.exists()) {
            // 返回静态内容
            return;
        }

        // 走到这里的逻辑都是动态内容了

        // 根据我们在配置中说的，按照 URL -> servlet-name -> Servlet 对象的链条
        // 最终找到要处理本次请求的 Servlet 对象
        Servlet ins = findInstance(req.getURL());

        // 调用 Servlet 对象的服务方法
        // 这里就会最终调用到我们自己写的 HttpServlet 的子类里的方法了
        try {
            ins.service(req, resp);
        } catch (Exception e) {
            // 返回 500 页面，表示服务器内部错误
        }
    }
}

```

## 总结

Tomcat 就是一个所谓的 Web Container，内部实现了一个 HTTP 服务器

同时会根据不同的 URL，区分出是静态内容还是动态内容

如果是动态内容，则根据 web.xml 中的配置，找到合适的对象进行处理

我们自己写的代码只是这个环节中的一个步骤，不再需要从 main 入口开始实现了

我们最终文件夹结构要按照之前的方式布局，这样 tomcat 才可以正确的找到对应的文件

我们在 Servlet 中写的代码，其实都是在一个多线程环境下运行的，要注意保护线程安全问题

每个 Servlet 对象，在其生命过程中，init() 在启动时被调用一次，destroy() 在退出时被调用一次，service() 在每次请求的处理过程中都会调用一次



### 3.4 回顾演示项目

对比下开发环境（IDEA 上）和运行环境（部署到 tomcat 上）的文件夹结构对应关系

#### 开发环境

```
src\  
  main\  
    java\      这里的代码编译后的类字节码，会被放到运行环境的 WEB-INF\classes 下  
  webapp\  
    WEB-INF\  
      web.xml  这里的代码编译后的类字节码，会被放到运行环境的 WEB-INF\web.xml  
// 其他无关路径省略
```

#### 运行环境

```
\  
  index.html  静态文件，从开发环境的 webapp 下复制过来  
  WEB-INF\  
    web.xml  从开发环境的 src\webapp\WEB-INF\web.xml 复制过来  
    classes\  
      HelloServlet.class 开发环境的 src\main\java\HelloServlet.java 编译后的类  
文件
```

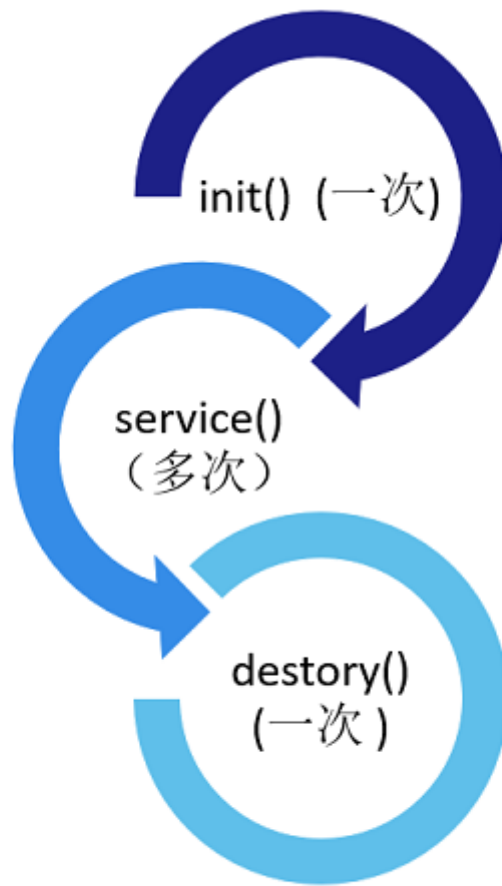
简单来说，idea 会根据一个预先设定好的规则，把指定的内容放到指定的路径下，所以我们需要自行保证环境环境的路径结构正确。

## 4. 总结核心知识点

### 4.1 Servlet 是什么，它有什么作用

Servlet 这个词本身是有一定的歧义性的

### 4.2 Servlet 对象的生命周期



Servlet 生命周期方法调用

### 4.3 Servlet 对象工作在多线程环境下

因为 tomcat 内部是使用线程处理每个请求的，而且每个 servlet 对象只会存在一个，所以我们覆写的 doGet 类似的方法，是在多线程环境下运行的，是需要考虑线程安全问题的。