第三篇 自定义SQL

> **虽然Mybatis Plus帮我们提供了大量的默认方法，但我们为了实现多表关联查询，或者根据不同的查询条件传参，实现不同的动态SQL。在这种情况下我们还是需要自定义SQL，不管怎样我们需要首先通过配置指定Mapper.xml文件的存储位置。**

1、原始的自定义SQL方法

> **将多表关联查询或动态SQL写在XML文件里面进行维护，大多数场景下仍然是Mybatis最佳实践。**
>
> **单表的增删改查使用Mybatis Plus或者mybatis generator生成代码，是最佳实践。**

- UserMapper 接口放在 `@MapperScan` 配置的扫描路径下面。这种方法是Mybatis 为我们提供的，在Mybatis Plus里面仍然可以继续使用!
- 使用最原始的Mybatis SQL定义方式，在集成BaseMapper的基础上（mybatis plus），新定义一个接口方法findUser。

```
public interface UserMapper extends BaseMapper<User> {
  List<User> findUser(@Param("name") String name,
@Param("email") String email);
}
```

新定义一个UserMapper.xml,放在 `mybatis-plus.mapper-locations` 配置路径下面。

下面的动态SQL表示：

- 当参数name不为null或空串的时候, `AND name = #{name}` 条件生效
- 当参数email不为null或空串的时候, `AND email = #{email}` 条件生效

```
<!--这个里面写动态SQL、多表关联查询都可以胜任-->
<select id="findUser" resultType="User">
    SELECT id,name,age,email
    FROM user
    <trim prefix="WHERE" prefixOverrides="AND|OR"
suffixOverrides="AND|OR">
        <if test="name != null and name != '' " >
            AND name = #{name}
        </if>
        <if test="email != null and email != '' " >
            AND email= #{email}
        </if>
    </trim>
</select>
```

使用测试

```
@Test
public void testCustomSQL() {
  String name = "Jack";  //name不为空
  String email = "";     //email为空串
  List<User> list = userMapper.findUser(name,email);
  list.forEach(System.out::println);
}
```

最终执行的SQL为（因为email为空串，所以对应的查询条件在动态SQL中未被构建）：

```
SELECT id,name,age,email
FROM user
WHERE name = ?
```

2、自定义接口方法使用Wrapper条件构造器

如果我们想在自定义的方法中，使用Wrapper条件构造器。可以参考下面的方式实现。这种方式虽然简单，但仍然只适用于单表（可以是多表关联查询，但查询条件也是基于单表的）。

- 使用注解方式 + Wrapper ,

```
 ${ew.customSqlSegment}
```

是一个查询条件占位符，代表Wapper查询条件。

```
@Select("select * from `user` ${ew.customSqlSegment}")
List<User> selectAll(@Param(Constants.WRAPPER) Wrapper
wrapper);
```

- 使用xml 配置方式 + Wrapper

```
List<User> selectAll(@Param(Constants.WRAPPER) Wrapper
wrapper);
```

```
<select id="selectAll" resultType="User">
   select * from `user` ${ew.customSqlSegment}
</select>
```

通过Wapper传递查询参数

上面两种方式任意选择一种，参数都是Wrapper

```
@Test
public void testCustomSQL2() {
  LambdaQueryWrapper<User> query = new
LambdaQueryWrapper<>();
  query.eq(User::getName, "Jack");
  List<User> list = userMapper.selectAll(query);
  list.forEach(System.out::println);
}
```

最终执行的SQL为（和上文原始的XML动态SQL实现效果一致，但是查询条件的构造是针对单表的）：

```
SELECT id,name,age,email
FROM user
WHERE name = ?
```