# SpringBoot高级

## 代理对象方式

### JDK动态代理

说明:是java原生提供的代理方式.

代理对象的特点:使用起来和真实的对象一模一样.

规则:要求被代理者(真是class)必须实现接口.否则代理对象没有创建的模板.

### CGLIB动态代理

说明:cglib代理是第三方提供的.使用时必须导包.

规则:cglib创建代理对象时被代理者可以**不**实现接口.

cglib创建的代理对象是目标对象的子类.

### spring框架关于动态代理说明

默认规则:

如果需要为被代理者创建代理对象.

如果被代理者是接口/实现了接口则统一采用JDK动态代理.

如果被代理者不是接口/没有实现接口,则采用cglib动态代理方式.

## Mybatis-plus介绍

### 官网介绍

[MyBatis-Plus](https://github.com/baomidou/mybatis-plus)（简称 MP）是一个 [MyBatis](http://www.mybatis.org/mybatis-3/) 的增强工具，在 MyBatis 的基础上**只做增强不做改变**，为简化开发、提高效率而生。

### 持久层框架的发展

1. JDBC操作数据库. 代码编辑繁琐.封装不便.
2. 数据库链接池 不需要频繁开关链接.
3. JPA思想:

JPA是Java Persistence API的简称，中文名**Java持久层API**，是JDK 5.0注解或XML**描述对象－关系表的映射关系**，**并将运行期的实体**[**对象持久化**](https://baike.baidu.com/item/%E5%AF%B9%E8%B1%A1%E6%8C%81%E4%B9%85%E5%8C%96/7316192)**到数据库中**.

以面向对象的方式操作数据库.

1. hibernate

提出了全自动的ORM映射关系.

4.1用户真正意义上实现了面向对象的操作数据库

userDao.insert(user); 底层实现全部由框架完成.

4.2 查询的结果集能够实现自动化的封装映射.

缺点:

1. 配置繁琐. 如果是单表映射操作简易.但是如果是多表关联映射时需要编辑复杂的配置文件.
2. 由全部都是面向对象的,所以内存耗费较高.
3. 由于都是通过反射操作.所以当遇到海量数据时性能低. 500万
4. 如果使用多表关联时会有很多冗余的sql语句.

select \* from id=xxxxx

insert into xxxxx

5.mybatis

半自动的ORM映射框架

1. 程序员自己编辑sql.
2. 可以实现自动化的结果集映射.

缺点:

无论多么简单的sql都需要自己手写!!!!.

6.mybatisplus

全方位的补充了mybaits.用户实现**单表**CURD不写sql.

## Mybatis-plus底层实现原理

需求:单表CRUD不写sql,以面向对象的方式操作.

### 实现步骤

1. POJO对象必须与表一一映射.
2. POJO中的属性与表中字段一一映射.
3. 架构师负责编辑第三方接口,第三方接口中包含了常用CURD方法.

使用者只需要**继承**接口即可.

1. 将公共的接口方法最终转化为sql语句(第三方架构完成).

例子1:

**insert into** **表名**(**字段名称…..**) **values**(**对应的数据….**);

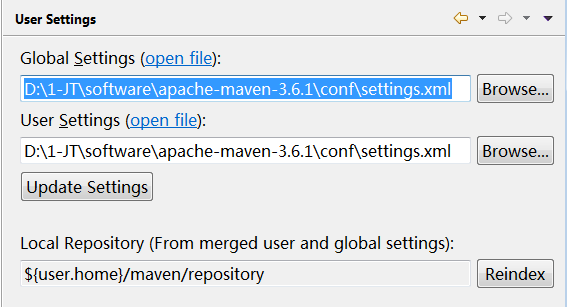
例子2:

**select count(\*) from** **表名**;

## Mybatis-plus入门案例

### 切换镜像

1. 查找maven settings文件位置



1. 添加达内镜像文件

<mirror>

<id>nexus</id>

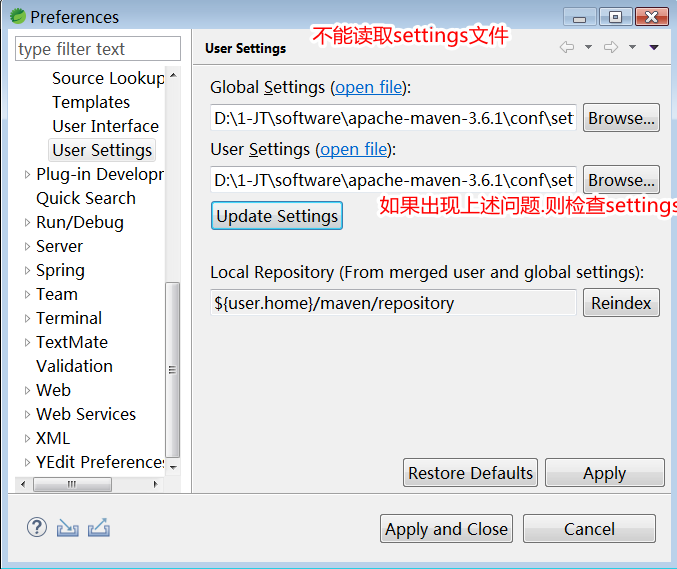
<name>Tedu Maven</name>

<mirrorOf>\*</mirrorOf>

<url>http://maven.tedu.cn/nexus/content/groups/public/</url>

</mirror>

1. 如果报错解决方案



### 引入jar包

<!--spring整合mybatis-plus -->

<dependency>

<groupId>com.baomidou</groupId>

<artifactId>mybatis-plus-boot-starter</artifactId>

<version>3.0.6</version>

</dependency>

### 编辑POJO对象

@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

@TableName//(value = "user")//将对象与表一一映射

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

/\*\*

\* 规则:

\* 1.如果表名与对象名一致可以省略

\* 2.如果表中字段与属性名称一致可以省略配置

\*/

@TableId(type = IdType.***AUTO***)//属性与表主键关联

**private** Integer id;

//@TableField(value ="user")

**private** String name;

**private** Integer age;

**private** String sex;

}

### 继承指定接口

//接口的定义 没有实现类

//@Mapper //将接口交给spring容器管理

//创建代理对象 JDK/CGLIB

**public** **interface** UserMapper **extends** **BaseMapper**<User>{

//编辑接口方法 查询全部数据

//List<User> findAll();

}

### Spring整合MybatisPlus

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-plush配置

**mybatis-plus:**

#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

## Mybatis-plus操作

### 新增操作

业务需求:要求用户new一个user对象.之后利用操作对象的方式入库.

/\*\*

\* SpringBoot测试类写法

\* **@author** Administrator

\* 规则:

\* SpringBoot自己封装了测试类写法.可以直接从容器

\* 对象中获取对象.

\*

\*/

@SpringBootTest

@RunWith(SpringRunner.**class**)

**public** **class** TestMybatisPlus {

/\*\*

\* 旧版本写法

\*/

/\*

\* @Test public void testOld() { //1.获取容器对象 ApplicationContext context = new

\* ClassPathXmlApplicationContext("spring配置文件.xml"); //2.从容器中获取对象 UserMapper

\* userMapper = context.getBean(UserMapper.class); //3.利用对象操作数据

\* userMapper.selectList(null); }

\*/

@Autowired

**private** UserMapper userMapper;

//1.新增用户

@Test

**public** **void** insertUser() {

User user = **new** User();

user.setName("1903班").setAge(4).setSex("都有");

userMapper.insert(user);

System.***out***.println("mybatisplus操作成功!!!!");

}

}

### 查询操作

//2.查询操作

/\*\*

\* queryWrapper表示条件构造器

\* 动态拼接where name="1903班"

\* 对象中不为null属性充当where条件

\*/

@Test

**public** **void** select\_Id\_count() {

//根据Id查询

User user = userMapper.selectById(53);

System.***out***.println(user);

//查询全部记录数

**int** count = userMapper.selectCount(**null**);

System.***out***.println("获取记录总数为:"+count);

//查询全部的女性记录数 sex="女"

//1.面向对象方式

User userCount = **new** User();

userCount.setSex("女");

QueryWrapper<User> queryWrapper =

**new** QueryWrapper<>(userCount);

**int** sexCount = userMapper.selectCount(queryWrapper);

System.***out***.println("获取记录总数:"+sexCount);

}

### 查询操作--条件构造器用法

/\*\*

\* 条件查询

\* 要求:

\* 1.age>18岁

\*

\* >gt <lt =eq

\* >=ge <=le

\*/

@Test

**public** **void** selectUser2() {

QueryWrapper<User> queryWrapper =

**new** QueryWrapper<>();

queryWrapper.gt("age", 18);

**int** count = userMapper.selectCount(queryWrapper);

System.***out***.println("age>18岁人数:"+count);

}

//2.age>=18 and sex=女

@Test

**public** **void** selectUser3() {

QueryWrapper<User> queryWrapper =

**new** QueryWrapper<User>();

queryWrapper.ge("age", 18)

.eq("sex", "女");

**int** count = userMapper.selectCount(queryWrapper);

System.***out***.println("数量:"+count);

}

### 添加日志

说明:编辑yml配置文件

#添加日志

logging:

level:

com.jt.mapper: debug

### 更新操作

/\*\*

\* 条件:将id为53的数据修改名称为如花 性别为女

\* mybatisplus更新操作

\* sql:

\* update user set name=如花,sex="女"

\* where id = 53

\*

\* 注意事项: 如果确定主键 可以使用byId方法更新.

\*

\* 2:没有id的更新操作语法

\*

\* entity:需要修改的数据.

\* updateWrapper: where条件

\* userMapper.update(entity, updateWrapper);

\*/

@Test

**public** **void** updateUser() {

//1.如果根据id进行更新.则id字段自动充当where条件

//User user = new User();

//user.setId(53).setName("如花")

//.setAge(120).setSex("女");

//userMapper.updateById(user);

/\*\*

\* 2.根据名称修改数据

\* name=如花的数据

\* name改为不知火舞 age=18

\*/

User tempUser = **new** User();

tempUser.setName("不知火舞")

.setAge(18);

UpdateWrapper<User> updateWrapper =

**new** UpdateWrapper<>();

updateWrapper.eq("name","如花");

userMapper.update(tempUser, updateWrapper);

}

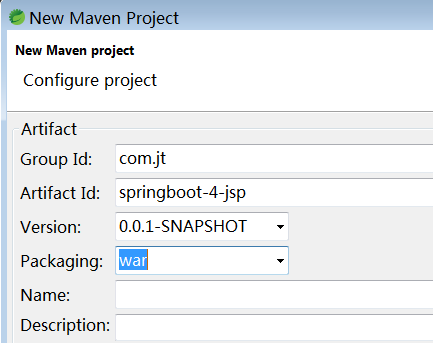
## SpringBoot整合JSP

### 前提说明

SpringBoot原生支持HTML页面.要求页面必须按照springBoot形式进行保存 static

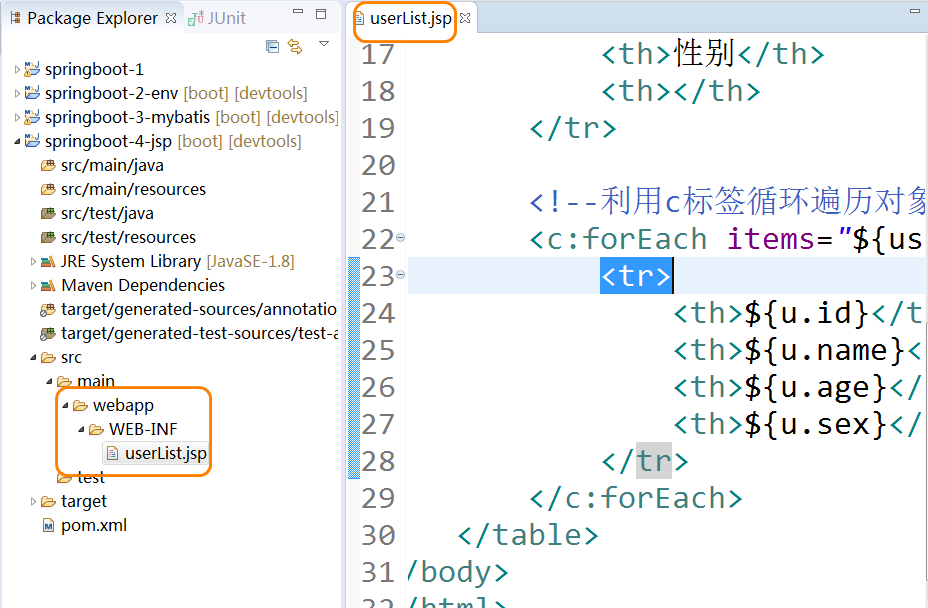
springboot原生不支持jsp.但是工作中有时需要使用.需要额外的添加jar包支持.

### 创建项目



将mybatis中pom.xml文件拷贝.导入新项目

### 导入静态资源文件



### 添加jar包

在原有基础之上.添加新的jar包

<!--springBoot整合JSP添加依赖 -->

<!--servlet依赖 -->

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>javax.servlet-api</artifactId>

</dependency>

<!--jstl依赖 -->

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>jstl</artifactId>

</dependency>

<!--使jsp页面生效 -->

<dependency>

<groupId>org.apache.tomcat.embed</groupId>

<artifactId>tomcat-embed-jasper</artifactId>

</dependency>

### 编辑YML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

mvc: #引入mvn配置

view:

prefix: /WEB-INF/ # /默认代表根目录 src/main/webapp

suffix: .jsp

#mybatis-plush配置

mybatis-plus:

#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

#添加日志

logging:

level:

com.jt.mapper: debug

### 实现页面用户信息展现

业务说明:

当用户通过localhost:8090/findAll请求时,要求跳转到userList.jsp页面.同时查询全部的user信息.之后进行页面展现.

#### 编辑主启动类

@SpringBootApplication

@MapperScan("com.jt.mapper")

**public** **class** SpringBootRun {

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

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

}

}

#### 编辑POJO对象

@Data

@Accessors(chain = **true**)

@TableName

**public** **class** User **implements** Serializable{

//序列化问题: 对象转化与字节数组互转时必须序列化

//定义POJO对象时一定使用包装类型

@TableId(type = IdType.***AUTO***) //主键自增

**private** Integer id;

**private** String name;

**private** Integer age;

**private** String sex;

}

#### 编辑Mapper接口/mapper映射文件

1.定义接口

**public** **interface** UserMapper **extends** BaseMapper<User>{

//单表CURD操作 几乎不写sql;

}

2.定义mapper映射文件

<?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"*>

<!--留着是为了以后扩展方便 -->

</mapper>

#### 编辑Service接口/实现类

@Service

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

@Autowired

**private** UserMapper userMapper;

@Override

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

**return** userMapper.selectList(**null**);

}

}

#### 编辑Controller

@Controller

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

@Autowired

**private** UserService userService;

//url:localhost:8090/user/findAll

@RequestMapping("/findAll")

**public** String userList(Model model) {

List<User> userList =

userService.findAll();

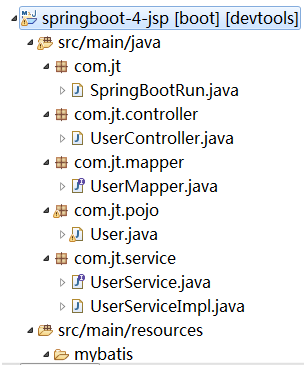
model.addAttribute("userList", userList);

**return** "userList"; //页面逻辑名称

}

}

#### 代码整体结构



### 作业

1. 当用户访问localhost:8090/userList,要求直接跳转userList.jsp页面
2. 利用ajax重新发起请求localhost:8090/findAll.获取userListJSON数据.之后手写js解析数据页面展现用户信息.