链滴

MyBatis-Plus 使用详解

作者：wlgzs-sjl

原文链接：https://ld246.com/article/1605407214400

来源网站：链滴

导入依赖

```
<!--MP依赖-->
        <dependency>
            <groupId>com.baomidou</groupId>
            <artifactId>mybatis-plus-boot-starter</artifactId>
            <version>3.4.1</version>
        </dependency>
        <dependency>
            <groupId>com.baomidou</groupId>
            <artifactId>mybatis-plus-generator</artifactId>
            <version>3.4.1</version>
        </dependency>
        <dependency>
            <groupId>org.apache.velocity</groupId>
            <artifactId>velocity-engine-core</artifactId>
            <version>2.2</version>
        </dependency>
```

连接数据库

```
spring:
  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver
    url: jdbc:mysql://localhost:3306/mybatis-plus? useSSL=false&useUnicode=true&characterE
coding=utf-8&serverTimezone=GMT%2B8
    username: root
    password: root
```

主键生成策略

主键自增

我们需要配置主键自增：

1. 实体类字段上 @TableId(type = IdType.AUTO)
2. 数据库字段一定要是自增！

其他的源码解释

```
public enum IdType {
AUTO(0), // 数据库id自增
NONE(1), // 未设置主键
INPUT(2), // 手动输入
ID_WORKER(3), // 默认的全局唯一id
UUID(4), // 全局唯一id uuid
ID_WORKER_STR(5); //ID_WORKER 字符串表示法
}
```

自动填充

创建时间、修改时间！这些个操作一遍都是自动化完成的，我们不希望手动更新！

阿里巴巴开发手册：所有的数据库表：gmt_create、gmt_modified几乎所有的表都要配置上！而需

要自动化！

方式一：数据库级别（工作中不允许修改数据库）

1. 在表中新增字段 create_time, update_time
2. 把实体类同步！

```
private Date createTime;
private Date updateTime;
```

方式二：代码级别

1. 删除数据库的默认值、更新操作！
2. 实体类字段属性上需要增加注解

```
// 字段添加填充内容
@TableField(fill = FieldFill.INSERT)
private Date createTime;
@TableField(fill = FieldFill.INSERT_UPDATE)
private Date updateTime;
```

3. 编写处理器来处理这个注解即可！

```
package com.wlgzs.mybatisplus.handler;

import com.baomidou.mybatisplus.core.handlers.MetaObjectHandler;
import org.apache.ibatis.reflection.MetaObject;
import org.springframework.stereotype.Component;

import java.util.Date;

/**
 * @author wlgzs-sjl
 * @date 2020/11/15 10:05
 */
@Component // 一定不要忘记把处理器加到IOC容器中！
public class MyMetaObjectHandler implements MetaObjectHandler {
    // 插入时的填充策略
    @Override
    public void insertFill(MetaObject metaObject) {
        // setFieldValByName(String fieldName, Object fieldVal, 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) {
        this.setFieldValByName("updateTime", new Date(), metaObject);
    }
}
```

乐观锁

乐观锁 : 故名思意十分乐观，它总是认为不会出现问题，无论干什么不去上锁！如果出现了问题，

再次更新值测试

悲观锁：故名思意十分悲观，它总是认为总是出现问题，无论干什么都会上锁！再去操作！

乐观锁实现方式：

- 取出记录时，获取当前 version
- 更新时，带上这个version
- 执行更新时， set version = newVersion where version = oldVersion
- 如果version不对，就更新失败

使用方法

1. 给数据库中增加version字段
2. 给实体类加对应的字段

```
@Version //乐观锁Version注解
```

```
private Integer version;
```

分页查询

1. 配置拦截器组件

```
// 扫描我们的 mapper 文件夹
@MapperScan("com.wlgzs.mybatisplus.mapper")
@Configuration // 配置类
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;
    }
}
```

2. 直接使用Page对象

```
// 测试分页查询
@Test public 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);
   System.out.println(page.getTotal());
}
```

逻辑删除

物理删除： 从数据库中直接移除

逻辑删除： 再数据库中没有被移除，而是通过一个变量来让他失效！ deleted = 0 => deleted = 1

管理员可以查看被删除的记录！防止数据的丢失，类似于回收站！

实现方式

1. 在数据表中增加一个 deleted 字段
2. 实体类中增加属性

```
@TableLogic //逻辑删除
private Integer deleted;
```

3. 配置

```
mybatis-plus:
  global-config:
    db-config:
      logic-delete-field: flag  # 全局逻辑删除的实体字段名(since 3.3.0,配置后可以忽略不配置步骤2)
      logic-delete-value: 1 # 逻辑已删除值(默认为 1)
      logic-not-delete-value: 0 # 逻辑未删除值(默认为 0)
```

代码自动生成器

在test包下新建GeneratorCode

```
package com.wlgzs.mybatisplus;

import com.baomidou.mybatisplus.annotation.DbType;
import com.baomidou.mybatisplus.annotation.IdType;
import com.baomidou.mybatisplus.generator.config.*;
import com.baomidou.mybatisplus.generator.AutoGenerator;
import com.baomidou.mybatisplus.generator.config.rules.DateType;
import com.baomidou.mybatisplus.generator.config.rules.NamingStrategy;


/**
 * @author wlgzs-sjl
 * @date 2020/11/14 15:52
 */
public class GeneratorCode {
    public static void main(String[] args) {
        // 代码生成器
        AutoGenerator mpg = new AutoGenerator();

        // 全局配置
        GlobalConfig gc = new GlobalConfig();
        String projectPath = System.getProperty("user.dir");
        gc.setOutputDir(projectPath + "/src/main/java");
        gc.setAuthor("wlgzs-sjl");
        gc.setOpen(false);
        gc.setFileOverride(false);// 是否覆盖
        gc.setServiceName("%sService");// 去Service的I前缀
        gc.setIdType(IdType.AUTO);
        gc.setDateType(DateType.ONLY_DATE);
        gc.setSwagger2(true); //实体属性 Swagger2 注解
        mpg.setGlobalConfig(gc);

        //2、设置数据源
        DataSourceConfig dsc = new DataSourceConfig();
        dsc.setUrl("jdbc:mysql://localhost:3306/mybatis-plus?useSSL=false&useUnicode=true&c
aracterEncoding=utf-8&serverTimezone=GMT%2B8");
        dsc.setDriverName("com.mysql.cj.jdbc.Driver");
        dsc.setUsername("root");
        dsc.setPassword("root");
        dsc.setDbType(DbType.MYSQL);
        mpg.setDataSource(dsc);
```

```
        // 包配置
        PackageConfig pc = new PackageConfig();
        pc.setParent("com.wlgzs.mybatisplus");
        pc.setEntity("entity");
        pc.setMapper("mapper");
        pc.setService("service");
        pc.setController("controller");
        mpg.setPackageInfo(pc);

        // 策略配置
        StrategyConfig strategy = new StrategyConfig();
        strategy.setNaming(NamingStrategy.underline_to_camel);
        strategy.setColumnNaming(NamingStrategy.underline_to_camel);
        strategy.setEntityLombokModel(true);// 自动lombok;
        strategy.setRestControllerStyle(true);
        strategy.setInclude("user");// 设置要映射的表名
        mpg.setStrategy(strategy);

    // 自动填充配置
    TableFill gmtCreate = new TableFill("gmt_create", FieldFill.INSERT);
    TableFill gmtModified = new TableFill("gmt_modified", FieldFill.INSERT_UPDATE);
    ArrayList<TableFill> tableFills = new ArrayList<>();
    tableFills.add(gmtCreate);
    tableFills.add(gmtModified);
    strategy.setTableFillList(tableFills);

        mpg.execute(); //执行
    }
}
```

注意：我们需要在主启动类上去扫描我们的mapper包下的所有接口

@MapperScan("com.wlgzs.mybatisplus.mapper")

```
@MapperScan("com.wlgzs.mybatisplus.mapper")
@SpringBootApplication
public class MybatisPlusApplication {

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

}
```

写在最后

博客参考B站狂神说Java之mybatis-plus教程以及mybatis-plus官方文档