# SpringBoot用法

## 配置文件

### 配置文件名称

说明:SpringBoot程序默认加载名为application这样的配置文件.进行加载.

### Application.properties说明

Springboot默认提供的是properties配置文件.但是该配置文件加载繁琐.官方不建议使用.

#修改端口号

server.port=8090

#默认项目部署到了ROOT文件夹中

server.servlet.context-path=/

### YML文件写法

#1.使用yml文件有层级结构 缩进

#2.使用yml文件key value 之间使用 :号连接

#3.赋值时:号后边必须加空格

server:

port: 8090

servlet:

context-path: /

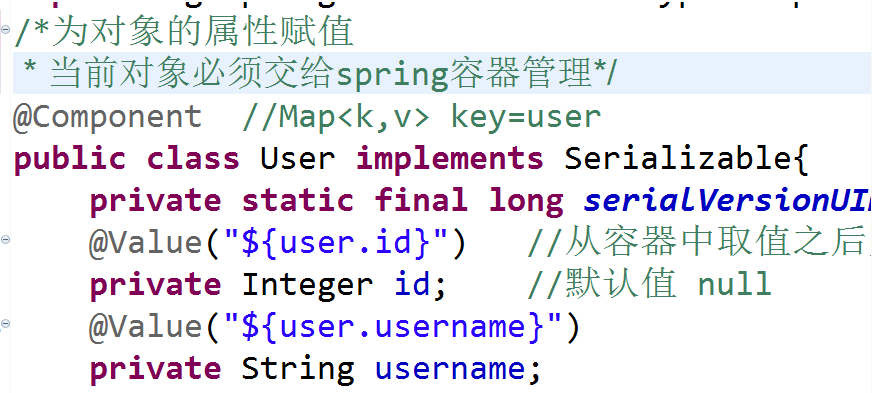
## SpringBoot为属性赋值

### 业务需求

说明:在配置文件中定义user信息id/username/age.之后将user信息赋值给user对象.

### @Value注解赋值

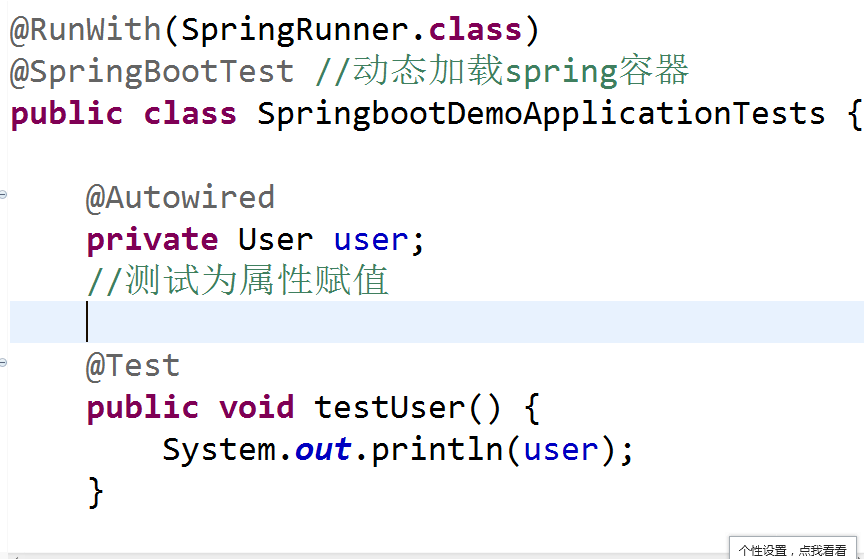
#### 编辑pojo对象



#### 编辑yml配置文件

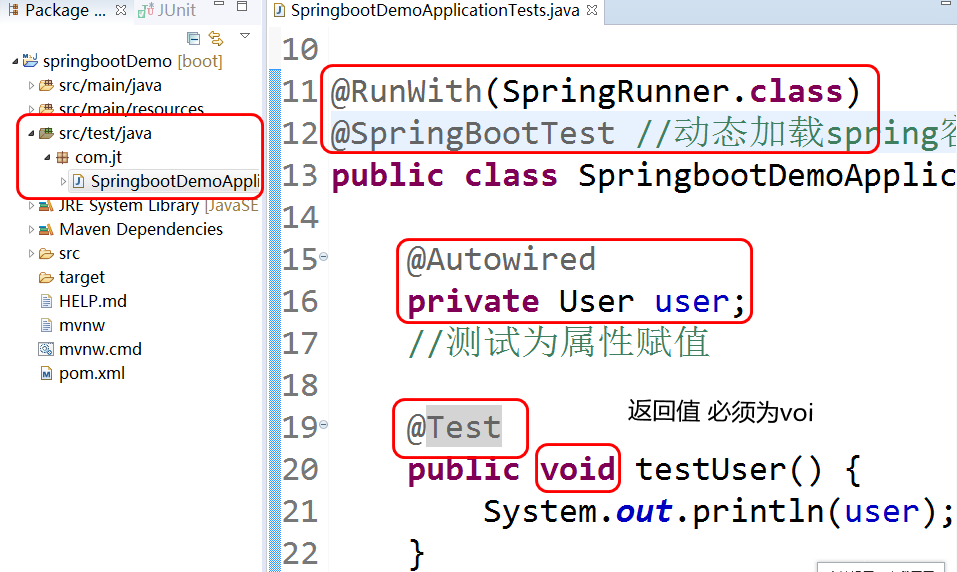
* 

#### 测试类取值



#### 关于赋值总结

1. 编辑yml配置文件 注意缩进
2. 编辑POJO对象时 必须交给Spring容器管理
3. 注意取值注解写法 @Value(“${key}”)
4. 注意测试类位置



#### 注意事项

利用容器取值时,需要额外的注意名称 不能为user.name,否则可能与计算机名称重复!!!!

### 利用前缀赋值

#### 业务说明

当对象的属性中有多个属性时,赋值时繁琐,则可以利用前缀为属性赋值.

#### 引入第三方jar包

<!--添加属性注入依赖 -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-configuration-processor</artifactId>

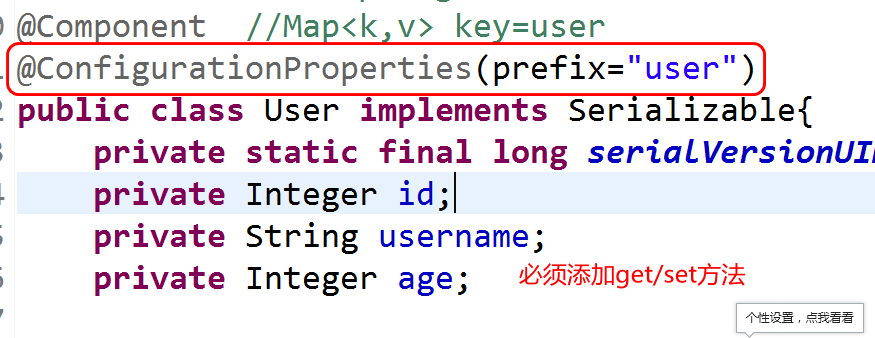
<optional>true</optional>

</dependency>

#### 添加注解

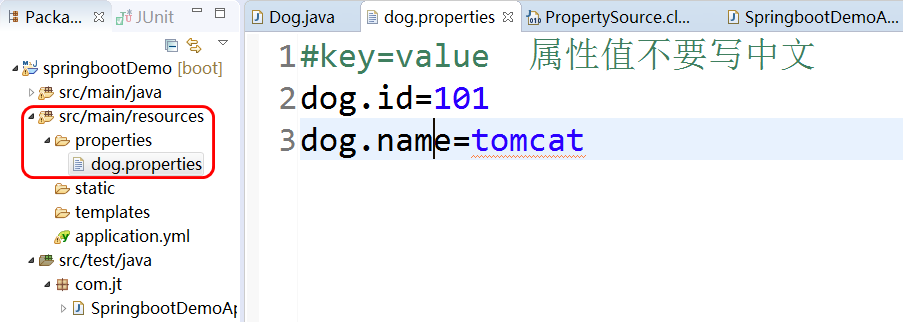
1. 编辑YML文件





### 制定配置文件为属性赋值

#### 编辑properties配置文件



#### 添加注解



## 多环境配置

### 需求说明

软件在开发阶段会有多个环境进行测试.这时需要频繁的修改yml配置,导致开发不便.

语法: --- 进行环境分割

### Yml文件写法

#编辑环境默认值

spring:

profiles:

active: dev

#1.使用yml文件有层级结构 缩进

#2.使用yml文件key value 之间使用 :号连接

#3.赋值时:号后边必须加空格

--- #环境分割线

spring:

profiles: dev

server:

port: 8090

servlet:

context-path: /

#为user属性赋值 可以编辑中文

user:

id: 100

username: 哥斯拉2

age: 2002

--- #环境分割线

spring:

profiles: prod

server:

port: 8081

servlet:

context-path: /jt

user:

id: 2000

username: 哥斯拉10

age: 10000

## 实现项目热部署

### 需求说明

当修改YML文件后,程序需要手动启动,需要实现自动的部署

### 引入jar包

在pom.xml文件中添加该文件信息.当ctrl+s时检查项目是否自动部署.

<!--支持热部署 -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>springloaded</artifactId>

<version>1.2.8.RELEASE</version>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-devtools</artifactId>

</dependency>

## 配置类引入

### 需求说明

在原有SSM框架中,需要通过配置文件 xxxx.xml进行对象的配置

利用<bean id=”XXX” class=”路径.名称” />

但是引入的配置文件太多以后导致程序加载速度慢.并且配置文件编辑繁琐.

引入配置类代替配置文件.

### 引入第三方jar包

//该类是写在第三方jar包中 其他程序调用即可

**public** **class** Shiro {

**public** **void** pri() {

System.***out***.println("我是权限控制的类");

}

}

### 编辑配置类

//该类中标识就是xml中数据 <bean id="方法名" class="返回值类型" />

@Configuration

**public** **class** ShiroConfig {

/\*\*

\* public 返回值类型 方法名称(参数列表)

\* <bean id="方法名" class="返回值类型" />

\* **@return**

\*/

@Bean

**public** Shiro shiro() {

**return** **new** Shiro();

}

}

### 编辑测试类

//测试配置类

@Autowired

**private** Shiro shiro;

@Test

**public** **void** testShiro() {

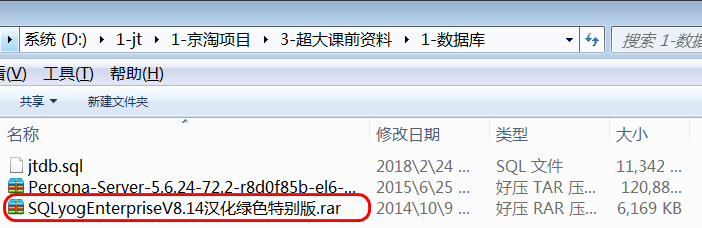
shiro.pri();

}

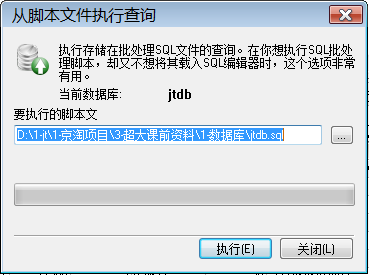
## 导入京淘数据库

### 导入数据库操作工具

说明:在课前资料中找到数据文件.



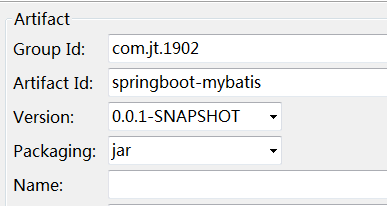
### 导入mysql数据库



之后点击执行导入数据库文件.

## SpringBoot整合Mybatis

### 创建项目



### 添加pom.xml文件

<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.jt.1902</groupId>

<artifactId>springboot-mybatis</artifactId>

<version>0.0.1-SNAPSHOT</version>

<properties>

<java.version>1.8</java.version>

</properties>

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>2.1.3.RELEASE</version>

<relativePath />

</parent>

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

<!--添加属性注入依赖 -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-configuration-processor</artifactId>

<optional>true</optional>

</dependency>

<!--引入插件lombok 自动的set/get/构造方法插件 -->

<dependency>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

</dependency>

<!--支持热部署 -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>springloaded</artifactId>

<version>1.2.8.RELEASE</version>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-devtools</artifactId>

</dependency>

<!--引入数据库驱动 -->

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<scope>runtime</scope>

</dependency>

<!--引入druid数据源 -->

<dependency>

<groupId>com.alibaba</groupId>

<artifactId>druid</artifactId>

<version>1.1.12</version>

</dependency>

<!--spring整合mybatis 暂时 -->

<dependency>

<groupId>org.mybatis.spring.boot</groupId>

<artifactId>mybatis-spring-boot-starter</artifactId>

<version>1.3.2</version>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

</project>

### 编辑YML映射文件

server:

port: 8090

servlet:

context-path: /

#引入数据源配置

spring:

datasource:

#引入druid数据源

type: com.alibaba.druid.pool.DruidDataSource

driver-class-name: com.mysql.jdbc.Driver

url: jdbc:mysql://127.0.0.1:3306/jtdb?serverTimezone=GMT%2B8&useUnicode=true&characterEncoding=utf8&autoReconnect=true&allowMultiQueries=true

username: root

password: root

#spring整合Mybatis

mybatis:

#配置别名包

type-aliases-package: com.jt.pojo

#配置mapper映射

mapper-locations: classpath:/mybatis/mappers/\*.xml

#开启驼峰映射

configuration:

map-underscore-to-camel-case: true

### 开启驼峰映射

表名:user

字段:user\_id,user\_name,user\_age,user\_sex

对象:User

属性:userId,username,userAge,userSex;

问题说明:

Mybatis中如果实现ORM对象映射,则要求对象中的属性与表中的字段必须一一映射. 但是现在表中的字段与属性名称不一致.中间多”\_”.导致映射不成功!!

解决方案:开启驼峰映射规则

Mybatis:

configuration:

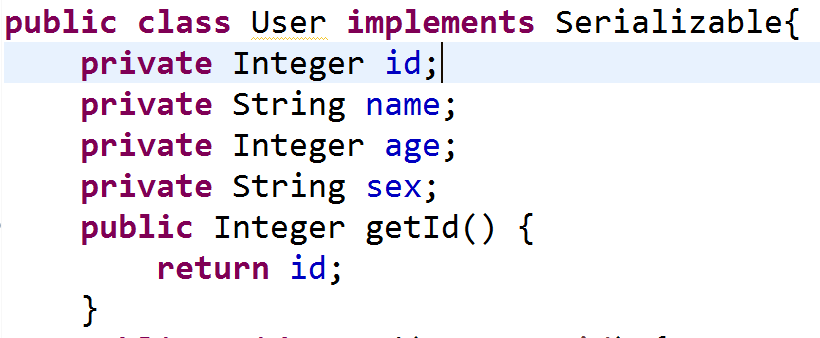
map-underscore-to-camel-case: true

则可以自动的实现对象的赋值.

驼峰映射原理:

1. 根据sql语句获取结果集对象.
2. 利用resultType实现赋值时发现字段与对象的属性不否.
3. 利用驼峰映射将字段user\_id中的”\_”去除---🡪userid
4. 将后边首字母大小------🡪userId
5. 将数据库中的值自动映射给对象中的属性userId

### 编辑POJO



### 编辑Mapper接口/映射文件



### 编辑Service

@Service

**public** **class** UserServiceImpl **implements** UserService {

//接口实例化: 1.创建实现类 2.创建代理对象 JDK CGLIB

//原则: 如果代理对象有接口则使用JDK代理

// 如果代理者没有实现接口 则使用cglib代理

@Autowired

**private** UserMapper userMapper;

@Override

**public** List<User> findAll() {

**return** userMapper.findAll();

}

}

### 编辑Controller

@RestController

**public** **class** UserController {

@Autowired

**private** UserService userService;

//用户通过localhost:8090/findAll获取用户数据

@RequestMapping("/findAll")

**public** List<User> findAll(){

**return** userService.findAll();

}

}

### 编辑主启动类

@SpringBootApplication

//为mapper接口创建代理对象

@MapperScan("com.jt.mapper")

**public** **class** SpringBootRun {

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

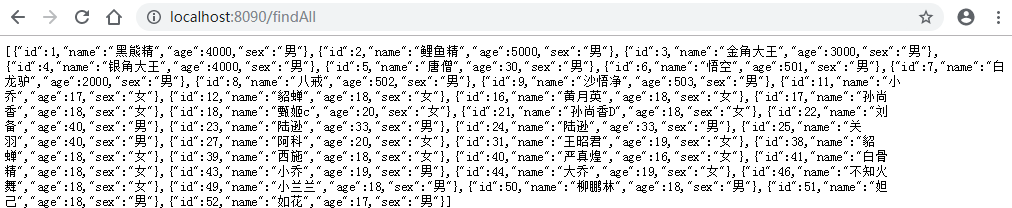
SpringApplication.*run*(SpringBootRun.**class**, args);

}

}

页面效果

### 页面效果



## Lombok(红番椒)

### 需求说明

对于POJO对象,不光需要编辑属性,还需要编辑get/set/toString/构造/equals…. 能否自动生成???

### 引入jar包

<!--引入插件lombok 自动的set/get/构造方法插件 -->

<dependency>

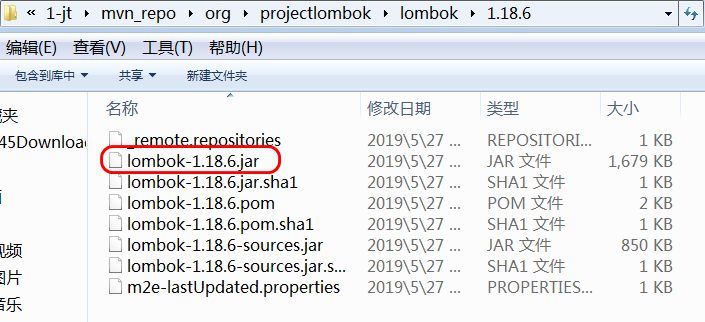
<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

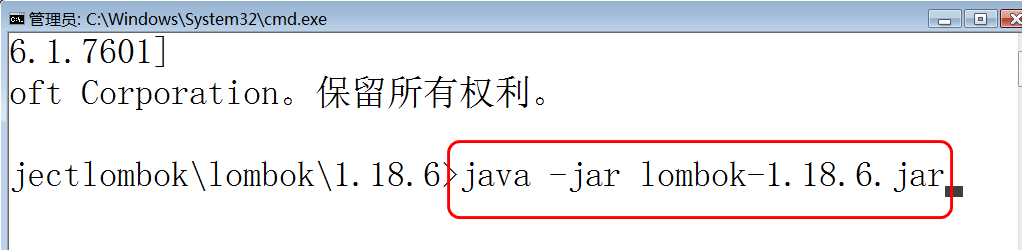
</dependency>

### 安装插件

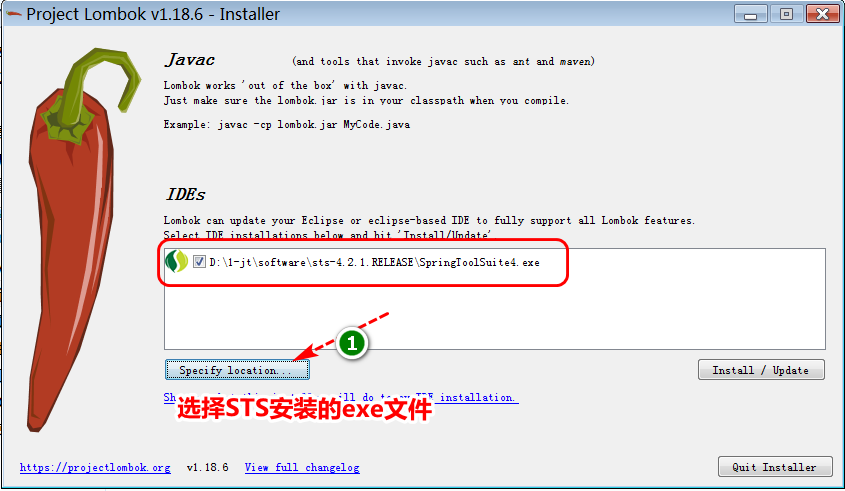
说明:根据pom.xml文件中的配置.找到lombok文件夹下的jar包文件



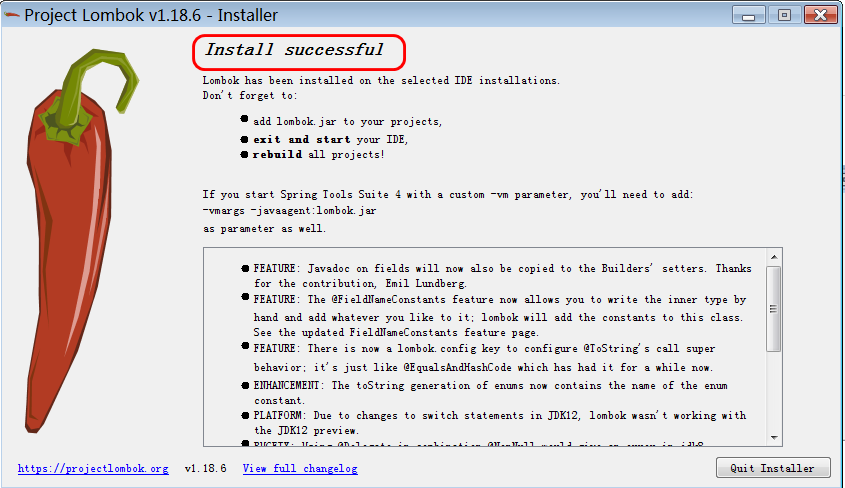
1. 利用java –jar 指令启动插件



选择STS文件

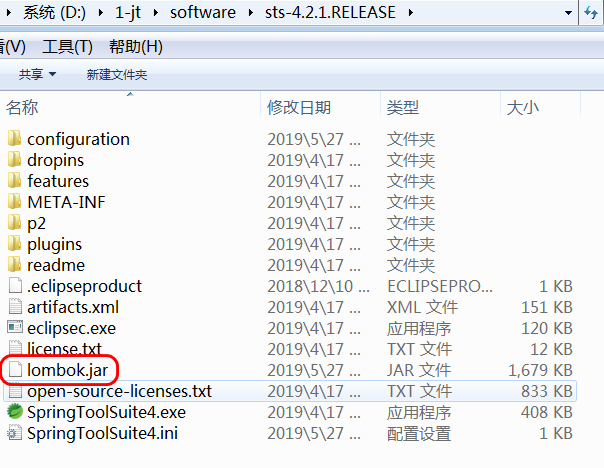


之后点击install 表示安装成功!!!



### 编辑配置文件

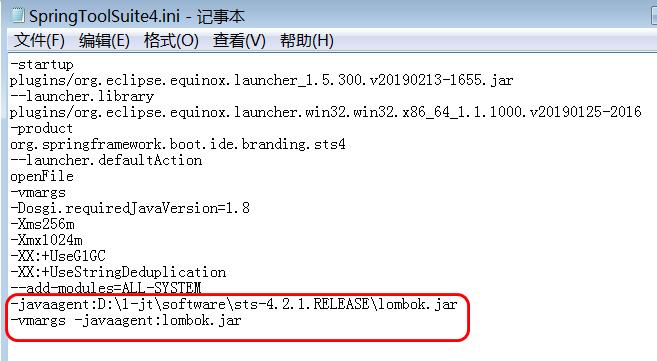
1. 检查是否生成lombok.jar包文件



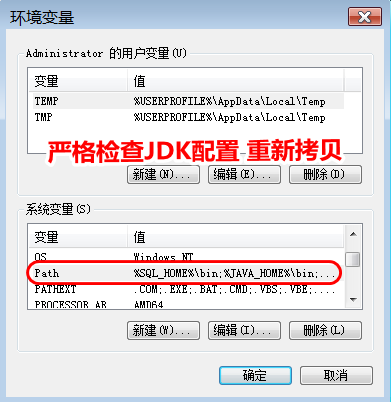
1. 编辑INI文件

在文件最后添加下行配置

-vmargs -javaagent:lombok.jar



修改完成之后,重启STS.如果重启报错,则检查JDK.之后重新执行上述操作.



### LomBok使用

 Lombok主要常用的注解有：@Data,@getter,@setter,@NoArgsConstructor,@AllArgsConstructor,@ToString,@EqualsAndHashCode,@Slf4j,@Log4j。我们一个一个来看：

@Data注解：在JavaBean或类JavaBean中使用，这个注解包含范围最广，它包含getter、setter、NoArgsConstructor注解，即当使用当前注解时，会自动生成包含的所有方法；

@getter注解：在JavaBean或类JavaBean中使用，使用此注解会生成对应的getter方法；

@setter注解：在JavaBean或类JavaBean中使用，使用此注解会生成对应的setter方法；

@NoArgsConstructor注解：在JavaBean或类JavaBean中使用，使用此注解会生成对应的无参构造方法；

@AllArgsConstructor注解：在JavaBean或类JavaBean中使用，使用此注解会生成对应的有参构造方法；

@ToString注解：在JavaBean或类JavaBean中使用，使用此注解会自动重写对应的toStirng方法；

@EqualsAndHashCode注解：在JavaBean或类JavaBean中使用，使用此注解会自动重写对应的equals方法和hashCode方法；

@Slf4j：在需要打印日志的类中使用，当项目中使用了slf4j打印日志框架时使用该注解，会简化日志的打印流程，只需调用info方法即可；

@Log4j：在需要打印日志的类中使用，当项目中使用了log4j打印日志框架时使用该注解，会简化日志的打印流程，只需调用info方法即可；

在使用以上注解需要处理参数时，处理方法如下（以@ToString注解为例，其他注解同@ToString注解）：

@ToString(exclude="column")

意义：排除column列所对应的元素，即在生成toString方法时不包含column参数；

@ToString(exclude={"column1","column2"})

意义：排除多个column列所对应的元素，其中间用英文状态下的逗号进行分割，即在生成toString方法时不包含多个column参数；

@ToString(of="column")

意义：只生成包含column列所对应的元素的参数的toString方法，即在生成toString方法时只包含column参数；；

@ToString(of={"column1","column2"})

意义：只生成包含多个column列所对应的元素的参数的toString方法，其中间用英文状态下的逗号进行分割，即在生成toString方法时只包含多个column参数；

@Accessors(chain=true)

链式加载

### 引用效果

说明:虽然自己没有写get/set方法.但是STS根据插件自动生成了class文件.

