**Springboot项目持久层Mybatis+Druid集成**

# 说明

版权归属：鑫学猿公司

微信：youhouduanyanfa

Java开发工程师转行辅导体系,保障拿到java工程师offer

图形用户界面, 文本, 应用程序

描述已自动生成

**背景**

**Mybatis**

Mybatis作为最常用的持久层框架，虽然Spring自己的data-jpa也很优秀，但是mybatis目前在行业里还是最常用的

Springboot整合Mybatis的方式有两种，一种是xml还有一种是注解的方式。个人建议在 SQL 语句不太长、比较简单的情况下，注解的方式比较简单、清晰；但是如果我们经常需要自己编写一些复杂的 SQL 建议使用xml的方式，不会和代码冗余在一起。

**Druid**

Druid是Java语言中最好的数据库连接池。Druid能够提供强大的监控和扩展功能

**集成教程**

1. 添加pom依赖

|  |
| --- |
| XML <!-- mysql驱动 --> <dependency>  <groupId>mysql</groupId>  <artifactId>mysql-connector-java</artifactId>  <version>8.0.25</version> </dependency>  <!-- mybatis依赖 --> <dependency>  <groupId>org.mybatis.spring.boot</groupId>  <artifactId>mybatis-spring-boot-starter</artifactId>  <version>1.3.1</version> </dependency>  <!-- druid依赖 --> <dependency>  <groupId>com.alibaba</groupId>  <artifactId>druid-spring-boot-starter</artifactId>  <version>1.1.10</version> </dependency> |

1. 配置application.properties文件

|  |
| --- |
| Properties spring.datasource.druid.stat-view-servlet.enabled=true #数据库连接 spring.datasource.druid.url=jdbc:mysql://localhost:3306/test\_db?useSSL=false&useUnicode=true&characterEncoding=utf-8&allowMultiQueries=true&autoReconnect=true&failOverReadOnly=false&maxReconnects=10 spring.datasource.druid.username=root spring.datasource.druid.password=cxxxxx8 spring.datasource.druid.driver-class-name=com.mysql.jdbc.Driver # 初始化时建立物理连接的个数 spring.datasource.druid.initial-size=5 # 最大连接池数量 spring.datasource.druid.max-active=30 # 最小连接池数量 spring.datasource.druid.min-idle=5 # 获取连接时最大等待时间，单位毫秒 spring.datasource.druid.max-wait=60000 # 配置间隔多久才进行一次检测，检测需要关闭的空闲连接，单位是毫秒 spring.datasource.druid.time-between-eviction-runs-millis=60000 # 连接保持空闲而不被驱逐的最小时间 spring.datasource.druid.min-evictable-idle-time-millis=300000 # 用来检测连接是否有效的sql，要求是一个查询语句 spring.datasource.druid.validation-query=SELECT 1 FROM DUAL # 建议配置为true，不影响性能，并且保证安全性。申请连接的时候检测，如果空闲时间大于timeBetweenEvictionRunsMillis，执行validationQuery检测连接是否有效。 spring.datasource.druid.test-while-idle=true # 申请连接时执行validationQuery检测连接是否有效，做了这个配置会降低性能。 spring.datasource.druid.test-on-borrow=false # 归还连接时执行validationQuery检测连接是否有效，做了这个配置会降低性能。 spring.datasource.druid.test-on-return=false # 是否缓存preparedStatement，也就是PSCache。PSCache对支持游标的数据库性能提升巨大，比如说oracle。在mysql下建议关闭。 spring.datasource.druid.pool-prepared-statements=true # 要启用PSCache，必须配置大于0，当大于0时，poolPreparedStatements自动触发修改为true。 spring.datasource.druid.max-pool-prepared-statement-per-connection-size=50 # 配置监控统计拦截的filters，去掉后监控界面sql无法统计 spring.datasource.druid.filters=stat,wall # 通过connectProperties属性来打开mergeSql功能；慢SQL记录 spring.datasource.druid.connection-properties=druid.stat.mergeSql=true;druid.stat.slowSqlMillis=500 # 合并多个DruidDataSource的监控数据 spring.datasource.druid.use-global-data-source-stat=true  #Mybatis配置 #xml文件的位置 #这里注意下，如果是多模块项目需要在classpath后面加个\*，这样才会去扫描每个模块的resources mybatis.mapper-locations=classpath\*:mapper/\*.xml #PO的位置 mybatis.type-aliases-package=com.xyy.scaffolddao.dataobject #驼峰命名 mybatis.configuration.map-underscore-to-camel-case=true |

1. 启动类加入mapper包扫描

|  |
| --- |
| Java @SpringBootApplication @MapperScan({"com.xyy.scaffolddao.mapper"}) public class ScaffoldWebApplication {  public static void main(String[] args) {  SpringApplication.run(ScaffoldWebApplication.class, args);  }  } |

1. PO类

|  |
| --- |
| Java @Data public class UserDO {  private Long id;   private String name; } |

1. Mapper类

|  |
| --- |
| Java @Mapper public interface UserMapper {  /\*\*  \* 根据姓名查找  \*  \* @param name  \* @return  \*/  UserDO findUserByName(String name);   /\*\*  \* 根据id查找  \*  \* @param id  \* @return  \*/  @Select("SELECT \* FROM user WHERE id = #{id}")  UserDO findUserById(@Param("id") Long id);   /\*\*  \* 插入用户  \*  \* @param name  \*/  @Insert("INSERT INTO user (name) VALUES (#{name})")  void insertUser(@Param("name") String name); } |

1. mapper.xml文件

|  |
| --- |
| XML <?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.xyy.scaffolddao.mapper.UserMapper">   <select id="findUserByName" parameterType="String" resultType="com.xyy.scaffolddao.dataobject.UserDO">  SELECT  id, name  FROM user  WHERE name = #{name}  </select> </mapper> |

1. controller类

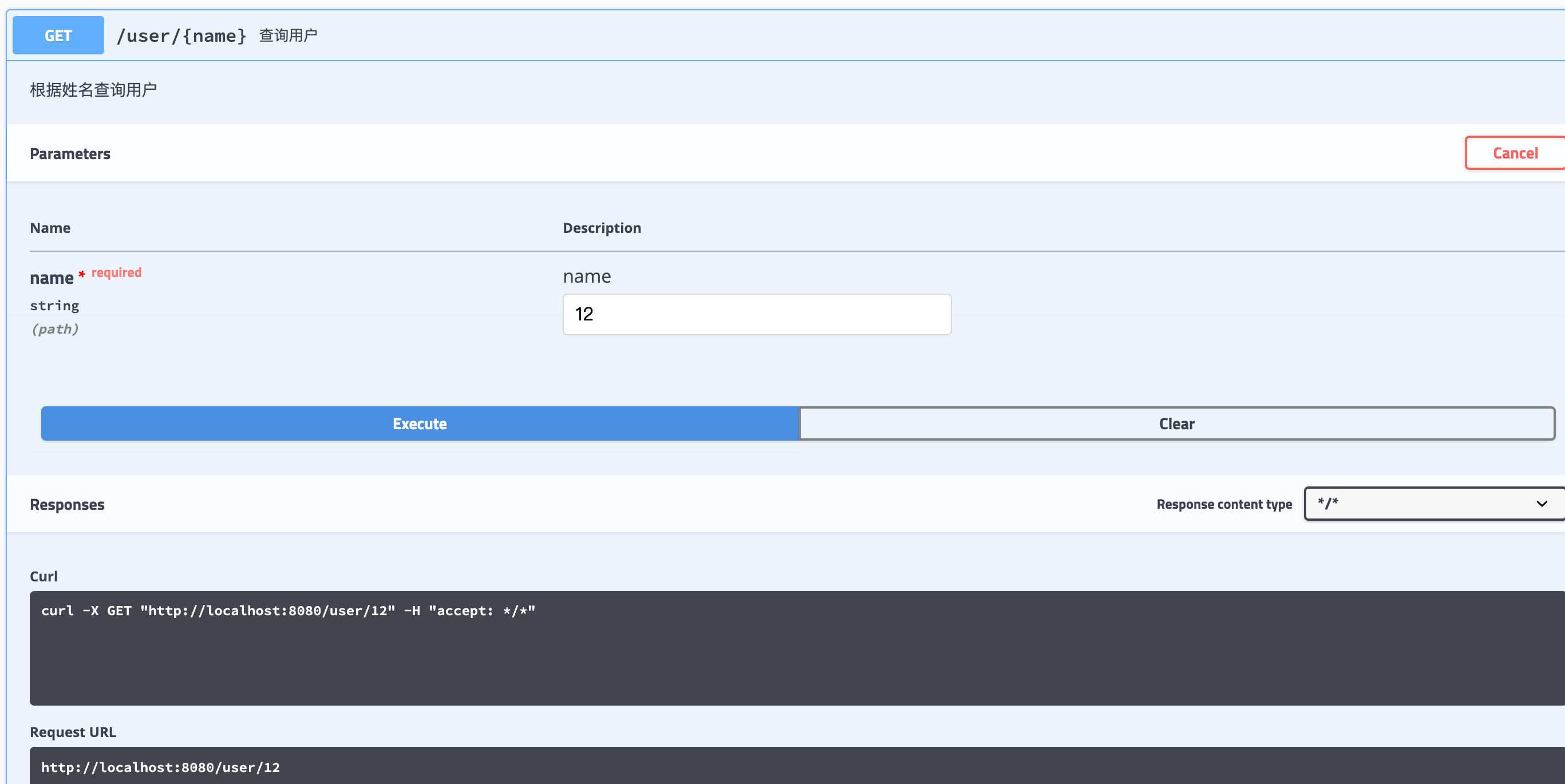
|  |
| --- |
| Java @RestController @Api(tags = "用户数据接口") public class UserController {  @Resource  private UserService userService;   @ApiOperation(value = "查询用户", notes = "根据姓名查询用户")  @GetMapping("/user/{name}")  public String queryUserByName(@PathVariable String name) {  UserDO user = userService.queryUserByName(name);  if (Objects.isNull(user)) {  return "未查询到用户：" + name;  }  return "查找的用户为：" + user.getName();  }   @ApiOperation(value = "新增用户", notes = "根据用户名和ID添加新用户")  @ApiImplicitParams({  @ApiImplicitParam(paramType = "query", name = "userName", value = "用户名",  required = true),  @ApiImplicitParam(paramType = "query", name="userID", value = "用户ID",  required = true)  })  @PostMapping("/user")  public String addUser(@RequestParam String userName, @RequestParam Integer userID) {  userService.createUser(userName);  return "新增用户：" + userName + "，ID：" + userID;  }   @ApiOperation(value = "修改用户信息", notes = "更新用户信息")  @PutMapping("/user")  public String updateUser(@RequestBody UserDTO userDTO) {  return userDTO.toString();  } } |

1. Service类

|  |
| --- |
| Java public interface UserService {  void createUser(String name);   UserDO queryUserByName(String name); } |

|  |
| --- |
| Java @Service public class UserServiceImpl implements UserService {  @Autowired  private UserMapper userMapper;   @Override  public void createUser(String name) {  userMapper.insertUser(name);  }   @Override  public UserDO queryUserByName(String name) {  return userMapper.findUserByName(name);  } } |

1. 通过Swagger发起测试



**课程代码**