MybatisPlus

- 官网

https://mp.baomidou.com/： MybatisPlus官网

> 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、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

快速入门

- Maven

```xml
<!--MybatisPlus-->
<dependency>
    <groupId>com.baomidou</groupId>
    <artifactId>mybatis-plus-boot-starter</artifactId>
    <version>3.0.5.tmp</version>
</dependency>
<dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
</dependency>
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
</dependency>
```

- 数据库

```
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');
```

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

> 注意：
>
> 1. 不要同时导入Mybatis和Mybatis-plus，因为有一些依赖的区别

- 配置

```
# Mysql5：com.mysql.jdbc.Driver，Mysql8：com.mysql.cj.jdbc.Driver，并且需要增加时区配置
# 8版本兼容低版本，所以也可以使用com.mysql.cj.jdbc.Driver
spring.datasource.username=root
spring.datasource.password=root
spring.datasource.url=jdbc:mysql://localhost:3306/mybatis_plus?
useSSL=false&useUnicode=true&characterEncoding=utf-8&serverTimezone=Asia/Shanghai
spring.datasource.driver-class-name=com.mysql.jdbc.Driver
```

- POJO-DAO 传统方式：很复杂
- POJO-DAO 使用了MybatisPlus之后

1. POJO

```java
package com.bean.pojo;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@AllArgsConstructor
@NoArgsConstructor
public class User {

    private Long id;
    private String name;
    private Integer age;
    private String email;

}
```

2. Mapper接口

```java
package com.bean.mapper;


import com.baomidou.mybatisplus.core.mapper.BaseMapper;
import com.bean.pojo.User;
import org.apache.ibatis.annotations.Mapper;
import org.springframework.stereotype.Repository;

//在对应的mapper上实现基本的接口BaseMapper
@Repository
public interface UserMapper extends BaseMapper<User> {
}
```

> 配完了，因为继承了父类，所以所有的方法都基于父类，但是也可以自己编写服务

3. 扫描包

```java
package com.bean;

import org.mybatis.spring.annotation.MapperScan;
import org.springframework.boot.SpringApplication;
```

```java
import org.springframework.boot.autoconfigure.SpringBootApplication;

@MapperScan("com.bean.mapper")
@SpringBootApplication
public class MybatisPlusApplication {

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

}
```

4. 使用

```java
package com.bean;

import com.bean.mapper.UserMapper;
import com.bean.pojo.User;
import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;

import java.util.List;

@SpringBootTest
class MybatisPlusApplicationTests {

    @Autowired
    private UserMapper userMapper;

    @Test
    void contextLoads() {

        //查询所有用户，参数是一个条件构造器，不用就写null
        List<User> users = userMapper.selectList(null);

        for (User user : users) {
            System.out.println(user);
        }

    }

}
```

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

配置日志输出

所有的SQL都是不可见的，我们要知道这个是怎么执行的，所以要配置日志

```
# 配置日志
mybatis-plus.configuration.log-impl=org.apache.ibatis.logging.stdout.StdOutImpl
```

> Log4j和Sel4j都需要导包，这里就是用默认的可以了

配置完成之后，后面的学习就需要注意这个自动生成的SQL

```
JDBC Connection [HikariProxyConnection@157201184 wrapping
com.mysql.cj.jdbc.ConnectionImpl@2f521c4] will not be managed by Spring
==>  Preparing: SELECT id,name,age,email FROM user
==> Parameters:
<==    Columns: id, name, age, email
<==        Row: 1, Jone, 18, test1@baomidou.com
<==        Row: 2, Jack, 20, test2@baomidou.com
<==        Row: 3, Tom, 28, test3@baomidou.com
<==        Row: 4, Sandy, 21, test4@baomidou.com
<==        Row: 5, Billie, 24, test5@baomidou.com
<==      Total: 5
Closing non transactional SqlSession
[org.apache.ibatis.session.defaults.DefaultSqlSession@71e35c4]
User(id=1, name=Jone, age=18, email=test1@baomidou.com)
User(id=2, name=Jack, age=20, email=test2@baomidou.com)
User(id=3, name=Tom, age=28, email=test3@baomidou.com)
User(id=4, name=Sandy, age=21, email=test4@baomidou.com)
User(id=5, name=Billie, age=24, email=test5@baomidou.com)
```

插入测试和雪花算法

CRUD扩展

插入

```
    @Test
    void testInsert() {

        User user = new User().setName("BEAN").setAge(20).setEmail("xxx@qq.com");
        System.out.println(user);
        int insert = userMapper.insert(user);
        System.out.println(insert);
    }
```

```
User(id=null, name=BEAN, age=20, email=xxx@qq.com)
Creating a new SqlSession
SqlSession [org.apache.ibatis.session.defaults.DefaultSqlSession@6fc6deb7] was not registered
for synchronization because synchronization is not active
JDBC Connection [HikariProxyConnection@232200992 wrapping
com.mysql.cj.jdbc.ConnectionImpl@767f6ee7] will not be managed by Spring
==>  Preparing: INSERT INTO user ( id, name, age, email ) VALUES ( ?, ?, ?, ? )
==> Parameters: 1240941797300097026(Long), BEAN(String), 20(Integer), xxx@qq.com(String)
<==    Updates: 1
```

	id	name	age	email
1	1	Jone	18	test1@baomidou.com
2	2	Jack	20	test2@baomidou.com
3	3	Tom	28	test3@baomidou.com
4	4	Sandy	21	test4@baomidou.com
5	5	Billie	24	test5@baomidou.com
6	1240941797300097026	BEAN	20	xxx@qq.com

> 虽然ID没有写，但是我们发现自动生成了id
>
> 数据库插入的id的默认值为唯一id

主键生成策略

主键生成策略

```
AUTO(0),         //自增
NONE(1),         //不操作
INPUT(2),        //手动输入
ID_WORKER(3),    //全局唯一，默认方案
UUID(4),         //UUID
ID_WORKER_STR(5);    //默认方案的字符串表示方案
```

比较多的：uuid，自增id，雪花算法，redis，zookeeper

雪花算法

雪花算法是推特开源的分布式ID生成算法，结果是一个Long类型的ID。其核心思想是：

使用41bit作为毫秒数

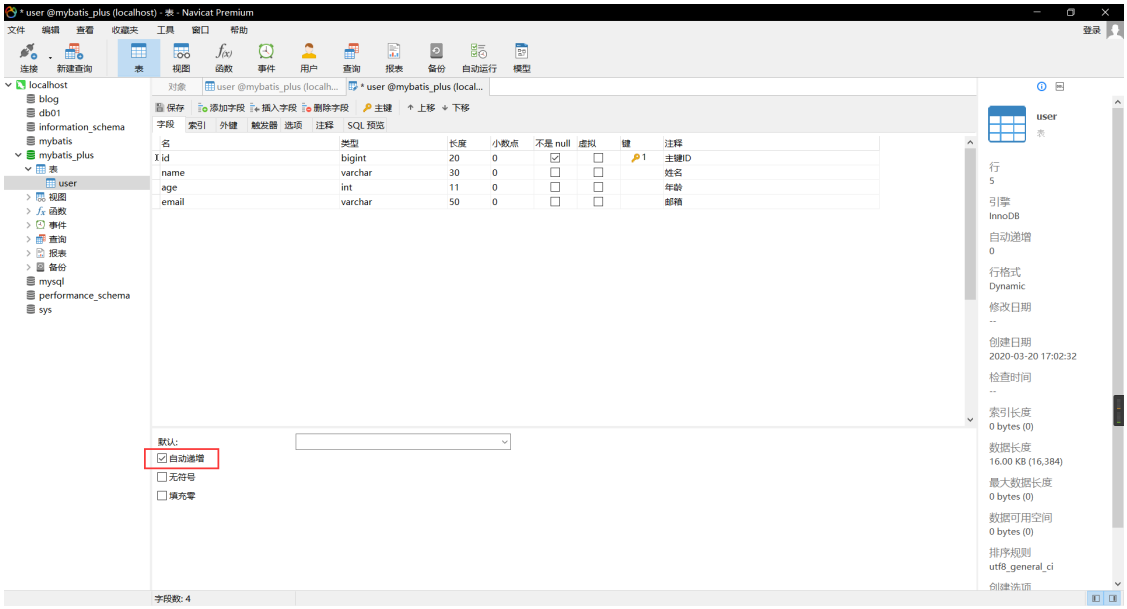
- 10bit作为机器的ID（5bit为数据中心【不同的数据中心，比如北京，上海，等等】，5bit为机器ID）
- 12bit作为毫秒内的流水号
- 最后一个符号位为0

主键自增

1. 在实体类上的主键加上注解 `@TableId(type = IdType.AUTO)`

```java
@Data
@AllArgsConstructor
@NoArgsConstructor
@Accessors(chain = true)
public class User {

    @TableId(type = IdType.AUTO)
    private Long id;
    private String name;
    private Integer age;
    private String email;

}
```

2. 数据库的主键字段一定要是自增的



测试：

	id	name	age	email
1	1	Jone	18	test1@baomidou.com
2	2	Jack	20	test2@baomidou.com
3	3	Tom	28	test3@baomidou.com
4	4	Sandy	21	test4@baomidou.com
5	5	Billie	24	test5@baomidou.com
6	1240941797300097026	BEAN	20	xxx@qq.com
7	1240941797300097027	BEAN	20	xxx@qq.com

> 其余的和这个差不多，就不写了

更新

```java
@Test
void testUpdate() {
    User user = new User().setId(1240941797300097026L).setName("howling").setAge(20).setEmail("xxx@163.com");
    userMapper.updateById(user);//这里需要的是对象不是id
}
```

	id	name	age	email
1	1	Jone	18	test1@baomidou.com
2	2	Jack	20	test2@baomidou.com
3	3	Tom	28	test3@baomidou.com
4	4	Sandy	21	test4@baomidou.com
5	5	Billie	24	test5@baomidou.com
6	1240941797300097026	howling	20	xxx@163.com
7	1240941797300097027	BEAN	20	xxx@qq.com

```
==>  Preparing: UPDATE user SET name=?, age=?, email=? WHERE id=?
==> Parameters: howling(String), 20(Integer), xxx@163.com(String), 1240941797300097026(Long)
<==    Updates: 1
```

看到这里的，我们惊喜的发现mybatisPlus可以根据条件自动拼接动态SQL，所以以后我们就不需要对动态SQL费脑筋了

自动填充

- 创建时间
- 修改时间

一般这个操作就是自动化完成的，我们不希望手动更新

阿里巴巴的开发手册中写道：所有的数据库表几乎都要有创建时间，修改时间这两个字段，而且需要自动化

方式一：数据库级别修改（工作中不建议使用）

1. 在表中新增字段：`create_time`，`update_time`

列名	数据类型	长度	默认	主键?	非空?	Unsigned	自增?	Zerofill?	更新	注释
id	bigint	20		☑	☑	☐	☑	☐	☐	主键ID
name	varchar	30		☐	☐	☐	☐	☐	☐	姓名
age	int	11		☐	☐	☐	☐	☐	☐	年龄
email	varchar	50	时间戳	☐	☐	☐	☐	☐	☐	邮箱
create_time	datetime		CURRENT_TIMESTAMP	☐	☐	☐	☐	☐	自动更新	创建时间
update_time	datetime		CURRENT_TIMESTAMP	☐	☐	☐	☐	☐	☑	更新时间
				☐	☐	☐	☐	☐	☐	

2. 把实体类同步

```java
@Data
@AllArgsConstructor
@NoArgsConstructor
```

```java
@Accessors(chain = true)
public class User {

    @TableId(type = IdType.AUTO)
    private Long id;
    private String name;
    private Integer age;
    private String email;
    private Data createTime;
    private Data updateTime;

}
```

3. 插入

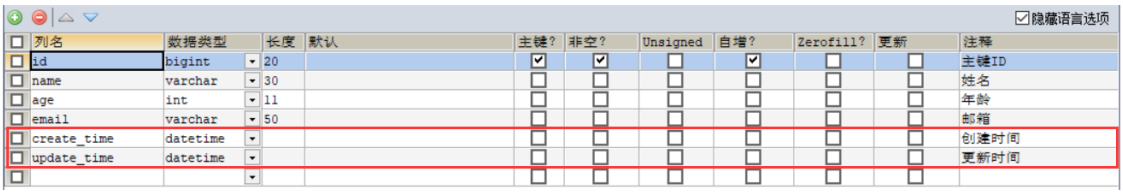
```java
@Test
void testInsert() {

    User user = new User().setName("BEAN").setAge(20).setEmail("xxx@qq.com");
    System.out.println(user);
    int insert = userMapper.insert(user);
    System.out.println(insert);
}
```

4. 查看

	id	name	age	email	create_time	update_time
1	1	Jone	18	test1@baomidou.com	<null>	<null>
2	2	Jack	20	test2@baomidou.com	<null>	<null>
3	3	Tom	28	test3@baomidou.com	<null>	<null>
4	4	Sandy	21	test4@baomidou.com	<null>	<null>
5	5	Billie	24	test5@baomidou.com	<null>	<null>
6	1240941797300097026	howling	20	xxx@163.com	<null>	<null>
7	1240941797300097027	BEAN	20	xxx@qq.com	<null>	<null>
8	1240941797300097028	BEAN	20	xxx@qq.com	2020-03-24 10:13:47	2020-03-24 10:13:47

方式二：代码级别（工作中建议使用）

1. 删除数据库的默认值



2. 实体类的字段属性上需要增加注解

   - 字段属性

     ```java
     @TableField
     ```

   - 填充策略

```
    public enum FieldFill {
        DEFAULT,     //默认，不操作
        INSERT,      //插入时操作
        UPDATE,      //更新时操作
        INSERT_UPDATE;  //插入和更新时操作

        private FieldFill() {
        }
    }
```

```
@Data
@AllArgsConstructor
@NoArgsConstructor
@Accessors(chain = true)
public class User {

    @TableId(type = IdType.AUTO)
    private Long id;
    private String name;
    private Integer age;
    private String email;
    @TableField(fill = FieldFill.INSERT)
    private Date createTime;
    @TableField(fill = FieldFill.INSERT_UPDATE)
    private Date updateTime;

}
```

3. 编写处理器处理这个注解

```
@Component  //把处理器加入到IOC中
@Slf4j      //日志，可以看清楚具体做了什么
public class MyDataObjectHandler implements MetaObjectHandler {

    //插入时的填充策略
    @Override
    public void insertFill(MetaObject metaObject) {
        log.info("start insert fill....");
        //String fieldName：字段名, Object fieldVal 插入的字段值, MetaObject
metaObject
        this.setFieldValByName("createTime",new Date(),metaObject);
        this.setFieldValByName("updateTime",new Date(),metaObject);
    }

    //更新时的填充策略
    @Override
    public void updateFill(MetaObject metaObject) {
        log.info("start update fill....");
        this.setFieldValByName("updateTime",new Date(),metaObject);
    }
}
```

4. 实验插入和更新
    - 插入

```java
@Test
void testInsert() {

    User user = new User().setName("BEAN").setAge(20).setEmail("xxx@qq.com");
    System.out.println(user);
    int insert = userMapper.insert(user);
    System.out.println(insert);
}
```

	id	name	age	email	create_time	update_time
1	1	Jone	18	test1@baomidou.com	<null>	<null>
2	2	Jack	20	test2@baomidou.com	<null>	<null>
3	3	Tom	28	test3@baomidou.com	<null>	<null>
4	4	Sandy	21	test4@baomidou.com	<null>	<null>
5	5	Billie	24	test5@baomidou.com	<null>	<null>
6	1240941797300097026	howling	20	xxx@163.com	<null>	<null>
7	1240941797300097027	BEAN	20	xxx@qq.com	<null>	<null>
8	1240941797300097028	BEAN	20	xxx@qq.com	2020-03-24 10:13:47	2020-03-24 10:13:47
9	1240941797300097029	BEAN	20	xxx@qq.com	2020-03-24 10:34:58	2020-03-24 10:34:58

- 更新

```java
 @Test
 void testUpdate() {
     User user = new User().setId(1240941797300097026L).setName("howling").setAge(20).setEmail("xxx@163.com");
     userMapper.updateById(user);//这里需要的是user不是id
 }
```

	id	name	age	email	create_time	update_time
1	1	Jone	18	test1@baomidou.com	<null>	<null>
2	2	Jack	20	test2@baomidou.com	<null>	<null>
3	3	Tom	28	test3@baomidou.com	<null>	<null>
4	4	Sandy	21	test4@baomidou.com	<null>	<null>
5	5	Billie	24	test5@baomidou.com	<null>	<null>
6	1240941797300097026	howling	20	xxx@163.com	<null>	2020-03-24 10:36:52
7	1240941797300097027	BEAN	20	xxx@qq.com	<null>	<null>
8	1240941797300097028	BEAN	20	xxx@qq.com	2020-03-24 10:13:47	2020-03-24 10:13:47
9	1240941797300097029	BEAN	20	xxx@qq.com	2020-03-24 10:34:58	2020-03-24 10:34:58

乐观锁

在面试过程中，我们经常被问到乐观锁

其实这个非常简单

乐观锁

非常乐观，总是认为不会出现问题，无论干什么都不会去上锁。

如果出现了问题就再次更新值测试

悲观锁

十分悲观，认为总是会出现问题，无论干什么都会上锁再去操作。

我们主要讲解乐观锁

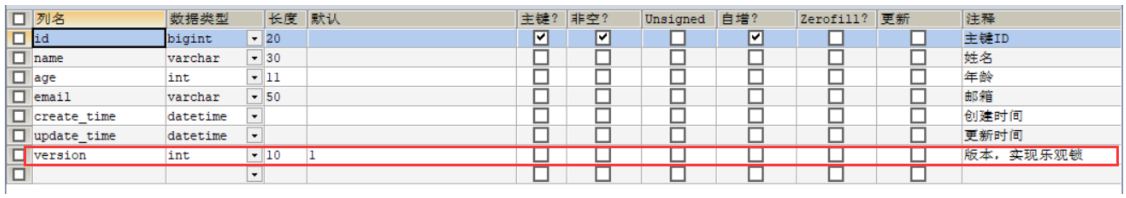
乐观锁实现方式：

- 取出记录时，获取当前version（版本）
- 更新时带上这个version
- 执行更新时， `set version = new Version where version = oldVersion`
- 假如version不对，就更新失败

```
#乐观锁： 1. 先查询，获得版本号(这里版本为1)
-- A线程，在B线程之后执行，因为version为2了，所以不会执行，这样保证了安全性
update user set name = "BEAN", version = version+1
where id=2 and version=1

-- B线程，抢先完成了，这个时候version就为2了，这样会导致A失败，保证了安全性
update user set name = "BEAN" , version = version+1
where id=2 and version=1
```

乐观锁的实现方式

1. 首先在数据库中增加一个 `version` 字段，默认值为1作为初始版本



列名	数据类型	长度	默认	主键?	非空?	Unsigned	自增?	Zerofill?	更新	注释
id	bigint	20		☑	☑	☐	☑	☐	☐	主键ID
name	varchar	30		☐	☐	☐	☐	☐	☐	姓名
age	int	11		☐	☐	☐	☐	☐	☐	年龄
email	varchar	50		☐	☐	☐	☐	☐	☐	邮箱
create_time	datetime			☐	☐	☐	☐	☐	☐	创建时间
update_time	datetime			☐	☐	☐	☐	☐	☐	更新时间
version	int	10	1	☐	☐	☐	☐	☐	☐	版本，实现乐观锁
				☐	☐	☐	☐	☐	☐	

	id	name	age	email	create_time	update_time	version
1	1	Jone	18	test1@baomidou.com	<null>	<null>	1
2	2	Jack	20	test2@baomidou.com	<null>	<null>	1
3	3	Tom	28	test3@baomidou.com	<null>	<null>	1
4	4	Sandy	21	test4@baomidou.com	<null>	<null>	1
5	5	Billie	24	test5@baomidou.com	<null>	<null>	1
6	1240941797300097026	howling	20	xxx@163.com	<null>	2020-03-24 10:36:52	1
7	1240941797300097027	BEAN	20	xxx@qq.com	<null>	<null>	1
8	1240941797300097028	BEAN	20	xxx@qq.com	2020-03-24 10:13:47	2020-03-24 10:13:47	1
9	1240941797300097029	BEAN	20	xxx@qq.com	2020-03-24 10:34:58	2020-03-24 10:34:58	1

2. 实体类增加对应的字段，并增加乐观锁注解

```
@Data
@AllArgsConstructor
@NoArgsConstructor
@Accessors(chain = true)
public class User {

    @TableId(type = IdType.AUTO)
    private Long id;
    private String name;
    private Integer age;
    private String email;
    @TableField(fill = FieldFill.INSERT)
    private Date createTime;
    @TableField(fill = FieldFill.INSERT_UPDATE)
    private Date updateTime;
    @Version //这是一个乐观锁的注解
    private Integer version;

```

```
    }
```

3. 注册组件

```
@MapperScan("com.bean.mapper") //扫描包
@EnableTransactionManagement//事物注解
@Configuration
public class MybatisPlusConfig {

    //注册乐观锁注解
    @Bean
    public OptimisticLockerInterceptor optimisticLockerInterceptor() {
        return new OptimisticLockerInterceptor();
    }
}
```

4. 测试

```
    //测试乐观锁，成功案例
    @Test
    void testOptimisticLockerSuccess() {

        //1. 查询用户信息
        User user = userMapper.selectById(1L);

        //2. 修改用户信息
        user.setName("ISHOWLING").setEmail("123@123.com");
        userMapper.updateById(user);
    }
```

	id	name	age	email	create_time	update_time	version
1	1	ISHOWLING	18	123@123.com	<null>	2020-03-24 10:55:55	2
2	2	Jack	20	test2@baomidou.com	<null>	<null>	1
3	3	Tom	28	test3@baomidou.com	<null>	<null>	1
4	4	Sandy	21	test4@baomidou.com	<null>	<null>	1
5	5	Billie	24	test5@baomidou.com	<null>	<null>	1
6	1240941797300097026	howling	20	xxx@163.com	<null>	2020-03-24 10:36:52	1
7	1240941797300097027	BEAN	20	xxx@qq.com	<null>	<null>	1
8	1240941797300097028	BEAN	20	xxx@qq.com	2020-03-24 10:13:47	2020-03-24 10:13:47	1
9	1240941797300097029	BEAN	20	xxx@qq.com	2020-03-24 10:34:58	2020-03-24 10:34:58	1

```
    //测试乐观锁，失败案例
    @Test
    void testOptimisticLockerFail() {
        //线程1
        User user = userMapper.selectById(1L);
        user.setName("ISHOWLING").setEmail("123@123.com");

        //线程2，模拟另外的线程执行插队操作
        User user2 = userMapper.selectById(1L);
        user2.setName("ISHOWLINGGGGG").setEmail("123123123@123.com");
        //线程2先执行
        userMapper.updateById(user2);

        //线程1后执行
        userMapper.updateById(user);

    }
```

	id	name	age	email	create_time	update_time	version
1	1	ISHOWLINGGGGG	18	123123123@123.com	<null>	2020-03-24 10:58:29	3
2	2	Jack	20	test2@baomidou.com	<null>	<null>	1
3	3	Tom	28	test3@baomidou.com	<null>	<null>	1
4	4	Sandy	21	test4@baomidou.com	<null>	<null>	1
5	5	Billie	24	test5@baomidou.com	<null>	<null>	1
6	1240941797300097026	howling	20	xxx@163.com	<null>	2020-03-24 10:36:52	1
7	1240941797300097027	BEAN	20	xxx@qq.com	<null>	<null>	1
8	1240941797300097028	BEAN	20	xxx@qq.com	2020-03-24 10:13:47	2020-03-24 10:13:47	1
9	1240941797300097029	BEAN	20	xxx@qq.com	2020-03-24 10:34:58	2020-03-24 10:34:58	1

> 我们发现线程1的没有执行，所以成功了
>
> 注意，多线程的一定要加锁

查询

在快速入门的时候查询就搞定了，但是还是要来几个

单个ID查询，多个ID查询，使用Map的条件查询

```
    @Test
    void testSelect() {

        //查询单个ID
        User user = userMapper.selectById(1);
        System.out.println(user);

        //测试批量查询ID
        List<User> users = userMapper.selectBatchIds(Arrays.asList(1, 2, 3));
        users.forEach(System.out::println);

        //条件查询之一：使用Map操作
        HashMap<String, Object> map = new HashMap<>();
        map.put("name","BEAN");
        map.put("age",20);
        List<User> list = userMapper.selectByMap(map);
        list.forEach(System.out::println);
    }
```

分页查询

分页在网站使用的十分多

1. 原始的：limit
2. pageHelper等第三方插件
3. MybatisPlus内置的插件

我们使用MybatisPlus的分页插件：官网上讲的

1. 配置拦截器组件

```
@MapperScan("com.bean.mapper") //扫描包
@EnableTransactionManagement//事物注解
@Configuration
```

```
public class MybatisPlusConfig {

    @Bean
    public PaginationInterceptor paginationInterceptor() {
        PaginationInterceptor paginationInterceptor = new PaginationInterceptor();
        // 设置请求的页面大于最大页后操作， true调回到首页，false 继续请求  默认false
        // paginationInterceptor.setOverflow(false);
        // 设置最大单页限制数量，默认 500 条，-1 不受限制
        // paginationInterceptor.setLimit(500);
        // 开启 count 的 join 优化,只针对部分 left join
        paginationInterceptor.setCountSqlParser(new JsqlParserCountOptimize(true));
        return paginationInterceptor;
    }
}
```

但是我们不需要这么多功能，只需要：

```
    @Bean
    public PaginationInterceptor paginationInterceptor() {
        return new PaginationInterceptor();
    }
```

2. 使用Page对象

```
import com.baomidou.mybatisplus.extension.plugins.pagination.Page;
    @Test
    void testPage() {
        //参数一：当前页，参数二：页面中的数据，这里就是显示第一页的5条数据
        Page<User> page = new Page<>(1,5);
        userMapper.selectPage(page,null);
        page.getRecords().forEach(System.out::println);
    }
```

```
<==    Columns: id, name, age, email, create_time, update_time, version
<==        Row: 1, ISHOWLINGGGGG, 18, 123123123@123.com, null, 2020-03-24 10:58:29, 3
<==        Row: 2, Jack, 20, test2@baomidou.com, null, null, 1
<==        Row: 3, Tom, 28, test3@baomidou.com, null, null, 1
<==        Row: 4, Sandy, 21, test4@baomidou.com, null, null, 1
<==        Row: 5, Billie, 24, test5@baomidou.com, null, null, 1
<==      Total: 5
```

```
    @Test
    void testPage() {
        //参数一：当前页，参数二：页面大小
        Page<User> page = new Page<>(2,5);
        userMapper.selectPage(page,null);
        page.getRecords().forEach(System.out::println);
    }
```

```
<==    Columns: id, name, age, email, create_time, update_time, version
<==        Row: 1240941797300097026, howling, 20, xxx@163.com, null, 2020-03-24 10:36:52, 1
<==        Row: 1240941797300097027, BEAN, 20, xxx@qq.com, null, null, 1
<==        Row: 1240941797300097028, BEAN, 20, xxx@qq.com, 2020-03-24 10:13:47, 2020-03-24 10:13:47, 1
<==        Row: 1240941797300097029, BEAN, 20, xxx@qq.com, 2020-03-24 10:34:58, 2020-03-24 10:34:58, 1
<==      Total: 4
```

> 第二次查询没有5条了，只有4条
>
> 使用了分页插件之后，一切都简单了

删除

基本删除

```
    @Test
    void testDelete() {
        //根据ID删除
        userMapper.deleteById(1240941797300097027L);

        //批量删除
        userMapper.deleteBatchIds(Arrays.asList(1240941797300097028L,1240941797300097029L));

        //根据Map删除
        HashMap<String, Object> map = new HashMap<>();
        map.put("name","howling");
        userMapper.deleteByMap(map);

    }
```

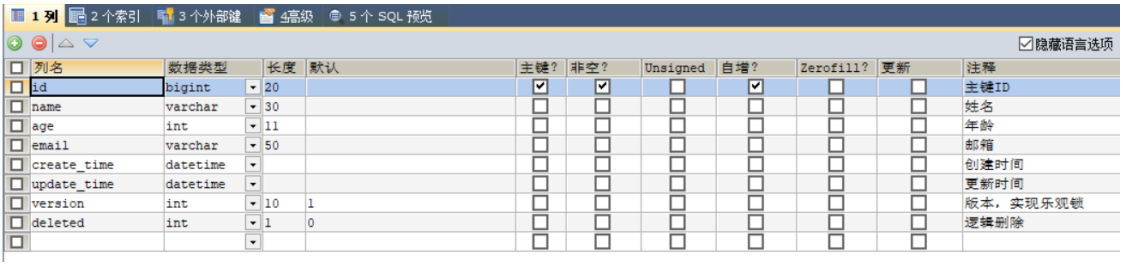
> 在工作中，会遇到逻辑删除

逻辑删除

我们分为两个删除

- 物理删除：在数据库中删除
- 逻辑删除：不在数据库中删除，而是通过一个变量让他失效，比如 `delete=0=>delete=1` 类似回收站

1. 在数据库中增加一个 `delete` 字段，默认为0，不删除



列名	数据类型	长度	默认	主键?	非空?	Unsigned	自增?	Zerofill?	更新	注释
id	bigint	20		☑	☑	☐	☑	☐	☐	主键ID
name	varchar	30		☐	☐	☐	☐	☐	☐	姓名
age	int	11		☐	☐	☐	☐	☐	☐	年龄
email	varchar	50		☐	☐	☐	☐	☐	☐	邮箱
create_time	datetime			☐	☐	☐	☐	☐	☐	创建时间
update_time	datetime			☐	☐	☐	☐	☐	☐	更新时间
version	int	10	1	☐	☐	☐	☐	☐	☐	版本，实现乐观锁
deleted	int	1	0	☐	☐	☐	☐	☐	☐	逻辑删除
				☐	☐	☐	☐	☐	☐	

	id	name	age	email	create_time	update_time	version	deleted
1	1	ISHOWLINGGGGG	18	123123123@123.com	<null>	2020-03-24 10:58:29	3	0
2	2	Jack	20	test2@baomidou.com	<null>	<null>	1	0
3	3	Tom	28	test3@baomidou.com	<null>	<null>	1	0
4	4	Sandy	21	test4@baomidou.com	<null>	<null>	1	0
5	5	Billie	24	test5@baomidou.com	<null>	<null>	1	0

2. 实体类更新

```
@Data
@AllArgsConstructor
@NoArgsConstructor
@Accessors(chain = true)
public class User {

    @TableId(type = IdType.AUTO)
    private Long id;
    private String name;
    private Integer age;
    private String email;
    @TableField(fill = FieldFill.INSERT)
    private Date createTime;
    @TableField(fill = FieldFill.INSERT_UPDATE)
    private Date updateTime;
    @Version //这是一个乐观锁的注解
    private Integer version;

    private Integer deleted;

}
```

3. 更新实体类，加注解

```
@Data
@AllArgsConstructor
@NoArgsConstructor
@Accessors(chain = true)
public class User {

    @TableId(type = IdType.AUTO)
    private Long id;
    private String name;
    private Integer age;
    private String email;
    @TableField(fill = FieldFill.INSERT)
    private Date createTime;
    @TableField(fill = FieldFill.INSERT_UPDATE)
    private Date updateTime;
    @Version
    private Integer version;
    @TableLogic //逻辑删除
    private Integer deleted;

}
```

4. 配置组件

```java
@MapperScan("com.bean.mapper") //扫描包
@EnableTransactionManagement//事物注解
@Configuration
public class MybatisPlusConfig {
    @Bean   //逻辑删除
    public ISqlInjector sqlInjector() {
        return new LogicSqlInjector();
    }
}
```

5. 配置中配置逻辑删除

```properties
# 配置逻辑删除# 配置删除了的为1# 配置没有删除的为0
mybatis-plus.global-config.db-config.logic-delete-value=1
mybatis-plus.global-config.db-config.logic-not-delete-value=0
```

6. 测试删除

```java
    @Test
    void testDelete() {
        //根据ID删除
        userMapper.deleteById(1L);
    }
```

	id	name	age	email	create_time	update_time	version	deleted
1	1	ISHOWLINGGGGG	18	123123123@123.com	<null>	2020-03-24 10:58:29	3	1
2	2	Jack	20	test2@baomidou.com	<null>	<null>	1	0
3	3	Tom	28	test3@baomidou.com	<null>	<null>	1	0
4	4	Sandy	21	test4@baomidou.com	<null>	<null>	1	0
5	5	Billie	24	test5@baomidou.com	<null>	<null>	1	0
6	12409417973000	BEAN	20	xxx@qq.com	2020-03-24 11:40:11	2020-03-24 11:40:11	1	0

7. 测试查询

```java
    @Test
    void testSelect() {

        //查询单个ID
        List<User> users = userMapper.selectList(null);
        users.forEach(System.out::println);

    }
```

```
 ==>  Preparing: SELECT id,name,age,email,create_time,update_time,version,deleted FROM user WHERE deleted=0
 ==> Parameters:
 <==    Columns: id, name, age, email, create_time, update_time, version, deleted
 <==        Row: 2, Jack, 20, test2@baomidou.com, null, null, 1, 0
 <==        Row: 3, Tom, 28, test3@baomidou.com, null, null, 1, 0
 <==        Row: 4, Sandy, 21, test4@baomidou.com, null, null, 1, 0
 <==        Row: 5, Billie, 24, test5@baomidou.com, null, null, 1, 0
 <==        Row: 12409417973000, BEAN, 20, xxx@qq.com, 2020-03-24 11:40:11, 2020-03-24 11:40:11, 1, 0
 <==      Total: 5
```

> 我们可以看到查询语句自动变为 `WHERE deleted=0`

以上的CRUD及其扩展操作都必须精通

性能分析插件

我们在平时的开发中，会遇到一些慢SQL，我们可以用其他的插件做，但是MybatisPlus也提供了

> 作用：性能分析拦截器，用于输出每条 SQL 语句及其执行时间

1. 导入插件

```
@MapperScan("com.bean.mapper") //扫描包
@EnableTransactionManagement//事物注解
@Configuration
public class MybatisPlusConfig {

    /**
     * SQL执行效率插件
     */
    @Bean
    @Profile({"dev","test"})// 设置 dev test 环境开启
    public PerformanceInterceptor performanceInterceptor() {

        PerformanceInterceptor performanceInterceptor = new PerformanceInterceptor();

        performanceInterceptor.setMaxTime(1);//设置sql执行的最大时间，如果超过了则不执行，这里设置为1毫秒

        performanceInterceptor.setFormat(true); //是否开启格式化支持

        return performanceInterceptor;
    }
}
```

   - 因为设置了只有开发和测试环境才可以开启，所以我们要更改一下配置，改为开发环境或者测试环境

```
# 设置为开发环境
spring.profiles.active=dev
```

2. 测试使用

> 首先我们的SQL不可能只是支持一毫秒，所以我们主要看的是报错

我们再改为100毫秒

```
performanceInterceptor.setMaxTime(100);
//设置sql执行的最大时间，如果超过了则不执行，这里设置为100毫秒`
```

```
Time: 33 ms - ID: com.bean.mapper.UserMapper.selectList
Execute SQL:
    SELECT
        id,
        name,
        age,
        email,
        create_time,
        update_time,
        version,
        deleted
    FROM
        user
    WHERE
        deleted=0
```

这次的时间又成了33毫秒，在100毫秒范围内

> 使用 性能分析插件可以显著提高开发效率

条件构造器

Wrapper更多请查看官网，这里说一些常用的

条件查询，非空和大小判断

```
@Test
void contextLoads() {

    //name不为空，邮箱不为空，年龄大于12：
    QueryWrapper<User> wrapper = new QueryWrapper<>();

    wrapper.isNotNull("name")//名字不为空
            .isNotNull("email")//邮箱不为空
            .ge("age",12)//大于等于：greater than，equal to，大于等于12
    ;

    userMapper.selectList(wrapper).forEach(System.out::println);
```

```
13
14      }
```

条件查询，相等语句查询

```
1       @Test
2       void test2() {
3
4           //name为BEAN：
5           QueryWrapper<User> wrapper = new QueryWrapper<>();
6
7           wrapper.eq("name","BEAN");  //equal：等于
8
9           //selectOne:查询一个数据
10          System.out.println(userMapper.selectOne(wrapper));
11
12      }
```

范围查询

```
1       @Test
2       void test3() {
3
4           //查询年龄在20~30之间的用户有多少
5           QueryWrapper<User> wrapper = new QueryWrapper<>();
6
7           wrapper.between("name",20,30);  //between：在xx~xx之间
8
9           //selectCount:查询一个总数据
10          System.out.println(userMapper.selectCount(wrapper));
11
12      }
```

模糊查询

```
1       @Test
2       void test4() {
3
4           //模糊查询，名字里不包含e，包含k的
5           QueryWrapper<User> wrapper = new QueryWrapper<>();
6
7           wrapper.notLike("name","e").like("name","k");
8
9
10          userMapper.selectMaps(wrapper).forEach(System.out::println);
11
12      }
```

左右查询

```
    @Test
    void test5() {

        //模糊查询，左查询：%xxx，右查询：xxx%
        //查询邮箱以t开头的，就是t%，也就是likeRight
        QueryWrapper<User> wrapper = new QueryWrapper<>();

        wrapper.likeRight("email","t");


        userMapper.selectMaps(wrapper).forEach(System.out::println);

    }
```

Sql嵌套查询

```
    @Test
    void test6() {

        //内查询，sql嵌套sql查询
        QueryWrapper<User> wrapper = new QueryWrapper<>();

        wrapper.inSql("id","select id from user where id< 3");


        userMapper.selectObjs(wrapper).forEach(System.out::println);

    }
```

排序

```
    @Test
    void test7() {

        //通过id进行排序
        QueryWrapper<User> wrapper = new QueryWrapper<>();

        //通过id进行降序排序
        wrapper.orderByDesc("id");

        userMapper.selectList(wrapper).forEach(System.out::println);

    }
```

代码自动生成器

dao、pojo、service、controller等不写了，让程序自动生成

AutoGenerator 是 MyBatis-Plus 的代码生成器，通过 AutoGenerator 可以快速生成 Entity、Mapper、Mapper XML、Service、Controller 等各个模块的代码，极大的提升了开发效率。

构建一个代码生成器对象（这个直接抄就行，有时间可以自己写）

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

        AutoGenerator autoGenerator = new AutoGenerator();

        //1. 全局配置：import com.baomidou.mybatisplus.generator.config.GlobalConfig;
        GlobalConfig config = new GlobalConfig();

        String property = System.getProperty("user.dir");//当前项目路径

        config.setOutputDir(property+"/src/main/java");//输出路径：生成的代码到当前项目路径/src/main/java下

        config.setAuthor("Bean");//作者

        config.setOpen(false);//是否打开资源管理器

        config.setFileOverride(false);//是否覆盖原来的代码

        config.setServiceName("%sService");//去掉Service的I前缀

        config.setIdType(IdType.ID_WORKER);//默认的ID生成策略

        config.setDateType(DateType.ONLY_DATE);//日期的类型

        config.setSwagger2(true);//设置Swagger


        autoGenerator.setGlobalConfig(config);//设置好全局配置

        //2. 设置数据源配置
        DataSourceConfig dataSource = new DataSourceConfig();

        dataSource.setUrl("jdbc:mysql://localhost:3306/mybatis_plus?useSSL=false&useUnicode=true&characterEncoding=utf-8&serverTimezone=Asia/Shanghai");
        dataSource.setDriverName("com.mysql.cj.jdbc.Driver");
        dataSource.setUsername("root");
        dataSource.setPassword("root");
        dataSource.setDbType(DbType.MYSQL);//设置数据库的类型
        autoGenerator.setDataSource(dataSource);    //设置好数据源配置

        //3. 包的配置
        PackageConfig packageConfig = new PackageConfig();

        packageConfig.setModuleName("blog");//模块名字
        packageConfig.setParent("com.bean");//放到哪个包下,那么就是com.bean.blog
        packageConfig.setEntity("pojo");//实体类的名字
        packageConfig.setMapper("mapper");//映射的名字
        packageConfig.setService("service");//Service名字
        packageConfig.setController("controller");//Controller的名字
        autoGenerator.setPackageInfo(packageConfig);//设置包的配置

```

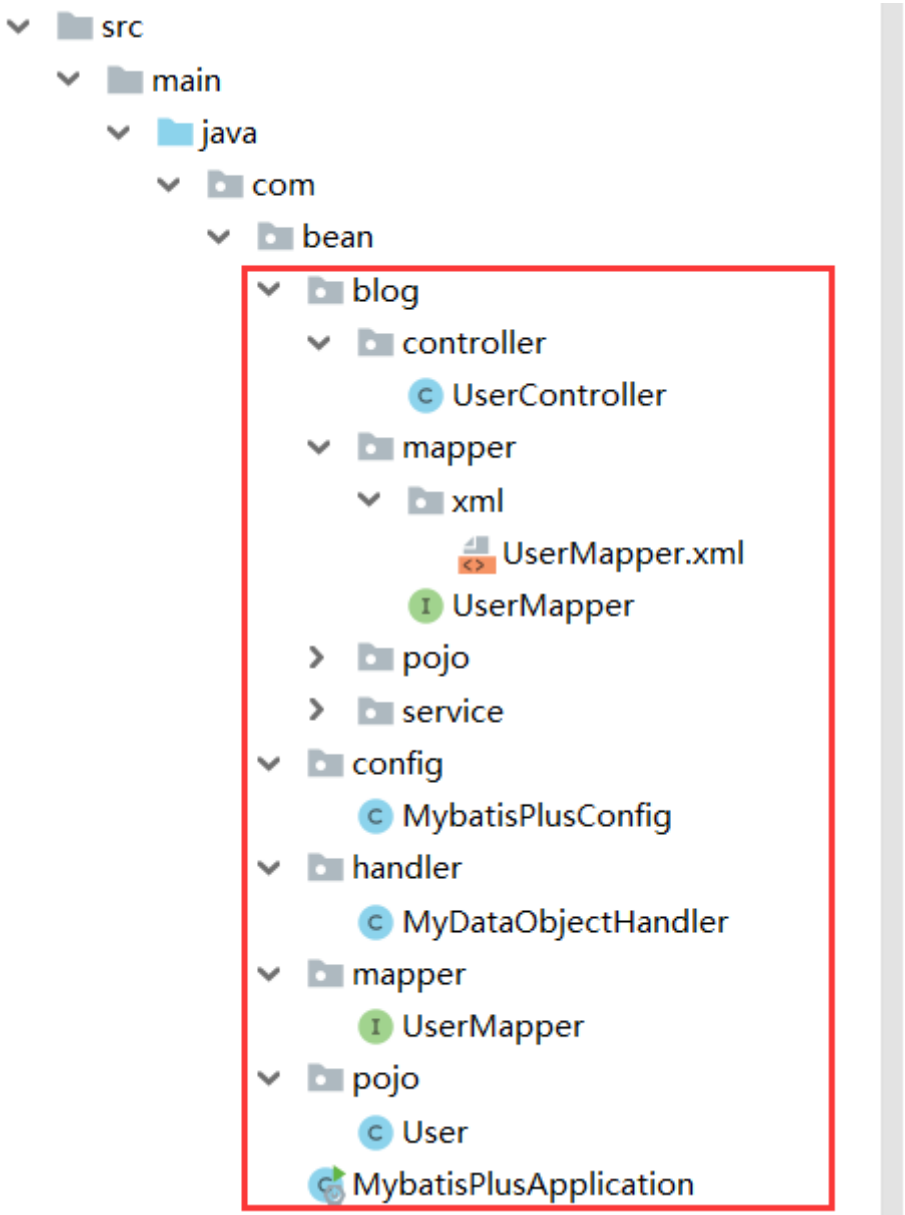
```
50
51          //4. 策略配置
52          StrategyConfig strategy = new StrategyConfig();
53          strategy.setInclude("user");//设置要映射的表名，这里是一个可变参数，可以设置多张表
54          strategy.setNaming(NamingStrategy.underline_to_camel);//下划线转驼峰
55          strategy.setColumnNaming(NamingStrategy.underline_to_camel);//下划线转驼峰
56          strategy.setEntityLombokModel(true);//是否使用lombok开启注解
57
58          strategy.setLogicDeleteFieldName("deleted");//逻辑删除字段
59
60          TableFill createTime = new TableFill("create_time", FieldFill.INSERT);//自动填充，创建
    时间
61          TableFill updateTime = new TableFill("update_time", FieldFill.UPDATE);//自动填充，修改
    时间
62
63          ArrayList<TableFill> tableFills = new ArrayList<>();
64          tableFills.add(createTime);
65          tableFills.add(updateTime);
66          strategy.setTableFillList(tableFills);  //添加到自动填充
67
68          strategy.setVersionFieldName("version");//乐观锁
69
70          strategy.setRestControllerStyle(true);//开启Controller的Rest的驼峰命名格式
71
72          strategy.setControllerMappingHyphenStyle(true);//开启Controller的下划线形式：
    localhost:8080/hello_id_2
73
74          autoGenerator.setStrategy(strategy);    //策略配置
75
76          autoGenerator.execute();//执行
77
78      }
```

报错：

1. `java.lang.NoClassDefFoundError: org/apache/velocity/context/Context`

原因是缺少依赖

```
1       <!-- https://mvnrepository.com/artifact/org.apache.velocity/velocity-engine-core -
    ->
2       <dependency>
3           <groupId>org.apache.velocity</groupId>
4           <artifactId>velocity-engine-core</artifactId>
5           <version>2.2</version>
6       </dependency>
```



这里一切的一切都生成好了，而只需要改变一个地方：

```
1    strategy.setInclude("user");//设置要映射的表名，这里是一个可变参数，可以设置多张表
```