代码地址

<https://github.com/lilewa/mybatisChicken/tree/master/relation>

本篇讲述接口方法参数和mapper文件中sql语句参数的关系。数据库表1对1关系，javabean 1对1关系和映射方法，懒加载

**为mapper中的sql传参**

mapper文件定义的sql语句中使用的**参数值**由接口方法提供。Sql语句中使用#加{}方式引用一个参数，如#{name}。

默认地mybatis按照接口方法中参数被定义的顺序提供引用名。接口方法的第一个参数使用#{0}或#{param1}引用，第二个参数使用 #{1}或#{param2}引用

**1个参数的接口方法**

当接口方法只有1个参数时，mybatis提供了一些方便的引用参数的规则

**基本类型的参数**：可以使用任意的参数名引用这个方法参数

有如下接口方法

SysUser selectUserCountry(Long id);

Mapper文件中的sql定义，在查询中使用了一个任意的参数名#{randomName}

<select id="selectUserCountry" resultMap="userCountryMap">  
 SELECT u.id,user\_name,user\_password,user\_email,user\_info,head\_img,create\_time,country\_id,  
 countryname country\_name,countrycode country\_code  
 FROM sys\_user u,country c where c.id=u.country\_id and c.id=#{randomName}  
</select>

**JavaBean参数**：JavaBean包含的字段使得只传入一个参数的情况下，在sql语句中可以使用多个参数值。Sql语句中可以直接使用字段名引用JavaBean的字段提供的值，例如传入如下类型的对象

public class SysUser {  
 private Long id;  
 private String userName;  
}

sql中可以使用的参数名：#{id}或#{userName}

**Map类型参数**：map类型的参数和javabean类似。Sql中通过key来引用map中该key对应的value值。如#{key}。使用map类型的好处是不用为了两三个参数创建新的javabean类

**多个参数的接口方法**

接口方法使用多个参数时，除了使用mybatis提供的默认参数名#{0}或#{param1}，还可以在定义接口方法时使用注解，手动指定方法参数在sql语句中的名字。例如

List<SysRole> selectRolesByUserIdAndRoleEnabled(@Param("userId") Long userId, @Param("enabled") Integer enabled);

Sql中使用#{ userId }和#{enabled}引用方法参数值

使用mybatis提供的默认名引用接口方法的多个参数示例。

新建两个JavaBean类，定义使用这两种JavaBean类型作为参数的接口方法

SysUser selectBy2JavaBean(SysUserId userId, SysUserName userName);

在mapper文件中定义sql语句，使用默认名引用这两个对象参数

<select id="selectBy2JavaBean" resultMap="userMap">  
 select \* from sys\_user where id = #{param1.id} and user\_name=#{param2.userName}  
</select>

**数据库表关系，javabean对象关系**

数据库中的表关联关系可以分为1对1，1对多，多对多

当我们在使用ORM框架时，通常将数据库表映射为javabean类。在javabean层面表示1对1关系时，我们使类A包含1个类B的对象

public class A{

private B objB;

}

在javabean层面表示1对多关系 时。类A包含类B的一个集合

public class A{

private List<B> listB;

}

在第一篇提到过，MyBatis并没有将Java 对象与数据库表关联起来，而是将Java 方法与SQL 语句关联。Mybatis不会窥探数据库的元数据，无法知道表之间的关系。因此javabean之间的关系无需和数据库表完全对应，开发人员可以根据自己的需要任意组合。

假如数据库中存在user表和country表。当程序需要1个用户的所有信息及他的国籍名时。为了方便程序实现，我们在定义user 实体类时，可以将country表，表示国籍名的列countryname添加到user实体类中。

public class Country{  
 private Long id;  
 private String userName;

private String countryname;

}

底层的sql语句查询需要关联这两张表作查询。

当业务发生变化，需要更多country表中的列，为了避免造成混乱。将一个country类对象定义在User实体类中，表明1对1关系是更好的做法

Javabean之间的包含只是定义了关联关系。javabean的字段值需要根据它们之间的关系选择恰当的映射方式（在mapper中配置）。

下面演示1对1关系的javabean及如何配置它们的映射

**1 准备数据库**

**1.1添加名为mybatis的数据库**

CREATE DATABASE mybatis DEFAULT CHARACTER SET utf8 COLLATE utf8\_general\_ci;

**1.2添加table**

CREATE TABLE `sys\_user` (  
 `id` bigint(20) NOT NULL AUTO\_INCREMENT COMMENT '用户ID',  
 `user\_name` varchar(50) DEFAULT NULL COMMENT '用户名',  
 `user\_password` varchar(50) DEFAULT NULL COMMENT '密码',  
 `user\_email` varchar(50) DEFAULT 'test@mybatis.tk' COMMENT '邮箱',  
 `user\_info` text COMMENT '简介',  
 `head\_img` blob COMMENT '头像',  
 `create\_time` datetime DEFAULT NULL COMMENT '创建时间',  
 `country\_id` int(11) DEFAULT NULL COMMENT '国籍',  
 PRIMARY KEY (`id`)  
) ENGINE=InnoDB AUTO\_INCREMENT=1035 DEFAULT CHARSET=utf8 COMMENT='用户表';

CREATE TABLE `country` (  
 `id` int(11) NOT NULL AUTO\_INCREMENT,  
 `countryname` varchar(255) DEFAULT NULL,  
 `countrycode` varchar(255) DEFAULT NULL,  
 PRIMARY KEY (`id`)  
) ENGINE=InnoDB AUTO\_INCREMENT=6 DEFAULT CHARSET=utf8;

**2 创建实体类**

public class Country {  
 private Long id;  
 private String countryname;  
 private String countrycode;

}

public class SysUser {  
 private Long id;  
 private String userName;  
 private String userPassword;  
 private String userEmail;  
 private String userInfo;  
 private byte[] headImg;  
 private Date createTime;  
 private Long countryId;  
 private Country country;

}

**3 创建接口类**

UserMapper.java

**3.1添加接口方法**

在UserMapper 接口中添加selectUserCountry 方法，此方法根据用户id查询用户信息及国籍信息

SysUser selectUserCountry(Long id);

**4 配置UserMapper接口对应的mapper文件**

**4.1增加查询sql语句**

<select id="selectUserCountry" resultMap="userCountryMap">  
 SELECT u.id,user\_name,user\_password,user\_email,user\_info,head\_img,create\_time,country\_id,  
 countryname country\_name,countrycode country\_code  
 FROM sys\_user u,country c where c.id=u.country\_id and c.id=#{id}  
</select>

**4.2使用<resultMap>元素配置查询结果映射规则**

<resultMap id="userCountryMap" type="mybatisChicken.relation.model.SysUser">  
 <id property="id" column="id"/>  
 <result property="userName" column="user\_name"/>

。。。。。。  
<result property="country.countryname" column="country\_name"/>  
<result property="country.countrycode" column="country\_code"/>

</resultMap>

注意到，在映射SysUser类型包含的country对象时，使用了country.countryname格式的字符串。指示将countryname列的值映射到country对象的countryname字段上

**5 增加测试类方法**

testselectUserCountry

在配置1对1关系的映射时也可以使用自动映射（不提供resultMap，在select元素中使用resultType特性） 。此时，列名或列别名要和对象字段相同，映射到关联对象的列使用"country.id"格式

<select id="selectUserCountry" resultType="mybatisChicken.relation.model.SysUser">  
 SELECT u.id,  
 user\_name userName,  
 user\_password userPassword,  
 user\_email userEmail,  
 country\_id countryId,  
 country\_id "country.id",  
 countryname "country.countryname",  
 countrycode "country.countrycode"  
 FROM sys\_user u,country c where c.id=u.country\_id and c.id=#{id}  
</select>

当列数据完全来自同一张表时，我们也可以假定列之间存在1对1关系。

**增加table**

CREATE TABLE `sys\_role` (  
 `id` bigint(20) NOT NULL AUTO\_INCREMENT COMMENT '角色ID',  
 `role\_name` varchar(50) DEFAULT NULL COMMENT '角色名',  
 `enabled` int(11) DEFAULT NULL COMMENT '有效标志',  
 `create\_by` bigint(20) DEFAULT NULL COMMENT '创建人',  
 `create\_time` datetime DEFAULT NULL COMMENT '创建时间',  
 PRIMARY KEY (`id`)  
) ENGINE=InnoDB AUTO\_INCREMENT=3 DEFAULT CHARSET=utf8 COMMENT='角色表';

**创建实体类**

public class CreateInfo {  
 private String createBy;// 创建人   
 private Date createTime;// 创建时间

}

public class SysRole {  
 private Long id;  
 private String roleName;  
 private String createBy;  
 private Date createTime;  
 private SysUser user;  
 private CreateInfo createInfo;

}

列create\_by和create\_time来自于同一张表sys\_role。我们在实体类SysRole中额外定义了一个1对1关系CreateInfo。createBy包含字段createBy和createTime

配置sql语句及resultMap映射规则

<select id="selectRoleById" resultMap="roleAutoMap">  
 select id, role\_name roleName,  
 create\_by createBy,  
 create\_time createTime  
 from sys\_role  
 where id = #{id}  
</select>

<resultMap id="roleAutoMap" autoMapping="true" type="mybatisChicken.relation.model.SysRole">  
 <id property="id" column="id" />  
 <result property="createInfo.createBy" column="createBy"/>  
 <result property="createInfo.createTime" column="createTime"/>  
</resultMap>

此处使用了autoMapping特性。当autoMapping特性值为true时，没有指定的列会被自动映射到同名对象字段中。打断点调试时可以看到role对象的roleName字段已经被自动映射。而role对象自身的createBy和createTime没有被自动映射，因为这两列已经被我们手动映射到createInfo的字段上。

**创建接口类和方法**

public interface RoleMapper {  
 SysRole selectRoleById(Long id);  
}

**创建测试类方法**

testselectUserCountry

**使用association元素显示配置1对1关联关系的映射**

association 标签包含以下特性。

property ：嵌套对象的名称，必填项。

javaType ：嵌套对象的Java 类型。

resultMap ：配置嵌套对象的映射规则。

columnPrefix ：查询列的前缀，配置前缀后，在配置result元素的column 时可以省略前缀。

现在改进上一示例selectUserCountry，使用association元素来配置1对1映射

**新建接口方法**

selectUserCountryAssociation

**配置方法对应的sql语句**

<select id="selectUserCountryAssociation" resultMap="associationUserCountryMap">  
 。。。。  
</select>

与selectUserCountry示例相比，唯一不同的是resultMap元素的配置，新增一个id为associationUserCountryMap的resultMap

<resultMap id="associationUserCountryMap" type="mybatisChicken.relation.model.SysUser">  
 <id property="id" column="id"/>  
 。。。。。。  
 <association property="country" javaType="mybatisChicken.relation.model.Country" >  
 <result property="id" column="country\_id"/>  
 <result property="countryname" column="country\_name"/>  
 <result property="countrycode" column="country\_code"/>  
</association>

</resultMap>

Association元素显示配置SysUser和Country的1对1关联关系的映射。Property特性值指定SysUser实体的关联对象名为country，javaType指定country的类型。映射规则作为子元素定义在Association元素中。Association元素也可以引用一个公共的映射配置resultMap，这样association 不必包含reslut子元素，javaType特性也不需设定，因为resultMap的type特性会指定映射的类型

定义映射规则时，不需要再使用country.countryname这种格式指定property的特性值

由于映射到country字段的数据列都有公共的前缀，我们可以使用columnPrefix提取公共前缀，这样在设置result的column时可以省略前缀

<association property="country" columnPrefix="country\_" javaType="mybatisChicken.relation.model.Country" >  
 <result property="id" column="id"/>  
 <result property="countryname" column="name"/>  
 <result property="countrycode" column="code"/>  
</association>

Association元素还有一个作用就是定义嵌套查询。

当需要获取的数据来自多张拥有关联关系的数据表时，我们有两种方式获取数据。

1 在一条sql查询语句中join多张数据表，只执行一次sql查询

2 分别执行sql查询每一张数据表，前面的查询结果作为后面sql查询条件的参数值

association 标签的嵌套查询常用的特性如下。

select ：另一个映射查询的id, MyBatis 会额外执行这个查询获取嵌套对象的结果。

column ：列名（或别名），将主查询中列的结果作为嵌套查询的参数，配置方式如column={propl=coll , prop2=col2}, propl 和prop2 将作为嵌套查询的参数。

fetchType ：数据加载方式，可选值为lazy 和eager ，分别为延迟加载和积极加载，这个配置会覆盖全局的lazyLoadingEnabled 配置。

**新建接口方法**

SysUser selectUserNest(Long id);

**在mapper中配置sql和resultMap**

<select id="selectUserNest" resultMap="userMapAssociation">  
 select \* from sys\_user where id = #{id}  
</select>

<resultMap id="userMapAssociation" type="mybatisChicken.relation.model.SysUser">  
 <id property="id" column="id"/>  
 <result property="userName" column="user\_name"/>  
 <result property="userPassword" column="user\_password"/>  
 <result property="userEmail" column="user\_email"/>  
 <result property="userInfo" column="user\_info"/>  
 <result property="headImg" column="head\_img" jdbcType="BLOB"/>  
 <result property="createTime" column="create\_time" jdbcType="TIMESTAMP"/>  
 <result property="countryId" column="country\_id" />  
 <association property="country" column="{id=country\_id}" select="selectCountry" />  
 </resultMap>

上面的Association元素没有设置country对象的javaType，因为country对象的类型可以由select特性指定的查询确定。

**配置sssociation元素指定的查询selectCountry**，该查询使用user的country\_id作为参数值查询国籍（column="{id=country\_id}"）

<select id="selectCountry" resultMap="countryMap">  
 select id , countryname ,countrycode from country where id=#{id}  
</select>

<resultMap id="countryMap" type="mybatisChicken.relation.model.Country">  
</resultMap>

在id为selectCountry的resultMap中，并没有定义内置的result元素。因为association默认开启了automapping，且Country类型的字段和查询得到列名相同，所以country对象的字段会被自动映射

**新建测试类方法**

testSelectUserNest

为了简化resultMap的配置，减少重复配置映射规则，resultMap使用extends特性继承其他resultMap的配置

前面项目配置的userMap包含 userMapAssociation需要用到的<result>元素，因此使用extends特性继承userMap后userMapAssociation可以简化为

<resultMap id="userMapAssociation" extends="userMap" type="mybatisChicken.relation.model.SysUser">  
 <association property="country" column="{id=country\_id}" select="selectCountry" />  
</resultMap>

**懒加载**

使用嵌套查询时，如果第一次查询返回n条记录，会再次触发n次查询得到完整的结果。这种情况下使用懒加载能大大提高程序的性能。

当使用association元素配置一个嵌套查询时（设置了select特性），那么association元素的property特性指定的对象就是一个关联对象。这个关联对象是否会得到延迟加载会受到3个值的影响。

lazyLoadingEnabled：延迟加载的全局开关。当开启时，所有关联对象都会延迟加载。 默认值false

fetchType ：数据加载方式，可选值为lazy 和eager ，分别为延迟加载和积极加载，这个配置会覆盖全局的lazyLoadingEnabled 配置。

aggressiveLazyLoading：当开启时，含有关联对象的对象上任何方法的调用都会加载该对象的所有关联对象。 否则，每个关联对象按需加载，false （在 3.4.1 及之前的版本默认值为 true）

根据说明，要想关联对象延迟加载，首先需要在association元素上设置fetchType特性的值为lazy来覆盖全局的设置。另外，在将查询列的值映射到一个javabean对象上时，会调用javabean的setter方法，当aggressiveLazyLoading值为true时，javabean上的关联对象会立即加载，因此需设置aggressiveLazyLoading为false

修改上个示例的resultMap

<resultMap id="userMapAssociation" extends="userMap" type="mybatisChicken.relation.model.SysUser">  
 <association property="country" fetchType="lazy" column="{id=country\_id}" select="selectCountry" />  
</resultMap>

修改config文件，设置aggressiveLazyLoading为false

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