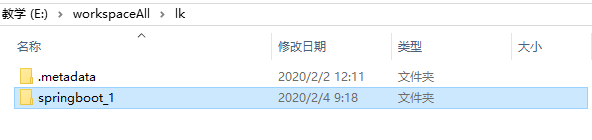
# SpringBoot高级

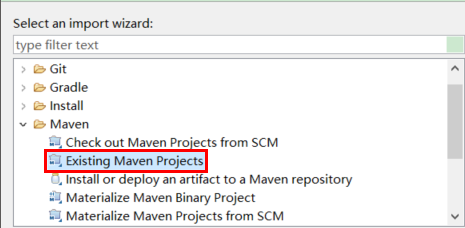
## 导入项目

### 导入Maven项目

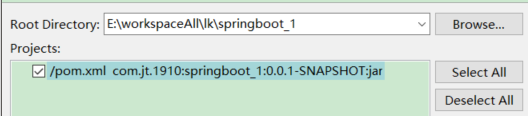
1. 将项目导入本地,本地工作空间中.



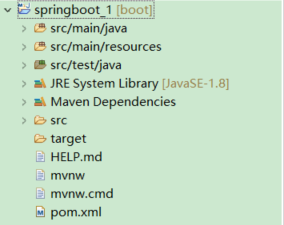
1. 使用导入工具进行maven导入
2. 选择maven项目



1. 导入maven项目



1. 导入效果



## SpringBoot-POM.XML文件说明

### parent标签作用

<!--导入公共的jar包文件 spring springmvc 并且定义2.2.4的相关联的jar包版本

防止jar包冲突.

-->

<parent>

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

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

<version>2.2.4.RELEASE</version>

<relativePath/> <!-- lookup parent from repository -->

</parent>

### 动态依赖

<!--项目的依赖项 -->

<dependencies>

<!--SpringMVC 传统方法需要手动的配置 springboot程序自动化部署. -->

<dependency>

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

<!--springboot的启动器包 -->

<!--maven中的jar包具有依赖的传递性 -->

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

</dependency>

<!--springboot启动项 加载test测试 -->

<dependency>

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

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

<scope>test</scope>

<exclusions>

<exclusion>

<groupId>org.junit.vintage</groupId>

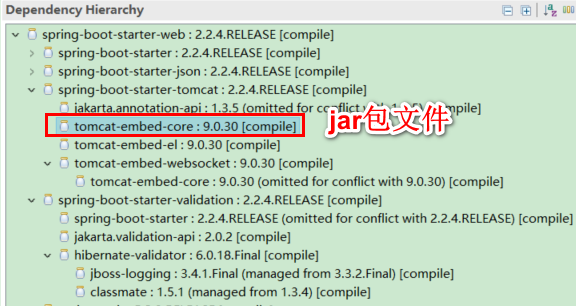
<artifactId>junit-vintage-engine</artifactId>

</exclusion>

</exclusions>

</dependency>

</dependencies>



### 插件配置

<!--build标签主要作用添加插件/配置 -->

<build>

<plugins>

<!--指定maven插件 负责构建maven项目 打包/部署/测试/文档生成 -->

<plugin>

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

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

</plugin>

</plugins>

</build>

## SpringBoot主启动类的说明

### SpringBoot程序启动过程

问题:springBoot项目中为什么只要导入jar包.程序就可以生效,而不需要特殊的配置.

1. main方法执行
2. 加载主启动类

@Target(ElementType.***TYPE***) //元注解 修饰注解的注解 标识修饰类

@Retention(RetentionPolicy.***RUNTIME***) //元注解 运行期有效

@Documented //文档的支持

@Inherited //可以被继承

@SpringBootConfiguration //springBoot的配置类注解. 主启动类同时也是配置类

@EnableAutoConfiguration //开启自动化的配置 引入依赖即可完成自动化的配置 开箱即用,

完成自动化的包扫描 @Controller @Service等

@ComponentScan

//指定注解生效/不生效

(excludeFilters = { @Filter(type = FilterType.***CUSTOM***, classes = TypeExcludeFilter.**class**),

@Filter(type = FilterType.***CUSTOM***, classes = AutoConfigurationExcludeFilter.**class**) })

**public** **@interface** SpringBootApplication {}

EnableAutoConfiguration : SpringBoot程序比较智能,只需要添加某些jar包文件一般以springBoot启动项居多.当程序启动时,可以自动化的部署启动.

1. 程序解析主启动类身上的注解.其中最为重要的是自动化的配置.其中包含配置文件自动加载.以及全部项目的**包扫描**

规则:以后写代码,必须写到主启动类的同包及子包下.

## SpringBoot为属性赋值

### @Value为属性赋值

#### 编辑YML配置文件

#语法: 1.K-V结构 使用":"号进行链接.

# 2.有层级关系 注意缩进(空格or tab键)

# 3.添加value时 注意空格.

server:

port: 8090 #指定端口号信息

#指定项目发布路径 tomcat 发布到哪个目录下?发布到webapp目录下的/ROOT中

servlet:

context-path: / #发布默认路径

#实现属性赋值操作 jdbc.driver

jdbc:

driver: mysqlDriver

name: mysql

#### 编辑JDBCService

@Service //将对象交给容器管理

**public** **class** JDBCService {

@Value("${jdbc.driver}") //spring中的spel表达式

**private** String driver;

@Value("${jdbc.name}")

**private** String name;

//利用批量赋值时,需要使用set方法实行数据的赋值.

**public** **void** testService() {

System.***out***.println(driver+":"+name);

}

}

#### 编辑Controller

@RestController //开启springMVC

**public** **class** JDBCController {

@Autowired

**private** JDBCService jdbcService;

@RequestMapping("/testJDBC")

**public** String testJDBC() {

jdbcService.testService();

**return** "测试成功!!!!";

}

}

### 批量为属性赋值

#### 编辑YML配置文件

#实现属性赋值操作 jdbc.driver

jdbc:

driver: mysqlDriver

name: mysql

#### 添加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>

#### 编辑Service类

@Service //将对象交给容器管理

@ConfigurationProperties(prefix="jdbc") //指定属性前缀

**public** **class** JDBCService {

//@Value("${jdbc.driver}") //spring中的spel表达式

**private** String driver;

//@Value("${jdbc.name}")

**private** String name;

//利用批量赋值时,需要使用set方法实行数据的赋值.

**public** **void** testService() {

System.***out***.println(driver+":"+name);

}

**public** String getDriver() {

**return** driver;

}

**public** **void** setDriver(String driver) {

**this**.driver = driver;

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

}

#### 关于属性赋值总结

1. 如果少量属性 使用@value方式赋值
2. 如果属性数量比较多 则使用批量赋值一般配置LOMBOK中的data



### 指定配置文件为属性赋值

说明:一般业务操作有会固定的properties文件.如何指定pro文件为属性赋值.

#### 新建配置文件

mysql.name=mysql

mysql.driver=mysqlDriver

#### 编辑业务处理类

@RestController

//指定配置文件 进行加载

@PropertySource("classpath:/properties/jdbc.properties")

**public** **class** MysqlController {

/\*\*

\* 要求:按照指定的配置文件 为属性赋值.

\* 1.利用@Value为属性赋值

\* 2.批量为属性赋值

\*/

@Value("${mysql.name}")

**private** String mysqlName;

@Value("${mysql.driver}")

**private** String mysqlDriver;

@RequestMapping("/getMsg")

**public** String getMsg() {

**return** mysqlName+":"+mysqlDriver;

}

}

#### 关于maven路径加载问题

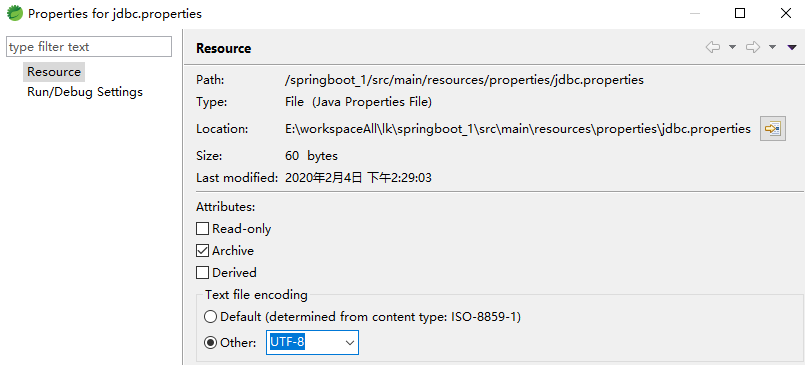
说明:

@PropertySource("classpath:/properties/jdbc.properties")

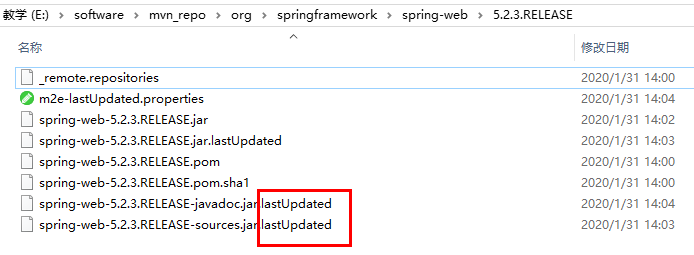
其中/指的是编译后的项目文件目录结构



#### 关于properties文件编辑中文设定



#### 关于源码导入问题



可以删除jar包文件之后重新下载即可.

## LOMBOK

### 引入lombok插件

#### 1.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>

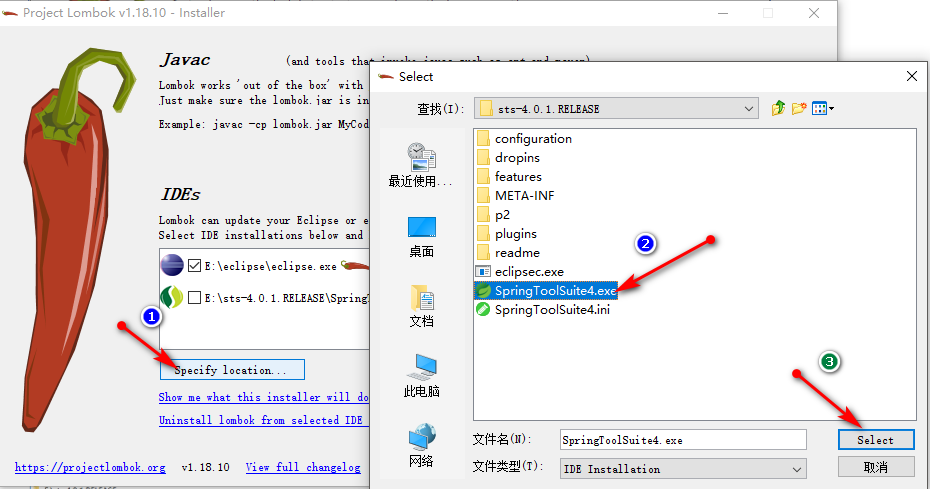
</dependency>

2.找到lombokjar包文件位置 以java方式运行\

计算机生成了可选文字:
E:\software\mvn0\or\ro•ectlombok\lombok\1.18.4
名称
囗
0
0
囗
囗
囗
0
《ombok一1．18.4.jar
《ombok一1．18.4.jar.sha1
《ombok一1．18.4.pom
《ombok一1．18.4.pom.sha1
《ombok一1．18.4-sources.Jar
《ombok一1．18.4-sources.jar.shal
m2e-lastUpdated.properties
修改日耻
2018/12/1022：45
2018/12/1022：45
2018/12/1022：45
2018/12/1022：45
2018/12/1022：45
2018/12/1022：45
2018/12/1022：45
2018/12/1022：45
REPOSITORIES，
JAR文亻牛
SHAI文亻牛
POM文亻牛
SHAI文亻牛
JAR文亻牛
SHAI文亻牛
PROPERTIES文亻牛
1KB
1，550KB
1KB
2KB
1KB
842KB
1KB
1KB

3.命令:

计算机生成了可选文字:
E：\software\mvnrepo.org/projectlombok\lombok\l.18·
ProjectLombokVI．1&4-Installer
(andtoolsthatinvokejavacsuchaSantandnaven)
Lombokworksoutoftheboxwithjavac.
JustmakeSuretheIombok.jar1S1nyourclasspathWhenyoucompile.
Example:a
4
JaVa
-jarlombok-l.18·
运行jar包文件
4·
Jar
Lombokcan
updateyour
Eclipseor
eclipse¯based
IDEt0ful1
supportall
Lombokfeatures.
SelectIDE
installations
belowandhit
Install/Update，
自动选中程序
[乏《E。\sts—4.0．1.RELEASE\SpringToolSuite4.ee
https:
Specifylocation.
Showmewhatthisinstaller
projectlombok.0禺V
Viewful1
1.1&4
wiIIdot0myIDEinstallation.
changel0
Install
Qult
Update
Installer



之后点击install/update即可

在eclipse安装的根目录下会生成lombok.jar包文件,表示插件安装成功

计算机生成了可选文字:
E:\sts-4.O.1.RELEASE
名称
dropins
features
META-INF
plugins
囗
.eclipseproduct
0
artifacts.xml
0
lombok.jar
SpringToolSuite4.exe
0SpringToolSuite4.ini
修改日耻
2018/12/249：05
2018/12/2014：03
2018/11/279：12
2018/11/279：12
2018/12/415：50
2018/11/279：12
2018/11/279：12
2018/9/57：47
2018/10/300：49
2018/10/300：49
2018/10/300：44
2018/12/2417：15
2018/10/300：49
2018/12/2417：15
文亻牛夹
文亻牛夹
文亻牛夹
文亻牛夹
文亻牛夹
文亻牛夹
文亻牛夹
ECLIPSEPRODUC..
XML文亻牛
应程序
TXT文亻牛
」AR文亻牛
应程序
《NI文亻牛
1KB
150KB
120KB
12KB
1，550KB
408KB
1KB
自动生成jar包文件

安装完成后重启Eclipse即可

5.常用注解

 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

### 安装LOMBOK疑问

问题说明:

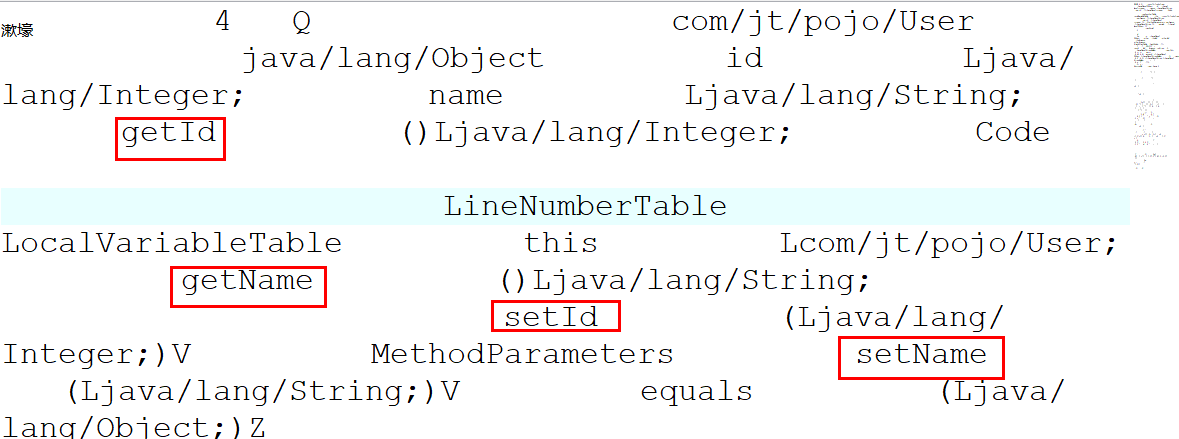
在本机中安装lombok插件,可以动态的为用户添加/set/get等方法.

如果将来项目发布到Linux系统中,问:是否需要再次安装lombok插件

答案: 不需要

工作原理:

当类添加了@Data注解时,IDE由于已经添加了LOMBOK插件,所以会动态的为类添加set/get等方法.形成全新的.CLASS文件



### 链式加载原理说明

@Data

@NoArgsConstructor //添加无参构造

@AllArgsConstructor //添加全参构造

@Accessors(chain=**true**) //链式加载 重写了set方法

**public** **class** User {

//常规POJO实体对象 最好使用包装类型进行定义.

**private** Integer id;

**private** String name;

/\*\*

\* 链式加载的原理说明,可以让对象连续调用方法

\*/

/\*public User setId(Integer id) {

this.id = id;

return this; //代表当前对象

}

public User setName(String name) {

this.name = name;

return this;

}\*/

}

## SpringBoot添加热部署

### 热部署需求

当代码发生变化,需要重新发布tomcat服务器.这样的操作称之为热部署.

### 导入jar包

<!--支持热部署 -->

<dependency>

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

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

</dependency>

## 环境切换

### 业务需求

环境介绍:

1. 本地/开发开发环境 dev
2. 测试环境 接近于真实的运行环境 test
3. 生产环境 真实的运行环境 prod

问题:

IP地址/端口号都相同吗? 答案:不一定都相同. 如果需要在不同的环境中运行服务器.则需要频繁的修改yml配置文件.

### 环境切换配置

#### 业务说明

虽然环境配置有所不同.环境中的配置的个数相同. 如果频繁修改环境配置,容易出错.

#### 编辑YML配置

#语法: 1.K-V结构 使用":"号进行链接.

# 2.有层级关系 注意缩进(空格or tab键)

# 3.添加value时 注意空格.

#选中默认的环境

spring:

profiles:

active: dev #指定当前使用的是生产环境配置

---

#环境1 定义环境名称 dev

spring:

profiles: dev

server:

port: 8090 #指定端口号信息

#指定项目发布路径 tomcat 发布到哪个目录下?发布到webapp目录下的/ROOT中

servlet:

context-path: / #发布默认路径

#实现属性赋值操作 jdbc.driver

jdbc:

driver: mysqlDriver

name: mysql

#环境分割

---

#环境2 prod

spring:

profiles: prod

server:

port: 8095 #指定端口号信息

#指定项目发布路径 tomcat 发布到哪个目录下?发布到webapp目录下的/ROOT中

servlet:

context-path: / #发布默认路径

#实现属性赋值操作 jdbc.driver

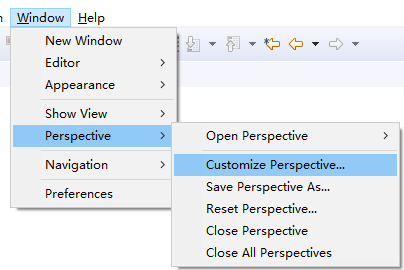
jdbc:

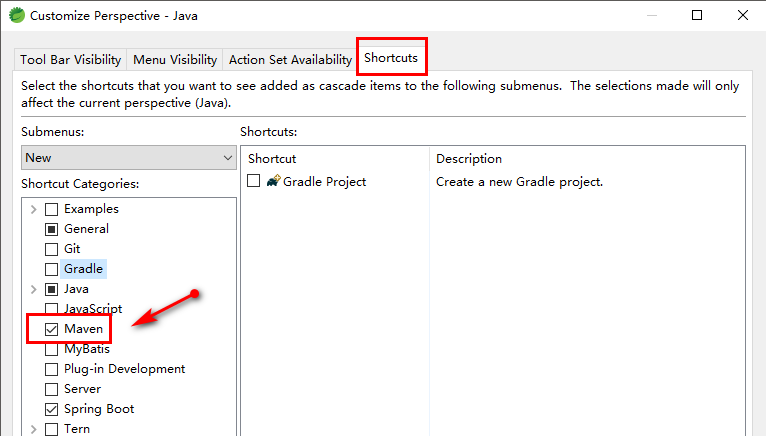
driver: mysqlDriver

name: mysql

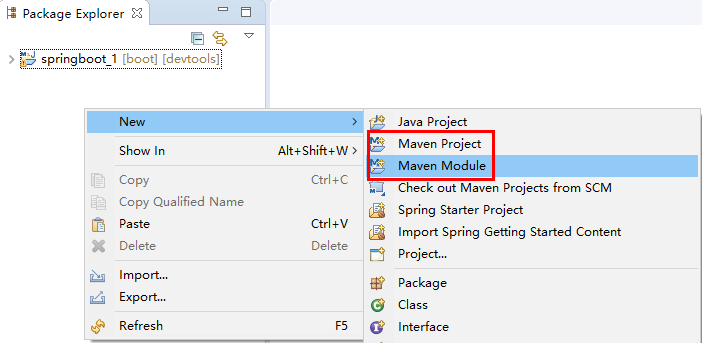
## SpringBoot整合Mybatis

### 修改new选项

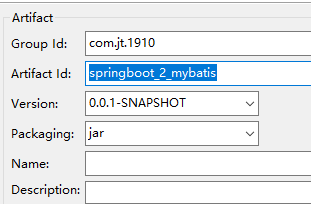




页面效果:



### 创建项目



### 编辑POM.xml文件

<!--导入公共的jar包文件 spring springmvc 并且定义2.2.4的相关联的jar包版本 防止jar包冲突. -->

<parent>

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

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

<version>2.2.4.RELEASE</version>

<relativePath /> <!-- lookup parent from repository -->

</parent>

<!--springBoot环境变量配置信息 -->

<properties>

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

<!--指定maven插件版本 -->

<maven-jar-plugin.version>3.1.1</maven-jar-plugin.version>

</properties>

<!--项目的依赖项 -->

<dependencies>

<!--SpringMVC 传统方法需要手动的配置 springboot程序自动化部署. -->

<dependency>

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

<!--springboot的启动器包 -->

<!--maven中的jar包具有依赖的传递性 -->

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

</dependency>

<!--springboot启动项 加载test测试 -->

<dependency>

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

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

<scope>test</scope>

<exclusions>

<exclusion>

<groupId>org.junit.vintage</groupId>

<artifactId>junit-vintage-engine</artifactId>

</exclusion>

</exclusions>

</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>

<!--开箱即用 -->

<!--支持热部署 -->

<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>

<version>5.1.32</version>

</dependency>

<!--springBoot数据库连接 -->

<dependency>

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

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

</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标签主要作用添加插件/配置 -->

<build>

<plugins>

<!--指定maven插件 负责构建maven项目 打包/部署/测试/文档生成 -->

<plugin>

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

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

</plugin>

</plugins>

</build>

### 编辑UserPOJO对象

//最好与数据库中的字段一致

@Data

@NoArgsConstructor

@AllArgsConstructor

@Accessors(chain=**true**)

**public** **class** User {

**private** Integer id;

**private** String name;

**private** Integer age;

**private** String sex;

}

### 编辑UserMapper

@Mapper //将接口交给Spring容器管理

**public** **interface** UserMapper {

//查询全部数据

List<User> findAll();

}

### 编辑UserMapper.xml配置

<!--

namespace: 当前mapper的唯一标识

mapper与接口包路径一致 -->

<mapper namespace=*"com.jt.mapper.UserMapper"*>

<!--编辑sql语法 sql标签的Id名称 应该和接口方法名称一致

resultType="包名.类名"

动态添加包路径.

-->

<select id=*"findAll"* resultType=*"User"*>

select \* from user

</select>

</mapper>

### 配置YML配置文件

server:

port: 8090

servlet:

context-path: / #项目发布路径

spring:

datasource:

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

#编辑Mybatis配置

mybatis:

type-aliases-package: com.jt.pojo #定义别名包

mapper-locations: classpath:/mybatis/mappers/\*.xml #加载mapper映射文件

#开启驼峰映射

configuration:

map-underscore-to-camel-case: true

### 驼峰规则映射

<!--开启驼峰规则映射

User user表

userId user\_id

userAge user\_age

userSex user\_sex

要求:字段名称和属性名称必然一致!!!!

解决方法: 开启驼峰规则映射

实现思路:

1.程序执行sql语句.

2.将结果集映射到对象中

字段user\_id~~~~去除"\_"线~~~~~之后首字母大写userId

3.将转化后的数据映射给user的属性.

注意事项:

1.一旦开启了驼峰规则映射,则必然按照要求执行

例子:

对象 字段

user\_id user\_id 不能映射

字段~~~user\_id~~~~userid~~~userId =!user\_id

-->

### 项目测试

@SpringBootTest

**class** Springboot1ApplicationTests {

@Autowired //动态注入

**private** UserMapper userMapper;

@Test

**public** **void** testFind() {

List<User> userList = userMapper.findAll();

System.***out***.println(userList);

}

}