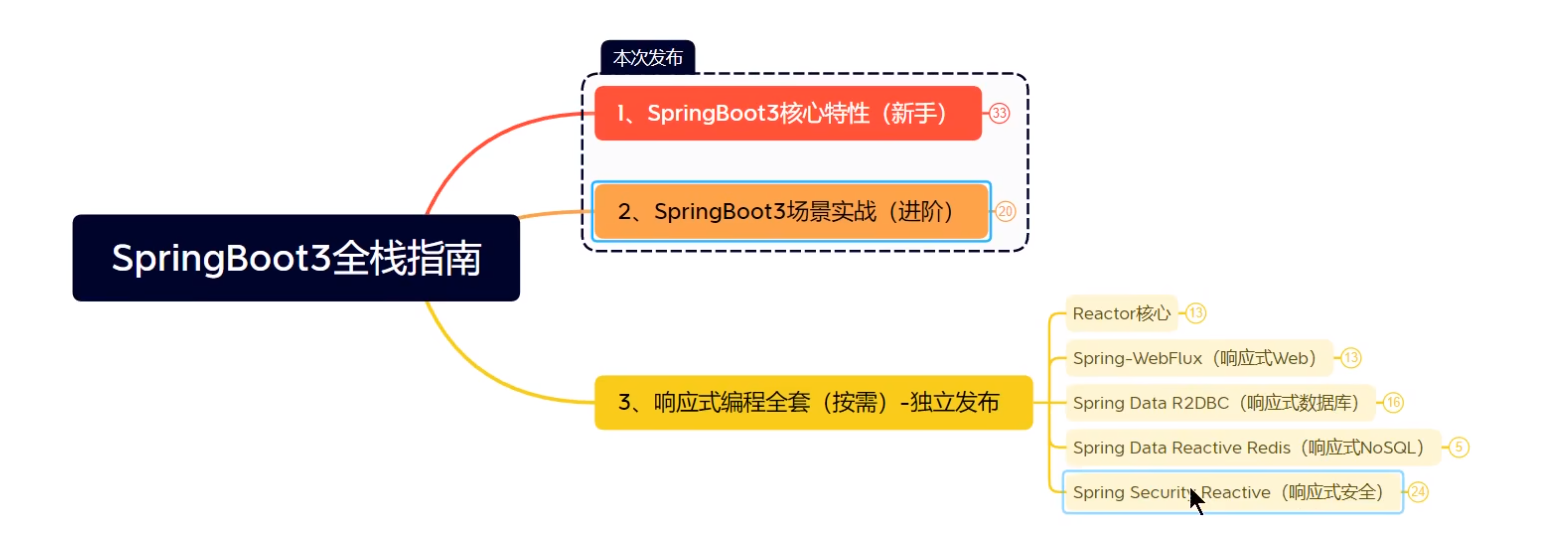
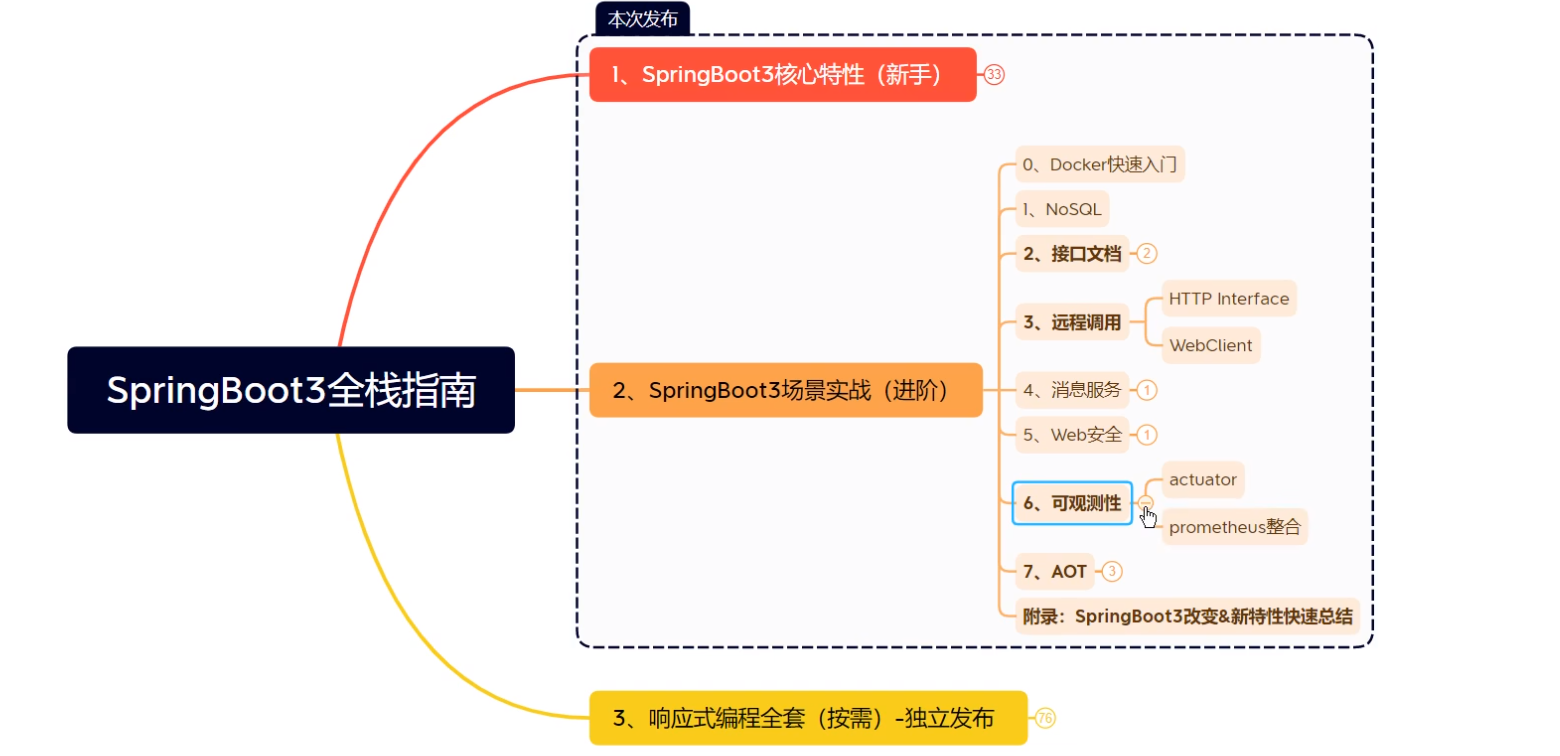
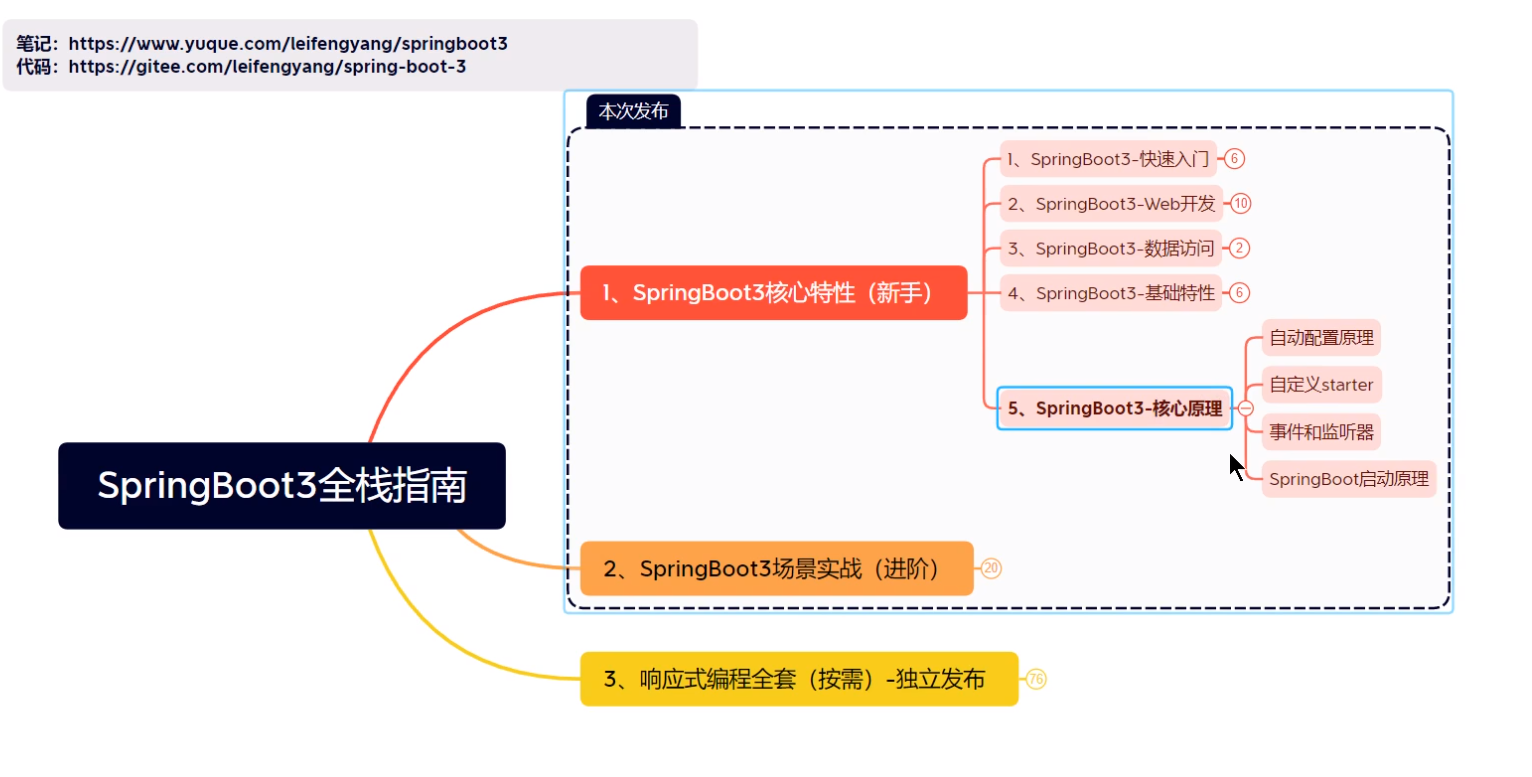
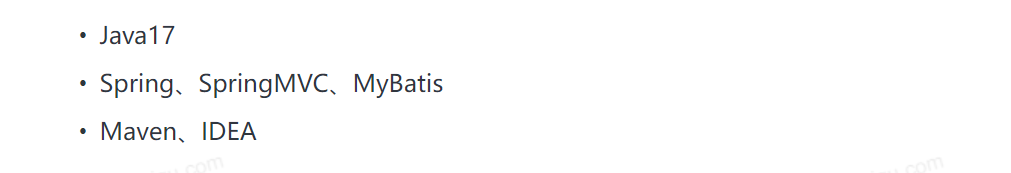
SpringBoot笔记



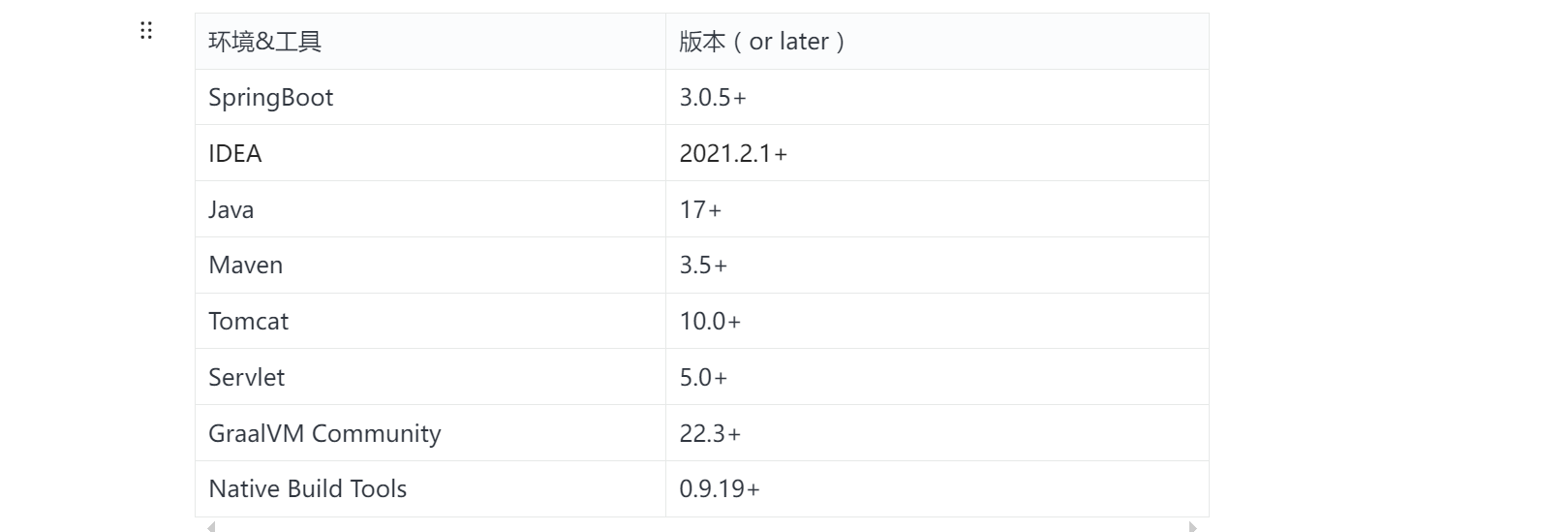
# SpringBoot3-快速入门

## 简介

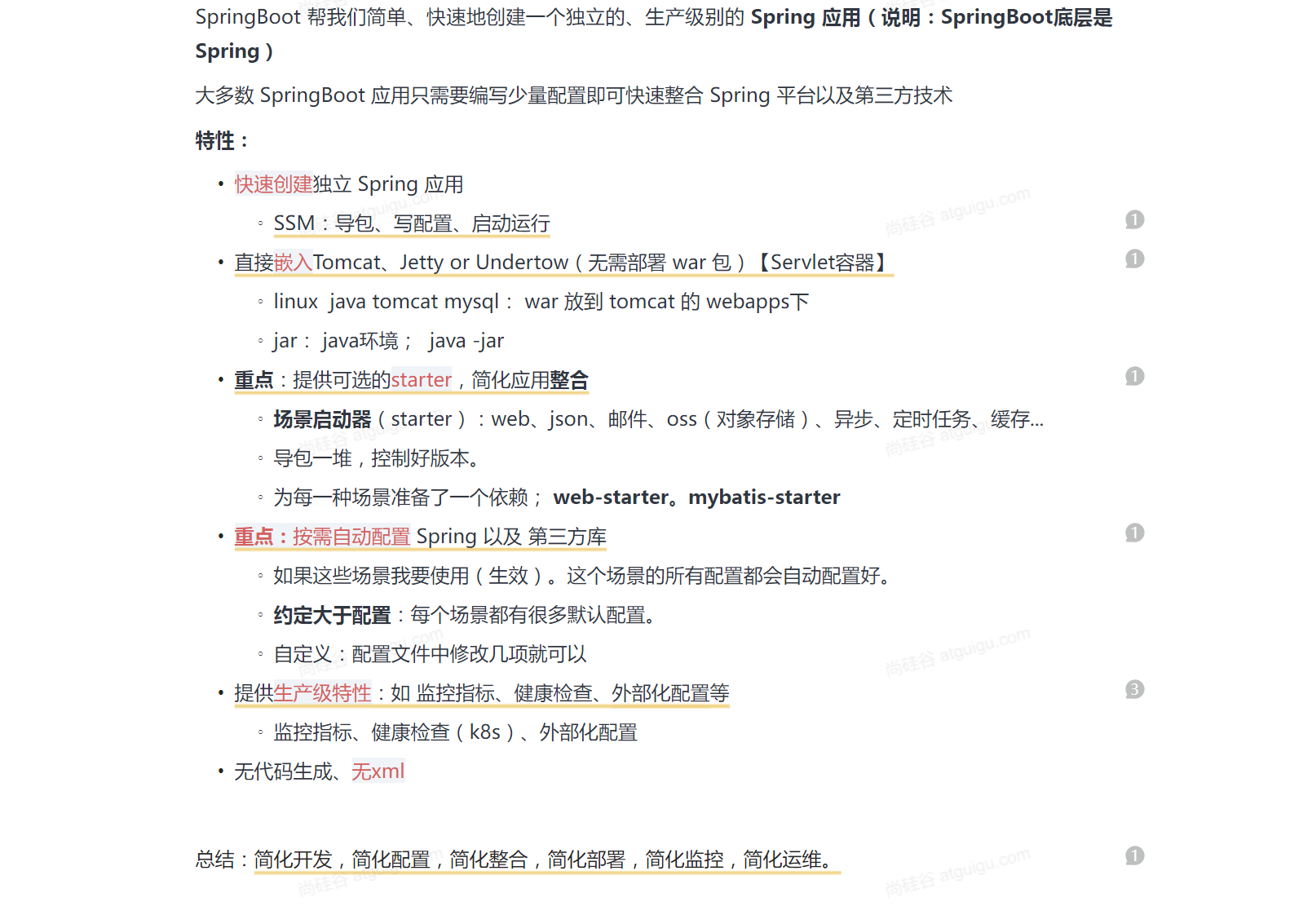
### 前置知识



### 1.1.2、环境要求



### 1.1.3、SpringBoot是什么

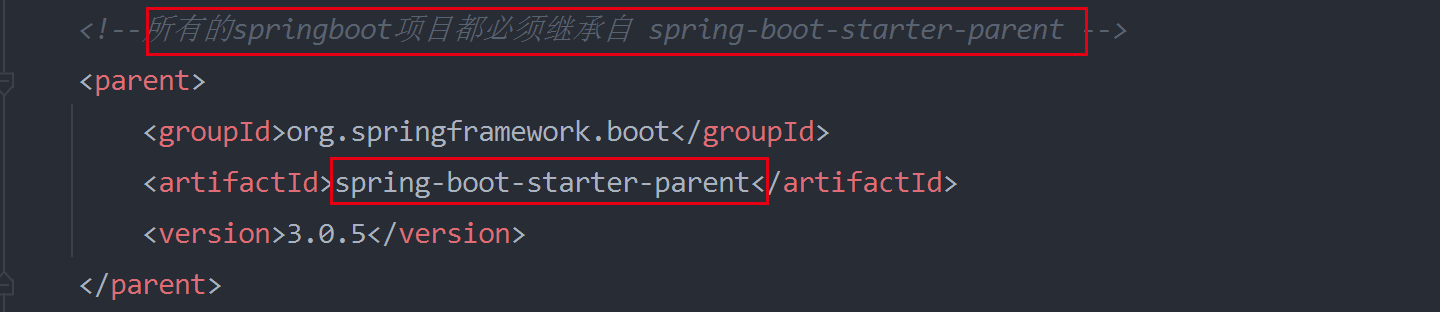


## 1.2、快速体验

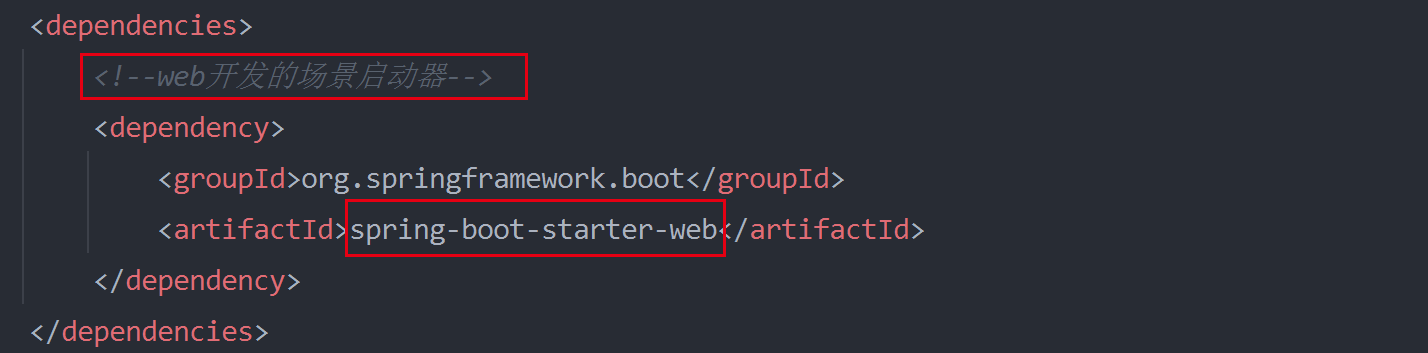


### 1.2.1、开发流程

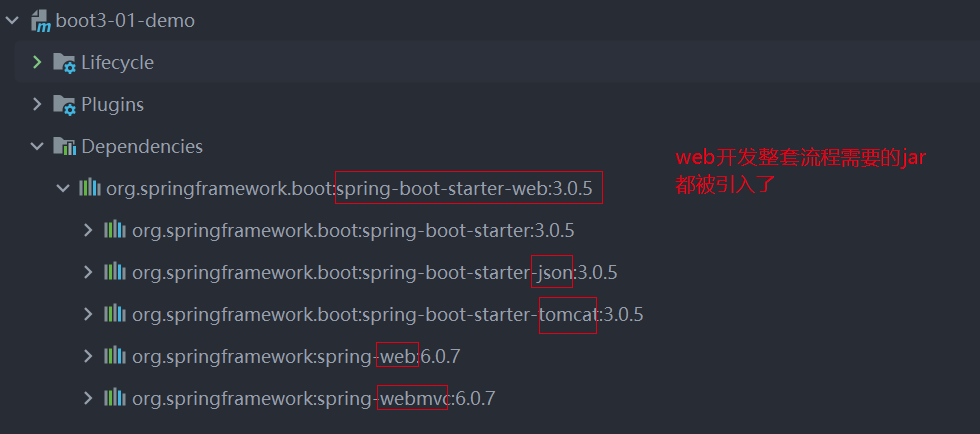
1、创建项目：创建一个maven项目，在pom.xml文件中引入springboot的父依赖。



2、导入场景：在pom.xml文件中引入场景启动器的依赖，此处以web场景启动器为例



引入web场景启动器后，web开发的所有jar都就被引入了项目。



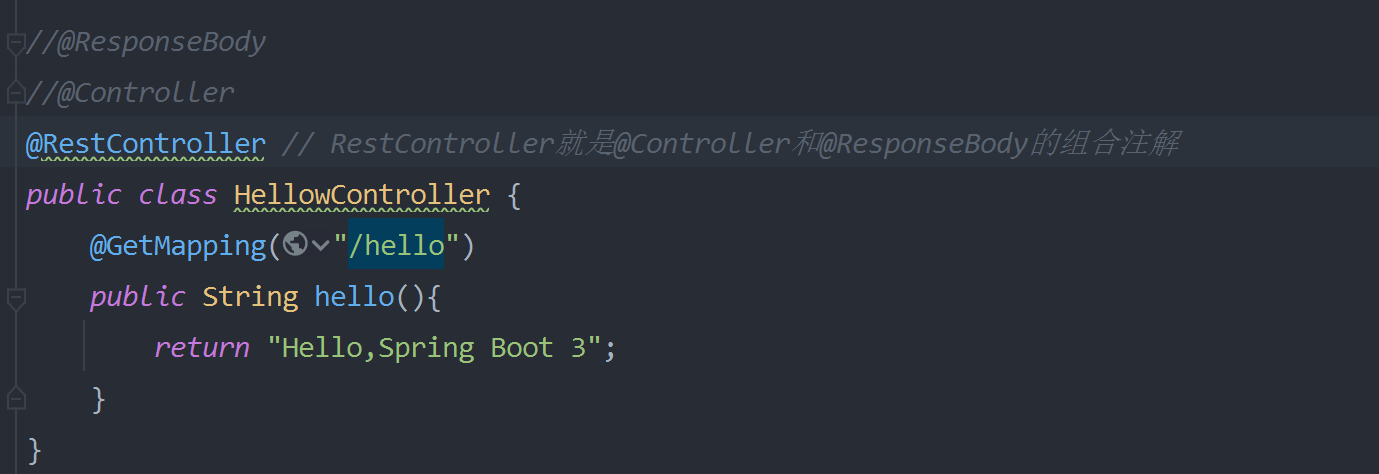
引入的这些依赖都已经做了相关的默认配置，不需要手动的再次配置。springboot主打的就是“**约定大于配置**”。

3、主程序



使用@SpringBootApplication注解标注该应用为springBoot的项目。启动SpringBoot项目的固定写法就是SpringApplication.run(类型.class, args);

4、编写业务代码

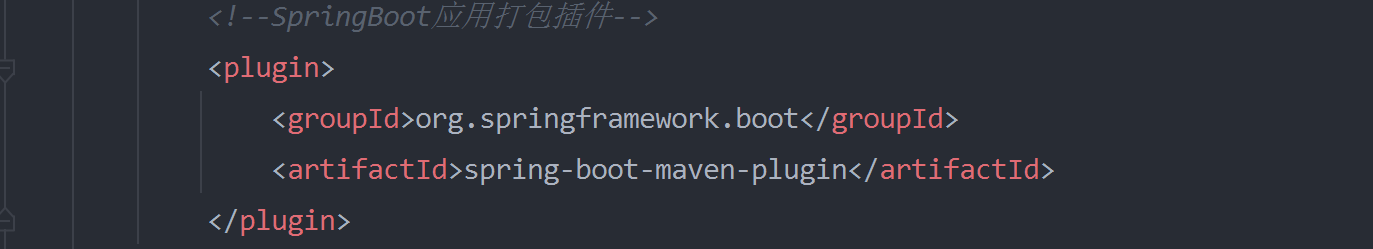


其中@RestController是@Controller和@ResponseBody的组合注解。

5、测试

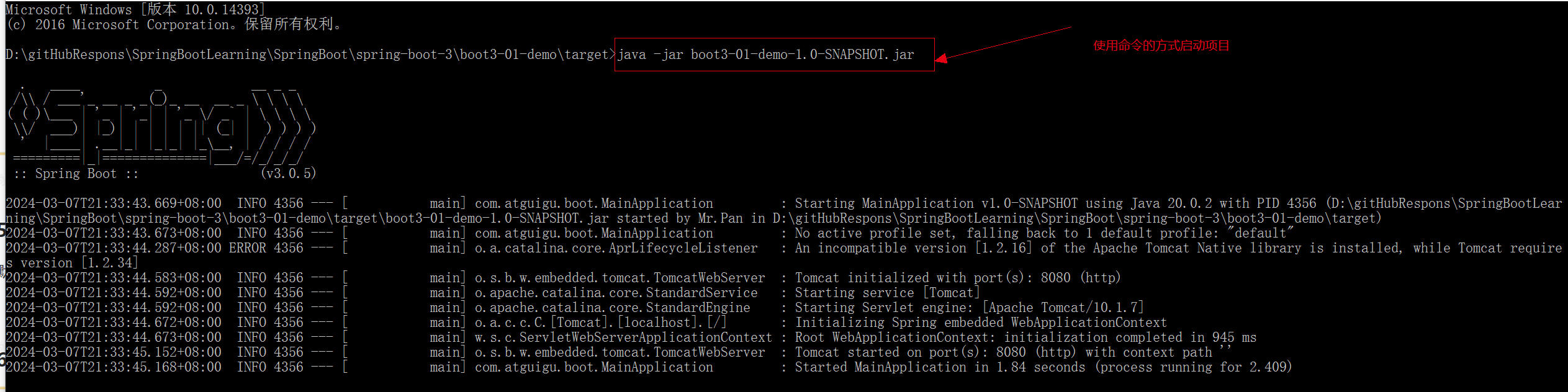


6、打包：可以将springboot项目打包成jar包，直接使用命令运行。但是打包需要引入springboot的打包插件。



打包命令：mvn clean package把项目打成可执行的jar包

运行：java -jar demo.jar启动项目



### 1.2.2、特性小结

1、简化整合

导入相关的场景，拥有相关的功能。场景启动器

默认支持的所有场景：<https://docs.spring.io/spring-boot/docs/current/reference/html/using.html#using.build-systems.starters>

● 官方提供的场景：命名为：spring-boot-starter-\*

● 第三方提供场景：命名为：\*-spring-boot-starter

**场景一导入，万物皆就绪**

2、简化开发

无需编写任何配置，直接开发业务

3、简化配置

只需要写一个配置文件application.properties就可以项目中所有需要配置进行配置

需要需要修改配置信息，可以demo.jar(springboot项目打成的jar包)同一目录下，编写一个application.properties进行修改配置（例如端口号等），然后重启项目（java -jar demo.jar），新的配置就可以生效。

● 集中式管理配置。只需要修改这个文件就行 。

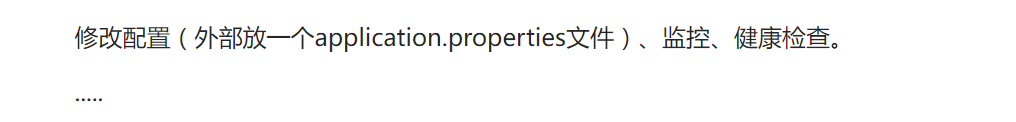
● 配置基本都有默认值

● 能写的所有配置都在： <https://docs.spring.io/spring-boot/docs/current/reference/html/application-properties.html#appendix.application-properties>

4、简化部署

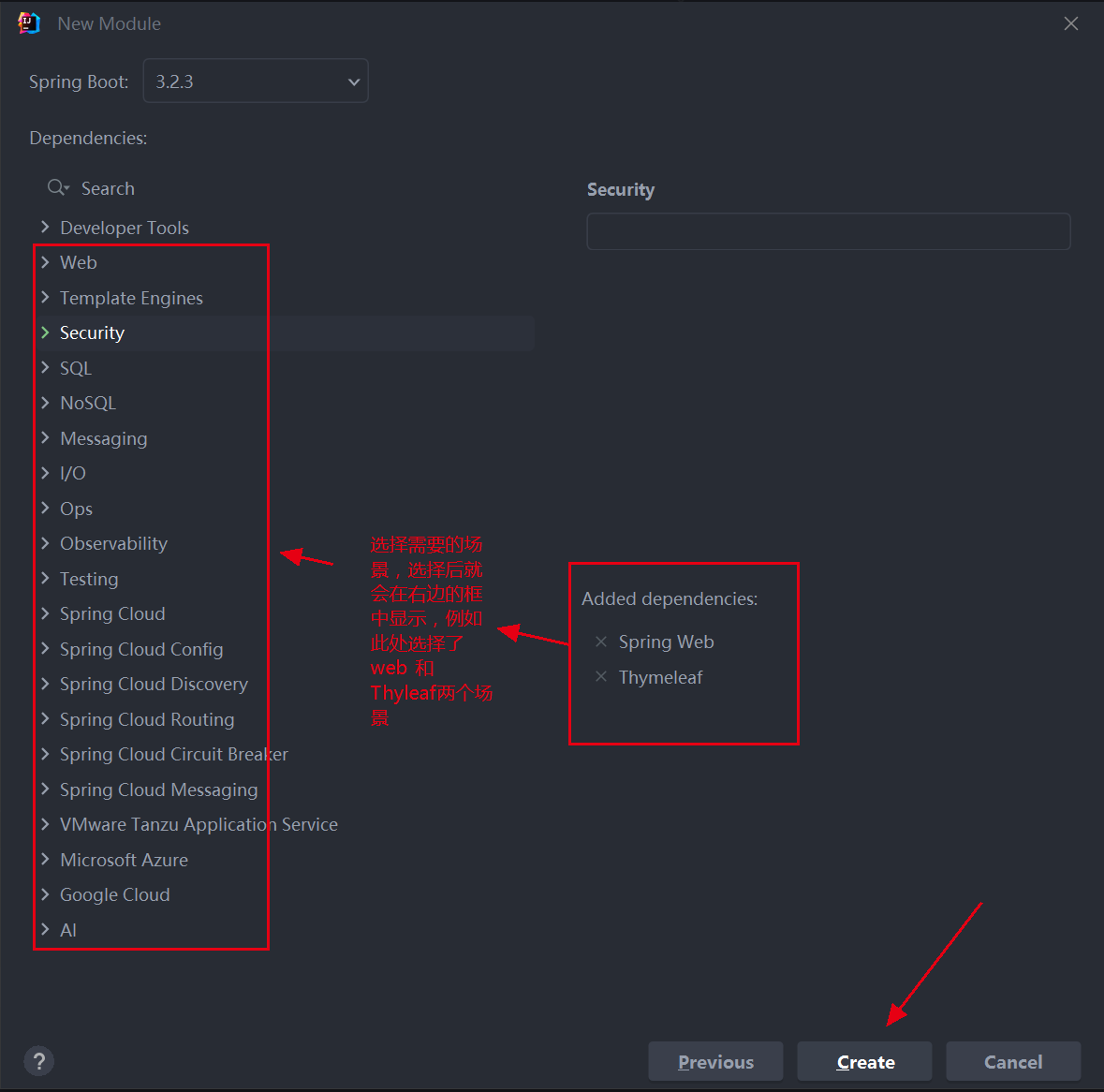
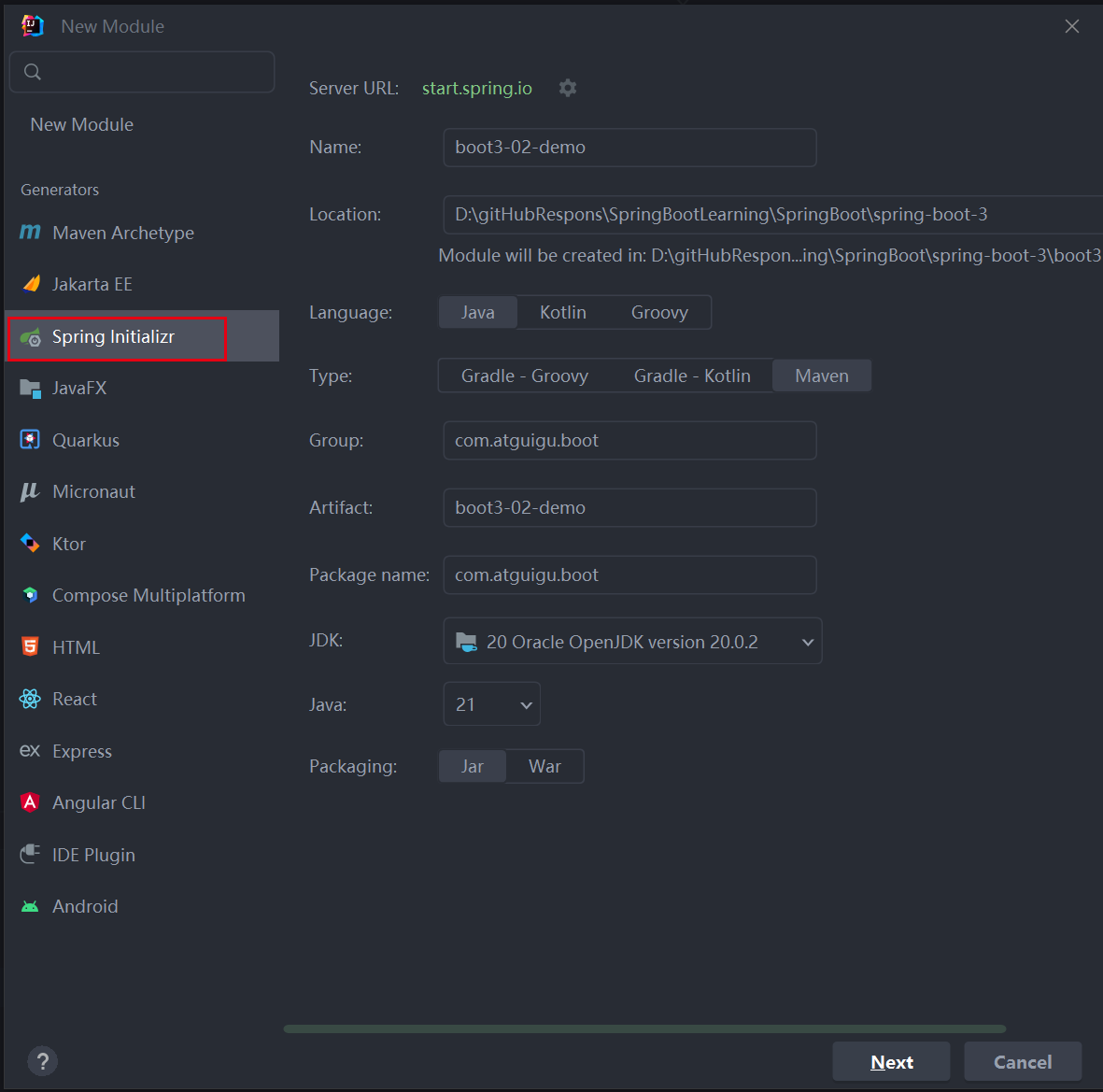


5、简化运维

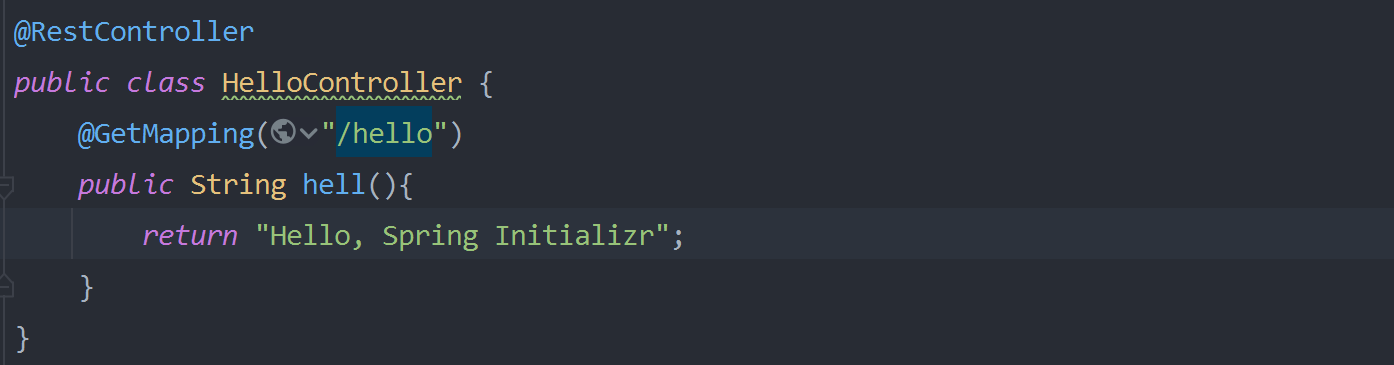


### 1.2.3、Spring Initializr创建导向

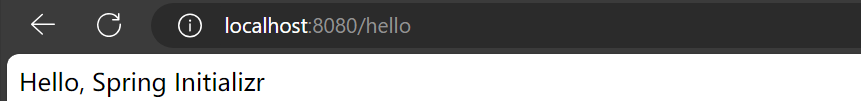
一键创建好整个项目结构，业务代码必须写在启动类包下的包中，springboot只扫描启动类下的包中的类。



测试案例：业务代码需要自己编写



测试结果：



## 1.3、应用分析

### 1.3.1、依赖管理



1. 版本号的集中管理：

在Spring Boot项目中，通常会使用spring-boot-starter-parent作为父项目，而spring-boot-starter-parent的父项目是spring-boot-dependencies。

这种层级结构允许进行版本号的集中管理。spring-boot-dependencies声明了各种常用库的版本，而spring-boot-starter-parent继承了这些版本，使得子项目可以继承这些版本号而无需显式声明。

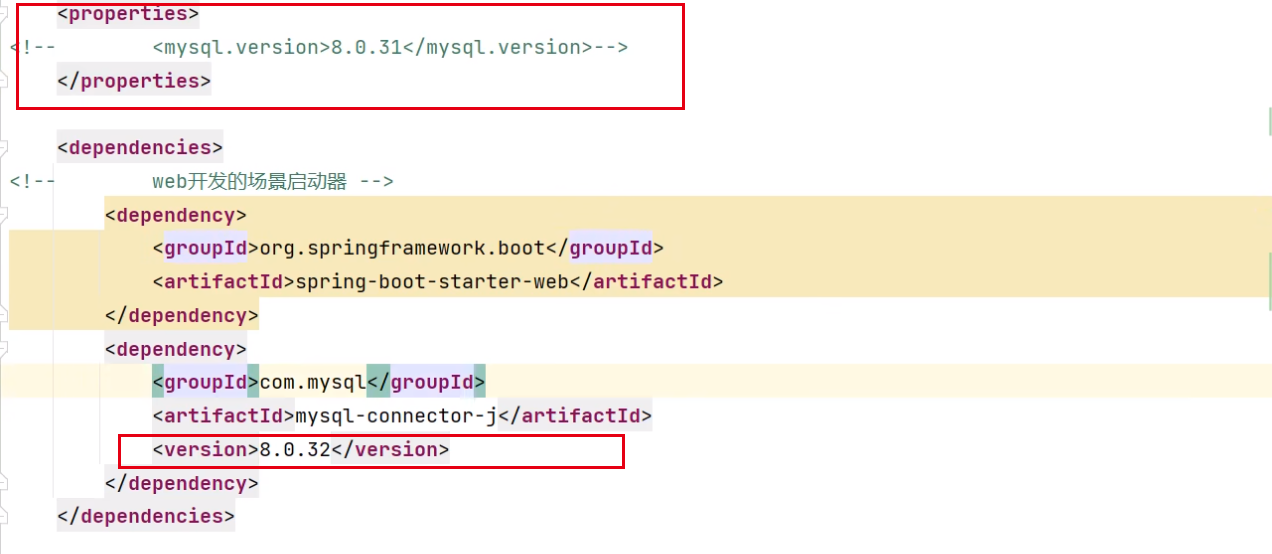
2. 自定义版本号：

如果需要使用自定义的版本号，可以通过Maven的"就近原则"来实现。

可以在当前项目的<properties>标签中声明父项目用的版本，指定一个版本。

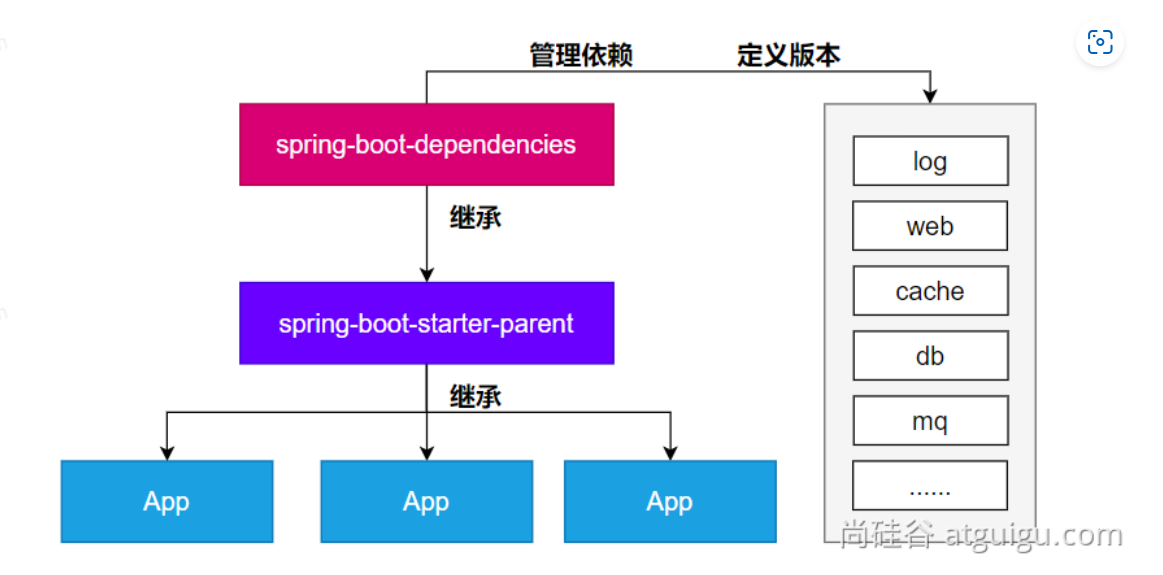


也可以在直接导入依赖的时候声明版本号，覆盖父项目中声明的版本号。



通过这种方式，Spring Boot项目可以更方便地管理和升级依赖库的版本，同时也保持了良好的灵活性和可维护性。这种做法有助于简化项目配置，减少重复代码，以及确保整个项目中使用的库都是兼容的。

springBoot的依赖管理图：



### 1.3.2、自动配置

1、初步了解：

1.1、自动哦配置的Tomcat、SpringMVC等

导入场景，容器中就会自动配置好这个场景的核心组件。

以前：配置DispatcherServlet 、ViewResolver、CharacterEncodingFilter等核心bean

现在：自动配置好的这些组件

验证：**容器中有了什么组件，就具有什么功能**

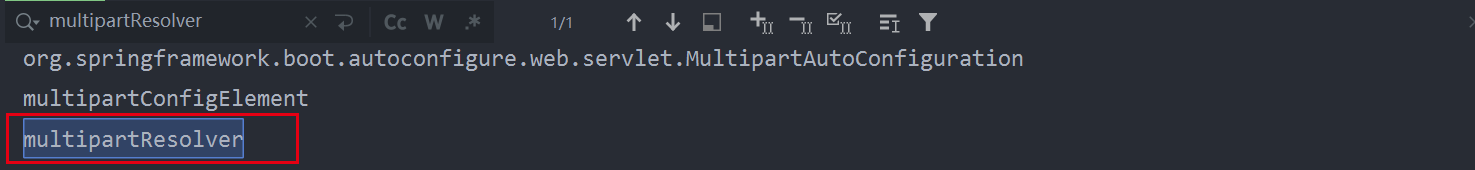
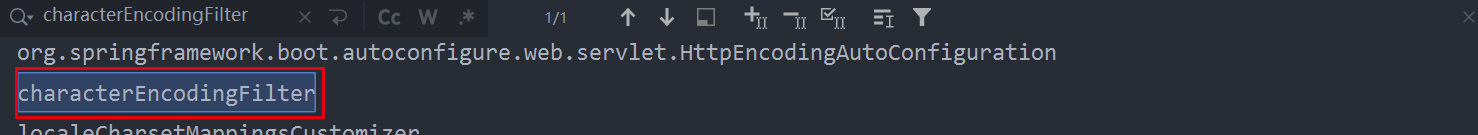
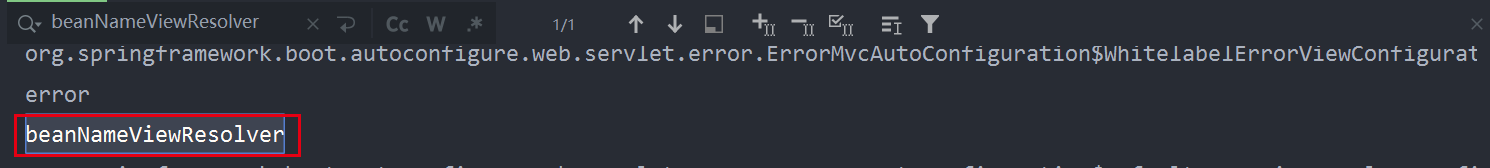
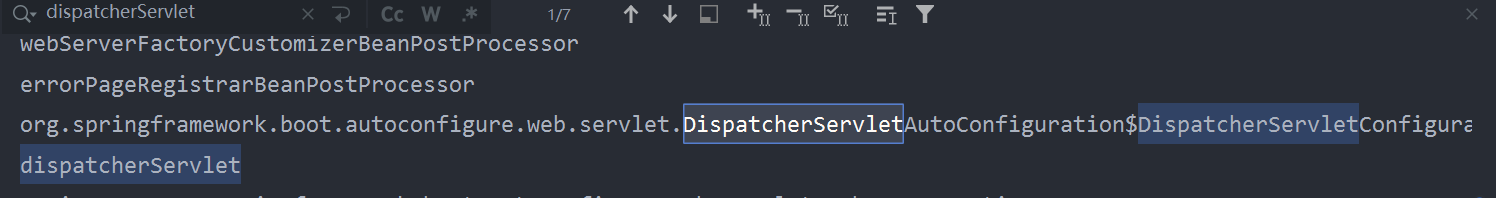
此处证明引入webMVC场景启动器后，是否存在springMVC的项目核心类举例。



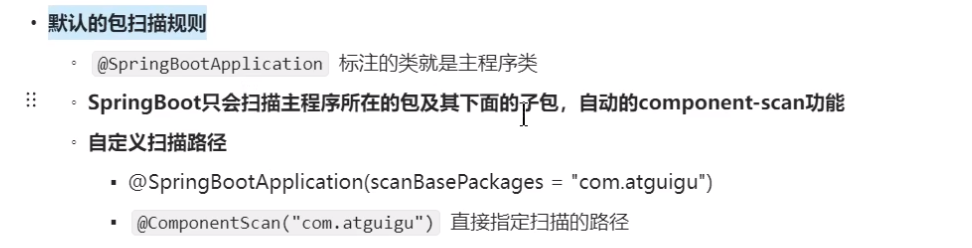
SpringBoot启动后，返回的是ConfigurabkeApplicationContext对象，这个ConfigurabkeApplicationContext对象就是IOC容器，所以只需要验证该IOC容器中是否存SpringMVC的核心组件就可以，例如是否存在DispatcherServlet对象。

验证案例：

验证结果：dispatcherServlet、beanNameViewResolver（用于视图解析）、characterEncodingFilter（字符编码）、multipartResolver（文件上传）等bean都已经配置好了。



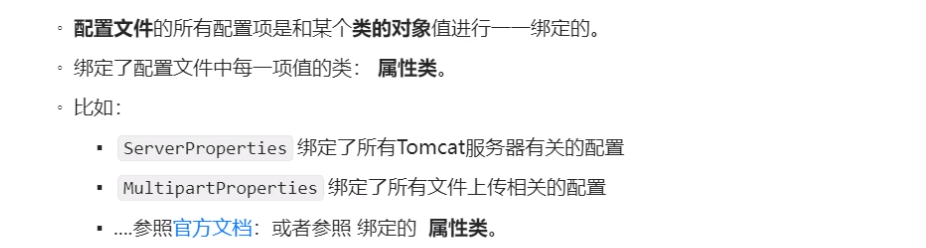
1.2、默认的包扫描规则。





预定大于配置，还是使用默认的配置就行，即扫描主程序所在包及其下面的子包路径。

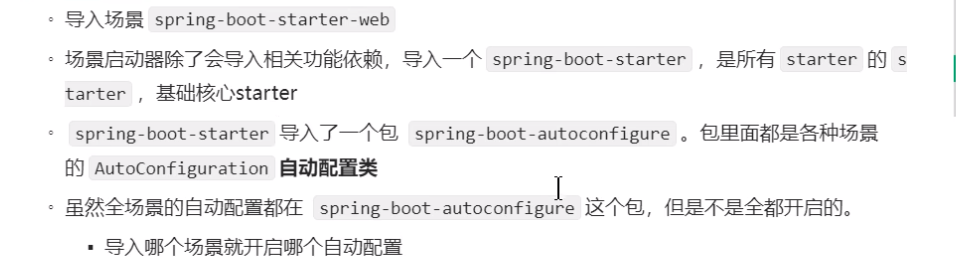
1.3、配置默认值

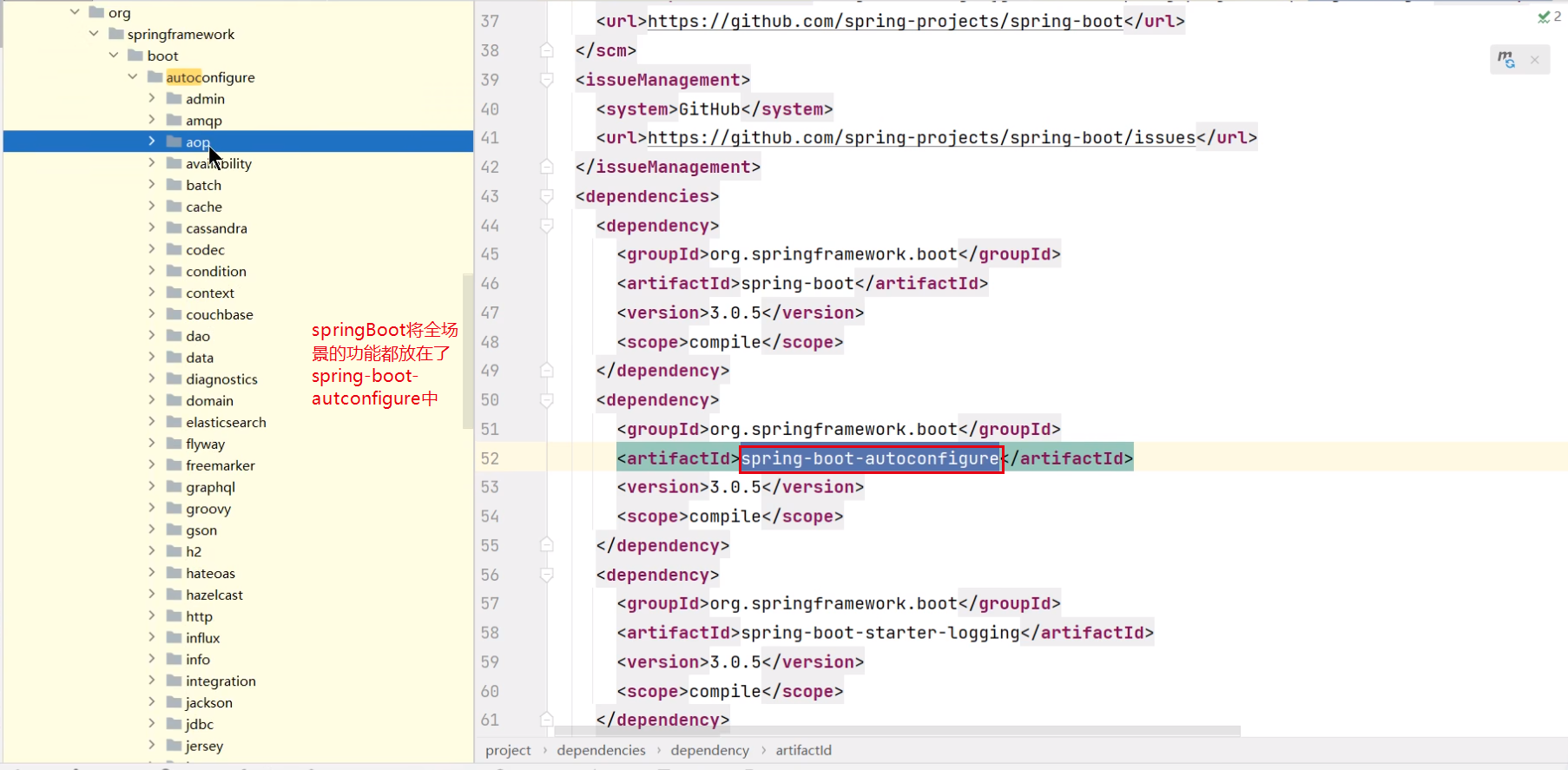




属性类的官方文档地址：<https://docs.spring.io/spring-boot/docs/current/reference/html/application-properties.html#appendix.application-properties>

1.4、按需加载自动配置





总结：导入场景启动器，触发spring-boot-autoconfigure这个包的自动配置生效，容器中就会具有相关创建的功能。

## 1.4、核心技能

### 1.4.1、常用注解

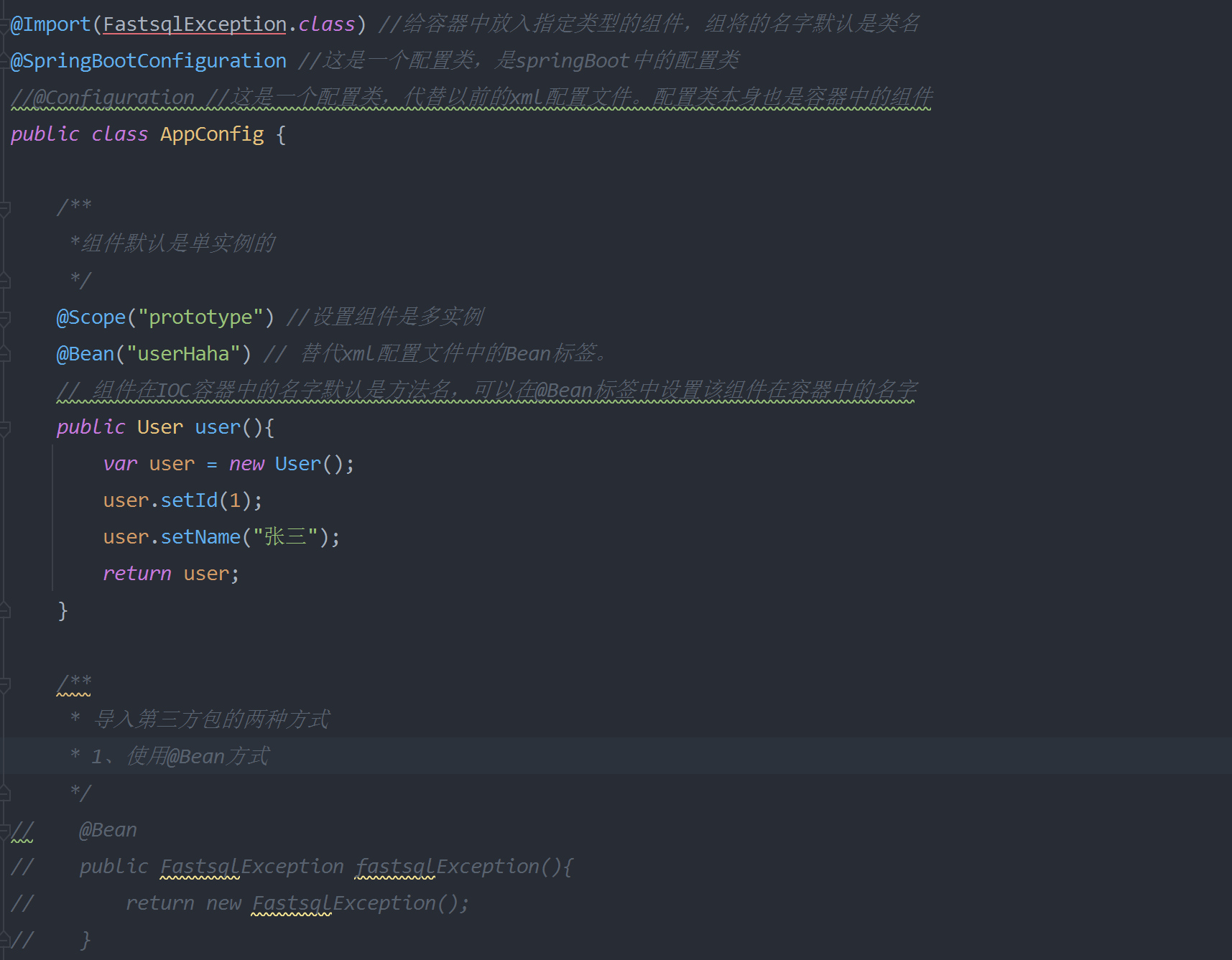
**SpringBoot**摒弃了XML配置方式，改为**全注解驱动**

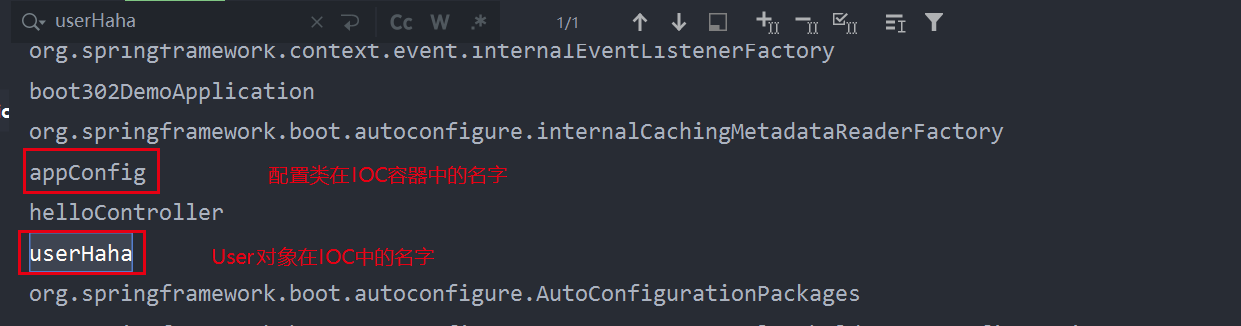
* 1. 组件注册



测试案例：







1.2、条件注解

1.3、属性绑定

### 1.4.2、Yaml文件

1**.4.3、日志文件**