MyBatis基础

1.1 Mybatis CRUD

Category属性：

- private int id
- private String name

```
<mapper>
    <!--增加-->
    <insert id = "addCategory" parameterType="Category">
        insert into category_ (name) values (#{name})
    </insert>

    <!--删除-->
    <delete id = "deleteCategory" parameterType="Category">
        delete from category_ where id = #{id}
    </delete>

    <!--获取-->
    <select id="getCategory" parameterType="_int" resultType="Category">
        select * from   category_  where id= #{id}
    </select>

    <!--修改-->
    <update id = "updateCategory" parameter="Category">
        update category_ set name = #{name} where id = #{id}
    </update>

    <!--获取所有-->
    <select id = "listCategory" resultType="Category">
        select * from category_
    </select>
</mapper>
```

```
//增加
Category c = new Category();
c.setName("新增加的Category");
session.insert("addCategory",c);

//删除
Category c = new Category();
c.setId(6);
session.delete("deleteCategory",c);

//获取
Category c= session.selectOne("getCategory",3);

//修改
Category c = session.selectOne("getCategory", 1);
```

```
c.setName("update_name");
session.update("updateCategory", c);

//获取所有
List<Category> cs=session.selectList("listCategory");
```

1.2. 多条件查询

Category属性:

- private int id
- private String name

```
<!--  CategoryMultiCondition.xml  -->
<!--按照name模糊查找-->
<select id="listCategoryByName" parameterType="string" resultType="Category">
    select * from category_ where name like concat('%', #{0}, '%')
</select>


<!--多条件查找 通过map传参数 查找id>#{id}并且name包含#{name}的Category-->
<select id = "listCategoryByIdAndName" parameterType="map" resultType="Category">
    select * from category_ where id > #{id} and name like concat ('%', #{name}, '%')
</select>
```

```
//按照name模糊查找
List<Category> cs = session.selectList("listCategoryByName","z");
for (Category c : cs) {
    System.out.println(c.getName());
}


//多条件查找 通过map传参数 查找id>#{id}并且name包含#{name}的Category]
Map<String, Object> params = new HashMap<String, Object>();
params.put("id", 3);
params.put("name", "cat");
List<Category> cs = session.selectList("listCategoryByIdAndName", params);
```

1.3. MyBatis 一对多 [collection]

Category属性:

- private int id
- private String name
- List products

Product属性:

- private int id;
- private String name;
- private float price;

因为Category和Product都有id所以需要使用cid和pid区分

```
<!-- Category.xml -->

<!--       Mybatis一对多  （一个Category对多个Product）  -->
<resultMap id="categoryBean" type="Category">
    <!--    column对应数据库的列 这里cid是SQL查询后为c.id重命名为cid的列名
            property 对应Category的id属性
    -->
    <id column="cid" property="id"/>
    <result column="cname" property="name"/>

    <!-- 一对多的关系 -->
    <!-- property: 指的是集合属性的值， ofType: 指的是集合中元素的类型 -->
    <collection property="products" ofType="Product">
        <id column="pid" property="id" />
        <result column="pname" property="name"/>
        <result column="price" property="price"/>
    </collection>
</resultMap>

<select id="newListCategory" resultMap="categoryBean">
    select c.*, p.*, c.id 'cid', p.id 'pid', c.name 'cname', p.name 'pname' from
category_ c left join product_ p on c.id = p.cid
</select>
```

```
//测试一对多 TestOneToMany.java
List<Category> cs = session.selectList("newListCategory");
for (Category c : cs) {
    System.out.println(c);
    List<Product> ps = c.getProducts();
    for (Product p : ps) {
        System.out.println("\t"+p);
    }
}
```

运行结果:

Category [id=1, name=category1]
* Product [id=1, name=product a, price=88.88]*
* Product [id=2, name=product b, price=88.88]*
* Product [id=3, name=product c, price=88.88]*
Category [id=2, name=category2]
* Product [id=4, name=product x, price=88.88]*
* Product [id=5, name=product y, price=88.88]*
* Product [id=6, name=product z, price=88.88]*

1.4. MyBatis 多对一 [association]

查询出所有的Product,同时对每个Product,能看出其所对应的Category

```
<resultMap type="Product" id="productBean">
    <id column="pid" property="id"/>
    <result column="pname" property="name"/>
    <result column="price" property="price"/>
```

```
    <!-- 多对一的联系 -->
    <!-- property: 指的是属性名称
            javaType: 指的是属性的类型
            使用association进行多对一关系关联，指定表字段名称与对象属性名称的一一对应关系
         -->
    <association property="category" javaType="Category">
        <id column="cid" property="id"/>
        <result column="cname" property="name"/>
    </association>
</resultMap>
<!-- 根据id查询Product，关联Category -->
<select id = "listProduct" resultMap="productBean">
    select c.*, p.*, c.id 'cid', p.id 'pid', c.name 'cname', p.name 'pname' from
category_ c left join product_ p on c.id = p.cid
</select>
```

```
List<Category> cs = session.selectList("newListCategory");
for (Category c : cs) {
    System.out.println(c);
    List<Product> ps = c.getProducts();
    for (Product p : ps) {
        System.out.println("\t"+p);
    }
}
```

输出结果:

Product [id=1, name=product a, price=88.88] 对应的分类是 Category [id=1, name=category1]
Product [id=2, name=product b, price=88.88] 对应的分类是 Category [id=1, name=category1]
Product [id=3, name=product c, price=88.88] 对应的分类是 Category [id=1, name=category1]
Product [id=4, name=product x, price=88.88] 对应的分类是 Category [id=2, name=category2]
Product [id=5, name=product y, price=88.88] 对应的分类是 Category [id=2, name=category2]
Product [id=6, name=product z, price=88.88] 对应的分类是 Category [id=2, name=category2]

1.5. Mybatis 多对多

订单Order和产品Product的关系为例:

- 一张订单可以包含多种产品
- 一种产品可以出现在多张订单中

多对多关系需要一个中间表，我们使用订单项OrderItem表作为中间表

Order属性:

- private int id
- private String code
- List orderItems

OrderItem属性:

- private int id
- private int number
- private **Order** order
- private **Product** product

查询出所有的订单，然后遍历每个订单下的多余订单项，以及订单项对应的产品名称，价格，购买数量

```
<resultMap type="Order" id="orderBean">
    <id column="oid" property="id" />
    <result column="code" property="code" />

    <collection property="orderItems" ofType="OrderItem">
        <id column="oiid" property="id" />
        <result column="number" property="number" />
        <association property="product" javaType="Product">
            <id column="pid" property="id"/>
            <result column="pname" property="name"/>
            <result column="price" property="price"/>
        </association>
    </collection>
</resultMap>
<select id="listOrder" resultMap="orderBean">
    select o.*,p.*,oi.*, o.id 'oid', p.id 'pid', oi.id 'oiid', p.name 'pname'
        from order_ o
        left join order_item_ oi    on o.id =oi.oid
        left join product_ p on p.id = oi.pid
</select>
```

```
List<Order> os = session.selectList("listOrder");
for(Order o: os){
    System.out.println(o.getCode());
    List<OrderItem> ois = o.getOrderItems();
    for(OrderItem oi: ois){
        System.out.format("\t%s\t%f\t%d%n",
oi.getProduct().getName(),oi.getProduct().getPrice(),oi.getNumber());
    }
}
```

1.6. Mybatis多条语句

eg. 删除订单A,那么就应该删除订单A在订单项里所对应的数据

通过Mybatis执行多条sql语句需要增加一个参数 ***allowMultiQueries***

```
<property name="url" value="jdbc:mysql://localhost:3306/how2java?
characterEncoding=UTF-8&amp;allowMultiQueries=true"/>
```

Order.xml：

```
<delete id="deleteOrder">delete from order_item_ where oid = #{id};delete from order_
where id= #{id}; </delete>
```

动态SQL

2.1. Mybatis if 【单条件】

```
<!--动态SQL： Mybatis If -->
<!-- 如果没有传参数name,那么就查询所有，如果有name参数，那么就进行模糊查询。 -->
<select id="listProduct_if" resultType="Product">
    select * from product_
    <if test="name!=null">
        where name like contat('%',#{name},'%')
    </if>
</select>
```

```
System.out.println("查询所有的");
List<Product> ps = session.selectList("listProduct");
for (Product p : ps) {
    System.out.println(p);
}

System.out.println("模糊查询");
Map<String, Object> params = new HashMap<String, Object>();
params.put("name","a");
List<Product> ps2 = session.selectList("listProduct",params);
for (Product p : ps2) {
    System.out.pr#intln(p);
}
```

2.2. Mybatis where 【多条件】

where标签

多条件

```
<select id="listProduct" resultType="Product">
    select * from product_
    <where>
        <if test="name!=null">
            and name like concat('%',#{name},'%')
        </if>
        <if test="price!=null and price!=0">
            and price > #{price}
        </if>
    </where>
</select>
```

注：标签会进行自动判断
如果任何条件都不成立，那么就在sql语句里就不会出现where关键字
如果有任何条件成立，会自动去掉多出来的 and 或者 or。

没条标签中加**and**

```
System.out.println("多条件查询");
Map<String,Object> params = new HashMap<>();
//        params.put("name","a");
params.put("price","10");
List<Product> ps2 = session.selectList("listProduct",params);
for (Product p : ps2) {
    System.out.println(p);
}
```

set标签 【在update语句中】

```
<update id="updateProduct" parameterType="Product" >
    update product_
    <set>
        <if test="name != null">name=#{name},</if>
        <if test="price != null">price=#{price}</if>

    </set>

    where id=#{id}
</update>
```

```
Product p = new Product();
p.setId(6);
p.setName("product zz");
p.setPrice(99.99f);
session.update("updateProduct",p);
```

trim标签

trim 用来定制想要的功能，比如where标签就可以用以下替换

```
<trim prefix="WHERE" prefixOverrides="AND |OR ">
  ...
</trim>
```

set标签就可以用以下替换

```
<trim prefix="SET" suffixOverrides=",">
  ...
</trim>
```

2.3. Mybatis choose 【if else】

Mybatis里面没有else标签，但是可以使用when otherwise标签来达到这样的效果。

```
<select id="listProduct" resultType="Product">
      SELECT * FROM product_
      <where>
        <choose>
          <when test="name != null">
            and name like concat('%',#{name},'%')
          </when>
          <when test="price !=null and price != 0">
```

```
            and price > #{price}
          </when>
          <otherwise>
            and id >1
          </otherwise>
        </choose>
      </where>
</select>
```

```
Map<String,Object> params = new HashMap<>();
//      params.put("name","a");
//      params.put("price","10");
List<Product> ps = session.selectList("listProduct",params);
for (Product p : ps) {
    System.out.println(p);
}
```

2.4. Mybatis foreach

```
<!-- mybatis foreach -->
<select id="listProduct_foreach" resultType="Product">
    SELECT * FROM product_ WHERE id in
    <foreach collection="list" item="item" index="index" open="(" separator="," 
close=")">
        #{item}
    </foreach>
</select>
```

```
List<Integer> ids = new ArrayList();
ids.add(1);
ids.add(3);
ids.add(5);

List<Product> ps = session.selectList("listProduct_foreach", ids);
```

2.5. Mybatis bind

bind标签就像是再做一次字符串拼接，方便后续使用
如本例，在模糊查询的基础上，把模糊查询改为bind标签。

```
<!-- 本来的模糊查询方式 -->
<!--         <select id="listProduct" resultType="Product"> -->
<!--             select * from   product_  where name like concat('%',#{0},'%') -->
<!--         </select> -->

<select id="listProduct_bind" resultType="Product">
    <bind name="likename" value="'%' + name + '%'" />
    select * from   product_  where name like #{likename}
</select>
```

```
Map<String, String> params =new HashMap();
params.put("name", "product");

List<Product> ps = session.selectList("listProduct_bind",params);
for (Product p : ps) {
    System.out.println(p);
}
```

注解

3.1. Mybatis CRUD 注解

对比配置文件xxx.xml，其实就是把SQL语句从XML挪到了注解上来

```
<!-- mybatis-config.xml -->
<mappers>
    <!--        <mapper resource="com/how2java/pojo/Category.xml"/>-->
    <mapper class="com.how2java.mapper.CategoryMapper"/>
</mappers>
```

Category类属性：

```
private int id;
private String name;
```

注解：**CategoryMapper.java**

```
public interface CategoryMapper {
    //增加Category
    @Insert(" insert into category_ ( name ) values (#{name}) ")
    public int add(Category category);

    //删除Category
    @Delete(" delete from category_ where id= #{id} ")
    public void delete(int id);

    //按照Id查询Category
    @Select("select * from category_ where id= #{id} ")
    public Category get(int id);

    //修改Category
    @Update("update category_ set name=#{name} where id=#{id} ")
    public int update(Category category);

    //查询所有Category
    @Select(" select * from category_ ")
    public List<Category> list();
}
```

测试：

```
public class TestCRUD {
```

```
    public static void main(String[] args) throws IOException {
        String resource = "mybatis-config.xml";
        InputStream inputStream = Resources.getResourceAsStream(resource);
        SqlSessionFactory sqlSessionFactory = new
SqlSessionFactoryBuilder().build(inputStream);
        SqlSession session = sqlSessionFactory.openSession();
        CategoryMapper mapper = session.getMapper(CategoryMapper.class);

//        add(mapper);
//        delete(mapper);
//        get(mapper);
//        update(mapper);
        listAll(mapper);

        session.commit();
        session.close();

    }

    private static void update(CategoryMapper mapper) {
        Category c= mapper.get(8);
        c.setName("修改了的Category名稱");
        mapper.update(c);
        listAll(mapper);
    }

    private static void get(CategoryMapper mapper) {
        Category c= mapper.get(8);
        System.out.println(c.getName());
    }

    private static void delete(CategoryMapper mapper) {
        mapper.delete(2);
        listAll(mapper);
    }

    private static void add(CategoryMapper mapper) {
        Category c = new Category();
        c.setName("新增加的Category");
        mapper.add(c);
        listAll(mapper);
    }

    private static void listAll(CategoryMapper mapper) {
        List<Category> cs = mapper.list();
        for (Category c : cs) {
            System.out.println(c.getName());
        }
    }
}
```

3.2 Mybatis 一对多 注解

```
//CategoryMapper
@Select(" select * from category_ ")
@Results({
    @Result(property = "id", column = "id"),
    @Result(property = "products", javaType = List.class, column = "id", many =
@Many(select = "com.how2java.mapper.ProductMapper.listByCategory"))
})
public List<Category> listAll();
```

```
//按照cid查询Product
@Select(" select * from product_ where cid = #{cid}")
public List<Product> listByCategory(int cid);
```

```
//TestOneToMany.java
List<Category> cs = mapper.listAll();
```

3.3 Mybatis 多对一 注解

```
//CategoryMapper
@Select("select * from category_ where id= #{id} ")
public Category get(int id);
```

```
//ProductMapper
@Select(" select * from product_")
@Results({
    @Result(property = "category", column = "cid", one=@One(select =
"com.how2java.mapper.CategoryMapper.get"))
})
public List<Product> listAll();
```

```
//TestManyToOne
List<Product> ps= mapper.listAll();
```

3.4 Mybatis 多对多 注解

```
//ProductMapper
@Select("select * from product_ where id = #{id}")
public Product get(int id);
```

```
//OrderItemMapper
@Select(" select * from order_item_ where oid = #{oid}")
@Results({

 @Result(property="product",column="pid",one=@One(select="com.how2java.mapper.ProductM
apper.get"))
})
public List<OrderItem> listByOrder(int oid);
```

```
//OrderMapper
@Select("select * from order_")
@Results({
    @Result(property = "id", column = "id"),
    @Result(property = "orderItems", javaType = List.class, column = "id",
            many = @Many(select = "com.how2java.mapper.OrderItemMapper.listByOrder"))
})
public List<Order> list();
```

3.5 动态SQL

SQL语句使用**SQL类**的方式构建

```
package com.how2java;

import org.apache.ibatis.jdbc.SQL;

public class CategoryDynaSqlProvider {
    public String list() {
         return new SQL()
                 .SELECT("*")
                 .FROM("category_")
                 .toString();

    }
    public String get() {
        return new SQL()
                .SELECT("*")
                .FROM("category_")
                .WHERE("id=#{id}")
                .toString();
    }

    public String add(){
        return new SQL()
                .INSERT_INTO("category_")
                .VALUES("name", "#{name}")
                .toString();
    }
    public String update(){
        return new SQL()
                .UPDATE("category_")
                .SET("name=#{name}")
                .WHERE("id=#{id}")
                .toString();
    }
    public String delete(){
        return new SQL()
                .DELETE_FROM("category_")
                .WHERE("id=#{id}")
                .toString();
    }

}
```

```java
//CategoryMapper修改为注解引用CategoryDynaSqlProvider类的方式
@InsertProvider(type=CategoryDynaSqlProvider.class,method="add")
public int add(Category category);

@DeleteProvider(type=CategoryDynaSqlProvider.class,method="delete")
public void delete(int id);

@SelectProvider(type=CategoryDynaSqlProvider.class,method="get")
public Category get(int id);

@UpdateProvider(type=CategoryDynaSqlProvider.class,method="update")
public int update(Category category);

@SelectProvider(type=CategoryDynaSqlProvider.class,method="list")
public List<Category> list();
```