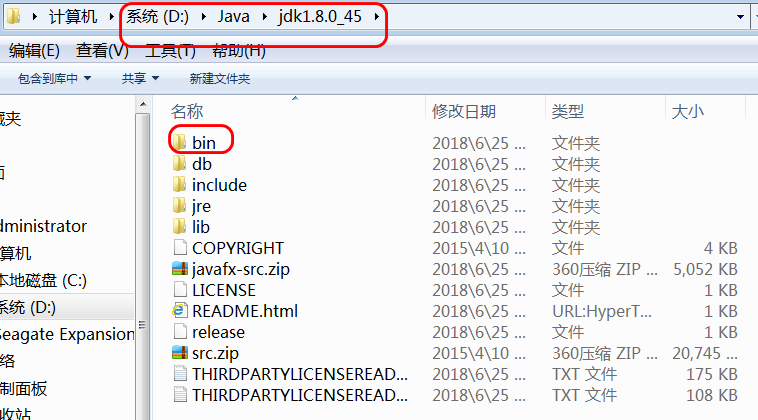
# SpringBoot

## 环境变量配置

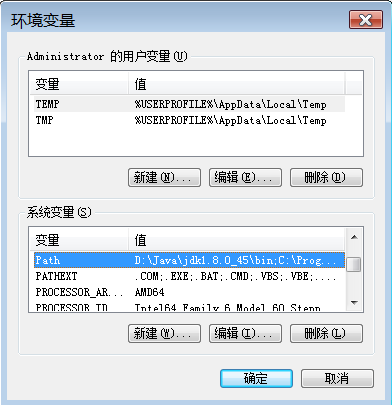
### JDK安装步骤

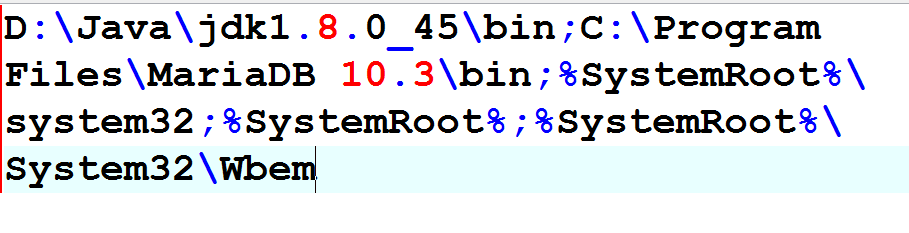
1.本机中有JDK文件



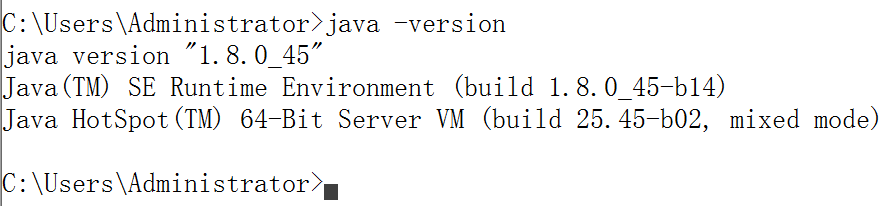
2.配置环境变量

着重检查path中的JDK配置有没有多余



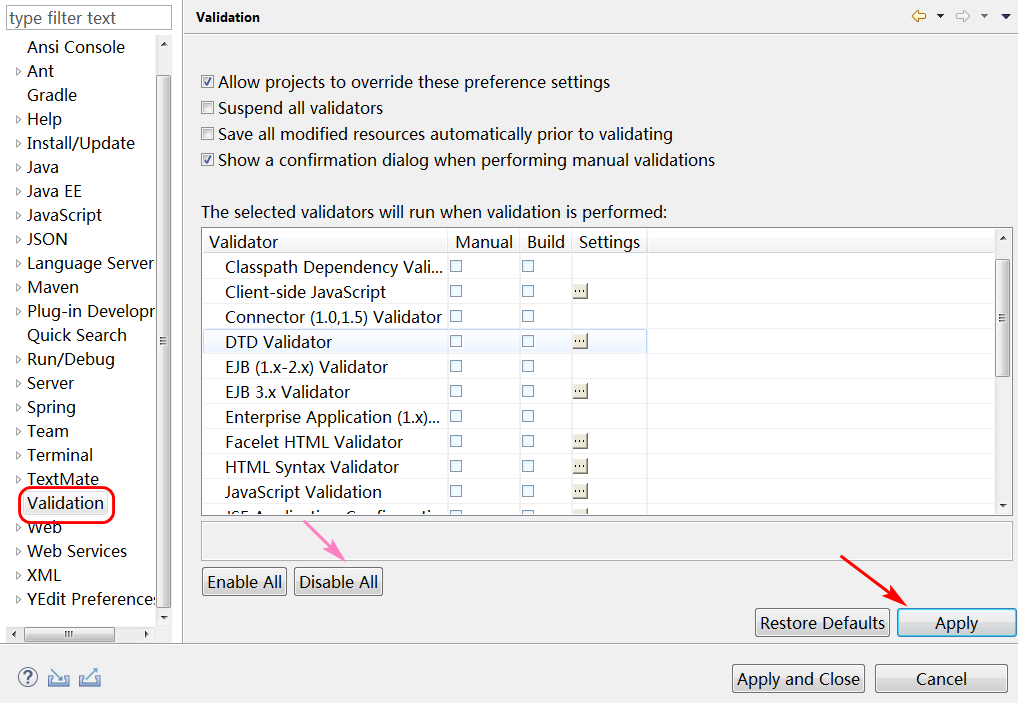


3.检查JDK安装是否正确

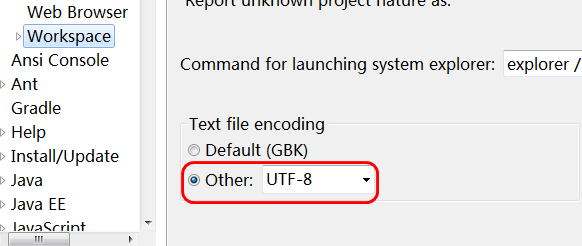


## STS配置

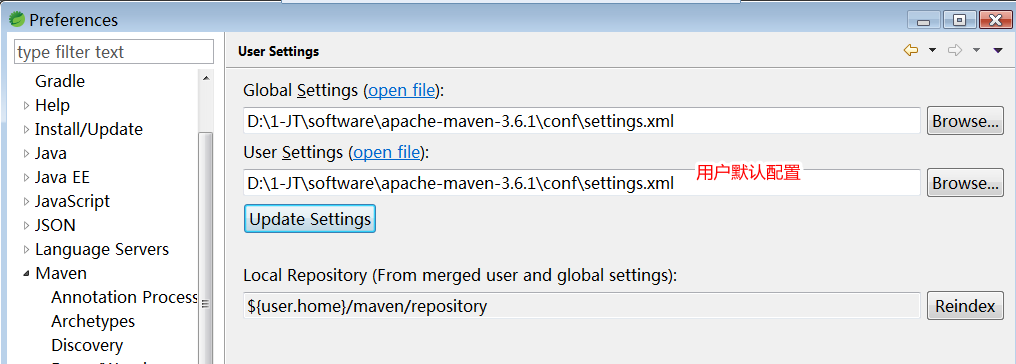
### 关闭STS校验



### 修改字符集编码

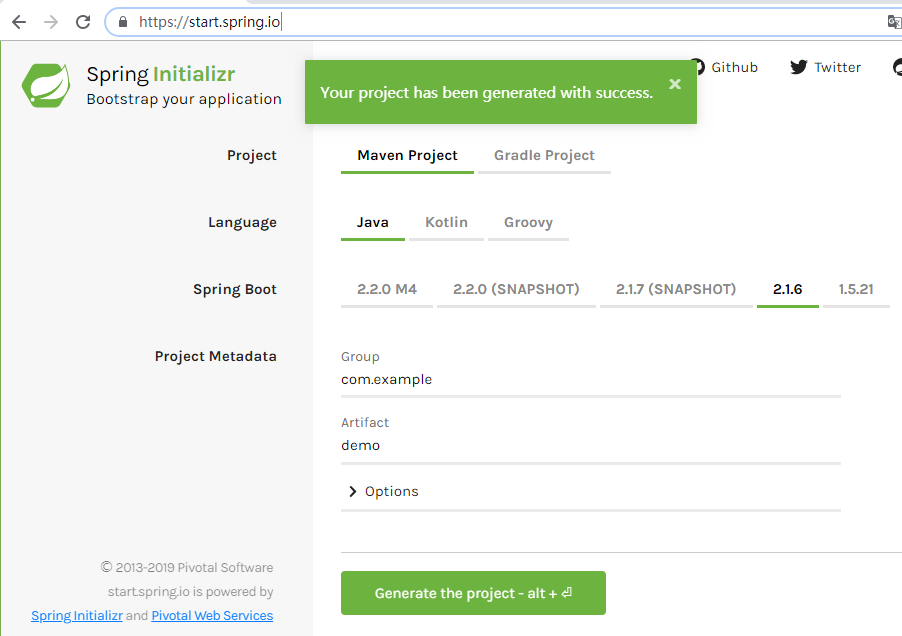


### 配置maven



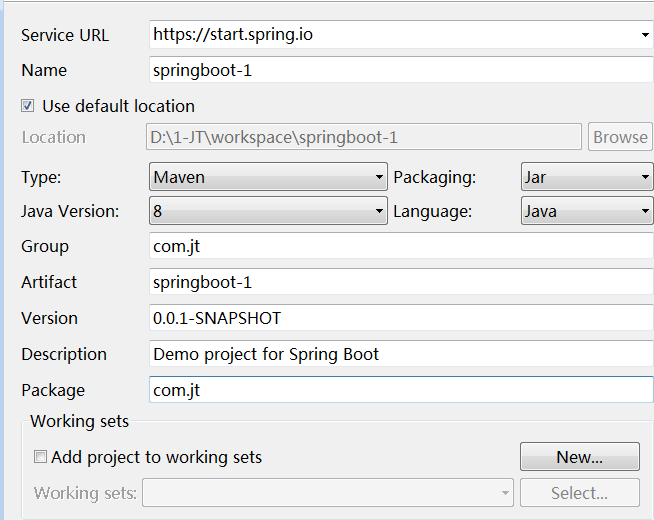
## SpringBoot入门案例

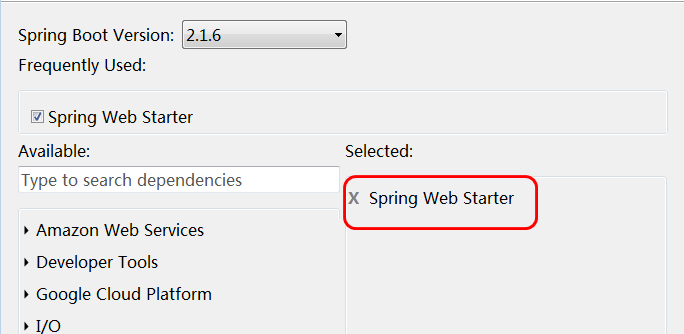
### 官网创建项目介绍



### STS创建项目

1.选择maven坐标,项目名称



2.选择依赖项  
 

## 关于springBoot pom.xml文件说明

### 坐标的作用

1. 可以将项目按照规定 打成jar/war包
2. 为项目的依赖提供了支持.

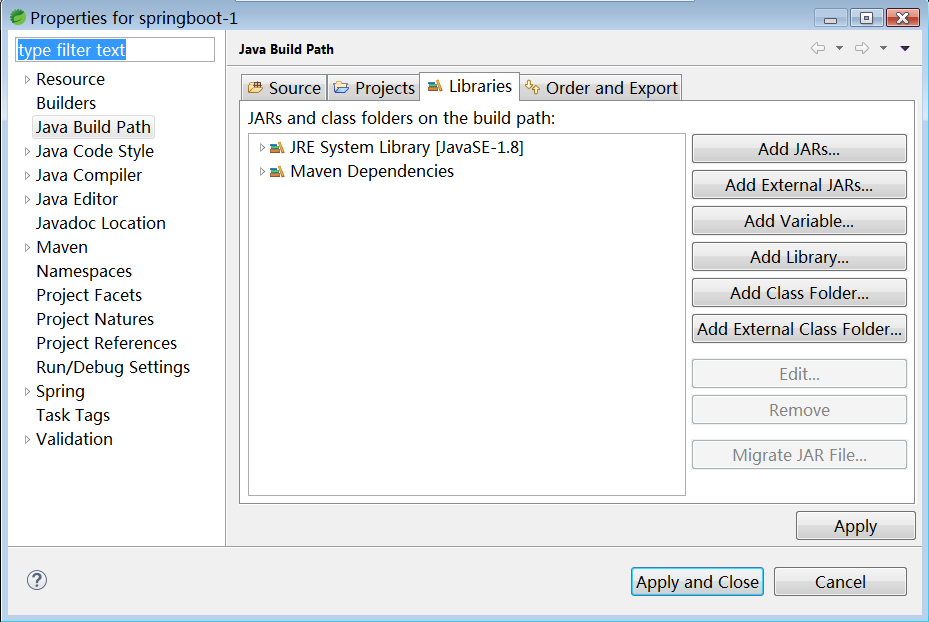


### 关于springBoot打包报错

项目执行时报错了.报错信息有 JRE/JDK.

问题原因

1. JDK不正确,开发环境应该使用JDK,但是自己配置的是JRE
2. STS引用的JDK不正确.

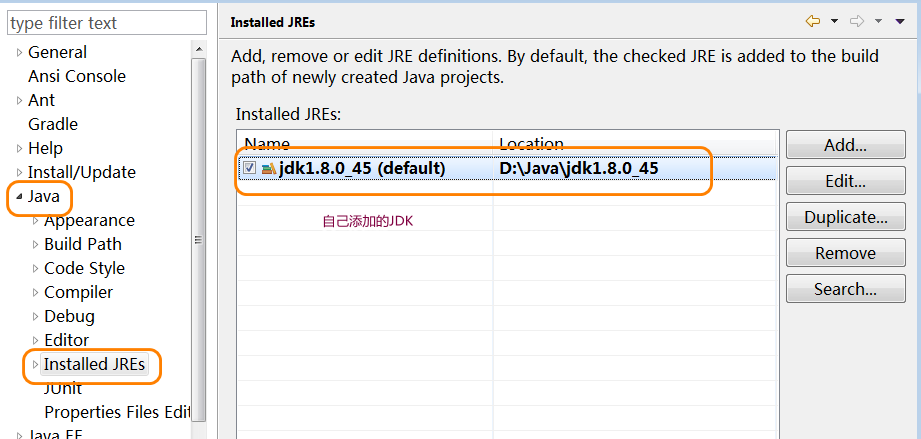


修改项目引入JDK路径



切换JDK路径

了解:如果需要可以修改IDE中配置的JDK.



### SpringBoot如何自动配置

说明:

1.SpringBoot程序会针对不同的框架开发不同的整合jar包形如:

<!--springBoot优势配置文件几乎不写 SpringMVC

springBoot针对不同的框架有不同的jar包的支持

如果见到spring-boot-starter-xxxx,springboot

已经将配置信息已经封装.

-->

<dependency>

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

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

</dependency>

2.SpringBoot主启动类中注解会自动添加依赖项

//1.包含了包扫描 扫描当前主启动类的包路径

//2.开启了自动配置过程

//@EnableAutoConfiguration 开启自动配置,如果pom.xml文件中添加了spring自动的配置jar包.则会在程序启动时自动加载实例化对象

@SpringBootApplication

**public** **class** Springboot1Application {

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

SpringApplication.*run*(Springboot1Application.**class**, args);

}

}

## 关于配置文件说明

### 关于properties的说明

说明:

1. properties中要求不能编辑中文.
2. properties文件是按照行的形式进行编辑.不能路径复用

### YML文件语法

1. yml配置文件是有层级关系的.注意缩进
2. yml配置文件中key/value中间必须添加 "**:** " 空格

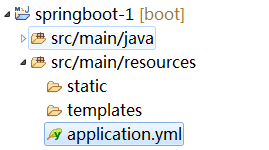
server:

port: 8091

servlet:

context-path: / #默认的发布路径就是 /

名称必须为application.yml



## SpringBoot属性赋值方式

### 业务需求

说明:当spring整合第三方架构时,有些类中要求实例化对象时要求赋值某些属性.这时需要利用Spring容器的方式为属性赋值.

如何处理耦合性低??

### @Value赋值

#### 利用YML形式为属性赋值

server:

port: 8091

servlet:

context-path: / #默认的发布路径就是 /

#案例1.为属性赋值 YML中可以写中文,但是开发时一般不用

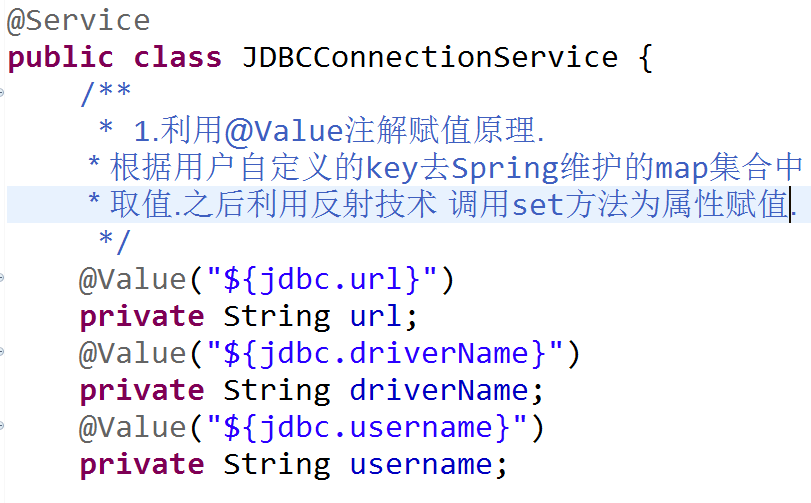
jdbc:

url: mysql的url

driverName: mysql数据库

username: 用户名A

#### 利用@Value为属性赋值



#### @RestController介绍

@RestController注解就是@Controller+@ResponseBody的组合注解

//@Controller +@ResponseBody

@RestController

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

#### 关于对象JSON转化问题

说明:当对象转化为json数据时,调用对象的getXXX().所以get/set方法必须添加.

**public** String getUrl() {

**return** url;

}

**public** **void** setUrl(String url) {

**this**.url = url;

}

**public** String getDriverName() {

**return** driverName;

}

**public** **void** setDriverName(String driverName) {

**this**.driverName = driverName;

}

**public** String getUsername() {

**return** username;

}

**public** **void** setUsername(String username) {

**this**.username = username;

}

页面效果:



### 批量赋值

#### 业务需求

有些框架中要求实例化对象时,需要为N多个属性赋值.如果这时依然采用@ｖａｌｕｅ赋值形式　太繁琐，所以使用批量赋值的形式操作．

#### 引入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YML配置文件

#案例2 为属性批量赋值

jdbcb:

jdbcName: mysql

jdbcDriver: mysql驱动

#### 批量为属性赋值

使用注解 @ConfigurationProperties(prefix="jdbcb"),额外记住添加set方法.

@Service

@ConfigurationProperties(prefix="jdbcb")

**public** **class** JDBCServiceB {

//规定:如果采用批量赋值的形式,

//一定要添加set方法.

**private** String jdbcName;

**private** String jdbcDriver;

**public** String getJdbcName() {

**return** jdbcName;

}

**public** **void** setJdbcName(String jdbcName) {

**this**.jdbcName = jdbcName;

}

**public** String getJdbcDriver() {

**return** jdbcDriver;

}

**public** **void** setJdbcDriver(String jdbcDriver) {

**this**.jdbcDriver = jdbcDriver;

}

}

#### 编辑测试类

@Autowired

**private** JDBCServiceB jdbcServiceB;

//2.测试批量赋值操作

@RequestMapping("/jdbcTestB")

**public** JDBCServiceB jdbcTestB() {

**return** jdbcServiceB;

}

#### 测试效果



### 指定配置文件为属性赋值.

#### 业务说明

YML文件是SpringBoot程序中最为重要的核心配置文件.一般用来整合第三方框架.

如果将依赖的配置信息也写入YML中,则文件冗余.可读性差.

所以一般采用properties的方式引用.

#### 自定义properties文件

#key=value

jdbcc.jdbcName=mysqlName

jdbcc.jdbcDriver=mysqlDriver

#### 对象赋值操作

说明:指定配置文件为属性赋值.

@Service

//自定义配置文件交给spring容器管理

**@PropertySource(value="classpath:/properties/jdbc.properties")**

**public** **class** JDBCServiceC {

@Value("${jdbcc.jdbcName}")

**private** String jdbcName;

@Value("${jdbcc.jdbcDriver}")

**private** String jdbcDriver;

**public** String getJdbcName() {

**return** jdbcName;

}

**public** **void** setJdbcName(String jdbcName) {

**this**.jdbcName = jdbcName;

}

**public** String getJdbcDriver() {

**return** jdbcDriver;

}

**public** **void** setJdbcDriver(String jdbcDriver) {

**this**.jdbcDriver = jdbcDriver;

}

}

#### 编辑测试类

@Autowired

**private** JDBCServiceC jdbcServiceC;

//3.测试自定义配置文件

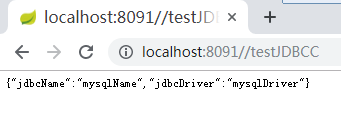
@RequestMapping("/testJDBCC")

**public** JDBCServiceC testJDBCC() {

**return** jdbcServiceC;

}

#### 测试效果



## YML切换环境配置

### 业务需求

基础:

开发环境: 程序编辑时使用的环境 环境都是临时搭建的为了开发方便.

测试环境: 一般的测试环境包含开发环境的配置,同时也需要进行压测

生产环境: 真实的业务处理的环境.

说明:由于上述的环境不同,如果每次调试时都需要修改大量的配置文件,所以使用环境切换功能

### YML配置写法

#语法 使用"---"进行环境分割

#程序启动时 指定环境进行加载

spring:

profiles:

active: prod

--- #第一个环境dev

spring:

profiles: dev #为该环境指定一个名称

server:

port: 8080

servlet:

context-path: /

---

spring:

profiles: test #为该环境指定一个名称

server:

port: 8090

servlet:

context-path: /test

---

spring:

profiles: prod #为该环境指定一个名称

server:

port: 9000

servlet:

context-path: /prod

#### 测试说明

修改YMl文件中活跃的环境测试,则可以切换环境.以后配置文件无需修改.

简化程序操作.

#程序启动时 指定环境进行加载

spring:

profiles:

active: prod

## 热部署

### 添加jar包

<!--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lombok插件

### 添加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>

### 作用

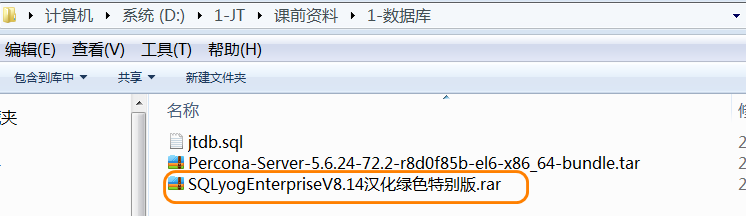
自动为pojo对象生成get/set方法等操作.

## SpringBoot整合Mybatis

### 导入数据库

#### 安装数据库操作工具

说明:将课前资料中数据库文件导入本地





#### 利用工具导入数据库



### 添加mybatis依赖包

<!--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>

### 编辑整合YMl配置文件

server:

port: 8090

servlet:

context-path: /

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

#如果数据库密码以0开头 使用:""包裹

mybatis:

#定义别名包 简化查询resultType配置

type-aliases-package: com.jt.pojo

#Mybatis的映射文件路径

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

#开启驼峰映射 字段 属性 对应 去除字段中的"\_"之后首字母大写

configuration:

map-underscore-to-camel-case: true

### 业务代码实现

#### 编辑POJO对象

使用lombok加载pojo

@Data //自动添加get/set方法

@Accessors(chain = **true**) //开启链式加载

@AllArgsConstructor //添加全部参数的构造

@NoArgsConstructor //添加无参构造

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

**private** Integer id;

**private** String name;

**private** Integer age;

**private** String sex;

}

#### 编辑Mapper接口

1.定义接口

//接口的定义 没有实现类

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

//编辑接口方法 查询全部数据

List<User> findAll();

}

2.定义mapper映射

<?xml version=*"1.0"* encoding=*"UTF-8"* ?>

<!DOCTYPE mapper

PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"

"http://mybatis.org/dtd/mybatis-3-mapper.dtd">

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

<!--基于别名包完成自动的映射 -->

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

select \* from user

</select>

</mapper>

### 编辑Service

@Service

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

//spring容器管理对象 Spring创建代理对象

@Autowired

**private** UserMapper userMapper;

@Override

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

**return** userMapper.findAll();

}

}

### 编辑Controller

@RestController

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

@Autowired

**private** UserService userService;

@RequestMapping("/findAll")

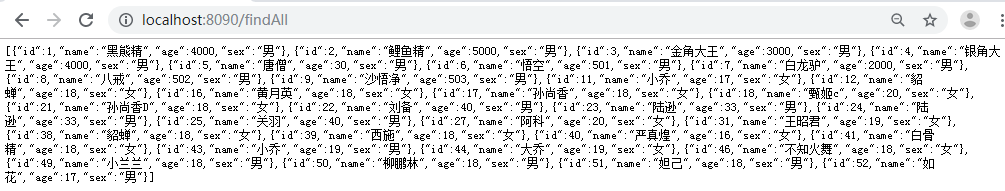
**public** List<User> findAll(){

**return** userService.findAll();

}

}

### 页面效果



## 关于项目问题

### Maven项目不更新

问题描述:

有时POM.xml文件报错,因为jar包下载不完全导致的.

解决方案:

让pom.xml文件重新下载即可.

万能方案:

