

课程介绍

- 了解Mybatis-Plus
- 整合Mybatis-Plus
- 通用CRUD
- Mybatis-Plus的配置
- 条件构造器

1、了解Mybatis-Plus

1.1、Mybatis-Plus介绍

MyBatis-Plus (简称 MP) 是一个 MyBatis 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。

官网：<https://mybatis.plus/> 或 <https://mp.baomidou.com/>



MyBatis-Plus

为简化开发而生

快速开始 →

润物无声 只做增强不做改变，引入它不会对现有工程产生影响，如丝般顺滑。	效率至上 只需简单配置，即可快速进行 CRUD 操作，从而节省大量时间。	丰富功能 热加载、代码生成、分页、性能分析等功能一应俱全。
---	--	---

愿景

我们的愿景是成为 MyBatis 最好的搭档，就像 魂斗罗 中的 1P、2P，基友搭配，效率翻倍。



TO BE THE BEST PARTNER OF MYBATIS

1.2、代码以及文档

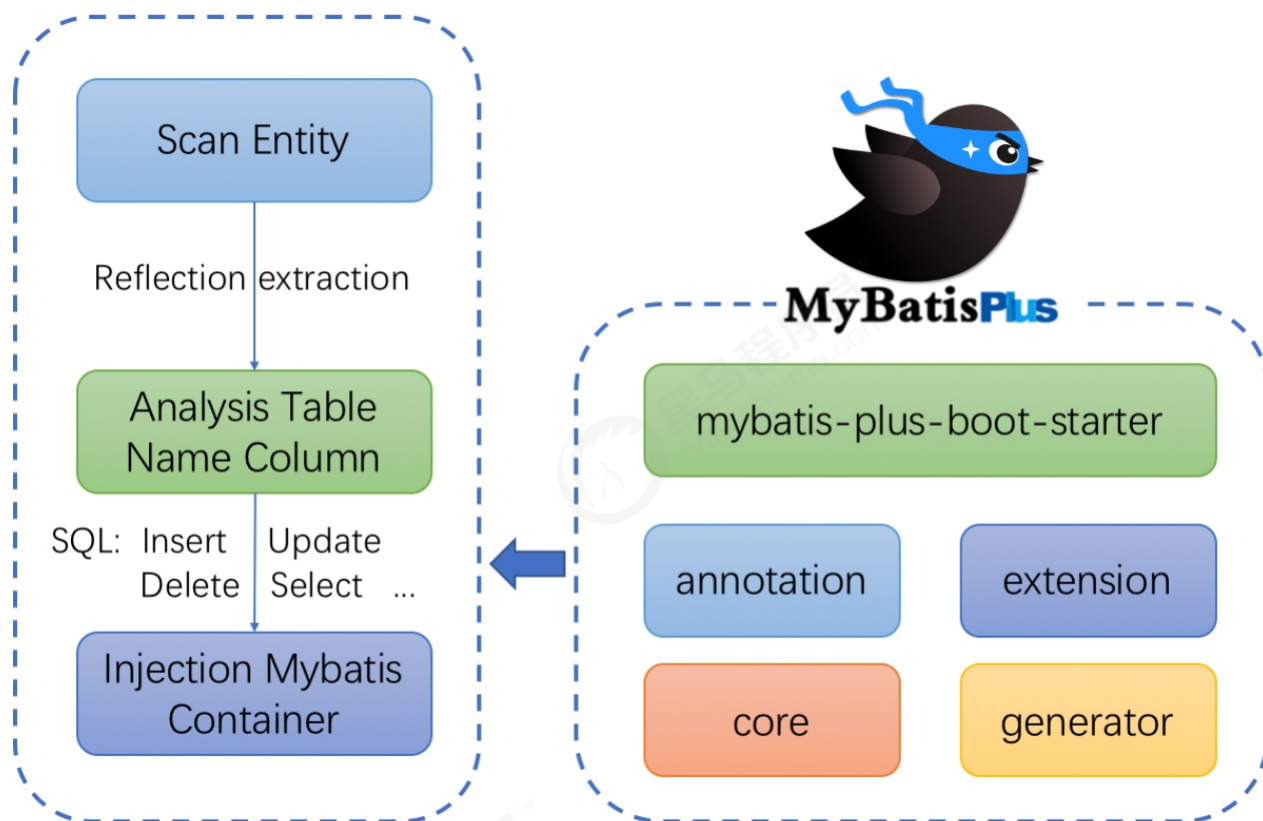
文档地址：<https://mybatis.plus/guide/>

源码地址：<https://github.com/baomidou/mybatis-plus>

1.3、特性

- **无侵入**：只做增强不做改变，引入它不会对现有工程产生影响，如丝般顺滑
- **损耗小**：启动即会自动注入基本 CURD，性能基本无损耗，直接面向对象操作
- **强大的 CRUD 操作**：内置通用 Mapper、通用 Service，仅仅通过少量配置即可实现单表大部分 CRUD 操作，更有强大的条件构造器，满足各类使用需求
- **支持 Lambda 形式调用**：通过 Lambda 表达式，方便的编写各类查询条件，无需再担心字段写错
- **支持多种数据库**：支持 MySQL、MariaDB、Oracle、DB2、H2、HSQL、SQLite、Postgre、SQLServer2005、SQLServer 等多种数据库
- **支持主键自动生成**：支持多达 4 种主键策略（内含分布式唯一 ID 生成器 - Sequence），可自由配置，完美解决主键问题
- **支持 XML 热加载**：Mapper 对应的 XML 支持热加载，对于简单的 CRUD 操作，甚至可以无 XML 启动
- **支持 ActiveRecord 模式**：支持 ActiveRecord 形式调用，实体类只需继承 Model 类即可进行强大的 CRUD 操作
- **支持自定义全局通用操作**：支持全局通用方法注入（Write once, use anywhere）
- **支持关键词自动转义**：支持数据库关键词（order、key.....）自动转义，还可自定义关键词
- **内置代码生成器**：采用代码或者 Maven 插件可快速生成 Mapper、Model、Service、Controller 层代码，支持模板引擎，更有超多自定义配置等您来使用
- **内置分页插件**：基于 MyBatis 物理分页，开发者无需关心具体操作，配置好插件之后，写分页等同于普通 List 查询
- **内置性能分析插件**：可输出 Sql 语句以及其执行时间，建议开发测试时启用该功能，能快速揪出慢查询
- **内置全局拦截插件**：提供全表 delete、update 操作智能分析阻断，也可自定义拦截规则，预防误操作
- **内置 Sql 注入剥离器**：支持 Sql 注入剥离，有效预防 Sql 注入攻击

1.4、架构



1.5、作者

Mybatis-Plus是由baomidou (苞米豆) 组织开发并且开源的，目前该组织大概有30人左右。

码云地址：<https://gitee.com/organizations/baomidou>

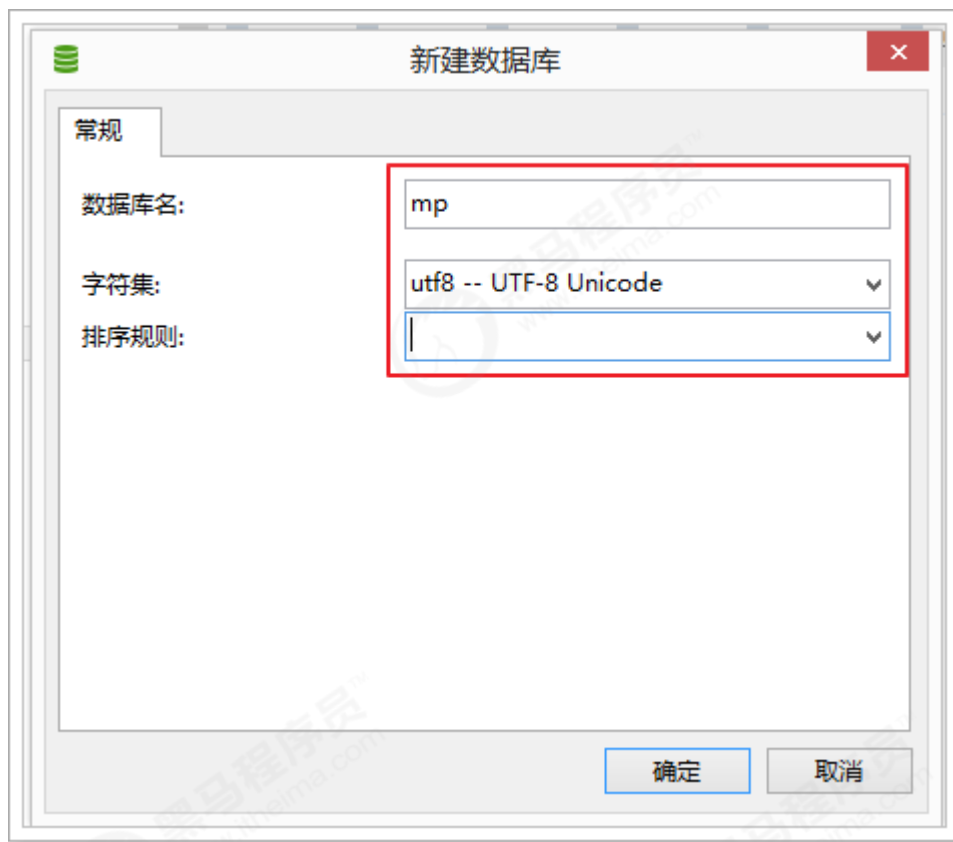
The screenshot shows the Gitee organization page for 'baomidou' (苞米豆). The page header includes the Gitee logo, navigation links (开源软件, 企业版, 高校版, 博客), a search bar, and login/register buttons. The organization's profile shows a corn icon, the name 'baomidou', and the website 'http://mp.baomidou.co...'. Below this, it shows '仓库 23', '动态', and '成员 30'. The '热门项目' (Popular Projects) section lists several projects:

- mybatis-plus** (GVP): mybatis 增强工具包，简化 CRUD 操作。文档 http://mp.baomidou.com. 1677 stars, 5135 forks, 1689 watchers.
- mybatis-plus-samples**: MyBatis-Plus Samples 文档. 63 stars, 256 forks, 93 watchers.
- mybatis-plus-doc**: Mybatis-Plus 文档 http://mp.baomidou.com/. 15 stars, 33 forks, 21 watchers.
- mybatis-3**: mybatis-3 china 分支. 13 stars, 22 forks, 3 watchers.
- dynamic-datasource-spring-boot-starter**: 基于 SpringBoot 多数据源 动态数据源 主从分离 快速启动器. 212 stars, 894 forks, 194 watchers.
- kisso** (GVP): java 基于 Cookie 的 SSO 中间件 kisso. 557 stars, 1547 forks, 581 watchers.

2、快速开始

对于Mybatis整合MP有常常有三种用法，分别是Mybatis+MP、Spring+Mybatis+MP、Spring Boot+Mybatis+MP。

2.1、创建数据库以及表



```
1  -- 创建测试表
2  CREATE TABLE `tb_user` (
3    `id` bigint(20) NOT NULL AUTO_INCREMENT COMMENT '主键ID',
4    `user_name` varchar(20) NOT NULL COMMENT '用户名',
5    `password` varchar(20) NOT NULL COMMENT '密码',
6    `name` varchar(30) DEFAULT NULL COMMENT '姓名',
7    `age` int(11) DEFAULT NULL COMMENT '年龄',
8    `email` varchar(50) DEFAULT NULL COMMENT '邮箱',
9    PRIMARY KEY (`id`)
10 ) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8;
11
12 -- 插入测试数据
13 INSERT INTO `tb_user` (`id`, `user_name`, `password`, `name`, `age`, `email`) VALUES
14 ('1', 'zhangsan', '123456', '张三', '18', 'test1@itcast.cn');
15 INSERT INTO `tb_user` (`id`, `user_name`, `password`, `name`, `age`, `email`) VALUES
16 ('2', 'lisi', '123456', '李四', '20', 'test2@itcast.cn');
17 INSERT INTO `tb_user` (`id`, `user_name`, `password`, `name`, `age`, `email`) VALUES
18 ('3', 'wangwu', '123456', '王五', '28', 'test3@itcast.cn');
19 INSERT INTO `tb_user` (`id`, `user_name`, `password`, `name`, `age`, `email`) VALUES
20 ('4', 'zhao Liu', '123456', '赵六', '21', 'test4@itcast.cn');
21 INSERT INTO `tb_user` (`id`, `user_name`, `password`, `name`, `age`, `email`) VALUES
22 ('5', 'sunqi', '123456', '孙七', '24', 'test5@itcast.cn');
```

2.2、创建工程



New Project

GroupId: cn.itcast.mp ☒ Inherit

ArtifactId: itcast-mybatis-plus

Version: 1.0-SNAPSHOT ☒ Inherit

Previous Next Cancel Help

导入依赖：

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0"
3         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
5     <modelVersion>4.0.0</modelVersion>
6
7     <groupId>cn.itcast.mp</groupId>
8     <artifactId>itcast-mybatis-plus</artifactId>
9     <version>1.0-SNAPSHOT</version>
10    <modules>
11        <module>itcast-mybatis-plus-simple</module>
12    </modules>
13    <packaging>pom</packaging>
14
15    <dependencies>
16        <!-- mybatis-plus插件依赖 -->
17        <dependency>
18            <groupId>com.baomidou</groupId>
19            <artifactId>mybatis-plus</artifactId>
20            <version>3.1.1</version>
21        </dependency>
22        <!-- MySql -->
```



```
23     <dependency>
24         <groupId>mysql</groupId>
25         <artifactId>mysql-connector-java</artifactId>
26         <version>5.1.47</version>
27     </dependency>
28     <!-- 连接池 -->
29     <dependency>
30         <groupId>com.alibaba</groupId>
31         <artifactId>druid</artifactId>
32         <version>1.0.11</version>
33     </dependency>
34     <!-- 简化bean代码的工具包 -->
35     <dependency>
36         <groupId>org.projectlombok</groupId>
37         <artifactId>lombok</artifactId>
38         <optional>true</optional>
39         <version>1.18.4</version>
40     </dependency>
41     <dependency>
42         <groupId>junit</groupId>
43         <artifactId>junit</artifactId>
44         <version>4.12</version>
45     </dependency>
46     <dependency>
47         <groupId>org.slf4j</groupId>
48         <artifactId>slf4j-log4j12</artifactId>
49         <version>1.6.4</version>
50     </dependency>
51 </dependencies>
52
53 <build>
54     <plugins>
55         <plugin>
56             <groupId>org.apache.maven.plugins</groupId>
57             <artifactId>maven-compiler-plugin</artifactId>
58             <configuration>
59                 <source>1.8</source>
60                 <target>1.8</target>
61             </configuration>
62         </plugin>
63     </plugins>
64 </build>
65
66 </project>
```

2.3、Mybatis + MP

下面演示，通过纯Mybatis与Mybatis-Plus整合。

2.3.1、创建子Module

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0"
```



```

3      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4      xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
5      <parent>
6          <artifactId>itcast-mybatis-plus</artifactId>
7          <groupId>cn.itcast.mp</groupId>
8          <version>1.0-SNAPSHOT</version>
9      </parent>
10     <modelVersion>4.0.0</modelVersion>
11     <packaging>jar</packaging>
12
13     <artifactId>itcast-mybatis-plus-simple</artifactId>
14
15 </project>

```

log4j.properties :

```

1 log4j.rootLogger=DEBUG,A1
2
3 log4j.appender.A1=org.apache.log4j.ConsoleAppender
4 log4j.appender.A1.layout=org.apache.log4j.PatternLayout
5 log4j.appender.A1.layout.ConversionPattern=[%t] [%c] - [%p] %m%n

```

2.3.2、Mybatis实现查询User

第一步，编写mybatis-config.xml文件：

```

1 <?xml version="1.0" encoding="UTF-8" ?>
2 <!DOCTYPE configuration
3     PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
4     "http://mybatis.org/dtd/mybatis-3-config.dtd">
5 <configuration>
6     <environments default="development">
7         <environment id="development">
8             <transactionManager type="JDBC"/>
9             <dataSource type="POOLED">
10                 <property name="driver" value="com.mysql.jdbc.Driver"/>
11                 <property name="url" value="jdbc:mysql://127.0.0.1:3306/mp?
useUnicode=true&characterEncoding=utf8&autoReconnect=true&allowMultiQuerie
s=true&useSSL=false"/>
12                 <property name="username" value="root"/>
13                 <property name="password" value="root"/>
14             </dataSource>
15         </environment>
16     </environments>
17     <mappers>
18         <mapper resource="UserMapper.xml"/>
19     </mappers>
20 </configuration>

```

第二步，编写User实体对象：（这里使用lombok进行了进化bean操作）



```
1 package cn.itcast.mp.simple.pojo;
2
3 import lombok.AllArgsConstructor;
4 import lombok.Data;
5 import lombok.NoArgsConstructor;
6
7 @Data
8 @NoArgsConstructor
9 @AllArgsConstructor
10 public class User {
11
12     private Long id;
13     private String userName;
14     private String password;
15     private String name;
16     private Integer age;
17     private String email;
18 }
19
```

第三步，编写UserMapper接口：

```
1 package cn.itcast.mp.simple.mapper;
2
3 import cn.itcast.mp.simple.pojo.User;
4
5 import java.util.List;
6
7 public interface UserMapper {
8
9     List<User> findAll();
10
11 }
12
```

第四步，编写UserMapper.xml文件：

```
1 <?xml version="1.0" encoding="UTF-8" ?>
2 <!DOCTYPE mapper
3     PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
4     "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
5 <mapper namespace="cn.itcast.mp.simple.mapper.UserMapper">
6
7     <select id="findAll" resultType="cn.itcast.mp.simple.pojo.User">
8         select * from tb_user
9     </select>
10
11 </mapper>
```

第五步，编写TestMybatis测试用例：



```
1 package cn.itcast.mp.simple;
2
3 import cn.itcast.mp.simple.mapper.UserMapper;
4 import cn.itcast.mp.simple.pojo.User;
5 import org.apache.ibatis.io.Resources;
6 import org.apache.ibatis.session.SqlSession;
7 import org.apache.ibatis.session.SqlSessionFactory;
8 import org.apache.ibatis.session.SqlSessionFactoryBuilder;
9 import org.junit.Test;
10
11 import java.io.InputStream;
12 import java.util.List;
13
14 public class TestMybatis {
15
16     @Test
17     public void testUserList() throws Exception{
18         String resource = "mybatis-config.xml";
19         InputStream inputStream = Resources.getResourceAsStream(resource);
20         SqlSessionFactory sqlSessionFactory = new
21         SqlSessionFactoryBuilder().build(inputStream);
22         SqlSession sqlSession = sqlSessionFactory.openSession();
23
24         UserMapper userMapper = sqlSession.getMapper(UserMapper.class);
25         List<User> list = userMapper.findAll();
26         for (User user : list) {
27             System.out.println(user);
28         }
29     }
30
31 }
32
```

测试结果：

```
1 [main] [cn.itcast.mp.simple.mapper.UserMapper.findAll]-[DEBUG] ==> Parameters:
2 [main] [cn.itcast.mp.simple.mapper.UserMapper.findAll]-[DEBUG] <==      Total: 5
3 User(id=1, userName=null, password=123456, name=张三, age=18, email=test1@itcast.cn)
4 User(id=2, userName=null, password=123456, name=李四, age=20, email=test2@itcast.cn)
5 User(id=3, userName=null, password=123456, name=王五, age=28, email=test3@itcast.cn)
6 User(id=4, userName=null, password=123456, name=赵六, age=21, email=test4@itcast.cn)
7 User(id=5, userName=null, password=123456, name=孙七, age=24, email=test5@itcast.cn)
```

2.3.3、Mybatis+MP实现查询User

第一步，将UserMapper继承BaseMapper，将拥有了BaseMapper中的所有方法：



```
1 package cn.itcast.mp.simple.mapper;
2
3 import cn.itcast.mp.simple.pojo.User;
4 import com.baomidou.mybatisplus.core.mapper.BaseMapper;
5
6 import java.util.List;
7
8 public interface UserMapper extends BaseMapper<User> {
9
10     List<User> findAll();
11
12 }
13
```

第二步，使用MP中的MybatisSqlSessionFactoryBuilder进行构建：

```
1 package cn.itcast.mp.simple;
2
3 import cn.itcast.mp.simple.mapper.UserMapper;
4 import cn.itcast.mp.simple.pojo.User;
5 import com.baomidou.mybatisplus.core.MybatisSqlSessionFactoryBuilder;
6 import org.apache.ibatis.io.Resources;
7
8 import org.apache.ibatis.session.SqlSession;
9 import org.apache.ibatis.session.SqlSessionFactory;
10 import org.apache.ibatis.session.SqlSessionFactoryBuilder;
11 import org.junit.Test;
12
13 import java.io.InputStream;
14 import java.util.List;
15
16 public class TestMybatisPlus {
17
18     @Test
19     public void testUserList() throws Exception{
20         String resource = "mybatis-config.xml";
21         InputStream inputStream = Resources.getResourceAsStream(resource);
22         //这里使用的是MP中的MybatisSqlSessionFactoryBuilder
23         SqlSessionFactory sqlSessionFactory = new
24         MybatisSqlSessionFactoryBuilder().build(inputStream);
25         SqlSession sqlSession = sqlSessionFactory.openSession();
26
27         UserMapper userMapper = sqlSession.getMapper(UserMapper.class);
28
29         // 可以调用BaseMapper中定义的方法
30         List<User> list = userMapper.selectList(null);
31         for (User user : list) {
32             System.out.println(user);
33         }
34     }
35 }
```

```
36 }  
37
```

运行报错：

```
ltParameterMap  
tting parameters  
assword,name,age,email FROM user  
tions.jdbc4.MySQLException: Table 'mp.user' doesn't exist  
  
ions.ExceptionFactory.wrapException(ExceptionFactory.java:30)  
n.defaults.DefaultSqlSession.selectList(DefaultSqlSession.java:150)  
n.defaults.DefaultSqlSession.selectList(DefaultSqlSession.java:141)  
.core.override.MybatisMapperMethod.executeForMany(MybatisMapperMethod.java:168)  
.core.override.MybatisMapperMethod.execute(MybatisMapperMethod.java:82)  
_core_override_MybatisMapperProxy.invoke(MybatisMapperProxy.java:61) <1 internal call
```

解决：在User对象中添加@TableName，指定数据库表名

```
@Data  
@NoArgsConstructor  
@AllArgsConstructor  
@TableName("tb_user")  
public class User {  
  
    private Long id;  
    private String userName;  
    private String password;  
    private String name;  
    private Integer age;  
    private String email;  
}
```

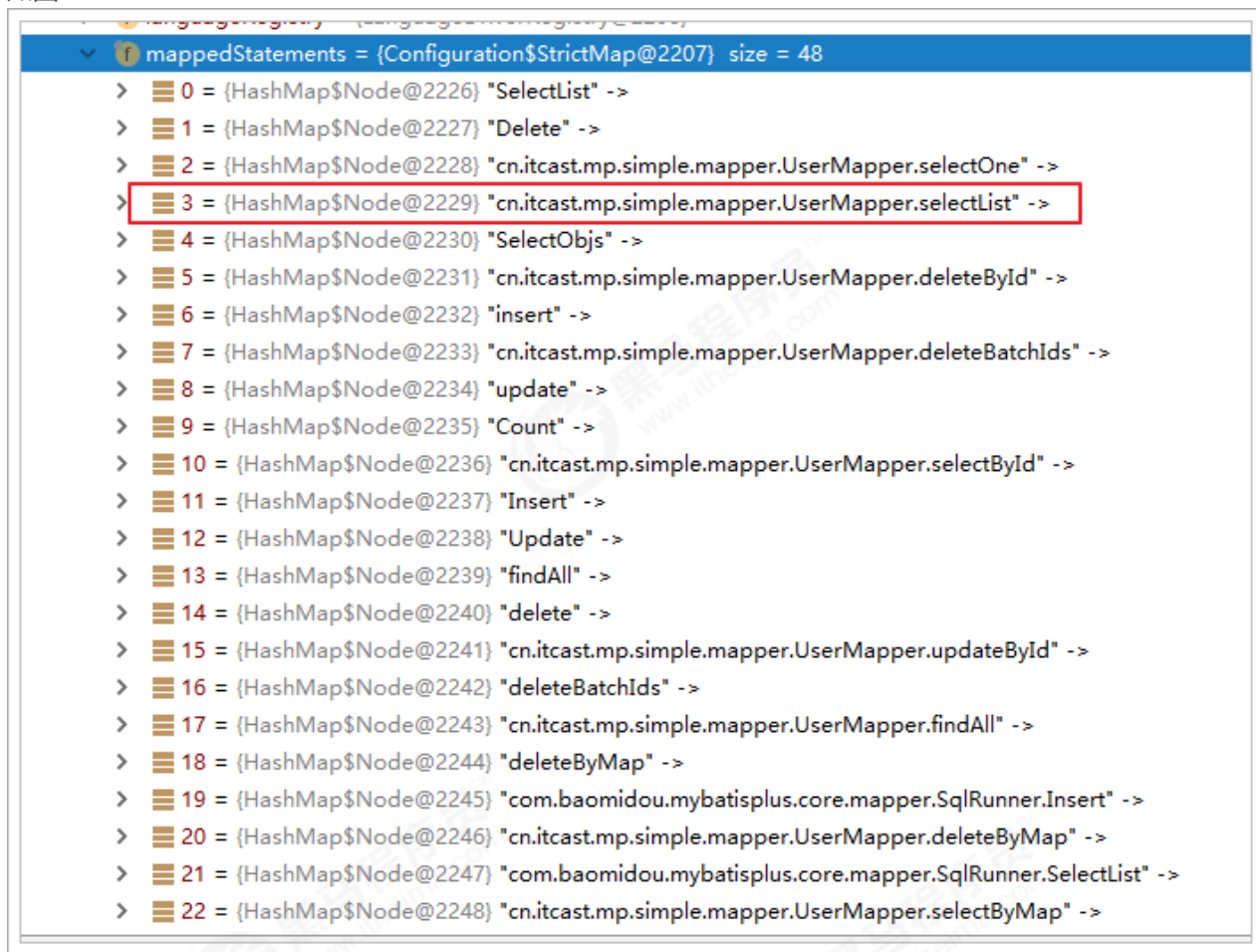
测试：

```
1 [main] [cn.itcast.mp.simple.mapper.UserMapper.selectList]-[DEBUG] ==> Preparing:  
SELECT id,user_name,password,name,age,email FROM tb_user  
2 [main] [cn.itcast.mp.simple.mapper.UserMapper.selectList]-[DEBUG] ==> Parameters:  
3 [main] [cn.itcast.mp.simple.mapper.UserMapper.selectList]-[DEBUG] <== Total: 5  
4 User(id=1, userName=zhangsan, password=123456, name=张三, age=18, email=test1@itcast.cn)  
5 User(id=2, userName=lisi, password=123456, name=李四, age=20, email=test2@itcast.cn)  
6 User(id=3, userName=wangwu, password=123456, name=王五, age=28, email=test3@itcast.cn)  
7 User(id=4, userName=zhaoliu, password=123456, name=赵六, age=21, email=test4@itcast.cn)  
8 User(id=5, userName=sunqi, password=123456, name=孙七, age=24, email=test5@itcast.cn)
```

简单说明：

- 由于使用了MybatisSqlSessionFactoryBuilder进行了构建，继承的BaseMapper中的方法就载入到了SqlSession中，所以就可以直接使用相关的方法；

• 如图：



2.4、Spring + Mybatis + MP

引入了Spring框架，数据源、构建等工作就交给了Spring管理。

2.4.1、创建子Module

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0"
3         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
5         http://maven.apache.org/xsd/maven-4.0.0.xsd">
6     <parent>
7         <artifactId>itcast-mybatis-plus</artifactId>
8         <groupId>cn.itcast.mp</groupId>
9         <version>1.0-SNAPSHOT</version>
10    </parent>
11    <modelVersion>4.0.0</modelVersion>
12
13    <artifactId>itcast-mybatis-plus-spring</artifactId>
14
15    <properties>
16        <spring.version>5.1.6.RELEASE</spring.version>
17    </properties>
```



```
18     <dependencies>
19         <dependency>
20             <groupId>org.springframework</groupId>
21             <artifactId>spring-webmvc</artifactId>
22             <version>${spring.version}</version>
23         </dependency>
24         <dependency>
25             <groupId>org.springframework</groupId>
26             <artifactId>spring-jdbc</artifactId>
27             <version>${spring.version}</version>
28         </dependency>
29         <dependency>
30             <groupId>org.springframework</groupId>
31             <artifactId>spring-test</artifactId>
32             <version>${spring.version}</version>
33         </dependency>
34     </dependencies>
35
36
37 </project>
```

2.4.2、实现查询User

第一步，编写jdbc.properties

```
1 jdbc.driver=com.mysql.jdbc.Driver
2 jdbc.url=jdbc:mysql://127.0.0.1:3306/mp?
  useUnicode=true&characterEncoding=utf8&autoReconnect=true&allowMultiQueries=true&useSSL
  =false
3 jdbc.username=root
4 jdbc.password=root
```

第二步，编写applicationContext.xml

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans xmlns="http://www.springframework.org/schema/beans"
3       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4       xmlns:context="http://www.springframework.org/schema/context"
5       xsi:schemaLocation="http://www.springframework.org/schema/beans
6         http://www.springframework.org/schema/beans/spring-beans.xsd
7         http://www.springframework.org/schema/context
8         http://www.springframework.org/schema/context/spring-context.xsd">
9
10    <context:property-placeholder location="classpath:*.properties"/>
11
12    <!-- 定义数据源 -->
13    <bean id="dataSource" class="com.alibaba.druid.pool.DruidDataSource"
14          destroy-method="close">
15        <property name="url" value="${jdbc.url}"/>
16        <property name="username" value="${jdbc.username}"/>
17        <property name="password" value="${jdbc.password}"/>
18        <property name="driverClassName" value="${jdbc.driver}"/>
```



```
19     <property name="maxActive" value="10"/>
20     <property name="minIdle" value="5"/>
21 </bean>
22
23 <!--这里使用MP提供的sqlSessionFactory，完成了Spring与MP的整合-->
24 <bean id="sqlSessionFactory"
25     class="com.baomidou.mybatisplus.extension.spring.MybatisSqlSessionFactoryBean">
26     <property name="dataSource" ref="dataSource"/>
27 </bean>
28
29 <!--扫描mapper接口，使用的依然是Mybatis原生的扫描器-->
30 <bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">
31     <property name="basePackage" value="cn.itcast.mp.simple.mapper"/>
32 </bean>
33 </beans>
34
```

第三步，编写User对象以及UserMapper接口：

```
1 package cn.itcast.mp.simple.pojo;
2
3 import com.baomidou.mybatisplus.annotation.TableName;
4 import lombok.AllArgsConstructor;
5 import lombok.Data;
6 import lombok.NoArgsConstructor;
7
8 @Data
9 @NoArgsConstructor
10 @AllArgsConstructor
11 @TableName("tb_user")
12 public class User {
13
14     private Long id;
15     private String userName;
16     private String password;
17     private String name;
18     private Integer age;
19     private String email;
20 }
21
```

```
1 package cn.itcast.mp.simple.mapper;
2
3 import cn.itcast.mp.simple.pojo.User;
4 import com.baomidou.mybatisplus.core.mapper.BaseMapper;
5
6 public interface UserMapper extends BaseMapper<User> {
7
8 }
```

第四步，编写测试用例：



```
1 package cn.itcast.mp.simple;
2
3 import cn.itcast.mp.simple.mapper.UserMapper;
4 import cn.itcast.mp.simple.pojo.User;
5 import org.junit.Test;
6 import org.junit.runner.RunWith;
7 import org.springframework.beans.factory.annotation.Autowired;
8 import org.springframework.test.context.ContextConfiguration;
9 import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;
10
11 import java.util.List;
12
13 @RunWith(SpringJUnit4ClassRunner.class)
14 @ContextConfiguration(locations = "classpath:applicationContext.xml")
15 public class TestSpringMP {
16
17     @Autowired
18     private UserMapper userMapper;
19
20     @Test
21     public void testSelectList(){
22         List<User> users = this.userMapper.selectList(null);
23         for (User user : users) {
24             System.out.println(user);
25         }
26     }
27
28 }
29
```

测试：

```
1 [main] [cn.itcast.mp.simple.mapper.UserMapper.selectList]-[DEBUG] ==> Preparing:
  SELECT id,user_name,password,name,age,email FROM tb_user
2 [main] [cn.itcast.mp.simple.mapper.UserMapper.selectList]-[DEBUG] ==> Parameters:
3 [main] [cn.itcast.mp.simple.mapper.UserMapper.selectList]-[DEBUG] <== Total: 5
4 [main] [org.mybatis.spring.SqlSessionUtils]-[DEBUG] Closing non transactional
  SqlSession [org.apache.ibatis.session.defaults.DefaultSqlSession@74287ea3]
5 User(id=1, userName=zhangsan, password=123456, name=张三, age=18,
  email=test1@itcast.cn)
6 User(id=2, userName=lisi, password=123456, name=李四, age=20, email=test2@itcast.cn)
7 User(id=3, userName=wangwu, password=123456, name=王五, age=28, email=test3@itcast.cn)
8 User(id=4, userName=zhao Liu, password=123456, name=赵六, age=21, email=test4@itcast.cn)
9 User(id=5, userName=sunqi, password=123456, name=孙七, age=24, email=test5@itcast.cn)
10
```

2.5、SpringBoot + Mybatis + MP

使用SpringBoot将进一步的简化MP的整合，需要注意的是，由于使用SpringBoot需要继承parent，所以需要重新创建工程，并不是创建子Module。

2.5.1、创建工程



New Project

GroupId: cn.itcast.mp ☒ Inherit

ArtifactId: itcast-mp-springboot

Version: 1.0-SNAPSHOT ☒ Inherit

Previous Next Cancel Help

2.5.2、导入依赖

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0"
3         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
5         http://maven.apache.org/xsd/maven-4.0.0.xsd">
6     <modelVersion>4.0.0</modelVersion>
7
8     <parent>
9         <groupId>org.springframework.boot</groupId>
10        <artifactId>spring-boot-starter-parent</artifactId>
11        <version>2.1.4.RELEASE</version>
12    </parent>
13
14    <groupId>cn.itcast.mp</groupId>
15    <artifactId>itcast-mp-springboot</artifactId>
16    <version>1.0-SNAPSHOT</version>
17
18    <dependencies>
19        <dependency>
20            <groupId>org.springframework.boot</groupId>
21            <artifactId>spring-boot-starter</artifactId>
22            <exclusions>
23                <exclusion>
```



```
23         <groupId>org.springframework.boot</groupId>
24         <artifactId>spring-boot-starter-logging</artifactId>
25     </exclusion>
26 </exclusions>
27 </dependency>
28 <dependency>
29     <groupId>org.springframework.boot</groupId>
30     <artifactId>spring-boot-starter-test</artifactId>
31     <scope>test</scope>
32 </dependency>
33
34 <!--简化代码的工具包-->
35 <dependency>
36     <groupId>org.projectlombok</groupId>
37     <artifactId>lombok</artifactId>
38     <optional>true</optional>
39 </dependency>
40 <!--mybatis-plus的springboot支持-->
41 <dependency>
42     <groupId>com.baomidou</groupId>
43     <artifactId>mybatis-plus-boot-starter</artifactId>
44     <version>3.1.1</version>
45 </dependency>
46 <!--mysql驱动-->
47 <dependency>
48     <groupId>mysql</groupId>
49     <artifactId>mysql-connector-java</artifactId>
50     <version>5.1.47</version>
51 </dependency>
52 <dependency>
53     <groupId>org.slf4j</groupId>
54     <artifactId>slf4j-log4j12</artifactId>
55 </dependency>
56
57 </dependencies>
58
59 <build>
60     <plugins>
61         <plugin>
62             <groupId>org.springframework.boot</groupId>
63             <artifactId>spring-boot-maven-plugin</artifactId>
64         </plugin>
65     </plugins>
66 </build>
67
68 </project>
```

log4j.properties :



```
1 log4j.rootLogger=DEBUG,A1
2
3 log4j.appender.A1=org.apache.log4j.ConsoleAppender
4 log4j.appender.A1.layout=org.apache.log4j.PatternLayout
5 log4j.appender.A1.layout.ConversionPattern=[%t] [%c]-[%p] %m%n
```

2.5.3、编写application.properties

```
1 spring.application.name = itcast-mp-springboot
2
3 spring.datasource.driver-class-name=com.mysql.jdbc.Driver
4 spring.datasource.url=jdbc:mysql://127.0.0.1:3306/mp?
  useUnicode=true&characterEncoding=utf8&autoReconnect=true&allowMultiQueries=true&useSSL
  =false
5 spring.datasource.username=root
6 spring.datasource.password=root
```

2.5.4、编写pojo

```
1 package cn.itcast.mp.pojo;
2
3 import com.baomidou.mybatisplus.annotation.TableName;
4 import lombok.AllArgsConstructor;
5 import lombok.Data;
6 import lombok.NoArgsConstructor;
7
8 @Data
9 @NoArgsConstructor
10 @AllArgsConstructor
11 @TableName("tb_user")
12 public class User {
13
14     private Long id;
15     private String userName;
16     private String password;
17     private String name;
18     private Integer age;
19     private String email;
20 }
```

2.5.5、编写mapper

```
1 package cn.itcast.mp.mapper;
2
3 import cn.itcast.mp.pojo.User;
4 import com.baomidou.mybatisplus.core.mapper.BaseMapper;
5
6 public interface UserMapper extends BaseMapper<User> {
7 }
8
```



2.5.6、编写启动类

```
1 package cn.itcast.mp;
2
3 import org.mybatis.spring.annotation.MapperScan;
4 import org.springframework.boot.SpringApplication;
5 import org.springframework.boot.WebApplicationType;
6 import org.springframework.boot.autoconfigure.SpringBootApplication;
7 import org.springframework.boot.builder.SpringApplicationBuilder;
8
9 @MapperScan("cn.itcast.mp.mapper") //设置mapper接口的扫描包
10 @SpringBootApplication
11 public class MyApplication {
12
13     public static void main(String[] args) {
14         SpringApplication.run(MyApplication.class, args);
15     }
16
17 }
```

2.5.7、编写测试用例

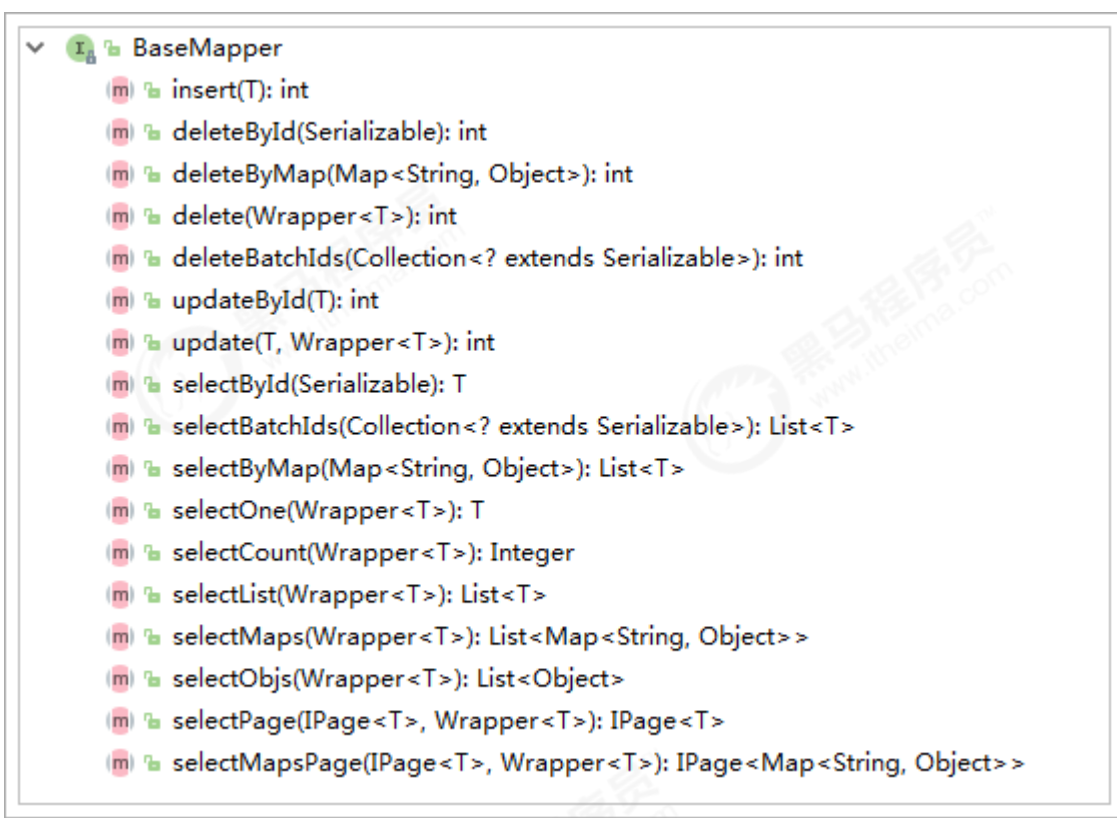
```
1 package cn.itcast.mp;
2
3 import cn.itcast.mp.mapper.UserMapper;
4 import cn.itcast.mp.pojo.User;
5 import org.junit.Test;
6 import org.junit.runner.RunWith;
7 import org.springframework.beans.factory.annotation.Autowired;
8 import org.springframework.boot.test.context.SpringBootTest;
9 import org.springframework.test.context.junit4.SpringRunner;
10
11 import java.util.List;
12
13 @RunWith(SpringRunner.class)
14 @SpringBootTest
15 public class UserMapperTest {
16
17     @Autowired
18     private UserMapper userMapper;
19
20     @Test
21     public void testSelect() {
22         List<User> userList = userMapper.selectList(null);
23         for (User user : userList) {
24             System.out.println(user);
25         }
26     }
27
28 }
```

测试：

```
1 [main] [cn.itcast.mp.mapper.UserMapper.selectList]-[DEBUG] ==> Preparing: SELECT
id,user_name,password,name,age,email FROM tb_user
2 [main] [cn.itcast.mp.mapper.UserMapper.selectList]-[DEBUG] ==> Parameters:
3 [main] [cn.itcast.mp.mapper.UserMapper.selectList]-[DEBUG] <== Total: 5
4 [main] [org.mybatis.spring.SqlSessionUtils]-[DEBUG] Closing non transactional
SqlSession [org.apache.ibatis.session.defaults.DefaultSqlSession@14faa38c]
5 User(id=1, userName=zhangsan, password=123456, name=张三, age=18, email=test1@itcast.cn)
6 User(id=2, userName=lisi, password=123456, name=李四, age=20, email=test2@itcast.cn)
7 User(id=3, userName=wangwu, password=123456, name=王五, age=28, email=test3@itcast.cn)
8 User(id=4, userName=zhaoliu, password=123456, name=赵六, age=21, email=test4@itcast.cn)
9 User(id=5, userName=sunqi, password=123456, name=孙七, age=24, email=test5@itcast.cn)
```

3、通用CRUD

通过前面的学习，我们了解到通过继承BaseMapper就可以获取到各种各样的单表操作，接下来我们将详细讲解这些操作。



3.1、插入操作

3.1.1、方法定义

```
1 /**
2  * 插入一条记录
3  *
4  * @param entity 实体对象
5  */
6 int insert(T entity);
```



3.1.2、测试用例

```
1 package cn.itcast.mp;
2
3 import cn.itcast.mp.mapper.UserMapper;
4 import cn.itcast.mp.pojo.User;
5 import org.junit.Test;
6 import org.junit.runner.RunWith;
7 import org.springframework.beans.factory.annotation.Autowired;
8 import org.springframework.boot.test.context.SpringBootTest;
9 import org.springframework.test.context.junit4.SpringRunner;
10
11 import java.util.List;
12
13 @RunWith(SpringRunner.class)
14 @SpringBootTest
15 public class UserMapperTest {
16
17     @Autowired
18     private UserMapper userMapper;
19
20     @Test
21     public void testInsert(){
22         User user = new User();
23         user.setAge(20);
24         user.setEmail("test@itcast.cn");
25         user.setName("曹操");
26         user.setUsername("caocao");
27         user.setPassword("123456");
28
29         int result = this.userMapper.insert(user); //返回的result是受影响的行数，并不是自增
        后的id
30         System.out.println("result = " + result);
31
32         System.out.println(user.getId()); //自增后的id会回填到对象中
33     }
34
35 }
```

3.1.3、测试

```
1 [main] [cn.itcast.mp.mapper.UserMapper.insert]-[DEBUG] ==> Preparing: INSERT INTO
  tb_user ( id, user_name, password, name, age, email ) VALUES ( ?, ?, ?, ?, ?, ? )
2 [main] [cn.itcast.mp.mapper.UserMapper.insert]-[DEBUG] ==> Parameters:
  1122045867793072130(Long), caocao(String), 123456(String), 曹操(String), 20(Integer),
  test@itcast.cn(String)
3 [main] [cn.itcast.mp.mapper.UserMapper.insert]-[DEBUG] <== Updates: 1
4 [main] [org.mybatis.spring.SqlSessionUtils]-[DEBUG] Closing non transactional
  SqlSession [org.apache.ibatis.session.defaults.DefaultSqlSession@411291e5]
5 result = 1
6 1122045867793072130
```

id	user_name	password	name	age	email
1	zhangsan	123456	张三	18	test1@itcast.cn
2	lisi	123456	李四	20	test2@itcast.cn
3	wangwu	123456	王五	28	test3@itcast.cn
4	zhaoliu	123456	赵六	21	test4@itcast.cn
5	sunqi	123456	孙七	24	test5@itcast.cn
1122045867793072130	caocao	123456	曹操	20	test@itcast.cn

可以看到，数据已经写入到了数据库，但是，id的值不正确，我们期望的是数据库自增长，实际是MP生成了id的值写入到了数据库。

如何设置id的生成策略呢？

MP支持的id策略：

```
1 package com.baomidou.mybatisplus.annotation;
2
3 import lombok.Getter;
4
5 /**
6  * 生成ID类型枚举类
7  *
8  * @author hubin
9  * @since 2015-11-10
10  */
11 @Getter
12 public enum IdType {
13     /**
14      * 数据库ID自增
15      */
16     AUTO(0),
17     /**
18      * 该类型为未设置主键类型
19      */
20     NONE(1),
21     /**
22      * 用户输入ID
23      * <p>该类型可以通过自己注册自动填充插件进行填充</p>
24      */
25     INPUT(2),
26
27     /** 以下3种类型、只有当插入对象ID 为空，才自动填充。 */
28     /**
29      * 全局唯一ID (idworker)
30      */
31     ID_WORKER(3),
32     /**
33      * 全局唯一ID (UUID)
34      */
35     UUID(4),
```




```
36     /**
37      * 字符串全局唯一ID (idworker 的字符串表示)
38      */
39     ID_WORKER_STR(5);
40
41     private final int key;
42
43     IdType(int key) {
44         this.key = key;
45     }
46 }
47
```

修改User对象：

```
1 package cn.itcast.mp.pojo;
2
3 import com.baomidou.mybatisplus.annotation.IdType;
4 import com.baomidou.mybatisplus.annotation.TableId;
5 import com.baomidou.mybatisplus.annotation.TableName;
6 import lombok.AllArgsConstructor;
7 import lombok.Data;
8 import lombok.NoArgsConstructor;
9
10 @Data
11 @NoArgsConstructor
12 @AllArgsConstructor
13 @TableName("tb_user")
14 public class User {
15
16     @TableId(type = IdType.AUTO) //指定id类型为自增长
17     private Long id;
18     private String userName;
19     private String password;
20     private String name;
21     private Integer age;
22     private String email;
23 }
```

数据插入成功：

id	user_name	password	name	age	email
1	zhangsan	123456	张三	18	test1@itcast.cn
2	lisi	123456	李四	20	test2@itcast.cn
3	wangwu	123456	王五	28	test3@itcast.cn
4	zhaoliu	123456	赵六	21	test4@itcast.cn
5	sunqi	123456	孙七	24	test5@itcast.cn
6	caocao	123456	曹操	20	test@itcast.cn



3.1.4、@TableField

在MP中通过@TableField注解可以指定字段的一些属性，常常解决的问题有2个：

- 1、对象中的属性名和字段名不一致的问题（非驼峰）
- 2、对象中的属性字段在表中不存在的问题

使用：

```
@Data
@NoArgsConstructor
@AllArgsConstructor
@Table("tb_user")
public class User {

    @TableId(type = IdType.AUTO)
    private Long id;
    private String userName;
    private String password;
    private String name;
    private Integer age;
    @TableField(value = "email") //解决字段名不一致
    private String mail;

    @TableField(exist = false)
    private String address; //该字段在数据库表中不存在
}
```

其他用法，如大字段不加入查询字段：

```
@Table("tb_user")
public class User {

    @TableId(type = IdType.AUTO)
    private Long id;
    private String userName;
    @TableField(select = false)
    private String password;
    private String name;
    private Integer age;
    @TableField(value = "email") //解决字段名不一致
    private String mail;
}
```

效果：



```
[main] [org.mybatis.spring.SqlSessionUtils]-[DEBUG] Closing non transactional SqlSession
User(id=1, userName=zhangsan, password=null, name=张三, age=18, mail=test1@itcast.cn, address=)
User(id=2, userName=lisi, password=null, name=李四, age=20, mail=test2@itcast.cn, address=)
User(id=3, userName=wangwu, password=null, name=王五, age=28, mail=test3@itcast.cn, address=)
User(id=4, userName=zhaoliu, password=null, name=赵六, age=21, mail=test4@itcast.cn, address=)
User(id=5, userName=sunqi, password=null, name=孙七, age=24, mail=test5@itcast.cn, address=)
User(id=6, userName=caocao, password=null, name=曹操, age=20, mail=test@itcast.cn, address=)
```

3.2、更新操作

在MP中，更新操作有2种，一种是根据id更新，另一种是根据条件更新。

3.2.1、根据id更新

方法定义：

```
1  /**
2   * 根据 ID 修改
3   *
4   * @param entity 实体对象
5   */
6  int updateById(@Param(Constants.ENTITY) T entity);
```

测试：

```
1  @RunWith(SpringRunner.class)
2  @SpringBootTest
3  public class UserMapperTest {
4
5      @Autowired
6      private UserMapper userMapper;
7
8      @Test
9      public void testUpdateById() {
10         User user = new User();
11         user.setId(6L); //主键
12         user.setAge(21); //更新的字段
13
14         //根据id更新，更新不为null的字段
15         this.userMapper.updateById(user);
16     }
17
18 }
```

结果：

```
1  [main] [cn.itcast.mp.mapper.UserMapper.updateById]-[DEBUG] ==> Preparing: UPDATE
   tb_user SET age=? WHERE id=?
2  [main] [cn.itcast.mp.mapper.UserMapper.updateById]-[DEBUG] ==> Parameters: 21(Integer),
   6(Long)
3  [main] [cn.itcast.mp.mapper.UserMapper.updateById]-[DEBUG] <== Updates: 1
```



id	user_name	password	name	age	email
1	zhangsan	123456	张三	18	test1@itcast.cn
2	lisi	123456	李四	20	test2@itcast.cn
3	wangwu	123456	王五	28	test3@itcast.cn
4	zhaoliu	123456	赵六	21	test4@itcast.cn
5	sunqi	123456	孙七	24	test5@itcast.cn
6	caocao	123456	曹操	21	test@itcast.cn

3.2.2、根据条件更新

方法定义：

```
1  /**
2   * 根据 whereEntity 条件，更新记录
3   *
4   * @param entity      实体对象 (set 条件值,可以为 null)
5   * @param updateWrapper 实体对象封装操作类 (可以为 null,里面的 entity 用于生成 where 语句)
6   */
7  int update(@Param(Constants.ENTITY) T entity, @Param(Constants.WRAPPER) Wrapper<T>
updateWrapper);
8
```

测试用例：

```
1  package cn.itcast.mp;
2
3  import cn.itcast.mp.mapper.UserMapper;
4  import cn.itcast.mp.pojo.User;
5  import com.baomidou.mybatisplus.core.conditions.wrapper;
6  import com.baomidou.mybatisplus.core.conditions.query.QueryWrapper;
7  import com.baomidou.mybatisplus.core.conditions.update.UpdateWrapper;
8  import net.minidev.json.writer.UpdaterMapper;
9  import org.junit.Test;
10 import org.junit.runner.RunWith;
11 import org.springframework.beans.factory.annotation.Autowired;
12 import org.springframework.boot.test.context.SpringBootTest;
13 import org.springframework.test.context.junit4.SpringRunner;
14
15 import java.util.List;
16
17 @RunWith(SpringRunner.class)
18 @SpringBootTest
19 public class UserMapperTest {
20
21     @Autowired
22     private UserMapper userMapper;
23
24     @Test
25     public void testUpdate() {
```



```
26     User user = new User();
27     user.setAge(22); //更新的字段
28
29     //更新的条件
30     QueryWrapper<User> wrapper = new QueryWrapper<>();
31     wrapper.eq("id", 6);
32
33     //执行更新操作
34     int result = this.userMapper.update(user, wrapper);
35     System.out.println("result = " + result);
36 }
37
38 }
```

或者，通过UpdateWrapper进行更新：

```
1     @Test
2     public void testUpdate() {
3         //更新的条件以及字段
4         UpdateWrapper<User> wrapper = new UpdateWrapper<>();
5         wrapper.eq("id", 6).set("age", 23);
6
7         //执行更新操作
8         int result = this.userMapper.update(null, wrapper);
9         System.out.println("result = " + result);
10    }
```

测试结果：

```
1 [main] [cn.itcast.mp.mapper.UserMapper.update]-[DEBUG] ==> Preparing: UPDATE tb_user
  SET age=? WHERE id = ?
2 [main] [cn.itcast.mp.mapper.UserMapper.update]-[DEBUG] ==> Parameters: 23(Integer),
  6(Integer)
3 [main] [cn.itcast.mp.mapper.UserMapper.update]-[DEBUG] <== Updates: 1
```

均可达到更新的效果。

关于wrapper更多的用法后面会详细讲解。

3.3、删除操作

3.3.1、deleteById

方法定义：

```
1 /**
2  * 根据 ID 删除
3  *
4  * @param id 主键ID
5  */
6 int deleteById(Serializable id);
```

测试用例：

```
1 package cn.itcast.mp;  
2  
3 import cn.itcast.mp.mapper.UserMapper;  
4 import org.junit.Test;  
5 import org.junit.runner.RunWith;  
6 import org.springframework.beans.factory.annotation.Autowired;  
7 import org.springframework.boot.test.context.SpringBootTest;  
8 import org.springframework.test.context.junit4.SpringRunner;  
9  
10 @RunWith(SpringRunner.class)  
11 @SpringBootTest  
12 public class UserMapperTest {  
13  
14     @Autowired  
15     private UserMapper userMapper;  
16  
17     @Test  
18     public void testDeleteById() {  
19         //执行删除操作  
20         int result = this.userMapper.deleteById(6L);  
21         System.out.println("result = " + result);  
22     }  
23  
24 }
```

结果：

```
1 [main] [cn.itcast.mp.mapper.UserMapper.deleteById]-[DEBUG] ==> Preparing: DELETE FROM  
tb_user WHERE id=?  
2 [main] [cn.itcast.mp.mapper.UserMapper.deleteById]-[DEBUG] ==> Parameters: 6(Long)  
3 [main] [cn.itcast.mp.mapper.UserMapper.deleteById]-[DEBUG] <== Updates: 1
```

id	user_name	password	name	age	email
1	zhangsan	123456	张三	18	test1@itcast.cn
2	lisi	123456	李四	20	test2@itcast.cn
3	wangwu	123456	王五	28	test3@itcast.cn
4	zhaoliu	123456	赵六	21	test4@itcast.cn
5	sunqi	123456	孙七	24	test5@itcast.cn

数据被删除。

3.3.2、deleteByMap

方法定义：



```
1  /**
2   * 根据 columnMap 条件，删除记录
3   *
4   * @param columnMap 表字段 map 对象
5   */
6  int deleteByMap(@Param(Constants.COLUMN_MAP) Map<String, Object> columnMap);
```

测试用例：

```
1  package cn.itcast.mp;
2
3  import cn.itcast.mp.mapper.UserMapper;
4  import org.junit.Test;
5  import org.junit.runner.RunWith;
6  import org.springframework.beans.factory.annotation.Autowired;
7  import org.springframework.boot.test.context.SpringBootTest;
8  import org.springframework.test.context.junit4.SpringRunner;
9
10 import java.util.HashMap;
11 import java.util.Map;
12
13 @RunWith(SpringRunner.class)
14 @SpringBootTest
15 public class UserMapperTest {
16
17     @Autowired
18     private UserMapper userMapper;
19
20     @Test
21     public void testDeleteByMap() {
22         Map<String, Object> columnMap = new HashMap<>();
23         columnMap.put("age", 20);
24         columnMap.put("name", "张三");
25
26         //将columnMap中的元素设置为删除的条件，多个之间为and关系
27         int result = this.userMapper.deleteByMap(columnMap);
28         System.out.println("result = " + result);
29     }
30
31 }
```

结果：

```
1  [main] [cn.itcast.mp.mapper.UserMapper.deleteByMap]-[DEBUG] ==>   Preparing: DELETE FROM
   tb_user WHERE name = ? AND age = ?
2  [main] [cn.itcast.mp.mapper.UserMapper.deleteByMap]-[DEBUG] ==> Parameters: 张三
   (String), 20(Integer)
3  [main] [cn.itcast.mp.mapper.UserMapper.deleteByMap]-[DEBUG] <==    Updates: 0
```




3.3.3、delete

方法定义：

```
1  /**
2   * 根据 entity 条件，删除记录
3   *
4   * @param wrapper 实体对象封装操作类（可以为 null）
5   */
6  int delete(@Param(Constants.WRAPPER) Wrapper<T> wrapper);
```

测试用例：

```
1  package cn.itcast.mp;
2
3  import cn.itcast.mp.mapper.UserMapper;
4  import cn.itcast.mp.pojo.User;
5  import com.baomidou.mybatisplus.core.conditions.query.QueryWrapper;
6  import org.junit.Test;
7  import org.junit.runner.RunWith;
8  import org.springframework.beans.factory.annotation.Autowired;
9  import org.springframework.boot.test.context.SpringBootTest;
10 import org.springframework.test.context.junit4.SpringRunner;
11
12 import java.util.HashMap;
13 import java.util.Map;
14
15 @RunWith(SpringRunner.class)
16 @SpringBootTest
17 public class UserMapperTest {
18
19     @Autowired
20     private UserMapper userMapper;
21
22     @Test
23     public void testDeleteByMap() {
24         User user = new User();
25         user.setAge(20);
26         user.setName("张三");
27
28         //将实体对象进行包装，包装为操作条件
29         QueryWrapper<User> wrapper = new QueryWrapper<>(user);
30
31         int result = this.userMapper.delete(wrapper);
32         System.out.println("result = " + result);
33     }
34
35 }
```

结果：



```
1 [main] [cn.itcast.mp.mapper.UserMapper.delete]-[DEBUG] ==> Preparing: DELETE FROM
  tb_user WHERE name=? AND age=?
2 [main] [cn.itcast.mp.mapper.UserMapper.delete]-[DEBUG] ==> Parameters: 张三(String),
  20(Integer)
3 [main] [cn.itcast.mp.mapper.UserMapper.delete]-[DEBUG] <== Updates: 0
```

3.3.4、deleteBatchIds

方法定义：

```
1 /**
2  * 删除（根据ID 批量删除）
3  *
4  * @param idList 主键ID列表(不能为 null 以及 empty)
5  */
6 int deleteBatchIds(@Param(Constants.COLLECTION) Collection<? extends Serializable>
  idList);
```

测试用例：

```
1 package cn.itcast.mp;
2
3 import cn.itcast.mp.mapper.UserMapper;
4 import org.junit.Test;
5 import org.junit.runner.RunWith;
6 import org.springframework.beans.factory.annotation.Autowired;
7 import org.springframework.boot.test.context.SpringBootTest;
8 import org.springframework.test.context.junit4.SpringRunner;
9
10 import java.util.Arrays;
11
12 @RunWith(SpringRunner.class)
13 @SpringBootTest
14 public class UserMapperTest {
15
16     @Autowired
17     private UserMapper userMapper;
18
19     @Test
20     public void testDeleteByMap() {
21         //根据id集合批量删除
22         int result = this.userMapper.deleteBatchIds(Arrays.asList(1L,10L,20L));
23         System.out.println("result = " + result);
24     }
25
26 }
```

结果：

```
1 [main] [cn.itcast.mp.mapper.UserMapper.deleteBatchIds]-[DEBUG] ==> Preparing: DELETE
FROM tb_user WHERE id IN ( ? , ? , ? )
2 [main] [cn.itcast.mp.mapper.UserMapper.deleteBatchIds]-[DEBUG] ==> Parameters: 1(Long),
10(Long), 20(Long)
3 [main] [cn.itcast.mp.mapper.UserMapper.deleteBatchIds]-[DEBUG] <== Updates: 1
```

3.4、查询操作

MP提供了多种查询操作，包括根据id查询、批量查询、查询单条数据、查询列表、分页查询等操作。

3.4.1、selectById

方法定义：

```
1 /**
2  * 根据 ID 查询
3  *
4  * @param id 主键ID
5  */
6 T selectById(Serializable id);
```

测试用例：

```
1 package cn.itcast.mp;
2
3 import cn.itcast.mp.mapper.UserMapper;
4 import cn.itcast.mp.pojo.User;
5 import org.junit.Test;
6 import org.junit.runner.RunWith;
7 import org.springframework.beans.factory.annotation.Autowired;
8 import org.springframework.boot.test.context.SpringBootTest;
9 import org.springframework.test.context.junit4.SpringRunner;
10
11 @RunWith(SpringRunner.class)
12 @SpringBootTest
13 public class UserMapperTest {
14
15     @Autowired
16     private UserMapper userMapper;
17
18     @Test
19     public void testSelectById() {
20         //根据id查询数据
21         User user = this.userMapper.selectById(2L);
22         System.out.println("result = " + user);
23     }
24
25 }
```

结果：



```
1 [main] [cn.itcast.mp.mapper.UserMapper.selectById]-[DEBUG] ==> Preparing: SELECT
  id,user_name,password,name,age,email FROM tb_user WHERE id=?
2 [main] [cn.itcast.mp.mapper.UserMapper.selectById]-[DEBUG] ==> Parameters: 2(Long)
3 [main] [cn.itcast.mp.mapper.UserMapper.selectById]-[DEBUG] <==      Total: 1
4
5 result = User(id=2, userName=lisi, password=123456, name=李四, age=20,
  email=test2@itcast.cn, address=null)
```

3.4.2、selectBatchIds

方法定义：

```
1 /**
2  * 查询(根据ID 批量查询)
3  *
4  * @param idList 主键ID列表(不能为 null 以及 empty)
5  */
6 List<T> selectBatchIds(@Param(Constants.COLLECTION) Collection<? extends Serializable>
  idList);
```

测试用例：

```
1 package cn.itcast.mp;
2
3 import cn.itcast.mp.mapper.UserMapper;
4 import cn.itcast.mp.pojo.User;
5 import org.junit.Test;
6 import org.junit.runner.RunWith;
7 import org.springframework.beans.factory.annotation.Autowired;
8 import org.springframework.boot.test.context.SpringBootTest;
9 import org.springframework.test.context.junit4.SpringRunner;
10
11 import java.util.Arrays;
12 import java.util.List;
13
14 @RunWith(SpringRunner.class)
15 @SpringBootTest
16 public class UserMapperTest {
17
18     @Autowired
19     private UserMapper userMapper;
20
21     @Test
22     public void testSelectBatchIds() {
23         //根据id集合批量查询
24         List<User> users = this.userMapper.selectBatchIds(Arrays.asList(2L, 3L, 10L));
25         for (User user : users) {
26             System.out.println(user);
27         }
28     }
29
30 }
```



结果：

```
1 [main] [cn.itcast.mp.mapper.UserMapper.selectBatchIds]-[DEBUG] ==> Preparing: SELECT
id,user_name,password,name,age,email FROM tb_user WHERE id IN ( ? , ? , ? )
2 [main] [cn.itcast.mp.mapper.UserMapper.selectBatchIds]-[DEBUG] ==> Parameters: 2(Long),
3(Long), 10(Long)
3 [main] [cn.itcast.mp.mapper.UserMapper.selectBatchIds]-[DEBUG] <==      Total: 2
4
5 User(id=2, userName=lisi, password=123456, name=李四, age=20, email=test2@itcast.cn,
address=null)
6 User(id=3, userName=wangwu, password=123456, name=王五, age=28, email=test3@itcast.cn,
address=null)
```

3.4.3、selectOne

方法定义：

```
1 /**
2  * 根据 entity 条件，查询一条记录
3  *
4  * @param querywrapper 实体对象封装操作类（可以为 null）
5  */
6 T selectOne(@Param(Constants.WRAPPER) Wrapper<T> querywrapper);
```

测试用例：

```
1 package cn.itcast.mp;
2
3 import cn.itcast.mp.mapper.UserMapper;
4 import cn.itcast.mp.pojo.User;
5 import com.baomidou.mybatisplus.core.conditions.query.QueryWrapper;
6 import org.junit.Test;
7 import org.junit.runner.RunWith;
8 import org.springframework.beans.factory.annotation.Autowired;
9 import org.springframework.boot.test.context.SpringBootTest;
10 import org.springframework.test.context.junit4.SpringRunner;
11
12 @RunWith(SpringRunner.class)
13 @SpringBootTest
14 public class UserMapperTest {
15
16     @Autowired
17     private UserMapper userMapper;
18
19     @Test
20     public void testSelectOne() {
21         QueryWrapper<User> wrapper = new QueryWrapper<User>();
22         wrapper.eq("name", "李四");
23
24         //根据条件查询一条数据，如果结果超过一条会报错
25         User user = this.userMapper.selectOne(wrapper);
```



```
26         System.out.println(user);
27     }
28
29 }
```

结果：

```
1 [main] [cn.itcast.mp.mapper.UserMapper.selectOne]-[DEBUG] ==> Preparing: SELECT
  id,user_name,password,name,age,email FROM tb_user WHERE name = ?
2 [main] [cn.itcast.mp.mapper.UserMapper.selectOne]-[DEBUG] ==> Parameters: 李四(String)
3 [main] [cn.itcast.mp.mapper.UserMapper.selectOne]-[DEBUG] <==      Total: 1
4
5 User(id=2, userName=lisi, password=123456, name=李四, age=20, email=test2@itcast.cn,
  address=null)
```

3.4.4、selectCount

方法定义：

```
1 /**
2  * 根据 wrapper 条件，查询总记录数
3  *
4  * @param queryWrapper 实体对象封装操作类（可以为 null）
5  */
6 Integer selectCount(@Param(Constants.WRAPPER) Wrapper<T> queryWrapper);
```

测试用例：

```
1 package cn.itcast.mp;
2
3 import cn.itcast.mp.mapper.UserMapper;
4 import cn.itcast.mp.pojo.User;
5 import com.baomidou.mybatisplus.core.conditions.query.QueryWrapper;
6 import org.junit.Test;
7 import org.junit.runner.RunWith;
8 import org.springframework.beans.factory.annotation.Autowired;
9 import org.springframework.boot.test.context.SpringBootTest;
10 import org.springframework.test.context.junit4.SpringRunner;
11
12 @RunWith(SpringRunner.class)
13 @SpringBootTest
14 public class UserMapperTest {
15
16     @Autowired
17     private UserMapper userMapper;
18
19     @Test
20     public void testSelectCount() {
21         QueryWrapper<User> wrapper = new QueryWrapper<User>();
22         wrapper.gt("age", 23); //年龄大于23岁
23
24         //根据条件查询数据条数
```



```
25     Integer count = this.userMapper.selectCount(wrapper);
26     System.out.println("count = " + count);
27 }
28
29 }
```

结果：

```
1 [main] [cn.itcast.mp.mapper.UserMapper.selectCount]-[DEBUG] ==> Preparing: SELECT
COUNT( 1 ) FROM tb_user WHERE age > ?
2 [main] [cn.itcast.mp.mapper.UserMapper.selectCount]-[DEBUG] ==> Parameters: 23(Integer)
3 [main] [cn.itcast.mp.mapper.UserMapper.selectCount]-[DEBUG] <== Total: 1
4
5 count = 2
```

3.4.5、selectList

方法定义：

```
1 /**
2  * 根据 entity 条件，查询全部记录
3  *
4  * @param queryWrapper 实体对象封装操作类（可以为 null）
5  */
6 List<T> selectList(@Param(Constants.WRAPPER) Wrapper<T> queryWrapper);
```

测试用例：

```
1 package cn.itcast.mp;
2
3 import cn.itcast.mp.mapper.UserMapper;
4 import cn.itcast.mp.pojo.User;
5 import com.baomidou.mybatisplus.core.conditions.query.QueryWrapper;
6 import org.junit.Test;
7 import org.junit.runner.RunWith;
8 import org.springframework.beans.factory.annotation.Autowired;
9 import org.springframework.boot.test.context.SpringBootTest;
10 import org.springframework.test.context.junit4.SpringRunner;
11
12 import java.util.List;
13
14 @RunWith(SpringRunner.class)
15 @SpringBootTest
16 public class UserMapperTest {
17
18     @Autowired
19     private UserMapper userMapper;
20
21     @Test
22     public void testSelectList() {
23         QueryWrapper<User> wrapper = new QueryWrapper<User>();
24         wrapper.gt("age", 23); //年龄大于23岁
```




```

25
26     //根据条件查询数据
27     List<User> users = this.userMapper.selectList(wrapper);
28     for (User user : users) {
29         System.out.println("user = " + user);
30     }
31 }
32
33 }
```

结果：

```

1 [main] [cn.itcast.mp.mapper.UserMapper.selectList]-[DEBUG] ==> Preparing: SELECT
  id,user_name,password,name,age,email FROM tb_user WHERE age > ?
2 [main] [cn.itcast.mp.mapper.UserMapper.selectList]-[DEBUG] ==> Parameters: 23(Integer)
3 [main] [cn.itcast.mp.mapper.UserMapper.selectList]-[DEBUG] <==      Total: 2
4
5 user = User(id=3, userName=wangwu, password=123456, name=王五, age=28,
  email=test3@itcast.cn, address=null)
6 user = User(id=5, userName=sunqi, password=123456, name=孙七, age=24,
  email=test5@itcast.cn, address=null)
```

3.4.6、selectPage

方法定义：

```

1 /**
2  * 根据 entity 条件，查询全部记录（并翻页）
3  *
4  * @param page          分页查询条件（可以为 RowBounds.DEFAULT）
5  * @param queryWrapper  实体对象封装操作类（可以为 null）
6  */
7 IPage<T> selectPage(IPage<T> page, @Param(Constants.WRAPPER) Wrapper<T> queryWrapper);
```

配置分页插件：

```

1 package cn.itcast.mp;
2
3 import com.baomidou.mybatisplus.extension.plugins.PaginationInterceptor;
4 import org.mybatis.spring.annotation.MapperScan;
5 import org.springframework.context.annotation.Bean;
6 import org.springframework.context.annotation.Configuration;
7
8 @Configuration
9 @MapperScan("cn.itcast.mp.mapper") //设置mapper接口的扫描包
10 public class MybatisPlusConfig {
11
12     /**
13      * 分页插件
14      */
15     @Bean
16     public PaginationInterceptor paginationInterceptor() {
```



```
17         return new PaginationInterceptor();
18     }
19 }
```

测试用例：

```
1 package cn.itcast.mp;
2
3 import cn.itcast.mp.mapper.UserMapper;
4 import cn.itcast.mp.pojo.User;
5 import com.baomidou.mybatisplus.core.conditions.query.QueryWrapper;
6 import com.baomidou.mybatisplus.core.metadata.IPage;
7 import com.baomidou.mybatisplus.extension.plugins.pagination.Page;
8 import org.junit.Test;
9 import org.junit.runner.RunWith;
10 import org.springframework.beans.factory.annotation.Autowired;
11 import org.springframework.boot.test.context.SpringBootTest;
12 import org.springframework.test.context.junit4.SpringRunner;
13
14 import java.util.List;
15
16 @RunWith(SpringRunner.class)
17 @SpringBootTest
18 public class UserMapperTest {
19
20     @Autowired
21     private UserMapper userMapper;
22
23     @Test
24     public void testSelectPage() {
25         QueryWrapper<User> wrapper = new QueryWrapper<User>();
26         wrapper.gt("age", 20); //年龄大于20岁
27
28         Page<User> page = new Page<>(1,1);
29
30         //根据条件查询数据
31         IPage<User> iPage = this.userMapper.selectPage(page, wrapper);
32         System.out.println("数据总条数：" + iPage.getTotal());
33         System.out.println("总页数：" + iPage.getPages());
34
35         List<User> users = iPage.getRecords();
36         for (User user : users) {
37             System.out.println("user = " + user);
38         }
39     }
40 }
41
42 }
```

结果：



```

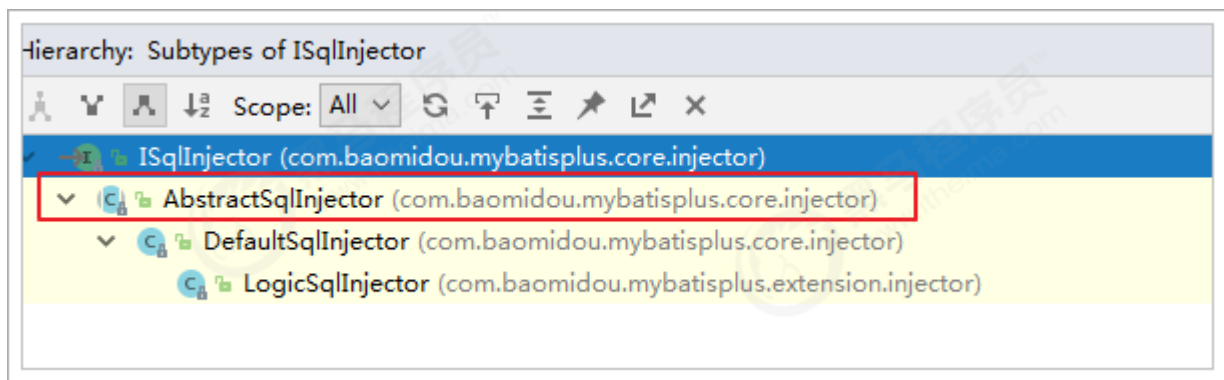
1 [main] [cn.itcast.mp.mapper.UserMapper.selectPage]-[DEBUG] ==> Preparing: SELECT
COUNT(1) FROM tb_user WHERE age > ?
2 [main] [cn.itcast.mp.mapper.UserMapper.selectPage]-[DEBUG] ==> Parameters: 20(Integer)
3 [main] [cn.itcast.mp.mapper.UserMapper.selectPage]-[DEBUG] ==> Preparing: SELECT
id,user_name,password,name,age,email FROM tb_user WHERE age > ? LIMIT ?,?
4 [main] [cn.itcast.mp.mapper.UserMapper.selectPage]-[DEBUG] ==> Parameters: 20(Integer),
0(Long), 1(Long)
5 [main] [cn.itcast.mp.mapper.UserMapper.selectPage]-[DEBUG] <== Total: 1
6 [main] [org.mybatis.spring.SqlSessionUtils]-[DEBUG] Closing non transactional
SqlSession [org.apache.ibatis.session.defaults.DefaultSqlSession@6ecd665]
7 数据总条数: 3
8 总页数: 3
9 user = User(id=3, userName=wangwu, password=123456, name=王五, age=28,
email=test3@itcast.cn, address=null)

```

3.5、SQL注入的原理

前面我们已经知道，MP在启动后会将BaseMapper中的一系列的方法注册到mappedStatements中，那么究竟是如何注入的呢？流程又是怎样的？下面我们将一起来分析下。

在MP中，ISqlInjector负责SQL的注入工作，它是一个接口，AbstractSqlInjector是它的实现类，实现关系如下：



在AbstractSqlInjector中，主要是由inspectInject()方法进行注入的，如下：

```

1 @Override
2 public void inspectInject(MapperBuilderAssistant builderAssistant, Class<?>
mapperClass) {
3     Class<?> modelClass = extractModelClass(mapperClass);
4     if (modelClass != null) {
5         String className = mapperClass.toString();
6         Set<String> mapperRegistryCache =
GlobalConfigUtils.getMapperRegistryCache(builderAssistant.getConfiguration());
7         if (!mapperRegistryCache.contains(className)) {
8             List<AbstractMethod> methodList = this.getMethodList();
9             if (CollectionUtils.isNotEmpty(methodList)) {
10                 TableInfo tableInfo = TableInfoHelper.initTableInfo(builderAssistant,
modelClass);
11                 // 循环注入自定义方法
12                 methodList.forEach(m -> m.inject(builderAssistant, mapperClass,
modelClass, tableInfo));
13             } else {

```



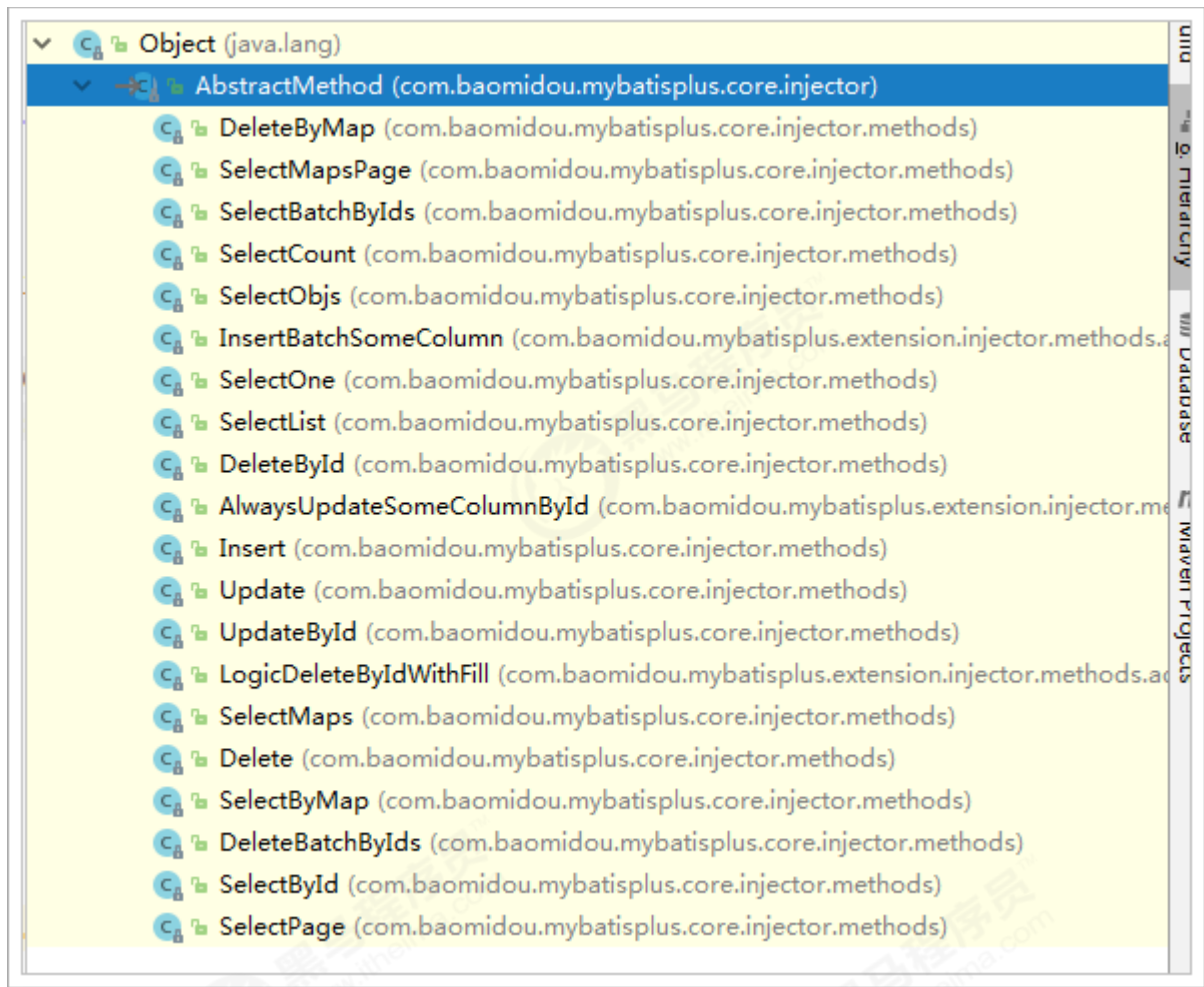
```
14         logger.debug(mapperClass.toString() + ", No effective injection method  
    was found.");  
15     }  
16     mapperRegistryCache.add(className);  
17 }  
18 }  
19 }
```

在实现方法中，`methodList.forEach(m -> m.inject(builderAssistant, mapperClass, modelClass, tableInfo))`;是关键，循环遍历方法，进行注入。

最终调用抽象方法`injectMappedStatement`进行真正的注入：

```
1  /**  
2   * 注入自定义 MappedStatement  
3   *  
4   * @param mapperClass mapper 接口  
5   * @param modelClass mapper 泛型  
6   * @param tableInfo 数据库表反射信息  
7   * @return MappedStatement  
8   */  
9   public abstract MappedStatement injectMappedStatement(Class<?> mapperClass, Class<?  
> modelClass, TableInfo tableInfo);
```

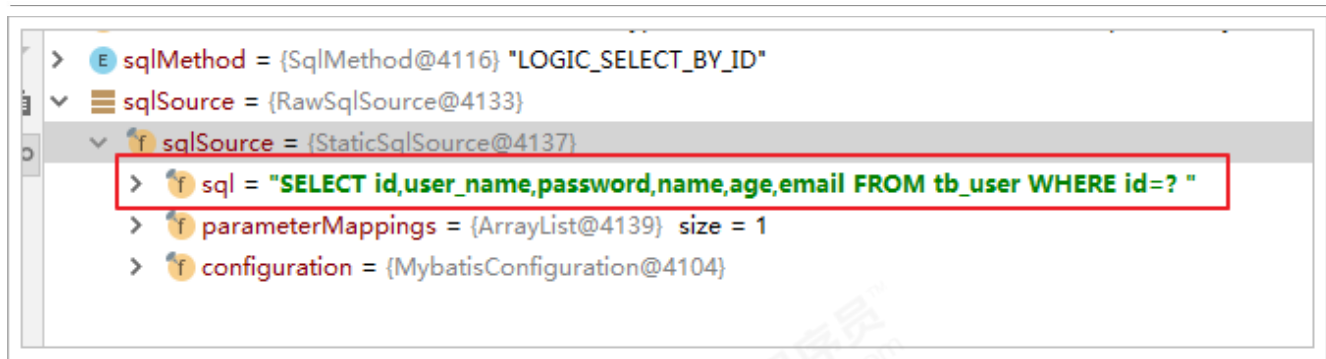
查看该方法的实现：



以SelectById为例查看：

```
1 public class SelectById extends AbstractMethod {
2
3     @Override
4     public MappedStatement injectMappedStatement(Class<?> mapperClass, Class<?>
modelClass, TableInfo tableInfo) {
5         SqlMethod sqlMethod = SqlMethod.LOGIC_SELECT_BY_ID;
6         SqlSource sqlSource = new RawSqlSource(configuration,
String.format(sqlMethod.getSql(),
7             sqlSelectColumns(tableInfo, false),
8             tableInfo.getTableName(), tableInfo.getKeyColumn(),
tableInfo.getKeyProperty(),
9             tableInfo.getLogicDeletesSql(true, false)), Object.class);
10        return this.addSelectMappedStatement(mapperClass, sqlMethod.getMethod(),
sqlSource, modelClass, tableInfo);
11    }
12 }
```

可以看到，生成了SqlSource对象，再将SQL通过addSelectMappedStatement方法添加到meppedStatements中。



4、配置

在MP中有大量的配置，其中有一部分是Mybatis原生的配置，另一部分是MP的配置，详情：<https://mybatis.plus/config/>

下面我们对常用的配置做讲解。

4.1、基本配置

4.1.1、configLocation

MyBatis 配置文件位置，如果您有单独的 MyBatis 配置，请将其路径配置到 configLocation 中。MyBatis Configuration 的具体内容请参考MyBatis 官方文档

Spring Boot :

```
1 | mybatis-plus.config-location = classpath:mybatis-config.xml
```

Spring MVC :

```
1 | <bean id="sqlSessionFactory"
  | class="com.baomidou.mybatisplus.extension.spring.MybatisSqlSessionFactoryBean">
2 |     <property name="configLocation" value="classpath:mybatis-config.xml"/>
3 | </bean>
```

4.1.2、mapperLocations

MyBatis Mapper 所对应的 XML 文件位置，如果您在 Mapper 中有自定义方法（XML 中有自定义实现），需要进行该配置，告诉 Mapper 所对应的 XML 文件位置。

Spring Boot :

```
1 | mybatis-plus.mapper-locations = classpath*:mybatis/*.xml
```

Spring MVC :

```
1 | <bean id="sqlSessionFactory"
  | class="com.baomidou.mybatisplus.extension.spring.MybatisSqlSessionFactoryBean">
2 |     <property name="mapperLocations" value="classpath*:mybatis/*.xml"/>
3 | </bean>
```



Maven 多模块项目的扫描路径需以 `classpath*` 开头（即加载多个 jar 包下的 XML 文件）

测试：

UserMapper.xml：

```
1 <?xml version="1.0" encoding="UTF-8" ?>
2 <!DOCTYPE mapper
3     PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
4     "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
5 <mapper namespace="cn.itcast.mp.mapper.UserMapper">
6
7     <select id="findById" resultType="cn.itcast.mp.pojo.User">
8         select * from tb_user where id = #{id}
9     </select>
10
11 </mapper>
```

```
1 package cn.itcast.mp.mapper;
2
3 import cn.itcast.mp.pojo.User;
4 import com.baomidou.mybatisplus.core.mapper.BaseMapper;
5
6 public interface UserMapper extends BaseMapper<User> {
7
8     User findById(Long id);
9 }
```

测试用例：

```
1 package cn.itcast.mp;
2
3 import cn.itcast.mp.mapper.UserMapper;
4 import cn.itcast.mp.pojo.User;
5 import org.junit.Test;
6 import org.junit.runner.RunWith;
7 import org.springframework.beans.factory.annotation.Autowired;
8 import org.springframework.boot.test.context.SpringBootTest;
9 import org.springframework.test.context.junit4.SpringRunner;
10
11 @RunWith(SpringRunner.class)
12 @SpringBootTest
13 public class UserMapperTest {
14
15     @Autowired
16     private UserMapper userMapper;
17
18     @Test
19     public void testSelectPage() {
20         User user = this.userMapper.findById(2L);
21         System.out.println(user);
22     }
```




```
23     }  
24  
25 }
```

运行结果：

```
ansaction]-[DEBUG] JDBC Connection [HikariProxyConnection@9063  
G] ==> Preparing: select * from tb_user where id = ?  
G] ==> Parameters: 2(Long)  
G] <==          Total: 1  
osing non transactional SqlSession [org.apache.ibatis.session.  
ge=20, email=test2@itcast.cn, address=null)
```

4.1.3、typeAliasesPackage

MyBatis 别名包扫描路径，通过该属性可以给包中的类注册别名，注册后在 Mapper 对应的 XML 文件中可以直接使用类名，而不用使用全限定的类名（即 XML 中调用的时候不用包含包名）。

Spring Boot：

```
1 mybatis-plus.type-aliases-package = cn.itcast.mp.pojo
```

Spring MVC：

```
1 <bean id="sqlSessionFactory"  
  class="com.baomidou.mybatisplus.extension.spring.MybatisSqlSessionFactoryBean">  
2   <property name="typeAliasesPackage"  
  value="com.baomidou.mybatisplus.samples.quickstart.entity"/>  
3 </bean>
```

4.2、进阶配置

本部分（Configuration）的配置大都为 MyBatis 原生支持的配置，这意味着您可以通过 MyBatis XML 配置文件的形式进行配置。

4.2.1、mapUnderscoreToCamelCase

- 类型：`boolean`
- 默认值：`true`

是否开启自动驼峰命名规则（camel case）映射，即从经典数据库列名 `A_COLUMN`（下划线命名）到经典 Java 属性名 `aColumn`（驼峰命名）的类似映射。

注意：

此属性在 MyBatis 中原默认值为 `false`，在 MyBatis-Plus 中，此属性也将用于生成最终的 SQL 的 select body

如果您的数据库命名符合规则无需使用 `@TableField` 注解指定数据库字段名

示例（SpringBoot）：

- ```
1 #关闭自动驼峰映射，该参数不能和mybatis-plus.config-location同时存在
2 mybatis-plus.configuration.map-underscore-to-camel-case=false
```

### 4.2.2、cacheEnabled

- 类型：boolean
- 默认值：true

全局地开启或关闭配置文件中的所有映射器已经配置的任何缓存，默认为 true。

示例：

```
1 mybatis-plus.configuration.cache-enabled=false
```

## 4.3、DB 策略配置

### 4.3.1、idType

- 类型：com.baomidou.mybatisplus.annotation.IdType
- 默认值：ID\_WORKER

全局默认主键类型，设置后，即可省略实体对象中的@TableId(type = IdType.AUTO)配置。

示例：

SpringBoot：

```
1 mybatis-plus.global-config.db-config.id-type=auto
```

SpringMVC：

```
1 <!--这里使用MP提供的sqlSessionFactory，完成了Spring与MP的整合-->
2 <bean id="sqlSessionFactory"
3 class="com.baomidou.mybatisplus.extension.spring.MybatisSqlSessionFactoryBean">
4 <property name="dataSource" ref="dataSource"/>
5 <property name="globalConfig">
6 <bean class="com.baomidou.mybatisplus.core.config.GlobalConfig">
7 <property name="dbConfig">
8 <bean
9 class="com.baomidou.mybatisplus.core.config.GlobalConfig$DbConfig">
10 <property name="idType" value="AUTO"/>
11 </bean>
12 </property>
13 </bean>
14 </property>
15 </bean>
```

### 4.3.2、tablePrefix

- 类型：String
- 默认值：null

表名前缀，全局配置后可省略@TableName()配置。

SpringBoot :

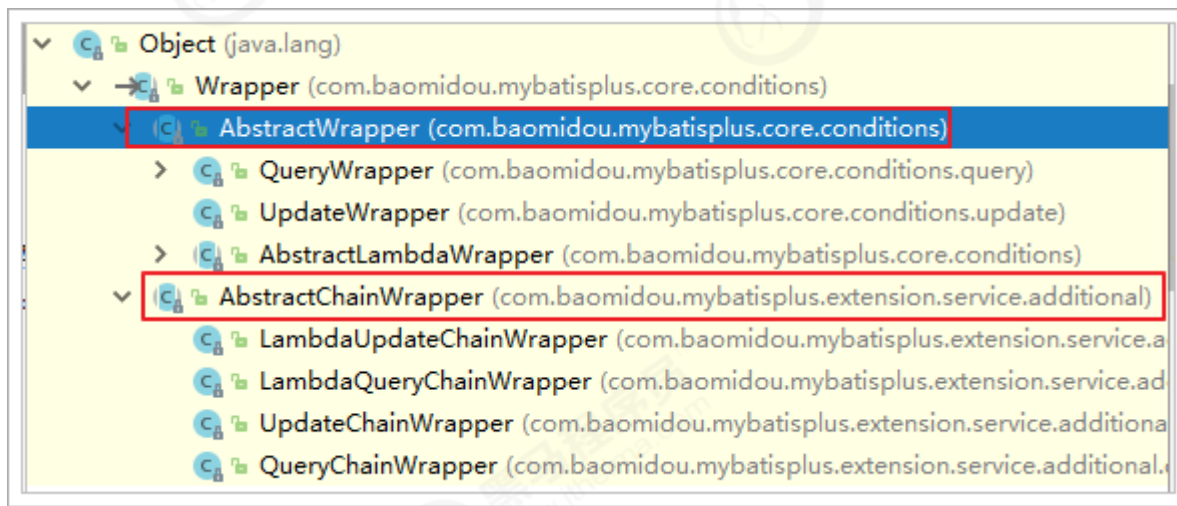
```
1 | mybatis-plus.global-config.db-config.table-prefix=tb_
```

SpringMVC :

```
1 <bean id="sqlSessionFactory"
 class="com.baomidou.mybatisplus.extension.spring.MybatisSqlSessionFactoryBean">
2 <property name="dataSource" ref="dataSource"/>
3 <property name="globalConfig">
4 <bean class="com.baomidou.mybatisplus.core.config.GlobalConfig">
5 <property name="dbConfig">
6 <bean
 class="com.baomidou.mybatisplus.core.config.GlobalConfig$DbConfig">
7 <property name="idType" value="AUTO"/>
8 <property name="tablePrefix" value="tb_" />
9 </bean>
10 </property>
11 </bean>
12 </property>
13 </bean>
```

## 5、条件构造器

在MP中，Wrapper接口的实现类关系如下：



可以看到，AbstractWrapper和AbstractChainWrapper是重点实现，接下来我们重点学习AbstractWrapper以及其子类。

说明:

QueryWrapper(LambdaQueryWrapper) 和 UpdateWrapper(LambdaUpdateWrapper) 的父类 用于生成 sql 的 where 条件, entity 属性也用于生成 sql 的 where 条件 注意: entity 生成的 where 条件与 使用各个 api 生成的 where 条件没有任何关联行为

官网文档地址：<https://mybatis.plus/guide/wrapper.html>



## 5.1、allEq

### 5.1.1、说明

```
1 allEq(Map<R, V> params)
2 allEq(Map<R, V> params, boolean null2IsNull)
3 allEq(boolean condition, Map<R, V> params, boolean null2IsNull)
```

- 全部eq(或个别isNull)

个别参数说明: `params`: `key` 为数据库字段名, `value` 为字段值 `null2IsNull`: 为 `true` 则在 `map` 的 `value` 为 `null` 时调用 `isNull` 方法, 为 `false` 时则忽略 `value` 为 `null` 的

- 例1: `allEq({id:1,name:"老王",age:null})` ---> `id = 1 and name = '老王' and age is null`
- 例2: `allEq({id:1,name:"老王",age:null}, false)` ---> `id = 1 and name = '老王'`

```
1 allEq(BiPredicate<R, V> filter, Map<R, V> params)
2 allEq(BiPredicate<R, V> filter, Map<R, V> params, boolean null2IsNull)
3 allEq(boolean condition, BiPredicate<R, V> filter, Map<R, V> params, boolean
 null2IsNull)
```

个别参数说明: `filter`: 过滤函数, 是否允许字段传入比对条件中 `params` 与 `null2IsNull`: 同上

- 例1: `allEq((k,v) -> k.indexOf("a") > 0, {id:1,name:"老王",age:null})` ---> `name = '老王' and age is null`
- 例2: `allEq((k,v) -> k.indexOf("a") > 0, {id:1,name:"老王",age:null}, false)` ---> `name = '老王'`

### 5.1.2、测试用例

```
1 package cn.itcast.mp;
2
3 import cn.itcast.mp.mapper.UserMapper;
4 import cn.itcast.mp.pojo.User;
5 import com.baomidou.mybatisplus.core.conditions.query.QueryWrapper;
6 import org.junit.Test;
7 import org.junit.runner.RunWith;
8 import org.springframework.beans.factory.annotation.Autowired;
9 import org.springframework.boot.test.context.SpringBootTest;
10 import org.springframework.test.context.junit4.SpringRunner;
11
12 import java.util.HashMap;
13 import java.util.List;
14 import java.util.Map;
15
16 @RunWith(SpringRunner.class)
17 @SpringBootTest
18 public class UserMapperTest {
19
20 @Autowired
21 private UserMapper userMapper;
```



```
22
23 @Test
24 public void testwrapper() {
25 QueryWrapper<User> wrapper = new QueryWrapper<>();
26
27 //设置条件
28 Map<String, Object> params = new HashMap<>();
29 params.put("name", "曹操");
30 params.put("age", "20");
31 params.put("password", null);
32
33 // wrapper.allEq(params);//SELECT * FROM tb_user WHERE password IS NULL AND
name = ? AND age = ?
34 // wrapper.allEq(params, false); //SELECT * FROM tb_user WHERE name = ? AND age
= ?
35
36 // wrapper.allEq((k, v) -> (k.equals("name") || k.equals("age"))
,params);//SELECT * FROM tb_user WHERE name = ? AND age = ?
37
38 List<User> users = this.userMapper.selectList(wrapper);
39 for (User user : users) {
40 System.out.println(user);
41 }
42 }
43 }
44
45 }
```

## 5.2、基本比较操作

- eq
  - 等于 =
- ne
  - 不等于 <>
- gt
  - 大于 >
- ge
  - 大于等于 >=
- lt
  - 小于 <
- le
  - 小于等于 <=
- between
  - BETWEEN 值1 AND 值2
- notBetween
  - NOT BETWEEN 值1 AND 值2
- in
  - 字段 IN (value.get(0), value.get(1), ...)



- notIn
  - 字段 NOT IN (v0, v1, ...)

测试用例：

```
1 package cn.itcast.mp;
2
3 import cn.itcast.mp.mapper.UserMapper;
4 import cn.itcast.mp.pojo.User;
5 import com.baomidou.mybatisplus.core.conditions.query.QueryWrapper;
6 import org.junit.Test;
7 import org.junit.runner.RunWith;
8 import org.springframework.beans.factory.annotation.Autowired;
9 import org.springframework.boot.test.context.SpringBootTest;
10 import org.springframework.test.context.junit4.SpringRunner;
11
12 import java.util.List;
13
14 @RunWith(SpringRunner.class)
15 @SpringBootTest
16 public class UserMapperTest {
17
18 @Autowired
19 private UserMapper userMapper;
20
21 @Test
22 public void testEq() {
23 QueryWrapper<User> wrapper = new QueryWrapper<>();
24
25 //SELECT id,user_name,password,name,age,email FROM tb_user WHERE password = ?
26 //AND age >= ? AND name IN (?, ?, ?)
27 wrapper.eq("password", "123456")
28 .ge("age", 20)
29 .in("name", "李四", "王五", "赵六");
30
31 List<User> users = this.userMapper.selectList(wrapper);
32 for (User user : users) {
33 System.out.println(user);
34 }
35 }
36 }
37 }
```

## 5.3、模糊查询

- like
  - LIKE '%值%'
  - 例: `like("name", "王")` ---> `name like '王%'`
- notLike
  - NOT LIKE '%值%'



- 例: `notLike("name", "王")` ---> `name not like '王%'`
- likeLeft
  - LIKE '%值'
  - 例: `likeLeft("name", "王")` ---> `name like '王'`
- likeRight
  - LIKE '值%'
  - 例: `likeRight("name", "王")` ---> `name like '王%'`

测试用例：

```
1 package cn.itcast.mp;
2
3 import cn.itcast.mp.mapper.UserMapper;
4 import cn.itcast.mp.pojo.User;
5 import com.baomidou.mybatisplus.core.conditions.query.QueryWrapper;
6 import org.junit.Test;
7 import org.junit.runner.RunWith;
8 import org.springframework.beans.factory.annotation.Autowired;
9 import org.springframework.boot.test.context.SpringBootTest;
10 import org.springframework.test.context.junit4.SpringRunner;
11
12 import java.util.List;
13
14 @RunWith(SpringRunner.class)
15 @SpringBootTest
16 public class UserMapperTest {
17
18 @Autowired
19 private UserMapper userMapper;
20
21 @Test
22 public void testWrapper() {
23 QueryWrapper<User> wrapper = new QueryWrapper<>();
24
25 //SELECT id,user_name,password,name,age,email FROM tb_user WHERE name LIKE ?
26 //Parameters: %曹%(String)
27 wrapper.like("name", "曹");
28
29 List<User> users = this.userMapper.selectList(wrapper);
30 for (User user : users) {
31 System.out.println(user);
32 }
33 }
34 }
35
36 }
```

## 5.4、排序

- orderBy
  - 排序：ORDER BY 字段, ...





- 例: `orderBy(true, true, "id", "name")` ---> `order by id ASC,name ASC`
- `orderByAsc`
  - 排序: ORDER BY 字段, ... ASC
  - 例: `orderByAsc("id", "name")` ---> `order by id ASC,name ASC`
- `orderByDesc`
  - 排序: ORDER BY 字段, ... DESC
  - 例: `orderByDesc("id", "name")` ---> `order by id DESC,name DESC`

测试用例:

```
1 package cn.itcast.mp;
2
3 import cn.itcast.mp.mapper.UserMapper;
4 import cn.itcast.mp.pojo.User;
5 import com.baomidou.mybatisplus.core.conditions.query.QueryWrapper;
6 import org.junit.Test;
7 import org.junit.runner.RunWith;
8 import org.springframework.beans.factory.annotation.Autowired;
9 import org.springframework.boot.test.context.SpringBootTest;
10 import org.springframework.test.context.junit4.SpringRunner;
11
12 import java.util.List;
13
14 @RunWith(SpringRunner.class)
15 @SpringBootTest
16 public class UserMapperTest {
17
18 @Autowired
19 private UserMapper userMapper;
20
21 @Test
22 public void testwrapper() {
23 QueryWrapper<User> wrapper = new QueryWrapper<>();
24
25 //SELECT id,user_name,password,name,age,email FROM tb_user ORDER BY age DESC
26 wrapper.orderByDesc("age");
27
28 List<User> users = this.userMapper.selectList(wrapper);
29 for (User user : users) {
30 System.out.println(user);
31 }
32 }
33 }
34
35 }
```

## 5.5、逻辑查询

- `or`
  - 拼接 OR
  - 主动调用 `or` 表示紧接着下一个方法不是用 `and` 连接!(不调用 `or` 则默认为使用 `and` 连接)



- and
  - AND 嵌套
  - 例: `and(i -> i.eq("name", "李白").ne("status", "活着"))` ---> `and (name = '李白' and status <> '活着')`

测试用例：

```
1 package cn.itcast.mp;
2
3 import cn.itcast.mp.mapper.UserMapper;
4 import cn.itcast.mp.pojo.User;
5 import com.baomidou.mybatisplus.core.conditions.query.QueryWrapper;
6 import org.junit.Test;
7 import org.junit.runner.RunWith;
8 import org.springframework.beans.factory.annotation.Autowired;
9 import org.springframework.boot.test.context.SpringBootTest;
10 import org.springframework.test.context.junit4.SpringRunner;
11
12 import java.util.List;
13
14 @RunWith(SpringRunner.class)
15 @SpringBootTest
16 public class UserMapperTest {
17
18 @Autowired
19 private UserMapper userMapper;
20
21 @Test
22 public void testWrapper() {
23 QueryWrapper<User> wrapper = new QueryWrapper<>();
24
25 //SELECT id,user_name,password,name,age,email FROM tb_user WHERE name = ? OR
26 age = ?
27 wrapper.eq("name", "李四").or().eq("age", 24);
28
29 List<User> users = this.userMapper.selectList(wrapper);
30 for (User user : users) {
31 System.out.println(user);
32 }
33 }
34
35 }
```

## 5.6、select

在MP查询中，默认查询所有的字段，如果有需要也可以通过select方法进行指定字段。

```
1 package cn.itcast.mp;
2
3 import cn.itcast.mp.mapper.UserMapper;
4 import cn.itcast.mp.pojo.User;
```



```
5 import com.baomidou.mybatisplus.core.conditions.query.QueryWrapper;
6 import org.junit.Test;
7 import org.junit.runner.RunWith;
8 import org.springframework.beans.factory.annotation.Autowired;
9 import org.springframework.boot.test.context.SpringBootTest;
10 import org.springframework.test.context.junit4.SpringRunner;
11
12 import java.util.List;
13
14 @RunWith(SpringRunner.class)
15 @SpringBootTest
16 public class UserMapperTest {
17
18 @Autowired
19 private UserMapper userMapper;
20
21 @Test
22 public void testWrapper() {
23 QueryWrapper<User> wrapper = new QueryWrapper<>();
24
25 //SELECT id,name,age FROM tb_user WHERE name = ? OR age = ?
26 wrapper.eq("name", "李四")
27 .or()
28 .eq("age", 24)
29 .select("id", "name", "age");
30
31 List<User> users = this.userMapper.selectList(wrapper);
32 for (User user : users) {
33 System.out.println(user);
34 }
35 }
36 }
37
38 }
```