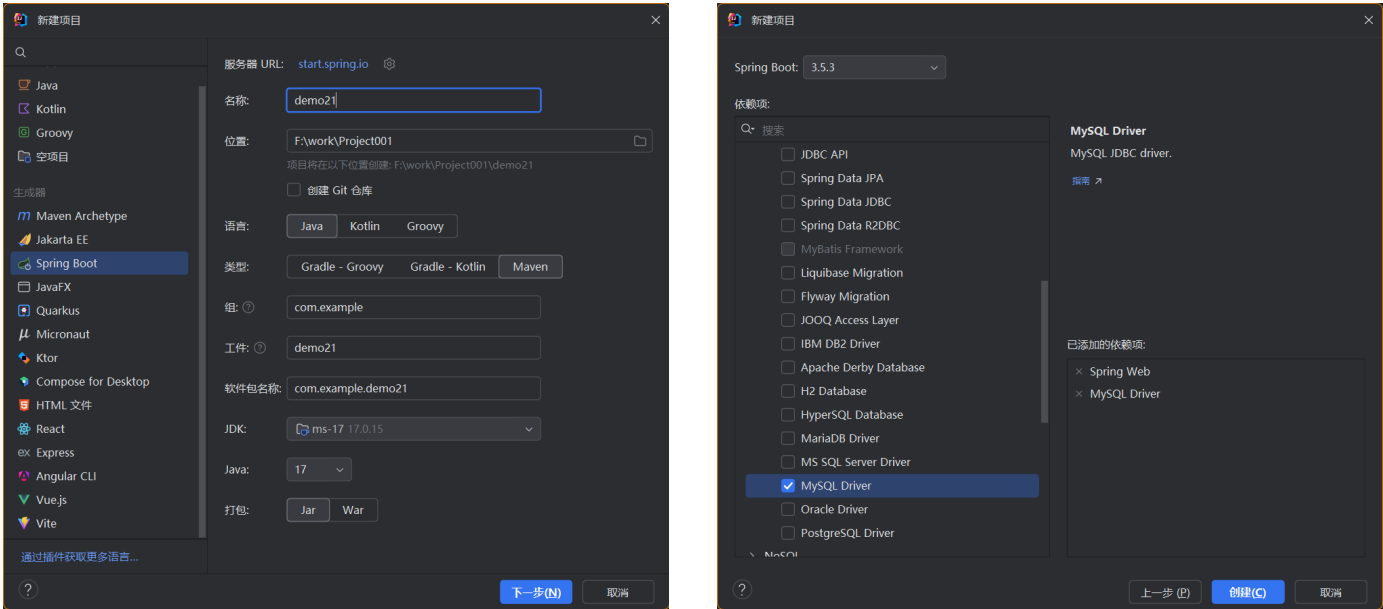


0624 SpringBoot与数据库

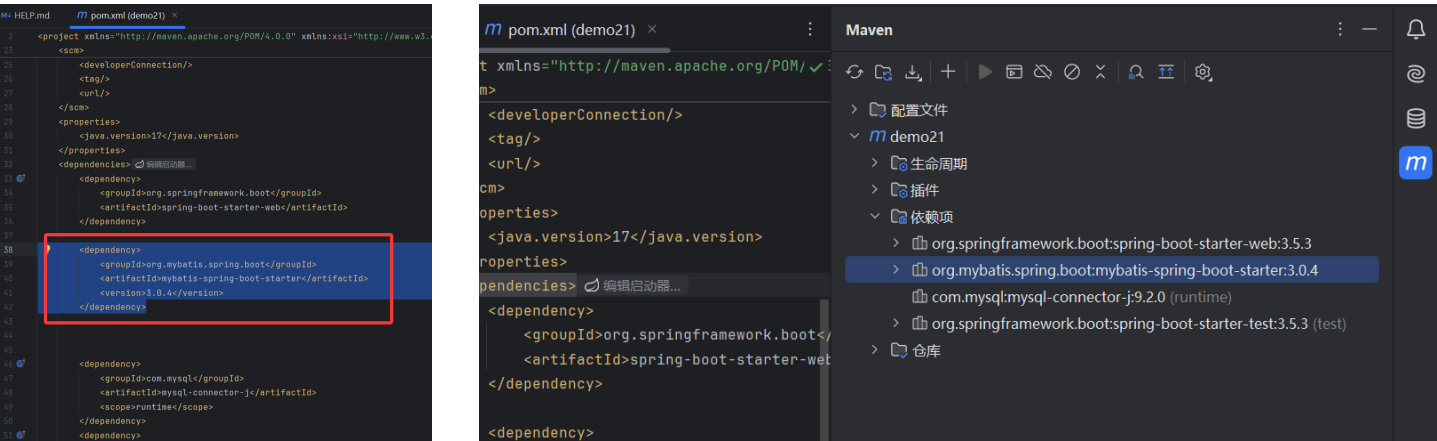
搭建SpringBoot开发环境

1. 下载InteliJ IDEA 2024.1，新建项目，设置如下图



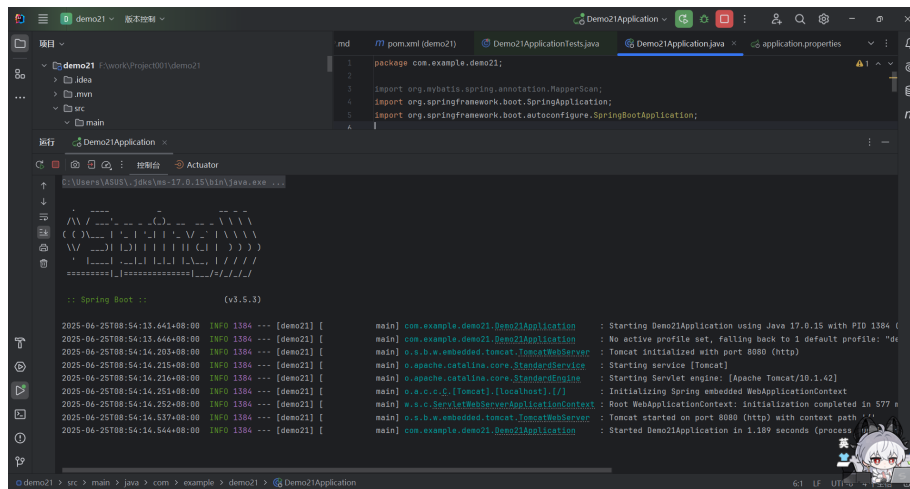
注意SpringBoot是3.5.3，此版本不能直接勾选MyBatis Framework，需要在第2步中自己设置

2. 打开pom.xml，加入mybatis-spring-boot-starter，善用自动补全，记得在右上角更新maven（有小图标）



3. 接下来，进行一些基础的尝试

a. 处理关于数据源的问题后运行Application.java，正常启动情况如下：



b. 在浏览器写出hello

i. 新建Controller文件夹，添加一个UserController.java文件，写一个返回为"hello"的函数

代码块

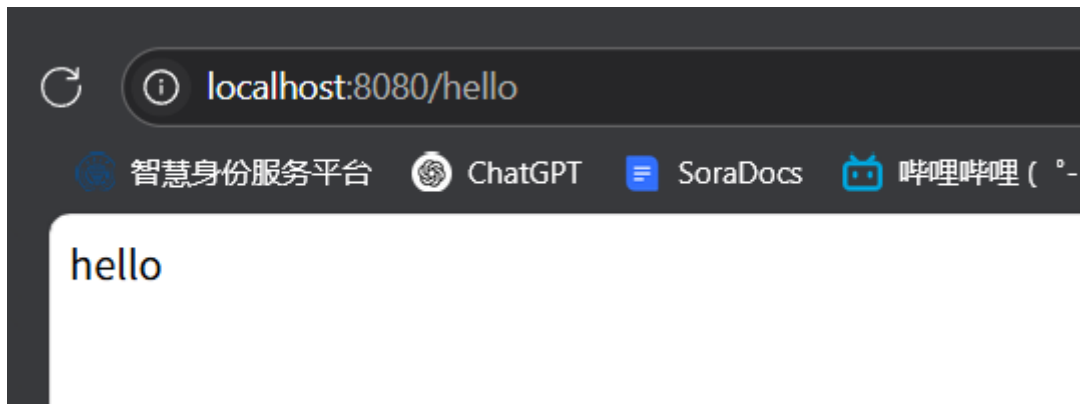
```
1 package com.example.demo21.Controller;
2
3 import org.springframework.web.bind.annotation.RequestMapping;
4 import org.springframework.web.bind.annotation.RestController;
5
6 @RestController
7 public class UserController {
8
9     @RequestMapping("/hello")
10    public String hello(){
11        return "hello";
12    }
13
14 }
```

ii. 在application.properties里添加端口

代码块

```
1 server.port=8080
```

iii. 启动项目，在浏览器访问localhost:8080/hello



SpringBoot基础

1. 数据类型分类

数据类型		内存占用(字节数)	数据范围
整型	byte	1	-128~127
	short	2	-32768~32767
	int(默认)	4	-2147483648~2147483647 (10位数, 大概21亿多)
	long	8	-9223372036854775808 ~ 9223372036854775807 (19位数)
浮点型(小数)	float	4	1.401298 E -45 到 3.4028235 E +38
	double (默认)	8	4.9000000 E -324 到 1.797693 E +308
字符型	char	2	0-65535
布尔型	boolean	1	true, false
字符串	String		"工夺a"

2. 定义类:

- a. 基本类型
- b. 方法
 - i. 有返回值 public String GetString()
 - ii. 无返回值 public void GetUserid()
- c. 控制台输出
 - i. System.out.println(tt);

3. 原生java: new 对象

- a. 在main函数文件同级新建类User.java

代码块

```
1 package com.example.demo21;
2
3 public class User {
4     int id;
```

```

5      String name;
6      Integer age;
7      boolean gender;
8
9      public void GetUerid(){
10         id = 8;
11         name = "Zhangsan";
12         age = 20;
13         gender = false;
14
15         System.out.println(id);
16     }
17 }

```

b. 在main函数new对象User，实例化为user1，调用函数

代码块

```

1  package com.example.demo21;
2
3  import org.mybatis.spring.annotation.MapperScan;
4  import org.springframework.boot.SpringApplication;
5  import org.springframework.boot.autoconfigure.SpringBootApplication;
6
7  //@MapperScan("com.example.demo21.mapper")
8  @SpringBootApplication
9  public class Demo21Application {
10
11      public static void main(String[] args) {
12          SpringApplication.run(Demo21Application.class, args);
13          User user1 = new User();
14          user1.GetUerid();
15      }
16
17
18  }

```

c. 成功运行结果如下

```
( ( ) \_ _ _ | ' _ | ' _ | | ' _ \ / _ ' | \ \ \ \
\ \ / _ _ _ ) | | _ ) | | | | | | ( _ | | ) ) )
' | _ _ _ | . _ _ | | | _ | | \ _ _ , | / / / /
===== | _ | ===== | _ _ / = / _ / _ / _ /

:: Spring Boot ::                (v3.5.3)

2025-06-25T10:55:43.191+08:00 INFO 30032 --- [demo21] [
2025-06-25T10:55:43.193+08:00 INFO 30032 --- [demo21] [
2025-06-25T10:55:43.819+08:00 INFO 30032 --- [demo21] [
2025-06-25T10:55:43.831+08:00 INFO 30032 --- [demo21] [
2025-06-25T10:55:43.831+08:00 INFO 30032 --- [demo21] [
2025-06-25T10:55:43.878+08:00 INFO 30032 --- [demo21] [
2025-06-25T10:55:43.879+08:00 INFO 30032 --- [demo21] [
2025-06-25T10:55:44.155+08:00 INFO 30032 --- [demo21] [
2025-06-25T10:55:44.162+08:00 INFO 30032 --- [demo21] [
8
```

d. 此时，在main函数中修改user1.id=1，修改失败；需要在User类中删除对id的定义，此时修改成功。其他项同理，也可以尝试新建一个示例user2

e. 又在User.java添加有返回类

代码块

```
1 public String GetString(){
2     return name = "lisi";
```

f. 并在main函数中添加

代码块

```
1 String tt = user1.GetString();
2 System.out.println(tt);
```

g. 成功运行结果如下

```
:: Spring Boot :: (v3.5.3)

2025-06-25T11:21:43.611+08:00 INFO 8316 --- [demo21]
2025-06-25T11:21:43.614+08:00 INFO 8316 --- [demo21]
2025-06-25T11:21:44.206+08:00 INFO 8316 --- [demo21]
2025-06-25T11:21:44.216+08:00 INFO 8316 --- [demo21]
2025-06-25T11:21:44.216+08:00 INFO 8316 --- [demo21]
2025-06-25T11:21:44.251+08:00 INFO 8316 --- [demo21]
2025-06-25T11:21:44.251+08:00 INFO 8316 --- [demo21]
2025-06-25T11:21:44.506+08:00 INFO 8316 --- [demo21]
2025-06-25T11:21:44.514+08:00 INFO 8316 --- [demo21]
1
lisi
```

h. 1

4. Controller: 定义一个类（自定义），当一个网络请求上来，需要给这个请求返回一个结果，在java文件中建立一个Controller文件夹，所有的请求返回类在此文件夹

a. 类命名规则: xxxController.class

5. Spring框架: 定义注入

a. @Component 用于标记该类为Spring组件（定义），交由Spring管理

b. @Autowired 自动注入依赖

c. 修改代码如下

```
ToUser

1  package com.example.demo21;
2
3  import lombok.Data;
4  import org.springframework.beans.factory.annotation.Value;
5  import org.springframework.stereotype.Component;
6
7  @Component
8  @Data
9
10 public class ToUser {
11     @Value("${user.name}")
12     String name;
13     @Value("${user.password}")
14     String password;
15 }
```

```

16     public String getString(){
17         return name+password;
18     }
19
20 }

```

properties

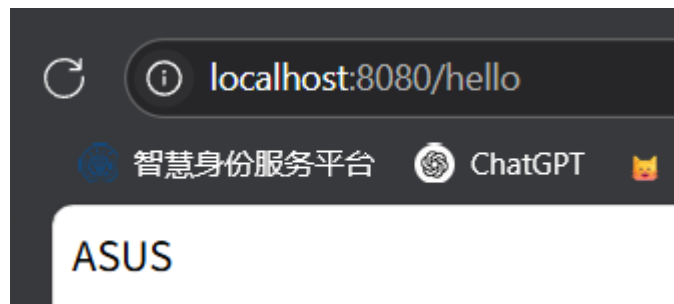
```

1  user.name=admin
2  user.password=admin

```

d. 成功运行，去localhost:8080/hello看到结果

properties里写的是admin,但是Java 会把user.name自动读取操作系统的用户名覆盖配置赋值,所以我这里显示的是ASUS,如果想要自定义可以把admin换为其他,或者把user.name改为其他变量名



熟悉Spring的一些注解

1. 依赖注入方法

- a. @Component 在类的上方，将类的实例化（new对象）交给SpringBoot管理
 - i. @Service：业务逻辑层
 - ii. @Repository：数据访问层
 - iii. @Controller：Web 控制层
- b. @Autowired 在对象的上方，调用管理器里的对象
- c. @Data 在类的上方，编译后的.class文件会自动包含常用方法，但源码保持简洁

2. @Component 以及其子项

注解	层/用途	特殊功能	典型使用场景
@Component	通用组件	无	工具类、第三方库组件、无法明确归类的 Bean
	业务逻辑层	语义化标识	

@Service			业务逻辑处理（如 UserService 、 OrderService ）
@Repository	数据访问层	自动异常转换（数据库异常 → Spring 异常）	数据库操作（如 UserRepository 、 OrderDao ）
@Controller	Web 控制层（视图）	处理 HTTP 请求	返回 HTML/JSP 页面（传统 MVC 应用）
@RestController	Web 控制层（REST API）	直接返回 JSON/XML 数据	前后端分离的 API 接口（如 UserApiController ）
@Configuration	配置类	定义 @Bean 方法	配置数据源、第三方库 Bean（如 RedisConfig ）

- 3. @Mapper
- 4. @Value
- 5. @Data
- 6. @RestController
- 7. @SpringBootApplication
- 8. @MapperScan
- 9. @GetMapping等

配置数据库

- 1. 数据库：117.72.192.208，端口3306，用户名：Z5，密码：Z5，数据库名z5。配置properties连接

```
properties
1  # DataSource
2  spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
3  spring.datasource.url=jdbc:mysql://117.72.192.208:3306/z5
4  spring.datasource.username=Z5
5  spring.datasource.password=Z5
```

- 2. 在库内建表，内容如下，表名testtable



3. 尝试在<http://localhost:8080/list>中查询表内容

a. 新建Mapper包，添加Z5Mapper文件，内容如下

代码块

```
1 package com.example.demo21.Mapper; //
   src/main/java/com/example/demo21/mapper/Z5Mapper.java
2 import org.apache.ibatis.annotations.Mapper;
3 import org.apache.ibatis.annotations.Select;
4 import java.util.List;
5 import java.util.Map;
6
7 @Mapper // 关键注解，标记为MyBatis接口
8 public interface Z5Mapper {
9
10     // 方法1: 返回List<Map> (通用键值对结构)
11     @Select("SELECT * FROM testtable")
12     List<Map<String, Object>> getAllZ5();
13
14 }
```

b. 在Controller包中添加Z5Controller文件，内容如下

代码块

```
1 package com.example.demo21.Controller; //
   src/main/java/com/example/demo21/controller/Z5Controller.java
```

```

2  import com.example.demo21.Mapper.Z5Mapper;
3  import org.springframework.beans.factory.annotation.Autowired;
4  import org.springframework.web.bind.annotation.GetMapping;
5  import org.springframework.web.bind.annotation.RestController;
6  import java.util.List;
7  import java.util.Map;
8
9  @RestController
10 public class Z5Controller {
11
12     @Autowired
13     private Z5Mapper z5Mapper; // 注入Mapper
14
15     @GetMapping("/list")
16     public List<Map<String, Object>> getZ5List() {
17         return z5Mapper.getAllZ5(); // 直接调用Mapper方法
18     }
19 }

```

c. 修改main文件，加上

代码块

```

1  @MapperScan("com.example.demo21.Mapper")

```

d. 成功结果如下



编写MyBatis查询语句

1. 在 MyBatis 中，查询语句可以通过以下两种主要方式编写
 - a. XML：复杂查询
 - b. 直接写在Mapper接口：简单查询
2. 主要讲使用@Mapper类下@Select("sql语句")的查询，步骤如下

- a. 配置maven
- b. 在Mapper包中新建对应接口类xxMapper，类上添加@Mapper，类中返回类型函数前加上带sql的@Select
- c. 对应xxController.java，依赖注入后直接调用该方法函数
- d. 在main函数前@Mapper("Mapper包地址")

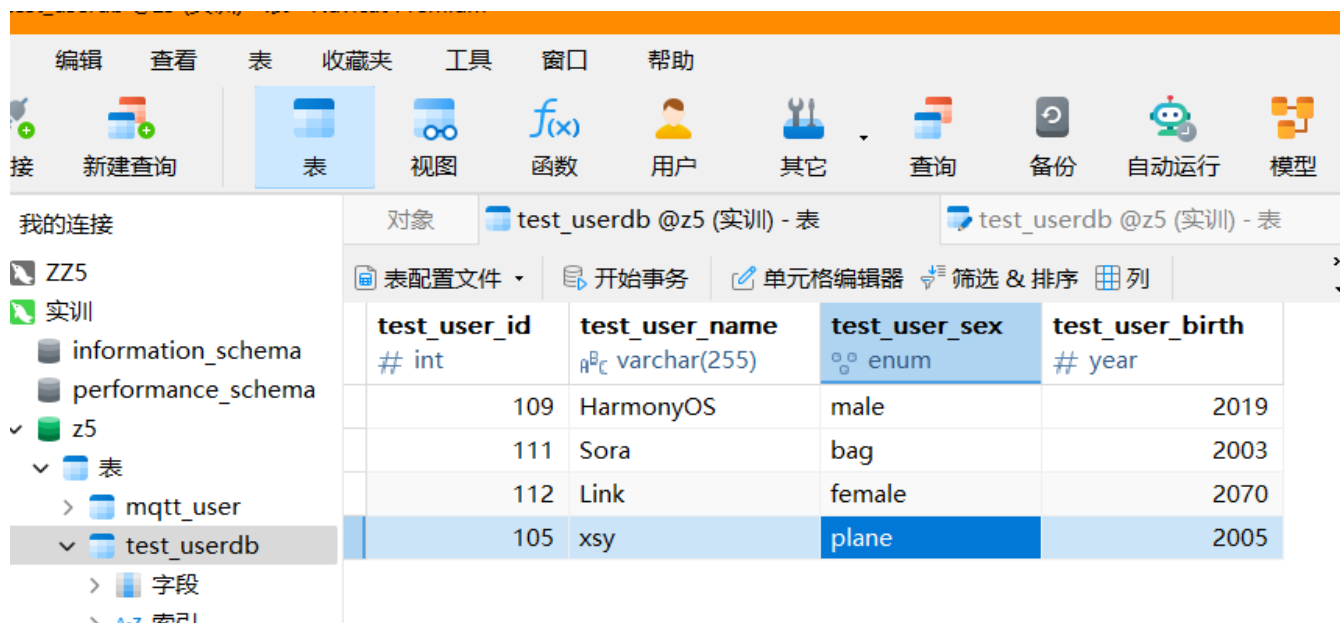
3. 执行流程如下

- Spring容器启动时，扫描所有带有@Mapper注解的接口，MyBatis自动为每个接口生成动态代理实现类
- 解析@Select注解中的SQL语句，构建参数映射
- 调用Mapper方法，实际调用的是MyBatis生成的代理对象

测试GET、POST与PUT

1. 测试Get：尝试在数据库中查询单个值

- a. 我的testtable被转生了我们换个数据库



test_user_id # int	test_user_name varchar(255)	test_user_sex enum	test_user_birth # year
109	HarmonyOS	male	2019
111	Sora	bag	2003
112	Link	female	2070
105	xsy	plane	2005

- b. 新建UserController

代码块

```
1 package com.example.demo21.Controller;
2
3 import com.example.demo21.Mapper.UserMapper;
4 import org.springframework.beans.factory.annotation.Autowired;
5 import org.springframework.web.bind.annotation.GetMapping;
6 import org.springframework.web.bind.annotation.RestController;
7
8 @RestController
```

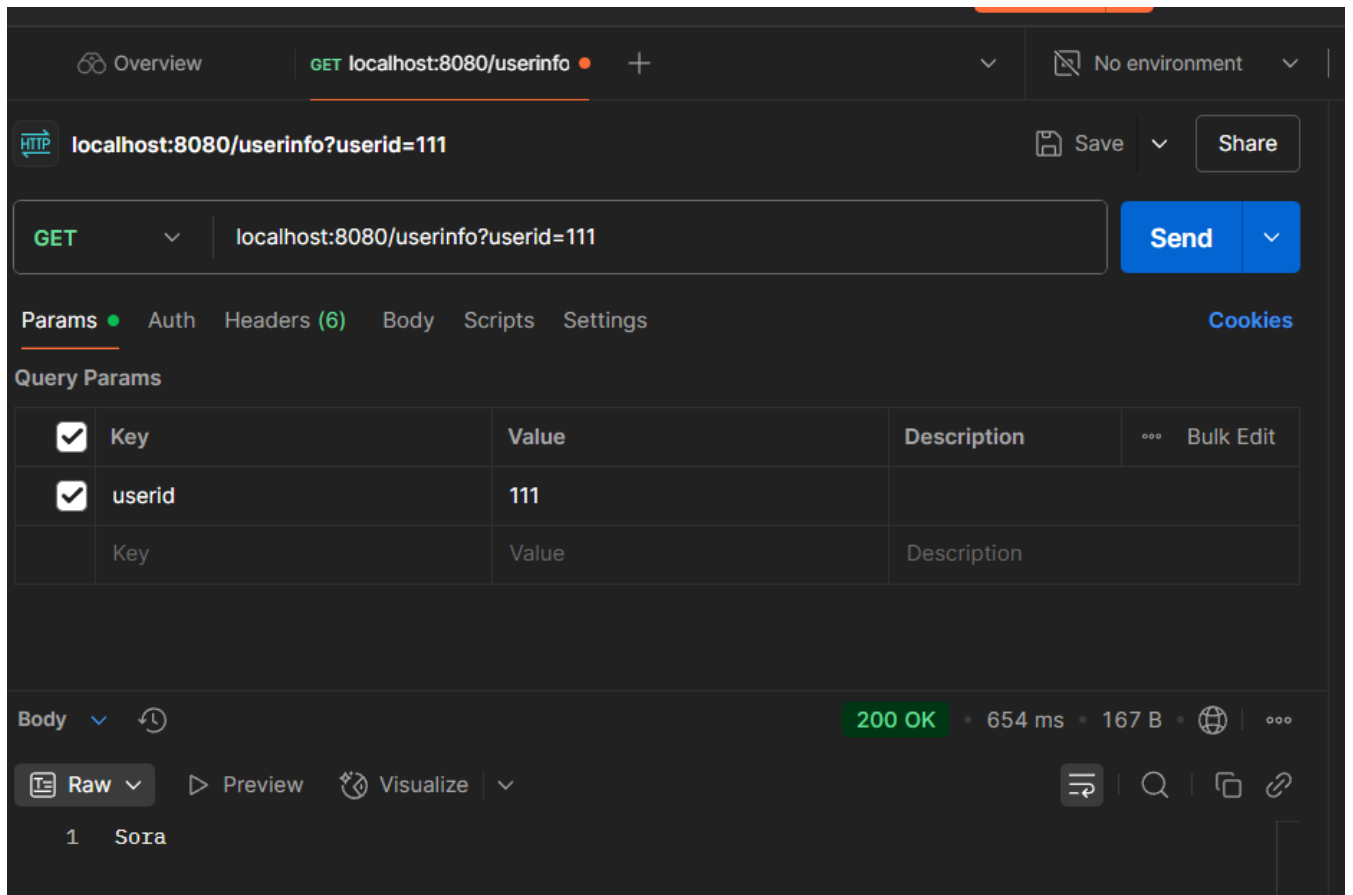
```
9  public class UserController {
10      @Autowired
11      UserMapper usrMapper;
12
13      @GetMapping("/userinfo")
14      public String getUsername(String userid){
15          return usrMapper.getUsername(userid);
16      }
17  }
```

c. 新建UserMapper

代码块

```
1  package com.example.demo21.Mapper;
2
3  import org.apache.ibatis.annotations.Mapper;
4  import org.apache.ibatis.annotations.Select;
5
6  @Mapper
7  public interface UserMapper {
8
9      @Select("SELECT test_user_name FROM test_userdb WHERE test_user_id =
      #{userid}")
10     public String getUsername(String userid);
11 }
```

d. 启动程序，用Postman查询，格式：localhost:8080/userinfo?userid=xxx，结果如下



2. 测试POST：尝试在数据库中插入一行

a. 在UserMapper里添加新方法

代码块

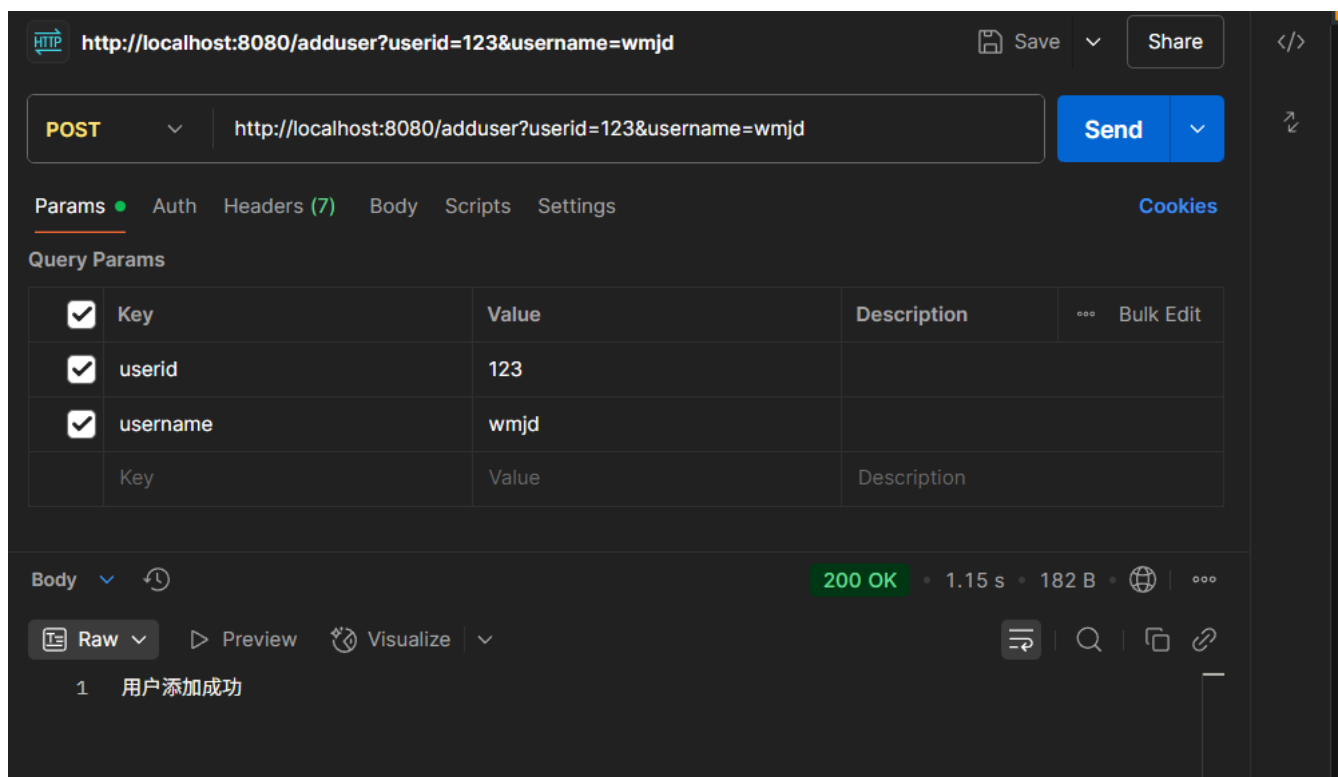
```
1 // 新增 POST 方法 (插入用户)
2 @Insert("INSERT INTO test_userdb(test_user_id, test_user_name) VALUES(#{userid}, #{username})")
3 int insertUser(String userid, String username);
```

b. 在UserController里添加新方法

代码块

```
1 @PostMapping("/adduser")
2 public String addUser(
3     String userid,
4     String username
5 ) {
6     int result = usrMapper.insertUser(userid, username); // 直接传参数
7     return result > 0 ? "用户添加成功" : "用户添加失败";
8 }
```

- c. 在Postman中测试添加一行，格式：localhost:8080/adduser?userid=123&username=wmjd，结果如下



3. 测试PUT：尝试在数据库中更新一行

- a. 在Mapper中新增

代码块

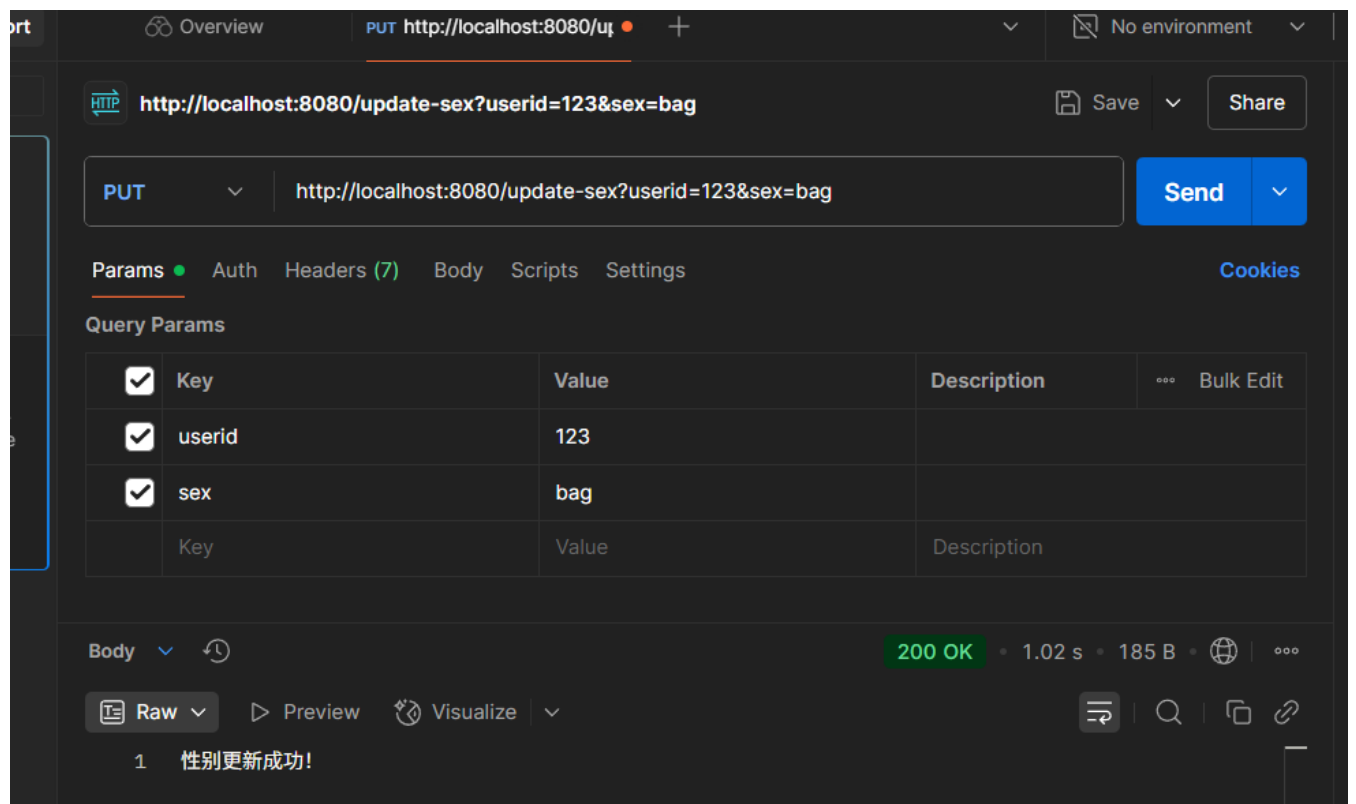
```
1 @Update("UPDATE test_userdb SET test_user_sex = #{sex} WHERE  
test_user_id = #{userid}")  
2 int updateUserSex(String userid, String sex);
```

- b. 在Controller中新增

代码块

```
1 @PostMapping("/update-sex")  
2 public String updateUserSex(  
3     String userid,  
4     String sex // 接收 "male", "female", "bag", "plane"  
5 ) {  
6     int result = usrMapper.updateUserSex(userid, sex);  
7     return result > 0 ? "性别更新成功!" : "更新失败（用户不存在或数据未变  
动）";  
8 }
```

- c. 在Postman中测试<http://localhost:8080/update-sex?userid=123&sex=bag>，成功结果如下



可能遇到的问题

1. JDK设置

报错：ERROR:JAVA: 错误: 不支持发行版本 5

可能的问题：JDK版本有误；JDK设置有误

解决方案：

- https://blog.csdn.net/xiao_yi_xiao/article/details/119142118
- 重建项目，检查JDK选择
- 有来自不可知域的科研舞狮经验表明，安装在非C盘的jdk容易发电

2. Maven设置

主播主播，我的Maven依赖一直跑不好怎么办？

建议按顺序尝试以下方法：

- 由于国内网络首次使用会有极大延迟，可以魔法上网
- 有可能是单纯的网速慢，可以多加等待
- 自行下载本地Maven并配置，给出参考文档
 - <https://blog.csdn.net/u012660464/article/details/114113349>
 - <https://blog.csdn.net/u012660464/article/details/114093066?spm=1001.2101.3001.10752>

3. 数据源连接问题

报错：

Error starting ApplicationContext. To display the condition evaluation report re-run your application with 'debug' enabled.

2025-06-25T01:03:28.477+08:00 ERROR 10252 --- [demo21] [main]

o.s.b.d.LoggingFailureAnalysisReporter :

APPLICATION FAILED TO START

Description:

Failed to configure a DataSource: 'url' attribute is not specified and no embedded datasource could be configured.

Reason: Failed to determine a suitable driver class

原因在于刚建起文件夹想要启动SpringBoot，但其实没有正确配置数据源，想要立刻成功启动可以尝试以下3种独立解决方法（不要一起用）：

a. 若不想用数据库，在application.properties中添加禁用数据源

代码块

```
1 spring.autoconfigure.exclude=org.springframework.boot.autoconfigure.jdbc.DataSourceAutoConfiguration
```

b. 若使用mysql等数据库，添加数据库依赖

代码块

```
1 spring.datasource.url=jdbc:mysql://localhost:3306/your_database
2 spring.datasource.username=your_username
3 spring.datasource.password=your_password
4 spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
```

c. 若使用Mybatis依赖（搭建步骤有过添加，还需新建mapper包等），需要在启动类上添加

代码块

```
1 @MapperScan("com.example.demo21.mapper") // 替换为你的 Mapper 接口所在包
2 @SpringBootApplication
3 public class Demo21Application {
4     // ...
```


4. 新建.yml文件

properties和yml是类似的，但properties更原始，有没有更优雅的yml文件方法呢？

有的兄弟有的。方法参考：<https://blog.csdn.net/English221/article/details/129347883>

5. 数据库配置查询时报错

Whitelabel Error Page

This application has no explicit mapping for /error, so you are seeing this as a fallback.

Wed Jun 25 18:59:44 CST 2025

There was an unexpected error (type=Not Found, status=404).或者500

对于404报错，是系统未找到端点。按顺序检查main前的扫描、Controller、Mapper

对于500报错，系统已找到端点，但未捕获，可能是数据库配置有误，检查账号密码、表名等，可以从控制台复制信息给ai排错