MyBatisPlus(高级)

作者:故事我忘了¢
个人微信公众号:程序猿的月光宝盒

说明:

本文是继上一篇入门的进阶版

相关连接:

慕课入门视频:

https://www.imooc.com/learn/1130

入门文章:

https://www.cnblogs.com/jsccc520/p/14669347.html

本文对应进阶视频:

https://www.imooc.com/learn/1171

整合的github地址:

https://github.com/monkeyKinn/StudyMyBatisPlus

觉得不错给个star呗~

star

star

star

Start~

数据准备

```
#创建用户表
CREATE TABLE user_high (
    id BIGINT(20) PRIMARY KEY NOT NULL COMMENT '主键',
    name VARCHAR(30) DEFAULT NULL COMMENT '姓名',
    age INT(11) DEFAULT NULL COMMENT '年龄',
    email VARCHAR(50) DEFAULT NULL COMMENT '邮箱',
```

```
    manager_id BIGINT(20) DEFAULT NULL COMMENT '直属上级id',
    create_time DATETIME DEFAULT NULL COMMENT '创建时间',
    update_time DATETIME DEFAULT NULL COMMENT '修改时间',
    version INT(11) DEFAULT '1' COMMENT '版本',
    deleted INT(1) DEFAULT '0' COMMENT '逻辑删除标识(0.未删除,1.已删除)',
    CONSTRAINT manager_fk FOREIGN KEY (manager_id)
        REFERENCES user_high (id)
)  ENGINE=INNODB CHARSET=UTF8;

#初始化数据:
INSERT INTO user_high (id, name, age, email, manager_id
    , create_time)
VALUES (1087982257332887553, '大boss', 40, 'boss@baomidou.com', NULL
        , '2019-01-11 14:20:20'),
    (1088248166370832385, '王天风', 25, 'wtf@baomidou.com', 1087982257332887553
        , '2019-02-05 11:12:22'),
    (1088250446457389058, '李艺伟', 28, 'lyw@baomidou.com', 1088248166370832385
        , '2019-02-14 08:31:16'),
    (1094590409767661570, '张雨琪', 31, 'zjq@baomidou.com', 1088248166370832385
        , '2019-01-14 09:15:15'),
    (1094592041087729666, '刘红雨', 32, 'lhm@baomidou.com', 1088248166370832385
        , '2019-01-14 09:48:16');
```

注意在user实体类中加上注解映射表名

```
@TableName("user_high")
```

因为项目直接是分支下来的 配置就不配置了

具体的到我github上看吧 有需要的话

逻辑删除

简介

加个字段,表示删除状态----软删除了,有选项是级联删除....这就约等于删库跑路,,老板疯狂追你三条gai

MP实现

首先配置文件中给配置上

```
spring.application.name=MyBatisPlus
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.url=jdbc:mysql://localhost:3306/mp?
useSSL=false&serverTimezone=GMT%2B8
spring.datasource.username=root
spring.datasource.password=admin


#日志输出配置
logging.level.root=warn
#只想看这个包里的sql日志 trace是最低级别
logging.level.com.jsc.mybatisplus.mapper=trace
# p:级别,m:内容,n:换行
logging.pattern.console=%p%m%n
```

```
#配置全局逻辑删除
#没删除 标志0
mybatis-plus.global-config.db-config.logic-not-delete-value=0
#删除 标志1
mybatis-plus.global-config.db-config.logic-delete-value=1
```

配置类

```
/**
 * mybatis的配置类
 *
 * @author 金聖聰
 * @version v1.0
 * @email jinshengcong@163.com
 * @date Created in 2021/04/16 15:48
 */
@Configuration
public class MyBatisPlusConfig {

    // MP 3.1.1开始    不需要配置
    // @Bean
    // public ISqlInjector sqlInjector() {
    //     return new LogicSqlInjector();
    // }
}
```

在实体类中加上逻辑删除的注解`@TableLogic`

```
    @TableLogic
    private Integer deleted;
```

创建测试类

```
    @Autowired
    UserMapper userMapper;

    @Test
    void delByIdLogic() {
        // 加了前面的配置后这里就是逻辑删除了
        int i = userMapper.deleteById(1087982257332887553L);
        System.out.println("影响行数: " + i);
    }
```

当逻辑删除后,以后进行的`查询`,`更新`等操作都会去除已经逻辑删除的数据

```
    @Test
    void selectAll() {
        // 查询全部,这时候把deleted为1的过滤了
        userMapper.selectList(null).forEach(System.out::println);
    }

    @Test
    void updateById() {
```

```
        User user = new User();
        // 更新我逻辑删掉的数据,英雄行数为0 更新失败
        // user.setId(1087982257332887553L);
        // 更新没删除的别人,就可以
        user.setId(1094592041087729666L);
        user.setAge(18);
        int i = userMapper.updateById(user);
        System.out.println("影响行数: " + i);
    }
```

查询中排除逻辑删除字段以及注意事项

在实体类的对应字段上加上注解`@TableField(select=false)`

```
/** 逻辑删除标识(0.未删除,1.已删除) */
    @TableLogic
    @TableField(select=false)
    private Integer deleted;
```

下次再查询的时候这个字段就不会出现

但是在自定义的sql中,逻辑删除的还是会查出来,得看你自己的sql了

```
public interface UserMapper extends BaseMapper<User> {
    @Select("select * from user_high ${ew.customSqlSegment}")
    List<User> mySelectList(@Param(Constants.WRAPPER) Wrapper<User> wrapper);
}
```

```
    @Test
    void selectMyOwn() {
        // 查询全部,这时候把deleted为1的过滤了
        userMapper.mySelectList(
                Wrappers.<User>lambdaQuery()
                        .gt(User::getAge,18)
        )
                .forEach(System.out::println);
    }
```

```
User(id=1087982257332887553, name=大boss, age=40, email=boss@baomidou.com,
managerId=null, createTime=2019-01-11T14:20:20, updateTime=null, version=1,
deleted=1)
User(id=1088248166370832385, name=王天风, age=25, email=wtf@baomidou.com,
managerId=1087982257332887553, createTime=2019-02-05T11:12:22, updateTime=null,
version=1, deleted=0)
User(id=1088250446457389058, name=李艺伟, age=28, email=lyw@baomidou.com,
managerId=1088248166370832385, createTime=2019-02-14T08:31:16, updateTime=null,
version=1, deleted=0)
User(id=1094590409767661570, name=张雨琪, age=31, email=zjq@baomidou.com,
managerId=1088248166370832385, createTime=2019-01-14T09:15:15, updateTime=null,
version=1, deleted=0)
```

但是你想过滤的话

```
    @Test
    void selectMyOwn() {
        // 查询全部,这时候把deleted为1的过滤了
        userMapper.mySelectList(
                Wrappers.<User>lambdaQuery()
                        .gt(User::getAge,18)
                        // 加上就过滤了
                        .eq(User::getDeleted,0)
        )
                .forEach(System.out::println);
    }
```

自动填充

简介

有的项目有新增时间啊 修改时间啊 修改人啊啥的,每次都重复插入有点麻烦,虽然有些也是一个函数的事,,哎,~~~~但我就是玩儿~(不是,我就是懒

而且要是我就是要记录 是谁 在什么时间点动了数据库,这就玩(懒)不起来了..

还好,MP够懂Coder,你懒归你懒,清风拂山岗

实现

1.修改实体类

```
/** 创建时间 ,在插入时候填充*/
@TableField(fill = FieldFill.INSERT)
private LocalDateTime createTime;
/** 修改时间 ,在更新时候填充*/
@TableField(fill = FieldFill.UPDATE)
private LocalDateTime updateTime;
```

2.新建处理器类

```
/**
 * 我的元数据处理器
 *
 * @author 金聖聰
 * @version v1.0
 * @email jinshengcong@163.com
 * @date Created in 2021/04/17 17:00
 */
@Component
public class MyMetaObjectHandler implements MetaObjectHandler {

    /**
     * 插入的时候的填充方法
     *
     * @param metaObject 元数据
     * @return void 空
     * @author 金聖聰
     * @email jinshengcong@163.com
```

```
     * Modification History:
     * Date         Author        Description        version
     *-----------------------------------------------------------*
     * 2021/04/17 17:02     金聖聰      修改原因            1.0
     */
    @Override
    public void insertFill(MetaObject metaObject) {
        // 3.3.0过期了 替代方法是 strictInsertFill
        // setInsertFieldValByName("createTime",
LocalDateTime.now(),metaObject);
        // 等效于
        // setFieldValByName("createTime", LocalDateTime.now(),metaObject);

 strictInsertFill(metaObject,"createTime",LocalDateTime.class,LocalDateTime.now(
));
    }

    /**
     * 更新时候的填充方法
     * @param metaObject 元数据
     * @return void 空
     * @author 金聖聰
     * @email jinshengcong@163.com
     * Modification History:
     * Date         Author        Description        version
     *-----------------------------------------------------------*
     * 2021/04/17 17:02     金聖聰      修改原因            1.0
     */
    @Override
    public void updateFill(MetaObject metaObject) {
        // 3.3.0过期了 替代方法是 strictUpdateFill
        // setUpdateFieldValByName("updateTime",LocalDateTime.now(),metaObject);
        // 等效于
        // setFieldValByName("updateTime",LocalDateTime.now(),metaObject);
        strictUpdateFill(metaObject, "updateTime",
LocalDateTime.class,LocalDateTime.now());

    }
}
```

3.测试类

```
    @Test
    void insertAutoFilled() {
        User user = new User();
        user.setName("小李");
        user.setAge(18);
        // 自动填充了创建时间
        int insert = userMapper.insert(user);
        System.out.println("影响行数: " + insert);
    }

    @Test
    void updateAutoFilled() {
        User user = new User();
        user.setId(1383349447410843650L);
        user.setName("夹心");
```

```
        user.setAge(16);
        // 自动填充了更新时间
        int insert = userMapper.updateById(user);
        System.out.println("影响行数: " + insert);
    }
```

自动填充优化

在处理器类中进行优化

```
    @Override
    public void insertFill(MetaObject metaObject) {
        // 3.3.0过期了 替代方法是 strictInsertFill
        // setInsertFieldValByName("createTime",
LocalDateTime.now(),metaObject);
        // 等效于
        // setFieldValByName("createTime", LocalDateTime.now(),metaObject);

        // 优化: 有没有这个属性,有的话才自动填充
        boolean createTime = metaObject.hasSetter("createTime");
        if (createTime) {
            strictInsertFill(metaObject, "createTime", LocalDateTime.class,
LocalDateTime.now());
        }
    }
```

当我更新的时候有设置好值,你就别帮我跟新,这么怎么设置呢

```
    @Override
    public void updateFill(MetaObject metaObject) {
        // 3.3.0过期了 替代方法是 strictUpdateFill
        // setUpdateFieldValByName("updateTime",LocalDateTime.now(),metaObject);
        // 等效于
        // setFieldValByName("updateTime",LocalDateTime.now(),metaObject);

        // 优化:  只有这个为null的时候才进行自动填充
        Object updateTime = getFieldValByName("updateTime", metaObject);
        if (ObjectUtils.isEmpty(updateTime)) {
            strictUpdateFill(metaObject, "updateTime", LocalDateTime.class,
LocalDateTime.now());
        }

    }
```

测试类

```
    @Test
    void updateById() {
        User user = new User();
        // 更新我逻辑删掉的数据,英雄行数为0 更新失败
        // user.setId(1087982257332887553L);
        // 更新没删除的别人,就可以
        user.setId(1383353200797118465L);
        user.setAge(10);
        // user.setUpdateTime(LocalDateTime.now().plusDays(1));
        int i = userMapper.updateById(user);
        System.out.println("影响行数: " + i);
    }
```

乐观锁插件

简介

意图: 当要更新一条记录的时候,希望这条数据没有被别人更新过,是为了防止更新冲突的问题

应用场景:

一般是悲观锁和乐观锁

悲观锁是通过数据库的锁机制实现 ----多写,少读

乐观锁是通过表的版本号实现 ---写比较少的场景,多读的场景

实现原理

1.版本号

a) 取出记录时,获取当前的version

b) 更新这条记录时,带上这个version

c) 版本正确,更新成功,版本错误,更新失败

举个

```
Update table set version = newVersionNo ,x=a,y=b where version = oldVersionNo
and z=c;


#示例
Update table set version = 3 ,name='小欣驾驶' where version = 1 and
id=1383352467997671425L;
```

功能实现

实现步骤

1.配置乐观锁插件

在配置类中配置

```
package com.jsc.mybatisplus.config;

import com.baomidou.mybatisplus.annotation.DbType;
import com.baomidou.mybatisplus.autoconfigure.ConfigurationCustomizer;
import com.baomidou.mybatisplus.extension.plugins.MybatisPlusInterceptor;
import
com.baomidou.mybatisplus.extension.plugins.inner.OptimisticLockerInnerIntercepto
r;
import
com.baomidou.mybatisplus.extension.plugins.inner.PaginationInnerInterceptor;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

/**
 * mybatis\Plus 的配置类
 *
 * @author 金聖聰
 * @version v1.0
 * @email jinshengcong@163.com
 * @date Created in 2021/04/16 15:48
 */
@Configuration
public class MyBatisPlusConfig {

    /**
     * 注册插件
     * 依赖以下版本+
     * <dependency>
     * <groupId>com.baomidou</groupId>
     * <artifactId>mybatis-plus-boot-starter</artifactId>
     * <version>3.4.0</version>
     * </dependency>
     *
     * @return com.baomidou.mybatisplus.extension.plugins.MybatisPlusInterceptor
拦截器
     * @author 金聖聰
     * @email jinshengcong@163.com
     * Modification History:
     * Date         Author        Description        version
     * ------------------------------------------------------------*
     * 2021/04/16 15:56     金聖聰      修改原因             1.0
     */
    @Bean
    public MybatisPlusInterceptor mybatisPlusInterceptor() {
        // 0.创建一个拦截器
        MybatisPlusInterceptor interceptor = new MybatisPlusInterceptor();

        // 1. 添加分页插件
```

```
        PaginationInnerInterceptor pageInterceptor = new
PaginationInnerInterceptor();
        // 2. 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
        pageInterceptor.setOverflow(false);
        // 3. 单页分页条数限制，默认无限制
        pageInterceptor.setMaxLimit(500L);
        // 4. 设置数据库类型
        pageInterceptor.setDbType(DbType.MYSQL);
        // 5.添加内部拦截器
        interceptor.addInnerInterceptor(pageInterceptor);

        // 6.乐观锁插件 添加乐观锁插件
        interceptor.addInnerInterceptor(new OptimisticLockerInnerInterceptor());
        return interceptor;
    }

    @Bean
    public ConfigurationCustomizer configurationCustomizer() {
        // 需要设置 MybatisConfiguration#useDeprecatedExecutor = false 避免缓存出现
问题(该属性会在旧插件移除后一同移除)
        return configuration -> configuration.setUseDeprecatedExecutor(false);
    }

}
```

2.在实体类中找到变量version属性,加上注解

```
    /** 版本 */
    @Version
    private Integer version;
```

3.测试类

```
    @Test
    void updateUseLock() {
        // 假设是从db中取出来了
        int version = 3;
        User user = new User();
        user.setId(1383352467997671425L);
        user.setName("饼干");
        user.setAge(16);
        // 自动更新成2
        user.setVersion(version);
        // 自动填充了更新时间
        int insert = userMapper.updateById(user);
        System.out.println("影响行数: " + insert);
    }
```

注意事项

说明:

- **支持的数据类型只有:int,Integer,long,Long,Date,Timestamp,LocalDateTime**
- 整数类型下 `newVersion = oldVersion + 1`
- `newVersion` 会回写到 `entity` 中
- 仅支持 `updateById(id)` 与 `update(entity, wrapper)` 方法
- **在 `update(entity, wrapper)` 方法下, `wrapper` 不能复用!!!**

```
    @Test
    void updateLockWrapperTwice() {
        int version = 5;

        // 乐观锁,wrapper复用
        User user = new User();
        user.setName("❤❤❤❤");
        user.setAge(18);
        user.setVersion(version);

        QueryWrapper<User> query = Wrappers.query();
        query.eq("name","❤❤");
        // 自动填充了创建时间
        int insert = userMapper.update(user,query);
        System.out.println("影响行数: " + insert);

        // 复用query
        User user2 = new User();
        user2.setName("❤❤❤");
        user2.setAge(18);
        user2.setVersion(6);
        query.eq("name","❤❤❤❤");
        // 自动填充了创建时间
        System.out.println(userMapper.update(user, query));


    }
```

性能分析插件

用途:

显示每条sql以及,执行时间

执行sql分析打印

建议官网

https://mybatis.plus/guide/p6spy.html

引入依赖

```
<!-- https://mvnrepository.com/artifact/p6spy/p6spy -->
<dependency>
    <groupId>p6spy</groupId>
    <artifactId>p6spy</artifactId>
    <version>3.9.1</version>
</dependency>
```

更改配置文件

```
#spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
#执行sql分析打印
spring.datasource.driver-class-name=com.p6spy.engine.spy.P6SpyDriver

#spring.datasource.url=jdbc:mysql://localhost:3306/mp?
useSSL=false&serverTimezone=GMT%2B8
#执行sql分析打印
spring.datasource.url=jdbc:p6spy:mysql://localhost:3306/mp?
useSSL=false&serverTimezone=GMT%2B8
```

增加配置文件

在resources文件夹中增加

`spy.properties`

```
#3.2.1以上使用
modulelist=com.baomidou.mybatisplus.extension.p6spy.MybatisPlusLogFactory,com.p6
spy.engine.outage.P6OutageFactory
#3.2.1以下使用或者不配置
#modulelist=com.p6spy.engine.logging.P6LogFactory,com.p6spy.engine.outage.P6Outa
geFactory
# 自定义日志打印
logMessageFormat=com.baomidou.mybatisplus.extension.p6spy.P6SpyLogger
#日志输出到控制台
appender=com.baomidou.mybatisplus.extension.p6spy.StdoutLogger
# 使用日志系统记录 sql
#appender=com.p6spy.engine.spy.appender.Slf4JLogger
# 设置 p6spy driver 代理
deregisterdrivers=true
# 取消JDBC URL前缀
useprefix=true
# 配置记录 Log 例外,可去掉的结果集有
error,info,batch,debug,statement,commit,rollback,result,resultset.
excludecategories=info,debug,result,commit,resultset
# 日期格式
dateformat=yyyy-MM-dd HH:mm:ss
# 实际驱动可多个
#driverlist=org.h2.Driver
# 是否开启慢SQL记录
outagedetection=true
# 慢SQL记录标准 2 秒
outagedetectioninterval=2
```

还可以配置文件形式

```
logflie=xxx.log
```

如果想设置到别的地方百度一下

`p6spy使用`

有性能损耗,不建议在生产环境中使用

多租户sql解析器(没人用(/"◑ ◑)/"(。•́‿•̀。),跳过.....)

概念:

是软件架构技术,是实现如何在多用户(一般是指面向企业的用户,共用相同的系统或者程序)环境下,实现数据的隔离

就是用户使用同一套程序,然后用户间数据隔离

隔离方案:

1.独立数据库,一个租户一个数据库,隔离级别最高

优点:为不同的用户提供独立的数据库有助于数据模型的扩展设计满足不同用户的个人需求,出现故障,恢复数据也很简单

缺点:增加了数据库的安装数量,带来了维护成本和购置成本

2.共享数据库,独立schema

优点:为安全性要求较高的用户提供一定程度的逻辑数据隔离,并不是完全隔离,每个数据库可以支持更多的租户数量

缺点:如果出现了故障,数据恢复困难,因为恢复数据会涉及到其他租户的数据

3.共享数据库和共享schema,共享数据表,但是在表中增加多租户的`租户id`这个字段,共享程度最高,隔离级别最低,简单来说,每插入一个数据都要有一个租户的标识,这样才能在同一张表中区分不同的租户数据

优点:维护和购置成本最低,每个数据库支持的用户数量最多

缺点:隔离级别,安全性最低,需要在开发时候增加对安全的开发量,数据恢复和备份最困难

多租户实现

依赖分页插件,需要在分页插件中设置解析器,用第三种方式实现

分页插件已经配置过了

```
package com.jsc.mybatisplus.config;

import com.baomidou.mybatisplus.annotation.DbType;
import com.baomidou.mybatisplus.autoconfigure.ConfigurationCustomizer;
import com.baomidou.mybatisplus.extension.plugins.MybatisPlusInterceptor;
import com.baomidou.mybatisplus.extension.plugins.handler.TenantLineHandler;
```

```
import
com.baomidou.mybatisplus.extension.plugins.inner.OptimisticLockerInnerIntercepto
r;
import
com.baomidou.mybatisplus.extension.plugins.inner.PaginationInnerInterceptor;
import
com.baomidou.mybatisplus.extension.plugins.inner.TenantLineInnerInterceptor;
import net.sf.jsqlparser.expression.Expression;
import net.sf.jsqlparser.expression.LongValue;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

/**
 * mybatis\Plus 的配置类
 *
 * @author 金聖聰
 * @version v1.0
 * @email jinshengcong@163.com
 * @date Created in 2021/04/16 15:48
 */
@Configuration
public class MyBatisPlusConfig {

    /**
     * 注册插件
     * 依赖以下版本+
     * <dependency>
     * <groupId>com.baomidou</groupId>
     * <artifactId>mybatis-plus-boot-starter</artifactId>
     * <version>3.4.1</version>
     * </dependency>
     *
     * @return com.baomidou.mybatisplus.extension.plugins.MybatisPlusInterceptor
拦截器
     * @author 金聖聰
     * @email jinshengcong@163.com
     * Modification History:
     * Date         Author        Description        version
     * ------------------------------------------------------------*
     * 2021/04/16 15:56     金聖聰      修改原因            1.0
     */
    @Bean
    public MybatisPlusInterceptor mybatisPlusInterceptor() {

        // 0.创建一个拦截器
        MybatisPlusInterceptor interceptor = new MybatisPlusInterceptor();

        // 1. 添加分页插件
        PaginationInnerInterceptor pageInterceptor = new
PaginationInnerInterceptor();
        // 2. 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
        pageInterceptor.setOverflow(false);
        // 3. 单页分页条数限制，默认无限制
        pageInterceptor.setMaxLimit(500L);
        // 4. 设置数据库类型
        pageInterceptor.setDbType(DbType.MYSQL);
        interceptor.addInnerInterceptor(new TenantLineInnerInterceptor(new
TenantLineHandler(){
```

```
            @Override
            public Expression getTenantId() {
                // 租户信息 一般是session或者配置文件中,或者静态变量中
                // 1088248166370832385  王天风的
                return new LongValue(1088248166370832385L);
            }

            @Override
            public String getTenantIdColumn() {
                // 多用户字段是什么,表中的字段名字
                return "manager_id";
            }

            // 这是 default 方法,默认返回 false 表示所有表都需要拼多租户条件
            // 某些表不加租户信息,
            @Override
            public boolean ignoreTable(String tableName) {
                // 如果查是user_high ->false不忽略(除了这个都忽略)
                // !"user_high"就是不过滤,"user_high"就是过滤
                return "user_high".equalsIgnoreCase(tableName);
            }
        }));
        // 5.添加内部拦截器
        interceptor.addInnerInterceptor(pageInterceptor);
        // 6.乐观锁插件 添加乐观锁插件
        interceptor.addInnerInterceptor(new OptimisticLockerInnerInterceptor());


        // 如果用了分页插件注意先 add TenantLineInnerInterceptor 再 add
PaginationInnerInterceptor
        // 用了分页插件必须设置 MybatisConfiguration#useDeprecatedExecutor = false
//        interceptor.addInnerInterceptor(new PaginationInnerInterceptor());
        return interceptor;
    }

    @Bean
    public ConfigurationCustomizer configurationCustomizer() {
        // 需要设置 MybatisConfiguration#useDeprecatedExecutor = false 避免缓存出现
问题(该属性会在旧插件移除后一同移除)
        return configuration -> configuration.setUseDeprecatedExecutor(false);
    }
}
```

这样在查询的时候就会调用租户信息 `manager_id = 1088248166370832385`

```
查询
DEBUG==>  Preparing: SELECT id, name, age, email, manager_id, create_time,
update_time, version FROM user_high WHERE deleted = 0 AND manager_id =
1088248166370832385
DEBUG==> Parameters:
 Consume Time：13 ms 2021-04-17 21:33:17
 Execute SQL：SELECT id, name, age, email, manager_id, create_time, update_time,
version FROM user_high WHERE deleted = 0 AND manager_id = 1088248166370832385
```

```
 更新
 DEBUG==>  Preparing: UPDATE user_high SET age = ?, update_time = ? WHERE
manager_id = 1088248166370832385 AND id = ? AND deleted = 0
DEBUG==> Parameters: 20(Integer), 2021-04-17T21:35:44.232(LocalDateTime),
1383352467997671425(Long)
 Consume Time：1 ms 2021-04-17 21:35:44
 Execute SQL：UPDATE user_high SET age = 20, update_time = '2021-04-
17T21:35:44.232' WHERE manager_id = 1088248166370832385 AND id =
1383352467997671425 AND deleted = 0

 新增
 DEBUG==>  Preparing: INSERT INTO user_high (id, name, age, create_time,
manager_id) VALUES (?, ?, ?, ?, 1088248166370832385)
DEBUG==> Parameters: 1383414202070806530(Long), test(String), 18(Integer), null
 Consume Time：9 ms 2021-04-17 21:37:03
 Execute SQL：INSERT INTO user_high (id, name, age, create_time, manager_id)
VALUES (1383414202070806530, 'test', 18, NULL, 1088248166370832385)

 删除
 DEBUG==>  Preparing: UPDATE user_high SET deleted = 1 WHERE manager_id =
1088248166370832385 AND id = ? AND deleted = 0
DEBUG==> Parameters: 1383415177636483073(Long)
 Consume Time：8 ms 2021-04-17 21:41:38
 Execute SQL：UPDATE user_high SET deleted = 1 WHERE manager_id =
1088248166370832385 AND id = 1383415177636483073 AND deleted = 0
```

特定sql过滤

就是过滤特定的方法

就比如说查询方法,selectbyid,我不想增加这个租户信息,而其他方法想增加租户信息,这种就是方法级的过滤

这里.....emm

新版都没了....注解已经废弃...

再见.....握个手吧

动态表名sql解析器

应用场景

有的项目有多个表...这多个表存的是同类型的数据,字段都是一样的,分表存储吧,大概是,比如日志表 xxxlog_年月日

还有针对不同机构的,一张表一个机构

规则就是前缀相同,后缀不同,在调用的时候才能确定哪张表,要动态的进行拼接表名,这时候就上场吧...

分库分表插件可以解决吧,,但是数据量得多大才需要分库分表?

以为天国面试吗,怎么优化,,,上来分库分表?

没有任何意义.....

但是感觉比上个多租户用的多.....

学一下

动态表名sql实现

也是在分页插件中

```
package com.jsc.mybatisplus.config;

import com.baomidou.mybatisplus.annotation.DbType;
import com.baomidou.mybatisplus.autoconfigure.ConfigurationCustomizer;
import com.baomidou.mybatisplus.core.parser.ISqlParser;
import com.baomidou.mybatisplus.core.parser.SqlInfo;
import com.baomidou.mybatisplus.extension.plugins.MybatisPlusInterceptor;
import com.baomidou.mybatisplus.extension.plugins.handler.TableNameHandler;
import com.baomidou.mybatisplus.extension.plugins.handler.TenantLineHandler;
import
com.baomidou.mybatisplus.extension.plugins.inner.DynamicTableNameInnerIntercepto
r;
import
com.baomidou.mybatisplus.extension.plugins.inner.OptimisticLockerInnerIntercepto
r;
import
com.baomidou.mybatisplus.extension.plugins.inner.PaginationInnerInterceptor;
import
com.baomidou.mybatisplus.extension.plugins.inner.TenantLineInnerInterceptor;
import net.sf.jsqlparser.expression.Expression;
import net.sf.jsqlparser.expression.LongValue;
import org.apache.ibatis.reflection.MetaObject;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

import java.util.HashMap;
import java.util.Random;

/**
 * mybatis\Plus 的配置类
 *
 * @author 金聖聰
 * @version v1.0
 * @email jinshengcong@163.com
 * @date Created in 2021/04/16 15:48
 */
@Configuration
public class MyBatisPlusConfig {

    /**
     * 注册插件
     * 依赖以下版本+
     * <dependency>
     * <groupId>com.baomidou</groupId>
     * <artifactId>mybatis-plus-boot-starter</artifactId>
     * <version>3.4.1</version>
     * </dependency>
     *
```

```
     * @return com.baomidou.mybatisplus.extension.plugins.MybatisPlusInterceptor
拦截器
     * @author 金聖聰
     * @email jinshengcong@163.com
     * Modification History:
     * Date         Author        Description        version
     * ------------------------------------------------------------*
     * 2021/04/16 15:56     金聖聰      修改原因            1.0
     */
    @Bean
    public MybatisPlusInterceptor mybatisPlusInterceptor() {

        // 0.创建一个拦截器
        MybatisPlusInterceptor interceptor = new MybatisPlusInterceptor();

        // 动态表名
        DynamicTableNameInnerInterceptor dynamicTableNameInnerInterceptor = new
DynamicTableNameInnerInterceptor();

        HashMap<String, TableNameHandler> map = new HashMap<String,
TableNameHandler>(2) {{
            put("user_high", (sql, tableName) -> {
                // 动态表名的主要换表逻辑在这
                String year = "_2018";
                int random = new Random().nextInt(10);
                if (random % 2 == 1) {
                    year = "_2019";
                }
                return tableName + year;
            });
        }};

        dynamicTableNameInnerInterceptor.setTableNameHandlerMap(map);
        interceptor.addInnerInterceptor(dynamicTableNameInnerInterceptor);
        // 动态表名结束,.,,,,

       // 1. 添加分页插件
        PaginationInnerInterceptor pageInterceptor = new
PaginationInnerInterceptor();
        // 2. 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
        pageInterceptor.setOverflow(false);
        // 3. 单页分页条数限制，默认无限制
        pageInterceptor.setMaxLimit(500L);
        // 4. 设置数据库类型
        pageInterceptor.setDbType(DbType.MYSQL);

        interceptor.addInnerInterceptor(new TenantLineInnerInterceptor(new
TenantLineHandler(){

            @Override
            public Expression getTenantId() {
                // 租户信息 一般是session或者配置文件中,或者静态变量中
                // 1088248166370832385  王天风的
                return new LongValue(1088248166370832385L);
            }

            @Override
            public String getTenantIdColumn() {
```

```
                // 多用户字段是什么,表中的字段名字
                return "manager_id";
            }

            // 这是 default 方法,默认返回 false 表示所有表都需要拼多租户条件
            // 某些表不加租户信息,
            @Override
            public boolean ignoreTable(String tableName) {
                // 如果查是user_high ->false不忽略(除了这个都忽略)
                // !"user_high"就是不过滤,"user_high"就是过滤
                return !"user_high".equalsIgnoreCase(tableName);
            }
        }));

        // 如果用了分页插件注意先 add TenantLineInnerInterceptor 再 add
PaginationInnerInterceptor
        // 用了分页插件必须设置 MybatisConfiguration#useDeprecatedExecutor = false
//        interceptor.addInnerInterceptor(new PaginationInnerInterceptor());

        // 5.添加内部拦截器
        interceptor.addInnerInterceptor(pageInterceptor);
        // 6.乐观锁插件 添加乐观锁插件
        interceptor.addInnerInterceptor(new OptimisticLockerInnerInterceptor());

        return interceptor;
    }

    @Bean
    public ConfigurationCustomizer configurationCustomizer() {
        // 需要设置 MybatisConfiguration#useDeprecatedExecutor = false 避免缓存出现
问题(该属性会在旧插件移除后一同移除)
        return configuration -> configuration.setUseDeprecatedExecutor(false);
    }
}
```

官网有案例自己看吧...很简单的 `user_high_2019` 和 `user_high_2018` 随机换

```
DEBUG==>  Preparing: SELECT id, name, age, email, manager_id, create_time,
update_time, version FROM user_high_2019 WHERE deleted = 0
DEBUG==> Parameters:
 Consume Time：11 ms 2021-04-17 22:19:20
 Execute SQL：SELECT id, name, age, email, manager_id, create_time, update_time,
version FROM user_high_2019 WHERE deleted = 0
```

注意事项

记得改表名

特定sql替换也没用

sql注入器

用到再说吧....

感觉不常用啊...