第一章：mybatis plus入门

快速入门

MP简介

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。

愿景：

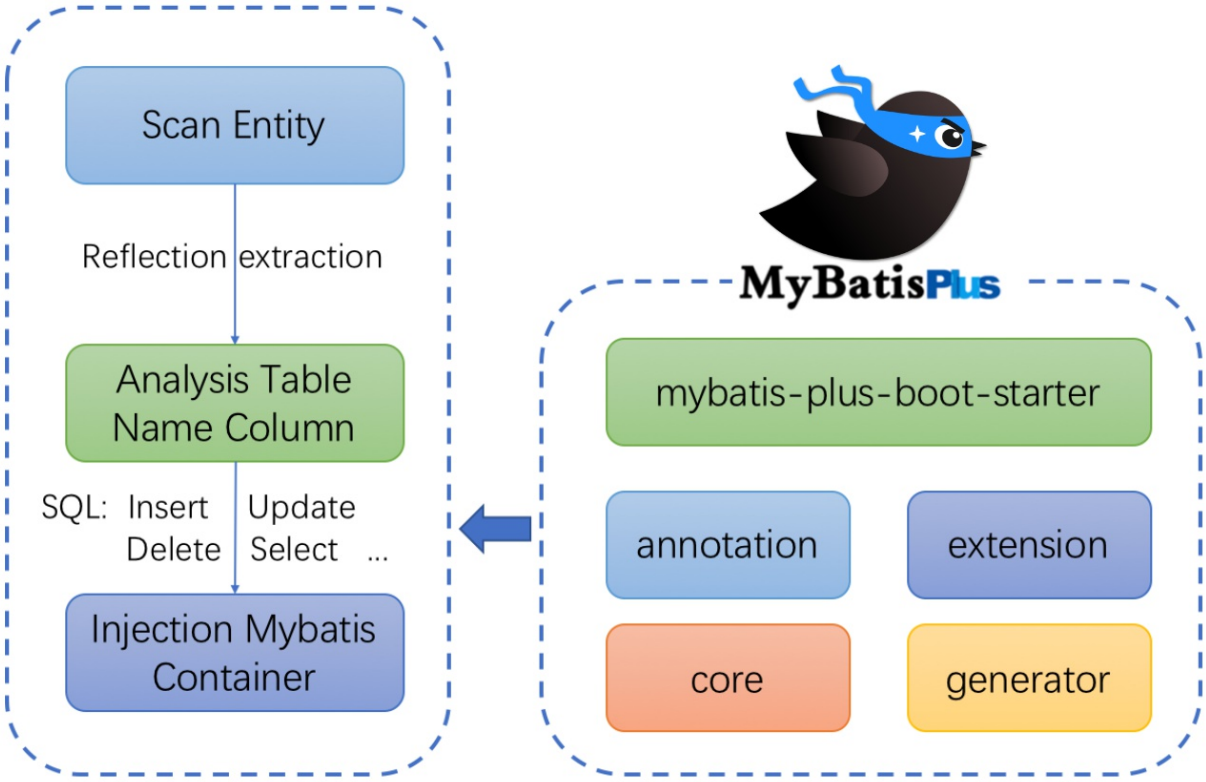
我们的愿景是成为 MyBatis 最好的搭档，就像 魂斗罗 中的 1P、2P，基友搭配，效率翻倍。

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 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 ）
- **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
- **分页插件支持多种数据库**：支持 MySQL、MariaDB、Oracle、DB2、H2、HSQL、SQLite、Postgre、SQLServer2005、SQLServer 等多种数据库
- **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

框架结构



快速开始

环境要求

> jdk8+
>
> mybatis plus 3.2.0
>
> springboot2+
>
> maven3+

数据库

现有一张 `User` 表，其表结构如下：

id	name	age	email
1	Jone	18	test1@baomidou.com
2	Jack	20	test2@baomidou.com
3	Tom	28	test3@baomidou.com
4	Sandy	21	test4@baomidou.com
5	Billie	24	test5@baomidou.com

其对应的数据库SQL脚本如下：

```sql
create database db_mp;

use db_mp;

DROP TABLE IF EXISTS user;
CREATE TABLE user
(
    id BIGINT(20) NOT NULL COMMENT '主键ID',
    name VARCHAR(30) NULL DEFAULT NULL COMMENT '姓名',
    age INT(11) NULL DEFAULT NULL COMMENT '年龄',
    email VARCHAR(50) NULL DEFAULT NULL COMMENT '邮箱',
    PRIMARY KEY (id)
);

DELETE FROM user;
INSERT INTO user (id, name, age, email) VALUES
(1, 'Jone', 18, 'test1@baomidou.com'),
(2, 'Jack', 20, 'test2@baomidou.com'),
(3, 'Tom', 28, 'test3@baomidou.com'),
(4, 'Sandy', 21, 'test4@baomidou.com'),
```

```
(5, 'Billie', 24, 'test5@baomidou.com');
```

初始化工程

加入依赖

> 创建springboot项目，加入 `mybatisplus` 、 `mysql` 等相关依赖

```
<!-- MySQL驱动包 -->
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <scope>runtime</scope>
</dependency>
<!-- MyBatisPlus驱动包 -->
<dependency>
    <groupId>com.baomidou</groupId>
    <artifactId>mybatis-plus-boot-starter</artifactId>
    <version>3.2.0</version>
</dependency>
```

application.properties

> 注意：
>
> 1. 注意mysql驱动版本，高版本驱动路径为com.mysql.cj.jdbc.Driver
> 2. 注意修改数据库信息（数据库名称、账号、密码）

```
#加载驱动
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
#数据库连接路径
spring.datasource.url=jdbc:mysql://localhost:3306/db_mp?
useUnicode=true&characterEncoding=UTF-8&serverTimezone=UTC
#数据库用户名
spring.datasource.username=root
#数据库密码
spring.datasource.password=root
```

启动器

> 在 Spring Boot 启动类中添加 `@MapperScan` 注解，扫描 Mapper 文件夹：

```java
@SpringBootApplication
//扫描加载mapper接口
@MapperScan("com.kazu.mybatisplus.dao")
public class MybatisplusApplication {

    public static void main(String[] args) {
        SpringApplication.run(MybatisplusApplication.class, args);
    }

}
```

实体类

> 创建实体类，建议属性名与数据库表中的列名一致

```java
public class User {
    private Integer id;//用户编号
    private String name;//用户名
    private Integer age;//年龄
    private String email;//邮箱

    public Integer getId() {
        return id;
    }

    public void setId(Integer id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public Integer getAge() {
        return age;
    }

    public void setAge(Integer age) {
        this.age = age;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

```

```
    @Override
    public String toString() {
        return "User{" +
                "id=" + id +
                ", name='" + name + '\'' +
                ", age=" + age +
                ", email='" + email + '\'' +
                '}';
    }
}
```

mapper接口

> 若想使用mp实现快速的CRUD操作，mapper接口集成BaseMapper接口即可

```
/**
 * 自定义Mapper接口集成BaseMapper<T>接口，即可使用MP
 */
public interface UserMapper extends BaseMapper<User> {
}
```

测试类

> springboot项目自带test测试类，在test包下可以找到测试类

```
@RunWith(SpringRunner.class)
@SpringBootTest
public class MybatisplusApplicationTests {

    //注入userMapper接口
    @Resource
    private UserMapper userMapper;

    @Test
    public void test1() {
        //调用mp底层查询方法
       List<User> users = userMapper.selectList(null);
       for (User user : users) {
            System.out.println(user);
       }
    }

}
```

> selectList()方法参数可以为null，可查看源码学习，源码如下：

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

mybatis plus的使用

通用CRUD的使用

提出问题

> 假设我们已存在一张 t_employee 表， 且已有对应的实体类 Employee ， 实现 t_employee 表的 CRUD 操作，我们需要做什么呢?

实现方式

> （1）基于Mybatis
>
> 需要编写EmployeeMapper 接口，并手动编写CRUD 方法
>
> 提供EmployeeMapper.xml 映射文件，并手动编写每个方法对应的SQL 语句.
>
> （2）基于MP
>
> 只需要创建 EmployeeMapper 接口, 并继承 BaseMapper 接口即可
>
> 若需要完成所有操作，不需要创建SQL

数据库文件

> 创建t_employee表，对应实体类为Employee
>
> 问题：表名与实体类名不一致，列名与属性名不一致是否可以成功运行?
>
> **注意：表名与实体类名称不一致时，需要使用@TableName注解指定表名**

```sql
-- 创建表
CREATE TABLE t_employee(
   id INT(11) PRIMARY KEY AUTO_INCREMENT,
   user_name VARCHAR(50),
   email VARCHAR(50),
   gender CHAR(1),
   age INT
);
INSERT INTO t_employee(user_name,email,gender,age) VALUES('Tom','tom@qq.com',1,22);
INSERT INTO t_employee(user_name,email,gender,age) VALUES('Jerry','jerry@qq.com',0,25);
INSERT INTO t_employee(user_name,email,gender,age) VALUES('Black','black@qq.com',1,30);
INSERT INTO t_employee(user_name,email,gender,age) VALUES('White','white@qq.com',0,35);
```

insert新增操作

MP源码

```java
    /**
     * 插入一条记录
     *
     * @param entity 实体对象
     */
    int insert(T entity);
```

实体类

> 遵循JavaBean规范编写实体类，提供get\set方法及toString()方法

```java
public class Employee {
    private Integer id;//编号
    private String userName;//用户名
    private String email;//邮箱
    private Integer gender;//性别
    private Integer age;//年龄
    public Integer getId() {
        return id;
    }
    public void setId(Integer id) {
        this.id = id;
    }
    public String getUserName() {
        return userName;
    }
    public void setUserName(String userName) {
        this.userName = userName;
    }
    public String getEmail() {
        return email;
    }
```

```
    public void setEmail(String email) {
        this.email = email;
    }
    public Integer getGender() {
        return gender;
    }
    public void setGender(Integer gender) {
        this.gender = gender;
    }
    public Integer getAge() {
        return age;
    }
    public void setAge(Integer age) {
        this.age = age;
    }

    @Override
    public String toString() {
        return "Employee{" +
                "id=" + id +
                ", userName='" + userName + '\'' +
                ", email='" + email + '\'' +
                ", gender=" + gender +
                ", age=" + age +
                '}';
    }
}
```

Mapper接口

> 在dao包下创建EmployeeMapper接口，继承BaseMapper接口，无需编写xml映射文件

```
public interface EmployeeMapper extends BaseMapper<Employee> {
}

```

测试新增

> 在test包下创建新的测试类TestEmployee

```
@RunWith(SpringRunner.class)
@SpringBootTest
public class TestEmployee {

    @Resource
    private EmployeeMapper employeeMapper;

```

```
    @Test
    public void testInsert(){
        Employee employee = new Employee();
        employee.setAge(20);
        employee.setEmail("1325698667@qq.com");
        employee.setGender(1);
        employee.setUserName("lucy");
        //新增
        int count  = employeeMapper.insert(employee);
        if(count>0){
            System.out.println("新增成功,主键id为: "+employee.getId());
        }else{
            System.out.println("新增失败");
        }
    }

}
```

错误1-没有设置主键

错误信息

> 运行测试后，发现出现以下错误
>
> org.mybatis.spring.MyBatisSystemException: nested exception is org.apache.ibatis.reflection.ReflectionException: `Could not set property 'id' of 'class com.kazu.mybatisplus.entity.Employee' with value '1158219669003407361'` Cause: java.lang.IllegalArgumentException: argument type mismatch

解决办法

> 在实体类中的主键属性加入注解 `@TableId`

```
//主键生成策略，value属性可选，当属性名与表中的列名不一致，必填
@TableId(value = "id",type = IdType.AUTO)//type:配置主键生成策略，IdType.AUTO表示自增
private Integer id;//编号
```

错误2-表名与实体类名不一致

错误信息

> 运行测试后，发现出现以下错误：

```
org.springframework.jdbc.BadSqlGrammarException:
### Error updating database.  Cause: java.sql.SQLSyntaxErrorException: Table 'db_mybatisplus.employee' doesn't exist
### The error may exist in com/kazu/mybatisplus/dao/EmployeeMapper.java (best guess)
### The error may involve com.kazu.mybatisplus.dao.EmployeeMapper.insert-Inline
### The error occurred while setting parameters
### SQL: INSERT INTO employee  ( user_name, email, gender, age )  VALUES  ( ?, ?, ?, ? )
### Cause: java.sql.SQLSyntaxErrorException: Table 'db_mybatisplus.employee' doesn't exist
; bad SQL grammar []; nested exception is java.sql.SQLSyntaxErrorException: Table 'db_mybatisplus.employee' doesn't exist
```

解决办法

> 在实体类名称上加入注解 `@TableName`

```java
//名称一致时，此注解可以省略
@TableName(value = "t_employee")
public class Employee {
}
```

列名与属性名不一致

错误信息

```
org.springframework.jdbc.BadSqlGrammarException:
### Error updating database.  Cause: java.sql.SQLSyntaxErrorException: Unknown column 'username' in 'field list'
### The error may exist in com/bdqn/dao/EmployeeMapper.java (best guess)
### The error may involve com.bdqn.dao.EmployeeMapper.insert-Inline
### The error occurred while setting parameters
### SQL: INSERT INTO t_employee  ( gender, email, age, username )  VALUES  ( ?, ?, ?, ? )
### Cause: java.sql.SQLSyntaxErrorException: Unknown column 'username' in 'field list'
; bad SQL grammar []; nested exception is java.sql.SQLSyntaxErrorException: Unknown column 'username' in 'field list'
```

列名不存在

解决办法

> 在实体类的属性上加入@TableField注解

> 注意：当表中的列名与实体类中的属性值不一致时，属性名遵循驼峰命名法,可以自动映射，无需加入@TableField注解，如：列名为user_name，属性名为userName【该情况下不需要加注解】

```java
//@TableField注解：当表中列名与实体类的属性名不一致，需要加入该注解
//value属性：该属性可选，当表中的列名与实体类中的属性值不一致时，该属性必填且填写的是列名
//注意：当表中的列名与实体类中的属性值不一致时，属性名遵循驼峰命名法,可以自动映射，无需加入@TableField注解，如：列名为user_name，属性名为userName
@TableField(value = "user_name")
private String username;
```

MP注解的使用

> 介绍 MybatisPlus 注解包相关类详解(更多详细描述可点击查看源码注释)

注解类包：

👉 mybatis-plus-annotation

@TableName

- 描述：表名注解

属性	类型	必须指定	默认值	描述
value	String	否	""	表名
resultMap	String	否	""	xml 中 resultMap 的 id
schema	String	否	""	schema(@since 3.1.1)
keepGlobalPrefix	boolean	否	false	是否保持使用全局的 tablePrefix 的值(如果设置了全局 tablePrefix 且自行设置了 value 的值)(@since 3.1.1)

@TableId

- 描述：主键注解

属性	类型	必须指定	默认值	描述
value	String	否	""	主键字段名
type	Enum	否	IdType.NONE	主键类型

@IdType

值	描述
AUTO	数据库自增
INPUT	自行输入
ID_WORKER	分布式全局唯一ID 长整型类型
UUID	32位UUID字符串
NONE	无状态
ID_WORKER_STR	分布式全局唯一ID 字符串类型

@TableField

- 描述：字段注解(非主键)

属性	类型	必须指定	默认值	描述
value	String	否	""	字段名
exist	boolean	否	true	是否为数据库表字段

> 具体参考官方文档https://mp.baomidou.com/guide/annotation.html#tablefield

排除非表中字段

> 测试非表中字段属性操作，在实体类中加入remark属性，提供get、set方法

```
//备注信息（数据库表中并无此列）
private String remark;
public String getRemark() {
    return remark;
}
public void setRemark(String remark) {
    this.remark = remark;
}

```

> 测试类

```
@Test
public void testInsert(){
    Employee employee = new Employee();
    employee.setAge(20);
    employee.setEmail("1325698667@qq.com");
    employee.setGender(1);
    employee.setUserName("lucy");
    //设置备注
    employee.setRemark("xxx");
    //新增
    int count  = employeeMapper.insert(employee);
    if(count>0){
        System.out.println("新增成功,主键id为: "+employee.getId());
    }else{
        System.out.println("新增失败");
    }
}

```

> 运行测试后发现出现以下错误：

```
org.springframework.jdbc.BadSqlGrammarException:
### Error updating database.  Cause: java.sql.SQLSyntaxErrorException: Unknown column 'remark' in 'field list'
### The error may exist in com/kazu/mybatisplus/dao/EmployeeMapper.java (best guess)
### The error may involve com.kazu.mybatisplus.dao.EmployeeMapper.insert-Inline
### The error occurred while setting parameters
### SQL: INSERT INTO t_employee  ( user_name, email, gender, age, remark )  VALUES  ( ?, ?, ?, ?, ? )
### Cause: java.sql.SQLSyntaxErrorException: Unknown column 'remark' in 'field list'
; bad SQL grammar []; nested exception is java.sql.SQLSyntaxErrorException: Unknown column 'remark' in 'field list'
```

方法1-transient关键字

> 在实体类属性上加入 `transient` 关键字

```
private transient String remark;

```

方法2-@TableField

> 在实体类属性上加入 `@TableField` 注解，设置 `exist` 属性为 `false`

```
//备注信息（数据库表中并无此列）
@TableField(exist = false)
private String remark;
public String getRemark() {
    return remark;
}
public void setRemark(String remark) {
    this.remark = remark;
}

```

条件构造器

构造器简介

> Mybatis-Plus 通过 QueryWrapper（简称 QW，MP 封装的一个查询条件构造器）来让用户自由的构建查询条件，简单便捷，没有额外的负担， 能够有效提高开发效率
>
> 实体包装器，主要用于处理 sql 拼接，排序，实体参数查询等
>
> **注意: 使用的是 `数据库表字段` ，不是 Java 属性!**

条件参数

> 具体条件参数详细用法可参考MP官方文档
>
> https://mp.baomidou.com/guide/wrapper.html

查询方式	说明
setSqlSelect	设置 SELECT 查询字段
where	WHERE 语句，拼接 + WHERE 条件
and	AND 语句，拼接 + AND 字段=值
andNew	AND 语句，拼接 + AND (字段=值)
or	OR 语句，拼接 + OR 字段=值
orNew	OR 语句，拼接 + OR (字段=值)
eq	等于=
allEq	基于 map 内容等于=
ne	不等于<>
gt	大于>
ge	大于等于>=
lt	小于<
le	小于等于<=
like	模糊查询 LIKE
notLike	模糊查询 NOT LIKE
in	IN 查询
notIn	NOT IN 查询
isNull	NULL 值查询
isNotNull	IS NOT NULL
groupBy	分组 GROUP BY
having	HAVING 关键词
orderBy	排序 ORDER BY
orderAsc	ASC 排序 ORDER BY
orderDesc	DESC 排序 ORDER BY
exists	EXISTS 条件语句
notExists	NOT EXISTS 条件语句
between	BETWEEN 条件语句
notBetween	NOT BETWEEN 条件语句
addFilter	自由拼接 SQL
last	拼接在最后，例如：last("LIMIT 1")

delete删除操作

MP删除的方法

> 具体参考BaseMapper接口源码

```
    /**
     * 根据 ID 删除
     *
     * @param id 主键ID
     */
    int deleteById(Serializable id);

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

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

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

```

MP删除的使用

deleteById()

> **根据id主键删除数据**

```java
@Test
public void testDeleteById(){
    //删除
    int count  = employeeMapper.deleteById(8);
    if(count>0){
        System.out.println("删除成功");
    }else{
        System.out.println("删除失败");
    }
}

```

deleteByMap()

> 根据 columnMap 条件，删除记录

```java
@Test
public void testDeleteByMap(){
    Map<String,Object> map = new LinkedHashMap<String, Object>();
    //指定条件列,key为数据库表中的列名
    map.put("user_name","lucy");
    map.put("gender",1);
    //删除
    int count  = employeeMapper.deleteByMap(map);
    if(count>0){
        System.out.println("删除成功");
    }else{
        System.out.println("删除失败");
    }
}

```

delete()

> 根据 entity 条件，删除记录

```java
@Test
public void testDelete(){
    //创建条件构造器对象
    QueryWrapper<Employee> wrapper = new QueryWrapper<Employee>();
    //指定条件,key为数据库表中的列名
    wrapper.eq("id",5);
    //删除
    int count  = employeeMapper.delete(wrapper);
    if(count>0){
        System.out.println("删除成功");
    }else{
        System.out.println("删除失败");
```

```
    }
}

```

deleteBatchIds()

> 删除（根据ID 批量删除）

```
@Test
public void testDeleteBatchIds(){
    //删除
    int count  = employeeMapper.deleteBatchIds(Arrays.asList(1,2,3));
    if(count>0){
        System.out.println("删除成功");
    }else{
        System.out.println("删除失败");
    }
}

```

update更新操作

MP更新的方法

> 具体方法参考源码BaseMapper接口

```
    /**
     * 根据 ID 修改
     *
     * @param entity 实体对象
     */
    int updateById(@Param(Constants.ENTITY) T entity);

    /**
     * 根据 whereEntity 条件，更新记录
     *
     * @param entity        实体对象 (set 条件值,可以为 null)
     * @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)
     */
    int update(@Param(Constants.ENTITY) T entity, @Param(Constants.WRAPPER) Wrapper<T>
updateWrapper);

```

MP更新方法的使用

updateById()

> 根据 ID 修改

```
@Test
public void testUpdateById(){
    Employee employee = new Employee();
    employee.setUserName("王明");
    employee.setId(6);
    //修改
    int count = employeeMapper.updateById(employee);
    if(count>0){
        System.out.println("修改成功");
    }else{
        System.out.println("修改失败");
    }
}

```

update()

> 根据 whereEntity 条件，更新记录

```
@Test
public void testUpdate(){
    UpdateWrapper<Employee> wrapper = new UpdateWrapper<Employee>();
    wrapper.eq("id",6);//指定修改条件
    //修改员工数据
    Employee employee = new Employee();
    employee.setUserName("jason");
    //修改
    int count = employeeMapper.update(employee,wrapper);
    if(count>0){
        System.out.println("修改成功");
    }else{
        System.out.println("修改失败");
    }
}

```

select查询

MP查询的方法

> 列举MyBatisPlus常用的查询方法，具体可参考BaseMapper接口源码

```
    /**
     * 根据 ID 查询
     *
     * @param id 主键ID
     */
    T selectById(Serializable id);

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

    /**
     * 查询（根据 columnMap 条件）
     *
     * @param columnMap 表字段 map 对象
     */
    List<T> selectByMap(@Param(Constants.COLUMN_MAP) Map<String, Object> columnMap);

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

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

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

```

MP查询方法的使用

selectById()

> 根据 ID 查询

```
@Test
public void testSelectById() {
    Employee employee =  this.employeeMapper.selectById(1);
    System.out.println(employee);
}

```

selectBatchIds()

> 查询（根据ID 批量查询）

```
@Test
public void testSelectBatchIds() {
    //创建集合保存员工id
    List<Integer> idList = Arrays.asList(1,2,3);
    //批量查询（使用in()查询）
    List<Employee> list =  this.employeeMapper.selectBatchIds(idList);
    for (Employee employee : list) {
        System.out.println(employee);
    }
}

```

selectByMap()

> 查询（根据 columnMap 条件）

```
@Test
public void testSelectByMap() {
    Map<String, Object> map = new HashMap<String, Object>();
    map.put("id", 1);
    List<Employee> list =  this.employeeMapper.selectByMap(map);
    for (Employee employee : list) {
        System.out.println(employee);
    }
}

```

selectOne()

> 根据 entity 条件，查询一条记录
>
> **注意:最多只能返回1条记录**

```
    @Test
    public void testSelectOne() {
        //创建条件构造器
        QueryWrapper<Employee> queryWrapper = new QueryWrapper<Employee>();
        //指定条件,key为表字段
        queryWrapper.eq("id", 1);
        //返回的结果最多1条，返回多条记录会报错
        Employee employee =  this.employeeMapper.selectOne(queryWrapper);
        System.out.println(employee);
    }

```

selectCount()

> 根据 Wrapper 条件，查询总记录数

```
@Test
public void testSelectCount() {
    //创建条件构造器
    QueryWrapper<Employee> queryWrapper = new QueryWrapper<Employee>();
    //指定条件,key为表字段
    queryWrapper.like("user_name", "t");
    //总数量
    int count = this.employeeMapper.selectCount(queryWrapper);
    System.out.println("总数量: "+count);
}

```

selectList()

> 根据 entity 条件，查询全部记录

```java
@Test
public void testSelectList() {
    //创建条件构造器
    QueryWrapper<Employee> queryWrapper = new QueryWrapper<Employee>();
    //指定条件,key为表字段
    queryWrapper.like("user_name", "c");
    queryWrapper.ge("age", 20);
    List<Employee> list  =  this.employeeMapper.selectList(queryWrapper);
    for (Employee employee : list) {
        System.out.println(employee);
    }
}

```

selectPage()

测试类

```java
    /**
     * 分页查询
     */
    @Test
    public void testSelectPage(){
        //创建分页信息（参数1:当前页码，参数2：每页显示的数量）
        IPage<Employee> page = new Page<Employee>(1,2);
        //创建条件构造器对象
        QueryWrapper<Employee> queryWrapper = new QueryWrapper<Employee>();
        queryWrapper.orderByDesc("id");
        //调用分页查询的方法
        IPage<Employee> employeeIPage = employeeMapper.selectPage(page, queryWrapper);//分页查询
        System.out.println("当前页码："+employeeIPage.getCurrent());
        System.out.println("每页显示数量："+employeeIPage.getSize());
        System.out.println("数据总数量"+employeeIPage.getTotal());
        System.out.println("总页数："+employeeIPage.getPages());
        //获取数据列表
        List<Employee> employees = employeeIPage.getRecords();
        for (Employee employee : employees) {
            System.out.println(employee);
        }
    }
```

配置类

> 注意：在MyBatis Plus中实现分页，需要注入MyBatis Plus分页拦截器
>
> 在com.bdqn.config包下创建MyBatisPlusConfig类，代码如下：

```java
package com.bdqn.config;

import com.baomidou.mybatisplus.extension.plugins.PaginationInterceptor;
import org.mybatis.spring.annotation.MapperScan;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.transaction.annotation.EnableTransactionManagement;

@EnableTransactionManagement//开启事务
@Configuration
@MapperScan("com.bdqn.dao")//加载mp接口
public class MyBatisPlusConfig {
    /**
     * 分页插件
     */
    @Bean
    public PaginationInterceptor paginationInterceptor() {
        PaginationInterceptor paginationInterceptor = new PaginationInterceptor();
        return paginationInterceptor;
    }
}

```

动态条件查询

定义扩展类EmployeeVo

> **注意：该类只做查询条件的参数，不做方法的返回值类型**

```java
package com.bdqn.vo;

import com.bdqn.entity.Employee;

/**
 * 员工扩展类（该类是做查询条件的参数对象）
 * 只做参数不做为返回值
 */
public class EmployeeVo extends Employee {

}
```

测试

```
/**
     * 动态条件查询
     */
@Test
public void test(){
    //参数
    EmployeeVo employeeVo = new EmployeeVo();
    employeeVo.setAge(15);
    employeeVo.setUserName("t");//用户名
    //创建条件构造器
    QueryWrapper<Employee> queryWrapper = new QueryWrapper<Employee>();
    //参数1：是否加入该条件（true:加入）
    //参数2：数据库表的列名
    //参数3：查询的内容
    //姓名模糊查询
  queryWrapper.like(!StringUtils.isEmpty(employeeVo.getUserName()),"user_name",employeeVo.getUserName());
    //年龄范围查询
    queryWrapper.ge(!StringUtils.isEmpty(employeeVo.getAge()),"age",employeeVo.getAge());
    //排序（排序的列名）
    queryWrapper.orderByDesc("id");//根据id降序
    List<Employee> list  =  this.employeeMapper.selectList(queryWrapper);
    for (Employee employee : list) {
        System.out.println(employee);
    }
}
```

自定义SQL查询及分页查询

自定义SQL查询

> 自定义SQL查询，常用的方式有注解方式和xml方式，与MyBatis的用法一致

注解方式

mapper接口

```java
public interface EmployeeMapper extends BaseMapper<Employee> {

    /**
     * 查询员工数据
     * @return
     */
    @Select("select * from t_employee")
    List<Employee> findEmployeeList();
}

```

测试

```java
@Test
public void testFindEmpList() {
    List<Employee> list = this.employeeMapper.findEmployeeList();
    for (Employee employee : list) {
        System.out.println(employee);
    }
}

```

xml方式

mapper接口

```java
public interface UserMapper extends BaseMapper<User> {
    /**
     * 查询用户列表
     * @return
     */
    List<User> findUserList();
}

```

mapper映射文件

> 如果在mapper接口相同目录下创建与接口同名的mapper映射文件，application.properties配置文件则不用进行任何配置（idea开发工具中需要在pom.xml文件中加入以下代码）

```xml
<resources>
    <resource>
        <!-- 编译目录 -->
        <directory>src/main/java</directory>
        <includes>
            <include>**/*.xml</include>
            <include>**/*.properties</include>
        </includes>
        <filtering>false</filtering>
    </resource>
</resources>
```

> 如果mapper映射文件是放在resource资源文件夹下的mapper目录，application.properties配置文件需要进行加载映射文件

```
#加载映射文件
mybatis-plus.mapper-locations=classpath:mapper/*.xml

```

> resource/mapper目录下的UserMapper映射文件：

```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.kazu.mybatisplus.dao.UserMapper">
    <select id="findUserList" resultType="com.kazu.mybatisplus.entity.User">
        select * from user
    </select>
</mapper>

```

测试

```
@Test
public void testFindUserList() {
    //调用自定义查询方法
    List<User> users = userMapper.findUserList();
    for (User user : users) {
        System.out.println(user);
    }
}

```

分页查询

概述

> MyBatisPlus支持分页插件(非PageHelper分页插件)
>
> MyBatisPlus3+版本不支持PageHelper插件
>
> 使用MyBatisPlus分页插件需要编写配置类

分页插件配置类

> 在config包下创建分页插件配置类

```
@EnableTransactionManagement//开启事务
@Configuration
@MapperScan("com.kazu.mybatisplus.dao")//加载mp接口
public class MyBatisPlusConfig {

    /**
     * 分页插件
     */
    @Bean
    public PaginationInterceptor paginationInterceptor() {
        PaginationInterceptor paginationInterceptor = new PaginationInterceptor();
        return paginationInterceptor;
    }
}

```

测试分页插件

```
@Test
public void testSelectPage() {
    //创建条件构造器
    QueryWrapper<Employee> queryWrapper = new QueryWrapper<Employee>();
    queryWrapper.ge("age",15);
    //创建分页对象
    Page<Employee> page = new Page<Employee>(1,2);
    //分页查询
    IPage<Employee> iPage = employeeMapper.selectPage(page,queryWrapper);
    System.out.println("当前页码: " + iPage.getCurrent());
    System.out.println("每页显示数量: " + iPage.getSize());
    System.out.println("总记录数: " + iPage.getTotal());
    System.out.println("总页数: " + iPage.getPages());
    List<Employee> employeeList = iPage.getRecords();//员工数据集合
```

```
    for (Employee employee : employeeList) {
        System.out.println(employee);
    }
}

```

案例-使用接口的方式实现分页

> 第一步：定义分页插件配置类
>
> 第二步：定义mapper接口
>
> 第三步：测试

定义分页插件配置类

> 配置类代码同上

定义mapper接口

```
/**
     * 分页查询员工信息
     * @param page
     * @param employeeVo
     * @return
     */
    IPage<Employee> findEmpByPage(@Param("page") IPage<Employee> page,@Param("employee")
EmployeeVo employeeVo);

```

测试

```
@Test
public void testFindEmployeeListByPage() {
    //创建分页对象
    Page<Employee> page = new Page<Employee>(1,2);
    //分页查询
    IPage<Employee> iPage = employeeMapper.findEmployeeListByPage(page);
    System.out.println("当前页码: " + iPage.getCurrent());
    System.out.println("每页显示数量: " + iPage.getSize());
    System.out.println("总记录数: " + iPage.getTotal());
    System.out.println("总页数: " + iPage.getPages());
    List<Employee> employeeList = iPage.getRecords();//员工数据集合
    for (Employee employee : employeeList) {
        System.out.println(employee);
    }
}
```

```
16
```

通用Service

概述

> 说明:
>
> - 通用 Service CRUD 封装IService接口，进一步封装 CRUD 采用 `get 查询单行` `remove 删除`list 查询集合` `page 分页` 前缀命名方式区分 `Mapper` 层避免混淆，
> - 泛型 `T` 为任意实体对象
> - 建议如果存在自定义通用 Service 方法的可能，请创建自己的 `IBaseService` 继承 `Mybatis-Plus` 提供的基类
> - 对象 `Wrapper` 为 **条件构造器**
> - 使用通用Service进行CRUD操作，业务层接口只需继承 `IService<T>` 接口即可，其中T为泛型。

通用Service的使用

自定义Service接口

> 自定义Service接口，继承IService接口

```
1  public interface UserService extends IService<User> {
2
3  }
4
```

实现Service接口

```
1  package com.bdqn.service.impl;
2
3  import com.baomidou.mybatisplus.extension.service.impl.ServiceImpl;
4  import com.bdqn.dao.UserMapper;
5  import com.bdqn.entity.User;
6  import com.bdqn.service.UserService;
7  import org.springframework.stereotype.Service;
8  import org.springframework.transaction.annotation.Transactional;
```

```
9
10  @Service
11  @Transactional
12  public class UserServiceImpl extends ServiceImpl<UserMapper, User> implements UserService {
13  }
14
```

测试通用Service接口

> 添加测试类，注入自定义Service接口

```
1   @RunWith(SpringRunner.class)
2   @SpringBootTest
3   public class Mybaitsplus02ApplicationTests {
4
5       //注入UserService
6       @Resource
7       private UserService userService;
8
9       /**
10       * 新增
11       */
12      @Test
13      public void testSave() {
14          User user = new User();
15          user.setAge(20);
16          user.setEmail("test@163.com");
17          user.setName("test");
18          //调用新增方法
19          boolean flag = userService.save(user);
20          System.out.println(flag ? "新增成功" :"新增失败");
21      }
22
23      /**
24       * 批量新增
25       */
26      @Test
27      public void testSaveBatch() {
28          User user1 = new User();
29          user1.setAge(22);
30          user1.setEmail("lucy@163.com");
31          user1.setName("露西");
32
33          User user2 = new User();
34          user2.setAge(20);
35          user2.setEmail("lulu@163.com");
36          user2.setName("露露");
37          //调用新增方法
38          boolean flag = userService.saveBatch(Arrays.asList(user1,user2));
39          System.out.println(flag ? "新增成功" :"新增失败");
```

```
    }
    /**
     * 批量新增或修改
     */
    @Test
    public void testSaveOrUpdateBatch() {
        User user1 = new User();
        user1.setId(9L);
        user1.setAge(22);
        user1.setEmail("ross@163.com");
        user1.setName("肉丝");

        User user2 = new User();
        user2.setAge(20);
        user2.setEmail("harry@163.com");
        user2.setName("harry");
        //调用新增方法
        boolean flag = userService.saveOrUpdateBatch(Arrays.asList(user1,user2));
        System.out.println(flag ? "成功" :"失败");
    }

    /**
     * 查询唯一
     */
    @Test
    public void testGetOne() {
        QueryWrapper<User> queryWrapper = new QueryWrapper<User>();
        queryWrapper.eq("age",20);
        //会报错
        //User user = userService.getOne(queryWrapper);
        //不会报错，但会发出警告
        User user = userService.getOne(queryWrapper,false);
        System.out.println(user);
    }
}

```

> 注意：其余方法类似，不一一进行代码演示，具体方法可参考IService接口源码或参考**官方文档**

代码生成器

概述

> MyBatisPlus提供了强大的代码生成器，可生成实体类、mapper接口、mapper映射文件、service接口及实现类、Controller控制器等
>
> 使用MyBatisPlus提供的代码生成器步骤：
>
> - 第一步：添加相关依赖
> - 第二步：编写代码生成器类

代码生成器的使用

添加相关依赖

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>

<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-tomcat</artifactId>
    <scope>provided</scope>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
</dependency>
<!-- MySQL驱动包 -->
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <scope>runtime</scope>
</dependency>
<!-- MyBatisPlus驱动包 -->
<dependency>
    <groupId>com.baomidou</groupId>
    <artifactId>mybatis-plus-boot-starter</artifactId>
    <version>3.2.0</version>
</dependency>
<!-- druid 数据库连接池 -->
<dependency>
    <groupId>com.alibaba</groupId>
    <artifactId>druid</artifactId>
    <version>1.0.9</version>
</dependency>


<!-- 以下是代码生成器的jar依赖 -->
```

```xml
<!-- 代码生成器核心jar依赖 -->
<dependency>
    <groupId>com.baomidou</groupId>
    <artifactId>mybatis-plus-generator</artifactId>
    <version>3.2.0</version>
</dependency>

<!-- 使用默认的velocity模板 -->
<dependency>
    <groupId>org.apache.velocity</groupId>
    <artifactId>velocity-engine-core</artifactId>
    <version>2.0</version>
</dependency>

<!-- sfl4j -->
<dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-api</artifactId>
</dependency>
<dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-log4j12</artifactId>
</dependency>

```

编写代码生成器类

```java
@RunWith(SpringRunner.class)
@SpringBootTest
public class TestMP {

    private static String author ="KazuGin";//作者名称
    private static String outputDir ="E:\\";//生成的位置
    private static String driver ="com.mysql.cj.jdbc.Driver";//驱动，注意版本
    //连接路径,注意修改数据库名称
    private static String url ="jdbc:mysql://localhost:3306/db_mybatisplus?
useUnicode=true&characterEncoding=UTF-8&serverTimezone=UTC";
    private static String username ="root";//数据库用户名
    private static String password ="root";//数据库密码
    private static String tablePrefix ="t_";//数据库表的前缀，如t_user
    private static String [] tables = {"t_employee","user"};     //生成的表
    private static String parentPackage = "com.kazu.mybatisplus02";//顶级包结构
    private static String dao = "dao";//数据访问层包名称
    private static String service = "service";//业务逻辑层包名称
    private static String entity = "entity";//实体层包名称
    private static String controller = "controller";//控制器层包名称
    private static String mapperxml = "dao";//mapper映射文件包名称


    /**
     * 代码生成     示例代码
     */
    @Test
    public void  testGenerator() {
        //1. 全局配置
        GlobalConfig config = new GlobalConfig();
```

```
        config.setAuthor(author) // 作者
                .setOutputDir(outputDir) // 生成路径
                .setFileOverride(true)  // 文件覆盖
                .setIdType(IdType.AUTO) // 主键策略
                .setServiceName("%sService")  // 设置生成的service接口的名字的首字母是否为I,
加%s则不生成I
                .setBaseResultMap(true) //映射文件中是否生成ResultMap配置
                .setBaseColumnList(true);   //生成通用sql字段

        //2. 数据源配置
        DataSourceConfig dsConfig  = new DataSourceConfig();
        dsConfig.setDbType(DbType.MYSQL)  // 设置数据库类型
                .setDriverName(driver)  //设置驱动
                .setUrl(url)            //设置连接路径
                .setUsername(username)  //设置用户名
                .setPassword(password); //设置密码

        //3. 策略配置
        StrategyConfig stConfig = new StrategyConfig();
        stConfig.setCapitalMode(true) //全局大写命名
                .setNaming(NamingStrategy.underline_to_camel) // 数据库表映射到实体的命名策略
                .setTablePrefix(tablePrefix) //表前缀
                .setInclude(tables);  // 生成的表

        //4. 包名策略配置
        PackageConfig pkConfig = new PackageConfig();
        pkConfig.setParent(parentPackage)//顶级包结构
                .setMapper(dao) //数据访问层
                .setService(service)    //业务逻辑层
                .setController(controller)  //控制器
                .setEntity(entity)  //实体类
                .setXml(mapperxml); //mapper映射文件

        //5. 整合配置
        AutoGenerator ag = new AutoGenerator();
        ag.setGlobalConfig(config)
                .setDataSource(dsConfig)
                .setStrategy(stConfig)
                .setPackageInfo(pkConfig);
        //6. 执行
        ag.execute();
    }
}

```