本节讲述两种Java数据库操作的方式，一是比较原生繁琐的JDBC方式，二是现在比较简便主流的Mybatis方式。

由于Mybatis在做数据库操作时更加简便，成为了比较主流之一的方式，值得我们关注学习。

但也需要了解原生的JDBC方式，毕竟这是基础，主流的框架也大多由此衍生。

# 一、Java数据库操作之JDBC

## 1、简介

Java数据库连接，（Java Database Connectivity，简称JDBC）是Java语言中用来规范客户端程序如何来访问数据库的应用程序接口(位于jdk的java.sql中)。我们通常说的JDBC是面向关系型数据库的，提供了诸如查询、更新、删除、增加数据库中数据的方法。在使用时候需要导入具体的jar包，不同数据库需要导入的jar包不同。

JDBC与MySQL进行连接交互，通常为以下6个流程：

- 1：注册驱动 (仅仅做一次)
- 2：建立连接(Connection)
- 3：创建运行SQL的语句(Statement)
- 4：运行语句
- 5：处理运行结果(ResultSet)
- 6：释放资源

JDBC中常用的API，推荐看这里：`https://book.itheima.net/course/1265899443273850881/1272721284588904449/1272772917125455877`

## 2、代码示例

下面通过一个简单的代码示例来理解JDBC链接Mysql，并查询数据的过程。

JDBC与Mysql交互需要在本机安装Mysql，我选择的是PHPStudy自带的Mysql，版本为`MySQL 5.7.26`，点击启动，即可使用，如下图所示：


①、首先，需要创建练习所需的数据库，同时创建数据表，以及添加对应的数据，分别执行以下三个部分的代码，最终执行结果如下图所示：

```
CREATE DATABASE jdbcdemo;
```

```
USE jdbcdemo;
CREATE TABLE `user` (
  `id` int(10) unsigned NOT NULL AUTO_INCREMENT COMMENT '唯一ID',
  `username` varchar(25) NOT NULL COMMENT '用户名',
  `password` varchar(25) NOT NULL COMMENT '密码',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8;
```

```
INSERT INTO user(id,username,password) VALUES (1, "power7089",
"power7089");
INSERT INTO user(id,username,password) VALUES (2, "root", "root");
INSERT INTO user(id,username,password) VALUES (3, "admin",
"admin0123");
```

```
GoodLuckToday$$$ mysql -u root -p 链接数据库
Enter password: ****
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 7
Server version: 5.7.26 MySQL Community Server (GPL)

Copyright (c) 2000, 2019, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> CREATE DATABASE jdbcdemo;      创建数据库
Query OK, 1 row affected (0.00 sec)

mysql> USE jdbcdemo; 切换使用数据库
Database changed
mysql> CREATE TABLE `user` (
    ->   `id` int(10) unsigned NOT NULL AUTO_INCREMENT COMMENT '唯一ID',
    ->   `username` varchar(25) NOT NULL COMMENT '用户名',
    ->   `password` varchar(25) NOT NULL COMMENT '密码',
    ->   PRIMARY KEY (`id`)
    -> ) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8;
Query OK, 0 rows affected (0.02 sec)

mysql> INSERT INTO user(id,username,password) VALUES (1, "power7089", "power7089");
Query OK, 1 row affected (0.01 sec)

mysql> INSERT INTO user(id,username,password) VALUES (2, "root", "root");
Query OK, 1 row affected (0.01 sec)

mysql> INSERT INTO user(id,username,password) VALUES (3, "admin", "admin@123");
Query OK, 1 row affected (0.00 sec)
```

创建名为user的数据表

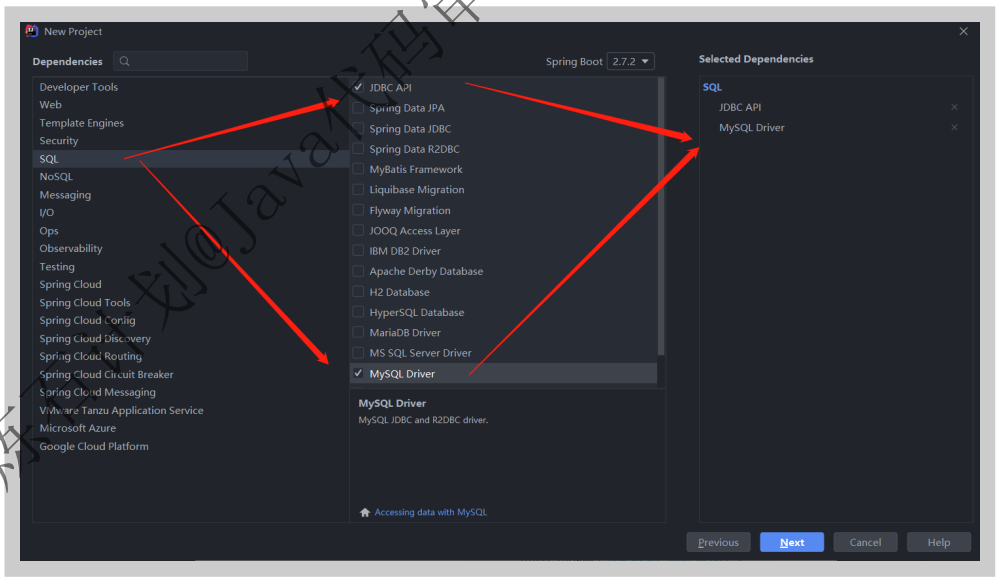
向user数据表中插入数据

②、下面创建一个项目工程，名为 `jdbcdemo` ，打开IDEA，选择 `Create New Porject` 。

③、左侧选择 `Spring Initializr` ，配置默认即可，点击Next。

④、在 `Spring Initializr Project Settings` 页面，将 `Java Version设置为8` ，其他配置项默认即可，点击Next。

④、在依赖选项界面，我们选择 `SQL -> JDBC API` ， `SQL -> Mysql Driver` ，注意，一共需要勾选两个依赖。如下图所示：

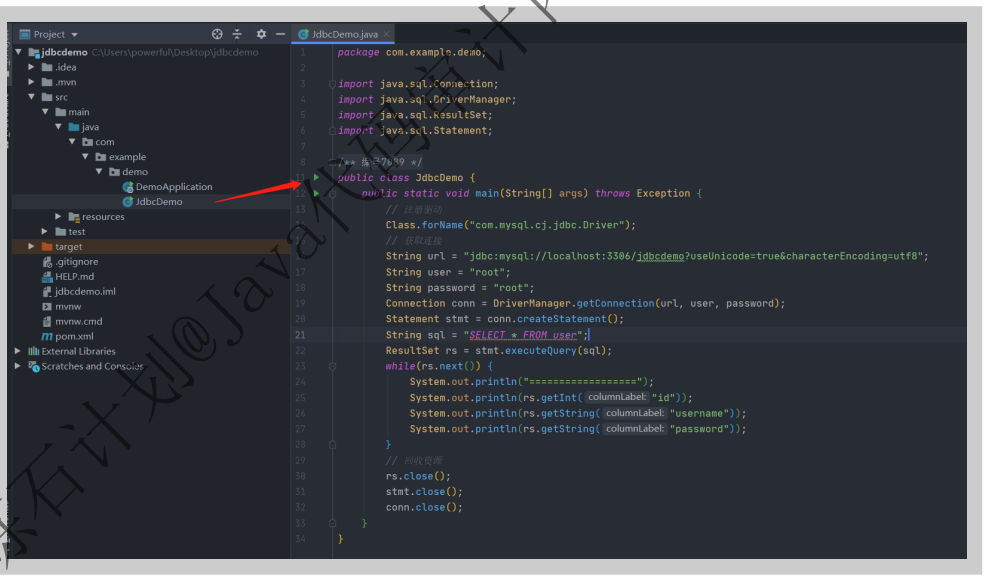



⑤、点击Next，起个项目名称为 `jdbcdemo` ，最后点击Finish。

⑥、在 `src.main.java.com.example.demo` 的文件下新建一个名为 `JdbcDemo` 的Java Class，并键入以下代码，如下图所示：

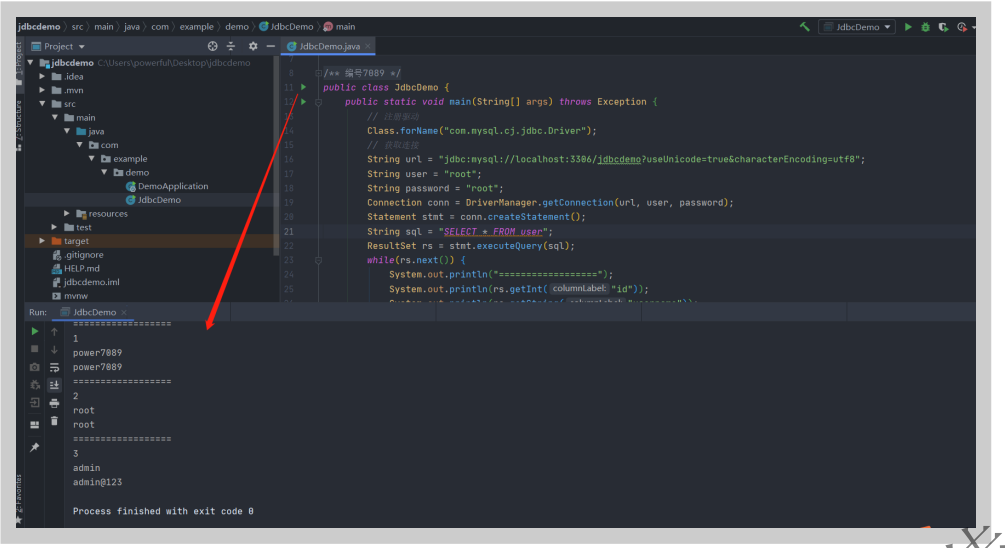
```
package com.example.demo;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;
```

```
public class JdbcDemo {
    public static void main(String[] args) throws Exception {
        // 注册驱动
        Class.forName("com.mysql.cj.jdbc.Driver");
        // 获取连接
        String url = "jdbc:mysql://localhost:3306/jdbcdemo?
useUnicode=true&characterEncoding=utf8";
        String user = "root";
        String password = "root";
        Connection conn = DriverManager.getConnection(url, user,
password);
        Statement stmt = conn.createStatement();
        String sql = "SELECT * FROM user";
        ResultSet rs = stmt.executeQuery(sql);
        while(rs.next()) {
            System.out.println("==================");
            System.out.println(rs.getInt("id"));
            System.out.println(rs.getString("username"));
            System.out.println(rs.getString("password"));
        }
        // 回收资源
        rs.close();
        stmt.close();
        conn.close();
    }
}
```






⑦、点击运行 `JdbcDemo` 类，观察运行响应结果，可以看到正确连接到数据库，并查询回来了所需数据，如下图所示：




一个简单的JDBC链接Mysql并查询数据的示例代码。

# 二、Java数据库操作之Mybatis

## 1、简述

MyBatis 是一款优秀的持久层框架，它支持自定义 SQL、存储过程以及高级映射。MyBatis 免除了几乎所有的 JDBC 代码以及设置参数和获取结果集的工作。MyBatis 可以通过简单的 XML 或注解来配置和映射原始类型、接口和 Java POJO（Plain Old Java Objects，普通老式 Java 对象）为数据库中的记录。

Mybatis中文文档：`https://mybatis.org/mybatis-3/zh/index.html`

## 2、代码示例

下面，我们通过SpringBoot整合Mybatis，来模拟一个增删改查的场景。

务必按顺序跟着下面每一个步骤进行练习。

### 2.1、创建项目工程

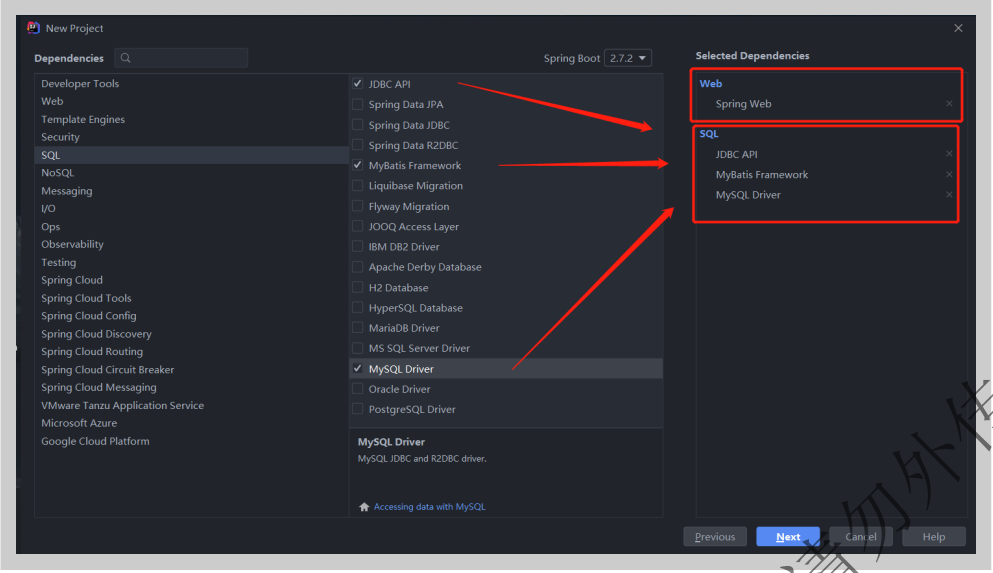
先创建一个名为`mybatis-springboot`的工程项目，用于后面编写示例代码。

①、打开IDEA，选择`Create New Porject`。

②、左侧选择`Spring Initializr`，配置默认即可，点击Next。

③、在`Spring Initializr Project Settings`页面，将`Java Version设置为8`，其他配置项默认即可，点击Next。

④、在依赖选项界面，我们选择 `web -> Spring Web` ， `SQL -> JDBC API` ， `SQL -> Mybatis Framework` ， `SQL -> Mysql Driver` ，注意，一共需要勾选四个依赖。如下图所示：




⑤、点击Next，这一步给项目起个名字，就叫 `mybatis-springboot` 吧。其他默认即可。最后点击Finish。

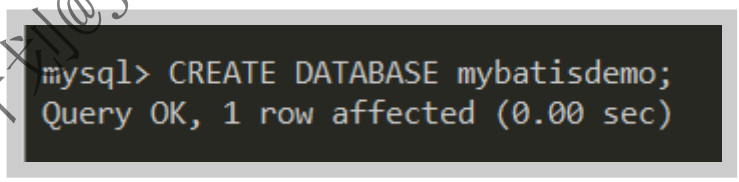
## 2.2、创建数据库

在编写代码之前，我们需要对项目做一些前置工作。

首先需要创建一个数据库并添加一些数据。我使用的是PHPStudy下自带的Mysql，版本为 `Mysql 5.7.26` 。

①、启动Mysql后，我使用的是命令行方式进入Mysql。创建一个名为 `mybatisdemo` 的数据库，如下图所示：

```
CREATE DATABASE mybatisdemo;
```




②、先切换使用 `mybatisdemo` 数据库。然后创建 `user` 数据表，如下图所示：

```
USE mybatisdemo;
CREATE TABLE `user` (
  `id` int(10) unsigned NOT NULL AUTO_INCREMENT COMMENT '唯一ID',
  `username` varchar(25) NOT NULL COMMENT '用户名',
  `password` varchar(25) NOT NULL COMMENT '密码',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8;
```

```
mysql> USE mybatisdemo;
Database changed
mysql> CREATE TABLE `user` (
    ->   `id` int(10) unsigned NOT NULL AUTO_INCREMENT COMMENT '唯一ID',
    ->   `username` varchar(25) NOT NULL COMMENT '用户名',
    ->   `password` varchar(25) NOT NULL COMMENT '密码',
    ->   PRIMARY KEY (`id`)
    -> ) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8;
Query OK, 0 rows affected (0.01 sec)
```

③、向 `info` 数据表中添加具体数据，如下图所示：

```
INSERT INTO user(id,username,password) VALUES (1, "power7089",
"power7089");
INSERT INTO user(id,username,password) VALUES (2, "root", "root");
INSERT INTO user(id,username,password) VALUES (3, "admin",
"admin@123");
```

```
mysql> INSERT INTO user(id,username,password) VALUES (1, "power7089", "power7089");
Query OK, 1 row affected (0.01 sec)

mysql>
mysql> INSERT INTO user(id,username,password) VALUES (2, "root", "root");
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO user(id,username,password) VALUES (3, "admin", "admin@123");
Query OK, 1 row affected (0.00 sec)
```

## 2.3、编写代码

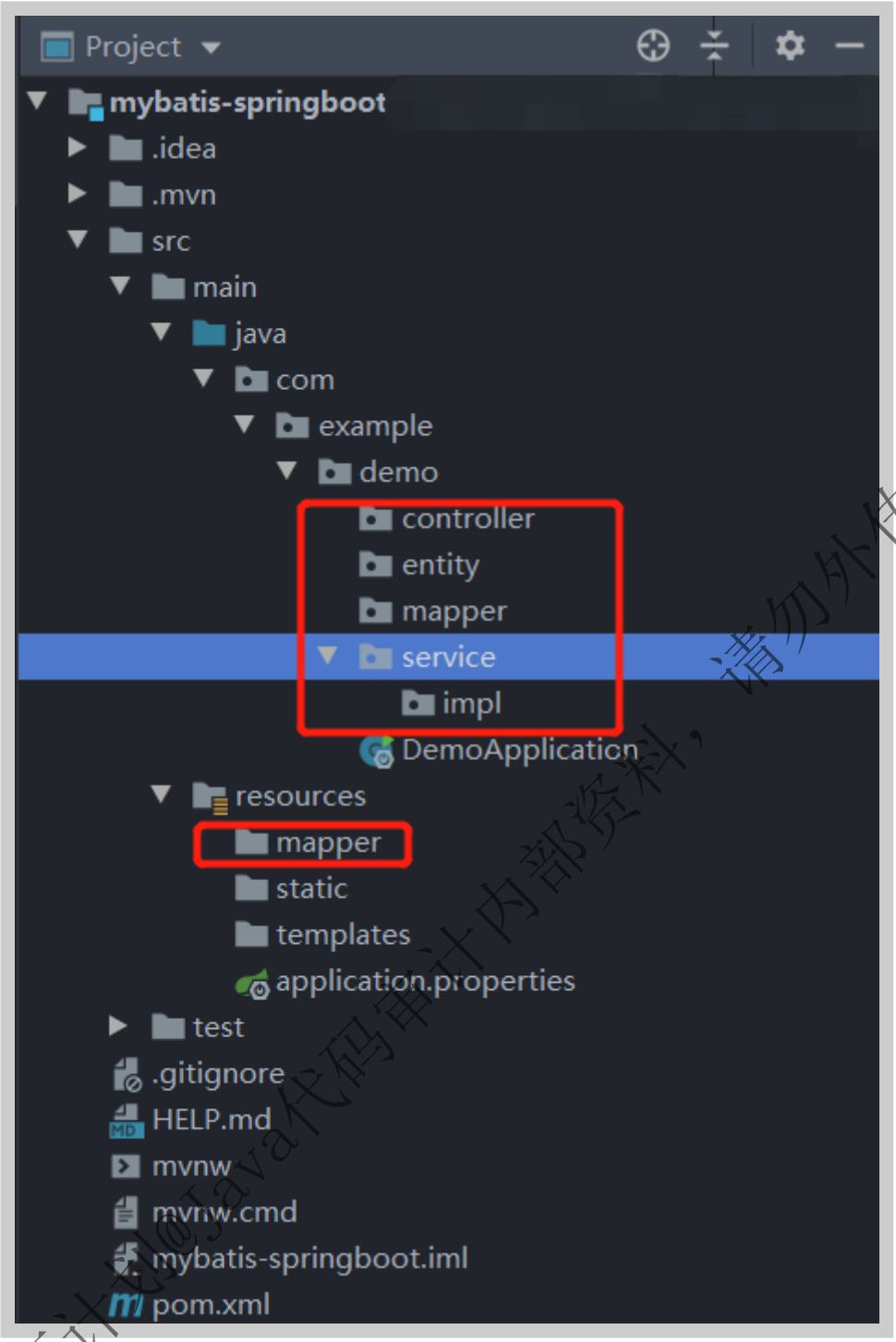
下面我们编写SpringBoot整合Mybatis实现增删改查的场景。

这个场景案例非常有意义，比较接近真实的SpringBoot项目。

并且通过示例代码，可以直观的了解到JavaWeb从接口到数据库的请求过程。

①、首先我们针对目录结构进行改动，根据不同作用类创建对应的代码包，在 `src.main.java.com.example.demo` 下分别创建 `controller` ， `service` ， `entity` ， `mapper` ， `service->impl` ，以及 `src.main.resources` 下创建 `mapper` ，最终目录结构如下图所示：

```
controller：Controller层
service：业务逻辑层
service/impl：service的实现
entity：实体类,作用一般是和数据表做映射。
mapper：数据操作层 DAO
```

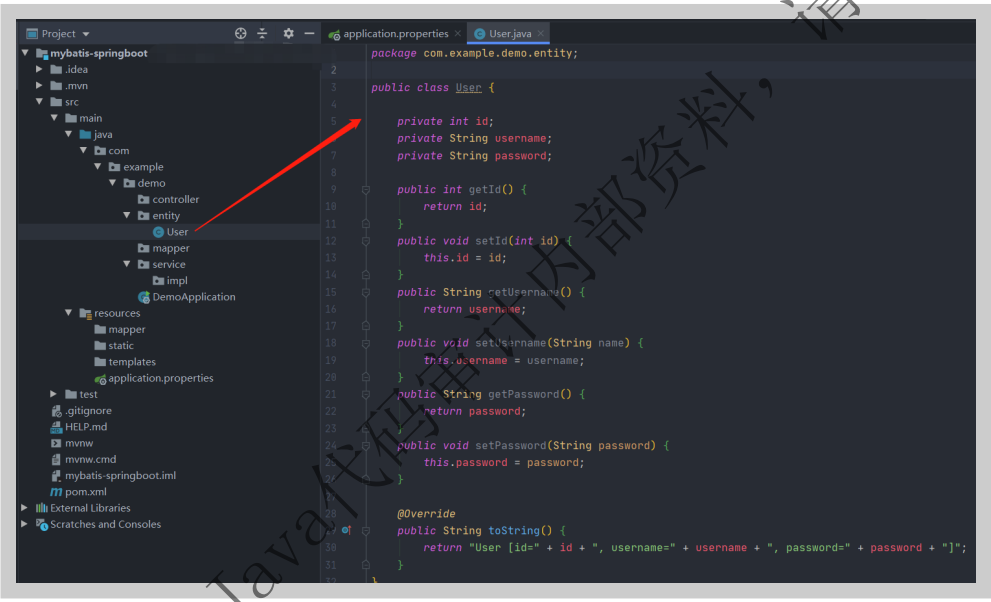





②、在 `src.main.java.com.example.demo.entiy` 包下新建一个Java Class，名为 `User`，这是一个实体类，和数据表做下映射，键入以下代码，最终如下图所示：

```
public class User {

    private int id;
    private String username;
    private String password;

    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
```

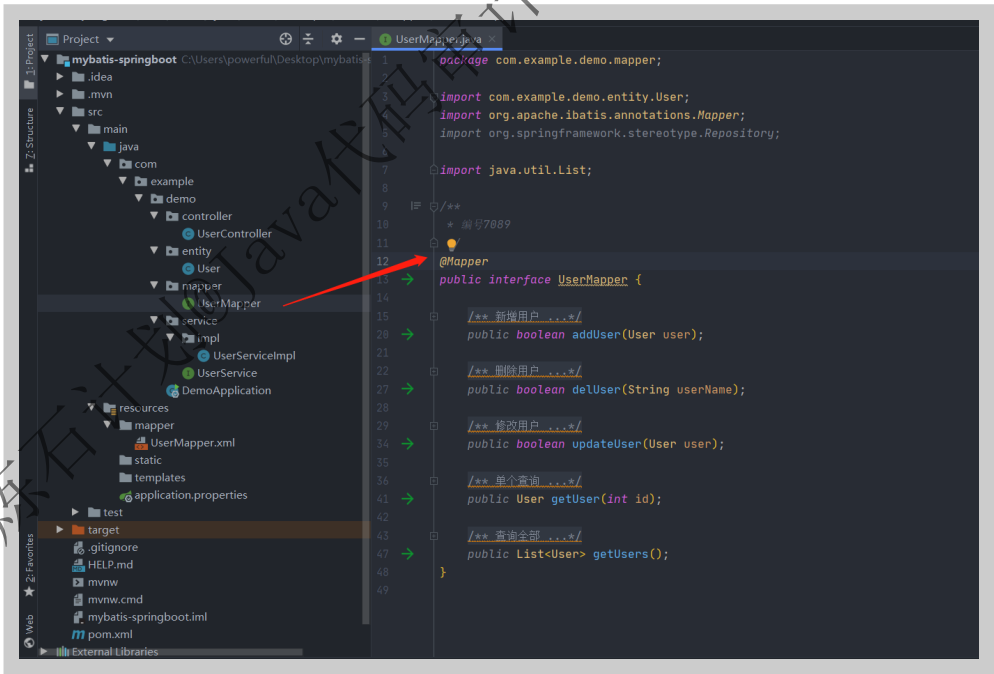
```
    public String getUsername() {
        return username;
    }
    public void setUsername(String name) {
        this.username = username;
    }
    public String getPassword() {
        return password;
    }
    public void setPassword(String password) {
        this.password = password;
    }

    @Override
    public String toString() {
        return "User [id=" + id + ", username=" + username + ",
password=" + password + "]";
    }
}
```




③、在 `src.main.java.com.example.demo.mapper` 文件下新建一个名为 `UserMapper` 的Java Interface，键入以下代码，最终如下图所示：

```
package com.example.demo.mapper;

import com.example.demo.entity.User;
import org.apache.ibatis.annotations.Mapper;
import org.springframework.stereotype.Repository;
import java.util.List;

@Mapper
public interface UserMapper {

    /**
     * 新增用户
     * @param user
     * @return
     */
    public boolean addUser(User user);
```

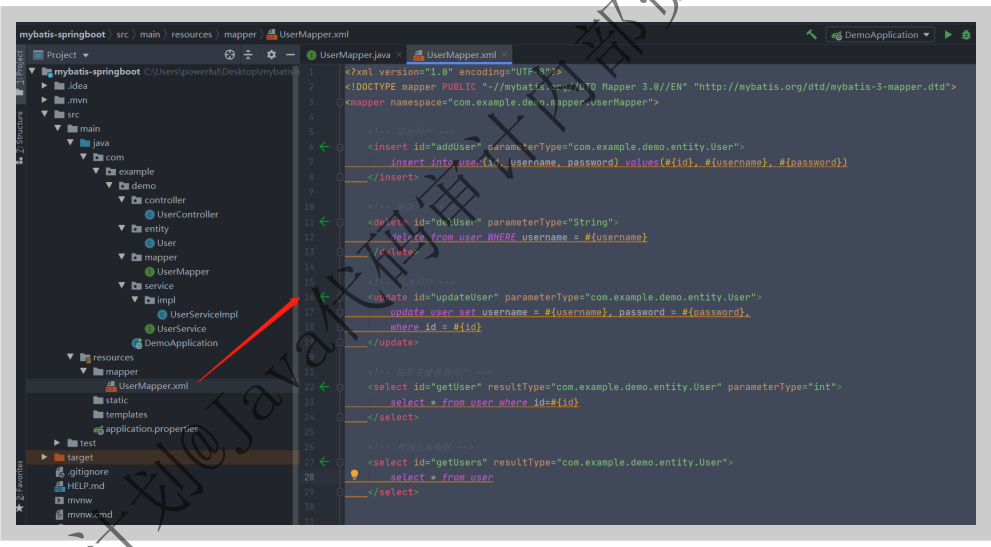
```
    /**
     * 删除用户
     * @param userName
     * @return
     */
    public boolean delUser(String userName);

    /**
     * 修改用户
     *@param user
     *@return
     * */
    public boolean updateUser(User user);

    /**
     * 单个查询
     * @param id
     * @return
     */
    public User getUser(int id);

    /**
     * 查询全部
     * @return
     */
    public List<User> getUsers();
}
```




④、在 `src.main.resources.mapper` 文件下新建一个名为 `UserMapper.xml` 文件，与dao层的 `UserMapper` 做好映射绑定，键入以下代码，最终如下图所示：

```
<?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.example.demo.mapper.UserMapper">
    <!-- 添加用户 -->
```

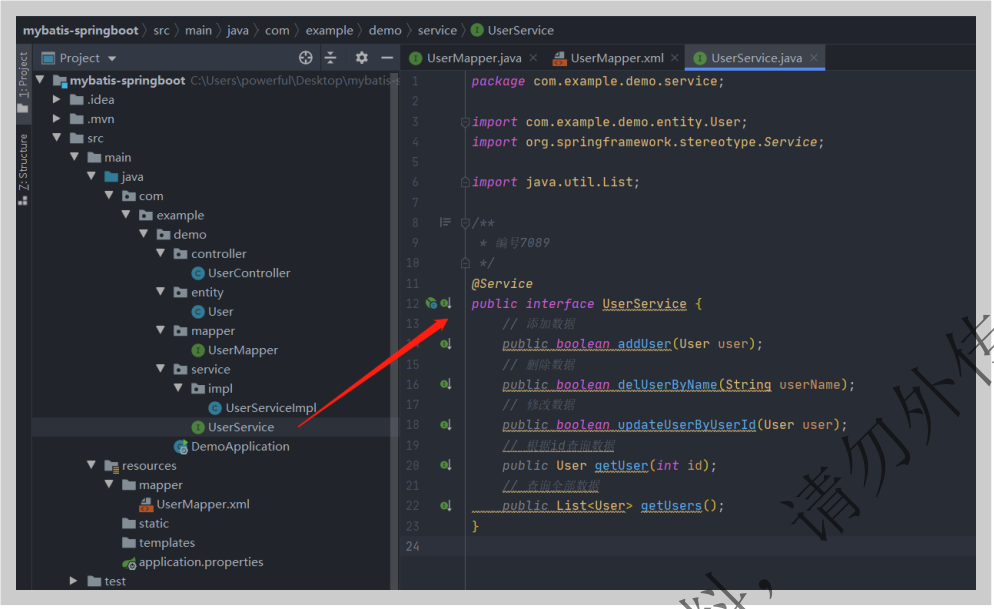
```
    <insert id="addUser" parameterType="com.example.demo.entity.User">
        insert into user(id, username, password) values(#{id}, #
{username}, #{password})
    </insert>
    <!-- 删除用户 -->
    <delete id="delUser" parameterType="String">
        delete from user WHERE username = #{username}
    </delete>
    <!-- 修改用户 -->
    <update id="updateUser"
parameterType="com.example.demo.entity.User">
        update user set username = #{username}, password = #{password},
        where id = #{id}
    </update>
    <!-- 根据主键查询用户 -->
    <select id="getUser" resultType="com.example.demo.entity.User"
parameterType="int">
        select * from user where id=#{id}
    </select>
    <!-- 查询全部数据 -->
    <select id="getUsers" resultType="com.example.demo.entity.User">
        select * from user
    </select>
</mapper>
```




⑤、在 `src.main.java.com.example.demo.service` 文件下新建一个名为 `UserService` 的Java interface，键入以下代码，最终如下图所示：

```
package com.example.demo.service;

import com.example.demo.entity.User;
import org.springframework.stereotype.Service;
import java.util.List;

@Service
public interface UserService {
    // 添加数据
    public boolean addUser(User user);
    // 删除数据
    public boolean delUserByName(String userName);
    // 修改数据
    public boolean updateUserByUserId(User user);
```

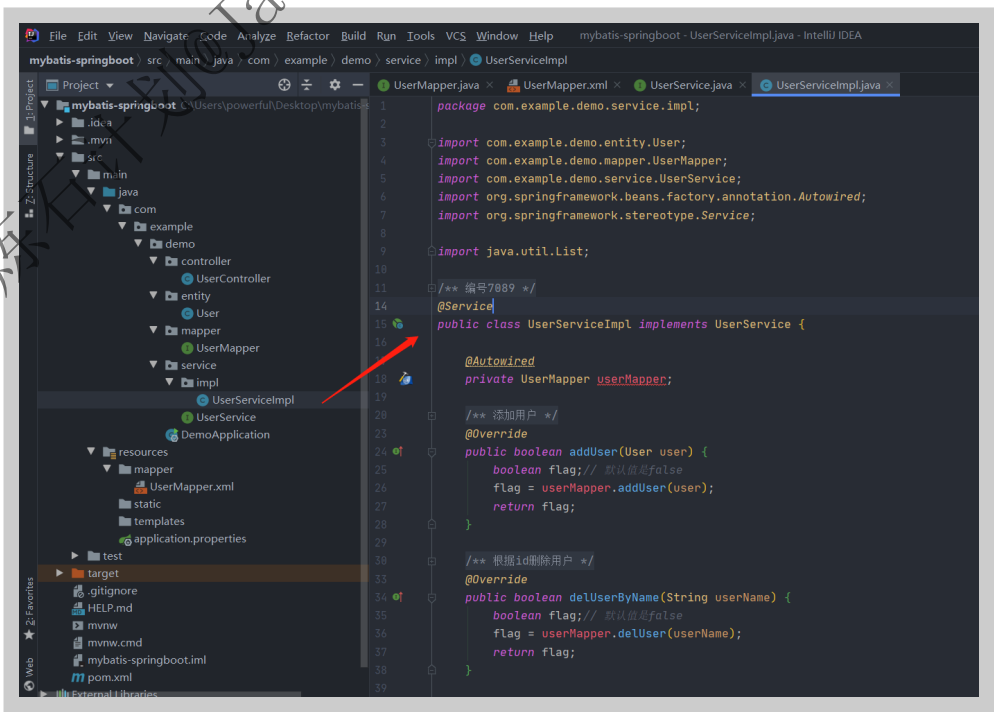
```
    // 根据id查询数据
    public User getUser(int id);
    // 查询全部数据
    public List<User> getUsers();
}
```

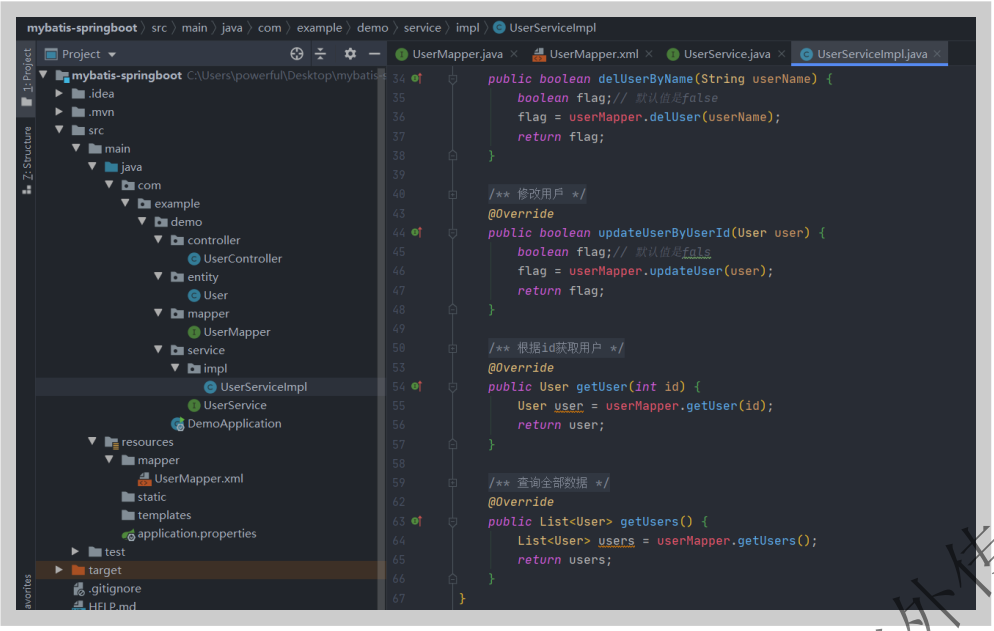



⑤、在 `src.main.java.com.example.demo.service.impl` 文件下新建一个名为 `UserServiceImpl` 的Java class，这是UserService实现方法，继承UserService并重写方法。键入以下代码，最终如下图所示：

```
package com.example.demo.service.impl;

import com.example.demo.entity.User;
import com.example.demo.mapper.UserMapper;
import com.example.demo.service.UserService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import java.util.List;

@Service
public class UserServiceImpl implements UserService {

    @Autowired
    private UserMapper userMapper;

    /**
     * 添加用户
     * */
    @Override
    public boolean addUser(User user) {
        boolean flag;// 默认值是false
        flag = userMapper.addUser(user);
        return flag;
    }

    /**
     * 根据id删除用户
     * */
```

```
    @Override
    public boolean delUserByName(String userName) {
        boolean flag;// 默认值是false
        flag = userMapper.delUser(userName);
        return flag;
    }

    /**
     * 修改用户
     * */
    @Override
    public boolean updateUserByUserId(User user) {
        boolean flag;// 默认值是fals
        flag = userMapper.updateUser(user);
        return flag;
    }

    /**
     * 根据id获取用户
     * */
    @Override
    public User getUser(int id) {
        User user = userMapper.getUser(id);
        return user;
    }

    /**
     * 查询全部数据
     * */
    @Override
    public List<User> getUsers() {
        List<User> users = userMapper.getUsers();
        return users;
    }
}
```

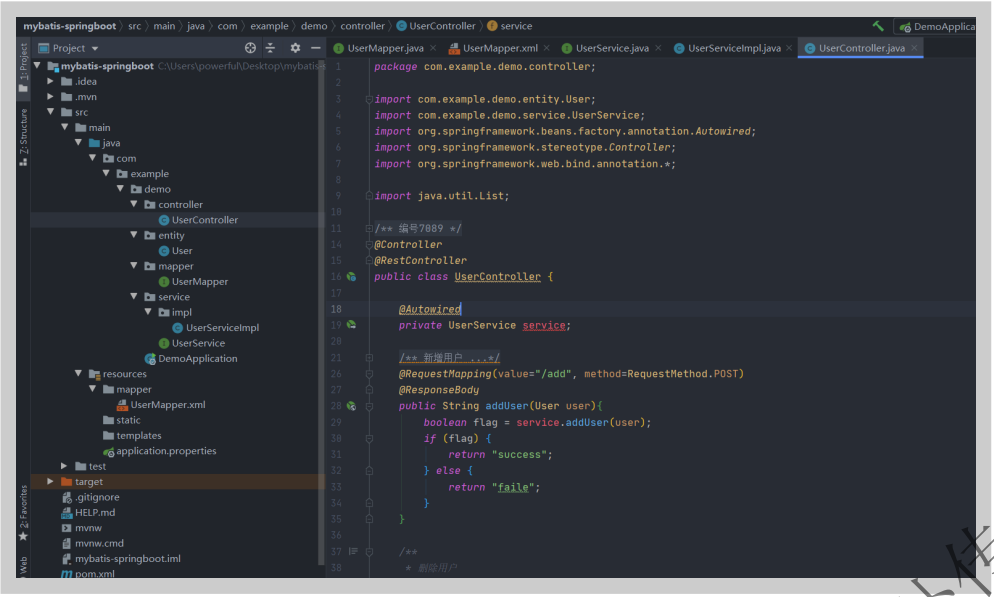





⑥、在 `src.main.java.com.example.demo.controller` 文件下新建一个名为 `UserController` 的Java class，键入以下代码，最终如下图所示：

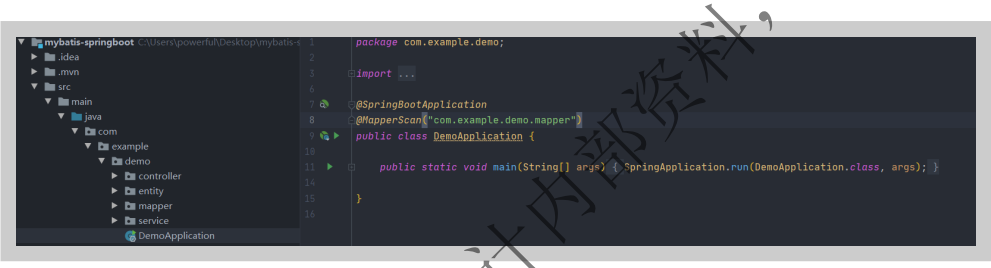
```
package com.example.demo.controller;

import com.example.demo.entity.User;
import com.example.demo.service.UserService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.*;
import java.util.List;

@Controller
@RestController
public class UserController {

    @Autowired
    private UserService service;

    /**
     * 新增用户
     * @param user
     * @return
     */
    @RequestMapping(value="/add", method=RequestMethod.POST)
    @ResponseBody
    public String addUser(User user){
        boolean flag = service.addUser(user);
        if (flag) {
            return "success";
        } else {
            return "faile";
        }
    }

    /**
     * 删除用户
     *
```

```
     * @param name
     * @return
     * */
    @RequestMapping(value="/del", method=RequestMethod.POST)
    @ResponseBody
    public String delUserByName(@RequestParam("name") String userName)
{

        boolean flag = service.delUserByName(userName);
        if (flag) {
            return "success";
        } else {
            return "faile";
        }
    }

    /**
     * 修改用户
     *
     * @param User
     * @return
     * */
    @RequestMapping(value="/updata", method=RequestMethod.POST)
    @ResponseBody
    public String updateUserByName(User user) {
        boolean flag = service.updateUserByUserId(user);
        if (flag) {
            return "success";
        } else {
            return "faile";
        }
    }

    /**
     * 单个查询
     * @param id
     * @return
     */
    @RequestMapping(value="/get/{id}", method= RequestMethod.GET)
    public User getUser(@PathVariable("id") int id){
        User user = service.getUser(id);
        return user;
    }

    /**
     * 查询全部
     * @return
     */
    @RequestMapping(value="/getUser/list", method=RequestMethod.GET)
    @ResponseBody // @ResponseBody - 返回json字符串
    public List<User> getUsers(){
        List<User> users = service.getUsers();
        return users;
    }
}
```



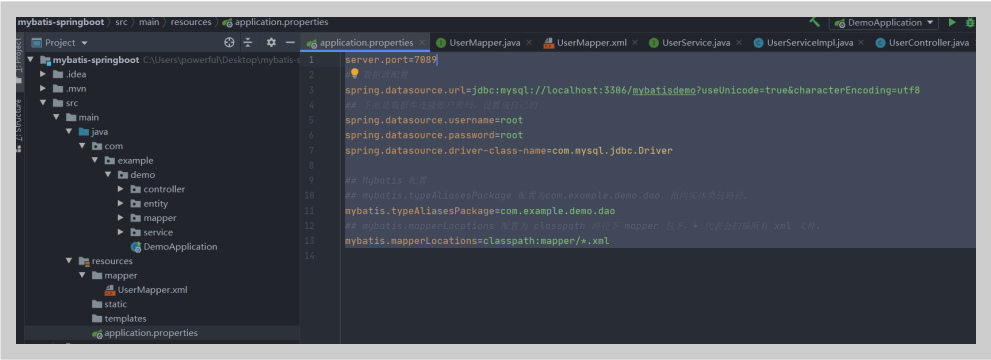

⑦、在 `DemoApplication` 启动类中添加注解 `@MapperScan()` ，如下图所示：

@MapperScan注解：扫描com.example.demo.dao下的所有的类作为Mapper映射文件

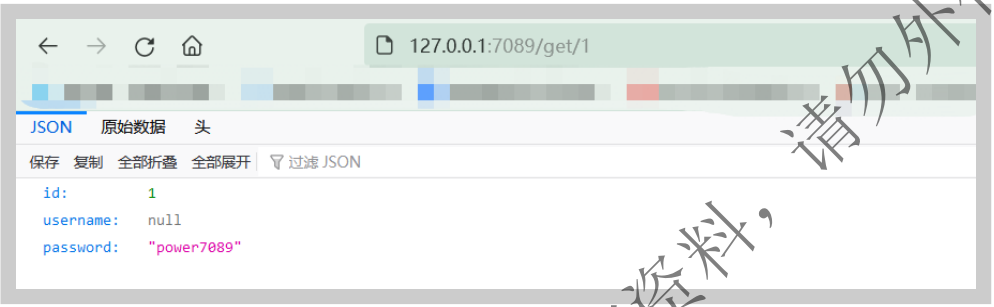


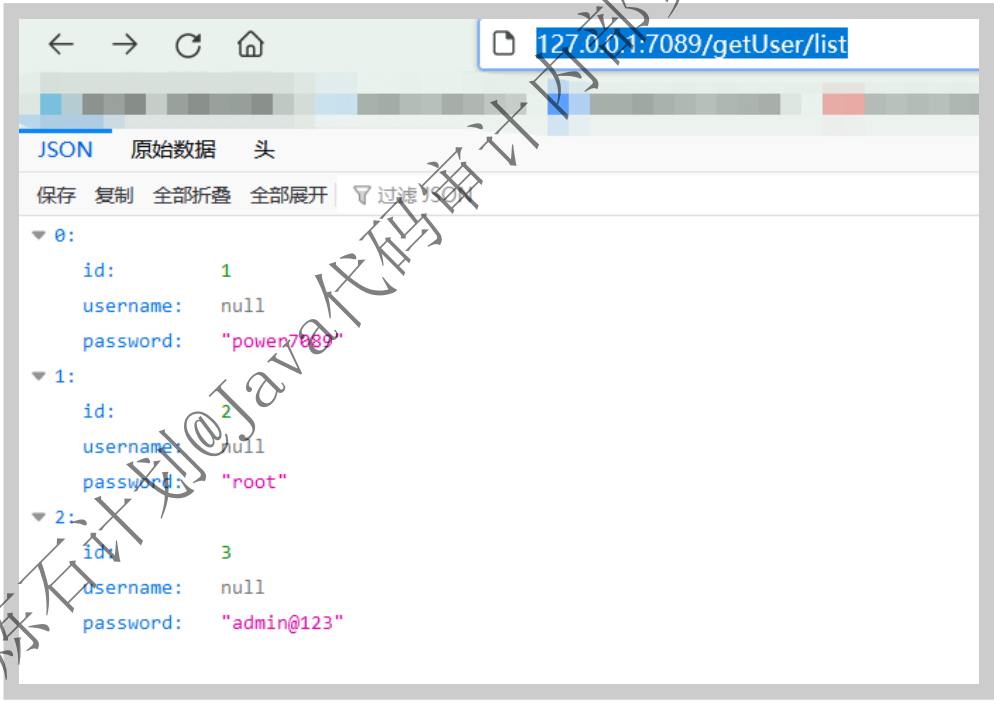

⑧、最后在 `src.main.resources -> application.properties` 文件中添加以下配置，如下图所示：

```
server.port=7089
## 数据源配置
spring.datasource.url=jdbc:mysql://localhost:3306/mybatisdemo?
useUnicode=true&characterEncoding=utf8
## 下面是数据库连接账户密码，设置成自己的
spring.datasource.username=root
spring.datasource.password=root
spring.datasource.driver-class-name=com.mysql.jdbc.Driver

## Mybatis 配置
## mybatis.typeAliasesPackage 配置为com.example.demo.dao，指向实体类包路径。
mybatis.typeAliasesPackage=com.example.demo.dao
## mybatis.mapperLocations 配置为 classpath 路径下 mapper 包下，* 代表会扫描
所有 xml 文件。
mybatis.mapperLocations=classpath:mapper/*.xml
```




⑨、启动项目，访问我们在 `UserController` 中定义的接口，比如：`http://127.0.0.1:7089/get/1`， `http://127.0.0.1:7089/getUser/list`，最终响应如下图所示：






还剩几个接口，大家自行调试下吧。有问题不要怕，迎难而上解决问题。

注意：

在实际项目开发中，每个人的风格不同，实现代码的方式因此也各有千秋。我上面举例的代码仅为最基础，便于理解。在我们以后代码审计中，肯定会遇见各种风格代码，以后遇见再说吧，毕竟以后我们的重心还是在代码审计上面。