# MyBatis注解开发之@Results

解决实体类字段和数据库结果集不匹配问题 所以需要手动指定映射关系

@Results各个属性的含义

id为当前结果集声明唯一标识

value值为结果集映射关系

@Result代表一个字段的映射关系

column指定数据库字段的名称

property指定实体类属性的名称

jdbcType数据库字段类型

@Result里的id值为true表明主键，默认false

使用@ResultMap来引用映射结果集，其中value可省略。

**声明结果集映射关系代码：**

```
@Select({"select id, name, class_id from my_student"})
@Results(id="studentMap", value={
    @Result(column="id", property="id", jdbcType=JdbcType.INTEGER, id=true),
    @Result(column="name", property="name", jdbcType=JdbcType.VARCHAR),
    @Result(column="class_id ", property="classId", jdbcType=JdbcType.INTEGER)
})
List<Student> selectAll();
```

**引用结果集代码：**

```
@Select({"select id, name, class_id from my_student where id = #{id}"})
@ResultMap(value="studentMap")
Student selectById(integer id);
```

这样我们就不用每次需要声明结果集映射的时候都复制冗余代码，简化开发，提高了代码的复用性。

MyBatis中使用@Results注解来映射查询结果集到实体类属性。

（1）@Results的基本用法。当数据库***字段名***与实体类对应的*属性名*不一致时，可以使用@Results映射来将其***对应***起来。

column为数据库字段名，porperty为实体类属性名，jdbcType为数据库字段数据类型，id为是否为主键。

```
@Select({"select id, name, class_id from my_student"})
@Results({
    @Result(column="id", property="id", jdbcType=JdbcType.INTEGER, id=true),
    @Result(column="name", property="name", jdbcType=JdbcType.VARCHAR),
    @Result(column="class_id", property="classId", jdbcType=JdbcType.INTEGER)
})
List<Student> selectAll();
```

如上所示的数据库字段名class_id与实体类属性名classId，就通过这种方式建立了映射关系。***名字相同的可以省略***。

（2）@ResultMap的用法。当这段@Results代码需要在多个方法用到时，为了提高***代码复用性***，我们可以为这个@Results***注解设置id***，然后使用@ResultMap注解来复用这段代码。

```
@Select({"select id, name, class_id from my_student"})
@Results(id="studentMap", value={
    @Result(column="id", property="id", jdbcType=JdbcType.INTEGER, id=true),
    @Result(column="class_id", property="classId", jdbcType=JdbcType.INTEGER)
})
List<Student> selectAll();

@Select({"select id, name, class_id from my_student where id = #{id}"})
@ResultMap(value="studentMap")
Student selectById(integer id);
```

（3）@One的用法。当我们需要通过查询到的一个***字段值作为参数***，去执行***另外一个方法***来查询关联的内容，而且两者是***一对一关系***时，可以使用***@One注解***来便捷的实现。比如当我们需要查询学生信息以及其所属班级信息时，需要以查询到的class_id为参数，来执行ClassesMapper中的selectById方法，从而获得学生所属的班级信息。可以使用如下代码。

```
@Select({"select id, name, class_id from my_student"})
@Results(id="studentMap", value={
    @Result(column="id", property="id", jdbcType=JdbcType.INTEGER, id=true),
    @Result(column="class_id", property="myClass", javaType=MyClass.class,
        one=@One(select="com.example.demo.mapper.MyClassMapper.selectById"))
})
List<Student> selectAllAndClassMsg();
```

（4）@Many的用法。与@One类似，只不过如果使用@One查询到的***结果是多行***，会抛出TooManyResultException异常，这种时候应该使用的是@Many注解，实现***一对多***的查询。比如在需要查询学生信息和每次考试的成绩信息时。

```
@Select({"select id, name, class_id from my_student"})
@Results(id="studentMap", value={
    @Result(column="id", property="id", jdbcType=JdbcType.INTEGER, id=true),
    @Result(column="class_id", property="classId", jdbcType=JdbcType.INTEGER),
    @Result(column="id", property="gradeList", javaType=List.class,

 many=@Many(select="com.example.demo.mapper.GradeMapper.selectByStudentId"))
})
List<Student> selectAllAndGrade();
```

（5）传递多个参数。首先我们给这张表增加age（年龄）和gender（性别）两个参数。当我们需要根据age和gender查询学生的午餐，这时需要改写column属性的格式。等号左侧的age和gender对应java接口的参数，右侧的对应数据库字段名。即将查到的my_student表中age和gender字段的值，分别赋给getLunchByAgeAndGender方法中的age和gender参数，去查询对应的name（午餐名）。

```
@Select("select id, name, age, gender from my_student")
@Results({
    @Result(column="id", property="id", jdbcType=JdbcType.INTEGER, id=true),
    @Result(column="class_id", property="classId", jdbcType=JdbcType.INTEGER),
    @Result(column="{age=age,gender=gender}", property="lunch",

 one=@One(select="com.example.demo.mapper.StudentMapper.getLunchByAgeAndGender")
),
    })
List<Student> selectAllAndLunch();

@Select("select name from lunch where student_age = #{age} and student_gender =
#{gender}")
String getLunchByAgeAndGender(@Param("age") int age, @Param("gender") int
gender);
```