

# JavaEE平台技术

## Maven和SpringBoot

邱明 博士

厦门大学信息学院

mingqiu@xmu.edu.cn

# 提纲

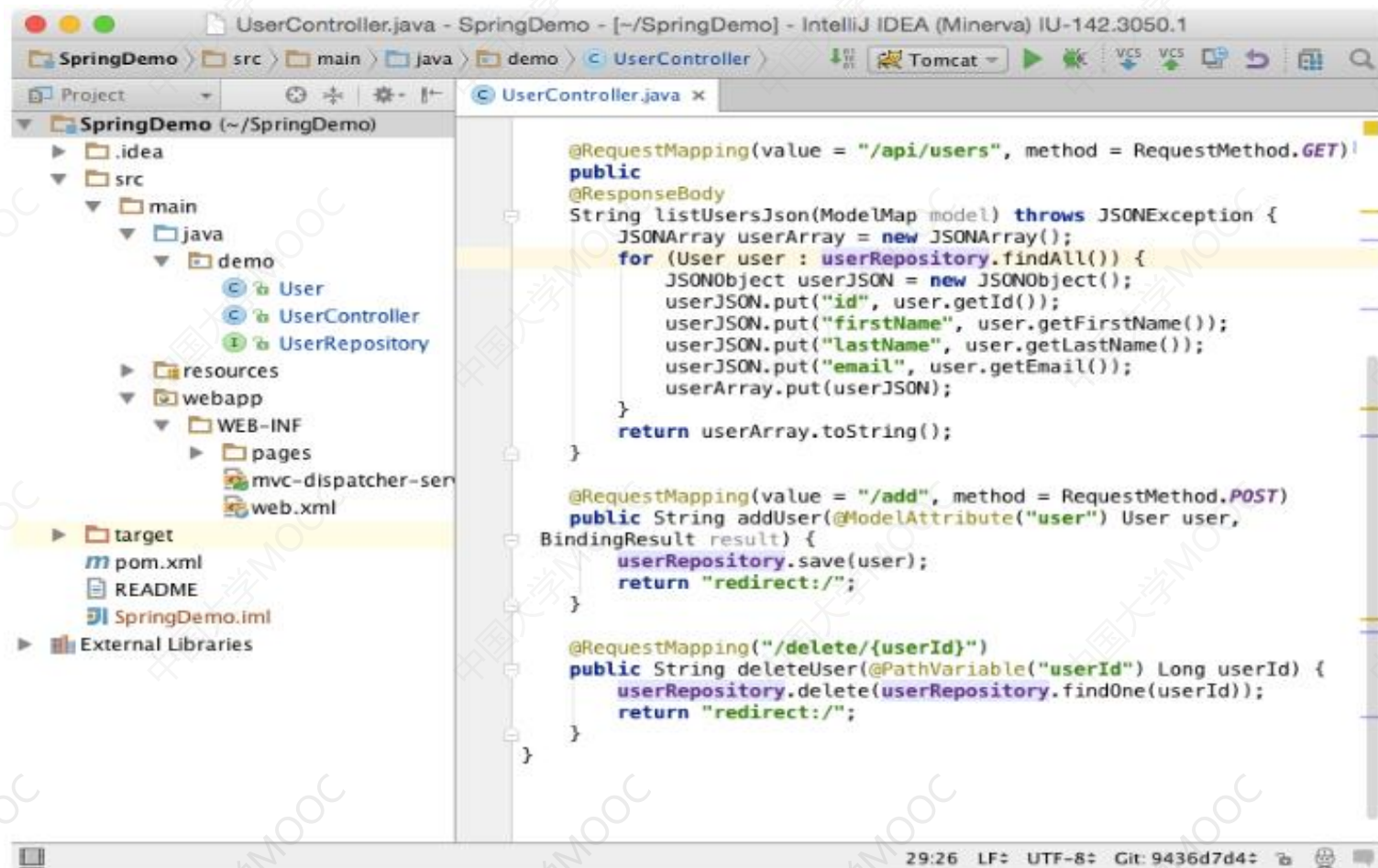
- IntelliJ IDEA
- Maven

# 1. IntelliJ IDEA

- IntelliJ IDEA

- IntelliJ IDEA（简称 IDEA），是 Java 语言开发的集成环境，IDEA 在业界被公认为最好的 Java 开发工具之一，尤其在智能代码助手、代码自动提示、重构、各类版本工具（Git、SVN、GitHub 等）、JUnit、CVS 整合、代码分析和创新的 GUI 设计等方面的功能都值得称道。

# 1. IntelliJ IDEA



## 2. Maven

- Apache 下的一个纯 Java 开发的开源项目。基于项目对象模型（缩写：POM）管理一个项目的构建、依赖、报告和文档等步骤。
  - Maven是一个构建工具，实现自动化构建。
  - Maven是跨平台的，对外提供一致的操作接口。
  - Maven是一个依赖管理工具和项目信息管理工具。它还提供了中央仓库,能帮我们自动下载构件。
  - Maven是一个标准，对于项目目录结构、测试用例命名方式等内容都有既定的规则。



## 2. Maven

- Maven提倡使用一个共同的标准目录结构

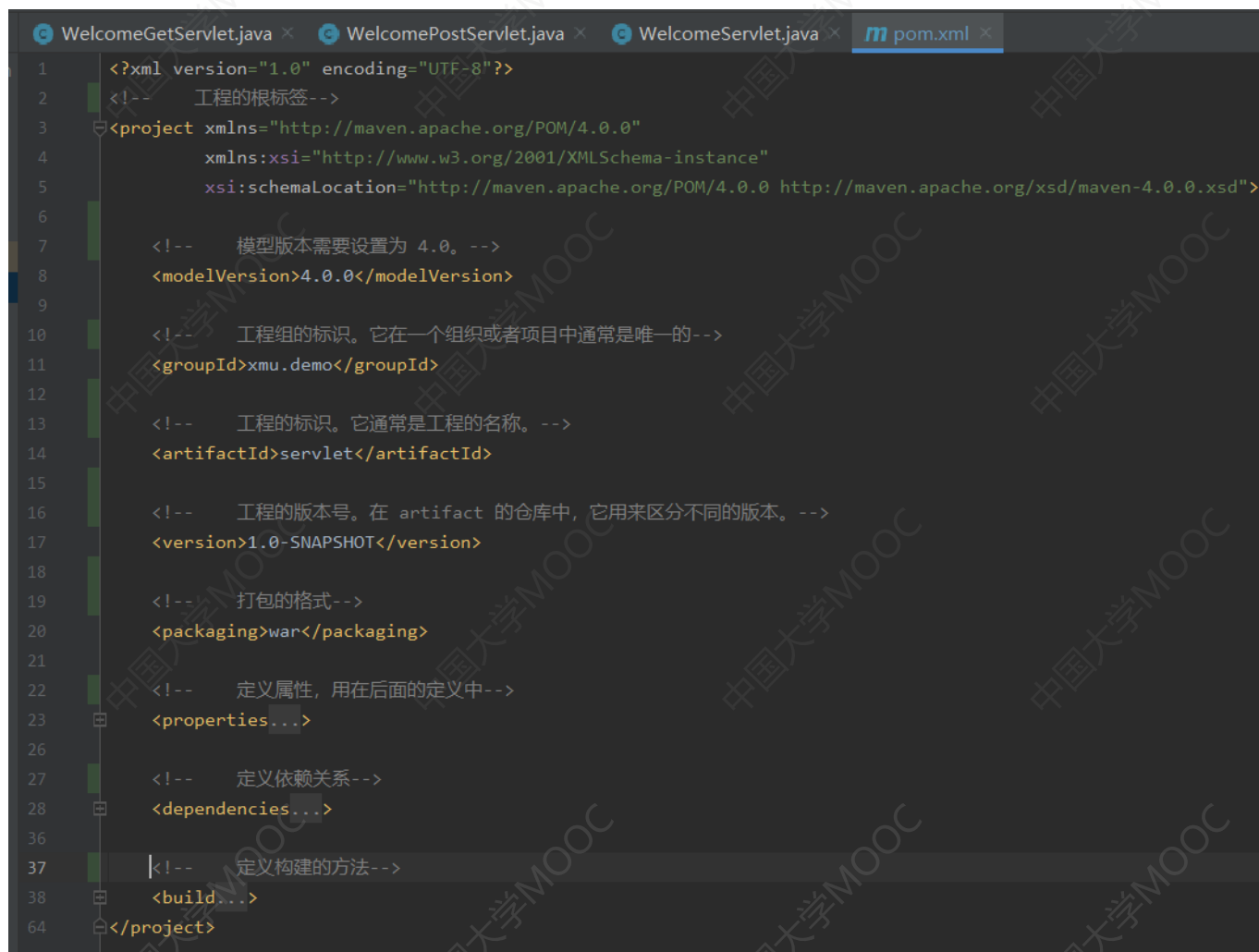
目录	用途
<code>\${basedir}</code>	存放pom.xml和所有的子目录
<code>\${basedir}/src/main/java</code>	项目的java源代码
<code>\${basedir}/src/main/resources</code>	项目的资源，比如说property文件，springmvc.xml
<code>\${basedir}/src/test/java</code>	项目的测试类，比如说JUnit代码
<code>\${basedir}/src/test/resources</code>	测试用的资源
<code>\${basedir}/src/main/webapp/WEB-INF</code>	web应用文件目录，web项目的信息，比如存放web.xml、本地图片、jsp视图页面
<code>\${basedir}/target</code>	打包输出目录
<code>\${basedir}/target/classes</code>	编译输出目录
<code>\${basedir}/target/test-classes</code>	测试编译输出目录
Test.java	Maven只会自动运行符合该命名规则的测试类
<code>~/.m2/repository</code>	Maven默认的本地仓库目录位置

## 2. Maven

- Maven POM

- POM( Project Object Model, 项目对象模型 ) 是一个XML文件, 包含了项目的基本信息, 用于描述项目如何构建, 声明项目依赖, 等等。
- Maven 会在当前目录中查找 并读取POM文件, 获取所需的配置信息, 然后执行目标。
- POM 中可以指定以下配置:
  - 项目依赖
  - 插件
  - 执行目标
  - 项目构建 profile
  - 项目版本
  - 项目开发者列表
  - 相关邮件列表信息

## 2. Maven



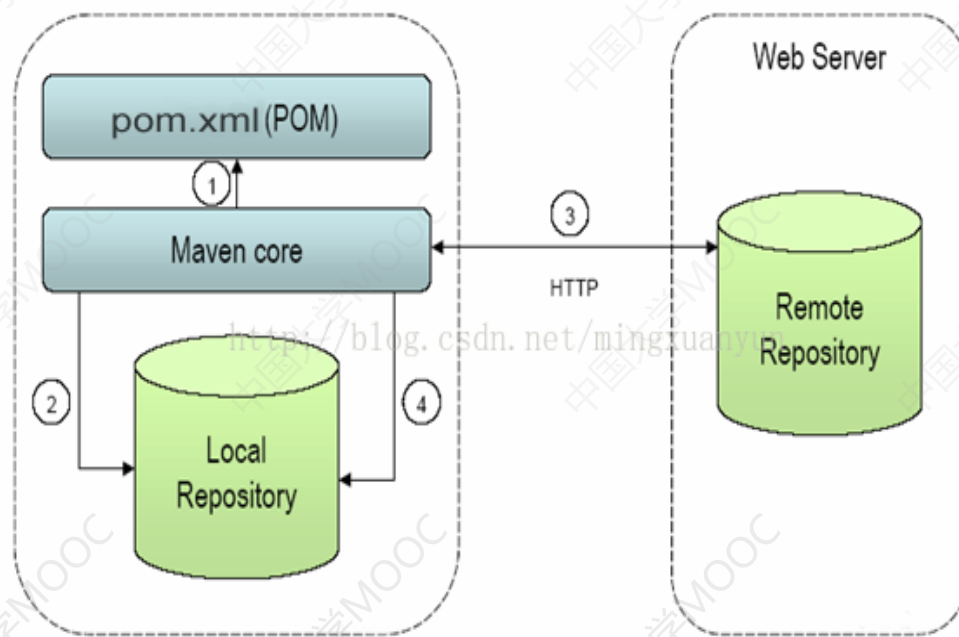
The screenshot shows a code editor with four tabs: WelcomeGetServlet.java, WelcomePostServlet.java, WelcomeServlet.java, and pom.xml. The pom.xml tab is active, displaying the following XML content with Chinese comments:

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!-- 工程的根标签-->
3 <project xmlns="http://maven.apache.org/POM/4.0.0"
4     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5     xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
6
7     <!-- 模型版本需要设置为 4.0。-->
8     <modelVersion>4.0.0</modelVersion>
9
10    <!-- 工程组的标识。它在一个组织或者项目中通常是唯一的-->
11    <groupId>xmu.demo</groupId>
12
13    <!-- 工程的标识。它通常是工程的名称。-->
14    <artifactId>servlet</artifactId>
15
16    <!-- 工程的版本号。在 artifact 的仓库中，它用来区分不同的版本。-->
17    <version>1.0-SNAPSHOT</version>
18
19    <!-- 打包的格式-->
20    <packaging>war</packaging>
21
22    <!-- 定义属性，用在后面的定义中-->
23    <properties...>
24
25
26
27    <!-- 定义依赖关系-->
28    <dependencies...>
29
30
31
32
33
34
35
36
37    <!-- 定义构建的方法-->
38    <build...>
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54 </project>
```



## 2. Maven

- Maven会自动根据dependency中的依赖配置，直接通过互联网在Maven中央仓库（<https://mvnrepository.com/>）下载相关依赖包到本地Maven库，本地Maven库默认是用户目录的.m2目录



## 2. Maven

- Maven 的三个标准生命周期：
  - clean: 项目构建前的清理工作，删除前一次构建在target文件夹下生成的各个Jar包等
  - default: 构建，包括项目的编译，测试，打包，安装，部署等等
  - site: 生成项目报告，发布站点，Maven可以根据pom所包含的信息，生成一个站点，方便团队交流和发布项目信息，

## 2. Maven

- Maven的清理(Clean)生命周期
  - pre-clean: 执行一些清理前需要完成的工作
  - clean: 清理上一次构建生成的文件
  - post-clean: 执行一些清理后需要完成的工作

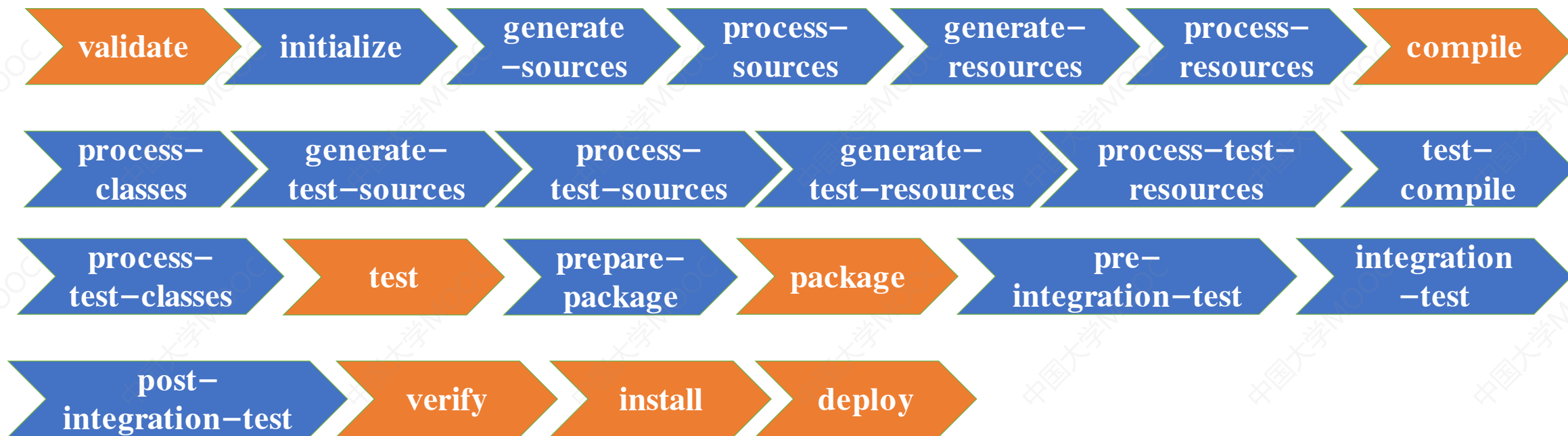


## 2. Maven

- Maven的构建(Default)生命周期
  - validate: 验证项目是否正确且所有必需资源是否可用
  - initialize:初始化编译的状态, 例如: 设置一些properties属性, 或者创建一些目录
  - generate-sources:通常是通过插件支持创建额外的源代码。
  - process-sources:处理源代码, 例如: 替换值 (filter any values)
  - generate-resources:生成这个项目包所有需要包含的资源文件
  - process-resources:复制并处理资源文件到目标目录, 为packaging 打包阶段做好准备
  - compile: 编译项目的源码
  - process-classes:后置处理编译阶段生成的文件, 例如: 做java字节码的增强操作
  - generate-test-sources:生成编译阶段需要的test源代码
  - process-test-sources:处理test源代码,
  - generate-test-resources:生成test测试需要的资源文件
  - process-test-resources:复制并处理资源文件到test测试目标目录
  - test-compile:编译项目单元测试代码
  - process-test-classes:后置处理test编译阶段生成的文件, 例如: 做java字节码的增强操作
  - test: 使用单元测试框架运行测试, 测试代码不会被打包或部署
  - prepare-package:处理任何需要在正式打包之前要完成的必须的准备工作。这一步的通常结果是解压, 处理包版本等
  - package: 创建JAR/WAR包如在 pom.xml 中定义提及的包
  - pre-integration-test:完成一些在集成测试之前需要做的预处理操作, 这通常包括建立需要的环境。
  - integration-test:处理并部署 (deploy) 包到集成测试可以运行的环境中执行系统集成测试。
  - post-integration-test:处理一些集成测试之后的事情, 通常包括一些环境的清理工作
  - verify: 对集成测试的结果进行检查, 以保证质量达标
  - install: 安装打包的项目到本地仓库, 以供其他项目使用
  - deploy: 拷贝最终的工程包到远程仓库中, 以共享给其他开发人员和工程

## 2. Maven

- Maven的构建(Default)生命周期



## 2. Maven

- Maven的站点(Site)生命周期
  - pre-site: 执行一些实际站点生成之前的预处理操作
  - site: 生成项目站点文档
  - post-site: 执行一些后置操作并完成最终生成站点操作, 并为最后站点发布做好准备
  - site-deploy: 将生成的项目站点发布到服务器上



## 2. Maven

- Maven 插件

- Maven 实际上是一个依赖插件执行的框架，每个任务实际上是由插件完成。每个插件可以完成多个功能，每个供称为插件目标（Plugin Goal）。

插件目标	描述
spring-boot:run	运行Spring Boot应用
spring-boot:repackage	重新打包jar/war包为可执行包
spring-boot:help	展示spring-boot-maven-plugin的帮助信息
spring-boot:start	启动Spring应用程序,和run目标不同，该目标不会阻塞，并且允许其他目标来操作应用程序
spring-boot:stop	停止使用start目标启动的spring应用程序，通常在测试套件完成后被调用

## 2. Maven

- 插件与阶段的内置绑定
  - Clean生命周期

阶段	内置绑定的插件目标
clean	maven-clean-plugin:clean



## 2. Maven

- 插件与阶段的内置绑定
  - Default生命周期（当packaging的值是jar/war）

阶段	内置绑定的插件目标
process-resource	maven-resources-plugin:resources
compiler	maven-compiler-plugin:compile
process-test-resources	maven-resources-plugin:testResources
test-compile	maven-compiler-plugin:testCompile
test	maven-surefire-plugin:test
package	maven-jar-plugin:jar/maven-war-plugin:war
install	maven-install-plugin:install
deploy	maven-deploy-plugin:deploy

## 2. Maven

- 插件与阶段的内置绑定
  - Default生命周期（当packaging的值是pom）

阶段	内置绑定的插件目标
install	maven-install-plugin:install
deploy	maven-deploy-plugin:deploy

## 2. Maven

- 插件与阶段的内置绑定
  - site生命周期

阶段	内置绑定的插件目标
site	maven-site-plugin:site
site-deploy	maven-site-plugin:deploy

## 2. Maven

- 父 (Super) POM
  - 父 (Super) POM是 Maven 默认的 POM。所有的 POM 都继承自一个父 POM (无论是否显式定义了这个父 POM)。父 POM 包含了一些可以被继承的默认设置
  - 可以用Show Effective POM看到最终有效的POM定义

## 2. Maven

- 父子项目

1. +- pom.xml
2. +- my-app
3. | +- pom.xml
4. | +- src
5. | +- main
6. | +- java
7. +- my-webapp
8. | +- pom.xml
9. | +- src
10. | +- main
11. | +- webapp

```
1.<project xmlns="http://maven.apache.org/POM/4.0.0"
2.  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3.  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
4.                      http://maven.apache.org/xsd/maven-4.0.0.xsd">
5.  <modelVersion>4.0.0</modelVersion>
6.
7.  <groupId>com.mycompany.app</groupId>
8.  <artifactId>app</artifactId>
9.  <version>1.0-SNAPSHOT</version>
10. <packaging>pom</packaging>
11.
12. <modules>
13.   <module>my-app</module>
14.   <module>my-webapp</module>
15. </modules>
16.</project>
```

# 3. SpringBoot是什么

- SpringBoot的作用的是方便开发独立的应用程序
  - 内嵌Tomcat、Jetty或Undertow
  - 采用Starter POM简化Maven的配置
  - 大量采用约定简化Spring的配置
  - 提供产品级的运行监控功能

# 3. SpringBoot是什么

- 系统要求

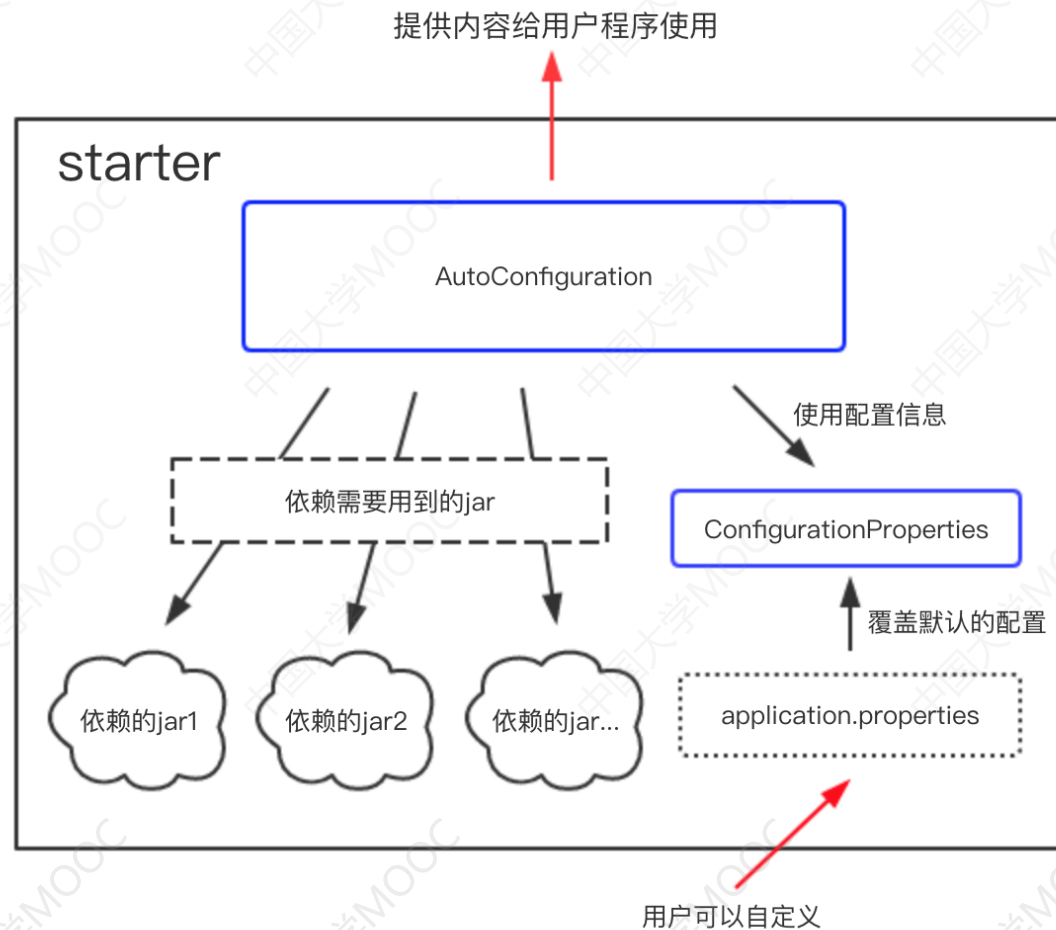
环境	版本
Java	8+
Spring Framework	5.0.0+
Maven	3.2+
Tomcat	8.5+ (Servlet 3.1)
Jetty	9.4+ (Servlet 3.1)
Undertow	1.3+ (Servlet 3.1)

# 3. SpringBoot是什么

- Starter
  - 没有Starter之前
    - 在Maven中引入使用的库
    - 引入使用的库所依赖的库
    - 在xxx.xml中配置一些属性信息
    - 反复的调试直到可以正常运行
  - 有了Starter
    - 只需要引入一个Starter
    - starter会把所有用到的依赖都给包含进来，避免了开发者自己去引入依赖所带来的麻烦



### 3. SpringBoot是什么



### 3. SpringBoot是什么

- 在POM文件中定义继承Spring-boot-starter-parent

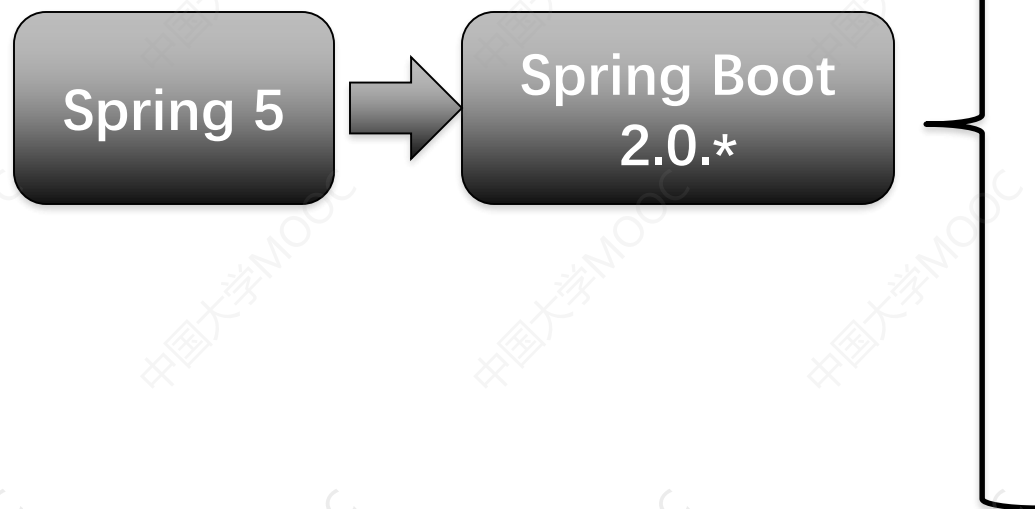
```
<parent>  
  <groupId>org.springframework.boot</groupId>  
  <artifactId>spring-boot-starter-parent</artifactId>  
  <version>2.1.6.RELEASE</version>  
  <relativePath/> <!-- lookup parent from repository -->  
</parent>
```

### 3. SpringBoot是什么

- 在插件中采用SpringBoot的插件来编译打包应用

```
<build>
  <plugins>
    <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
    </plugin>
  </plugins>
</build>
```

# 3. SpringBoot是什么



spring-boot-starter	Core starter, including auto-configuration support, logging and YAML
spring-boot-starter-aop	Starter for aspect-oriented programming with Spring AOP and AspectJ
spring-boot-starter-cache	Starter for using Spring Framework's caching support
spring-boot-starter-data-redis	Starter for using Redis key-value data store with Spring Data Redis and the Lettuce client
spring-boot-starter-freemarker	Starter for building MVC web applications using FreeMarker views
spring-boot-starter-quartz	Spring Boot Quartz Starter
spring-boot-starter-security	Starter for using Spring Security
spring-boot-starter-test	Starter for testing Spring Boot applications with libraries including JUnit, Hamcrest and Mockito
spring-boot-starter-web	Starter for building web, including RESTful, applications using Spring MVC. Uses Tomcat as the default embedded container

## 4. SpringBoot的配置

- Spring Boot应用的Main函数

```
@SpringBootApplication
```

```
public class DemoApplication {
```

```
    public static void main(String[] args) {
```

```
        SpringApplication.run(DemoApplication.class, args);
```

```
    }
```

```
}
```

# 4. SpringBoot的配置

- @SpringBootApplication是一个复合注解

@Target(ElementType.TYPE)

@Retention(RetentionPolicy.RUNTIME)

@Documented

@Inherited

@Configuration

@EnableAutoConfiguration

@ComponentScan

@interface

- @Configuration 注解，实现配置文件的功能。
- @EnableAutoConfiguration：打开自动配置的功能，也可以关闭某个自动配置的选项，如关闭数据源自动配置功能：  
@SpringBootApplication(exclude = { DataSourceAutoConfiguration.class })。
- @ComponentScan：Spring组件扫描。

## 4. SpringBoot的配置

- SpringBoot使用一个全局的配置文件application.properties或者application.yml(或者是yaml)
  - 作用是修改SpringBoot自动配置的默认值；

<https://docs.spring.io/spring-boot/docs/current/reference/html/common-application-properties.html>

## 4. SpringBoot的配置

- 例如修改Servlet容器的监听端口：

- 在application.yml中定义

```
server:  
  port: 9090
```

- 在Java的命令中定义

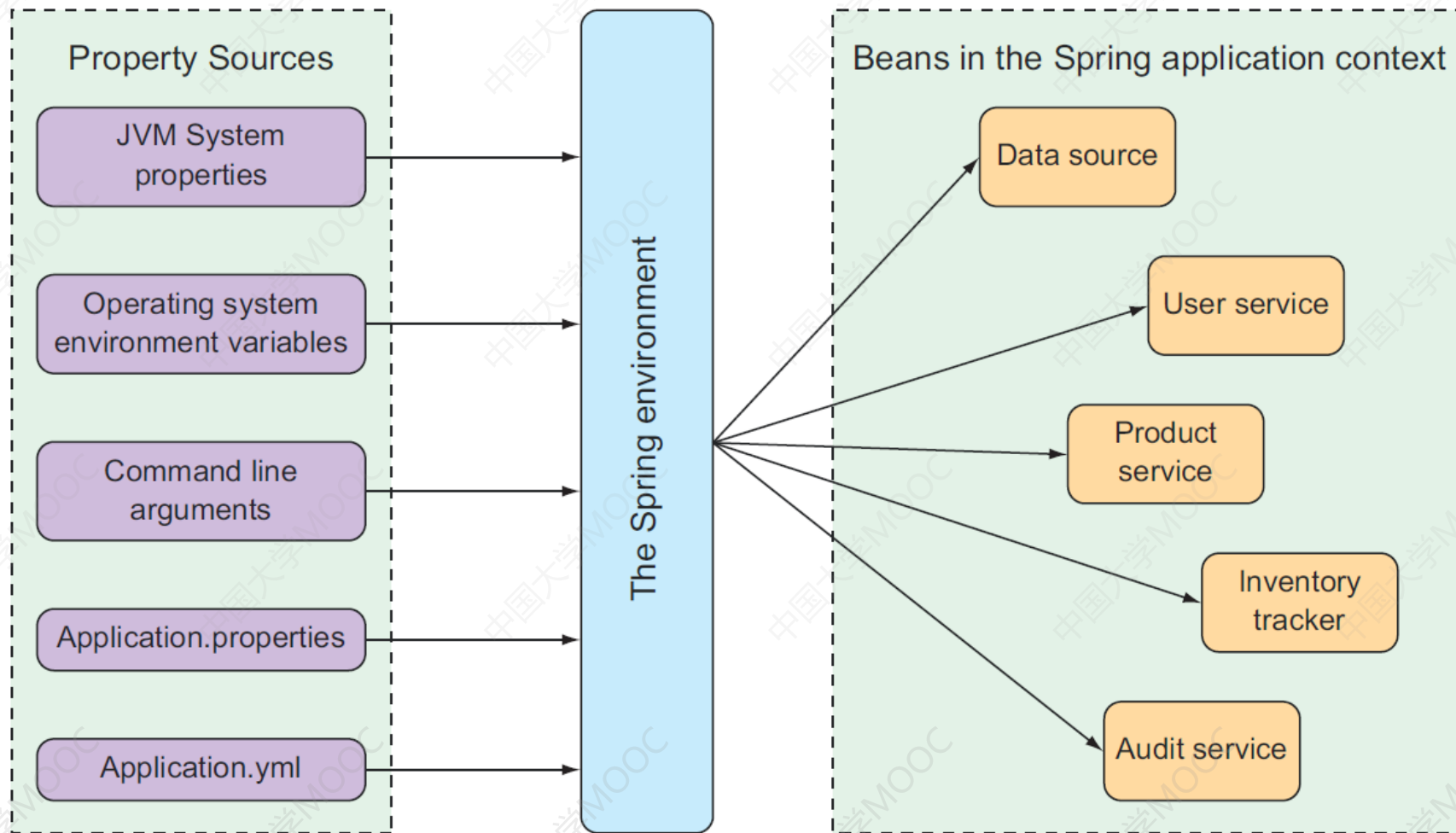
```
$ java -jar RestfulDemo-0.0.1-SNAPSHOT.jar --server.port=9090
```

- 在环境变量中定义

```
$ export SERVER_PORT=9090
```

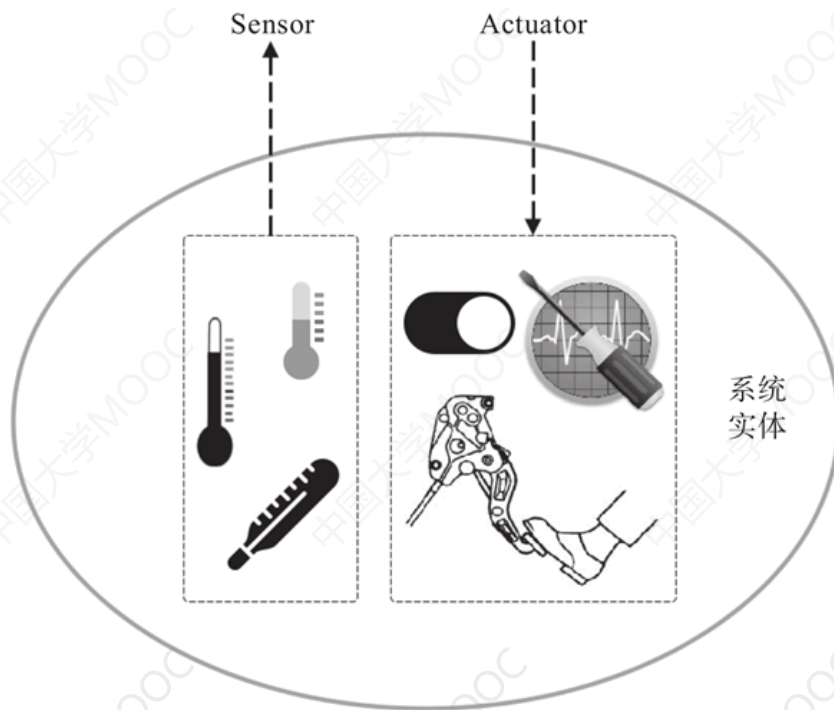


## 4. SpringBoot的配置



# 5. SpringBoot Actuator

- Spring Boot Actuator
  - 健康检查
  - 审计
  - 统计
  - 监控



# 5. SpringBoot Actuator

- Sensors类Endpoints

名称	说明
autoconfig	这个 endpoint 会为我们提供一份 SpringBoot 的自动配置报告，告诉我们哪些自动配置模块生效了，以及哪些没有生效，原因是什么。
beans	给出当前应用的容器中所有 bean 的信息。
configprops	对现有容器中的 ConfigurationProperties 提供的信息进行“消毒”处理后给出汇总信息。
info	提供当前 SpringBoot 应用的任意信息，我们可以通过 Environment 或者 application.properties 等形式提供以 info. 为前缀的任何配置项，然后 info 这个 endpoint 就会将这些配置项的值作为信息的一部分展示出来。
health	针对当前 SpringBoot 应用的健康检查用的 endpoint。
env	关于当前 SpringBoot 应用对应的 Environment 信息。
metrics	当前 SpringBoot 应用的 metrics 信息。
trace	当前 SpringBoot 应用的 trace 信息。
mapping	如果是基于 SpringMVC 的 Web 应用，mapping 这个 endpoint 将给出 @RequestMapping 相关信息。

# 5. SpringBoot Actuator

- Actuator类Endpoints
  - shutdown: 用于关闭当前 SpringBoot 应用的 endpoint。
  - dump: 用于执行线程的 dump 操作。