

作者：千锋-索尔

版本：QF1.0

版权：千锋Java教研院

# 一、为什么要用Spring框架

## 1.1 Spring框架概述

- Spring 使企业创建 Java 应用程序变得更加容易。
- Spring是众多优秀设计模式的组合（工厂、单例、代理、适配器、包装器、观察者、模板、策略）。
- Spring并未替代现有框架产品，而是将众多框架进行有机整合，简化企业级开发，俗称"胶水框架"。

## 1.2 原生web开发中存在哪些问题？

- 传统Web开发存在硬编码所造成的过度程序耦合（例如：Service中作为属性Dao对象）。
- 部分Java EE API较为复杂，使用效率低（例如：JDBC开发步骤）。
- 侵入性强，移植性差（例如：DAO实现的更换，从Connection到SqlSession）。

## 1.3 没有使用Spring框架的案例

简单实现MVC三层结构的项目逻辑，发现如果要修改一个DAO的实现，则需要修改原业务代码。而生产环境中，修改原业务代码的风险是非常高的。

- Product 商品类

```
package entity;

public class Product {
    private Long id;
    private String name;

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}
```

```
}
```

- ProductService 商品服务接口

```
package service;

import entity.Product;

public interface ProductService {
    Product getProductById();
}
```

- ProductServiceImpl 商品服务实现类

```
package service.impl;

import dao.ProductDao;
import dao.impl.ProductDaoMySQLImpl;
import dao.impl.ProductDaoOracleImpl;
import entity.Product;
import service.ProductService;

public class ProductServiceImpl implements ProductService {

    //    ProductDao productDao = new ProductDaoOracleImpl();
    ProductDao productDao = new ProductDaoMySQLImpl();

    /**
     * 根据商品id去数据库查询得到该商品
     * @return
     */
    @Override
    public Product getProductById() {
        return productDao.selectProductById();
    }
}
```

- ProductDao 商品数据访问接口

```
package dao;

import entity.Product;

public interface ProductDao {

    Product selectProductById();

}
```

- ProductDaoMySQLImpl 使用MySQL数据库的数据访问实现类

```
package dao.impl;

import dao.ProductDao;
import entity.Product;

public class ProductDaoMySQLImpl implements ProductDao {
    /**
     * 查询MySQL数据库获得数据
     * @return
     */
    @Override
    public Product selectProductById() {
        //模拟查询数据库操作
        System.out.println("查询MySQL数据库");
        return new Product();
    }
}
```

- TestDemo1 测试类

```
package controller;

import entity.Product;
import service.ProductService;
import service.impl.ProductServiceImpl;

public class TestDemo1 {

    public static void main(String[] args) {
        ProductService productService = new ProductServiceImpl();
        Product product = productService.getProductById();

    }
}
```

```
}
```

执行后打印结果：

```
查询MySQL数据库
```

如果要修改成使用Oracle查询数据库，则需要添加Oracle的Dao实现类

- ProductDaoOracleImpl

```
package dao.impl;

import dao.ProductDao;
import entity.Product;

public class ProductDaoOracleImpl implements ProductDao {
    /**
     * 根据Id查询Oracle数据库获得数据
     * @return
     */
    @Override
    public Product selectProductById() {
        //模拟数据库查询
        System.out.println("查询Oracle数据库");
        return new Product();
    }
}
```

还得修改业务代码：ProductServiceImpl

```
package service.impl;

import dao.ProductDao;
import dao.impl.ProductDaoMySQLImpl;
import dao.impl.ProductDaoOracleImpl;
import entity.Product;
import service.ProductService;

public class ProductServiceImpl implements ProductService {

    // 查询Oracle数据库
    ProductDao productDao = new ProductDaoOracleImpl();
    // 查询MySQL数据库
    //ProductDao productDao = new ProductDaoMySQLImpl();

    /**
```

```
    * 根据商品id去数据库查询得到该商品
    * @return
    */
@Override
public Product getProductById() {
    return productDao.selectProductById();
}
}
```

## 1.4 Spring的核心：IoC控制反转

- **Inverse Of Control：控制反转**

反转了依赖关系的满足方式，由之前的自己创建依赖对象，变为由工厂推送。(变主动为被动，即反转)

解决了具有依赖关系的组件之间的强耦合，使得项目形态更加稳健。

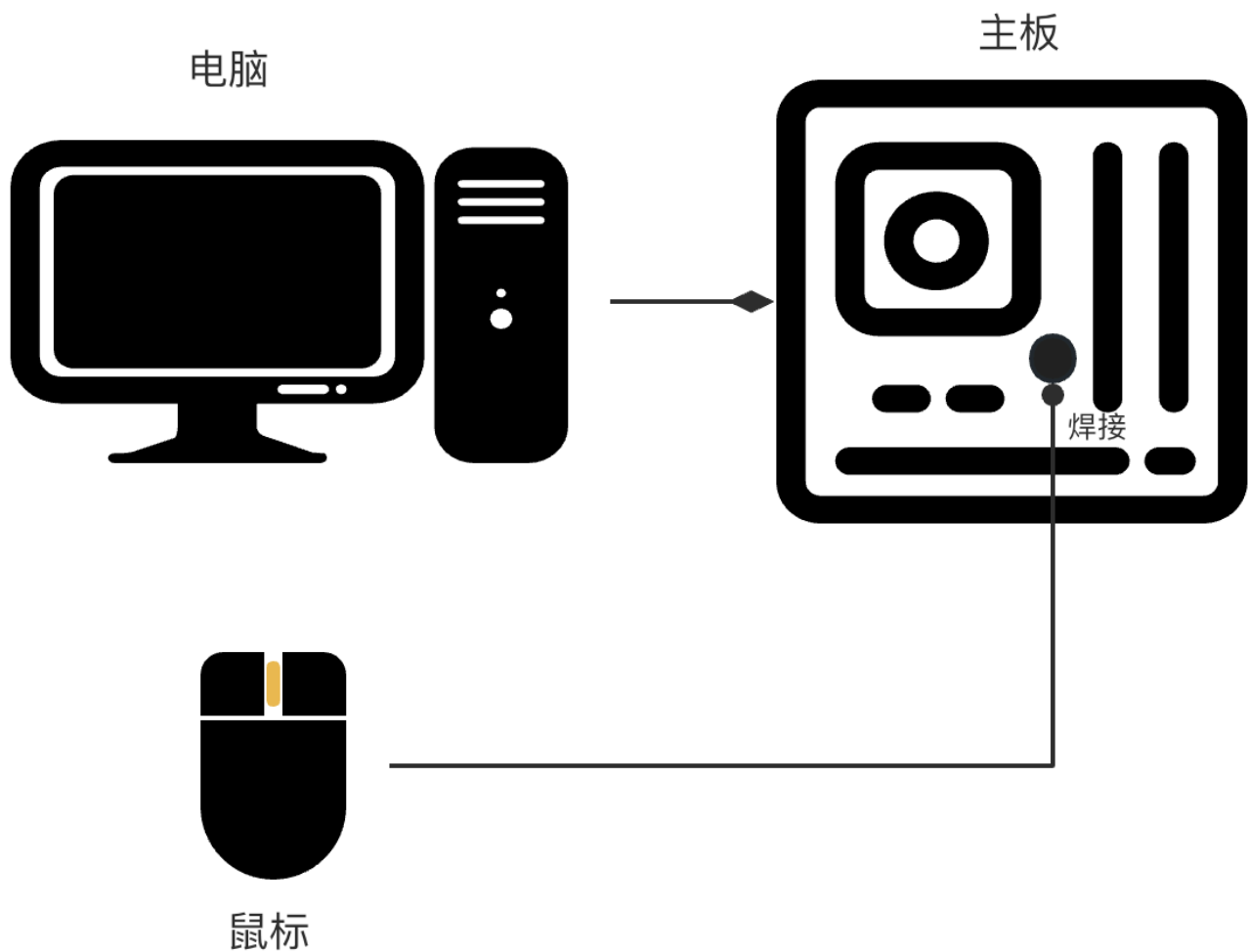
- 没有控制反转的场景

一台电脑的主机的主板上，直接焊接了鼠标连线。在启动电脑前，如果鼠标是坏的，那么主板也会报错，导致电脑没有办法启动。此时电脑能否启动成功的控制权在鼠标。

**此时为正向控制：鼠标----控制--->电脑**

- 正向控制的悲催：

由于鼠标坏了导致没办法启动电脑，于是要拆主板，重新焊接一个新的鼠标，不仅工作量大，而且风险也很大，主板很容易修出其他毛病。



- 控制反转的第一步：接口分离

将“焊接”动作改为使用可插拔的方式。在面向对象的概念里就是使用“接口”。

```
ProductDao productDao = new ProductDaoOracleImpl();
```

这样做的好处：主要是接口ProductDao的实现类即可。

这样做的坏处：修改实现类时需要修改代码。

回到例子中：鼠标连接主板的插口不再是焊死，而是使用了主机老式接口PS/2，换了鼠标后需要重启才能生效。

- 控制反转的第二步：依赖注入

若想实现鼠标的热插拔，需要把PS/2接口换成USB接口。只要支持USB接口的鼠标，就能够随意插拔和更换。此时会有两层含义：

**第一层，控制反转：**

即使鼠标坏了，电脑也能正常启动。原来由焊接的鼠标控制着电脑启动，现在由电脑自身控制，实现了控制反转。

**第二层，依赖注入（Dependency Injection）：**

依赖注入是控制反转具体的实现。

什么是依赖注入？把支持USB接口的鼠标理解成是依赖注入。只要某个鼠标是支持USB连接的，那就能连上主板，解决问题。从字面上理解，只要注入一个满足依赖的（符合USB接口规范的）鼠标即可。

## 1.5 Spring框架初体验

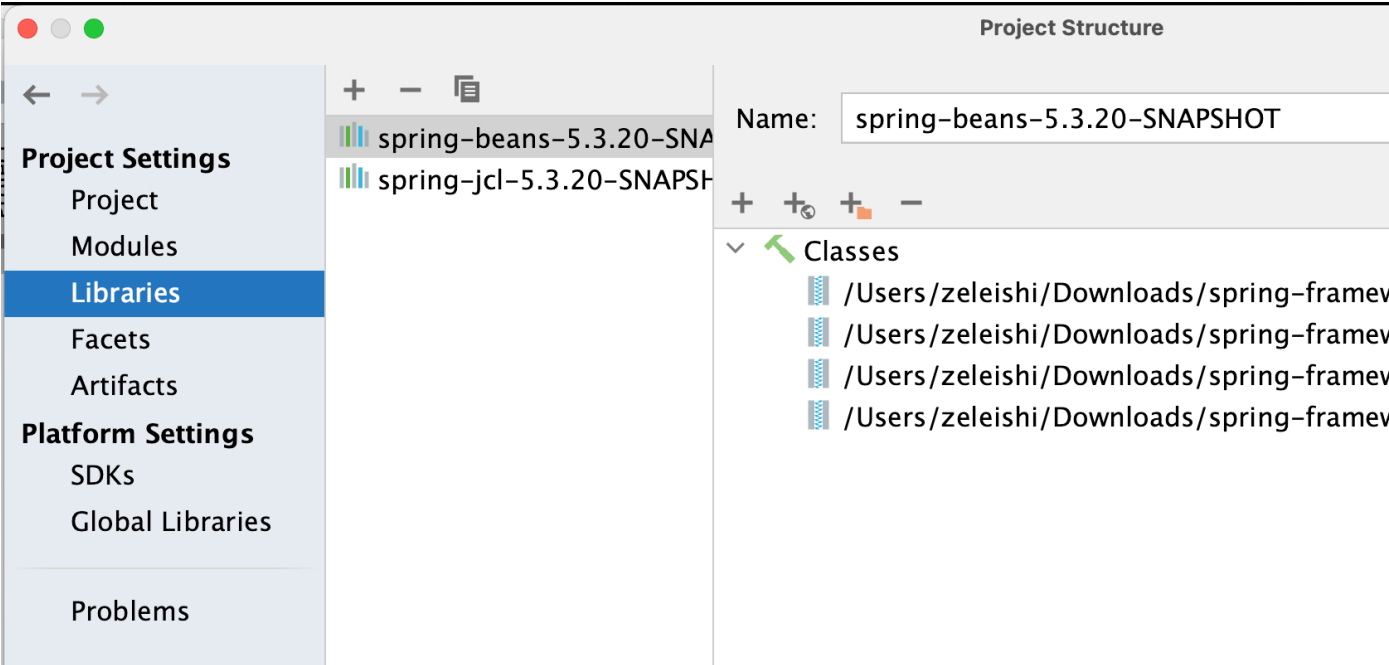
- 下载Spring框架jar包

```
https://repo.spring.io/
```

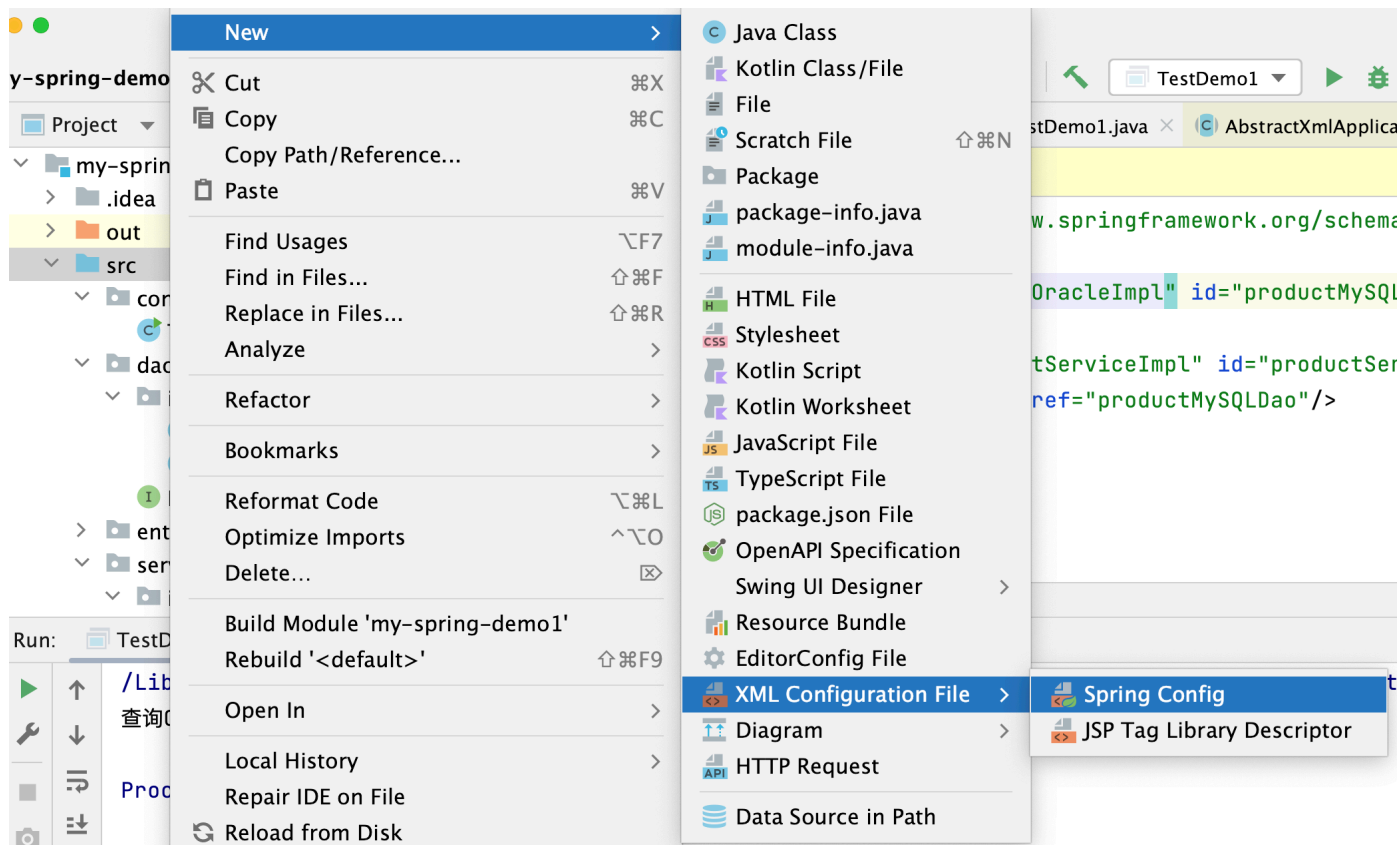
具体下载路径为：

```
snapshot->org->springframework->spring->具体版本
```

- 为项目添加类库



- 添加spring.xml文件



- 修改spring.xml文件内容

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">
    <!--实现控制反转-->
    <bean class="dao.impl.ProductDaoMySQLImpl" id="productMySQLDao" />
    <bean class="service.impl.ProductServiceImpl" id="productService" >
        <!--实现依赖注入-->
        <property name="productDao" ref="productMySQLDao" />
    </bean>
</beans>
```

- 通过修改productMySQLDal的实现类为 dao.impl.ProductDaoOracleImpl 实现类的切换。

其中，spring.xml中通过配置bean的方式实现了控制反转，不需要在代码中手动创建对象，而是通过配置文件配置bean的方式来把创建对象的工作交给IoC容器。第二，通过依赖注入的方式把ProductService需要的ProductDao实现类的对象注入进去，直接通过配置来指定不同类型的bean的注入，而不需要修改代码。

## 二、Spring框架介绍



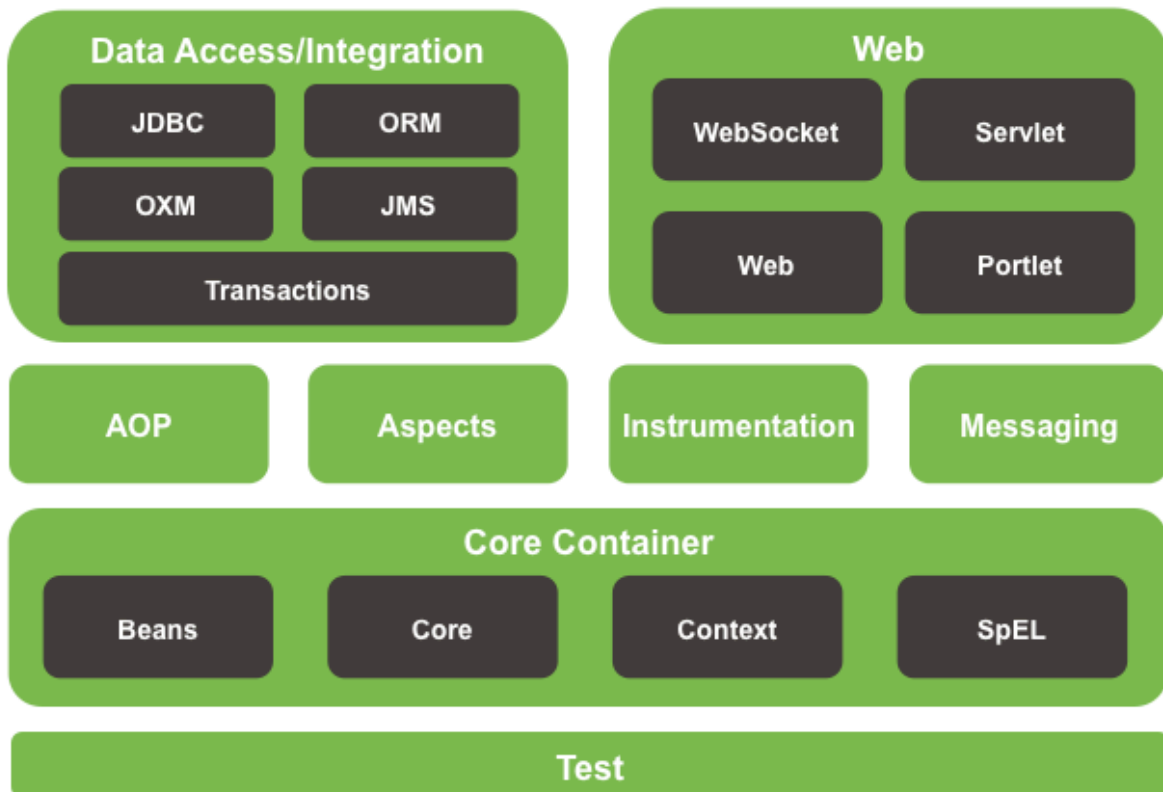
# 1."Spring"是什么？

Spring 在不同的背景下有这不同的含义. 它可以指 Spring Framework 项目本身,这也是创建他的初衷. 随着时间的推移,其他的 "Spring" 项目已经构建在 Spring 框架之上. 大多数时候,当人们说 "Spring",他们指的是整个项目家族. 本参考文档主要侧重于 Spring 的基础: 也就是 Spring 框架本身.

整个 Spring 框架被分成多个模块,应用程序可以选择需要的部分. core 是核心容器的模块,包括模块配置和依赖注入机制. Spring 框架还为不同的应用程序体系结构提供了基础支持,包括消息传递,事务数据和持久化以及 Web,还包括基于 Servlet 的 Spring MVC Web 框架,以及 Spring WebFlux 响应式Web框架.



## Spring Framework Runtime



Spring官网地址: [spring.io](https://spring.io)

Spring中文文档地址: <https://www.jcohy.com/projects/spring-framework#learn>

## 2.Spring 和 Spring 框架的历史

Spring 的初版发布在 2003 年,是为了克服早期 J2EE 规范的复杂性. 虽然有些人认为 (J2EE) 和 Spring 是竞争的,但是 Spring 实际上是对 Java EE 的补充. Spring 编程模型不受 Java EE 的平台制约,相反 它与精心挑选的个别规范的 Java EE 项目结合:

- Servlet API ([JSR 340](#))
- WebSocket API ([JSR 356](#))
- Concurrency Utilities ([JSR 236](#))
- JSON Binding API ([JSR 367](#))

- Bean Validation ([JSR 303](#))
- JPA ([JSR 338](#))
- JMS ([JSR 914](#))
- 用于事务协调的 JTA/JCA 设置.

Spring 框架还支持依赖注入 ([JSR 330](#)) 和通用注解 ([JSR 250](#)) 规范, 应用程序开发人员可以选择使用这些规范, 而不是 Spring Framework 提供的 Spring 特定机制.

在 Spring 框架 5.0 版本中, Spring 最低要求使用 Java EE 7 的版本(例如 Servlet 3.1+, JPA 2.1+ ), 同时在运行时能与使用 Java EE 8 的最新 API 集成(例如 Servlet 4.0, JSON Binding API) . 这使得 Spring 能完全兼容 Tomcat 8/9、WebSphere 9 或者 JBoss EAP 7 等等.

GroupId	ArtifactId	说明
org.springframework	<a href="#">spring-beans</a>	<a href="#">Beans</a> 支持, <a href="#">包含 Groovy</a>
org.springframework	<a href="#">spring-aop</a>	<a href="#">基于代理的AOP支持</a>
org.springframework	<a href="#">spring-aspects</a>	<a href="#">基于AspectJ 的切面</a>
org.springframework	<a href="#">spring-context</a>	<a href="#">应用上下文运行时, 包括调度和远程抽象</a>
org.springframework	<a href="#">spring-context-support</a>	<a href="#">支持将常见的第三方类库集成到 Spring 应用上下文</a>
org.springframework	<a href="#">spring-core</a>	<a href="#">其他模块所依赖的核心模块</a>
org.springframework	<a href="#">spring-expression</a>	<a href="#">Spring 表达式语言, <a href="#">SpEL</a></a>
org.springframework	spring-instrument	JVM 引导的仪表 (监测器) 代理
org.springframework	spring-instrument-tomcat	Tomcat 的仪表 (监测器) 代理
org.springframework	spring-jdbc	支持包括数据源设置和 JDBC 访问支持
org.springframework	spring-jms	支持包括发送/接收JMS消息的助手类
org.springframework	spring-messaging	对消息架构和协议的支持
org.springframework	spring-orm	对象/关系映射, 包括对 JPA 和 Hibernate 的支持
org.springframework	spring-oxm	对象/XML 映射 (Object/XML Mapping, OXM)
org.springframework	<a href="#">spring-test</a>	<a href="#">单元测试和集成测试支持组件</a>
org.springframework	<a href="#">spring-tx</a>	<a href="#">事务基础组件, 包括对 DAO 的支持及 JCA 的集成</a>
org.springframework	<a href="#">spring-web</a>	<a href="#">web支持包, 包括客户端及web远程调用</a>
org.springframework	<a href="#">spring-webmvc</a>	<a href="#">REST web 服务及 web 应用的 MVC 实现</a>
org.springframework	spring-webmvc-portlet	用于 Portlet 环境的MVC实现
org.springframework	spring-websocket	WebSocket 和 SockJS 实现, 包括对 STOMP 的支持
org.springframework	<a href="#">spring-jcl</a>	<a href="#">Jakarta Commons Logging 日志系统</a>

随着时间的不断推移,Java EE在应用程序开发中越发重要,也不断发展、改善. 在 Java EE 和 Spring 的早期,应用程序被创建为部署到服务器的应用. 如今,在有 Spring Boot 的帮助后,应用可以创建在 devops 或云端. 而 Servlet 容器的嵌入和一些琐碎的东西也发生了变化. 在 Spring framework 5 中,WebFlux 应用程序甚至可以不直接使用 Servlet 的 API,并且可以在非 Servlet 容器的服务器(如 Netty) 上运行.

Spring 还在继续创新和发展,如今,除了 Spring 框架以外,还加入了其他项目,如: `Spring Boot`, `Spring Security`, `Spring Data`, `Spring Cloud`, `Spring Batch` 等等. 请记住,每一个 Spring 项目都有自己的源代码库, 问题跟踪以及发布版本. 请上 [spring.io/projects](https://spring.io/projects) 查看所有 Spring 家族的项目名单。

## 3.Spring 的设计理念

当你学习一个框架时,不仅需要知道他是如何运作的,更需要知道他遵循什么样的原则,以下是 Spring 框架遵循的原则:

- 提供各个层面的选择. Spring 允许您尽可能延迟设计决策. 例如,您可以在不更改代码的情况下通过配置切换持久性功能. 对于其他基础架构的问题以及与第三方 API 的集成也是如此.
- 包含多个使用视角. Spring 的灵活性非常高,而不是规定了某部分只能做某一件事.他以不同的视角支持广泛的应用需求
- 保持向后兼容. Spring 的发展经过了精心的设计,在不同版本之间保持与前版本的兼容. Spring 支持一系列精心挑选的 JDK 版本和第三方库,以方便维护依赖于 Spring 的应用程序和库.
- 关心 API 的设计. Spring 团队投入了大量的思考和时间来制作直观的 API,并在许多版本和许多年中都保持不变.
- 高标准的代码质量. Spring 框架提供了强有力的、精确的、即时的 Javadoc. Spring 这种要求干净、简洁的代码结构、包之间没有循环依赖的项目在 Java 界是少有的.

## 三、Maven快速入门

### 1、引言

#### 1.1 项目管理问题

项目中jar包资源越来越多，jar包的管理越来越沉重。

##### 1.1.1 繁琐

要为每个项目手动导入所需的jar，需要搜集全部jar

##### 1.1.2 复杂

项目中的jar如果需要版本升级，就需要再重新搜集jar

##### 1.1.3 冗余

相同的jar在不同的项目中保存了多份

#### 1.2 项目管理方案

java项目需要一个统一的便捷的管理工具：Maven

## 2、介绍

Maven这个单词来自于意第绪语（犹太语），意为知识的积累。

Maven是一个基于项目对象模型（POM）的概念的纯java开发的开源的项目管理工具。主要用来管理java项目，进行依赖管理(jar包依赖管理)和项目构建(项目编译、打包、测试、部署)。此外还能分模块开发，提高开发效率。

## 3、Maven安装

### 3.1 下载Maven

下载Maven

<http://us.mirrors.quenda.co/apache/maven/maven-3/3.5.4/binaries/>

# Index of /apache/maven/n

<a href="#">Name</a>	<a href="#">Last modified</a>	<a href="#">Size</a>	<a href="#">Description</a>
 <a href="#">Parent Directory</a>		-	
 <a href="#">apache-maven-3.5.4-bin.tar.gz</a>	2018-06-17 20:05	8.4M	
 <a href="#">apache-maven-3.5.4-bin.zip</a>	2018-06-17 20:05	8.6M	

### 3.2 Maven安装

#### 3.2.1 解压

注意：解压文件尽量不要放在含有中文或者特殊字符的目录下。

解压后，有如下目录：

- ~bin: 含有mvn运行的脚本~
- ~boot: 含有plexus-classworlds类加载器框架,Maven 使用该框架加载自己的类库。~
- ~conf: 含有settings.xml配置文件~
- ~lib: 含有Maven运行时所需要的java类库~

### 3.2.2 环境变量

maven依赖java环境，所以要确保java环境已配置好（maven-3.3+ 需要jdk7+）

maven本身有2个环境变量要配置：

```
`MAVEN_HOME` = maven的安装目录`  
`PATH` = maven的安装目录下的bin目录`
```

### 3.2.3 测试

查看maven版本信息

```
mvn -v
```

## 4、Maven配置

### 4.1 本地仓库

maven的conf目录中有 [settings.xml](#)，是maven的配置文件，做如下配置：

```
<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"  
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
           xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0  
http://maven.apache.org/xsd/settings-1.0.0.xsd">  
  <!-- localRepository  
  | The path to the local repository maven will use to store artifacts.  
  |  
  | Default: ${user.home}/.m2/repository  
<localRepository>/path/to/local/repo</localRepository>  
  -->  
  <!-- 选择一个磁盘目录，作为本地仓库 -->  
  <localRepository>D:\Program Files\maven\myrepository</localRepository>
```

### 4.2 JDK配置

在 `<profiles>` 标签中 增加 一个 `<profile>` 标签，限定maven项目默认的jdk版本。

内容如下：

```
<profiles>  
  <!-- 在已有的profiles标签中添加profile标签 -->  
  <profile>  
    <id>myjdk</id>  
    <activation>  
      <activeByDefault>true</activeByDefault>  
      <jdk>1.8</jdk>  
    </activation>
```

```
<properties>
    <maven.compiler.source>1.8</maven.compiler.source>
    <maven.compiler.target>1.8</maven.compiler.target>
    <maven.compiler.compilerVersion>1.8</maven.compiler.compilerVersion>
</properties>
</profile>
</profiles>
<!-- 让增加的 profile生效 -->
<activeProfiles>
    <activeProfile>myjdk</activeProfile>
</activeProfiles>
```

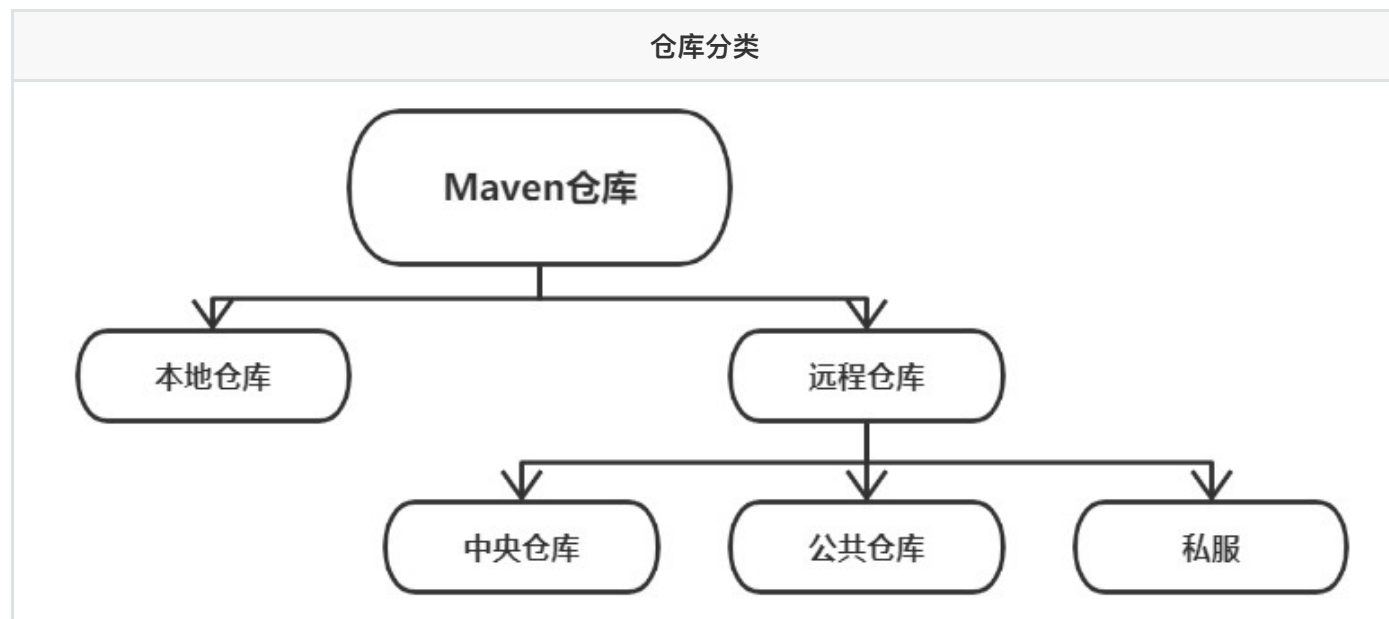
## 5、仓库

### 5.1 概念

- 存储依赖的地方，体现形式就是本地的一个目录。
- 仓库中不仅存放依赖，而且管理着每个依赖的唯一标识(坐标)，Java项目凭坐标获取依赖。

### 5.2 仓库分类

仓库分类如下：



当需要依赖时，会从仓库中取查找，优先顺序为：

本地仓库 > 私服(如果配置了的话) > 公共仓库(如果配置了的话) > 中央仓库

### 5.3 本地仓库

即在[settings.xml](#) 中配置的目录。

使用过了的依赖都会自动存储在本地仓库中，后续可以复用。

## 5.4 远程仓库

### 5.4.1 中央仓库

- Maven 中央仓库是由 Maven 社区提供的仓库，不用任何配置，maven中内置了中央仓库的地址。其中包含了绝大多数流行的开源Java构件。
- <https://mvnrepository.com/> 可以搜索需要的依赖的相关信息（仓库搜索服务）  
<http://repo.maven.apache.org/maven2/> 中央仓库地址

### 5.4.2 公共仓库【重点】

- 除中央仓库之外，还有其他远程仓库。  
比如aliyun仓库（<http://maven.aliyun.com/nexus/content/groups/public/>）
- 中央仓库在国外，下载依赖速度过慢，所以都会配置一个国内的公共仓库替代中央仓库。

```
<!--setting.xml中添加如下配置-->
<mirrors>
  <mirror>
    <id>aliyun</id>
    <!-- 中心仓库的 mirror(镜像) -->
    <mirrorOf>central</mirrorOf>
    <name>Nexus aliyun</name>
    <!-- aliyun仓库地址 以后所有要指向中心仓库的请求，都会指向aliyun仓库-->
    <url>http://maven.aliyun.com/nexus/content/groups/public</url>
  </mirror>
</mirrors>
```

### 5.4.3 私服【了解】

公司范围内共享的仓库，不对外开放。

可以通过 Nexus来创建、管理一个私服。

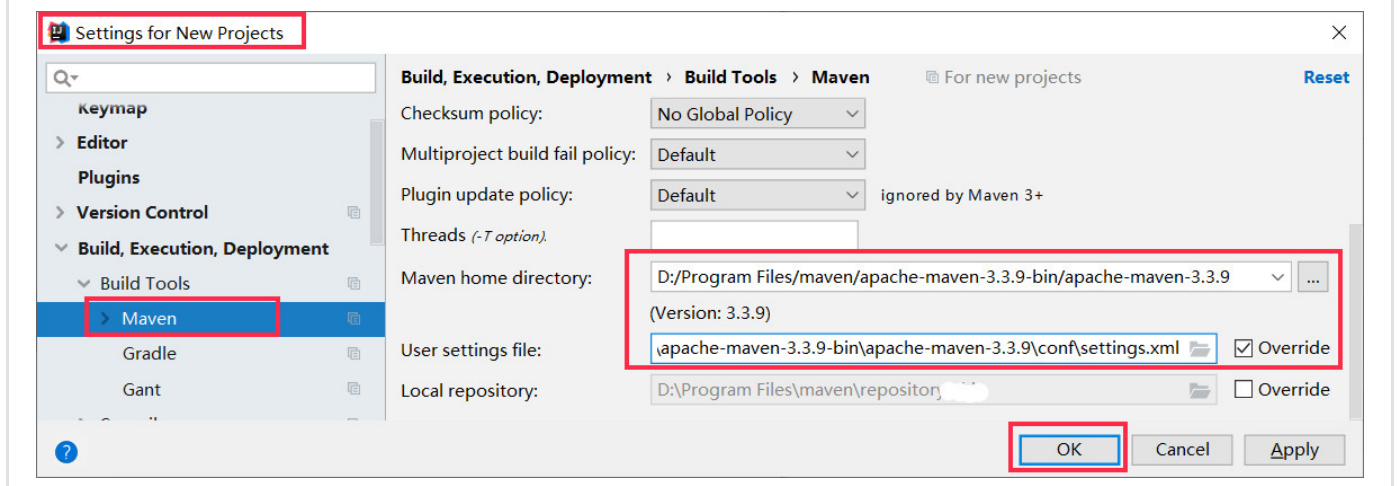
## 6、Idea-Maven

### 6.1 在Idea中关联Maven

在idea中关联本地安装的maven，后续就可以通过idea使用maven，管理项目啦。



## 在全局设置中，关联Maven

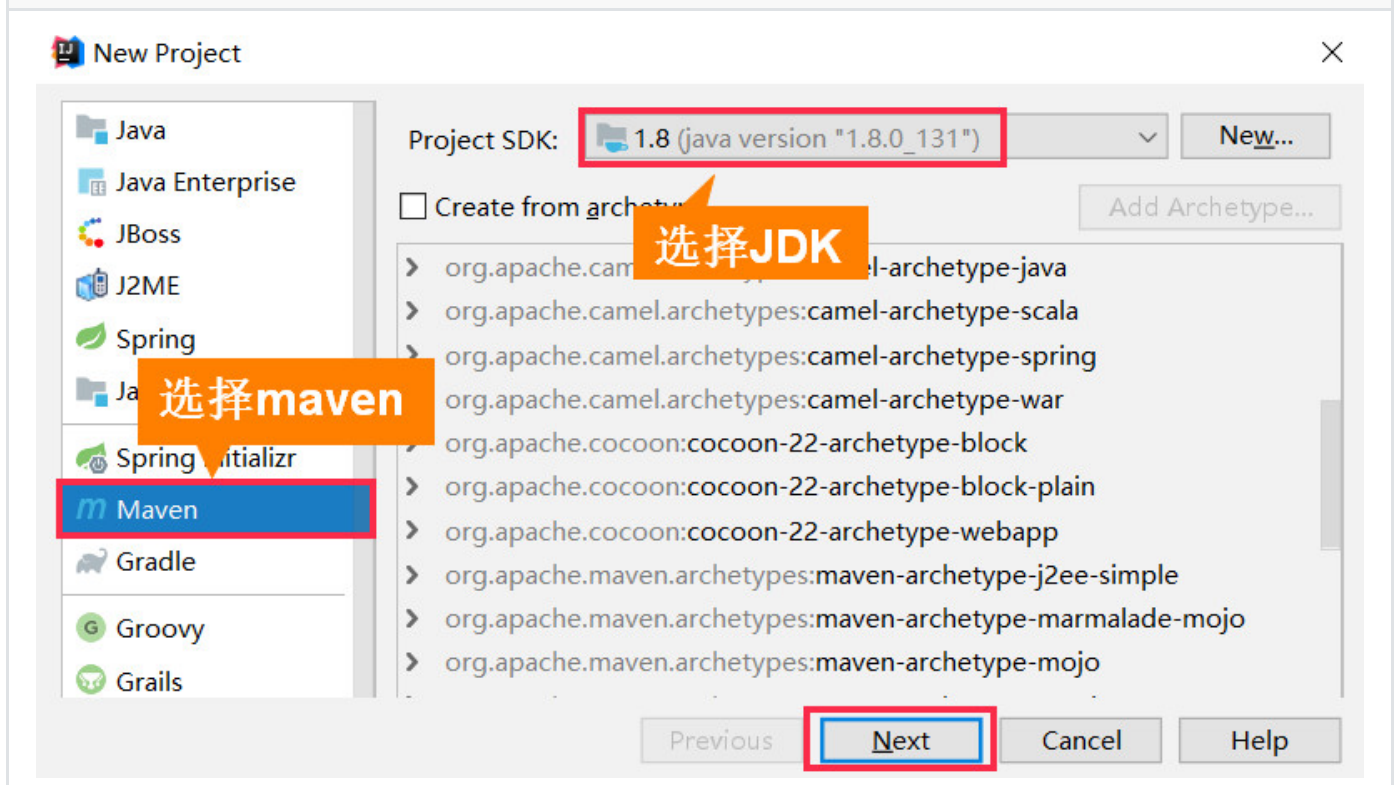


## 6.2 创建Maven项目

### 6.2.1 新建项目

新建项目，要选择 [Maven](#) 选项

### 如下选项



### 6.2.2 指定项目名

设置项目名

New Project

GroupId

com.qf

ArtifactId

test01

Version

1.0-SNAPSHOT

☒ Inherit

☒ Inherit

项目标识

项目名

GAV坐标

项目版本号

Previous

Next

Cancel

Help

### 6.2.3 项目位置

项目位置如下

New Project

Project name:

test01

Project location:

D:\working\code\test01

...

项目名

项目保存位置

More Settings

Module name:

test01

Content root:

D:\working\code\test01

Module file location:

D:\working\code\test01

Project format:

.idea (directory based)

Previous

Finish

Cancel

Help

### 6.2.4 项目结构

- src/main/java 存放源代码，建包，放项目中代码(service,dao,User,...)
- src/main/resources 书写配置文件，项目中的配置文件(jdbc.properties)
- src/test/java 书写测试代码，项目中测试案例代码
- src/test/resources 书写测试案例相关配置文件
- 目根/pom.xml (project object model) maven项目核心文件，其中定义项目构建方式，声明依赖等
- 注意: 项目中的建包，建类，执行，都和普通项目无差异

项目结构如下：



### 6.2.5 项目类型

根据项目类型，在 `pom.xml` 中做出对应配置，添加配置：[war/jar](#)

```
<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.qf</groupId>
  <artifactId>test01</artifactId>
  <version>1.0-SNAPSHOT</version>
  <!-- 打包方式，如果是java项目则用 jar，
        如果是web项目则用war -->
  <!--<packaging>war</packaging>-->
  <packaging>jar</packaging>
</project>
```

## 6.3 导入依赖jar

建好项目后，需要导入需要的jar，要通过[坐标](#)

- 每个构件都有自己的坐标 = groupId + artifactId + version = 项目标识 + 项目名 + 版本号
- 在maven项目中只需要配置坐标，maven便会自动加载对应依赖。删除坐标则会移除依赖

6.3.1 查找依赖

依赖查找服务：<https://mvnrepository.com/>，获得依赖的坐标，在maven项目中导入。

查找依赖坐标

×

🔒

mvnrepository.com/search?q=commons

REPOSITORY

commons

tory

ntal 4.7k

natype 1.5k

ing Plugins 1.0k


ing Lib M 975

enter 256

blio 218

Found 6400 results

Sort: **relevance** | popular | newest



1. **Apache Commons IO**

[commons-io](#) » [commons-io](#)

The Apache Commons IO library c  
classes, and much more.



## Apache Commons IO » 2.6

The Apache Commons IO library contains utility classes, classes, and much more.

License	Apache 2.0
Categories	I/O Utilities
HomePage	<a href="http://commons.apache.org/proper/commons">http://commons.apache.org/proper/commons</a>
Date	(Oct 15, 2017)
Files	<a href="#">pom (13 KB)</a> <a href="#">jar (209 KB)</a> <a href="#">View All</a>
Repositories	Central
Used By	18,323 artifacts

[Maven](#)[Gradle](#)[SBT](#)**依赖坐标**[Leiningen](#)[Buildr](#)

```
<!-- https://mvnrepository.com/artifact/commons-io/commons-io -->
<dependency>
  <groupId>commons-io</groupId>
  <artifactId>commons-io</artifactId>
  <version>2.6</version>
</dependency>
```

### 6.3.2 导入依赖

在项目的pom文件中，增加依赖

在项目的pom.xml文件添加依赖

```
<packaging>war</packaging>
```

```
<dependencies>
```

增加依赖

```
<!-- https://mvnrepository.com/artifact/commons-io/commons-io -->
```

```
<dependency>
```

```
<groupId>commons-io</groupId>
```

```
<artifactId>commons-io</artifactId>
```

```
<version>2.6</version>
```

```
</dependency>
```

```
</dependencies>
```

```
</project>
```

### 6.3.3 同步依赖

引入坐标后，同步依赖，确认导入。

窗口右下角弹窗，刷新依赖，使新加的配置被maven加载

手动刷新依赖

自动刷新依赖



**Maven projects need to be imported**

Import Changes

Enable Auto-Import



Event Log



JRebel Console

12:11

LF

UTF-8

4 spaces



## 6.4 创建web项目

### 6.4.1 打包方式

pom.xml中设置 [war](#)



web项目打包方式为: [war](#)

```
<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.qf</groupId>
  <artifactId>test01</artifactId>
  <version>1.0-SNAPSHOT</version>

  <packaging>war</packaging>

  </project>
```

web项目

#### 6.4.2 web依赖

导入 [JSP](#) 和 [Servlet](#) 和 [JSTL](#) 依赖, 使项目具有web编译环境

```
<?xml version="1.0" encoding="UTF-8"?>
<project ...>
  ...
  <packaging>war</packaging>

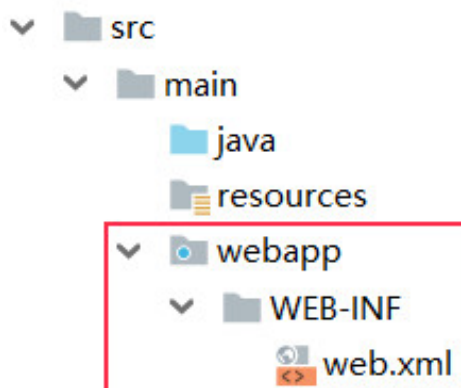
  <!-- 导入JSP 和 Servlet 和 JSTL 依赖 -->
  <dependencies>
    <dependency>
      <!-- jstl 支持 -->
      <groupId>javax.servlet</groupId>
      <artifactId>jstl</artifactId>
      <version>1.2</version>
    </dependency>
    <dependency>
      <!-- servlet编译环境 -->
      <groupId>javax.servlet</groupId>
      <artifactId>javax.servlet-api</artifactId>
      <version>3.1.0</version>
      <scope>provided</scope>
    </dependency>
    <dependency>
      <!-- jsp编译环境 -->
      <groupId>javax.servlet</groupId>
      <artifactId>jsp-api</artifactId>
```

```
<version>2.0</version>
<scope>provided</scope>
</dependency>
</dependencies>
</project>
```

### 6.4.3 webapp目录

按照maven规范，新建web项目特有目录

新建如下目录和文件

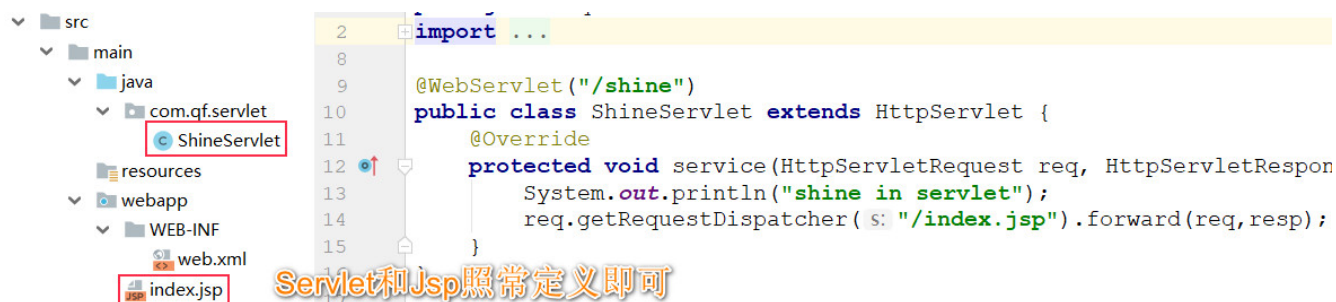


新建如上两级目录和web.xml文件

```
<?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">
  <!-- 这是一个空白的web.xml文件模板 -->
</web-app>
```

### 6.4.4 定义Servlet和Jsp

照常定义即可

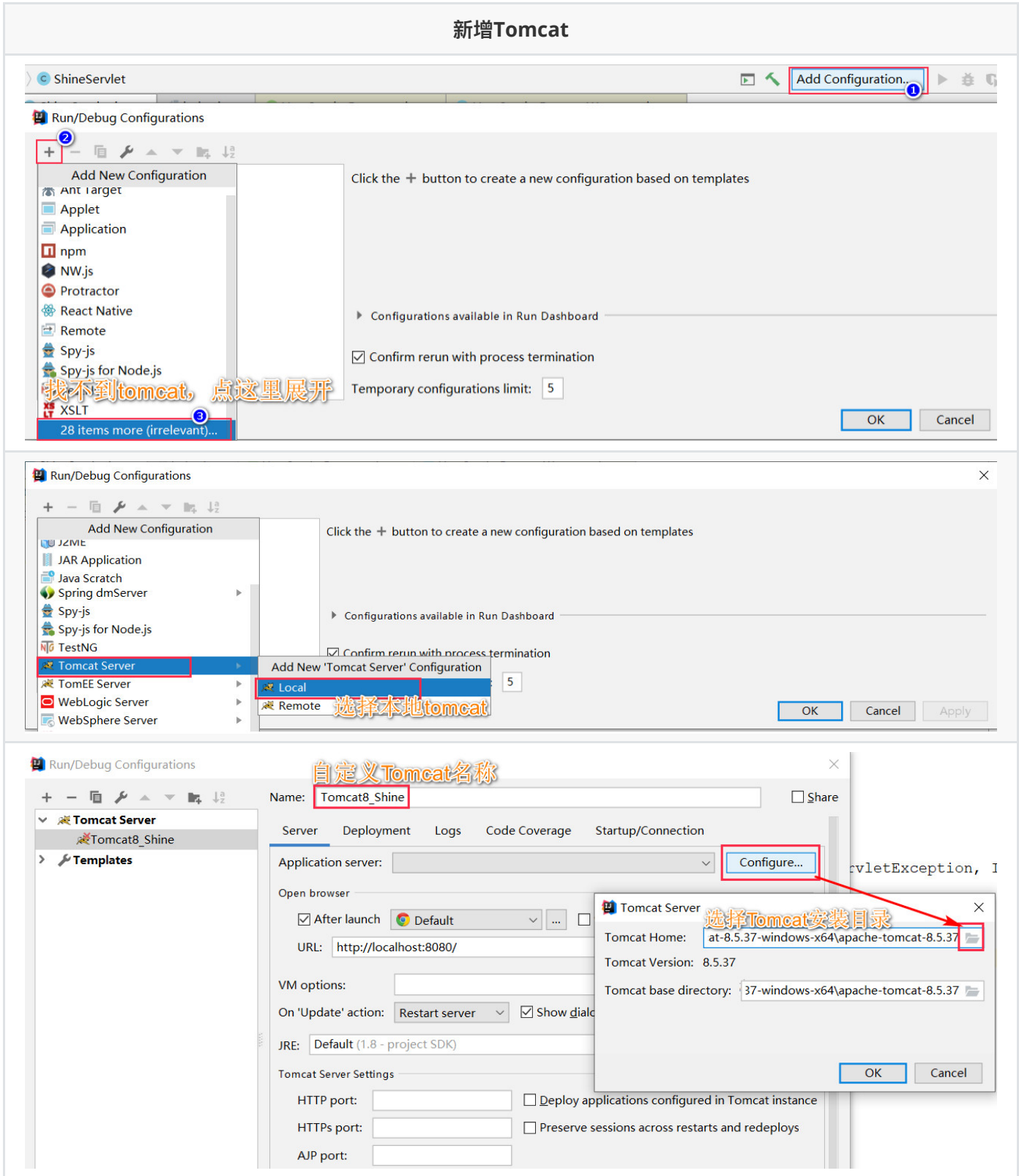


Servlet和Jsp照常定义即可

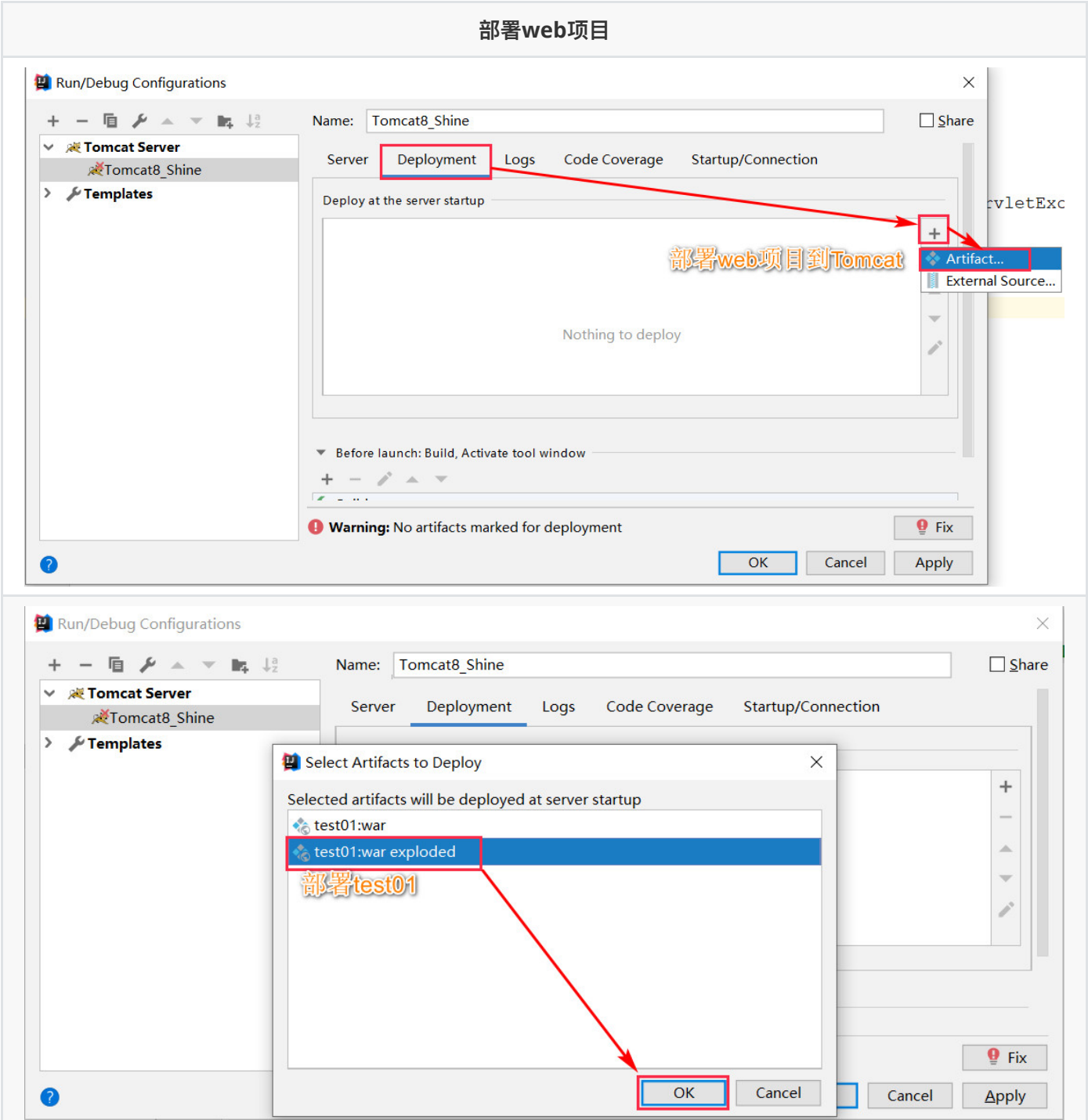


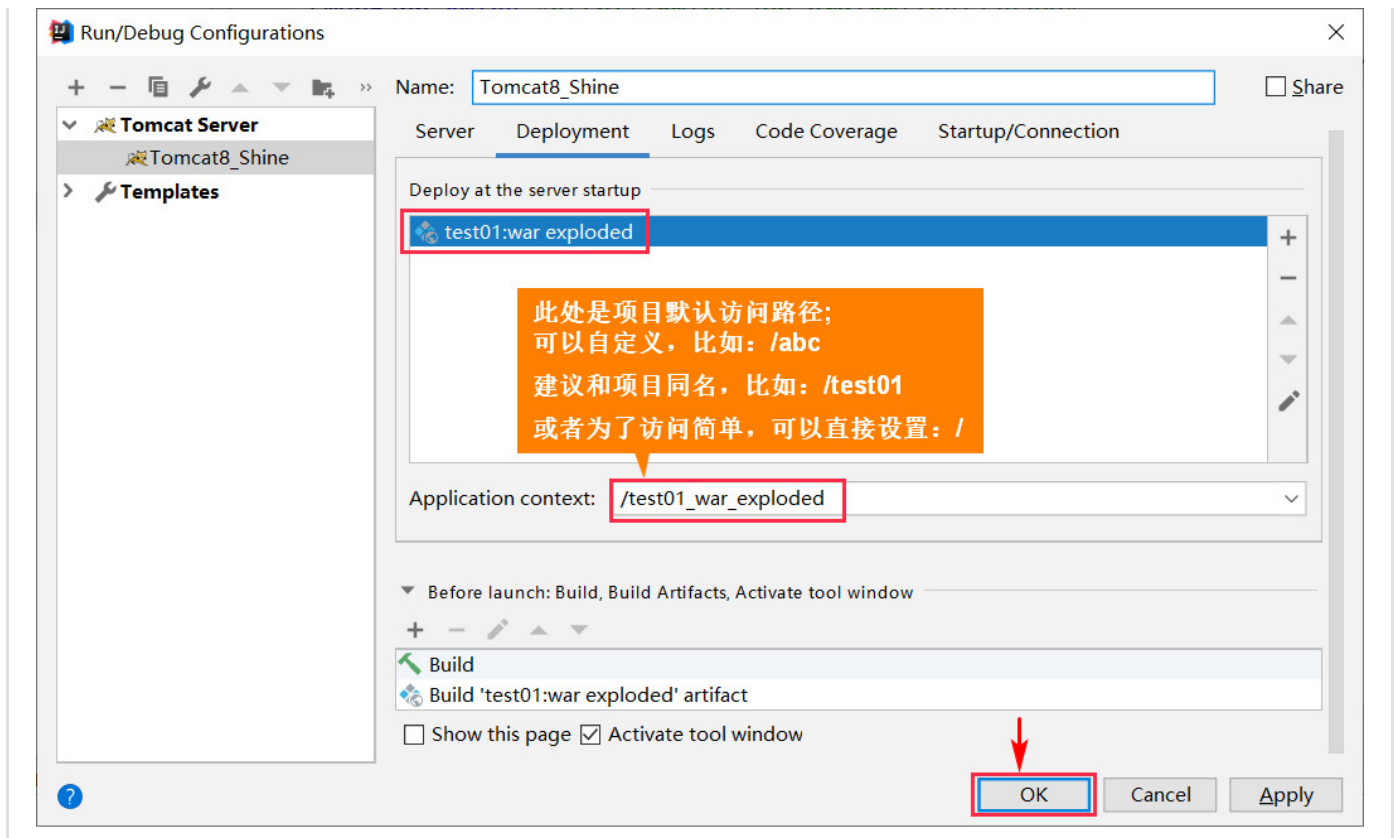
## 6.5 部署web项目

### 6.5.1 新增Tomcat

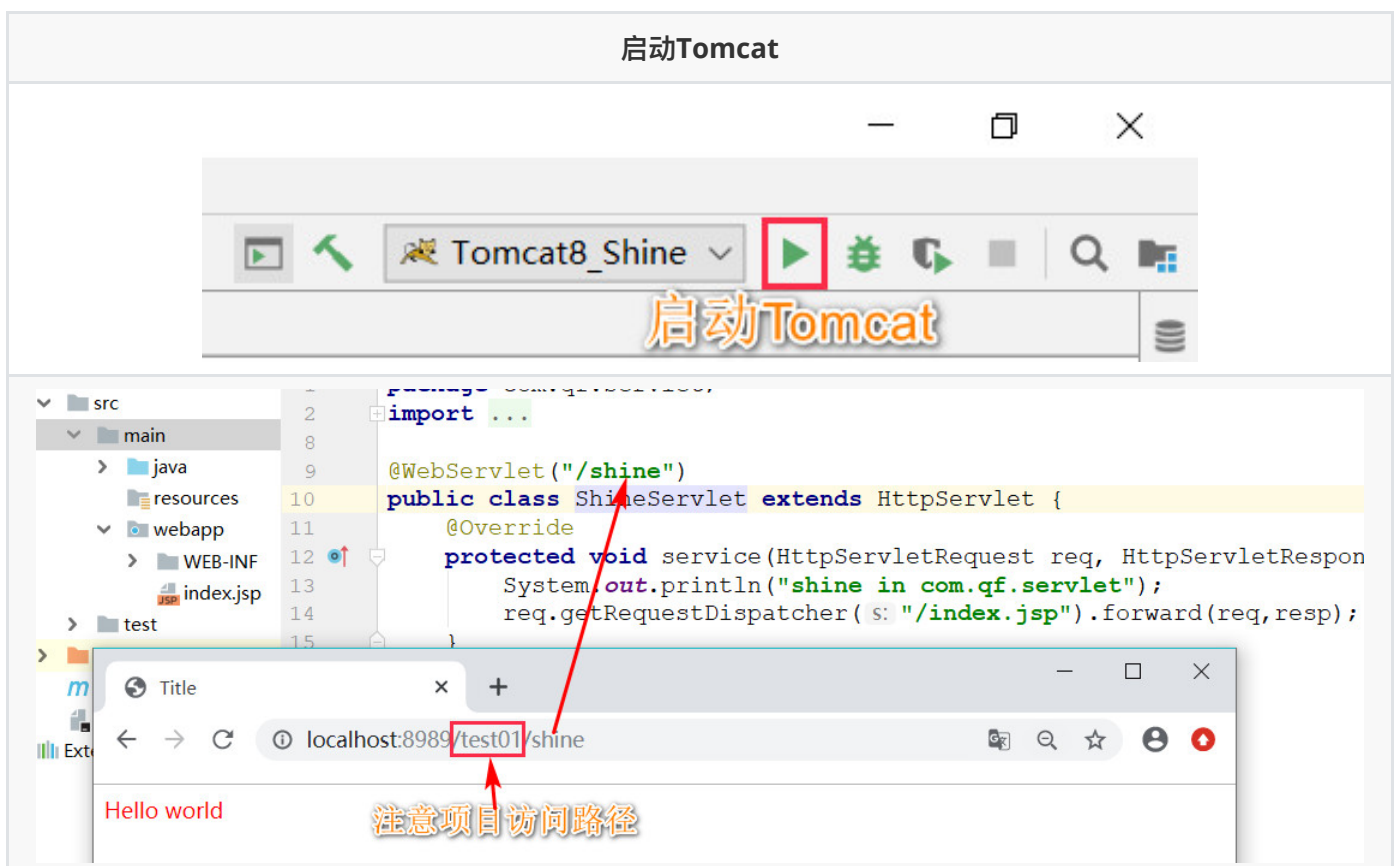


6.5.2 部署web项目





### 6.5.3 启动Tomcat



## 6.6 依赖生命周期

### 6.6.1 概念

Jar包生效的时间段，即Jar的生命周期

### 6.6.2 使用方式

项目中导入的依赖可以做生命周期的管理

```
<dependency>
  <groupId>commons-io</groupId>
  <artifactId>commons-io</artifactId>
  <version>2.6</version>
  <!-- 生命周期 -->
  <scope>compile</scope>
</dependency>
<dependency>
  <!-- servlet编译环境 -->
  <groupId>javax.servlet</groupId>
  <artifactId>javax.servlet-api</artifactId>
  <version>3.1.0</version>
  <!-- 生命周期 -->
  <scope>provided</scope>
</dependency>
<dependency>
  <groupId>junit</groupId>
  <artifactId>junit</artifactId>
  <version>4.12</version>
  <!-- 生命周期 -->
  <scope>test</scope>
</dependency>
```

### 6.6.3 生命周期详解

标识	周期
compile	缺省值，适用于所有阶段（测试运行，编译，运行，打包）
provided	类似compile，期望JDK、容器或使用后会提供这个依赖。如servlet-api.jar；适用于（测试运行，编译）阶段
runtime	只在运行时使用，如mysql的驱动jar，适用于（运行，测试运行）阶段
test	只在测试时使用，适用于（编译，测试运行）阶段，如junit.jar
system	Maven不会在仓库中查找对应依赖，在本地磁盘目录中查找；适用于（编译，测试运行，运行）阶段

# 七、Maven指令

## 7.1 命令行

通过Idea打开 [cmd](#)，然后执行Maven指令

打开 cmd，并定位到项目目录

test01

项目上  
右键

CutCtrl+X

CopyCtrl+C

Copy PathCtrl+Shift+C

Copy Relative PathCtrl+Alt+Shift+C

PasteCtrl+V

Build Module 'test01'

Rebuild Module 'test01'Ctrl+Shift+F9

Show in Explorer

Open in Terminal

执行maven指令

Terminal: Local × +

D:\working\code\test01>mvn clean

[INFO] Scanning for projects

执行clean指令

[INFO]

[INFO] -----

[INFO] Building test01 1.0-SNAPSHOT

[INFO] -----

[INFO]

[INFO] --- maven-clean-plugin:2.5:clean (default)

[INFO] -----

[INFO] BUILD SUCCESS

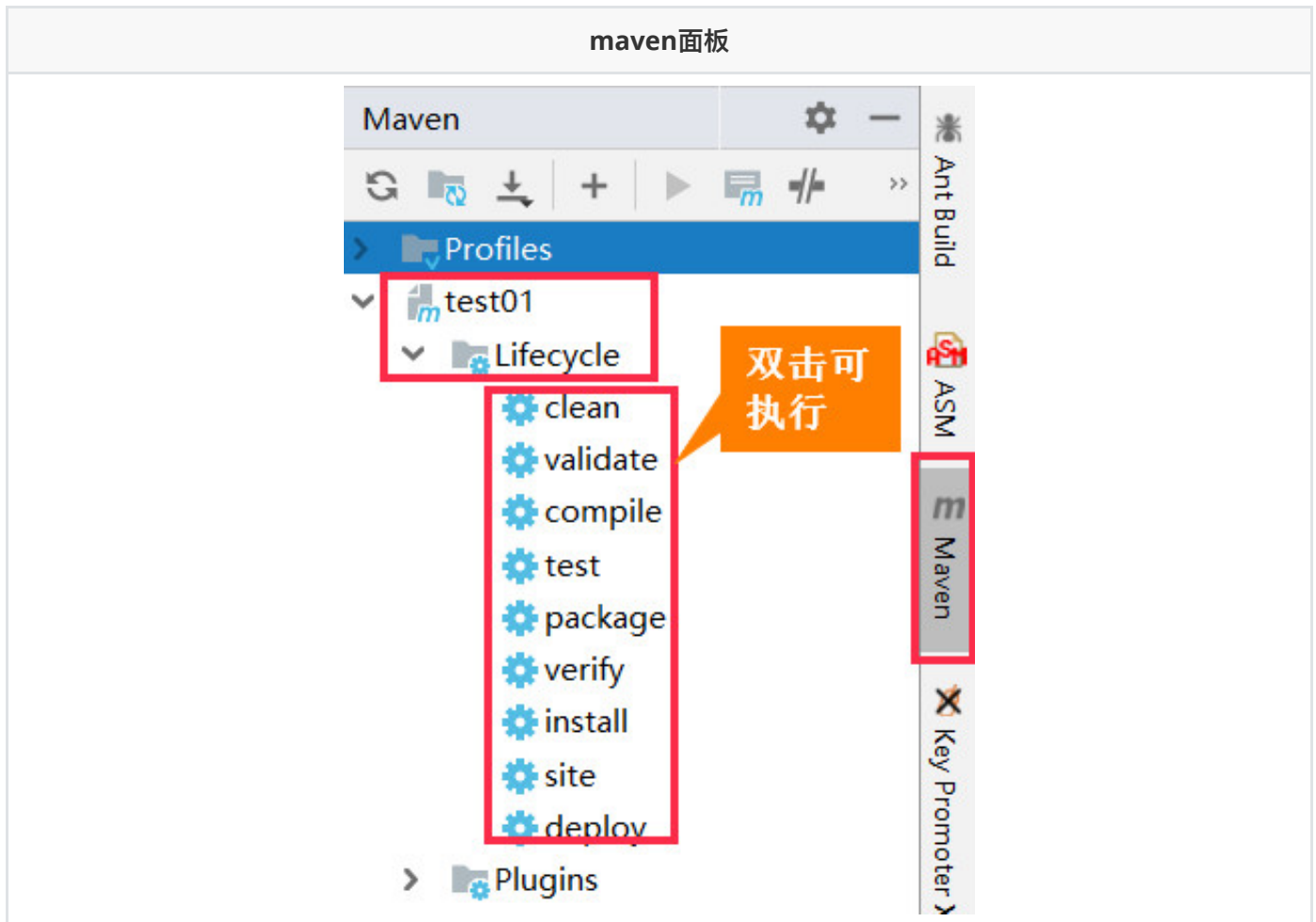
[INFO] -----

[INFO] Total time: 0.162 s

- -

## 7.2 Maven面板

Idea中有Maven面板，其中可以快速执行Maven指令

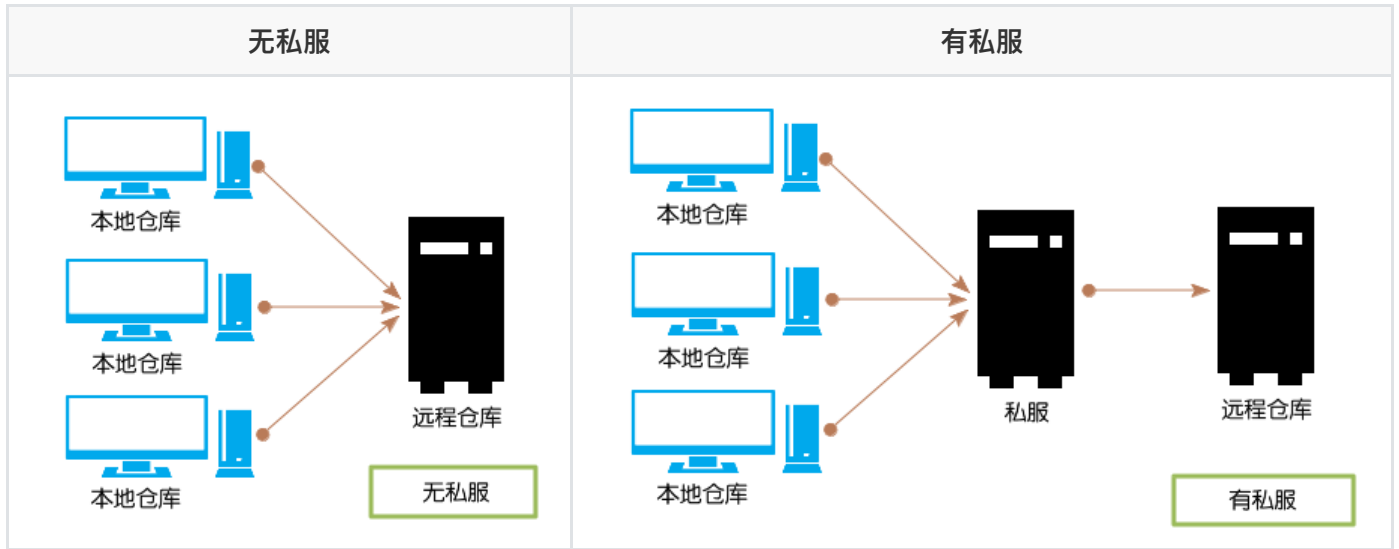


## 八、私服

### 8.1 概念

- 私服是架设在局域网的一种特殊的远程仓库，目的是代理远程仓库及部署第三方构件。
- 有了私服之后，当 Maven 需要下载依赖时，直接请求私服，私服上存在则下载到本地仓库；否则，私服请求外部的远程仓库，将构件下载到私服，再提供给本地仓库下载。
- 私服可以解决在企业做开发时每次需要的jar包都要在中心仓库下载,且每次下载完只能被自己使用,不能被其他开发人员使用
- 所谓私服就是一个服务器,但是不是本地层面的,是公司层面的,公司中所有的开发人员都在使用同一个私服

### 8.2 架构



我们可以使用专门的 Maven 仓库管理软件来搭建私服，比如：[Apache Archiva](#)，[Artifactory](#)，[Sonatype Nexus](#)。这里我们使用 [Sonatype Nexus](#)