MybatisPlus

官方地址:

<http://mp.baomidou.com>

代码发布地址:

Github: <https://github.com/baomidou/mybatis-plus>

Gitee: <https://gitee.com/baomidou/mybatis-plus>

pom依赖:

<dependency>

<groupId>com.baomidou</groupId>

<artifactId>mybatis-plus</artifactId>

<version>2.3</version>

</dependency>

PS: Mybatis和Mybatis-Spring就无需导入,自动集成

SqlSessionFactoryBean对比：

Mybatis: org.mybatis.spring.SqlSessionFactoryBean

MP: com.baomidou.mybatisplus.spring.MybatisSqlSessionFactoryBean

MP的全局策略配置:

<bean id=”globalConfiguration” class="com.baomidou.mybatisplus.entity.

GlobalConfiguration">

<property name="dbColumnUnderline" value="true"></property>

<property name="idType" value="0"></property>

<property name="tablePrefix" value=""></property>

</bean>

dbColumnUnderline: 处理实体类驼峰命名法到数据库下滑线命名法的处理 在2.3版本以后默认值是true

idType: 全局的主键策略 0自增, 1用户输入, 2全局唯一id, 3全局唯一iduuid

4未设置主键类型,5全局字符串唯一id

tablePrefix: 全局的表前缀策略配置默””

实体类注解

@TableName 表名注解



@TableId 主键注解



主键策略: IdType.***AUTO*** 0 数据库ID自增

IdType.***INPUT*** 1 用户输入ID

PS: 以下类型、只有当插入对象ID 为空，才自动填充

IdType.***ID\_WORKER*** 2 全局唯一ID

IdType.***UUID*** 3 全局唯一ID

IdType.***NONE*** 4 该类型为未设置主键类型

IdType.***ID\_WORKER\_STR*** 5 字符串全局唯一ID

@TableField 字段注解



字段填充策略 fill



@Version 乐观锁标记注解

@TableLogic 逻辑删除标记注解

逻辑删除策略: value 默认逻辑未删除值 (该值可无、会自动获取全局配置)

delval 默认逻辑删除值 (该值可无、会自动获取全局配置)

通用 CRUD

实现方式:

基于 Mybatis

需要编写 Mapper 接口，并手动编写 CRUD 方法

提供 Mapper.xml 映射文件，并手动编写每个方法对应的 SQL 语句.

基于 MP

只需要创建 Mapper 接口, 并继承 BaseMapper 接口.这就是使用 MP

需要完成的所有操作，甚至不需要创建 SQL 映射文件。

插入操作:

Integer insert(T entity);

Integer insertAllColumn(T entity);

insert会忽略掉实体类为空的字段 sql语句忽略掉为空的字段

insertAllColumn 不会忽略掉实体类为空的字段 sql语句不忽略掉为空的字段

更新操作:

Integer updateById(@Param("et") T entity);

Integer updateAllColumnById(@Param("et") T entity);

updateById 会忽略掉实体类为空的字段 sql语句忽略掉为空的字段

updateAllColumnById 不会忽略掉实体类为空的字段 sql语句不忽略掉为空的字段

查询操作:

T selectById(Serializable id);

T selectOne(@Param("ew") T entity);

List<T> selectBatchIds(List<? extends Serializable> idList);

List<T> selectByMap(@Param("cm") Map<String, Object> columnMap);

List<T> selectPage(RowBounds rowBounds, @Param("ew") Wrapper<T> wrapper);

selectById 通过id来查找数据 参数是序列化对象

selectOne 通过条件来查找一条数据 如果查询到大于或等于两条数据,就会报错 如果没查到不报错

selectBatchIds 通过多个id来查找数据

selectByMap 通过多个map来查找数据

selectPage 分页查询 不是真实的分页 而是内存分页,第二个参数是条件构造器

删除操作:

Integer deleteById(Serializable id);

Integer deleteByMap(@Param("cm") Map<String, Object> columnMap);

Integer deleteBatchIds(List<? extends Serializable> idList);

deleteById 通过id删除信息

deleteByMap 通过条件map进行删除

deleteBatchIds 通过多个id来删除

MP 启动注入 SQL 原理分析

SQL语句所在:

xxxMapper > MapperProxy > sqlSession > sqlSessionFactory > configuration > mappedStatement > value > sqlSource > sqlSource > sql

启动时就会把baseMapper的每一个方法的每一个sql语句保存到configuration中 的mappedStatement里面

MP在启动就会挨个分析xxxMapper中的方法,并且将对应的SQL语句处理好,保 存到configuration对象的mappedStatement中

本质:

启动时实例化xxxMapper,通过addMappedStatement方法,将每一个sql语句

添加到mappedStatement里面

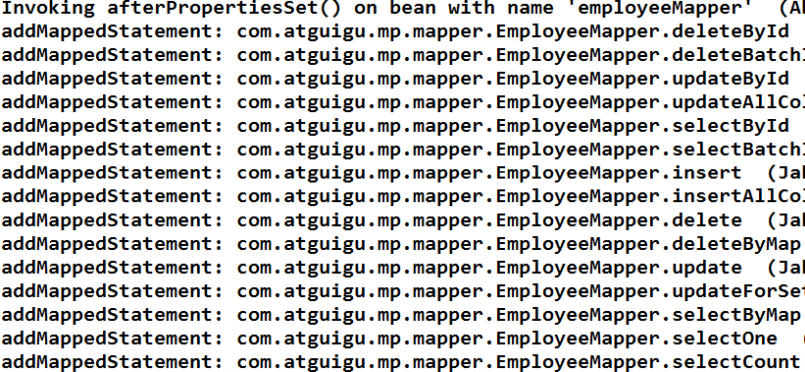
1. xxxMapper 的本质 org.apache.ibatis.binding.MapperProxy
2. MapperProxy 中 sqlSession > SqlSessionFactory
3. SqlSessionFacotry 中 → Configuration→ MappedStatements

每一个mappedStatement 都表示 Mapper接口中的一个方法与Mapper 映射文件

中的一个 SQL

MP 在启动就会挨个分析 xxxMapper 中的方法，并且将对应的 SQL 语句处理好

保存到 configuration 对象中的 mappedStatements 中



Configuration: MyBatis或者MP全局配置对象

AutoSqlInjector: 自动装配sql的类

MappedStatement: 一个MappedStatement对象对应Mapper配置文件中的一个

select/update/insert/delete节点,主要描述的是一条SQL语句

SqlMethod: 枚举对象, MP支持的SQL方法

TableInfo: 数据库表反射信息, 可以获取到数据库表相关的信息

SqlSource: SQL语句处理对象

MapperBuilderAssistant: 用于缓存、SQL 参数、查询方剂结果集处理等

通过MapperBuilderAssistant将每一个mappedStatement添加到configuration中的 mappedstatements 中

通用CRUD小结

以上是基本的CRUD操作,如您所见,我们仅仅需要继承一个BaseMapper即可实现大 部分单表CRUD操作。BaseMapper提供了多达 17 个方法给大家使用, 可 以极其 方便的实现单一、批量、分页等操作。极大的减少开发负担。

条件构造器EntityWrapper

MP通过EntityWrapper(简称 EW, MP 封装的一个查询条件构造器)或者

Condition(与EW类似) 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

PS: 使用的是数据库字段，不是 Java 属性!



条件参数说明:





PS: xxNew 都是另起 ( ... ) 括号包裹。

带条件的查询:

List<T> selectList(@Param("ew") Wrapper<T> wrapper);

带条件的修改:

Integer update(@Param("et") T entity, @Param("ew") Wrapper<T> wrapper);

带条件的删除:

Integer delete(@Param("ew") Wrapper<T> wrapper);

MP条件构造器: EntityWrapper Condition

MyBatis MBG: xxxExample --> Criteria : QBC(Query By Criteria)

AR模式 ActiveRecord(活动记录)

Active Record(活动记录), 是一种领域模型模式, 特点是一个模型类对应关系型数据库中的一个表, 而模型类的一个实例对应表中的一行记录。

ActiveRecord一直广受动态语言(PHP、Ruby 等)的喜爱, 而Java作为准静态语言, 对于ActiveRecord往往只能感叹其优雅, 所以MP也在AR道路上进行了一定的探索

如何使用AR模式:

仅仅需要让实体类继承 Model 类且实现主键指定方法，即可开启 AR 之旅

AR基本CRUD:

插入操作

public boolean insert()

修改操作

public boolean updateById()

查询操作

public T selectById()

public T selectById(Serializable id)

public List<T> selectAll()

public List<T> selectList(Wrapper wrapper)

public int selectCount(Wrapper wrapper)

删除操作

public boolean deleteById()

public boolean deleteById(Serializable id)

public boolean delete(Wrapper wrapper)

分页复杂操作

public Page<T> selectPage(Page<T> page, Wrapper<T> wrapper)

AR小结

AR模式提供了一种更加便捷的方式实现 CRUD 操作，其本质还是调用的 Mybatis 对应的方法，类似于语法糖

语法糖是指计算机语言中添加的某种语法, 这种语法对原本语言的功能并没有影响, 可以更方便开发者使用, 可以避免出错的机会, 让程序可读性更好

代码生成器

pom依赖:

MP的代码生成器默认使用的是Apache的Velocity模板, 也可以更换别的模板, 例如 freemarker

<!-- Velocity -->

<dependency>

<groupId>org.apache.velocity</groupId>

<artifactId>velocity-engine-core</artifactId>

<version>2.0</version>

</dependency>

<!-- slf4j -->

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-api</artifactId>

<version>1.7.7</version>

</dependency>

<!-- slf4j-log4j -->

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-log4j12</artifactId>

<version>1.7.7</version>

</dependency>

ServiceImpl 说明:

xxxServiceImpl继承了ServiceImpl类, MP通过这种方式为我们注入了 xxxMapper, 这样可以使用 service 层默认为我们提供的很多方法类似AR模 式 ,也可以调 用我们自己在 dao 层编写的操作数据库的方法.

插件扩展

插件机制:

Mybatis 通过插件(Interceptor)可以做到拦截四大对象相关方法的执行, 根据需求, 完成相关数据的动态改变。

mybatis四大对象:

Executor 执行器对象

StatemeHandler 编译器对象

ParameterHandler 参数处理器对象

ResultSetHandler结果集处理器对象

插件原理:

四大对象的每个对象在创建时, 都会执行interceptorChain.pluginAll(), 会经过 每个插件对象的plugin()方法, 目的是为当前的四大对象创建代理。代理对象 就可以拦截到四大对象相关方法的执行, 因为要执行四大对象的方法需要经 过代理.如果没有插件, 那就不会生成

分页插件:

com.baomidou.mybatisplus.plugins.PaginationInterceptor

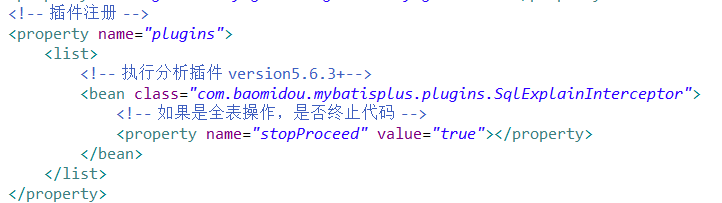
spring.xml



执行分析插件:

com.baomidou.mybatisplus.plugins.SqlExplainInterceptor

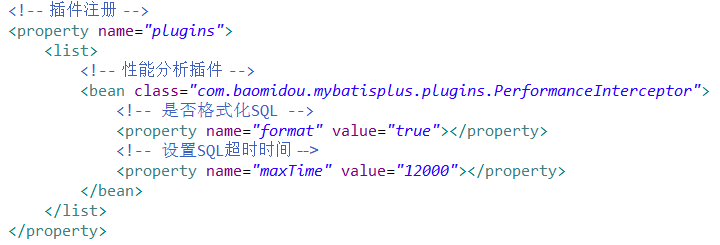
1. 只支持MySQL5.6.3以上版本
2. 分析DELETE UPDATE语句,防止小白或者恶意进行DELETE UPDATE全表操作
3. 只建议在开发环境中使用, 不建议在生产环境使用
4. 在插件的底层通过SQL语句分析命令:Explain分析当前的SQL语句,根据结果 集中的Extra列来断定当前是否全表操作。



性能分析插件:

com.baomidou.mybatisplus.plugins.PerformanceInterceptor

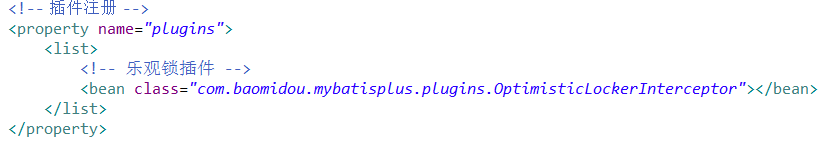
1. 性能分析拦截器, 用于输出每条SQL语句及其执行时间
2. 建议开发环境使用, 超过指定时间, 停止运行, 有助于发现问题



乐观锁插件:

com.baomidou.mybatisplus.plugins.OptimisticLockerInterceptor

1. 当要更新一条记录的时候, 希望这条记录没有被别人更新就可以使用
2. 乐观锁的实现原理: 取出记录时,获取当前version 2更新时, 带上这个 version 2 执行更新时, set version = yourVersion+1 where version = yourVersion 如果 version不对, 就更新失败
3. @Version 用于注解实体字段，必须要有。



自定义全局操作

AutoSqlInjector简介:

根据MP的AutoSqlInjector可以自定义各种你想要的sql ,注入到全局中, 相当 于自定义MP自动注入的方法。

之前需要在xml中进行配置的SQL语句, 现在通过扩展AutoSqlInjector在加载 mybatis环境时就注入。

AutoSqlInjector使用:

1. 在xxxMapper接口中定义相关的CRUD方法
2. 扩展AutoSqlInjector inject方法, 实现Mapper接口中方法要注入的SQL
3. 在MP全局策略中, 配置自定义注入器

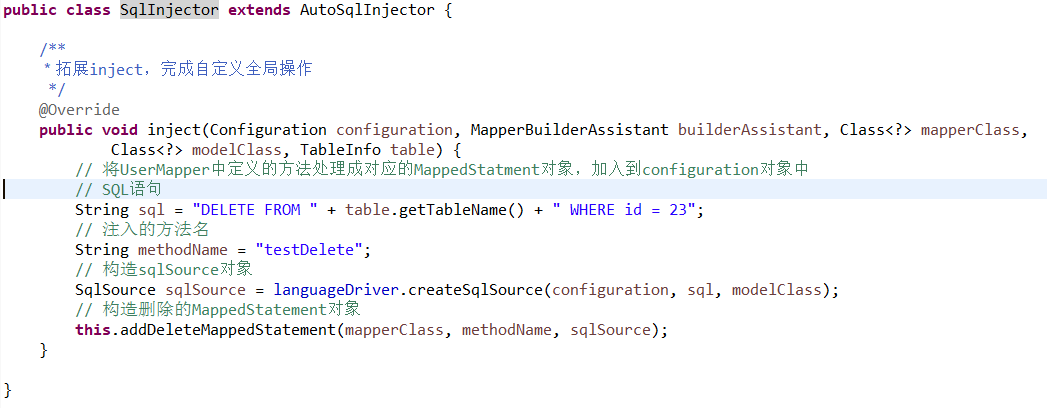
spring.xml



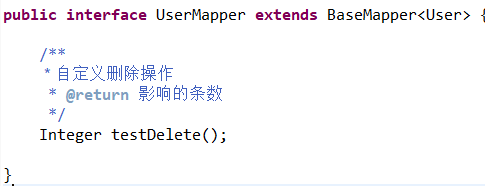
AutoSqlInjector.java

自定义delete操作, 都需要继承AutoSqlInjector

如果需要其他操作 直接**this**.add其他方法构造对应的MappedStatement对象



UserMapper.java



自定义注入器 — 逻辑删除 简介:

假删除、逻辑删除: 并不会真正的从数据库中将数据删除掉, 而是将当前被删 除的这条数据中的一个逻辑删除字段置为删除状态

逻辑删除 使用:

com.baomidou.mybatisplus.mapper.LogicSqlInjector

logic\_flag = 1 --> -1

1. logicDeleteValue 逻辑删除全局值
2. logicNotDeleteValue 逻辑未删除全局值

在实体类的逻辑删除字段添加 @TableLogic 注解

会在mp自带查询和更新方法的sql后面

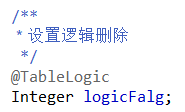
追加[逻辑删除字段] = [LogicNotDeleteValue默认值]

删除方法: deleteById()和其他delete方法, 底层SQL调用的是更新操作

spring.xml



pojo



公共字段自动填充

元数据处理器接口:

com.baomidou.mybatisplus.mapper.MetaObjectHandler

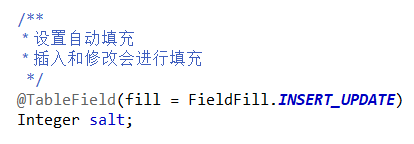
spring.xml



MyMetaObjectHandler.java:

需要继承MetaObjectHandler





metaobject: 元对象 是Mybatis提供的一个用于更加方便, 更加优雅的访问对 象的属性,给对象的属性设置值的一个对象 还会用于包装对象, 支持对 Object 、Map、Collection等对象进行包装

本质上 metaObject 获取对象的属性值或者是给对象的属性设置值, 最终是要

通过 Reflector 获取到属性的对应方法的 Invoker, 最终 invoke

XML热加载

参数说明：

sqlSessionFactory: sqlSession工厂

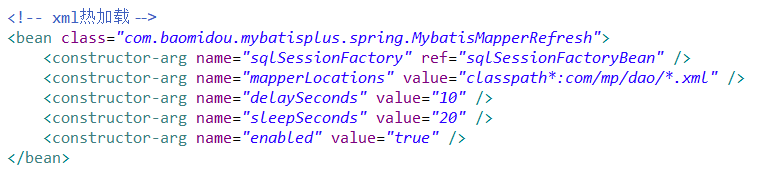
mapperLocations: mapper路径

enabled: 是否开启动态加载 默认:false

delaySeconds: 项目启动延迟加载时间 单位: 秒 默认: 10s

sleepSeconds: 刷新时间间隔 单位: 秒 默认: 20s

spring.xml



<bean class=*"com.baomidou.mybatisplus.spring.MybatisMapperRefresh"*>

<constructor-arg name=*"sqlSessionFactory"* ref=*"sqlSessionFactoryBean"*/>

<constructor-arg name=*"mapperLocations"* value=*"classpath\*:com/mp/dao/\*.xml"*/>

<constructor-arg name=*"delaySeconds"* value=*"10"*/>

<constructor-arg name=*"sleepSeconds"* value=*"20"*/>

<constructor-arg name=*"enabled"* value=*"true"*/>

</bean>