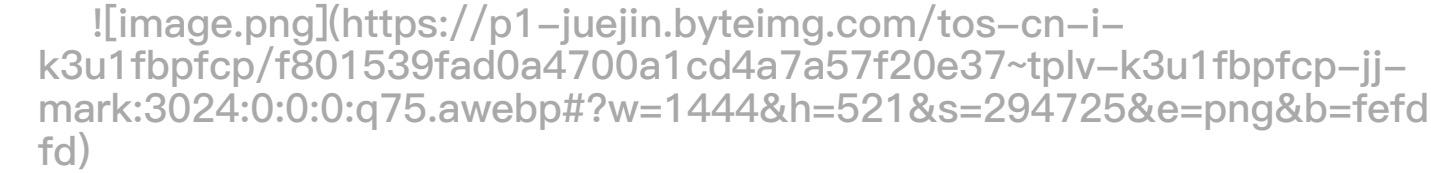
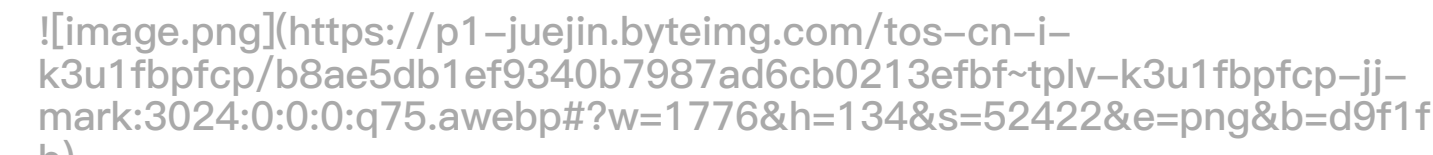
MybatisPlus实现数据权限隔离



引言

Mybatis Plus对Mybatis做了无侵入的增强，非常的好用，今天就给大家介绍它的其中一个实用功能：数据权限插件。

数据权限插件的应用场景和多租户的动态拦截拼接SQL一样。建议点赞+收藏+，方便以后复习查阅。



依赖

首先导入Mybatis Plus的maven依赖，我使用的是3.5.3.2版本。

```
<properties>
    <mybatis-plus.version>3.5.3.2</mybatis-plus.version>
</properties>

<dependency>
    <groupId>com.baomidou</groupId>
    <artifactId>mybatis-plus-boot-starter</artifactId>
    <version>${mybatis-plus.version}</version>
</dependency>
```

```

数据权限拦截器
=======

写一个自定义的权限注解，该注解用来标注被拦截方法，注解上可以配置数据权限的表别名和表字段，它们会在拼接sql的时候用到。

```
import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

@Target({ElementType.METHOD, ElementType.TYPE})
@Retention(RetentionPolicy.RUNTIME)
public @interface MyDataScope {
    /**
     * 表别名设置
     */
    String alias() default "";

    /**
     * 数据权限表字段名
     */
    String dataId() default "";
}
```

接下来就是写最核心的拦截器的处理逻辑了。创建一个接口实现类，实现Mybatis Plus的DataPermissionHandler接口。DataPermissionHandler的接口方法getSqlSegment有两个参数。

* Expression where。where参数是mapper接口在xml中定义的sql的where条件表达式，在拦截处理器中我们可以给where条件表达式添加一些 and 或 or 的条件。
* String mappedStatementId。mappedStatementId参数是mapper接口方法的全限定名，通过它我们可以得到mapper接口的Class类名以及接口方法名。

DataPermissionHandler的接口方法getSqlSegment会返回一个Expression类型的结果，即通过拦截器方法我们将原始的where条件表达式做了修改之后返回
```

给Mybatis Plus并在代码运行时生效。

在拦截器方法中还使用到了一开始我们自定义的MyDataScope注解，没有被MyDataScope注解标注过的mapper方法我们直接返回原始的where条件表达式即可。

```
import cn.hutool.core.util.StrUtil;
import
com.baomidou.mybatisplus.extension.plugins.handler.DataPermissionHandler;
import com.itguoguo.annotation.MyDataScope;
import com.itguoguo.system.api.model.LoginUser;
import lombok.extern.slf4j.Slf4j;
import net.sf.jsqlparser.JSQLParserException;
import net.sf.jsqlparser.expression.Expression;
import net.sf.jsqlparser.expression.operators.conditional.AndExpression;
import net.sf.jsqlparser.parser.CCJSqlParserUtil;

import java.lang.reflect.Method;
import java.util.Objects;

import static com.itguoguo.utils.LoginUserUtils.getLoginUser;

@Slf4j
public class MyDataScopeHandler implements DataPermissionHandler {
    /**
     * 获取数据权限 SQL 片段表达式
     * @param where             待执行 SQL Where 条件表达式
     * @param mappedStatementId Mybatis MappedStatement Id 根据该参数可以判断具体执行方法
     * @return 数据权限 SQL 片段表达式
     */
    @Override
    public Expression getSqlSegment(Expression where, String mappedStatementId) {
        try {
            String className = mappedStatementId.substring(0, mappedStatementId.lastIndexOf("."));
            String methodName = mappedStatementId.substring(mappedStatementId.lastIndexOf(".") + 1);
            Method[] methods = Class.forName(className).getMethods();
            for (Method m : methods) {
                if (StrUtil.isBlank(m.getName()) || !m.getName().equals(methodName)) {
```

```
                continue;
            }
            MyDataScope annotation =
m.getAnnotation(MyDataScope.class);
            if (Objects.isNull(annotation)) {
                return where;
            }
            String sqlSegment = getSqlSegment(annotation);
            return StrUtil.isBlank(sqlSegment) ? where :
getExpression(where, sqlSegment);
        }
    } catch (ClassNotFoundException e) {
        log.error(e.getMessage(), e);
    }
    return null;
}

/**
 * 拼接需要在业务 SQL 中额外追加的数据权限 SQL
 * @param annotation
 * @return 数据权限 SQL
 */
private String getSqlSegment(MyDataScope annotation) {
    LoginUser loginUser = getLoginUser();
    String userType = loginUser.getSysUser().getUserType();
    Long userId = loginUser.getSysUser().getUserId();
    String sqlSegment = "";
    if (StrUtil.isBlank(userType)) {
        return sqlSegment;
    }
    if ("0".equals(userType)) {
        return sqlSegment;
    } else {
        sqlSegment = StrUtil.format(" {}.{} IN (SELECT project_id FROM
sys_user where user_id = '{}') ",
                annotation.alias(), annotation.dataId(), userId);
    }
    return sqlSegment;
}

/**
 * 将数据权限 SQL 语句追加到数据权限 SQL 片段表达式里
 * @param where         待执行 SQL Where 条件表达式
 * @param sqlSegment    数据权限 SQL 片段
 * @return
 */
private Expression getExpression(Expression where, String
sqlSegment) {
```

```
        try {
            Expression sqlSegmentExpression =
CCJSqlParserUtil.parseCondExpression(sqlSegment);
            return (null != where) ? new AndExpression(where,
sqlSegmentExpression) : sqlSegmentExpression;
        } catch (JSQLParserException e) {
            log.error(e.getMessage(), e);
        }
        return null;
    }
}
```

拦截器配置
=====

配置Mybatis Plus拦截器，数据权限handler作为参数传给拦截器构造方法。

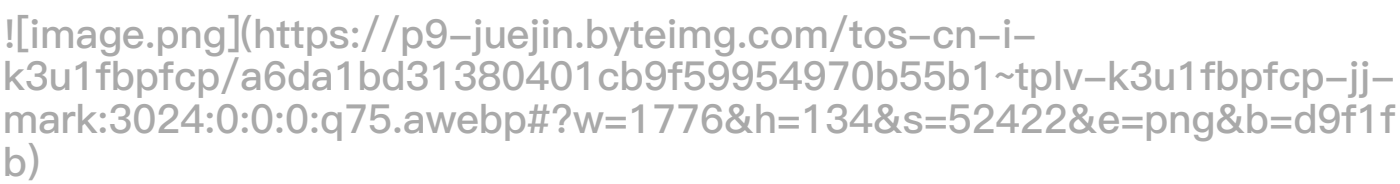
```
import
com.baomidou.mybatisplus.extension.plugins.MybatisPlusInterceptor;
import
com.baomidou.mybatisplus.extension.plugins.inner.DataPermissionInterc
eptor;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class MybatisPlusConfig {
    @Bean
    public MybatisPlusInterceptor mybatisPlusInterceptor() {
        MybatisPlusInterceptor interceptor = new MybatisPlusInterceptor();
        interceptor.addInnerInterceptor(new DataPermissionInterceptor(new
MyDataScopeHandler()));
        return interceptor;
    }
}
```

使用
==

使用时，在mapper接口的方法上标注MyDataScope注解，给注解标上表别名和表字段。

```
public interface MyMapper {
    @MyDataScope(alias = ”a”, dataId = ”id”)
    List findList();
}
```

建议点赞+收藏+，方便以后复习查阅。



原文链接: https://juejin.cn/post/7355759709167271947