# SpringBoot+Mybatis+Hibernate

## 一、准备开发环境

1、JDK : JDK 1.8.x及以上版本

2、Maven : Maven 3.x或更高版本

3、IDE : 选择你比较喜欢的一个代码编辑器。本文使用的是IntelliJ IDEA

4、Databases : 选择一个你熟练使用的数据库系统。在本文中使用的是MySQL 5.7.18

## 二、搭建项目

### 2.1引入依赖

为了快速构建项目，笔者采用Spring Boot来构建项目，同时使用加入Spring Data JPA和Mybatis两个ORM框架的依赖包。在此需要特别说明，Hibernate是一个JPA标准的实现，尔Spring Data JPA是一个JPA数据访问抽象，通过Spring Data JPA，可以轻松使用Hibernate框架。

你可以通过Spring Initializer来初始化项目，也可以通过IDEA自带的Spring Initializer功能构建项目，项目构建完成之后，pom.xml文件中的配置如下（包含但不限于文中给出的依赖项）：

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. **<parent>**
6. **<groupId>**org.springframework.boot**</groupId>**
7. **<artifactId>**spring-boot-starter-parent**</artifactId>**
8. **<version>**2.2.2.RELEASE**</version>**
9. **<relativePath/>** <!-- lookup parent from repository -->
10. **</parent>**
11. **<groupId>**com.tellhow**</groupId>**
12. **<artifactId>**demo2**</artifactId>**
13. **<version>**0.0.1-SNAPSHOT**</version>**
14. **<name>**demo2**</name>**
15. **<description>**Demo project for Spring Boot**</description>**
17. **<properties>**
18. **<java.version>**1.8**</java.version>**
19. **</properties>**
21. **<dependencies>**
22. **<dependency>**
23. **<groupId>**org.springframework.boot**</groupId>**
24. **<artifactId>**spring-boot-starter-web**</artifactId>**
25. **</dependency>**
26. **<dependency>**
27. **<groupId>**org.mybatis.spring.boot**</groupId>**
28. **<artifactId>**mybatis-spring-boot-starter**</artifactId>**
29. **<version>**1.3.2**</version>**
30. **</dependency>**
31. **<dependency>**
32. **<groupId>**mysql**</groupId>**
33. **<artifactId>**mysql-connector-java**</artifactId>**
34. **<scope>**runtime**</scope>**
35. **</dependency>**
36. **<dependency>**
37. **<groupId>**org.springframework.boot**</groupId>**
38. **<artifactId>**spring-boot-starter-test**</artifactId>**
39. **<scope>**test**</scope>**
40. **</dependency>**
41. **<dependency>**
42. **<groupId>**org.apache.commons**</groupId>**
43. **<artifactId>**commons-lang3**</artifactId>**
44. **<version>**3.4**</version>**
45. **</dependency>**
47. **<dependency>**
48. **<groupId>**com.fasterxml.jackson.core**</groupId>**
49. **<artifactId>**jackson-core**</artifactId>**
50. **</dependency>**
51. **<dependency>**
52. **<groupId>**com.fasterxml.jackson.core**</groupId>**
53. **<artifactId>**jackson-databind**</artifactId>**
54. **</dependency>**
55. **<dependency>**
56. **<groupId>**com.fasterxml.jackson.datatype**</groupId>**
57. **<artifactId>**jackson-datatype-joda**</artifactId>**
58. **</dependency>**
59. **<dependency>**
60. **<groupId>**com.fasterxml.jackson.module**</groupId>**
61. **<artifactId>**jackson-module-parameter-names**</artifactId>**
62. **</dependency>**
63. <!-- 分页插件 -->
64. **<dependency>**
65. **<groupId>**com.github.pagehelper**</groupId>**
66. **<artifactId>**pagehelper-spring-boot-starter**</artifactId>**
67. **<version>**1.2.5**</version>**
68. **</dependency>**
69. <!-- alibaba的druid数据库连接池 -->
70. **<dependency>**
71. **<groupId>**com.alibaba**</groupId>**
72. **<artifactId>**druid-spring-boot-starter**</artifactId>**
73. **<version>**1.1.9**</version>**
74. **</dependency>**
76. <!--hibernate-->
77. **<dependency>**
78. **<groupId>**org.springframework.boot**</groupId>**
79. **<artifactId>**spring-boot-starter-data-jpa**</artifactId>**
80. **</dependency>**
81. **</dependencies>**
83. **<build>**
84. **<plugins>**
85. **<plugin>**
86. **<groupId>**org.springframework.boot**</groupId>**
87. **<artifactId>**spring-boot-maven-plugin**</artifactId>**
88. **</plugin>**
89. **</plugins>**
90. **</build>**
92. **</project>**

### 2.2配置Hibernate和Mybatis

只需要在application.yml或者application.properties配置文件中加入几行代码，就可以完成两个框架的整合。以application.yml配置文件为例，配置代码如下：

1. server:
2. port: 8081
3. spring:
4. datasource:
5. type: com.alibaba.druid.pool.DruidDataSource
6. url: jdbc:mysql://localhost:3306/myTest?serverTimezone=UTC&characterEncoding=utf-8
7. username: root
8. password: 123456
9. jpa:
10. database: mysql
11. show-sql: false
12. generate-ddl: false
13. hibernate:
14. ddl-auto: update
15. database-platform: org.hibernate.dialect.MySQL5Dialect
16. mybatis:
17. type-aliases-package: com.tellhow.demo2.entity
18. mapper-locations: classpath:mapper/\*.xml

这是一种较为简单的整合方式。Hibernate和Mybatis共用一个数据源.

yml的常用语法规则如下：

1）大小写敏感

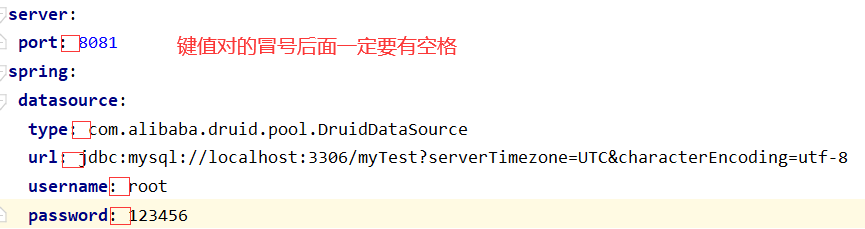
2）使用缩进表示层级关系

3）缩进时不允许使用Tab键，只允许使用空格。

4）缩进的空格数目不重要，只要相同层级的元素左侧对齐即可

5）使用 # 来表示注释内容

6）使用 :+ 空格 表示单个键值对



### 2.3定义实体类User.java

1. **package** com.tellhow.demo2.entity;
3. **import** lombok.AllArgsConstructor;
4. **import** lombok.Data;
5. **import** lombok.NoArgsConstructor;
6. **import** javax.persistence.\*;
7. **import** java.io.Serializable;
9. @Entity//表明该类 (User) 为一个实体类
10. @Data //@Data注解可以自动生成类属性的getter、setter和toString方法
11. @NoArgsConstructor //自动生成无参数构造函数
12. @AllArgsConstructor //自动生成全参数构造函数
13. @Table(name ="t\_user")//当实体类与其映射的数据库表名不同名时需要使用 @Table注解说明，name用于指明数据库的表名，@Table注解还有两个选项 catalog 和 schema 用于设置表所属的数据库目录或模式，通常为数据库名
14. **public** **class** User **implements** Serializable {
16. **private** **static** **final** **long** serialVersionUID = 7858262691294481727L;
18. @Id //用于声明一个实体类的属性映射为数据库的主键列
19. @GeneratedValue(strategy = GenerationType.IDENTITY) //用于标注主键的生成策略
20. //在javax.persistence.GenerationType中定义了以下几种可供选择的策略：
21. //–IDENTITY：采用数据库ID自增长的方式来自增主键字段，Oracle 不支持这种方式；
22. //–AUTO： JPA自动选择合适的策略，是默认选项；
23. //–SEQUENCE：通过序列产生主键，通过@SequenceGenerator 注解指定序列名，MySql不支持这种方式
24. //–TABLE：通过表产生主键，框架借由表模拟序列产生主键，使用该策略可以使应用更易于数据库移植。
26. @Column(name = "id")
27. **private** Integer id;
28. @Column(name = "name")
29. **private** String name;
30. @Column(name = "password")
31. **private** String password;
32. @Column(name = "phone")
33. **private** String phone;
34. }

### 2.4定义数据持久化接口

本文使用Spring Data JPA来完成写操作，如新增、修改、删除；使用Mybatis来完成读操作，如根据用户ID查询、查询所有的用户等。Spring Data JPA和MyBatis的持久化接口都位于com.ramostear.hm.orm包下，Spring Data JPA的持久化接口相对比较简单，之间继承JpaRepository类即可，代码如下：

1. **package** com.tellhow.demo2.mapper;
3. **import** com.tellhow.demo2.entity.User;
4. **import** org.springframework.data.jpa.repository.JpaRepository;
6. **public** **interface** UserRepository **extends** JpaRepository<User, Integer> {
8. }

说明：因为JPA只负责写操作，所以直接继承并使用JpaRepository提供的API即可，不需要额外的定义其他的接口方法。

Mybatis的映射接口，定义了一个方法：查询所有的用户信息。代码如下：

1. **package** com.tellhow.demo2.mapper;
3. **import** com.tellhow.demo2.entity.User;
4. **import** org.apache.ibatis.annotations.Mapper;
5. **import** org.springframework.stereotype.Component;
6. **import** java.util.List;
8. @Component //@Component注解标注此接口后，Spring会自动扫描并配置此类
9. @Mapper //把这个mapper的DAO交由Spring进行管理
10. **public** **interface** UserMapper {
11. List<User> selectUsers();
12. }

定义完Mybatis 映射接口后，需要提供一个进行数据库查询的xml配置文件。该文件位于resources/mapper文件夹中，UserMapper.xml完整代码如下：

1. **<?xml** version="1.0" encoding="UTF-8" **?>**
2. <!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "http://mybatis.org/dtd/mybatis-3-mapper.dtd" **>**
3. **<mapper** namespace="com.tellhow.demo2.mapper.UserMapper" **>**
5. **<select** id="selectUsers" resultType="com.tellhow.demo2.entity.User"**>**
6. SELECT \* FROM t\_user
7. **</select>**
9. **</mapper>**

### 2.5定义UserService

在UserService接口中，提供两个方法：保存用户信息、查询所有的用户。UserService接口代码如下：

1. **package** com.tellhow.demo2.service;
3. **import** com.tellhow.demo2.entity.User;
4. **import** java.util.List;
6. **public** **interface** UserService {
7. User addUser(User user);
8. List<User> findAllUser();
9. }

在UserService接口的实现类中，需要同时注入UserRepository和UserMapper两个依赖。我们使用构造函数的方式来注入这两个依赖。代码如下：

*说明：*  
*在本次教程中，为了编码IDEA报警告，所有的依赖注入都采用构造函数的方式注入相关的依赖。*

1. **package** com.tellhow.demo2.service.impl;
3. **import** com.tellhow.demo2.entity.User;
4. **import** com.tellhow.demo2.mapper.UserMapper;
5. **import** com.tellhow.demo2.mapper.UserRepository;
6. **import** com.tellhow.demo2.service.UserService;
7. **import** org.springframework.beans.factory.annotation.Autowired;
8. **import** org.springframework.stereotype.Service;
9. **import** org.springframework.transaction.annotation.Transactional;
11. **import** java.util.List;
13. @Service("userService") //声明该类是一个服务提供类，且设置了该类被Spring初始化时Bean对象的名称为“userService”
14. **public** **class** UserServiceImpl **implements** UserService {
15. **private** UserMapper userMapper;
16. **private** UserRepository userRepository;
18. @Autowired
19. UserServiceImpl(UserRepository userRepository, UserMapper userMapper) {
20. **this**.userMapper = userMapper;
21. **this**.userRepository = userRepository;
22. }
24. @Override
25. @Transactional //用于设置每个方法的事务控制方式
26. **public** User addUser(User user) {
27. **return** userRepository.save(user);
28. }
30. @Override
31. @Transactional(readOnly = **true**)
32. **public** List<User> findAllUser() {
33. List<User> view = userMapper.selectUsers();
34. **return** userMapper.selectUsers();
35. }
36. }

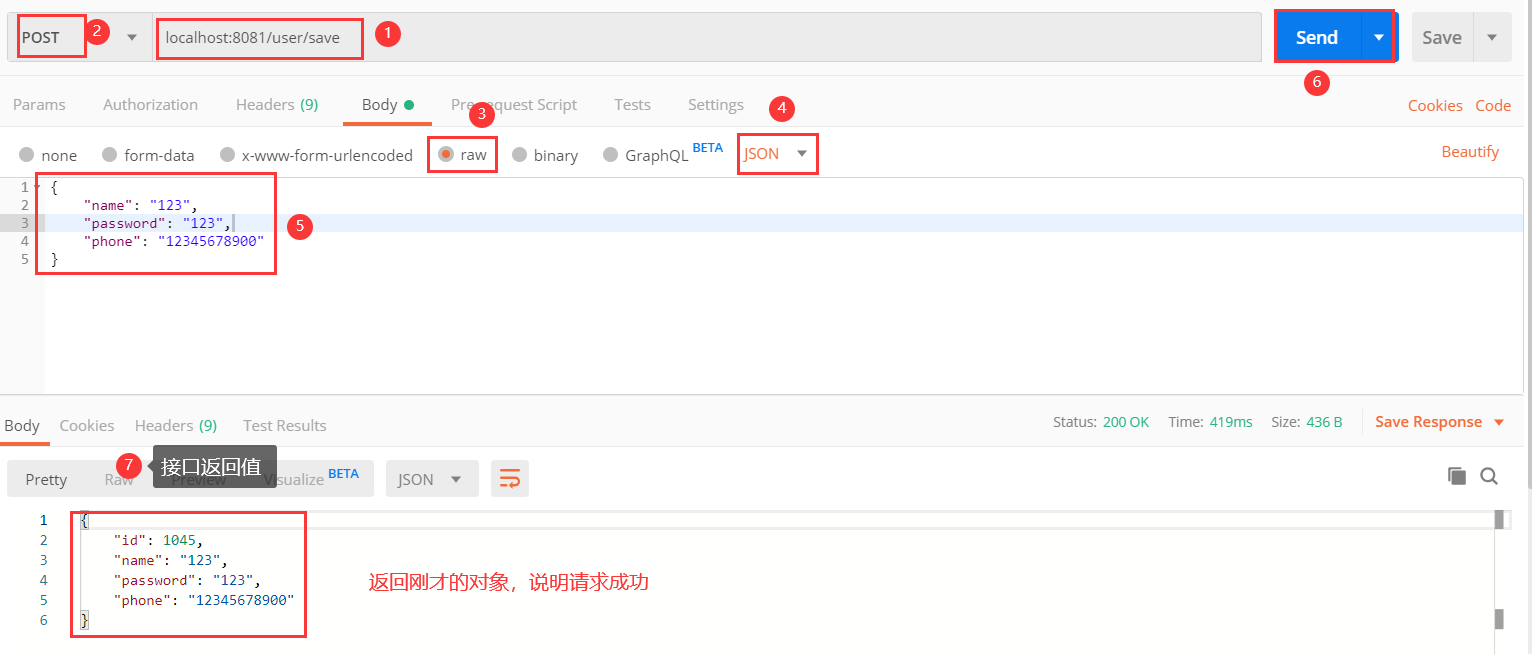
### 2.6定义控制器

最后，提供一个控制器，用于处理客户端的相关请求。在控制器中，提供了两个请求处理方法，分别处理客户端新增用户和查询所有用户的请求。控制器代码如下：

## 三、测试

测试工具：Postman

首先测试 POST <http://localhost:8081/users>/save ，验证Hibernate是否能够成功将用户信息持久化。打开Postman工具，在地址栏输入<http://localhost:8081/users>[/save 请求地址，请求方式选择POST,在Body栏输入如下的信息：](http://localhost/users%E8%AF%B7%E6%B1%82%E5%9C%B0%E5%9D%80%EF%BC%8C%E8%AF%B7%E6%B1%82%E6%96%B9%E5%BC%8F%E9%80%89%E6%8B%A9POST,%E5%9C%A8Body%E6%A0%8F%E8%BE%93%E5%85%A5%E5%A6%82%E4%B8%8B%E7%9A%84%E4%BF%A1%E6%81%AF%EF%BC%9A" \t "_blank)



然后测试 GET<http://localhost:8081/users>/findall ，验证Mybatis是否能够成功查询用户信息。打开Postman工具，在地址栏输入<http://localhost:8081/users>/findall 请求地址，请求方式选择GET：

