

SpringBoot 探秘到实战开发

官方文档

<https://docs.spring.io/spring-boot/docs/2.5.8-SNAPSHOT/reference/htmlsingle/#getting-started>

01、SpringBoot的概述


Spring Boot 是由 Pivotal 团队提供的全新框架。可以轻松创建独立的、生产级的基于 Spring 的应用程序。可用于快速开发扩展性强、微小项目、业界称之为：“微框架”。毋庸置疑SpringBoot的诞生不仅给传统的企业级项目与系统架构带来了全面改进以及升级的可能。同时也给Java程序员带来诸多益处。是Java开发的一大利器。

从最根本上来讲，Spring Boot 就是一些库的集合，它能够被任意项目的构建系统所使用。它使用“习惯优于配置”（项目中存在大量的配置，此外还内置一个习惯性的配置）的理念让你的项目快速运行起来。用大佬的话来理解，就是 **spring boot** 其实不是什么新的框架，它默认配置了很多框架的使用方式，就像 **maven** 整合了所有的 jar 包，**spring boot** 整合了所有的框架，总结一下及几点：

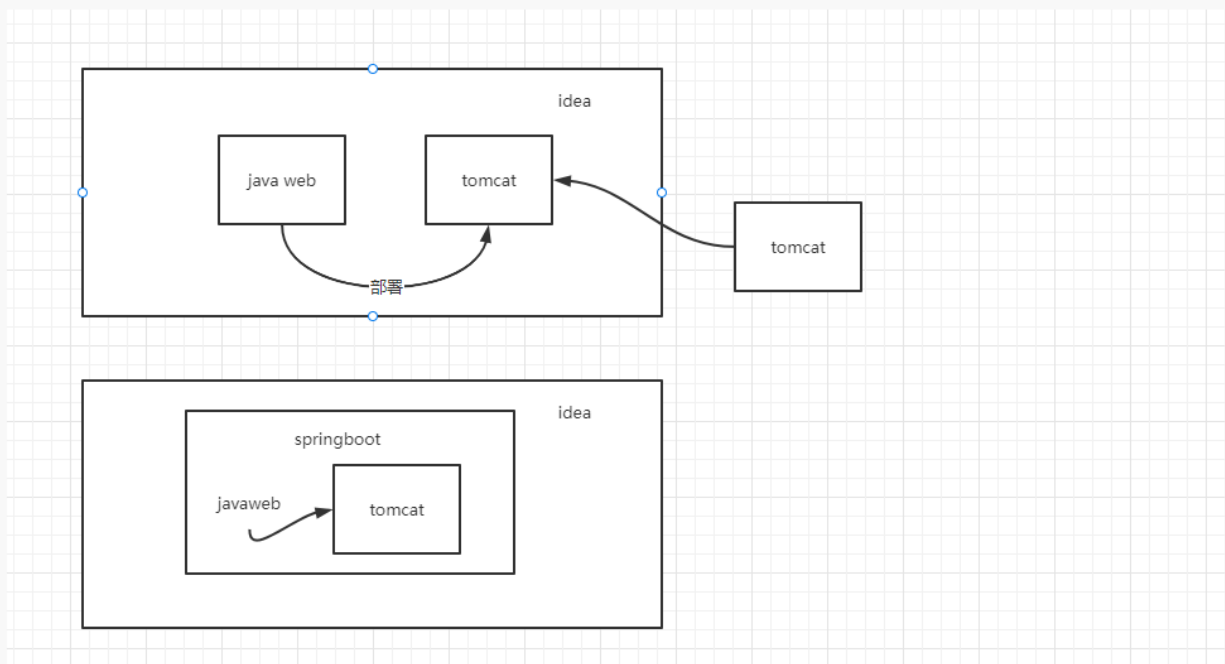
*springboot*其实从真正意义上来说，不能称之为框架，它是基于*spring*的应用程序的脚手架（造好的轮子）。可用于快速开发扩展性强、微小项目。比如：摒弃了传统的开发模式大量*xml*配置的问题，依赖外部容器问题，以及依赖*jar*包的问题。说白了：就把程序员的开发工作从每天日常的开发中繁复问题全部解放，变得非常的和易用，让程序员更多的时间关注到业务，而不应该每天在做配置和大量的启动工作。所以这也就是说为什么*springboot*不是一个框架。

底层依赖：spring

特点

 **01**、能够快速创建独立的构建 **Spring** 应用程序

 **02**、直接嵌入**Tomcat**、**Jetty**或**Undertow**（无需部署**WAR**文件）



03、提供自以为是的“**starter**”依赖项以简化您的构建配置

依赖**jar**的完整的开发历程是：

- 从jsp/servlet - ssh 这个时代依赖第三**jar**是通过：
- 1、手动 + 官网下载
 - 2、下载好放入 /webroot/WEB-INF/lib
 - 3、很多框架的包相互依赖，每个**jar**官网都不一样

- 1 **缺点：**官网太多，网速太慢，包的依赖版本必须要匹配，非常耗时。
- 2 **优点：**版本是自己去查找，比较清晰，不需要额外在下载了。

- ssh--ssm : 出现**jar**的依赖管理工具: ant -- > maven
 - ant 使用非常的复杂，一般可配置太啰嗦
 - maven 其实就是简单ant的版本，取代ant的繁复的配置和构建的问题。maven不仅仅只是为了解决**jar**的依赖问题，

还解决项目依赖和构建问题，以及jar仓库管理问题。

- **gradle**：服务anroid应用程序开发的包管理，但是有企业用，用并没有maven那么多。

**** 缺点：** 你需要额外学习**maven**, 学习成本变高， 它一定依赖网络

好处：不需要自己手动下载**jar**依赖，构建的问题。

没有解决：**jar**依赖的版本问题，比如如果你开发ssm项目，maven的pom.xml文件中就必须把spring的七大核心包都要进行依赖， spring.jar ---- fastjson.xml，文件上传：spring-web.jar common-fileuploader.jar,common-file.jar**


- **springboot**提供了 **starter**机制：默认装配配置类的问题 和 **jar**依赖的问题。

- **spring-boot-starter-web.jar**
 - tomcat
 - log
 - spring-web spring-webmvc
 - json

你需要额外学习**maven**, 学习成本变高， 它一定依赖网络


好处：就程序员不需要在去关注，这个**jar**依赖另外一个**jar**包的问题，全部自动进行依赖匹配。如果没有这个机制，比如未来要依赖 **mybatis-plus-boot-starter**，你可以就自己去手动的依赖下面的包：

- mybatis-version
- mybatis-plus-version
- spring-jdbc--version
- myabtis-spring-version

 **04**、集成了大量常用的第三方库的配置， **Spring Boot** 应用为这些第三方库提供了几乎可以零配置的开箱即用的能力。

 **05**、零配置。无冗余代码生成和**XML** 强制配置，遵循“约定大于配置”。

 **06**、 **Spring Boot** 不是**Spring** 的替代者， **Spring** 框架是通过 **IOC** 机制来管理 **Bean** 的。 **Spring Boot** 依赖 **Spring** 框架来管理对象的依赖。 **Spring Boot** 并不是 **Spring** 的精简版本，而是为使用 **Spring** 做好各种产品级准备。

 **07**、提供一系列大型项目通用的非功能特性（例如嵌入式服务器、安全性、指标、健康检查和外部化配置）提供生产就绪功能，例如指标、运行状况检查和外部化配置--**Actual**。

总结

用来简化新 **Spring** 应用的初始搭建以及开发过程。它依赖spring，只不过是spring应用程序开发的一种简化。或者你可以这样理解，springboot是spring框架的一个产品。

02、Spring Boot 在应用中的角色

概述

Spring Boot 是基于 Spring Framework 来构建的，Spring Framework 是一种 J2EE 的框架（什么是 J2EE？）

Spring Boot 是一种快速构建 Spring 应用脚手架。

Spring Cloud 是构建 Spring Boot 分布式环境，也就是常说的云应用

总结

Spring Boot 中流砥柱，承上启下。如果你不学习springBoot后面的springcloud你将无法开发。

理解

vue ---框架 *vue-cli* 脚手架 - 产品若依、*antd* 你们公司的产品
spring--框架 *springboot*脚手架 --- 个人资讯系统，旅游项目

03、SpringBoot系统环境要求（重点）

进入企业对环境理解的重要性

未来如果你进入到企业，你开发的

- 01、一定先把代码下载下来，
- 02、确定你项目的技术架构
- 03、运行你的项目。但是在运行之前，最好一定尽早的问，公司是否有产品的项目白皮书（说明书），如果没有，一定问如果遇到问题，我可以咨询哪位。
- 04、运行这个项目，现在公司的一个基础环境是什么样子的呢？
- 05、然后在去运行项目。

SpringBoot的环境要求

- （1）JDK 环境**必须**是 1.8 或者**jdk11**版本及以上
- （2）后面要使用到 Maven 管理工具 3.5+ 及以上版本 建议是：3.6 不要用最新。
- （3）内置了Tomcat9.x/Servlet4.x
- （4）开发工具建议使用 IDEA，也可以 MyEclipse，为了实现一站式服务。

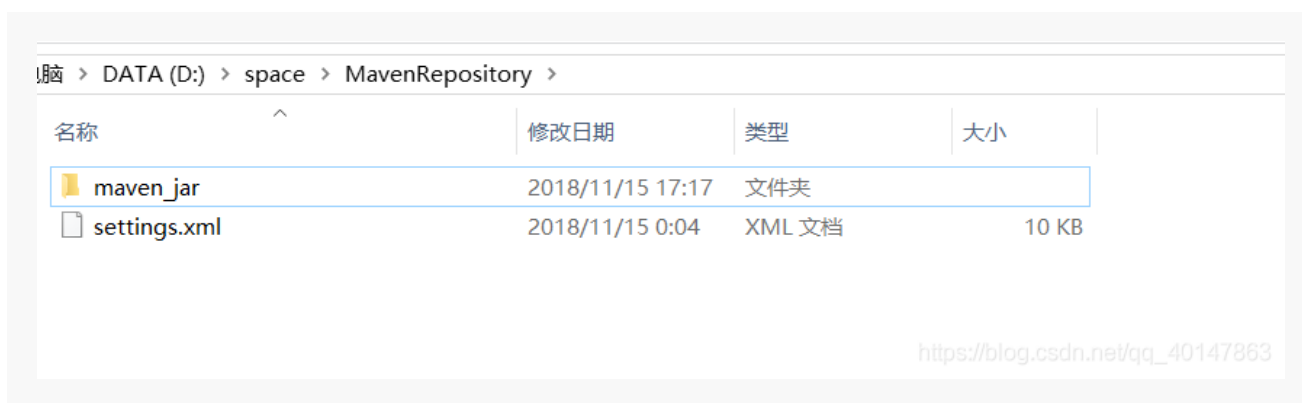
官方提供的eclipse工具：<https://spring.io/tools> (热加载 改的类自动重启)

Maven 安装与环境变量配置

(1) Maven 安装:

在官网下载: <http://maven.apache.org/download.cgi>

历史版本下载: <https://archive.apache.org/dist/maven/maven-3/>



(2) Maven 配置环境变量:

解压到一个路径, 然后配置环境变量:

- 新建变量名: MAVEN_HOME 变量值: D:\server\apache-maven-3.6.0 (这是我的 MAVEN 路径)
- 编辑变量名: Path 在最前面加上: %MAVEN_HOME%\bin

(3) 检查是否配置成功:

在命令行输入:


```
1 mvn -V
```

然后会一大堆东西：

```
C:\Users\Administrator>mvn -V
Apache Maven 3.6.0 (97c98ec64a1fdfee7767ce5fffb20918da4f719f3; 2018-10-25T02:41:47+08:00)
Maven home: C:\Program Files\apache-maven-3.6.0\bin\..
Java version: 1.8.0_191, vendor: Oracle Corporation, runtime: C:\Program Files\Java\jdk1.8.0_191\jre
Default locale: zh_CN, platform encoding: GBK
OS name: "windows 10", version: "10.0", arch: "amd64", family: "windows"
```

(4) 配置 **maven** 仓库：

- 1.打开 maven 文件夹下的 config 文件夹下的 settings.xml；
- 2.找到 localRepository 标签，此时是被注释掉的，我们解除注释，然后配置一个路径，例如：

D:/space/MavenRepository/maven_jar，这样以后 MAVEN 管理下载的jar 包都会在这个路径下。

【注意】：注意结点的位置，先找到注释掉的部分，贴在下面

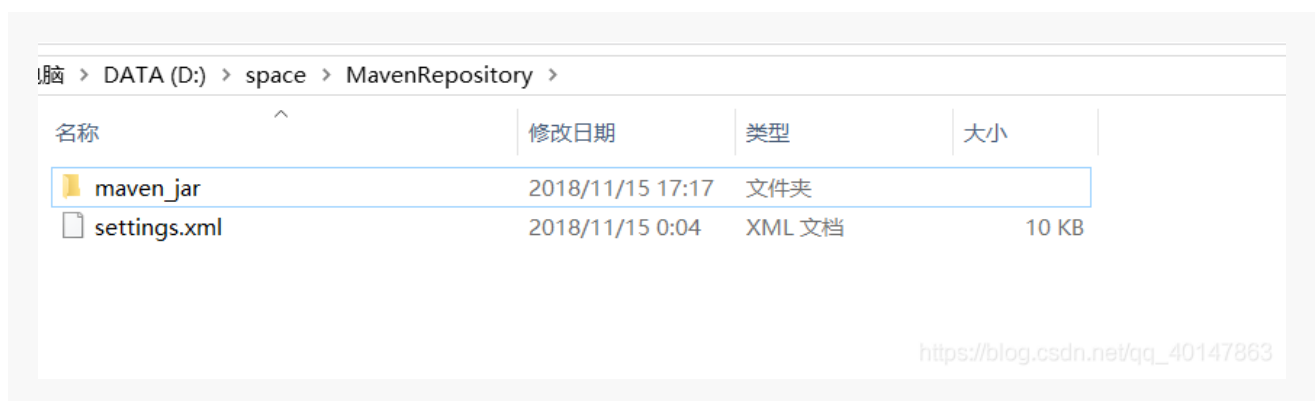
```
1 <localRepository>D:\space\MavenRepository\maven_jar</localRepository>
```

```
49 <!-- localRepository
50 | The path to the local repository maven will use to store artifacts.
51 |
52 | Default: ${user.home}/.m2/repository
53 | -->
54 <localRepository>D:\space\MavenRepository\maven_jar</localRepository>
55
56
57
```

- 3.配置远程仓库，找到 mirrors 标签

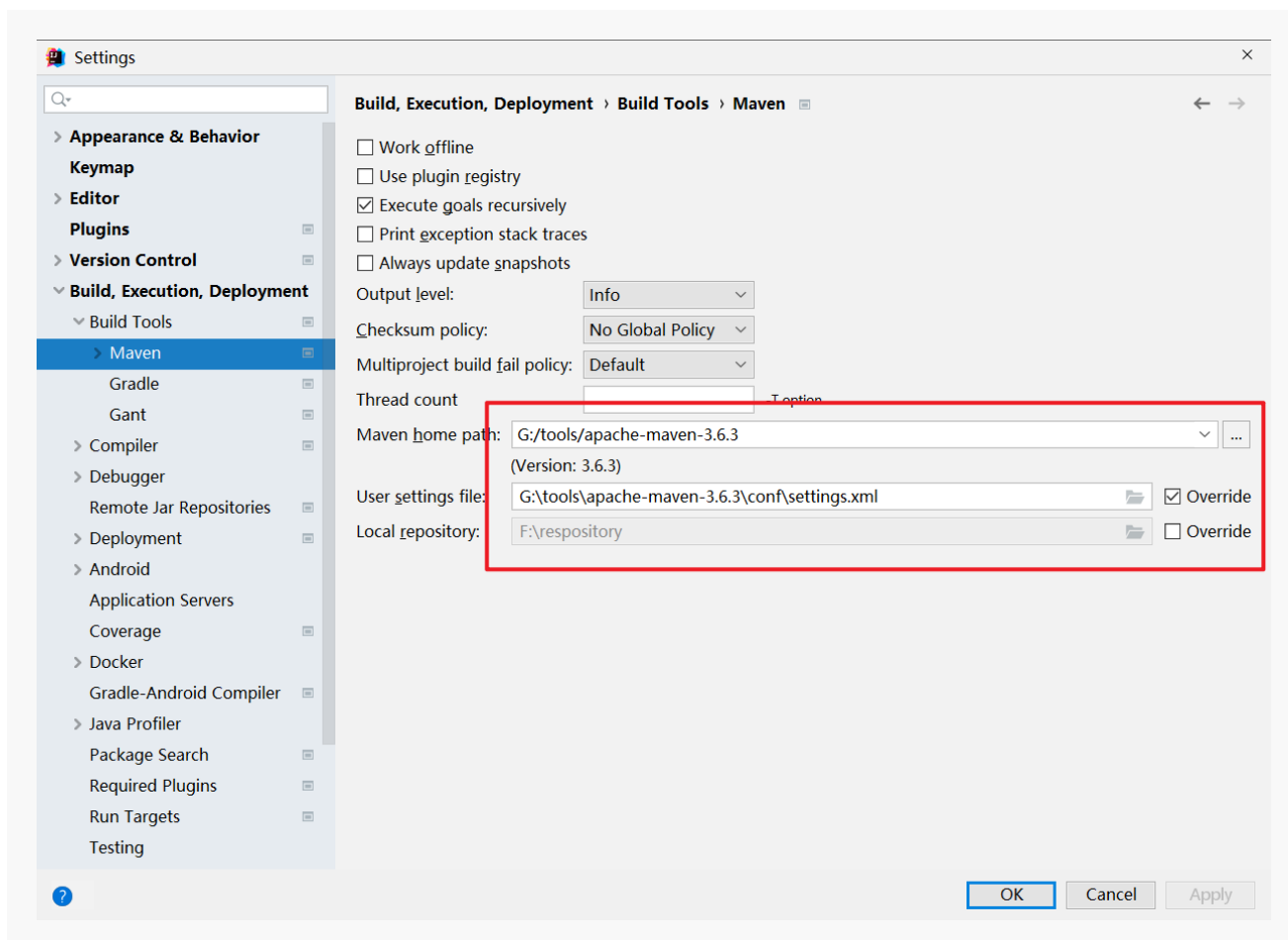
```
1 <!--远程仓库-->
2 <mirror>
3     <id>aliyun</id>
4     <name>aliyun Maven</name>
5     <mirrorOf>*</mirrorOf>
6
7     <url>http://maven.aliyun.com/nexus/content/groups/
      public/</url>
8 </mirror>
```

4.当然我们需要先建这样一个目录结构，然后还要把settings.xml 复制一份到 D:/space/MavenRepository 下



(5) 在 idea 配置 maven

点击【File】>【Settings】>搜索【Maven】，按截图配置安装目录和选择刚刚 settings 配置文件；

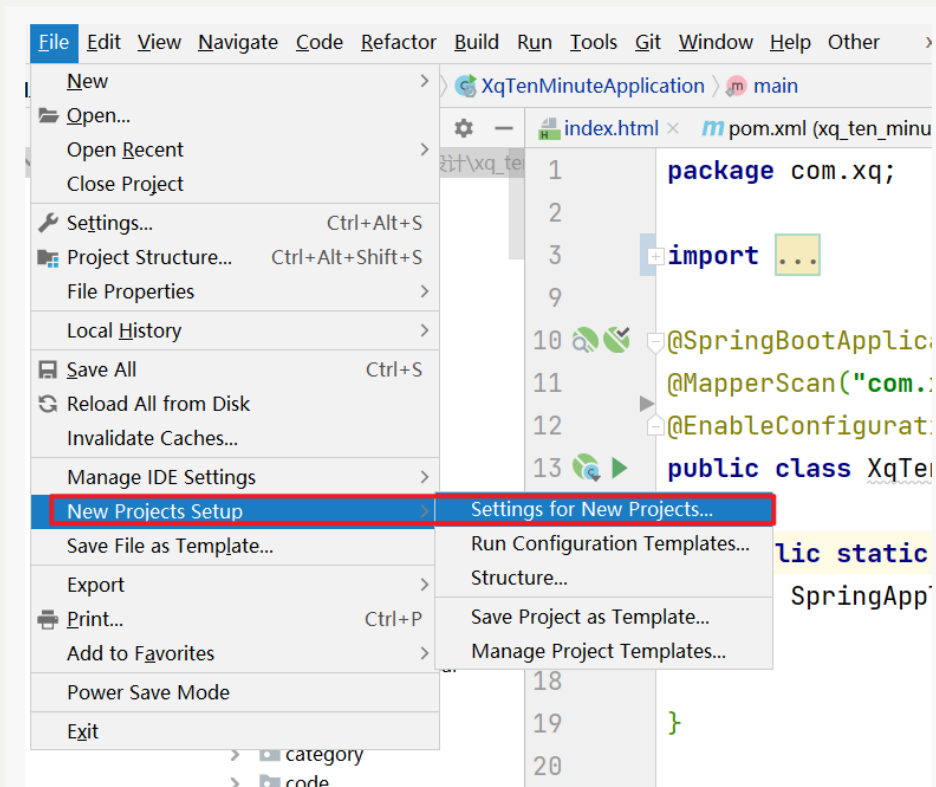


选择完settings之后，本地仓库自动改成settings文件中配置的；点击apply，再点击ok即配置完成。

 给未来的新工程提供配置

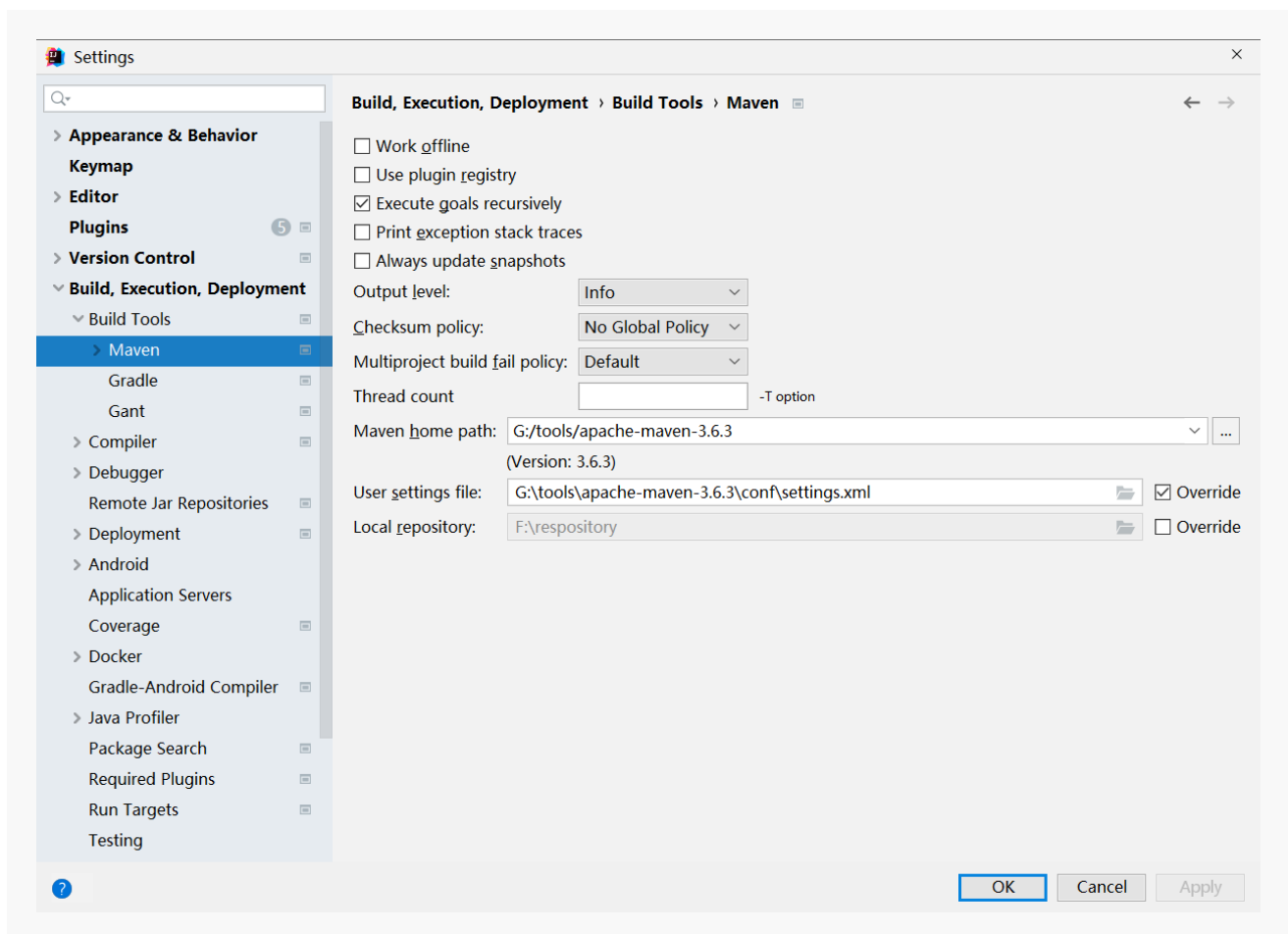
 概述

上面的配置只限于当前工程，如果未来你新建的工程要想只配置一次，然后后续不在配置你就必须配置如下：

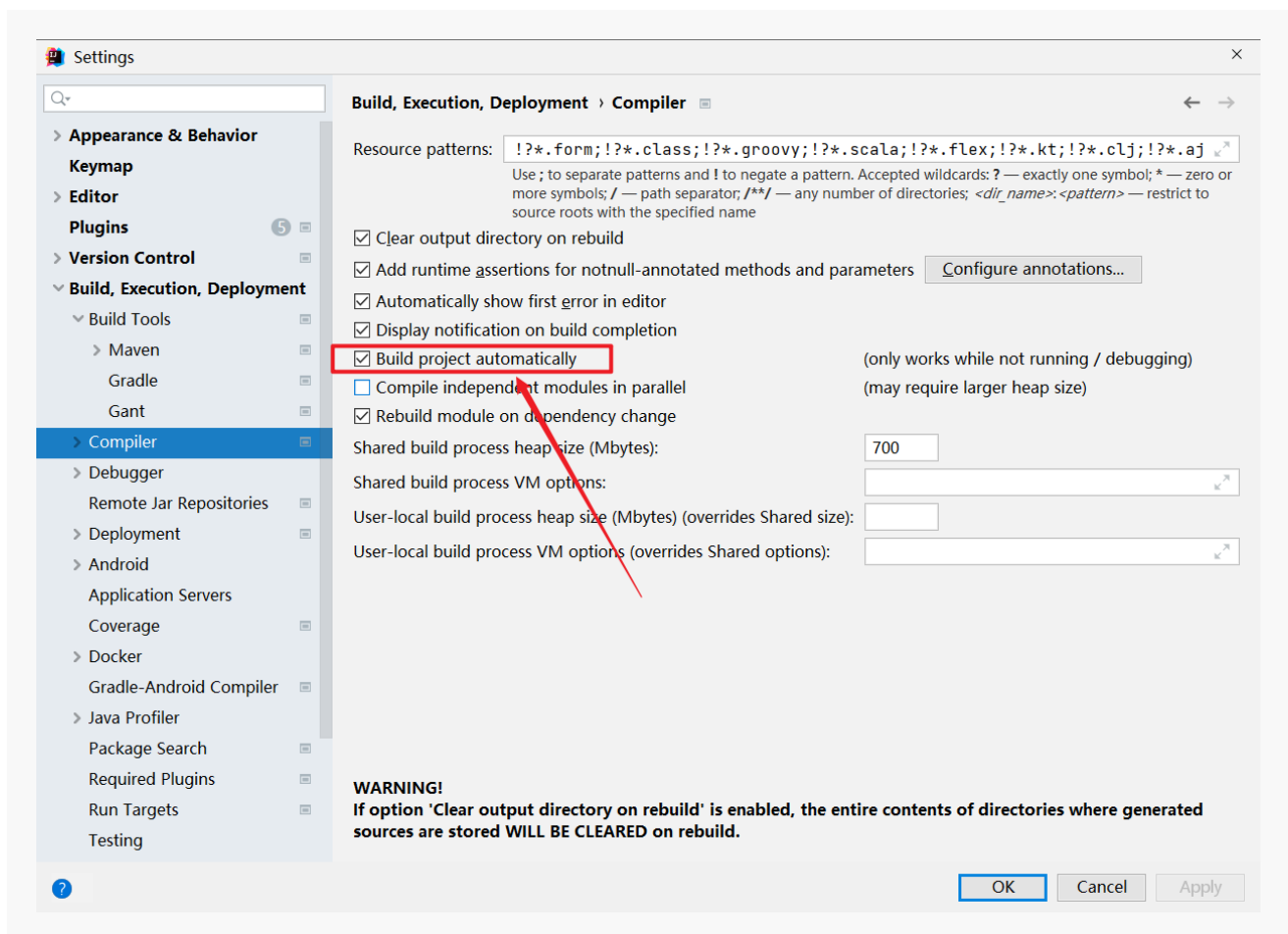


我给未来新建的maven工程，提供一劳永逸的配置。如下：

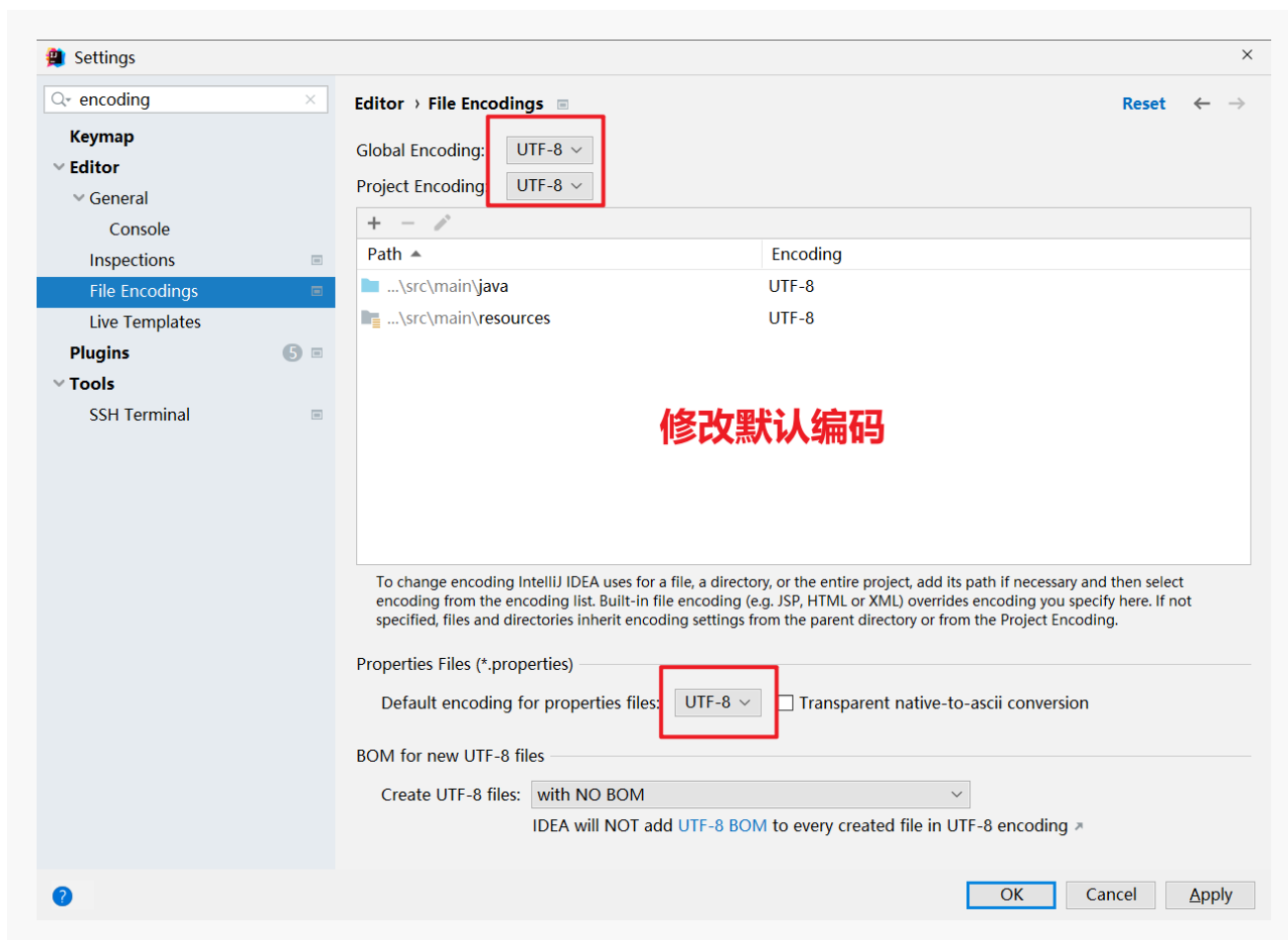
maven配置



自动编译：



修改默认编码



🍷 04、使用 **Idea** 快速搭建 **Spring Boot**

🍷 官网构建器

快速构建器: <https://start.spring.io/>

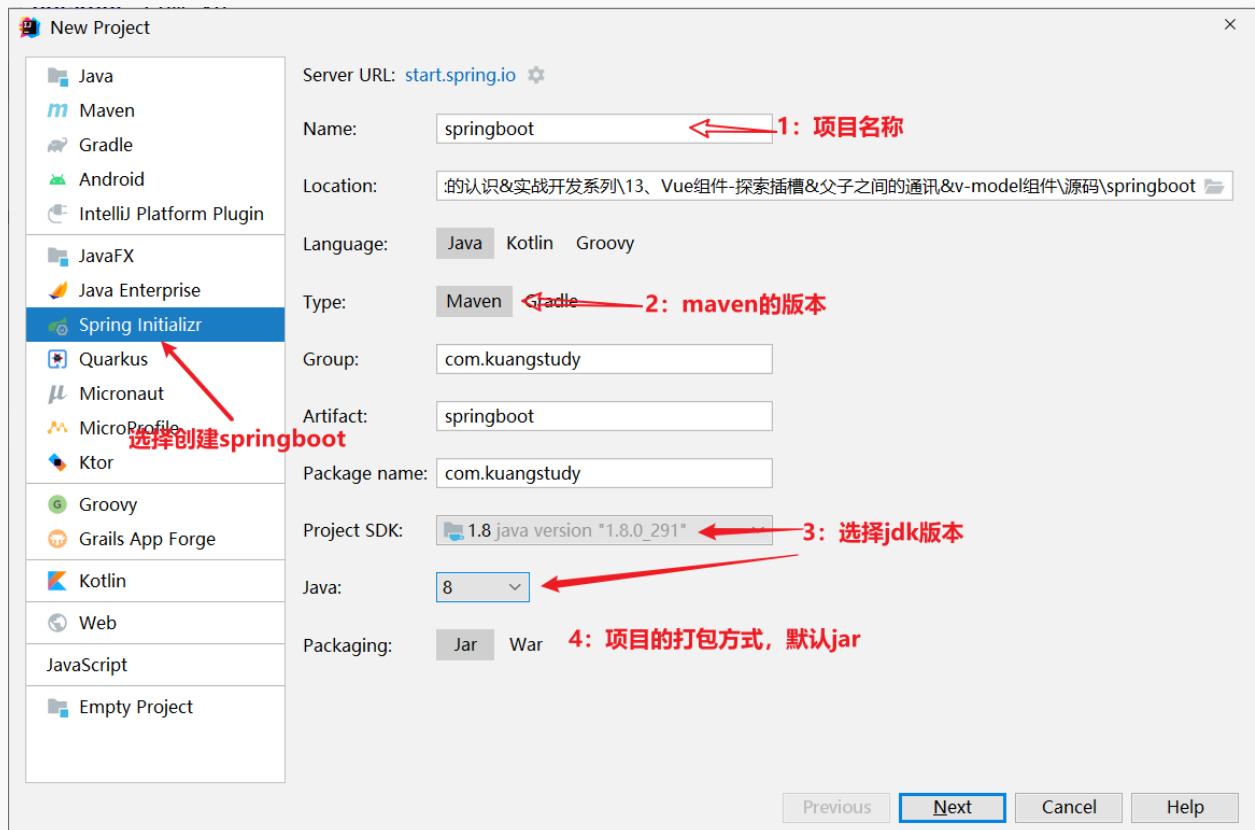
快速入门指南: <https://spring.io/quickstart>

👉 第一步：新建 **Spring Initializr** 项目：

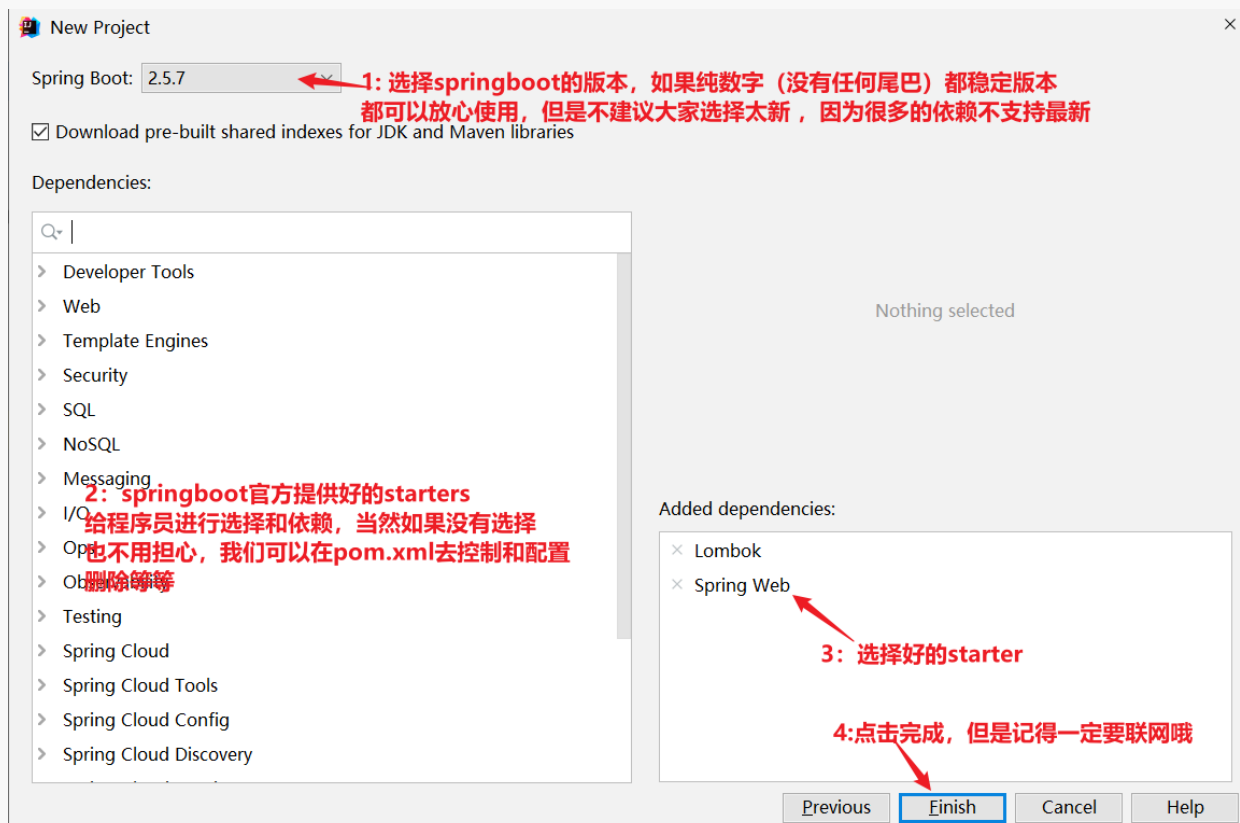
（1）选择 Spring Initializr



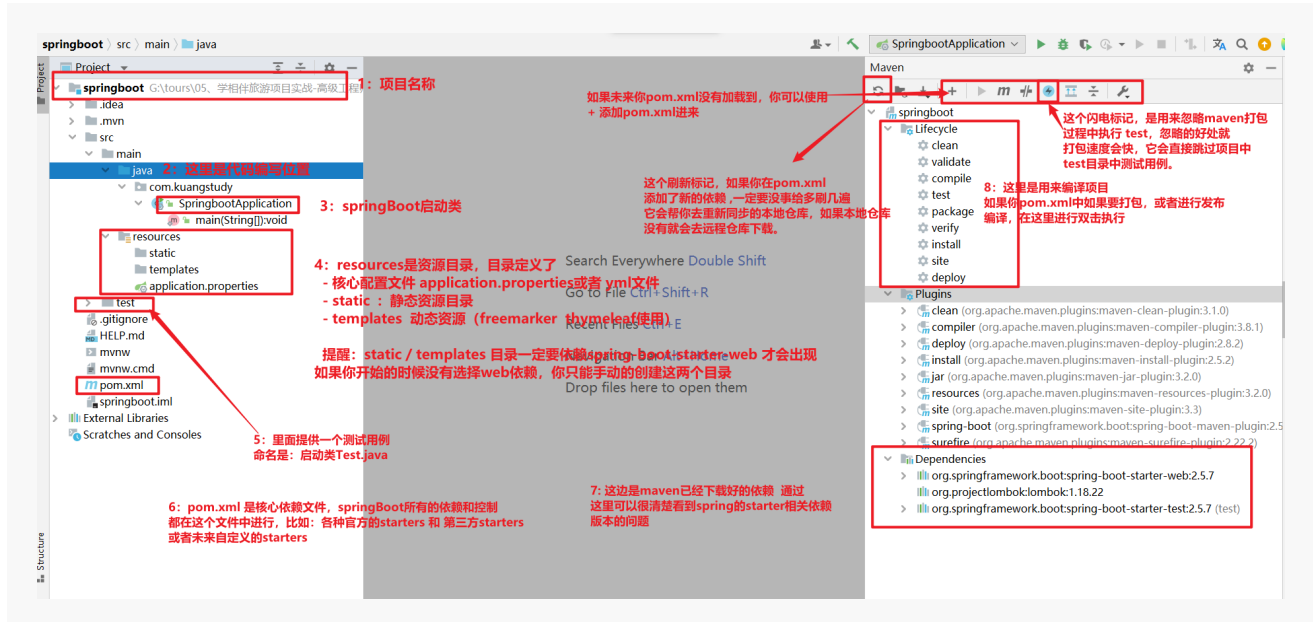
（2）选择 SDK，点击 **new** 这里就是使用 JAVA SDK 要求版本 1.8+，选择你默认安装在 C:\Program Files\Java\jdk1.8.0_191 目录：然后选择默认的 url（不用管）点击 **Next**：



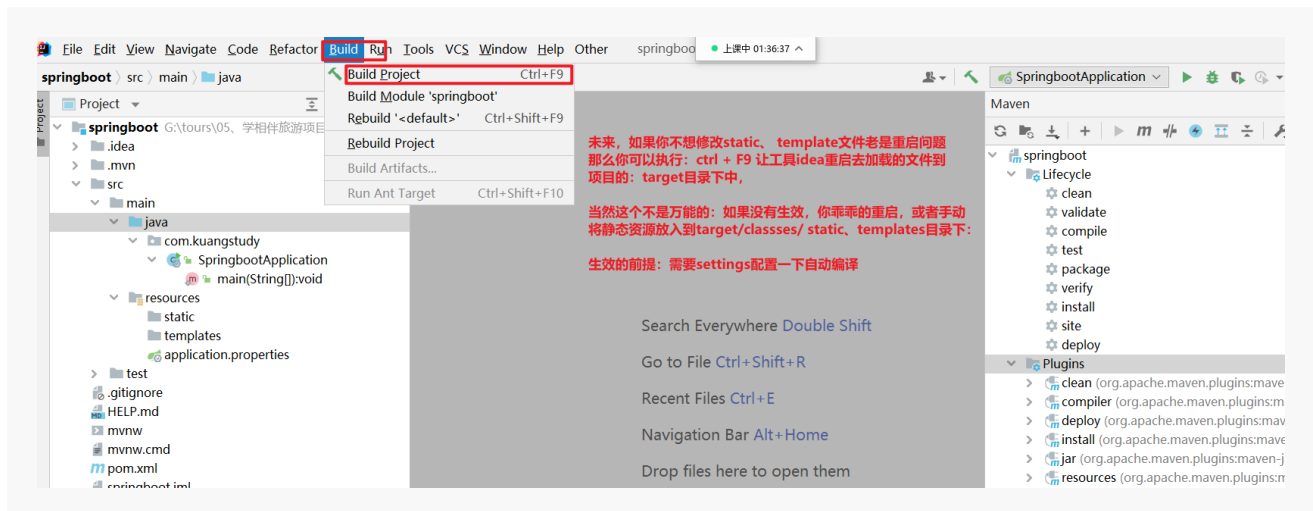
(3) 先勾选上 Web 依赖:

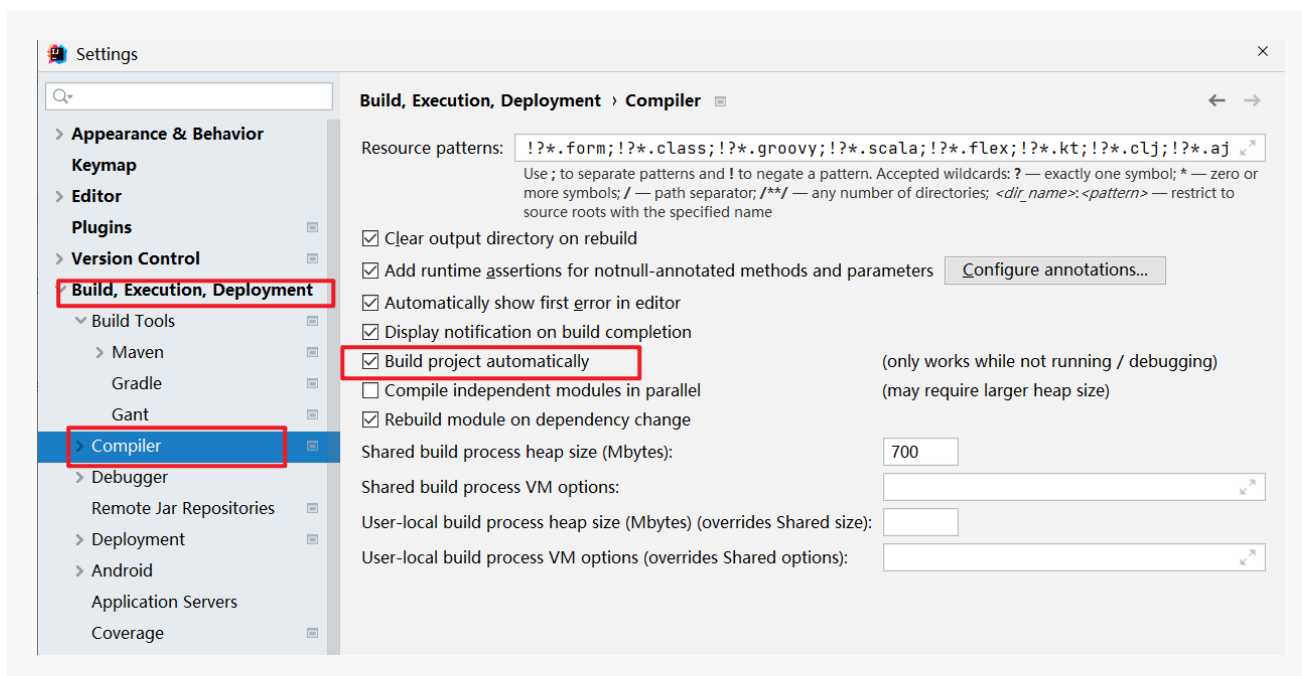


(4) 如果是第一次配置 Spring Boot 的话可能需要等待一会儿 IDEA 下载相应的 依赖包，默认创建好的项目结构如下：



(5) 关于静态资源修改不重启的问题。如下





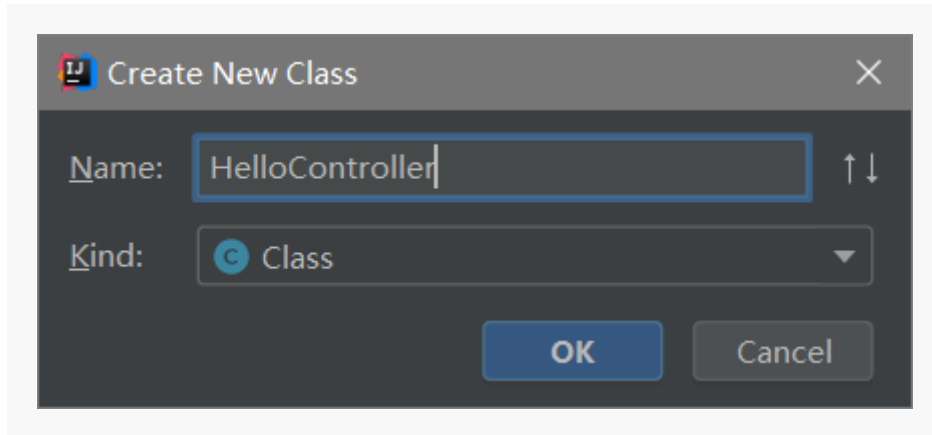
提醒：上面的自动加载，不包括修改类，添加方法，添加属性是不会有有效的。其实有一个自动热加载的工具 *jrebel* 建议不去玩，因为收费的。而且手动启动不会浪费很多时间，所以不要去研究。

项目结构还是看上去挺清爽的，少了很多配置文件，我们来了解一下默认生成的有什么：

- **SpringbootApplication**: 一个带有 `main()` 方法的类，用于启动应用程序
- **SpringbootApplicationTests**: 一个空的 Junit 测试了，它加载了一个使用 Spring Boot 字典配置功能的 Spring 应用程序上下文
- **application.properties**: 一个空的 properties 文件，可以根据需要添加配置属性
- **pom.xml**: Maven 构建说明文件

第二步：HelloController

在【main/java/com.kuangstudy.web】包下新建一个【HelloController】：

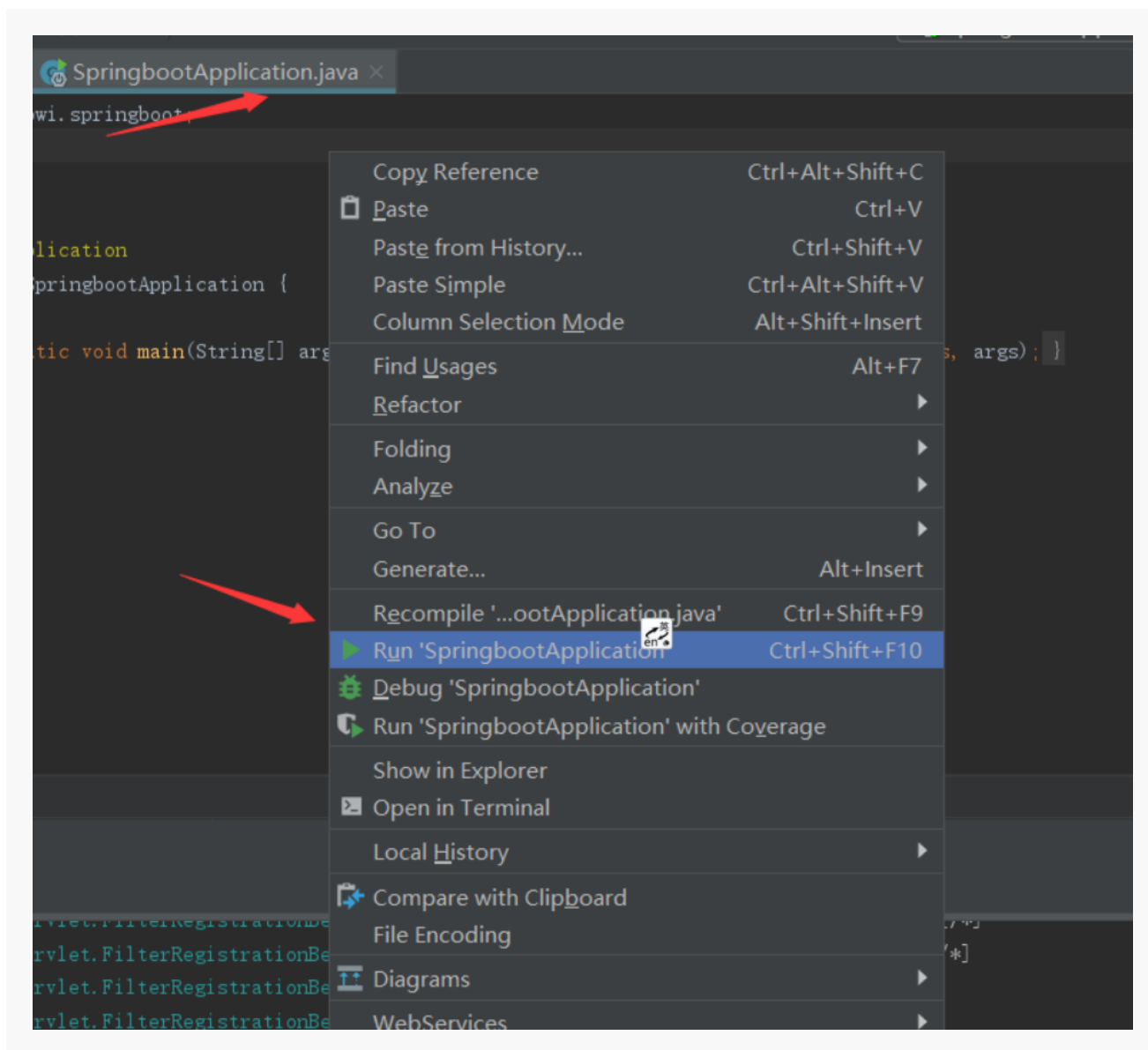


```
1 package com.kuangstudy.web;
2
3 import
  org.springframework.web.bind.annotation.GetMapping;
4 import
  org.springframework.web.bind.annotation.RestController;
5
6 /**
7  * Description:
8  * Author: yykk Administrator
9  * Version: 1.0
10 * Create Date Time: 2021/12/11 21:25.
11 * Update Date Time:
12 *
13 * @see
14 */
15 @RestController
```

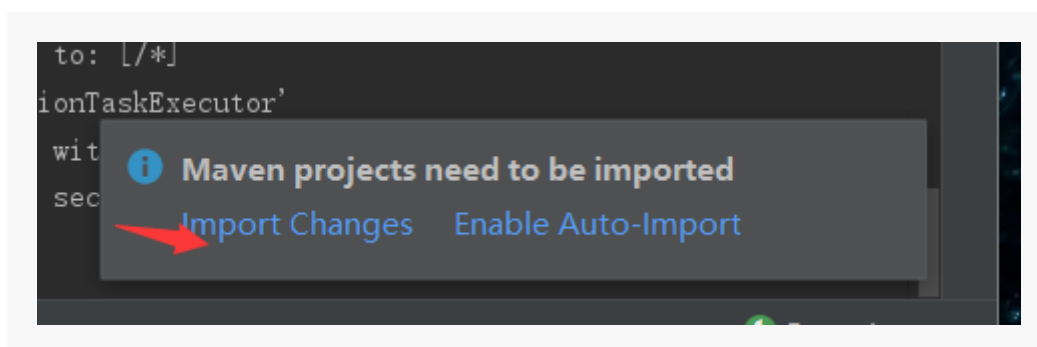
```
16 public class HelloWorld {  
17  
18     @GetMapping("/hello")  
19     public String hello(){  
20         return "Hello Springboot!!!";  
21     }  
22 }  
23
```

第三步：利用 **IDEA** 启动 **Spring Boot**

（1）我们回到 `SpringbootApplication` 这个类中，然后右键点击运行：



(2) 会提示 Maven 导包，点击 import



(3) 注意：我们之所以在上面的项目中没有手动的去配置 Tomcat 服务器，是因为 Spring Boot 内置了 Tomcat 等待一会儿就会看到下方的成功运行的提示信息：


```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project
  xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-
    instance"
3
  xsi:schemaLocation="http://maven.apache.org/POM/4
    .0.0 https://maven.apache.org/xsd/maven-
    4.0.0.xsd">
4     <modelVersion>4.0.0</modelVersion>
5     <!--springboot版本的控制-->
6     <parent>
7
8         <groupId>org.springframework.boot</groupId>
9         <artifactId>spring-boot-starter-
parent</artifactId>
10        <version>2.5.7</version> <!--如果未来你要升
级springboot修改此处的版本就可以了-->
11        <relativePath/>
12    </parent>
13    <!--项目的maven的坐标骨架，把当前项目生成在本地仓库的
目录-->
14    <groupId>com.kuangstudy</groupId>
15    <artifactId>springboot</artifactId>
16    <version>0.0.1-SNAPSHOT</version>
17    <name>springboot</name>
18    <description>springboot</description>
19
20    <!--maven属性配置，一般用来做version控制居多-->
21    <properties>
        <java.version>1.8</java.version>
```



```
22     </properties>
23
24     <!--项目依赖-->
25     <dependencies>
26         <!--springboot的web的starter-->
27         <dependency>
28
29             <groupId>org.springframework.boot</groupId>
30             <artifactId>spring-boot-starter-
web</artifactId>
31             </dependency>
32
33             <!--lombok-->
34             <dependency>
35                 <groupId>org.projectlombok</groupId>
36                 <artifactId>lombok</artifactId>
37                 <optional>true</optional>
38             </dependency>
39
40             <!--springboot的test的starter-->
41             <dependency>
42
43                 <groupId>org.springframework.boot</groupId>
44                 <artifactId>spring-boot-starter-
test</artifactId>
45                 <scope>test</scope>
46             </dependency>
47
48         </dependencies>
49
50     </build>
```

```

49         <!--springboto的插件，用来打包和构建使用-->
50         <plugins>
51             <plugin>
52
53                 <groupId>org.springframework.boot</groupId>
54                 <artifactId>spring-boot-maven-
55 plugin</artifactId>
56                 <configuration>
57                     <excludes>
58                         <exclude>
59
60                             <groupId>org.projectlombok</groupId>
61                             <artifactId>lombok</artifactId>
62                         </exclude>
63                     </excludes>
64                 </configuration>
65             </plugin>
66         </plugins>
67     </build>
68 </project>

```

1: 这个是当前项目的坐标骨架

2: 执行install的时候，它会在本地仓库中把你工程jar放入到仓库中

3: 生成好的jar

(2) 我们可以看到一个比较陌生一些的标签，这个标签是在配置 Spring Boot 的父级依赖：

```
1 <parent>
2     <groupId>org.springframework.boot</groupId>
3     <artifactId>spring-boot-starter-
parent</artifactId>
4     <version>2.5.8</version>
5     <relativePath/>
6 </parent>
```

有了这个，当前的项目才是 Spring Boot 项目，spring-boot-starter-parent 是一个特殊的 starter，它用来提供相关的 Maven 默认依赖，使用它之后，常用的包依赖就可以省去 version 标签。

下面的版本为什么不加 version 号：

```
1 <!--springboot的web的starter-->
2 <dependency>
3     <groupId>org.springframework.boot</groupId>
4     <artifactId>spring-boot-starter-
web</artifactId>
5 </dependency>
```

因为：spring-boot-starter-parent 是一个父工程，spring-boot-starter-web 和 spring-boot-starter-test 等都是子工程，而子工程的版本都已经由父工程已经管理起来了。所以在项目开发中我们就需要指定版本号。那到底 parent 工程管理了多少呢？点击进去查看一下呗：其实马上要讲的 starter。

```
1 <activemq.version>5.16.3</activemq.version>
2   <antlr2.version>2.7.7</antlr2.version>
3   <appengine-sdk.version>1.9.91</appengine-
  sdk.version>
4   <artemis.version>2.17.0</artemis.version>
5   <aspectj.version>1.9.7</aspectj.version>
6   <assertj.version>3.19.0</assertj.version>
7   <atomikos.version>4.0.6</atomikos.version>
8
9   <awaitility.version>4.0.3</awaitility.version>
10  <build-helper-maven-
  plugin.version>3.2.0</build-helper-maven-
  plugin.version>
11  <byte-buddy.version>1.10.22</byte-
  buddy.version>
12  <caffeine.version>2.9.2</caffeine.version>
13  <cassandra-driver.version>4.11.3</cassandra-
  driver.version>
14  <classmate.version>1.5.1</classmate.version>
15  <commons-codec.version>1.15</commons-
  codec.version>
16  <commons-dbcp2.version>2.8.0</commons-
  dbcp2.version>
17  <commons-lang3.version>3.12.0</commons-
  lang3.version>
18  <commons-pool.version>1.6</commons-
  pool.version>
19  <commons-pool2.version>2.9.0</commons-
  pool2.version>
20  <couchbase-client.version>3.1.7</couchbase-
  client.version>
```

```
20     <db2-jdbc.version>11.5.6.0</db2-  
    jdbc.version>  
21     <dependency-management-  
    plugin.version>1.0.11.RELEASE</dependency-  
    management-plugin.version>  
22     <derby.version>10.14.2.0</derby.version>  
23     <dropwizard-  
    metrics.version>4.1.25</dropwizard-  
    metrics.version>  
24     <ehcache.version>2.10.9.2</ehcache.version>  
25     <ehcache3.version>3.9.5</ehcache3.version>  
26  
    <elasticsearch.version>7.12.1</elasticsearch.ve  
    rsion>  
27     <embedded-mongo.version>3.0.0</embedded-  
    mongo.version>  
28     <flyway.version>7.7.3</flyway.version>  
29  
    <freemarker.version>2.3.31</freemarker.version>  
30     <git-commit-id-plugin.version>4.0.5</git-  
    commit-id-plugin.version>  
31     <glassfish-el.version>3.0.3</glassfish-  
    el.version>  
32     <glassfish-jaxb.version>2.3.5</glassfish-  
    jaxb.version>  
33     <groovy.version>3.0.8</groovy.version>  
34     <gson.version>2.8.7</gson.version>  
35     <h2.version>1.4.200</h2.version>  
36     <hamcrest.version>2.2</hamcrest.version>  
37     <hazelcast.version>4.1.5</hazelcast.version>
```

```
38     <hazelcast-
hibernate5.version>2.2.1</hazelcast-
hibernate5.version>
39
    <hibernate.version>5.4.32.Final</hibernate.vers
ion>
40     <hibernate-
validator.version>6.2.0.Final</hibernate-
validator.version>
41     <hikaricp.version>4.0.3</hikaricp.version>
42     <hsqldb.version>2.5.2</hsqldb.version>
43     <htmlunit.version>2.49.1</htmlunit.version>
44
    <httpasyncclient.version>4.1.4</httpasyncclient
.version>
45
    <httpClient.version>4.5.13</httpClient.version>
46
    <httpClient5.version>5.0.4</httpClient5.version
>
47     <httpcore.version>4.4.14</httpcore.version>
48     <httpcore5.version>5.1.1</httpcore5.version>
49
    <infinispan.version>12.1.7.Final</infinispan.ve
rsion>
50     <influxdb-java.version>2.21</influxdb-
java.version>
51     <jackson-bom.version>2.12.4</jackson-
bom.version>
52     <jakarta-activation.version>1.2.2</jakarta-
activation.version>
```

```
53      <jakarta-annotation.version>1.3.5</jakarta-  
annotation.version>  
54      <jakarta-jms.version>2.0.3</jakarta-  
jms.version>  
55      <jakarta-json.version>1.1.6</jakarta-  
json.version>  
56      <jakarta-json-bind.version>1.0.2</jakarta-  
json-bind.version>  
57      <jakarta-mail.version>1.6.7</jakarta-  
mail.version>  
58      <jakarta-persistence.version>2.2.3</jakarta-  
persistence.version>  
59      <jakarta-servlet.version>4.0.4</jakarta-  
servlet.version>  
60      <jakarta-servlet-jsp-  
jstl.version>1.2.7</jakarta-servlet-jsp-  
jstl.version>  
61      <jakarta-transaction.version>1.3.3</jakarta-  
transaction.version>  
62      <jakarta-validation.version>2.0.2</jakarta-  
validation.version>  
63      <jakarta-websocket.version>1.1.2</jakarta-  
websocket.version>  
64      <jakarta-ws-rs.version>2.1.6</jakarta-ws-  
rs.version>  
65      <jakarta-xml-bind.version>2.3.3</jakarta-  
xml-bind.version>  
66      <jakarta-xml-soap.version>1.4.2</jakarta-  
xml-soap.version>  
67      <jakarta-xml-ws.version>2.3.3</jakarta-xml-  
ws.version>
```

```
68     <janino.version>3.1.6</janino.version>
69     <javax-activation.version>1.2.0</javax-
activation.version>
70     <javax-annotation.version>1.3.2</javax-
annotation.version>
71     <javax-cache.version>1.1.1</javax-
cache.version>
72     <javax-jaxb.version>2.3.1</javax-
jaxb.version>
73     <javax-jaxws.version>2.3.1</javax-
jaxws.version>
74     <javax-jms.version>2.0.1</javax-jms.version>
75     <javax-json.version>1.1.4</javax-
json.version>
76     <javax-jsonb.version>1.0</javax-
jsonb.version>
77     <javax-mail.version>1.6.2</javax-
mail.version>
78     <javax-money.version>1.1</javax-
money.version>
79     <javax-persistence.version>2.2</javax-
persistence.version>
80     <javax-transaction.version>1.3</javax-
transaction.version>
81     <javax-
validation.version>2.0.1.Final</javax-
validation.version>
82     <javax-websocket.version>1.1</javax-
websocket.version>
83     <jaxen.version>1.2.0</jaxen.version>
```


84

<jaybird.version>4.0.3.java8</jaybird.version>

85

<jboss-logging.version>3.4.2.Final</jboss-logging.version>

86

<jboss-transaction-spi.version>7.6.1.Final</jboss-transaction-spi.version>

87

<jdom2.version>2.0.6</jdom2.version>

88

<jedis.version>3.6.3</jedis.version>

89

<jersey.version>2.33</jersey.version>

90

<jetty-e1.version>9.0.48</jetty-e1.version>

91

<jetty-jsp.version>2.2.0.v201112011158</jetty-jsp.version>

92

<jetty-reactive-httpclient.version>1.1.10</jetty-reactive-httpclient.version>

93

<jetty.version>9.4.43.v20210629</jetty.version>

94

<jmustache.version>1.15</jmustache.version>

95

<johnzon.version>1.2.14</johnzon.version>

96

<jolokia.version>1.6.2</jolokia.version>

97

<jooq.version>3.14.13</jooq.version>

98

<json-path.version>2.5.0</json-path.version>

99

<json-smart.version>2.4.7</json-smart.version>

100

<jsonassert.version>1.5.0</jsonassert.version>

101

<jstl.version>1.2</jstl.version>

102

<jtds.version>1.3.1</jtds.version>

103

<junit.version>4.13.2</junit.version>

```
104     <junit-jupiter.version>5.7.2</junit-  
jupiter.version>  
105     <kafka.version>2.7.1</kafka.version>  
106     <kotlin.version>1.5.21</kotlin.version>  
107     <kotlin-coroutines.version>1.5.1</kotlin-  
coroutines.version>  
108  
    <lettuce.version>6.1.4.RELEASE</lettuce.version  
>  
109     <liquibase.version>4.3.5</liquibase.version>  
110     <log4j2.version>2.14.1</log4j2.version>  
111     <logback.version>1.2.5</logback.version>  
112     <lombok.version>1.18.20</lombok.version>  
113     <mariadb.version>2.7.4</mariadb.version>  
114     <maven-antrun-plugin.version>1.8</maven-  
antrun-plugin.version>  
115     <maven-assembly-plugin.version>3.3.0</maven-  
assembly-plugin.version>  
116     <maven-clean-plugin.version>3.1.0</maven-  
clean-plugin.version>  
117     <maven-compiler-plugin.version>3.8.1</maven-  
compiler-plugin.version>  
118     <maven-dependency-  
plugin.version>3.1.2</maven-dependency-  
plugin.version>  
119     <maven-deploy-plugin.version>2.8.2</maven-  
deploy-plugin.version>  
120     <maven-enforcer-plugin.version>3.0.0</maven-  
enforcer-plugin.version>
```

```
121     <maven-failsafe-  
plugin.version>2.22.2</maven-failsafe-  
plugin.version>  
122     <maven-help-plugin.version>3.2.0</maven-  
help-plugin.version>  
123     <maven-install-plugin.version>2.5.2</maven-  
install-plugin.version>  
124     <maven-invoker-plugin.version>3.2.2</maven-  
invoker-plugin.version>  
125     <maven-jar-plugin.version>3.2.0</maven-jar-  
plugin.version>  
126     <maven-javadoc-plugin.version>3.2.0</maven-  
javadoc-plugin.version>  
127     <maven-resources-  
plugin.version>3.2.0</maven-resources-  
plugin.version>  
128     <maven-shade-plugin.version>3.2.4</maven-  
shade-plugin.version>  
129     <maven-source-plugin.version>3.2.1</maven-  
source-plugin.version>  
130     <maven-surefire-  
plugin.version>2.22.2</maven-surefire-  
plugin.version>  
131     <maven-war-plugin.version>3.3.1</maven-war-  
plugin.version>  
132  
    <micrometer.version>1.7.3</micrometer.version>  
133    <mimepull.version>1.9.15</mimepull.version>  
134    <mockito.version>3.9.0</mockito.version>  
135    <mongodb.version>4.2.3</mongodb.version>
```

```
136     <mssql-jdbc.version>9.2.1.jre8</mssql-jdbc.version>
137     <mysql.version>8.0.26</mysql.version>
138     <nekohtml.version>1.9.22</nekohtml.version>
139     <neo4j-java-driver.version>4.2.7</neo4j-java-driver.version>
140     <netty.version>4.1.67.Final</netty.version>
141     <netty-tcnative.version>2.0.40.Final</netty-tcnative.version>
142     <oauth2-oidc-sdk.version>9.9.1</oauth2-oidc-sdk.version>
143     <nimbus-jose-jwt.version>9.10.1</nimbus-jose-jwt.version>
144     <ojdbc.version>19.3.0.0</ojdbc.version>
145     <okhttp3.version>3.14.9</okhttp3.version>
146     <oracle-database.version>21.1.0.0</oracle-database.version>
147     <pooled-jms.version>1.2.2</pooled-jms.version>
148     <postgresql.version>42.2.23</postgresql.version>
149     <prometheus-pushgateway.version>0.10.0</prometheus-pushgateway.version>
150     <quartz.version>2.3.2</quartz.version>
151     <querydsl.version>4.4.0</querydsl.version>
152     <r2dbc-bom.version>Arabba-SR10</r2dbc-bom.version>
153     <rabbit-amqp-client.version>5.12.0</rabbit-amqp-client.version>
```

```
154     <reactive-streams.version>1.0.3</reactive-  
streams.version>  
155     <reactor-bom.version>2020.0.10</reactor-  
bom.version>  
156     <rest-assured.version>4.3.3</rest-  
assured.version>  
157     <rsocket.version>1.1.1</rsocket.version>  
158     <rxjava.version>1.3.8</rxjava.version>  
159     <rxjava-adapter.version>1.2.1</rxjava-  
adapter.version>  
160     <rxjava2.version>2.2.21</rxjava2.version>  
161     <saaj-impl.version>1.5.3</saaj-impl.version>  
162  
    <selenium.version>3.141.59</selenium.version>  
163     <selenium-htmlunit.version>2.49.1</selenium-  
htmlunit.version>  
164     <sendgrid.version>4.7.4</sendgrid.version>  
165     <servlet-api.version>4.0.1</servlet-  
api.version>  
166     <slf4j.version>1.7.32</slf4j.version>  
167     <snakeyaml.version>1.28</snakeyaml.version>  
168     <solr.version>8.8.2</solr.version>  
169     <spring-amqp.version>2.3.10</spring-  
amqp.version>  
170     <spring-batch.version>4.3.3</spring-  
batch.version>  
171     <spring-data-bom.version>2021.0.4</spring-  
data-bom.version>  
172     <spring-framework.version>5.3.9</spring-  
framework.version>
```

```
173     <spring-hateoas.version>1.3.3</spring-  
hateoas.version>  
174     <spring-integration.version>5.5.3</spring-  
integration.version>  
175     <spring-kafka.version>2.7.6</spring-  
kafka.version>  
176     <spring-ldap.version>2.3.4.RELEASE</spring-  
ldap.version>  
177     <spring-  
restdocs.version>2.0.5.RELEASE</spring-  
restdocs.version>  
178     <spring-retry.version>1.3.1</spring-  
retry.version>  
179     <spring-security.version>5.5.2</spring-  
security.version>  
180     <spring-session-  
bom.version>2021.0.2</spring-session-  
bom.version>  
181     <spring-ws.version>3.1.1</spring-ws.version>  
182     <sqlite-jdbc.version>3.34.0</sqlite-  
jdbc.version>  
183     <sun-mail.version>1.6.7</sun-mail.version>  
184  
    <thymeleaf.version>3.0.12.RELEASE</thymeleaf.ve  
rsion>  
185     <thymeleaf-extras-data-  
attribute.version>2.0.1</thymeleaf-extras-data-  
attribute.version>  
186     <thymeleaf-extras-  
java8time.version>3.0.4.RELEASE</thymeleaf-  
extras-java8time.version>
```

```
187     <thymeleaf-extras-  
springsecurity.version>3.0.4.RELEASE</thymeleaf-  
extras-springsecurity.version>  
188     <thymeleaf-layout-  
dialect.version>2.5.3</thymeleaf-layout-  
dialect.version>  
189     <tomcat.version>9.0.52</tomcat.version>  
190     <unboundid-  
ldapsdk.version>4.0.14</unboundid-  
ldapsdk.version>  
191     <undertow.version>2.2.10.Final</undertow.version>  
192     <versions-maven-  
plugin.version>2.8.1</versions-maven-  
plugin.version>  
193     <webjars-hal-  
browser.version>3325375</webjars-hal-  
browser.version>  
194     <webjars-locator-core.version>0.46</webjars-  
locator-core.version>  
195     <wsdl4j.version>1.6.3</wsdl4j.version>
```

那什么情况下的依赖需要去指定版本号呢？

- 不想用这个版本号的时候
- 第三方开发的**starter**就必须指定版本号。

```
1 <dependency>
2     <groupId>com.baomidou</groupId>
3     <artifactId>mybatis-plus-boot-
  starter</artifactId>
4     <version>3.4.2</version>
5 </dependency>
```

- 上面没有提供的就必须指定版本号
 - 首先我不加版本号，看能不能下载，如果能拿忽略
 - 如果不加下载不下例子啊，我就就指定版本号。
- 可能版本出来冲突，也可能会指定版本号。

关于具体 Spring Boot 提供了哪些 jar 包的依赖，我们可spring-boot-starter-web以查看本地 Maven 仓库下：

\repository\org\springframework\boot\spring-boot-dependencies\2.5.7RELEASE\spring-boot-dependencies-2.5.7.RELEASE.pom 文件来查看，挺长的...

二、应用入口类 **SpringbootApplication**&核心注解

官网：<https://docs.spring.io/spring-boot/docs/2.5.8-SNAPSHOT/reference/htmlsingle/#getting-started.installing>

许多 Spring Boot 开发人员喜欢他们的应用程序使用自动配置、组件扫描并能够在他们的“应用程序类”上定义额外的配置。

@SpringBootApplication 可以使用单个注释来启用这三个功能，即：


```
1 @SpringBootConfiguration
2 @EnableAutoConfiguration
3 @ComponentScan
```

- **@EnableAutoConfiguration**: 启用Spring Boot 的自动配置机制 其实告诉为什么项目里能够自动把各种starter依赖进来以后就能够使用和生效。就是这个注解来完成。
- **@ComponentScan:@Component** 在应用程序所在的包上启用扫描（查看[最佳实践](#)）在传统的spring开发中，我们如果要进行包的扫描，就必须去在配置文件xml指定。如下：

```
1 <component-scan base-package="com.xxx.service">
  </component-scan>
2 <component-scan base-package="com.xxx.dao">
  </component-scan>
3 <component-scan base-package="com.xxx.web">
  </component-scan>
```

上面的方式是传统的加载bean到springioc容器的方式，目的是告诉spring容器在启动的时候会去扫描上面的三个目录，然后找到这个包下所有的类，并且加了@Service，@Component、@Controller、@Repository等，会加入ioc容器，其它不会加载。

*springboot*的默认加载机制：是以当前启动的类包作为 *component-scan* 默认扫描的包。

- **@SpringBootConfiguration**: 启用在上下文中注册额外的 bean 或导入额外的配置类。Spring 标准的替代方案 **@Configuration**，可帮助您在集成测试中[检测配置](#)。

```

1 @SpringBootApplication // same as
  @SpringBootConfiguration @EnableAutoConfiguration
  @ComponentScan
2 public class MyApplication {
3
4     public static void main(String[] args) {
5         SpringApplication.run(MyApplication.class,
  args);
6     }
7
8 }

```

- 配置类必须加注解@Configuration 或者 @SpringBootConfiguration
- 和@ComponentScan 的作用是一样的。

总结

springboot核心注解：@SpringBootApplication，它是由

- @ComponentScan
- @SpringConguration
- @EnableAutoConfiguration

三个注解的复合。三个注解的作用：都是把项目中bean，第三容器的bean，把官方提供starter的配置类的bean加载springioc容器的作用：

- **@ComponentScan**: 是把你项目中，自己编写的那些bean加载ioc容器中，比如：UserService,UserMapper.UserConntroller
- **@EnableAutoConfiguration** : 是把官方提供starter里面，内置的配置类的bean加载ioc容器冲
 - 内部提供的配置类：xxxxAutoConfiguration 比如：
 - RedisAutoConfiguration 这些都配置类
- **@SpringConfiguration: +@Bean** （避免重复造轮子）
 - 如果你对官方的starter提供的配置不满意，你可以考虑用这个放去覆盖内部的配置。
 - 或者未来你要自己去扩展starter机制，就必须自己去定义配置类。（自定义starter）
 - 方便扩展，可以便于后续去开发的依赖公共模板
 - 或者未来你想自定义starter你就可以用配置类完成。
 - 传统的方式的扩展，通过xml去配置，配置类就是xml的替代。
 - **@SpringConfiguration+@Bean** 更深层次含义：就说官方没提供的你自己去扩展把。
 - **@Bean必须要配置配置,或者@Component组件或者其子组件都有用。否则无意义。**但是还推荐：配置注解

最后：无论上面那种方式，其目的都是把项目中，其他人写好的，或官方的提供的bean记载到ioc容器中。

三、SpringBoot的全局配置文件

查看官网地址：<https://docs.spring.io/spring-boot/docs/2.5.8-SNAPSHOT/reference/htmlsingle/#application-properties>

Spring Boot 使用一个全局的配置文件 `application.properties` 或 `application.yml`，放置在【`src/main/resources`】目录或者类路径的 `/config` 下。Spring Boot 不仅支持常规的 `properties` 配置文件，还支持 `yaml` 语言的配置文件。`yaml` 是以数据为中心的语言，在配置数据的时候具有面向对象的特征。Spring Boot 的全局配置文件的作用是对一些默认配置的配置值进行修改

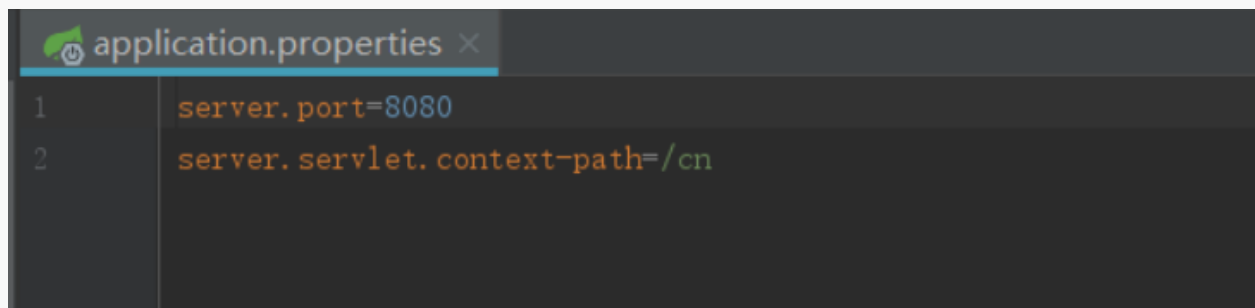
 写法：

```
1 server:
2   port: 8081
```

上方的 `: value` 的时候，一定至少要有有一个空格。

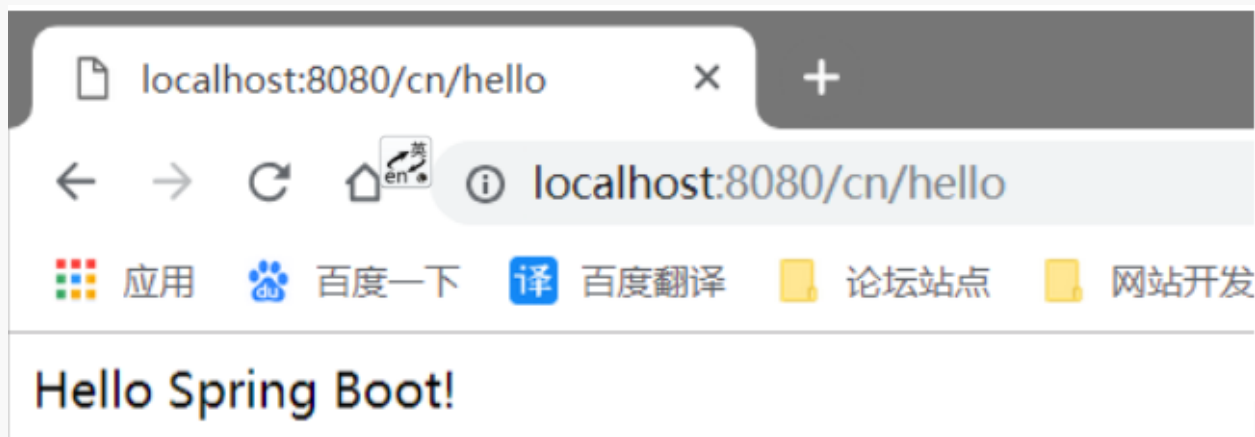
修改 `properties` 配置文件实例：

(1) 打开 `resources` 下的 `application.yml`

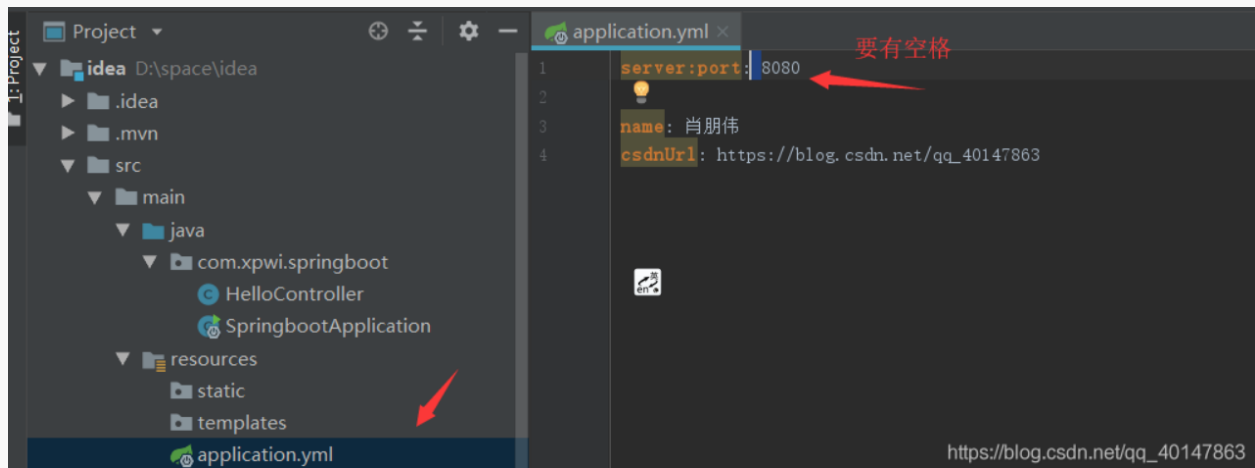
A screenshot of a text editor showing the contents of an application.properties file. The file has two lines: line 1 contains 'server.port=8080' and line 2 contains 'server.servlet.context-path=/cn'. The text is color-coded: 'server' is orange, 'port' is blue, '8080' is green, 'servlet' is orange, 'context-path' is blue, and '/cn' is green.

```
1 server.port=8080
2 server.servlet.context-path=/cn
```

（2）在这里我们可以设置访问的端口，将 Tomcat 默认端口设置为 8080（默认的不修改），并将默认访问路径从“/”修改为“/cn”时，再访问 <http://localhost:8080/> 是什么都没有的，此时要访问 hello 是要使用 <http://localhost:8080/cn/hello>



（3）使用 **yml** 文件作为配置文件，我们直接把 **.properties** 后缀的文件删掉，使用 **.yml** 文件来进行简单的配置



(4) 在然后使用在我们的 HelloController.java 类中使用 @Value 来获取配置属性，代码（请看注释）：

```
1 package com.xpwi.springboot;
2
3 import
  org.springframework.beans.factory.annotation.Value
  e;
4 import
  org.springframework.web.bind.annotation.RequestMapping;
5 import
  org.springframework.web.bind.annotation.RestController;
6
7 /**
8  * 测试控制器
9  *
10 * @author: @肖朋伟CSDN
11 * @create: 2018-11-18
12 */
13 @RestController
```

```
14 public class HelloController {
15
16     // 获取.yml 文件中值
17     @value("${name}")
18     private String name;
19
20     // 获取 age
21     @value("${csdnUrl}")
22     private String csdnUrl;
23
24     //路径映射，对应浏览器访问的地址，访问该路径则执行下面
    函数
25     @RequestMapping("/hello")
26     public String hello() {
27         return name + " CSDN 博客: " + csdnUrl;
28     }
29 }
30
```

(5) 重启 Spring Boot ，输入地址：<http://localhost:8080/hello> 能看到正确的结果：

总结

大家可能会有疑虑，springboot不是说没有xml配置文件吗？那 application.yml 是什么东西？

传统的所谓spring的xml文件是指，用初始化bean的文件，而application.yml它仅仅只是一个属性配置文件。因为官方提供了很多的starter（我官方提供很多的配置类），但是这个配置的信息我是不可能写死的，所以我提供配置属性类，让你进行扩展和覆盖。所以这个application.yml是一个全局属性配置文件，在这里配置的信息是覆盖starter中提供配置的类的信息。

1 RedisAutoConfiguration.java + RedisProperties.java

RedisProperties.java : 属性配置类。它的配置就是application.yml去配置信息，

*如果没有这个机制。你思考这个问题吗？我们指定连接redis需要指定ip和端口，和密码？假设你提供的ip: 152.150.47.5 port:6378 pwd:154545 .. 思考我如何把这ip、端口、密码等让RedisAutoConfiguration.java去知晓。你不可能说去改源码重新编译。如果是这样的话，我还不自己去写一个RedisConfiguration.java。所以官方已经想的很清楚了。每个配置类如果未来需要动态配置和修改里面的参数，就必须提供属性配置类: xxxProperties. 而我们在application.yml配置的属性都是给xxxProperties.java类的属性进行赋值。原理就是：

- 第一步：读取你项目的applicaiton.yml文件内容到内存中
- 第二步：然后通过前缀 spring.redis ，找到你的类xxxProperties。

```
1 @ConfigurationProperties(prefix="spring.redis")
2 public class RedisProperties(){
3     private String ip;
4 }
```



```
1 spring:
2     redis:
3         ip: 127.0.0.1
```

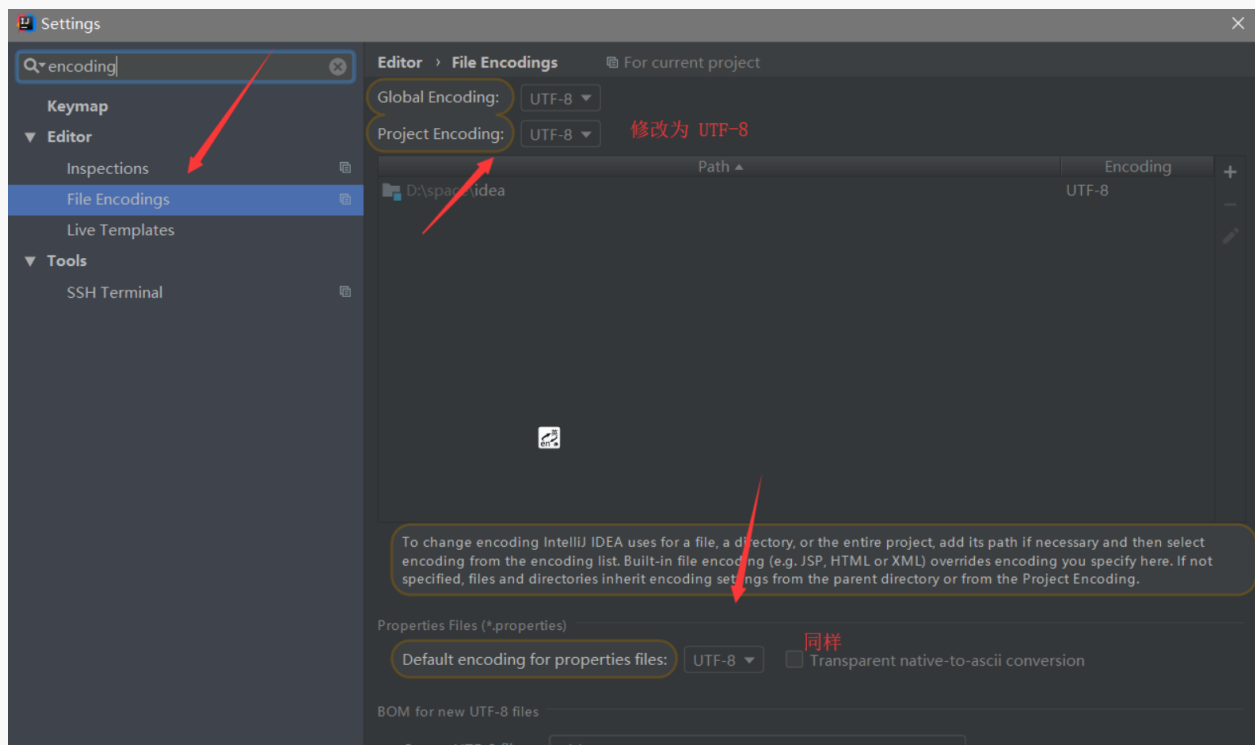
- 第三步：通过反射把对应配置文件的值，映射到xxxProperties的属性中去

总结

application.yml会把这个文件称之为：全属性配置文件

 **【注意】**：此时如果你第一次使用 **idea** 出现中文乱码，解决办法：

- 然后在修改idea的encoding统一成：UTF-8
- 如果遇到yml文件保存，先把里面中文全部剔除
- 如果实在不行，把文件删除，重新创建，然后复制和粘贴。



🧠 06、什么是Starters

Starters 是一组方便的依赖描述符，您可以将它们包含在您的应用程序中。您可以获得所需的所有 Spring 和相关技术的一站式服务，而无需搜索示例代码和复制粘贴加载的依赖项描述符。例如，如果您想开始使用 Spring 和 JPA 进行数据库访问，请 `spring-boot-starter-data-jpa` 在您的项目中包含依赖项。

👉 starters作用

- 解决第三个的依赖问题（一堆配置类，可以帮助我们解决很多初始化配置类的问题，我们可以做拿来主义者，不需要造轮子）
- 解决依赖版本的问题。就程序员不需要在去关注每个jar依赖另外一个jar包的问题，全部自动进行依赖管理version的统一升降

的问题。

starters 包含许多依赖项，您需要这些依赖项来快速启动和运行项目，并使用一组一致的、受支持的托管传递依赖项。

1 starters有什么

2

3 所有官方首发都遵循类似的命名模式：**spring-boot-starter-**
4 *****，其中*****是特定类型的应用程序。此命名结构旨在在您需要查找入门
5 时提供帮助。许多 IDE 中的 Maven 集成允许您按名称搜索依赖
项。例如，安装适当的 Eclipse 或 Spring Tools 插件后，您
可以**ctrl-space**在 POM 编辑器中按并键入“**spring-boot-**
starter”以获得完整列表。

4

5 正如“创建您自己的启动器”部分所述，第三方启动器不应以 开头
spring-boot，因为它是为官方 **Spring Boot** 工件保留的。相
反，第三方启动器通常以项目名称开头。例如，名为的第三方启动项
目**thirdpartyproject**通常命名为**thirdpartyproject-**
spring-boot-starter。

starters命名规范

- 官方的：**spring-boot-starter-***，在依赖的时候不需要指定版本号
- 第三个的或者自定义的：**xxx-boot-starter**，这个在开发的时候必须要指定版本号。比如；

```
1 mybatis-plus-boot-starter
```

官方提供的starters

以下应用启动器由 Spring Boot `org.springframework.boot` 组下提供:

NAME	描述
<code>spring-boot-starter</code>	核心启动器，包含 Tomcat、Spring MVC 和 Spring 记录器，以及 YAML 支持
<code>spring-boot-starter-activemq</code>	使用 Apache ActiveMQ 的入门者
<code>spring-boot-starter-amqp</code>	使用 Spring AMQP 的入门者
<code>spring-boot-starter-aop</code>	使用 Spring AOP 方面编程的入门者
<code>spring-boot-starter-artemis</code>	使用 Apache Artemis 的入门者
<code>spring-boot-starter-batch</code>	使用 Spring Batch 的入门者
<code>spring-boot-starter-cache</code>	使用 Spring Framework 的 Starter
<code>spring-boot-starter-data-cassandra</code>	使用 Cassandra 的 Starter Data Cassandra
<code>spring-boot-starter-data-cassandra-reactive</code>	Starter 使用 Cassandra Spring Data Cassandra
<code>spring-boot-starter-data-couchbase</code>	使用 Couchbase 的 Starter Spring Data Couchbase
<code>spring-boot-starter-data-couchbase-reactive</code>	使用 Couchbase 的 Starter Spring Data Couchbase

NAME	描述
	Spring Data Co 动器
spring-boot-starter-data-elasticsearch	使用 Elasticsea Spring Data El
spring-boot-starter-data-jdbc	使用 Spring Da
spring-boot-starter-data-jpa	将 Spring Data 使用的入门者
spring-boot-starter-data-ldap	使用 Spring Da
spring-boot-starter-data-mongodb	使用 MongoDB Spring Data Mo
spring-boot-starter-data-mongodb-reactive	Starter 使用 Mo 库和 Spring Da
spring-boot-starter-data-neo4j	使用 Neo4j 图开 Neo4j 的入门者
spring-boot-starter-data-r2dbc	使用 Spring Da
spring-boot-starter-data-redis	将 Redis 键值数 Redis 和 Lettuc 入门者
spring-boot-starter-data-redis-reactive	将 Redis 键值数 Redis 反应式和 用的启动器
spring-boot-starter-data-rest	使用 Spring Da Spring Data 存
spring-boot-starter-freemarker	使用 FreeMark 应用程序的初当
spring-boot-starter-groovy-templates	使用 Groovy 模

NAME	描述
<code>spring-boot-starter-hateoas</code>	使用 Spring MVC 构建基于超媒体程序的启动器
<code>spring-boot-starter-integration</code>	使用 Spring Integration
<code>spring-boot-starter-jdbc</code>	将 JDBC 与 Hikari 连接池一起使用
<code>spring-boot-starter-jersey</code>	使用 JAX-RS 和 Jersey 构建 Web 应用程序的启动器 spring-boot-starter-jersey
<code>spring-boot-starter-jooq</code>	使用 jOOQ 访问数据库的启动器。 spring-boot-starter-jooq 是 spring-boot-starter-jdbc 的替代品 spring-boot-starter-jooq
<code>spring-boot-starter-json</code>	读写 json 的 Starter
<code>spring-boot-starter-jta-atomikos</code>	使用 Atomikos JTA 的启动器
<code>spring-boot-starter-mail</code>	使用 Java Mail Framework 的启动器
<code>spring-boot-starter-mustache</code>	使用 Mustache 模板引擎的初学者
<code>spring-boot-starter-oauth2-client</code>	使用 Spring Security OAuth2/OpenID Connect 的入门者
<code>spring-boot-starter-oauth2-resource-server</code>	使用 Spring Security 服务器功能的入门者
<code>spring-boot-starter-quartz</code>	使用 Quartz 调度任务的启动器

NAME	描述
spring-boot-starter-rsocket	用于构建 RSocket Starter
spring-boot-starter-security	使用 Spring Security
spring-boot-starter-test	Starter 用于使用 JUnit、Hamcrest 和 Mockito 测试 Spring Boot 应用程序
spring-boot-starter-thymeleaf	使用 Thymeleaf 模板引擎的初学者友好型 Starter
spring-boot-starter-validation	将 Java Bean 验证器与 Spring Validator 结合使用
spring-boot-starter-web	使用 Spring MVC 和 Tomcat 构建 RESTful 应用
spring-boot-starter-web-services	使用 Spring Web Services 构建 SOAP 应用
spring-boot-starter-webflux	使用 Spring Framework 5 的 Web 支持构建 Web 应用
spring-boot-starter-websocket	使用 Spring Framework 5 的 Web 支持构建 WebSocket 应用

除了应用程序启动器之外，以下启动器还可用于添加[生产就绪](#)功能：

NAME	描述

NAME	描述
<code>spring-boot-starter-actuator</code>	使用 Spring Boot 的 Actuator 的启动器，它提供了生产就绪的特性来帮助你监控和管理你的应用程序

最后，Spring Boot 还包括以下启动器，如果您想排除或交换特定的技术方面，可以使用它们：

NAME	描述
<code>spring-boot-starter-jetty</code>	使用 Jetty 作为嵌入式 serv 器的启动器。替代方案 spring-boot-starter-tomcat
<code>spring-boot-starter-log4j2</code>	使用 Log4j2 进行日志记录器。替代方案 spring-boot-starter-logging
<code>spring-boot-starter-logging</code>	使用 Logback 进行日志记录器。默认日志记录启动器
<code>spring-boot-starter-reactor-netty</code>	使用 Reactor Netty 作为嵌 入式 HTTP 服务器的启动器
<code>spring-boot-starter-tomcat</code>	使用 Tomcat 作为嵌入式 se 容器的启动器。使用的默认 servlet 容器启动器 spring-starter-web
<code>spring-boot-starter-undertow</code>	使用 Undertow 作为嵌入式

NAME	描述
	Tomcat 容器的启动器。替代 <code>spring-boot-starter-tomcat</code>

要了解如何交换技术方面，请参阅有关[交换 Web 服务器](#)和[日志系统的操作](#)说明文档。

07、SpringBoot项目最佳实践

Spring Boot 不需要任何特定的代码布局即可工作。但是，有一些最佳实践会有所帮助。

官方：<https://docs.spring.io/spring-boot/docs/2.5.8-SNAPSHOT/reference/htmlsingle/#using.structuring-your-code.locating-the-main-class>

7.1.1. 使用“默认”包

当一个类不包含 `package` 声明时，它被认为是在“默认包”中。通常不建议不适用“默认包”。这可能会导致使用了Spring启动应用程序的特殊问题 `@ComponentScan`，`@ConfigurationPropertiesScan`，`@EntityScan`，或 `@SpringBootApplication` 注解，因为从每一个 `package` 每一个类被读取

- 1 我们建议您遵循 [Java](#) 推荐的包命名约定的方式：（例如，`com.example.project`）。

我们通常建议您将主应用程序类放在其他类之上的根包中。该 `@SpringBootApplication` 注解往往放在你的主类，它隐含地定义为某些项目一基地“搜索包”。例如，如果您正在编写 JPA 应用程序，`@SpringBootApplication` 则使用带注释的类的包来搜索 `@Entity` 项目。使用根包还允许组件扫描仅应用于您的项目。

以下清单显示了一个典型的布局：

```
1  com
2    +- 示例
3      +- myapplication
4          +- MyApplication.java
5          |
6          +- 客户 customer
7              +- Customer.java
8              +- CustomerController.java
9              +- CustomerService.java
10             +- CustomerRepository.java
11             |
12             +- 订单 order
13                 +- Order.java
14                 +- OrderController.java
15                 +- OrderService.java
16                 +- OrderRepository.java
```

该 `MyApplication.java` 文件将声明该 `main` 方法以及基本的 `@SpringBootApplication`，如下所示：

```

1 @SpringBootApplication
2 public class MyApplication {
3
4     public static void main(String[] args) {
5         SpringApplication.run(MyApplication.class,
6             args);
7     }
8 }

```

08、SpringBoot定制Banner

定制官网如下: <http://www.patorjk.com/software/taag/#p=display&f=Graffiti&t=Type%20Something%20>

今天看到springboot可以自定义启动时的banner, 然后自己试了一下, 这里弄的是一个佛祖头像, 步骤很简单, 首先在resources目录下, 新建一个banner.txt文件。然后把下面的内容给复制进去:

```

1
2 //                      _ooOoo_
3 //                      //
4 //                      o8888888o
5 //                      //
6 //                      88"  . "88
7 //                      //

```

```

5 //                                     (| ^_^ |)
                                     //
6 //                                     0\  =  /0
                                     //
7 //                                     ____/'---'\____
                                     //
8 //                                     .'  \\\|      |//  `'.
                                     //
9 //                                     /  \\\|||  :  |||//  \
                                     //
10 //                                    /  _|||||  -:-  ||||| -  \
                                     //
11 //                                    |  |  \\\  -  ///  |  |
                                     //
12 //                                    |  \_|  '"\---/'  |  |
                                     //
13 //                                    \  .-\__  `-'  ____/- .  /
                                     //
14 //                                    ____` .  .'  /-- .--\  ` .  .  ____
                                     //
15 //                                    .""  '<  ` .__\_<|>_/_ . .'  >'"" .
                                     //
16 //                                    |  |  :  ` -  \`. ;  \ _  /`; .`/  -  ` :  |  |
                                     //
17 //                                    \  \  `-.  \_  _\  /_  _/  .-` /  /
                                     //
18 //      =====` - .__ _ ` - .__ \____/_ _ .-` ____ .-
      '=====      //
19 //                                     `===== '
                                     //

```

```

20 //  
   ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^  
   ^          //  
  
21 //           佛祖保佑                永无BUG              永  
    不修改                        //  
  
22  
23
```

可以通过配置进行关闭：

```
1 spring.main.banner-mode = off | console
```

- off 是关闭
- console 是打开

🍷 09、SpringBoot 关闭某个特定的配置

使用@EnableAutoConfiguration注解可以让SpringBoot根据当前应用项目所依赖的jar自动配置项目的相关配置，如下：

Maven: org.springframework.boot:spring-boot-autoconfigure:2.5.4

spring-boot-autoconfigure-2.5.4.jar library root

META-INF

org

springframework

boot

autoconfigure

admin
amqp
aop
availability
batch
cache
cassandra
codec
condition
context
couchbase
dao
data
diagnostics
domain
elasticsearch
flyway
freemarker
groovy
gson
h2
hateoas
hazelcast
http
influx
info
integration
jackson
jdbc
jersey
jms
jmx
jooq
jsonb
kafka
ldap
liquibase
logging
mail
mongo
mustache

如果开发者不需要SpringBoot的某一项，该如何实现呢？可以在@SpringBootApplication注解上进行关闭特定的自动配置，比如关闭：数据源

用途

如果未来你看到springboot中自己提供starter 提供几十个上百个配置类，如果有一些看着不爽，我想自己去写，怎么办呢？

- 你可以覆盖它

- 你可以先排除它，然后自己写。

举例子

面试题：如果在springBoot项目中，我们如何把官方提供好的配置类，排除掉？比如：

```
1 <dependency>
2     <groupId>com.baomidou</groupId>
3     <artifactId>mybatis-plus-boot-
  starter</artifactId>
4     <version>3.4.2</version>
5 </dependency>
```

上面的依赖是mybatis依赖，默认情况myabtis的依赖，是一定要初始化一个数据源

```
1 <bean id="dataSource" class="druid|c3p0"></bean>
2 <bean id="sqlSessionFactory"
  class="SqlSessionFactoryBean">
3     <property name="dataSource"
  ref="dataSource"/>
4 </bean>
```

这也就为什么，如果项目直接依赖了mybatis, 直接启动就报错原因：

```
1
2 Description:
3
```

```
4 Failed to configure a DataSource: 'url' attribute
  is not specified and no embedded datasource could
  be configured.
5
6 Reason: Failed to determine a suitable driver
  class
7
8
9 Action:
10
11 Consider the following:
12     If you want an embedded database (H2, HSQL or
  Derby), please put it on the classpath.
13     If you have database settings to be loaded
  from a particular profile you may need to
  activate it (no profiles are currently active).
14
```

解决方案：排除数据源依赖：

```
1 @SpringBootApplication(exclude =
  {DataSourceAutoConfiguration.class})
```

提示：但是关闭数据源不推荐，既然你在开发中引入了 *mybatis* 为什么又不配置数据源呢？除非你自己去定义数据源。你可以这样做，否则不建议

总结

- 在开发，如果你依赖一个**starter**，你肯定是要去使用配置信息。如果你不使用就不要去依赖。
- `@SpringBootApplication(exclude = {DataSourceAutoConfiguration.class})` 去排除掉，但是不建议。
- 如果你一旦用了 `@SpringBootApplication(exclude = {DataSourceAutoConfiguration.class})` 就说明你肯定要覆盖，或者你想要重写它。你就选择剔除。
- **exclude** 真正作用就是：看着不官方提供不爽，我想自己造轮子。

09、读取配置文件（重点）

SpringBoot分别提供3中方式读取项目的`application.properties`配置文件的内容。这个方式分别为：`Environment`类、`@Value`注解以及`@ConfigurationProperties`注解。

你必须要知道的事情：下面提供的三种方式，都可以拿到配置文件的信息，不要纠结那种方式好与坏。你爱用中方式就用那种方式。只要能解决问题就可以了。

09-01、Environment

Environment是用来读取应用程序运行时的环境变量的类，可以通过key-value的方式读取application.properties和系统环境变量，命令行输入参数，系统属性等，具体如下：

在application.yml文件定义如下：

```
1 # 属性配置类的
2 server:
3     port: 8082
4
5 spring:
6     main:
7         banner-mode: console
8
9 # 自定义
10 alipay:
11     pay:
12         appid: 123456
13         notify: http://www.xxx.com
14
```

定义读取的类如下：

```
1 package com.kuangstudy.web.properties;
2
3 import
    org.springframework.beans.factory.annotation.Auto
    wired;
4 import org.springframework.core.env.Environment;
```

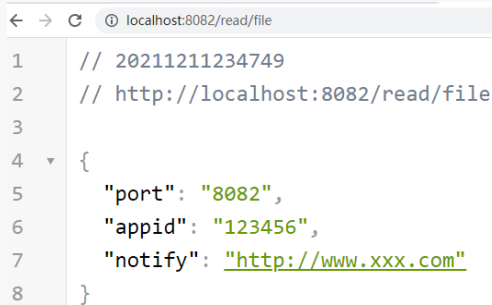
```
5 import
  org.springframework.web.bind.annotation.GetMapping;
6 import
  org.springframework.web.bind.annotation.RestController;
7
8 import java.util.HashMap;
9 import java.util.Map;
10 import java.util.Properties;
11 import java.util.Set;
12
13 /**
14  * Description:
15  * Author: yykk Administrator
16  * Version: 1.0
17  * Create Date Time: 2021/12/11 21:25.
18  * Update Date Time:
19  *
20  * @see
21  */
22 @RestController
23 public class ReadPropertiesEnvironment {
24
25     @Autowired
26     private Environment environment;
27
28
29     @GetMapping("/read/file")
30     public Map<String, Object> readInfo() {
31         Map<String, Object> map = new HashMap<>();
```

```
32     map.put("port",environment.getProperty("server.p
33     ort"));
34
35     map.put("appid",environment.getProperty("alipay.
36     pay.appid"));
37
38     map.put("notify",environment.getProperty("alipay
39     .pay.notify"));
40
41     map.put("javaversion",environment.getProperty("j
42     ava.version"));
43
44     map.put("javahome",environment.getProperty("JAVA
45     _HOME"));
46
47     map.put("mavenhome",environment.getProperty("MAV
48     EN_HOME"));
49
50     return map;
51 }
52
53 public static void main(String[] args) {
54     Properties properties =
55     System.getProperties();
56     Set<String> strings =
57     properties.stringPropertyNames();
58     for (String string : strings) {
59
60         System.out.println(string+"==>"+properties.get(
61         string));
62     }
63 }
```

```
47
48     }
49
50 }
51
```

在浏览器访问

<http://localhost:8082/read/file>



The screenshot shows a web browser window with the address bar displaying 'localhost:8082/read/file'. The main content area shows a JSON object with the following fields: 'port' (8082), 'appid' (123456), and 'notify' (http://www.xxx.com). The JSON is formatted with syntax highlighting.

```
1 // 20211211234749
2 // http://localhost:8082/read/file
3
4 {
5   "port": "8082",
6   "appid": "123456",
7   "notify": "http://www.xxx.com"
8 }
```

09-02、@Value

使用@Value注解读取配置文件内容，具体如下：

```
1 package com.kuangstudy.web.properties;
2
3 import
  org.springframework.beans.factory.annotation.Auto
  wired;
```

```
4  import
    org.springframework.beans.factory.annotation.Value;
5  import org.springframework.core.env.Environment;
6  import
    org.springframework.web.bind.annotation.GetMapping;
7  import
    org.springframework.web.bind.annotation.RestController;
8
9  import java.util.HashMap;
10 import java.util.Map;
11 import java.util.Properties;
12 import java.util.Set;
13
14 /**
15  * Description:
16  * Author: yykk Administrator
17  * Version: 1.0
18  * Create Date Time: 2021/12/11 21:25.
19  * Update Date Time:
20  *
21  * @see
22  */
23 @RestController
24 public class ReadPropertiesValue {
25
26     @Value("${server.port}")
27     private Integer port;
28     @Value("${alipay.pay.appid}")
```

```
29     private String appId;
30     @Value("${alipay.pay.notify}")
31     private String notify;
32     @Value("${java.version}")
33     private String javaVersion;
34     @Value("${JAVA_HOME}")
35     private String javaHome;
36     @Value("${MAVEN_HOME}")
37     private String mavenHome;
38
39
40     @GetMapping("/read/value")
41     public Map<String, Object> readInfo() {
42         Map<String, Object> map = new HashMap<>
43         ();
44         map.put("port", port);
45         map.put("appid", appId);
46         map.put("notify", notify);
47         map.put("javaversion", javaVersion);
48         map.put("javahome", javaHome);
49         map.put("mavenhome", mavenHome);
50         return map;
51     }
52 }
```

浏览器如下：

```
// 20211211235916
// http://localhost:8083/read/value

{
  "port": 8083,
  "appid": "123456",
  "mavenhome": "G:\\tools\\apache-maven-3.6.3",
  "javahome": "C:\\Program Files\\Java\\jdk1.8.0_291",
  "notify": "http://www.xxx.com",
  "javaversion": "1.8.0_291"
}
```

结论：其实@Value底层就是Environment.java

09-03、@ConfigurationProperties（属性配置类的底层原理）

使用@ConfigurationProperties首先建立配置文件与对象的映射关系，然后在控制器方法中使用@Autowired注解将对象注入。具体如下：

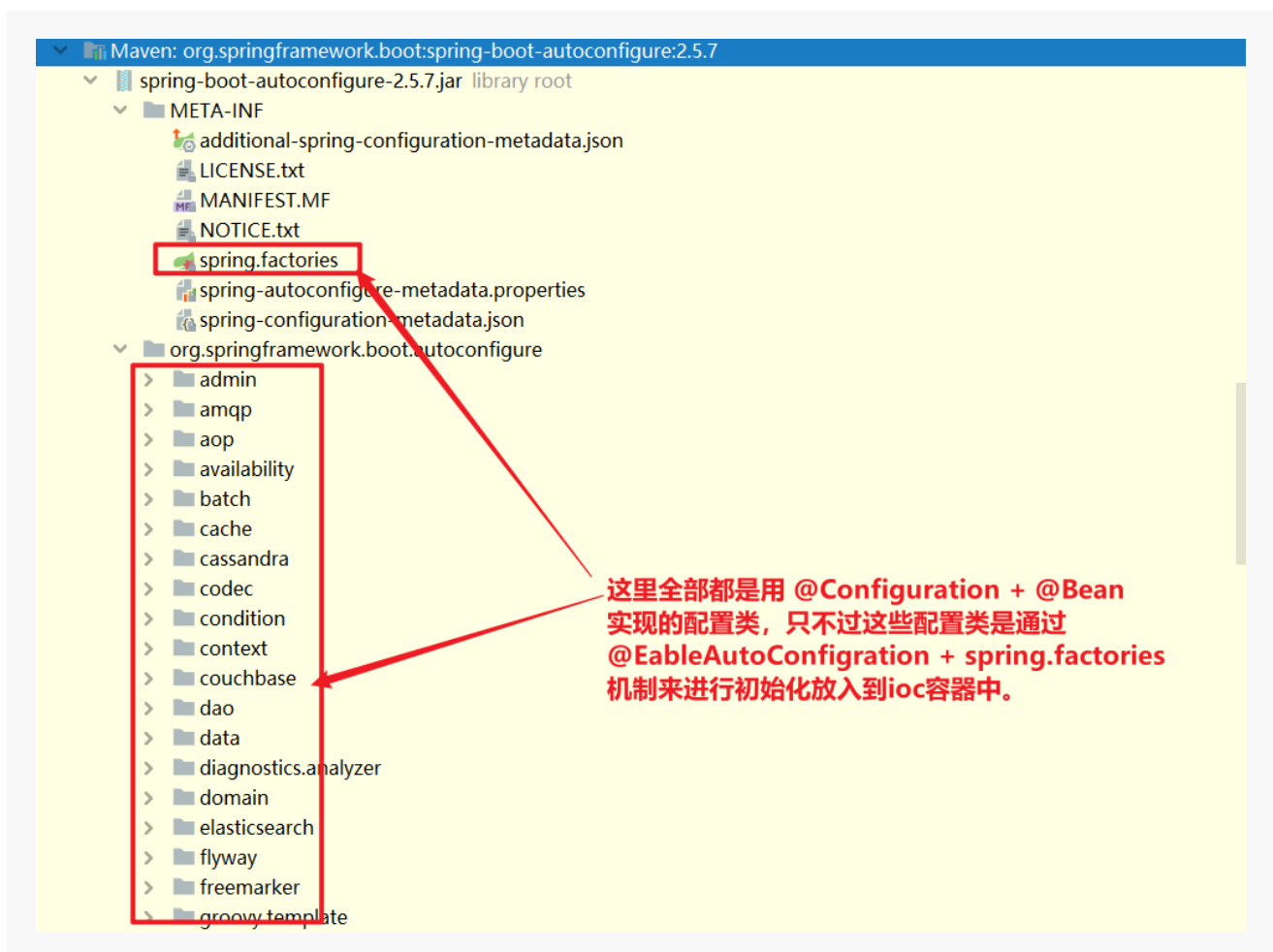
我们指定：@SpringBootApplication 注解是一个复合注解：其中有两个注解是和配置类和属性配置类有关：如下：

1、@SpringBootConfiguration：对@Configuration注解的升级版。一般用来扩展和覆盖内部的配置的。自定义starter的时候使用的一种机制（springboot提供让开发人员扩展使用的一直机制，因为官方提供配置类不可能满足未来所有业务场景和需求，或者一些新的技术的诞生，springboot团队是不可能马上就开发所谓starter，这个时候就只能去扩展或者自定义，未来所讲的starter其实就通过

配置类和属性配置完成这个过程其中使用的原理就是：

@SpringBootConfiguration+@Bean 或者 @Configuration + @Bean)

2、@EnableAutoConfiguration：去加载springBoot官方提供的starter包提供的配置类和属性配置类。官方提供了一系列的starters的依赖，这些依赖分别在：spring-boot-autoconfigure-2.5.7.jar



这个思想：java9中新特性：SPI，把项目中一些类的初始化放在一个文件中，然后去加载和读取这个文件，将其实例化的过程。

每个配置类一般都搭配一个属性配置类：

- DataSourceConfiguration + DataSourceProperties
- JacksonAutoConfiguration + JacksonProperties
- KafkaAutoConfiguration + KafkaProperties

属性配置类的作用就是让你去扩展和动态修改属性值的一个机制，如果没有这个机制其实starter其实毫无意义。比如：我的：
kafka: 158.12.41.41:9092 内部：localhost:9092 .

同时告诉我们一个道理：每天在全局配置文件中配置其实都是在给starter属性配置类在进行赋值。也是就说没个配置都会对应一个属性配置类。每个值都对应属性配置类的属性。比如：

server.port=8080 对应的配置类：ServerProperties port就是属性：
port

```
1 ServerProperties serverp = new ServerProperties();  
2 serverp.setPort(8080)
```

一句话：配置文件在每天配置，其实背后都有一个类，定义值其实都是给类的属性赋值。你就想象成调用set方法。只不过这个过程创建对象的过程，和赋值你过程看不到而已，但是绝对是java基础，创建对象给属性赋值。

09-04、你为什么要用属性配置

传统的代码编写就通过耦合的方式编写如下：

```

1
2     @GetMapping("/alipay")
3     public String alipay(){
4         String appid = "2021003100625328";
5         String mkey =
        "MIIEvwIBADANBgkqhkiG9w0BAQEFAASCBAKkwggS1AgEAAoIB
        AQC9kGK4VMbYm";
6         String ckey =
        "MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAx5i5
        LhEtDZW6Q6mUxkC5f6sAvZm90ncAkRXwfoBdDeKk";
7         String callbackurl =
        "https://www.txnh.net/api/alipay/returnUrl";
8         String charset = "utf-8";
9         log.info("你支付是的appid:{}, {}, {},
        {} ", appid, ckey, mkey, callbackurl, charset);
10        return "success";
11    }
12

```

- 上面的代码没有错误的，是正确的
- 上面是耦合的，如果我换了公司，换企业就必须重新在代码进行修改，就不方便进行调整。

09-05、使用@Value的方式进行定义

第一步：在application.yml文件中自定义属性：

```
1 # 自定义属性
2 ksd:
3     alipay:
4         appid: 2021003100625328
5         mkey:
6             MIIEVwIBADANBgkqhkiG9w0BAQEFAASCBBKkwggS1AgEAAoIBAQC9kGK4VMbYm
7             ckey:
8                 MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAX5i5Lh
9                 EtDZw6Q6mUxkC5f6sAvZm9OncAkRXwfoBdDeKk
10            callbackurl:
11                https://www.txnh.net/api/alipay/returnUrl
12            charset: utf-8
```

第二步：在代码中使用@Value的方式进行获取全局配置文件中对应的值如下：

```
1 package com.kuangstudy.web.properties;
2
3 import lombok.extern.slf4j.Slf4j;
4 import
5     org.springframework.beans.factory.annotation.Auto
6     wired;
7 import
8     org.springframework.beans.factory.annotation.Valu
9     e;
10 import
11     org.springframework.web.bind.annotation.GetMappin
12     g;
```

```
7 import
  org.springframework.web.bind.annotation.RestController;

8

9 import javax.crypto.MacSpi;

10

11 /**
12  * Description:
13  * Author: yykk Administrator
14  * Version: 1.0
15  * Create Date Time: 2021/12/14 20:36.
16  * Update Date Time:
17  *
18  * @see
19  */
20 @RestController
21 @Slf4j
22 public class AlipayController {
23
24     @Value("${ksd.alipay.appid}")
25     private String appid;
26     @Value("${ksd.alipay.mkey}")
27     private String mkey;
28     @Value("${ksd.alipay.ckey}")
29     private String ckey;
30     @Value("${ksd.alipay.callback}")
31     private String callbackurl;
32     @Value("${ksd.alipay.charset}")
33     private String charset;
34
35 }
```

```

36     @GetMapping("/alipay")
37     public String alipay(){
38         log.info("你支付是的appid:{}, {}, {}, {}, {}", appid, ckey, mkey, callbackurl, charset);
39         return "success";
40     }
41 }
42

```

第三步：在浏览器访问如下

<http://localhost:8083/alipay>

The screenshot shows an IDE with a Java file containing the following code:

```

@Value("${ksd.alipay.appid}")
private String appid; appid: "2021003100625328"
@Value("${ksd.alipay.mkey}")
private String mkey; mkey: "MIIEvwIBADANBgkqhkiG9w0BAQEFAASCBBKkwwgSIAgEAAoIBAQC9kGK4VMbYm"
@Value("${ksd.alipay.ckey}")
private String ckey; ckey: "MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCGKCAQEAX5i5LhEtDZw6Q6mUxkC5f6sAvZm90ncAkI
@Value("${ksd.alipay.callback}")
private String callbackurl; callbackurl: "https://www.txnh.net/api/alipay/returnUrl"
@Value("${ksd.alipay.charset}")
private String charset; charset: "utf-8"

@GetMapping("/alipay")
public String alipay(){
    log.info("你支付是的appid:{}, {}, {}, {}, {}", appid, ckey, mkey, callbackurl, charset); appid: "2021003100625328"
    return "success";
}

```

Below the code, a red box highlights the runtime state of the controller in the debugger's Variables window:

```

this = (AlipayController@7257)
  mkey = "MIIEvwIBADANBgkqhkiG9w0BAQEFAASCBBKkwwgSIAgEAAoIBAQC9kGK4VMbYm"
  charset = "utf-8"
  ckey = "MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCGKCAQEAX5i5LhEtDZw6Q6mUxkC5f6sAvZm90ncAkRXwfoBdDeK"
  appid = "2021003100625328"
  callbackurl = "https://www.txnh.net/api/alipay/returnUrl"

```

Text annotation: 通过@Value可以很清楚的看到对应全局配置文件中对应的属性都可以获取到

- 通过上面可以很清楚看到@Value可以读取全局配置文件对应的属性的值。
- 好处：封装和统一管理，解决耦合性。
- 缺点1：不具备面向对象特征，如果属性过多，而且有一类的很难区分，

- 缺点2：如果@Value(key)使用的key如果在配置文件中没有定义，直接报错

09-06、springboot官方提供的starter配置类为什么不用@Value

- 属性太多，而且用@Value不具备面向对象的特征和行为，所以就用@ConfigurationProperties去取代@Value的方式。
- @Value有毛病，就是无法在全部配置文件中自动提示

09-07、springBoot取而代之的是@Configuration + @ConfigurationProperties

- 具有面向对象的特征
 - 可以在代码中自动提示
 - 维护起来很方便。在配置文件也可以很方便的控制和提示
- 具体实现步骤：

第一步：在application.yml自定义支付属性

```
1 # 自定义属性
2 ksd:
3     alipay:
4         appid: 2021003100625328
5         mkey:
6             MIIEVwIBADANBgkqhkiG9w0BAQEFAASCBBKkwggS1AgEAAoIBAQC9kGK4VMbYm
7             ckey:
8                 MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAX5i5Lh
9                 EtDZw6Q6mUxkC5f6sAvZm9OncAkRXwfoBdDeKk
10            callback:
11                https://www.txnh.net/api/alipay/returnUrl
12            charset: utf-8
```

第二步：定义一个属性配置类如下：

```
1 package com.kuangstudy.web.properties;
2
3 import lombok.Data;
4 import
5     org.springframework.beans.factory.annotation.Value;
6
7 import
8     org.springframework.boot.context.properties.ConfigurationProperties;
9
10 /**
11  * Description:
12  * Author: yykk Administrator
13  * Version: 1.0
14  * Create Date Time: 2021/12/14 20:53.
```



```

12  * Update Date Time:
13  *
14  * @see
15  */
16  @ConfigurationProperties(prefix = "ksd.alipay")//
    这个注解是用找到类
17  @Data
18  public class AlipayProperties {
19      private String appid;
20      private String mkey;
21      private String ckey;
22      private String callback;
23      private String charset;
24  }
25

```

- @ConfigurationProperties(prefix = "ksd.alipay")//这个注解的作用是：找到类
- prefix = "ksd.alipay" 前缀未来可以在全局配置文件中自动进行提示。

第三步：属性配置类必须找到一个配置类进行注册如下：

```

1  package com.kuangstudy.web.properties;
2
3  import lombok.Data;
4  import
    org.springframework.boot.SpringBootConfiguration;

```

```
5 import
  org.springframework.boot.context.properties.Config
  urationProperties;
6 import
  org.springframework.boot.context.properties.Enabl
  eConfigurationProperties;
7 import
  org.springframework.context.annotation.Configurat
  ion;
8
9 /**
10  * Description:
11  * Author: yykk Administrator
12  * Version: 1.0
13  * Create Date Time: 2021/12/14 20:53.
14  * Update Date Time:
15  *
16  * @see
17  */
18 // 用配置类去注册：属性配置类
19 @EnableConfigurationProperties(AlipayProperties.c
  lass)
20 @SpringBootConfiguration
21 public class AlipayConfiguration {
22
23 }
24
```

建议：一个属性配置类 -- 找到自己对应配置类进行注册，
@EnableConfigurationProperties(AlipayProperties.class)

问：可以在启动类注册吗？可以，因为启动类本身也是一个配置类，但是不推荐。考虑后续的维护以及可插拔性。

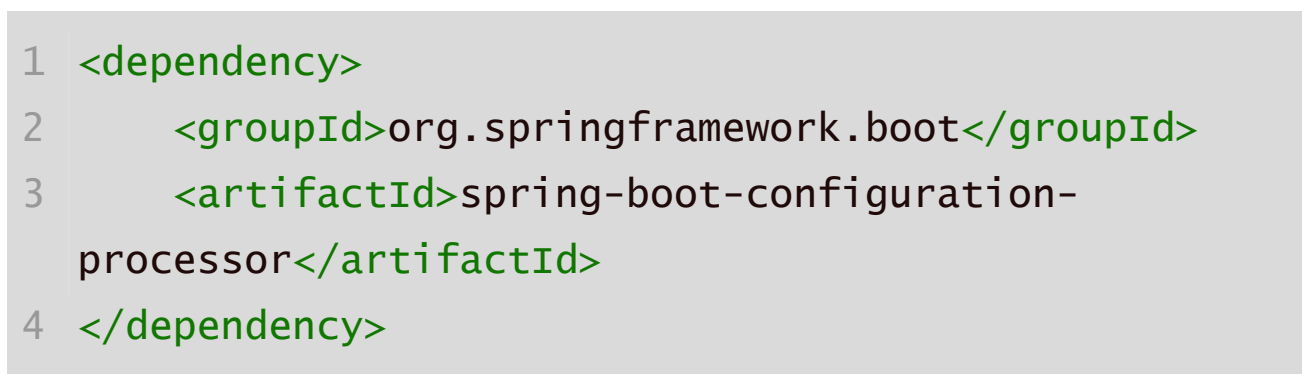
09-08、总结

- 在开发中不要纠结到底使用那种好那种坏，你可以解决业务问题就可以了
- 如果属性仅仅就1~3个其实就用@Value就可以。但是如果很多还是建议大家使用@ConfigurationProperties + @Configuration机制会更好，可以更加清晰底层原来在做什么事情。
- 这也是为什么底层不用@Value的原因。 @Configuration + @Value 这种理论是可以，但是有毛病不能在全局配置文件中自动提示。

09-09、关于属性配置类的自动提示的问题

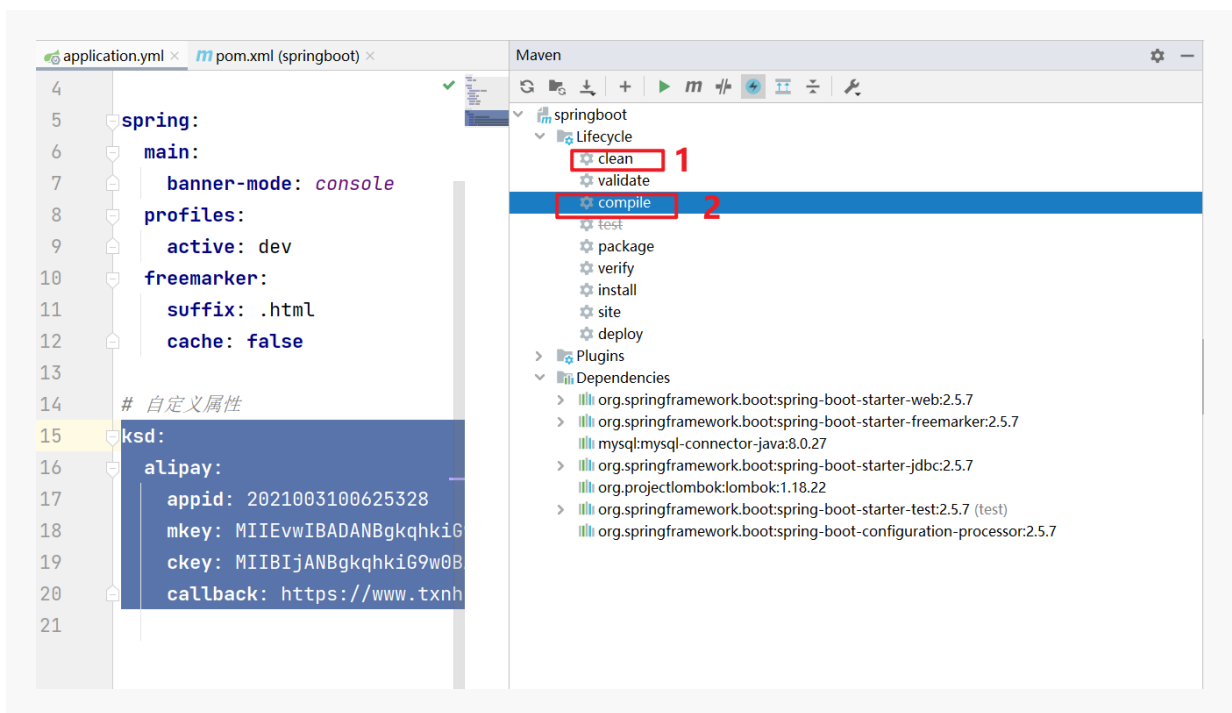


第一步：pom.xml中增加配置



第二步：关闭所有

- 把idea打开的所有的类和全局配置文件全部关闭。（很重要）
- 然后打开maven的控制，执行clean和compile即可



- 重新开applicaiton.yml文件进行查看是否自动提示，如果可以就说明生效了：



👉 09-10、@PropertySource（不推荐）

开发者希望读取项目的其他配置文件，而不是全局配置文件中的 application.properties，该如何实现呢？可以使用 @PropertySource 注解找到项目的其他的配置文件。

方式：@PropertySource + @Value

具体如下：

定义两个属性配置文件a.yml和b.yml

a.yml

```
1 user.nickname: zhangsan
```

b.yml

```
1 user.age: 32
```

第二步：定义类

```
1 package com.kuangstudy.web.properties;
2
3 import
  com.kuangstudy.web.properties.config.AlipayProperties;
4 import lombok.extern.slf4j.Slf4j;
5 import
  org.springframework.beans.factory.annotation.Autowired;
6 import
  org.springframework.beans.factory.annotation.Value;
7 import
  org.springframework.context.annotation.PropertySource;
```

```
8  import
   org.springframework.web.bind.annotation.GetMapping;
9  import
   org.springframework.web.bind.annotation.RestController;
10
11  /**
12   * Description:
13   * Author: yykk Administrator
14   * Version: 1.0
15   * Create Date Time: 2021/12/14 20:36.
16   * Update Date Time:
17   *
18   * @see
19   */
20  @RestController
21  @Slf4j
22  @PropertySource({"classpath:a.yml","classpath:b.y
   ml"})
23  public class AlipayController3 {
24
25      @Value("${user.nickname}")
26      private String userName;
27      @Value("${user.age}")
28      private String usage;
29
30      @GetMapping("/alipay3")
31      public String alipay3(){
32          log.info("你支付是的:{}",
   {},",userName,usage);
```

```
33         return "success";
34     }
35 }
36
```

第三步：访问测试

<http://localhost:8083/alipay3>

```
@RestController
@Slf4j
@PropertySource({"classpath:a.yml", "classpath:b.yml"})
public class AlipayController3 {

    @Value("${user.nickname}")
    private String userName;    userName: "zhangsan"
    @Value("${user.age}")
    private String userage;    userage: "32"

    @GetMapping("/alipay3")
    public String alipay3(){
        log.info("你支付的是:{},{}", userName, userage);    userName: "zhangsan"    user
        return "success";
    }
}
```

总结：

- 这种@PropertySource机制是一种扩展属性配置文件的机制。
- 一般不建议大家去使用，学习目的是未来能看得懂别人写的逻辑和代码。