SpringBoot

1、SpringBoot工程入门开发步骤

入门案例

- SpringBoot是由Pivotal团队提供的全新框架，其设计目的是用来简化Spring应用的初始搭建以及开发过程

- 原生开发SpringMVC程序过程

```
<dependencies>
  <dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>javax.servlet-api</artifactId>
    <version>3.1.0</version>
    <scope>provided</scope>
  </dependency>
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-webmvc</artifactId>
    <version>5.2.10.RELEASE</version>
  </dependency>
</dependencies>
```

- 原生开发SpringMVC程序过程

```
<dependencies>
  <
  <
  <
</dependencies>
```

```
public class ServletConfig extends AbstractAnnotationConfigDispatcherServletInitializer {
    protected Class<?>[] getRootConfigClasses() {
        return new Class[]{SpringConfig.class};
    }
    protected Class<?>[] getServletConfigClasses() {
        return new Class[]{SpringMvcConfig.class};
    }
    protected String[] getServletMappings() {
        return new String[]{"/"};
    }
}
```

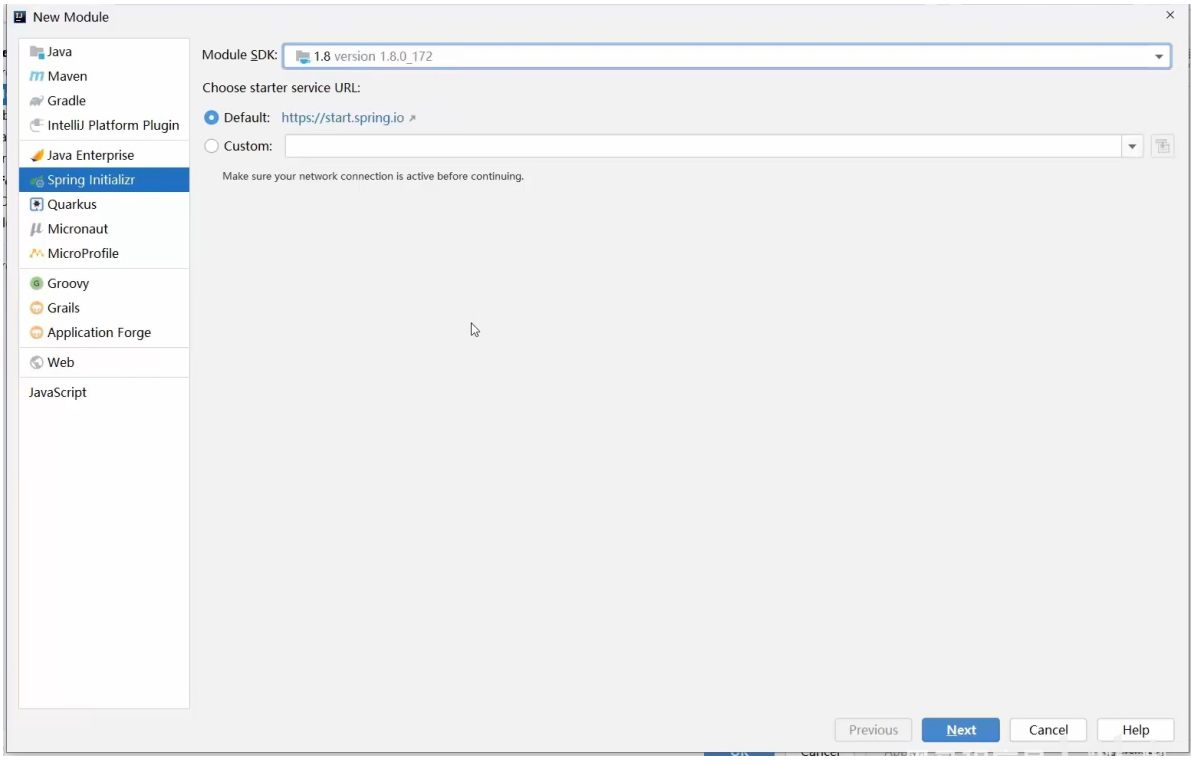
- 原生开发SpringMVC程序过程

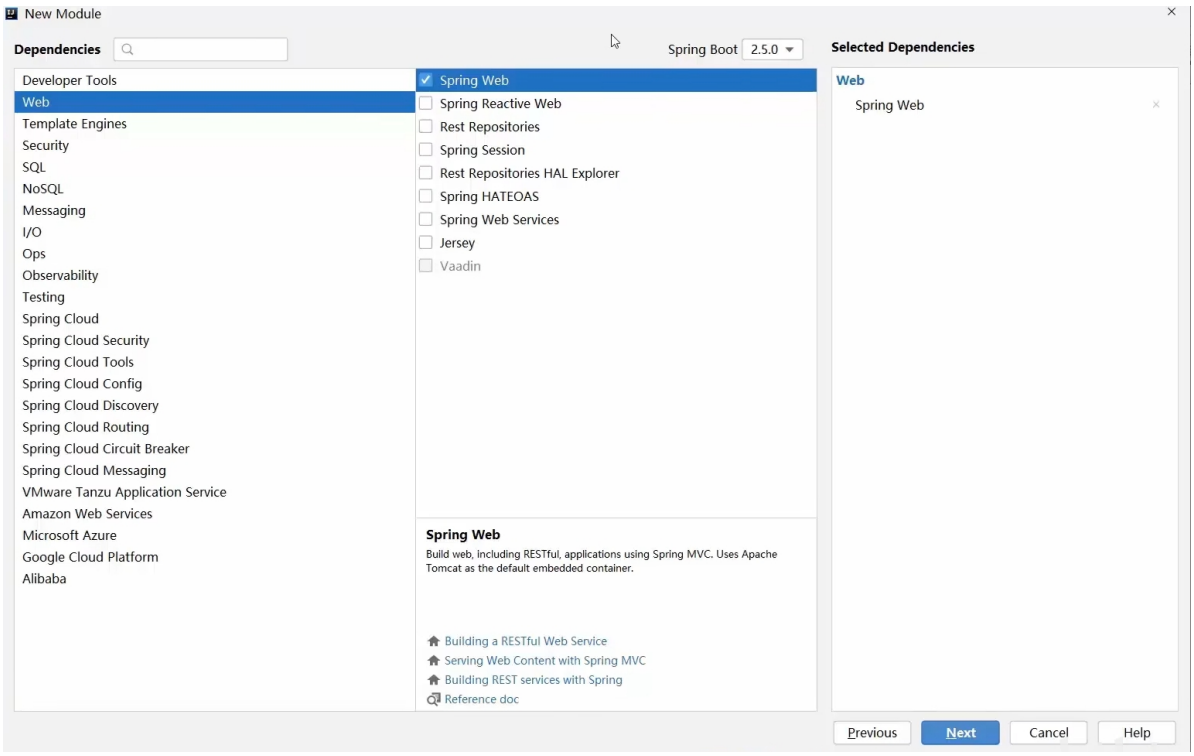
```
<dependencies>
</dependencies>
```

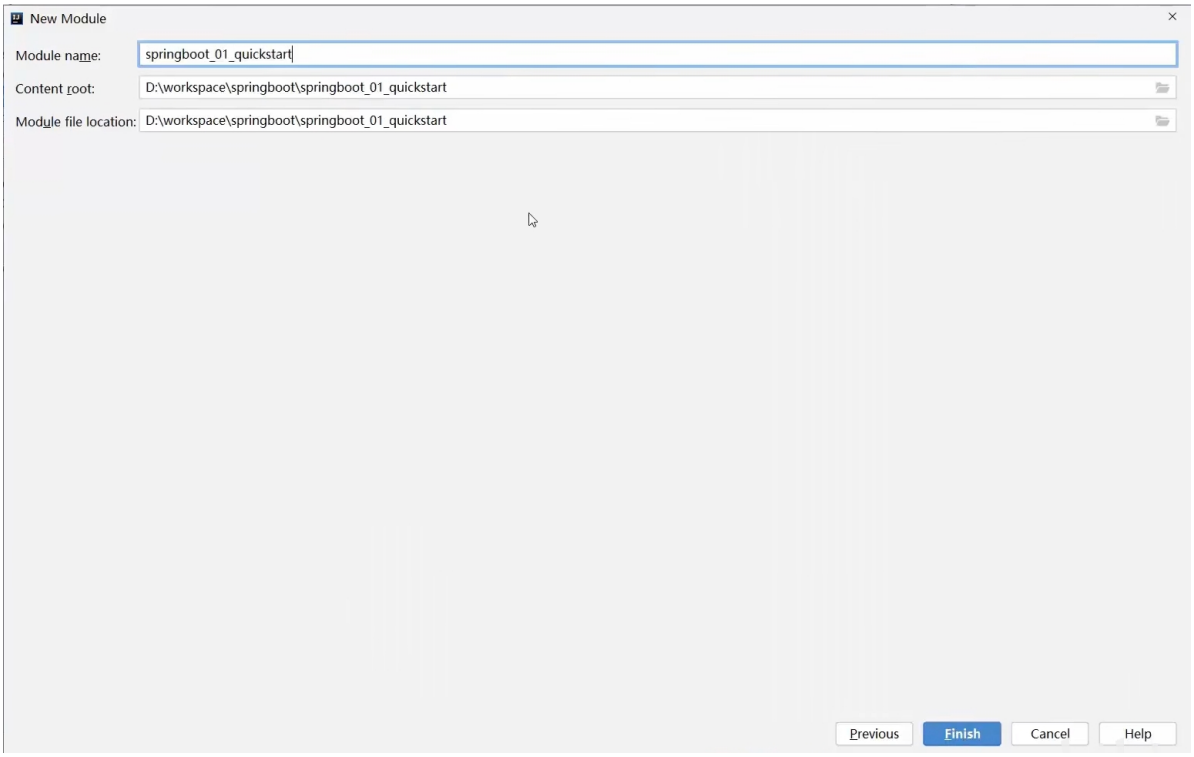
```
public class ServletConfig extends AbstractAnnotationConfigDispatcherServletInitializer {
}
```

```
@Configuration
```

```
@RestController
@RequestMapping("/books")
public class BookController {
    @Autowired
    private BookService bookService;
    @GetMapping("/{id}")
    public Result getById(@PathVariable Integer id) {
        Book book = bookService.getById(id);
        Integer code = book != null ? Code.GET_OK : Code.GET_ERR;
        String msg = book != null ? "" : "数据查询失败，请重试！";
        return new Result(code,book,msg);
    }
}
```







③：开发控制器类

```
@RestController
@RequestMapping("/books")
public class BookController {
    @GetMapping("/{id}")
    public String getById(@PathVariable Integer id){
        System.out.println("id ==> "+ id);
        return "hello , spring boot! ";
    }
}
```

④：运行自动生成的Application类

2、SpringBoot工程官网创建方式

- 最简SpringBoot程序所包含的基础文件
  - pom.xml文件

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.5.0</version>
    </parent>
    <groupId>com.itheima</groupId>
    <artifactId>springboot-01-quickstart</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
    </dependencies>
</project>
```

- 最简SpringBoot程序所包含的基础文件
  - pom.xml文件
  - Application类

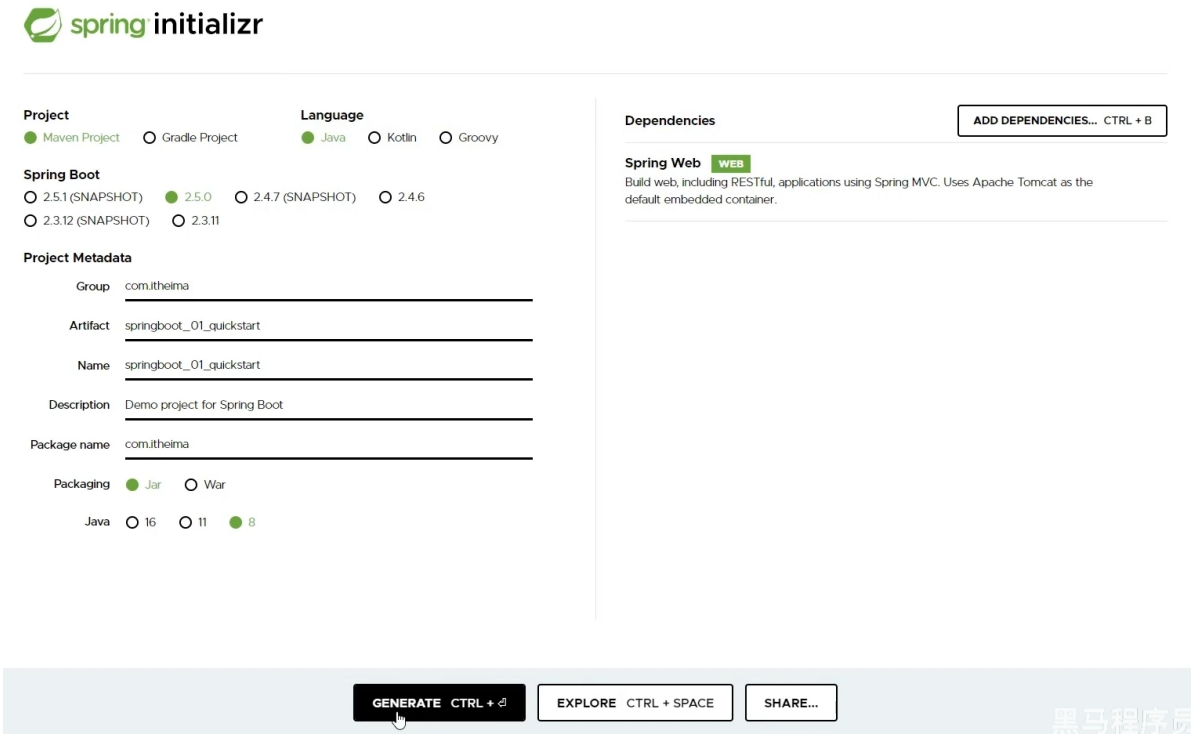
```java
@SpringBootApplication
public class Application {
    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }
}
```

- Spring程序与SpringBoot程序对比

类/配置文件	Spring	SpringBoot
pom文件中的坐标	手工添加	勾选添加
web3.0配置类	手工制作	无
Spring/SpringMVC配置类	手工制作	无
控制器	手工制作	手工制作

注意事项

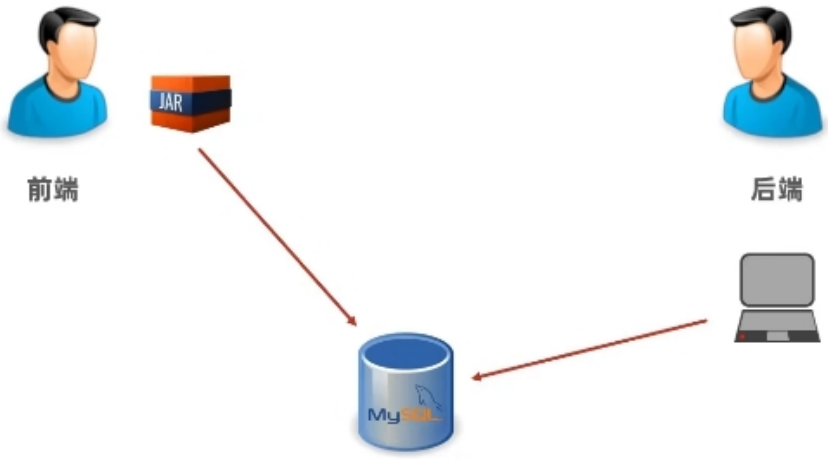
基于idea开发SpringBoot程序需要确保联网且能够加载到程序框架结构



3、SpringBoot程序快速启动

SpringBoot项目快速启动

- 前后端分离合作开发



步骤 SpringBoot项目快速启动

①：对SpringBoot项目打包（执行Maven构建指令package）

步骤 SpringBoot项目快速启动

①：对SpringBoot项目打包（执行Maven构建指令package）

步骤 SpringBoot项目快速启动

①：对SpringBoot项目打包（执行Maven构建指令package）

4、SpringBoot简介（起步依赖）

SpringBoot概述

- SpringBoot是由Pivotal团队提供的全新框架，其设计目的是用来简化Spring应用的初始搭建以及开发过程

- Spring程序缺点
  - 配置繁琐
  - 依赖设置繁琐
- SpringBoot程序优点
  - 自动配置
  - 起步依赖（简化依赖配置）
  - 辅助功能（内置服务器，……）

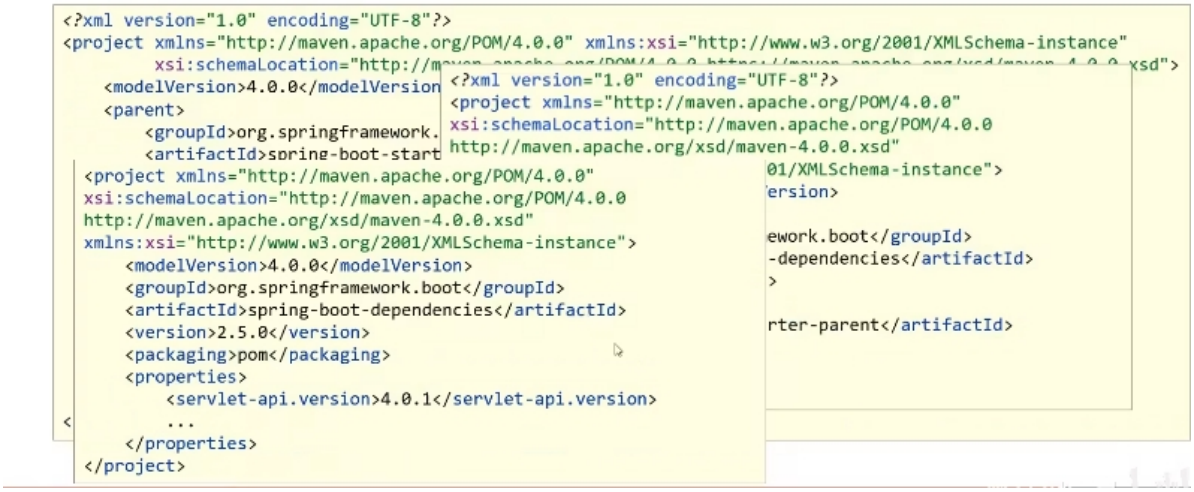
- 起步依赖

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.5.0</version>
    </parent>
    <groupId>com.itheima</groupId>
    <artifactId>springboot-01-quickstart</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
    </dependencies>
</project>
```

- 起步依赖


```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.5.0</version>
    </parent>
    <groupId>com.itheima</groupId>
    <artifactId>springboot-01-quickstart</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
    </dependencies>
</project>
```

- 起步依赖



- 起步依赖

- 起步依赖

SpringBoot起步依赖

- starter
  - SpringBoot中常见项目名称，定义了当前项目使用的所有项目坐标，以达到减少依赖配置的目的
- parent
  - 所有SpringBoot项目要继承的项目，定义了若干个坐标版本号（依赖管理，而非依赖），以达到减少依赖冲突的目的
  - spring-boot-starter-parent（2.5.0）与 spring-boot-starter-parent（2.4.6）共计57处坐标版本不同
- 实际开发
  - 使用任意坐标时，仅书写GAV中的G和A，V由SpringBoot提供
  - 如发生坐标错误，再指定version（要小心版本冲突）

SpringBoot起步依赖

- starter
  - SpringBoot中常见项目名称，定义了当前项目使用的所有项目坐标，以达到减少依赖配置的目的
- parent
  - 所有SpringBoot项目要继承的项目，定义了若干个坐标版本号（依赖管理，而非依赖），以达到减少依赖冲突的目的
  - spring-boot-starter-parent（2.5.0）与 spring-boot-starter-parent（2.4.6）共计57处坐标版本不同
- 实际开发
  - 使用任意坐标时，仅书写GAV中的G和A，V由SpringBoot提供
  - 如发生坐标错误，再指定version（要小心版本冲突）

5、Spring简介（辅助功能之切换Web服务器）

- 启动方式

```
@SpringBootApplication
public class Springboot01QuickstartApplication {
    public static void main(String[] args) {
        SpringApplication.run(Springboot01QuickstartApplication.class, args);
    }
}
```

- SpringBoot在创建项目时，采用jar的打包方式
- SpringBoot的引导类是项目的入口，运行main方法就可以启动项目

- 使用maven依赖管理变更起步依赖项

```
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
        <!--web起步依赖环境中，排除Tomcat起步依赖-->
        <exclusions>
            <exclusion>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-tomcat</artifactId>
            </exclusion>
        </exclusions>
    </dependency>
    <!--添加Jetty起步依赖，版本由SpringBoot的starter控制-->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-jetty</artifactId>
    </dependency>
</dependencies>
```

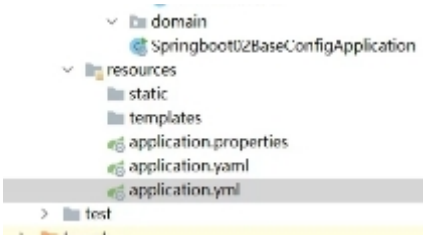
- Jetty比Tomcat更轻量级，可扩展性更强（相较于Tomcat），谷歌应用引擎（GAE）已经全面切换为Jetty

6、配置文件格式（3种）

- 修改服务器端口

```
http://localhost:8080/books/1
```

```
http://localhost/books/1
```



- SpringBoot提供了多种属性配置方式
  - application.properties

```
server.port=80
```

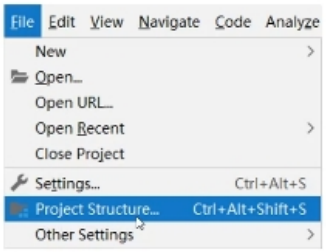
  - application.yml

```
server:
  port: 81
```

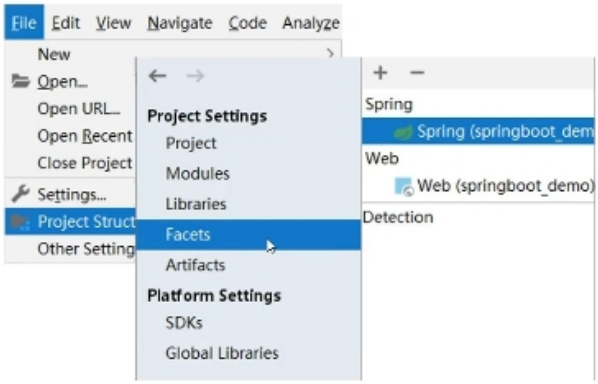
  - application.yaml

```
server:
  port: 82
```

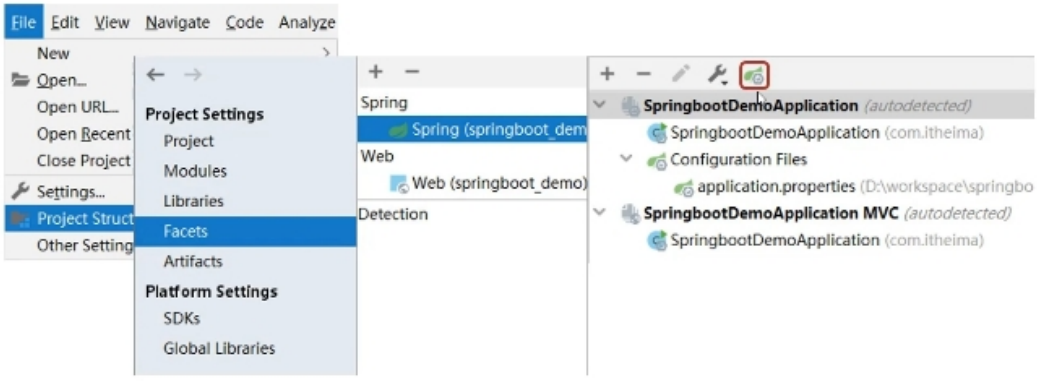
自动提示功能消失解决方案



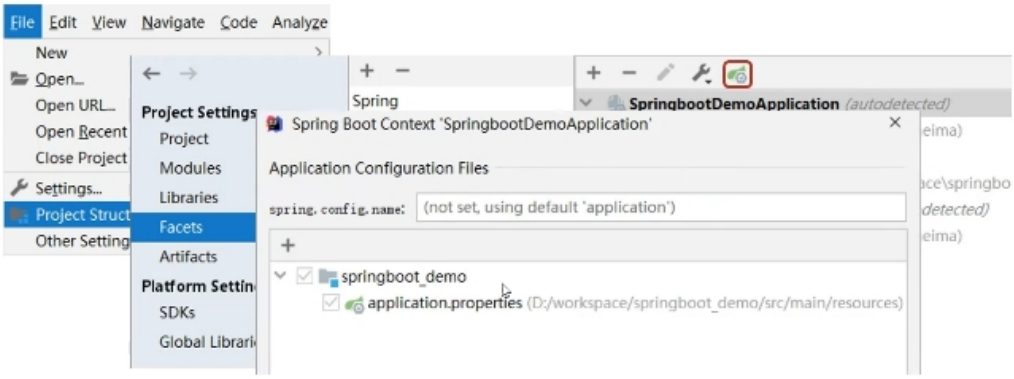
自动提示功能消失解决方案



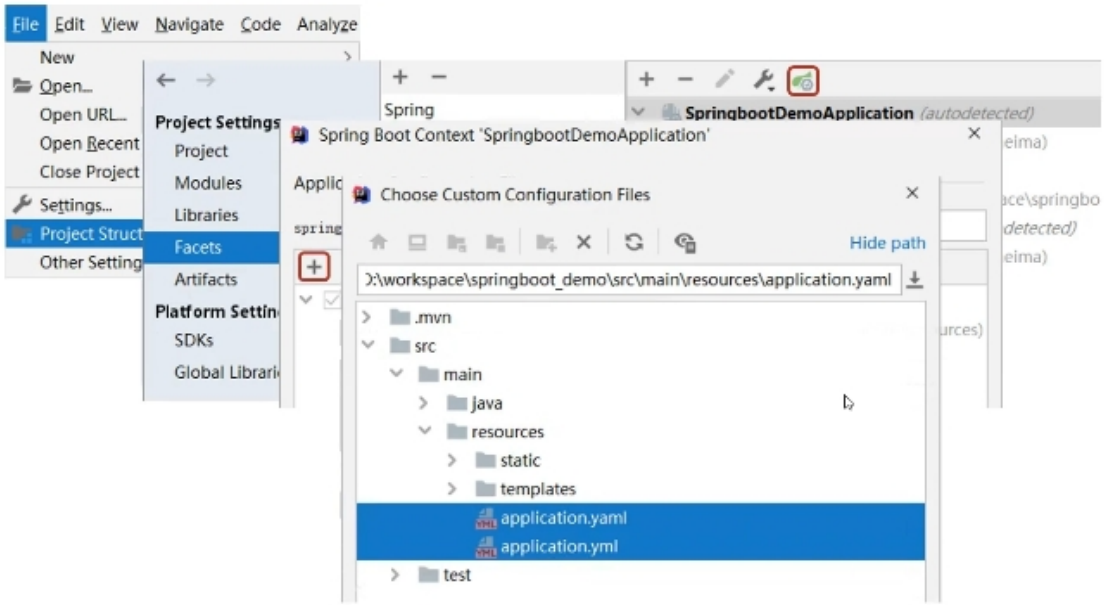
自动提示功能消失解决方案



自动提示功能消失解决方案



自动提示功能消失解决方案



配置格式

- SpringBoot配置文件加载顺序（了解）
  - application.**properties** > application.**yml** > application.**yaml**

注意事项

SpringBoot核心配置文件名为application

SpringBoot内置属性过多，且所有属性集中在一起修改，在使用时，通过提示键+关键字修改属性

7、yaml格式

yaml

- YAML（YAML Ain't Markup Language），一种数据序列化格式
- 优点：
  - 容易阅读
  - 容易与脚本语言交互
  - 以数据为核心，重数据轻格式
- YAML文件扩展名
  - **.yml（主流）**
  - .yaml

- YAML（YAML Ain't Markup Language），一种数据序列化格式
- 优点：
  - 容易阅读
  - 容易与脚本语言交互
  - 以数据为核心，重数据轻格式
- YAML文件扩展名
  - **.yml（主流）**
  - .yaml

XML

```xml
<enterprise>
    <name>itcast</name>
    <age>16</age>
    <tel>4006184000</tel>
</enterprise>
```

Properties

```properties
enterprise.name=itcast
enterprise.age=16
enterprise.tel=4006184000
```

yaml

```yaml
enterprise:
  name: itcast
  age: 16
  tel: 4006184000
```

yaml语法规则

- 大小写敏感
- 属性层级关系使用多行描述，每行结尾使用冒号结束
- 使用缩进表示层级关系，同层级左侧对齐，只允许使用空格（不允许使用Tab键）
- 属性值前面添加空格（属性名与属性值之间使用冒号+空格作为分隔）
- # 表示注释

yml

```yaml
enterprise:
  name: itcast
  age: 16
  tel: 4006184000
```

写数组的格式：

```yaml
likes:
  - music
  - game
  - PE
```

8、yaml数据读取方式（3种）

- 使用@Value读取单个数据，属性名引用方式：**${一级属性名.二级属性名……}**

```yaml
lesson: SpringBoot

server:
  port: 82

enterprise:
  name: itcast
  age: 16
  tel: 4006184000
  subject:
    - Java
    - 前端
    - 大数据
```

```java
@RestController
@RequestMapping("/books")
public class BookController {

    @Value("${lesson}")
    private String lessonName;

    @Value("${server.port}")
    private int port;

    @Value("${enterprise.subject[1]}")
    private String[] subject_01;
}
```

yaml数据读取

- 封装全部数据到Environment对象

```yaml
lesson: SpringBoot

server:
  port: 82

enterprise:
  name: itcast
  age: 16
  tel: 4006184000
  subject:
    - Java
    - 前端
    - 大数据
```

```java
@RestController
@RequestMapping("/books")
public class BookController {
    @Autowired
    private Environment env;
    @GetMapping("/{id}")
    public String getById(@PathVariable Integer id){
        System.out.println(env.getProperty("lesson"));
        System.out.println(env.getProperty("enterprise.name"));
        System.out.println(env.getProperty("enterprise.subject[0]"));
        return "hello , spring boot!";
    }
}
```

- 自定义对象封装指定数据

```yaml
lesson: SpringBoot

server:
  port: 82

enterprise:
  name: itcast
  age: 16
  tel: 4006184000
  subject:
    - Java
    - 前端
    - 大数据
```

```java
@Component
@ConfigurationProperties(prefix = "enterprise")
public class Enterprise {
    private String name;
    private Integer age;
    private String[] subject;
}
```

```java
@RestController
@RequestMapping("/books")
public class BookController {
    @Autowired
    private Enterprise enterprise;
}
```

自定义对象封装数据警告解决方案

```xml
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-configuration-processor</artifactId>
    <optional>true</optional>
</dependency>
```

warning:put it into pom file

9、多环境开发配置

多环境启动

```yaml
spring:
  profiles:
    active: pro
---
spring:
  profiles: pro

server:
  port: 80
---
spring:
  profiles: dev

server:
  port: 81
---
spring:
  profiles: test

server:
  port: 82
```

启动指定环境

设置生产环境

生产环境具体参数设定

设置开发环境

开发环境具体参数设定

设置测试环境

测试环境具体参数设定

多环境启动

```yaml
spring:
  profiles:
    active: pro
---
server:
  port: 80

spring:
  profiles: pro
```

过时格式

```yaml
spring:
  profiles:
    active: pro
---
server:
  port: 80

spring:
  config:
    activate:
      on-profile: pro
```

推荐格式

properties文件多环境启动

- 主启动配置文件application.properties

```properties
spring.profiles.active=pro
```

- 环境分类配置文件application-**pro**.properties

```properties
server.port=80
```

- 环境分类配置文件application-**dev**.properties

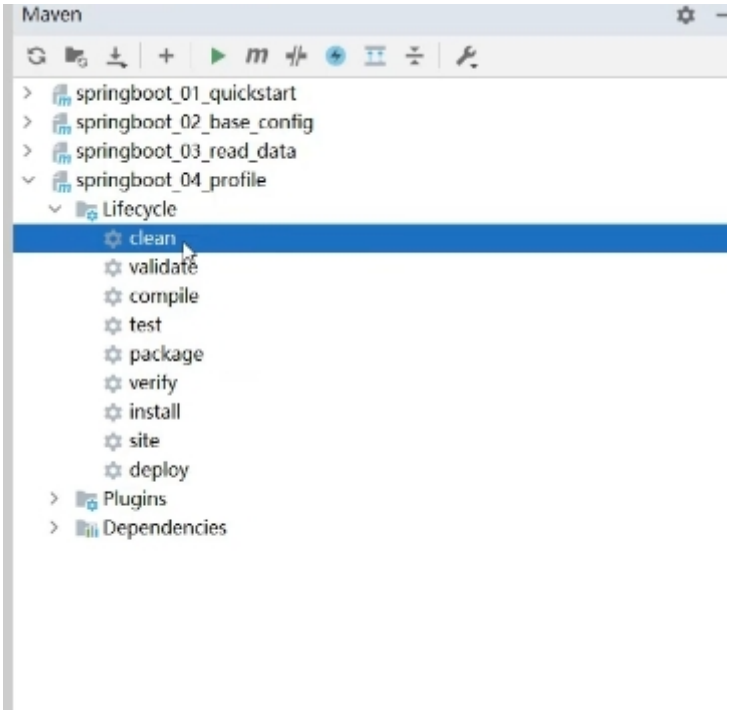
```properties
server.port=81
```

- 环境分类配置文件application-**test**.properties

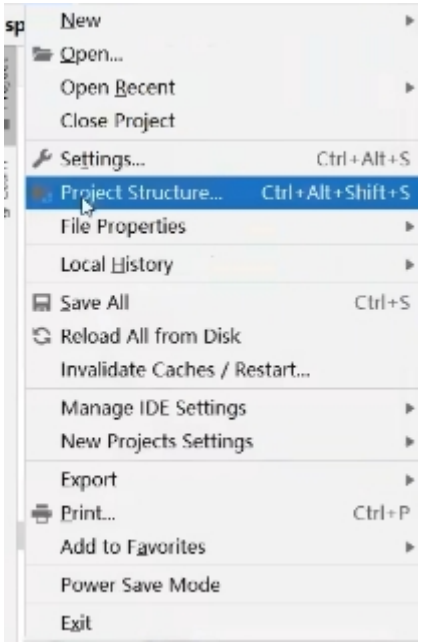
```properties
server.port=82
```

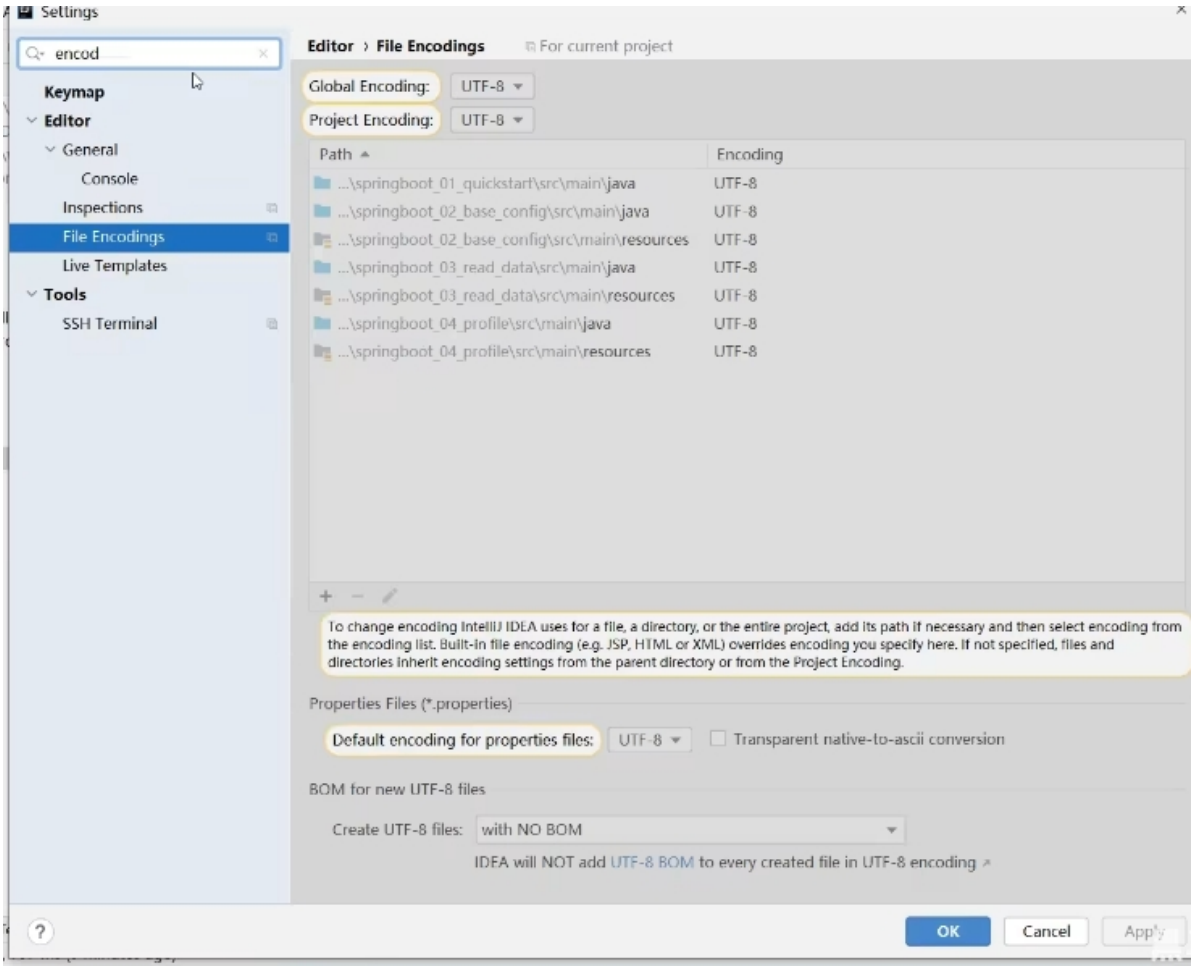
10、多环境命令行启动参数设置

1、先执行clean

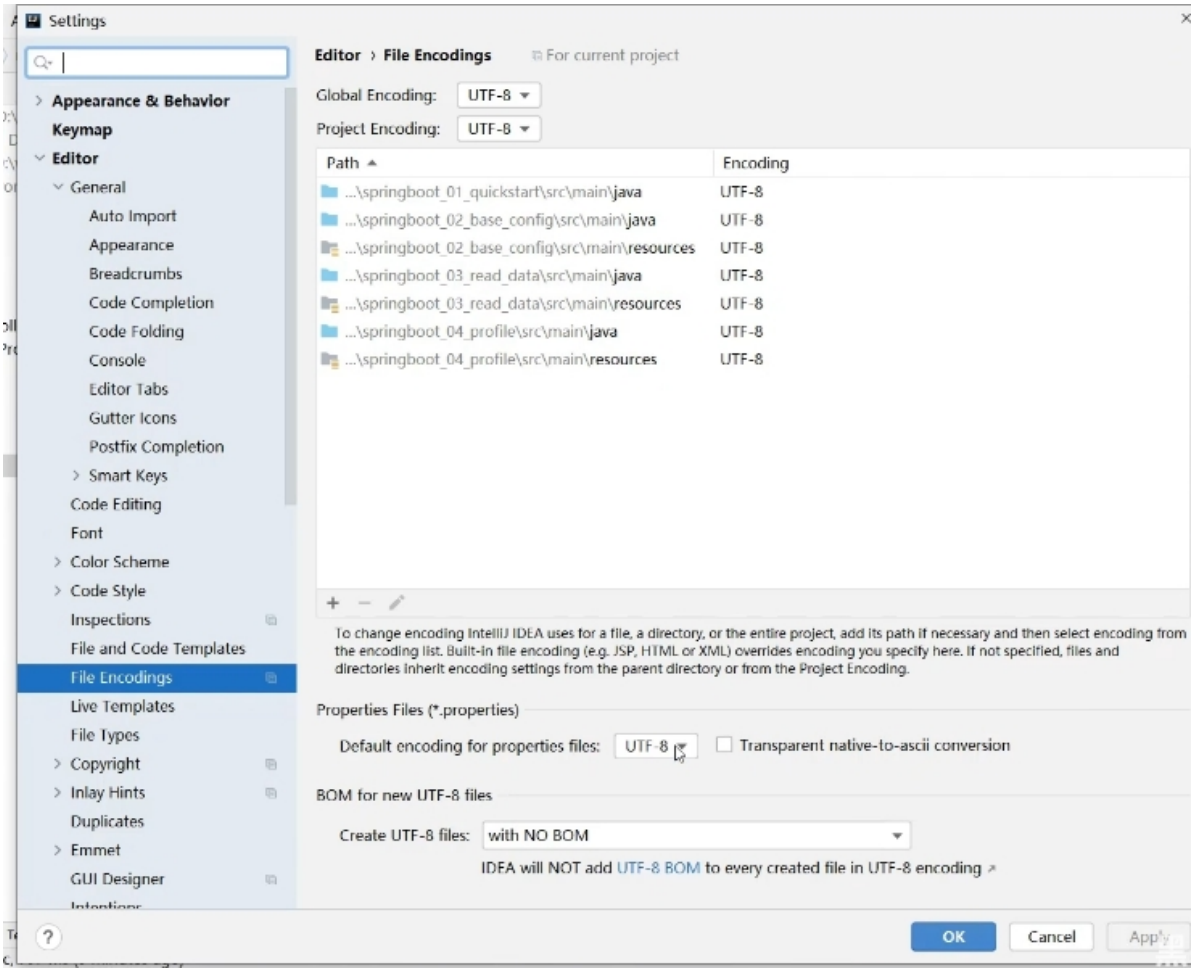


2、改配置




Settings
encod
Keymap
Editor
General
Console
Inspections
File Encodings
Live Templates
Tools
SSH Terminal
Editor › File Encodings
For current project
Global Encoding: UTF-8
Project Encoding: UTF-8
Path
Encoding
...\springboot_01_quickstart\src\main\java UTF-8
...\springboot_02_base_config\src\main\java UTF-8
...\springboot_02_base_config\src\main\resources UTF-8
...\springboot_03_read_data\src\main\java UTF-8
...\springboot_03_read_data\src\main\resources UTF-8
...\springboot_04_profile\src\main\java UTF-8
...\springboot_04_profile\src\main\resources UTF-8
To change encoding IntelliJ IDEA uses for a file, a directory, or the entire project, add its path if necessary and then select encoding from the encoding list. Built-in file encoding (e.g. JSP, HTML or XML) overrides encoding you specify here. If not specified, files and directories inherit encoding settings from the parent directory or from the Project Encoding.
Properties Files (*.properties)
Default encoding for properties files: UTF-8
Transparent native-to-ascii conversion
BOM for new UTF-8 files
Create UTF-8 files: with NO BOM
IDEA will NOT add UTF-8 BOM to every created file in UTF-8 encoding
OK
Cancel
Apply

3、全改


Settings
Editor › File Encodings
For current project
Global Encoding: UTF-8
Project Encoding: UTF-8
Path
Encoding
...\springboot_01_quickstart\src\main\java UTF-8
...\springboot_02_base_config\src\main\java UTF-8
...\springboot_02_base_config\src\main\resources UTF-8
...\springboot_03_read_data\src\main\java UTF-8
...\springboot_03_read_data\src\main\resources UTF-8
...\springboot_04_profile\src\main\java UTF-8
...\springboot_04_profile\src\main\resources UTF-8
To change encoding IntelliJ IDEA uses for a file, a directory, or the entire project, add its path if necessary and then select encoding from the encoding list. Built-in file encoding (e.g. JSP, HTML or XML) overrides encoding you specify here. If not specified, files and directories inherit encoding settings from the parent directory or from the Project Encoding.
Properties Files (*.properties)
Default encoding for properties files: UTF-8
Transparent native-to-ascii conversion
BOM for new UTF-8 files
Create UTF-8 files: with NO BOM
IDEA will NOT add UTF-8 BOM to every created file in UTF-8 encoding
Appearance & Behavior
Keymap
Editor
General
Auto Import
Appearance
Breadcrumbs
Code Completion
Code Folding
Console
Editor Tabs
Gutter Icons
Postfix Completion
Smart Keys
Code Editing
Font
Color Scheme
Code Style
Inspections
File and Code Templates
File Encodings
Live Templates
File Types
Copyright
Inlay Hints
Duplicates
Emmet
GUI Designer
OK
Cancel
Apply

多环境启动命令格式


临时改端口?
JAR
测试
后端

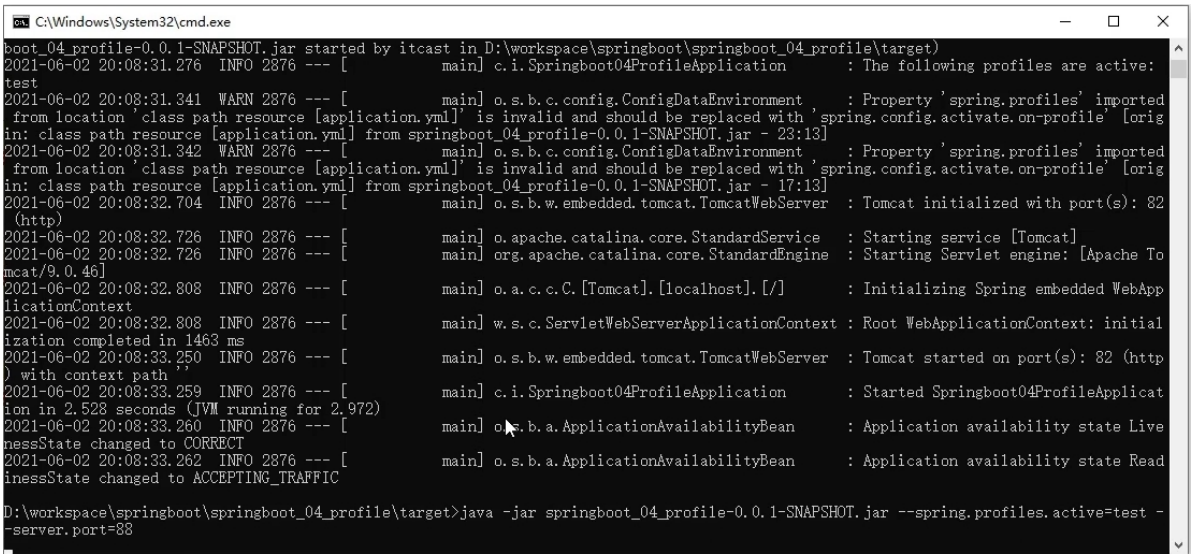
多环境启动命令格式

- 带参数启动SpringBoot

```
java -jar springboot.jar --spring.profiles.active=test
```

```
java -jar springboot.jar --server.port=88
```

```
java -jar springboot.jar --server.port=88 --spring.profiles.active=test
```



参数加载优先顺序

- 参看https://docs.spring.io/spring-boot/docs/current/reference/html/spring-boot-features.html#boot-features-external-config

1. Default properties (specified by setting `SpringApplication.setDefaultProperties`).
2. `@PropertySource` annotations on your `@Configuration` classes. Please note that such property sources are not added to the `Environment` until the application context is being refreshed. This is too late to configure certain properties such as `logging.*` and `spring.main.*` which are read before refresh begins.
3. Config data (such as `application.properties` files)
4. A `RandomValuePropertySource` that has properties only in `random.*`.
5. OS environment variables.
6. Java System properties (`System.getProperties()`).
7. JNDI attributes from `java:comp/env`.
8. `ServletContext` init parameters.
9. `ServletConfig` init parameters.
10. Properties from `SPRING_APPLICATION_JSON` (inline JSON embedded in an environment variable or system property).
11. Command line arguments.
12. `properties` attribute on your tests. Available on `@SpringBootTest` and the test annotations for testing a particular slice of your application.
13. `@TestPropertySource` annotations on your tests.
14. Devtools global settings properties in the `$HOME/.config/spring-boot` directory when devtools is active.

11、多环境开发兼容问题（Maven与boot）

多环境开发控制

profile

profile

步骤 Maven与SpringBoot多环境兼容

①：Maven中设置多环境属性

```
<profiles>
    <profile>
        <id>dev_env</id>
        <properties>
            <profile.active>dev</profile.active>
        </properties>
        <activation>
            <activeByDefault>true</activeByDefault>
        </activation>
    </profile>
    <profile>
        <id>pro_env</id>
        <properties>
            <profile.active>pro</profile.active>
        </properties>
    </profile>
    <profile>
        <id>test_env</id>
        <properties>
            <profile.active>test</profile.active>
        </properties>
    </profile>
</profiles>
```

步骤 Maven与SpringBoot多环境兼容

②：SpringBoot中引用Maven属性

```
spring:
  profiles:
    active: ${profile.active}
---
spring:
  profiles: pro
server:
  port: 80
---
spring:
  profiles: dev
server:
  port: 81
---
spring:
  profiles: test
server:
  port: 82
```

步骤 Maven与SpringBoot多环境兼容

②：SpringBoot中引用Maven属性

```yaml
spring:
  profiles:
    active: ${profile.active}
---
spring:
  profiles: pro
server:
  port: 80
---
spring:
  profiles: dev
server:
  port: 81
---
spring:
  profiles: test
server:
  port: 82
```

步骤 Maven与SpringBoot多环境兼容

②：SpringBoot中引用Maven属性

```yaml
spring:
  profiles:
    active: ${profile.active}
---
spring:
  profiles: pro
server:
  port: 80
---
spring:
  profiles: dev
server:
  port: 81
---
spring:
  profiles: test
server:
  port: 82
```

```xml
<profile>
    <id>dev_env</id>
    <properties>
        <profile.active>dev</profile.active>
    </properties>
    <activation>
        <activeByDefault>true</activeByDefault>
    </activation>
</profile>
```

多环境开发控制

- Maven指令执行完毕后，生成了对应的包，其中类参与编译，但是配置文件并没有编译，而是复制到包中

```yaml
spring:
  profiles:
    active: ${profile.active}
```

复制前 → 复制后

```yaml
spring:
  profiles:
    active: ${profile.active}
```

- 解决思路：对于源码中非java类的操作要求加载Maven对应的属性，解析${}占位符

多环境开发控制

- Maven指令执行完毕后，生成了对应的包，其中类参与编译，但是配置文件并没有编译，而是复制到包中

```yaml
spring:
  profiles:
    active: ${profile.active}
```

复制前 ⟶ 复制后

```yaml
spring:
  profiles:
    active: ${profile.active}
```

- 解决思路：对于源码中非java类的操作要求加载Maven对应的属性，解析${}占位符

12、配置文件分类

配置文件分类


```
java -jar springboot.jar --spring.profiles.active=test --server.port=85 --server.servlet.context-path=/heima --server.tomcat.connection-timeout=-1 …… …… …… …… ……
```

Springboot2.4.0还有一个小bug:

config中必须有子目录

13、springboot整合Junit

- SpringBoot整合JUnit

```
@SpringBootTest
class Springboot07JunitApplicationTests {
    @Autowired
    private BookService bookService;
    @Test
    public void testSave(){
        bookService.save();
    }
}
```

- Spring整合JUnit（复习）

```
@RunWith(SpringJUnit4ClassRunner.class)                    设置运行器
@ContextConfiguration(classes = SpringConfig.class)        加载环境
public class UserServiceTest {
    @Autowired
    private BookService bookService;                       注入测试对象

    @Test
    public void testSave(){
        bookService.save();                                测试功能
    }
}
```

整合JUnit

- 名称：@SpringBootTest
- 类型：**测试类注解**
- 位置：测试类定义上方
- 作用：设置JUnit加载的SpringBoot启动类
- 范例：

```
@SpringBootTest(classes = Springboot07JunitApplication.class)
class Springboot07JunitApplicationTests {}
```

- 相关属性
  - classes：设置SpringBoot启动类

注意事项

如果测试类在SpringBoot启动类的包或子包中，可以省略启动类的设置，也就是省略classes的设定

14、springboot整合mybatis

基于SpringBoot实现SSM整合

- SpringBoot整合Spring（不存在）
- SpringBoot整合SpringMVC（不存在）
- SpringBoot整合MyBatis（主要）

整合MyBatis

- Spring整合MyBatis（复习）
  - SpringConfig
    - 导入JdbcConfig
    - 导入MyBatisConfig

```
@Configuration
@ComponentScan("com.itheima")
@PropertySource("classpath:jdbc.properties")
@Import({JdbcConfig.class,MyBatisConfig.class})
public class SpringConfig {
}
```

- Spring整合MyBatis（复习）
  - SpringConfig
    - 导入JdbcConfig
    - 导入MyBatisConfig
  - JDBCConfig
    - 定义数据源（加载propertie

```
public class JdbcConfig {
    @Value("${jdbc.driver}")
    private String driver;
    @Value("${jdbc.url}")
    private String url;
    @Value("${jdbc.username}")
    private String userName;
    @Value("${jdbc.password}")
    private String password;

    @Bean
    public DataSource getDataSource(){
        DruidDataSource ds = new DruidDataSource();
        ds.setDriverClassName(driver);
        ds.setUrl(url);
        ds.setUsername(userName);
        ds.setPassword(password);
        return ds;
    }
}
```

整合MyBatis

- Spring整合MyBatis（复习）
  - SpringConfig
    - 导入JdbcConfig
    - 导入MyBatisConfig
  - JDBCConfig
    - 定义数据源（加载propertie

```java
public class JdbcConfig {
    @Value("${jdbc.driver}")
    private String driver;
    @Value("${jdbc.url}")
    private String url;
    @Value("${jdbc.username}")
    private String userName;
    @Value("${jdbc.password}")
    private String password;

    @Bean
    public DataSource g
        DruidDataSource
        ds.setDriverCla
        ds.setUrl(url);
        ds.setUsername(userName);
        ds.setPassword(password);
        return ds;
    }
}
```

```properties
#jdbc.properties
jdbc.driver=com.mysql.jdbc.Driver
jdbc.url=jdbc:mysql://localhost:3306/spring_db
jdbc.username=root
jdbc.password=itheima
```

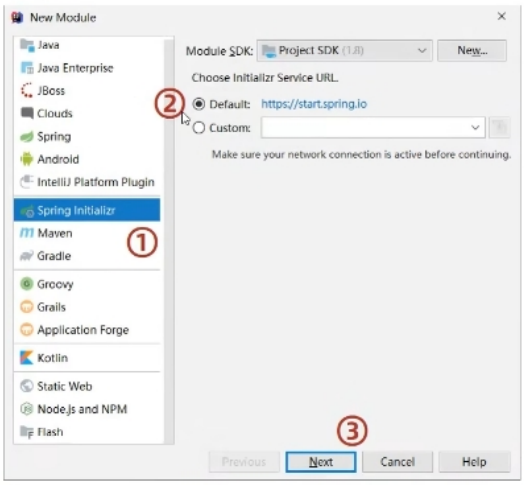
整合MyBatis

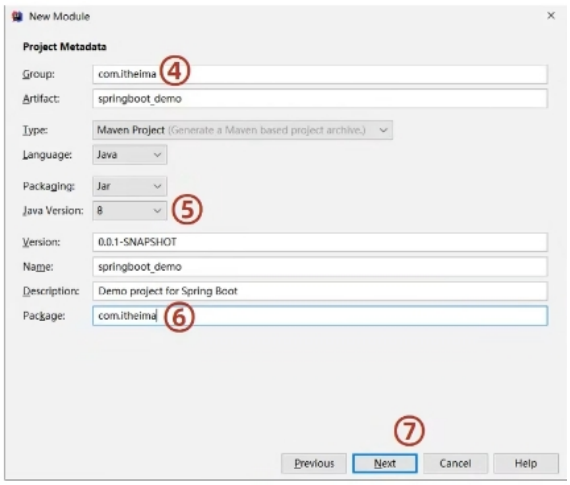
- Spring整合MyBatis（复习）
  - SpringConfig
    - 导入JdbcConfig
    - 导入MyBatisConfig
  - JDBCConfig
    - 定义数据源（加载propertie
  - MyBatisConfig
    - 定义SqlSessionFactoryBean
    - 定义映射配置

```java
@Bean
public SqlSessionFactoryBean getSqlSessionFactoryBean(DataSource dataSource){
    SqlSessionFactoryBean ssfb = new SqlSessionFactoryBean();
    ssfb.setTypeAliasesPackage("com.itheima.domain");
    ssfb.setDataSource(dataSource);
    return ssfb;
}
```

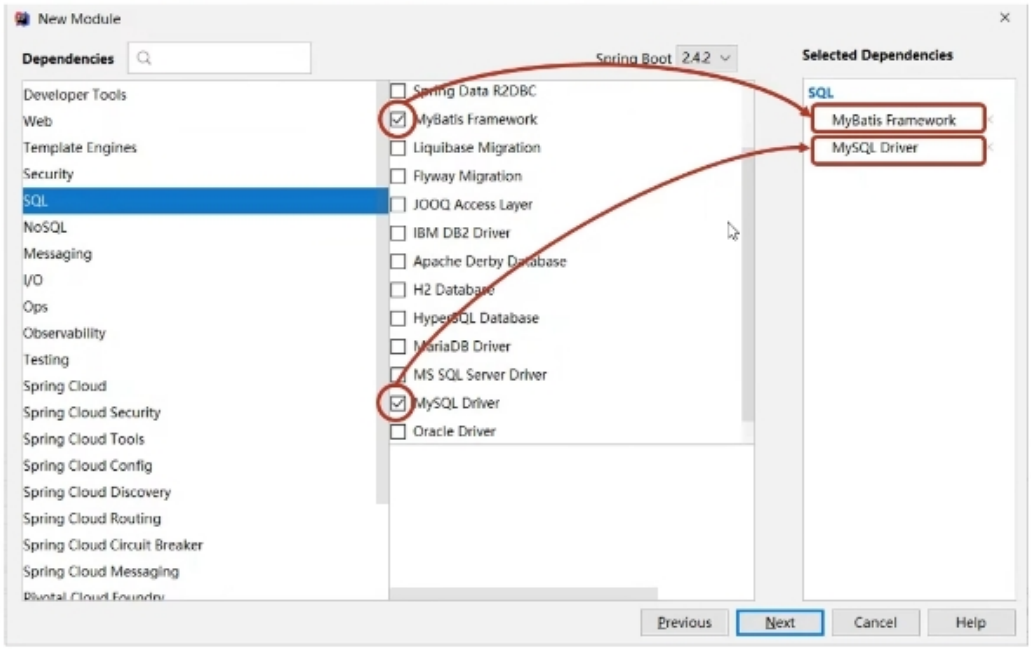
```java
@Bean
public MapperScannerConfigurer getMapperScannerConfigurer(){
    MapperScannerConfigurer msc = new MapperScannerConfigurer();
    msc.setBasePackage("com.itheima.dao");
    return msc;
}
```

①：创建新模块，选择Spring初始化，并配置模块相关基础信息





②：选择当前模块需要使用的技术集（MyBatis、MySQL）



③：设置数据源参数

```yaml
spring:
  datasource:
    type: com.alibaba.druid.pool.DruidDataSource
    driver-class-name: com.mysql.cj.jdbc.Driver
    url: jdbc:mysql://localhost:3306/ssm_db
    username: root
    password: root
```

注意事项

SpringBoot版本低于2.4.3(不含)，Mysql驱动版本大于8.0时，需要在url连接串中配置时区

```
jdbc:mysql://localhost:3306/ssm_db?serverTimezone=UTC
```

或在MySQL数据库端配置时区解决此问题

④：定义数据层接口与映射配置

```java
@Mapper
public interface UserDao {
    @Select("select * from user")
    public List<User> getAll();
}
```

⑤：测试类中注入dao接口，测试功能

```
@SpringBootTest
class Springboot08MybatisApplicationTests {
    @Autowired
    private BookDao bookDao;
    @Test
    public void testGetById() {
        Book book = bookDao.getById(1);
        System.out.println(book);
    }
}
```

15、案例：基于springboot实现ssm整合