# 一 关系映射

MyBatis 有两种不同的关联方式:

嵌套查询:通过执行另外一个 SQL 映射语句来返回预期的复杂类型。

嵌套结果:使用嵌套结果映射来处理重复的联合结果的子集。

使用的标签：association 一的关系、collection 多的关系

## 1 一对多/多对一

部门和员工之间关系

1）表结构

CREATE TABLE `t\_dept` (

`deptId` int(11) NOT NULL AUTO\_INCREMENT,

`deptName` varchar(20) DEFAULT NULL,

PRIMARY KEY (`deptId`)

)

CREATE TABLE `t\_employee` (

`empId` int(11) NOT NULL AUTO\_INCREMENT,

`empName` varchar(20) DEFAULT NULL,

`salary` double DEFAULT NULL,

`dept\_id` int(11) DEFAULT NULL,

PRIMARY KEY (`empId`),

KEY `FK\_jgo5u3npdj1lborc1v50bsens` (`dept\_id`),

CONSTRAINT `FK\_jgo5u3npdj1lborc1v50bsens` FOREIGN KEY (`dept\_id`) REFERENCES `t\_dept` (`deptId`)

)

2）嵌套结果：

DepartmentMapper.xml

<resultMap type=*"Department"* id=*"deptMap"*>

<!-- 主键 -->

<id property=*"id"* column=*"did"*/>

<!-- 其他字段 -->

<result property=*"name"* column=*"dname"*/>

<!-- 嵌套结果的设置方式

collention 表示集合关系

ofType 指定集合中对象的类型

javaType 指定存储数据的的集合的类型

-->

<collection property=*"emps"* javaType=*"java.util.ArrayList"* ofType=*"Employee"* resultMap=*"com.rr.one2many.Employee.empMap"*></collection>

</resultMap>

<!-- 多表查询 必须使用resultMap -->

<select id=*"findById"* parameterType=*"int"* resultMap=*"deptMap"*>

select d.id as did, d.name as dname,e.\*

from department d

inner join employee e

on d.id=e.dept\_id

where d.id=#{id};

</select>

其他操作：

<!-- 插入数据 -->

<insert id=*"add"* parameterType=*"Department"*>

<!-- keyproperty的值 是实体类中的属性

order 有两个值，

BEFORE表示插入前获取值

AFTER表示插入后获取值，对于mysql，插入后获取值 -->

<selectKey keyProperty=*"id"* resultType=*"int"* order=*"AFTER"*>

<!-- LAST\_INSERT\_ID() 获取最后插入的值 ,将获取的值赋值给keyProperty指定的属性 -->

select LAST\_INSERT\_ID()

</selectKey>

insert into department(name) values(#{name});

</insert>

<!-- 如果是mysql，可以使用useGeneratedKeys="true"属性 -->

<insert id=*"addByGK"* parameterType=*"Department"* useGeneratedKeys=*"true"* keyProperty=*"id"*>

insert into department(name) values(#{name});

</insert>

<!-- 得到表中的记录数，resultType 使用基本类型 -->

<select id=*"getCount"* resultType=*"int"*>

select count(1) from department;

</select>

EmployeeMapper.xml

<resultMap type=*"Employee"* id=*"empMap"*>

<!-- 主键 -->

<id property=*"id"* column=*"id"*/>

<!-- 其他字段 -->

<result property=*"name"* column=*"name"*/>

<result property=*"age"* column=*"age"*/>

<result property=*"birth"* column=*"birth"*/>

<!-- 嵌套结果 -->

<association property=*"dept"* resultMap=*"com.rr.one2many.Department.deptMap"*></association>

</resultMap>

3)嵌套查询：

DepartmentMapper.xml

<resultMap type=*"Department"* id=*"deptMap1"*>

<!-- 主键 -->

<id property=*"id"* column=*"id"*/>

<!-- 其他字段 -->

<result property=*"name"* column=*"name"*/>

<!-- 嵌套查询的设置方式

collention 表示集合关系

ofType 指定集合中对象的类型

javaType 指定存储数据的的集合的类型

-->

<collection property=*"emps"* javaType=*"java.util.ArrayList"* ofType=*"Employee"* column=*"id"* select=*"com.rr.one2many.Employee.findByDeptId"*></collection>

</resultMap>

<select id=*"findByDId"* parameterType=*"int"* resultMap=*"deptMap1"*>

select \*

from department d

where d.id=#{id};

</select>

EmployeeMapper.xml

<resultMap type=*"Employee"* id=*"empMap1"*>

<!-- 主键 -->

<id property=*"id"* column=*"id"*/>

<!-- 其他字段 -->

<result property=*"name"* column=*"name"*/>

<result property=*"age"* column=*"age"*/>

<result property=*"birth"* column=*"birth"*/>

<!-- 嵌套查询 -->

<association property=*"dept"* column=*"dept\_id"* select=*"com.rr.one2many.Department.findByDId"* fetchType=*"eager"*></association>

</resultMap>

<select id=*"findByDeptId"* parameterType=*"int"* resultMap=*"empMap"*>

select \* from employee where dept\_id=#{id};

</select>

<select id=*"findAll"* resultMap=*"empMap1"*>

select \* from employee;

</select>

## 2 多对多

学生和课程之间的关系

1）表结构

CREATE TABLE `t\_student` (

`sid` int(11) NOT NULL AUTO\_INCREMENT,

`sname` varchar(20) DEFAULT NULL,

PRIMARY KEY (`sid`)

)

CREATE TABLE `t\_course` (

`cid` int(11) NOT NULL AUTO\_INCREMENT,

`cname` varchar(20) DEFAULT NULL,

PRIMARY KEY (`cid`)

)

CREATE TABLE `t\_relation` (

`sid` int(11) NOT NULL,

`cid` int(11) NOT NULL,

PRIMARY KEY (`sid`,`cid`),

KEY `FK\_ma6ynxw6eng27dnn2h242ec6g` (`sid`),

CONSTRAINT `FK\_fje0x78jps0jamjagtdc80r8d` FOREIGN KEY (`sid`) REFERENCES `t\_student` (`sid`),

CONSTRAINT `FK\_ma6ynxw6eng27dnn2h242ec6g` FOREIGN KEY (`cid`) REFERENCES `t\_course` (`cid`)

)

2）嵌套结果

StudentMapper.xml

<resultMap type=*"Student"* id=*"stuMap"*>

<!-- 主键 -->

<id property=*"id"* column=*"stuid"*/>

<!-- 其他字段 -->

<result property=*"name"* column=*"stuname"*/>

<!-- 嵌套结果的设置方式

collention 表示集合关系

ofType 指定集合中对象的类型

javaType 指定存储数据的的集合的类型

-->

<collection property=*"courses"* javaType=*"java.util.ArrayList"* ofType=*"Course"* resultMap=*"com.rr.many2many.Course.courseMap"*></collection>

</resultMap>

<select id=*"findById"* parameterType=*"int"* resultMap=*"stuMap"*>

select s.id as stuid, s.name as stuname,c.id,c.name

from student s

inner join stu\_course sc

on s.id=sc.sid

inner join course c

on c.id=sc.cid

where s.id=#{id};

</select>

CourseMapper.xml

<resultMap type=*"Course"* id=*"courseMap"*>

<!-- 主键 -->

<id property=*"id"* column=*"id"*/>

<!-- 其他字段 -->

<result property=*"name"* column=*"name"*/>

<!-- 嵌套结果的设置方式

collention 表示集合关系

ofType 指定集合中对象的类型

javaType 指定存储数据的的集合的类型

-->

<collection property=*"students"* javaType=*"java.util.ArrayList"* ofType=*"Student"* resultMap=*"com.rr.many2many.Student.stuMap"*></collection>

</resultMap>

<!-- 使用嵌套查询，传过来的是学生id -->

<select id=*"findByStuId"* parameterType=*"int"* resultMap=*"courseMap"*>

select c.\*

from course c

inner join stu\_course sc

on c.id=sc.cid

where sc.sid=#{id};

</select>

3）嵌套查询

StudentMapper.xml

<resultMap type=*"Student"* id=*"stuMap1"*>

<!-- 主键 -->

<id property=*"id"* column=*"id"*/>

<!-- 其他字段 -->

<result property=*"name"* column=*"name"*/>

<!-- 嵌套查询的设置方式 -->

<collection property=*"courses"* javaType=*"java.util.ArrayList"* ofType=*"Course"* column=*"id"* select=*"com.rr.many2many.Course.findByStuId"*></collection>

</resultMap>

<select id=*"findBySId"* parameterType=*"int"* resultMap=*"stuMap1"*>

select \*

from student s

where s.id=#{id};

</select>

# 二 懒加载

也称为延时加载，刚开始查询的时候，不会查询出关联的数据，当使用这些数据时，再进行查询

映射文件中：fetchType="lazy"

主配置文件中：

<!-- 参数设置 -->

<settings>

<!-- 启用懒加载，必须按照下面方式配置 -->

<setting name=*"lazyLoadingEnabled"* value=*"true"*/>

<!-- 将积极加载改为消极加载即按需加载。必须写,且为false才会懒加载 -->

<setting name=*"aggressiveLazyLoading"* value=*"false"*/>

</settings>

# 三 缓存 理解

## 1 一级缓存

session级别的缓存

一级缓存是SqlSession级别的缓存。在操作数据库时需要构造 sqlSession对象，在对象中有一个(内存区域)数据结构（HashMap）用于存储缓存数据。不同的sqlSession之间的缓存数据区域（HashMap）是互相不影响的。

一级缓存的作用域是同一个SqlSession，在同一个sqlSession中两次执行相同的sql语句，第一次执行完毕会将数据库中查询的数据写到缓存（内存），第二次会从缓存中获取数据将不再从数据库查询，从而提高查询效率。当一个sqlSession结束后该sqlSession中的一级缓存也就不存在了。**Mybatis**默认开启一级缓存。

如果没有启动事务，mybatis的一级缓存在spring中是没有作用的.

## 2 二级缓存

mapper级别的缓存

**二级缓存**是mapper级别的缓存，多个SqlSession去操作同一个Mapper的sql语句，多个SqlSession去操作数据库得到数据会存在**二级缓存**区域，多个SqlSession可以共用**二级缓存**，**二级缓存**是跨SqlSession的。

**二级缓存**是多个SqlSession共享的，其作用域是mapper的同一个namespace，不同的sqlSession两次执行相同namespace下的sql语句且向sql中传递参数也相同即最终执行相同的sql语句，第一次执行完毕会将数据库中查询的数据写到缓存（内存），第二次会从缓存中获取数据将不再从数据库查询，从而提高查询效率。**Mybatis**默认没有开启二级缓存需要在setting全局参数中配置开启**二级缓存**。

注意：相关实体类需要实现Serializable接口，否则关闭session时候报错

主配置文件：

<!-- 开启二级缓存 -->

<setting name=*"cacheEnabled"* value=*"true"*/>

映射配置文件

<!-- 相关的映射启用二级缓存 -->

<!-- <cache /> -->

<!-- eviction 缓存策略

flushInterval 刷新间隔，单位毫秒

size 引用数目

readOnly 是否只读

-->

<cache

eviction=*"FIFO"*

flushInterval=*"10000"*

size=*"1000"*

readOnly=*"true"*

></cache>

flushInterval:

刷新间隔,可以被设置为任意的正整数，单位毫秒。默认情况是不设置，也就是没有刷新间隔。不设置，对数据进行修改时，被动刷新缓存

Size：

引用数目，可以被设置为任意正整数，缓存的对象数目和你运行环境的可用内存资源数目。默认值是1024。

readOnly：

是否只读，属性可以被设置为true或false。只读的缓存会给所有调用者返回缓存对象的相同实例（同一个对象的引用），默认是false（对象的副本）。

Eviction：

收回策略，

 LRU – 最近最少使用的:移除最长时间不被使用的对象。

FIFO – 先进先出:按对象进入缓存的顺序来移除它们。