CH01_MyBatis框架概述

UserMappper.java

```java
//映射器接口：定义对持久化对象的增删改查操作的抽象方法
public interface UserMapper {

    List<User> findAllUsers();
}
```

UserMapper.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper
    PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
    "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
    <!-- namespace为唯一标识，映射器接口的全限定名 -->
<mapper namespace="com.mybatis.mapper.UserMapper">

    <!-- select用来映射查询语句
        id属性同映射器接口的某个方法名称相同
        resultType同映射接口中该方法的返回值类型一致，或跟返回值中元素类型一致 -->
    <select id="findAllUsers" resultType="com.mybatis.entity.User">
        select * from user
    </select>

    <select id=""></select>
</mapper>
```

CH02_MyBatis的CRUD操作

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper
    PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
    "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
    <!-- namespace为唯一标识，映射器接口的全限定名 -->
<mapper namespace="com.mybatis.mapper.UserMapper">

    <!--
    如果User类的属性和数据库字段值不一致，两种方式
    1.结果映射：指定数据库表字段和实体类属性之间的映射关系 ，用的更多，因为它有更多用途
    2.为字段名设置别名为属性名
    -->
    <resultMap type="com.mybatis.entity.User" id="userMap">
        <!-- 指定查询语句执行成功以后，查询结果中某个字段的值，赋值给对象的哪一个属性 -->
```

```
        <!-- 映射主键 -->
        <id column="id" property="id"/>
        <!-- 映射非主键字段 -->
        <result column="user_name" property="userName"/>
        <result column="password" property="password"/>
    </resultMap>

    <!-- select用来映射查询语句
        id属性同映射器接口的某个方法名称相同
        resultType同映射接口中该方法的返回值类型一致，或跟返回值中元素类型一致 -->
    <select id="findAllUsers" resultType="com.mybatis.entity.User">
        select id, user_name userName, password from user
    </select>

    <!-- insert映射插入语句
        parameterType指定方法的参数类型:可选
        SQL语句传参使用#{User类型的属性名}
        如果方法参数是某个实体类类型，那么{}中是该实体中定义的某个属性的名字
        useGeneratedKeys="true"自动递增
        keyProperty="id"表示要将插入以后记录主键字段的值，赋值给对象的id属性-->
    <insert id="insertUser" parameterType="com.mybatis.entity.User" useGeneratedKeys="true"
        keyProperty="id">
        insert into user(id,username,password)
        values(#{id},#{userName},#{password})
    </insert>

    <!-- 对于不支持自动递增字段的数据库，如Oracle
        resultType
        keyProperty
        order="AFTER"真正执行插入之后，才获得值，如果用Oracle，设为"BEFORE"
     -->
    <insert id="insertUser1" >
        insert into user(id,username,password)
        values(#{id},#{userName},#{password})
        <selectKey resultType="int" keyProperty="id" order="AFTER">
            select last_insert_id()
        </selectKey>
    </insert>

    <!-- select元素中指定resultType（找到与字段同名的属性名，赋值）或者 resultMap(用于字段和属性不一
致)-->
    <select id="findUserById" resultMap="userMap">
        select * from user where id = #{id}
    </select>

    <!-- 模糊查询 -->
    <select id="findLike" resultMap="userMap">
        select * from user where user_name like "%"#{name}"%"
    </select>

    <update id="updateUser">
        update user set
        user_name=#{userName},
        password=#{password}
        where
        id=#{id}
    </update>

    <delete id="deleteUser">
        delete from user where id=#{id}
    </delete>
```

```
</mapper>
```

CH03_MyBatis关联映射

一对一

ShoopingCarMapper.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper
    PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
    "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
    <!-- namespace为唯一标识，映射器接口的全限定名 -->
<mapper namespace="com.mybatis.mapper.ShoppingCartMapper">

    <!-- 定义结果映射 -->
    <resultMap type="com.mybatis.entity.ShoppingCart" id="cartMap">
        <id column="cart_id" property="id"/>
        <result column="price" property="price"/>
    </resultMap>

    <!-- 定一个查询，但方法中没有抽象方法，这是可以的
        要传一个参数，为user_id
    -->
    <select id="findShoppingCartByUserId" resultMap="cartMap">
        select *
        from shopping_cart
        where user_id = #{user_id}
    </select>

    <insert id="insert">
        insert into shopping_cart(cart_id,price,user_id) values(#{id},#{price},#{user.id})
    </insert>
</mapper>
```

UserMapper.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper
    PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
    "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
    <!-- namespace为唯一标识，映射器接口的全限定名 -->
<mapper namespace="com.mybatis.mapper.UserMapper">

    <!-- 结果映射：指定数据库表字段和实体类属性之间的映射关系 -->
    <resultMap type="com.mybatis.entity.User" id="userMap">
        <!-- 指定查询语句执行成功以后，查询结果中某个字段的值，赋值给对象的哪一个属性 -->
        <!-- 映射主键 -->
        <id column="id" property="id"/>
        <!-- 映射非主键字段 -->
```

```
        <result column="user_name" property="userName"/>
        <result column="password" property="password"/>
    </resultMap>

    <!-- select用来映射查询语句
        id属性同映射器接口的某个方法名称相同
        resultType同映射接口中该方法的返回值类型一致，或跟返回值中元素类型一致 -->
    <select id="findAllUsers" resultType="com.mybatis.entity.User">
        select id, user_name userName, password from user
    </select>

    <!-- insert映射插入语句
        parameterType指定方法的参数类型:可选
        SQL语句传参使用#{}
        如果方法参数是某个实体类类型，那么{}中是该实体中定义的某个属性的名字
        keyProperty="id"表示要将插入以后记录主键字段的值，赋值给对象的id属性-->
    <insert id="insertUser" parameterType="com.mybatis.entity.User" useGeneratedKeys="true"
        keyProperty="id">
        insert into user(id,username,password)
        values(#{id},#{userName},#{password})
    </insert>

    <insert id="insertUser1" >
        insert into user(id,username,password)
        values(#{id},#{userName},#{password})
        <selectKey resultType="int" keyProperty="id" order="AFTER">
            select last_insert_id()
        </selectKey>
    </insert>

    <!-- select元素中指定resultType或者 resultMap-->
    <select id="findUserById" resultMap="userMap">
        select * from user where id = #{id}
    </select>

    <!-- 模糊查询 -->
    <select id="findLike" resultMap="userMap">
        select * from user where user_name like "%"#{name}"%"
    </select>

    <update id="updateUser">
        update user set
        user_name=#{userName},
        password=#{password}
        where
        id=#{id}
    </update>

    <delete id="deleteUser">
        delete from user where id=#{id}
    </delete>


    <!-- 方法一：自动映射（给查询结果的字段起别名）
        shopping_cart表的字段不能直接写别名，要不就认为这个字段是
resultType="com.mybatis.entity.User"所制定的User类型的某个属性
        需要用User类中定义的shopping属性的类中定义的id和price属性，所以别名只能是User类中定义的属性
        方法只有一个参数为简单类型：where u.id=#{id}中的id只是一个占位符，不一定要与
findUserAndShoppingCartById(Integer id)的属性id名字一样
        方法有一个User类属性，#{}中一定是User类的属性名，是固定的，不能随意写
     -->
```

```
    <!-- <select id="findUserAndShoppingCartById" resultType="com.mybatis.entity.User">
        select u.id, u.user_name userName, u.password,
        s.cart_id "shoppingCart.id",
        s.price "shoppingCart.price"
        from user u
        left outer join shopping_cart s
        on u.id = s.user_id
        where u.id=#{id}
    </select> -->

    <!-- 方法二：使用resultMap完成简单地结果映射 -->
    <!-- <resultMap type="com.mybatis.entity.User" id="userMap1">
        <id column="id" property="id"/>
        <result column="user_name" property="userName"/>
        <result column="password" property="password"/>
        <result column="cart_id" property="shoppingCart.id"/>
        <result column="price" property="shoppingCart.price"/>
    </resultMap>

    <select id="findUserAndShoppingCartById" resultMap="userMap1">
        select u.id,u.user_name,u.password,s.cart_id,s.price
        from user u
        left outer join shopping_cart s
        on u.id = s.user_id
        where u.id = #{id}
    </select> -->

    <!-- 方式三：使用resultMap中association子元素,嵌套的结果映射(resultMap嵌套association(属性对象的结果映射))，简化了shoppingCart.id"，不用.了
        javaType：可选
        <association>中的属性映射，可以放到ShoppingCartMapping.xml中
    -->
    <!-- <resultMap type="com.mybatis.entity.User" id="userMap2">
        <id column="id" property="id"></id>
        <result column="user_name" property="userName"/>
        <result column="password" property="password"/>
        <association property="shoppingCart" javaType="com.mybatis.entity.ShoppingCart">
            <id column="cart_id" property="id"></id>
            <result column="price" property="price"/>
        </association>
    </resultMap>


    <select id="findUserAndShoppingCartById" resultMap="userMap2">
        select u.id,u.user_name,u.password,s.cart_id,s.price
        from user u
        left outer join shopping_cart s
        on u.id = s.user_id
        where u.id = #{id}
    </select> -->

    <!-- 修改，引用属性对象的结果映射，使用
resultMap="com.mybatis.mapper.ShoppingCartMapper.cartMap" -->
    <resultMap type="com.mybatis.entity.User" id="userMap2">
        <id column="id" property="id"></id>
        <result column="user_name" property="userName"/>
        <result column="password" property="password"/>
        <!-- javaType指定映射的属性类型
            property指定映射的属性名称
            resultMap指定了嵌套的resultMap（如果resultMap定义在其他映射文件中namespace+点号+id的值）
-->
```

```
        <association property="shoppingCart"
            javaType="com.mybatis.entity.ShoppingCart"
            resultMap="com.mybatis.mapper.ShoppingCartMapper.cartMap">
        </association>
    </resultMap>

    <select id="findUserAndShoppingCartById" resultMap="userMap2">
        select u.id,u.user_name,u.password,s.cart_id,s.price
        from user u
        left outer join shopping_cart s
        on u.id = s.user_id
        where u.id = #{id}
    </select>

    <!-- 方法四：使用resultMap中的association资源嵌套查询语句 :不需要使用SQL连接查询，容易产生N+1次查
询
        column:将主查询列的结果，作为嵌套查询的参数，指定findShoppingCartByUserId查询语句的参数
user_id
        多个参数用,分割
        select=""指定一个嵌套的查询语句:某个查询语句映射文件的namespace+点号+映射时select元素中id的值
        column将主查询查询结果中某个字段的值，赋值给嵌套查询的参数
        1.查询user表，结果映射为userMap4
        2.根据resultMap，为属性赋值，找到嵌套的查询语句，执行
        3.得到shopping_cart表的结果，结果映射为cartMap，为shoppingCart属性的各个属性赋值
        -->
     <resultMap type="com.mybatis.entity.User" id="userMap4">
        <id column="id" property="id"></id>
        <result column="user_name" property="userName"/>
        <result column="password" property="password"/>
        <association property="shoppingCart"
            select="com.mybatis.mapper.ShoppingCartMapper.findShoppingCartByUserId"
            column="{user_id=id}"></association>
    </resultMap>
    <!-- 只需要查询当前用户表中的字段 -->
    <select id="findUserAndShoppingCartById" resultMap="userMap4">
        select * from user where id=#{id}
    </select>

    <!--避免 N+1次查询问题fetchType="lazy"，主查询有N条数据，第一次：关联查询N条数据，后面N次子查询 --
>
    <resultMap type="com.mybatis.entity.User" id="userMap4">
        <id column="id" property="id"></id>
        <result column="user_name" property="userName"/>
        <result column="password" property="password"/>
        <!-- select=""指定一个嵌套的查询语句:某个查询语句映射文件的namespace+点号+映射时select元素中
id的值
            column将主查询查询结果中某个字段的值，赋值给嵌套查询的参数
            fetchType="lazy"使用延迟加载
            fetchType="eager"不使用延迟加载 -->
        <association property="shoppingCart"
            select="com.mybatis.mapper.ShoppingCartMapper.findShoppingCartByUserId"
            column="{user_id=id}"
            fetchType="lazy"></association>
    </resultMap>
    <!-- 只需要查询当前用户表中的字段 -->
    <select id="findUserAndShoppingCartByName" resultMap="userMap4">
        select * from user where user_name=#{name}
    </select>
</mapper>
```

一对多

OrderMapper.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper
    PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
    "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
    <!-- namespace为唯一标识，映射器接口的全限定名 -->
<mapper namespace="com.mybatis.mapper.OrderMapper">

    <resultMap type="com.mybatis.entity.Order"
        id="orderMap">
        <id column="order_id" property="id"></id>
        <result column="price" property="price"/>
    </resultMap>

    <select id="findOrders" resultMap="orderMap">
        select order_id,price
        from orders
        where user_id = #{uId}
    </select>
</mapper>
```

UserMapper.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper
    PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
    "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
    <!-- namespace为唯一标识，映射器接口的全限定名 -->
<mapper namespace="com.mybatis.mapper.UserMapper">

    <!-- 结果映射：指定数据库表字段和实体类属性之间的映射关系 -->
    <resultMap type="com.mybatis.entity.User" id="userMap">
        <!-- 指定查询语句执行成功以后，查询结果中某个字段的值，赋值给对象的哪一个属性 -->
        <!-- 映射主键 -->
        <id column="id" property="id"/>
        <!-- 映射非主键字段 -->
        <result column="user_name" property="userName"/>
        <result column="password" property="password"/>
    </resultMap>

    <!-- select用来映射查询语句
        id属性同映射器接口的某个方法名称相同
        resultType同映射接口中该方法的返回值类型一致，或跟返回值中元素类型一致 -->
    <select id="findAllUsers" resultType="com.mybatis.entity.User">
        select id, user_name userName, password from user
    </select>

    <!-- insert映射插入语句
        parameterType指定方法的参数类型:可选
        SQL语句传参使用#{}
```

```
    如果方法参数是某个实体类类型，那么{}中是该实体中定义的某个属性的名字
    keyProperty="id"表示要将插入以后记录主键字段的值，赋值给对象的id属性-->
<insert id="insertUser" parameterType="com.mybatis.entity.User" useGeneratedKeys="true"
    keyProperty="id">
    insert into user(id,username,password)
    values(#{id},#{userName},#{password})
</insert>

<insert id="insertUser1" >
    insert into user(id,username,password)
    values(#{id},#{userName},#{password})
    <selectKey resultType="int" keyProperty="id" order="AFTER">
        select last_insert_id()
    </selectKey>
</insert>

<!-- select元素中指定resultType或者 resultMap-->
<select id="findUserById" resultMap="userMap">
    select * from user where id = #{id}
</select>

<!-- 模糊查询 -->
<select id="findLike" resultMap="userMap">
    select * from user where user_name like "%"#{name}"%"
</select>

<update id="updateUser">
    update user set
    user_name=#{userName},
    password=#{password}
    where
    id=#{id}
</update>

<delete id="deleteUser">
    delete from user where id=#{id}
</delete>

<!-- 集合（List、Set）的映射 -->

<!-- 方式一：嵌套的resultMap  extends属性能够实现resultMap之间的继承-->
<resultMap type="com.mybatis.entity.User" id="userMap1"
    extends="userMap">
    映射集合类型的属性，ofType指定集合中元素的类型
    <collection property="orders"
        ofType="com.mybatis.entity.Order"
        resultMap="com.mybatis.mapper.OrderMapper.orderMap">
    </collection>
</resultMap>
<!--左外连接查询-->
<select id="findUserAndOrderListById" resultMap="userMap1">
    select u.id,u.user_name,u.password,o.order_id,o.price
    from user u
    left outer join orders o
    on o.user_id = u.id
    where u.id=#{n}
</select>

<!-- 方式二：嵌套查询语句 -->
<resultMap type="com.mybatis.entity.User" id="userMap1"
    extends="userMap">
```

```
        <!-- 映射集合类型的属性，ofType指定集合中元素的类型 -->
        <collection property="orders"
            ofType="com.mybatis.entity.Order"
            select="com.mybatis.mapper.OrderMapper.findOrders"
            column="{uId=id}"
            fetchType="lazy">

        </collection>
    </resultMap>

    <select id="findUserAndOrderListById" resultMap="userMap1">
        select *
        from user u
        where u.id=#{n}
    </select>
</mapper>
```

多对多

继承映射

CH04_动态SQL

UserMapper.java

```
package com.mybatis.mapper;

import java.util.List;
import java.util.Map;

import org.apache.ibatis.annotations.Param;

import com.mybatis.entity.User;

//映射器接口：定义对持久化对象的增删改查操作的抽象方法
public interface UserMapper {

    List<User> findAllUsers();

    //方法中有多个参数，需要使用@Param注解指定SQL参数名称
    User findUserByNameAndPassword(@Param("username")String name,
            @Param("psw")String password);

    //方法参数是Map类型，map对象中的key的值就是SQL参数名称
    User findUserByMap(Map<String, Object> map);

    List<User> findUserByNameOrAge(@Param("name")String name,
```

```
            @Param("age")int age);

    int updateUserById(User u);

    int insertUser(User u);

    List<User> findUserByIds(Integer[] ids);

    //批量插入
    int insertUsers(List<User> u);

    int updateUserByMap(Map<String, Object> map);

}
```

UserMapper.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper
    PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
    "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
    <!-- namespace为唯一标识，映射器接口的全限定名 -->
<mapper namespace="com.mybatis.mapper.UserMapper">

    <!-- select用来映射查询语句
        id属性同映射器接口的某个方法名称相同
        resultType同映射接口中该方法的返回值类型一致，或跟返回值中元素类型一致 -->
    <select id="findAllUsers" resultType="com.mybatis.entity.User">
        select * from user
    </select>

    <resultMap type="com.mybatis.entity.User" id="userMap">
        <id column="id" property="id"></id>
        <result column="user_name" property="userName"/>
        <result column="password" property="password"/>
        <result column="age" property="age"/>
    </resultMap>

    <select id="findUserByNameAndPassword" resultMap="userMap">
        select * from user where user_name =#{username} and password=#{psw}
    </select>

    <select id="findUserByMap" resultMap="userMap">
        select * from user where user_name =#{username} and password=#{psw}
    </select>

    <!-- 使用if实现动态查询（where子句中使用if元素） -->
     <select id="findUserByNameOrAge" resultMap="userMap">
        select * from user
        where 1=1
        <if test="name != null and !name.equals('')">
            and user_name=#{name}
        </if>
        <if test="age neq 0">
            and age=#{age}
        </if>
    </select>
```

```
<!-- if+where -->
 <select id="findUserByNameOrAge" resultMap="userMap">
    select * from user
    <where>
        <if test="name != null and !name.equals('')">
            and user_name=#{name}
        </if>
        <if test="age neq 0">
            and age=#{age}
        </if>
    </where>
</select>

<!-- 使用if使用动态的列更新 -->
<update id="updateUserById">
    update user set
    <if test="userName != null and !userName.equals('')">
        user_name=#{userName},
    </if>
    <if test="password != null and !password.equals('')">
        password=#{password},
    </if>
    <if test="age != 0">
        age=#{age},
    </if>
    id = #{id}
    where id = #{id}
</update>

<!-- if+set -->
<update id="updateUserById">
    update user
    <set>
        <if test="userName != null and !userName.equals('')">
            user_name=#{userName},
        </if>
        <if test="password != null and !password.equals('')">
            password=#{password},
        </if>
        <if test="age != 0">
            age=#{age},
        </if>
    </set>
    where id = #{id}
</update>


<!-- 使用trim实现set元素的功能 -->
<update id="updateUserById">
    update user
    <trim prefix="set" suffixOverrides=",">
        <if test="userName != null and !userName.equals('')">
            user_name=#{userName},
        </if>
        <if test="password != null and !password.equals('')">
            password=#{password},
        </if>
        <if test="age != 0">
            age=#{age},
        </if>
```

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


<!-- 使用if实现动态的列插入 -->
<insert id="insertUser">
    insert into user(user_name,password
    <if test="age != 0">
        ,age
    </if>) values(#{userName},#{password}
    <if test="age != 0">
        ,#{age}
    </if>)
</insert>


<!-- when+otherwise实现动态查询 ， 类似if。。。else-->
<select id="findUserByNameOrAge" resultMap="userMap">
    select * from user
    <where>
        <choose>
            <when test="name != null and !name.equals('')">
                and user_name=#{name}
            </when>
            <when test="age neq 0">
                and age=#{age}
            </when>
            <otherwise>
                and age > 18
            </otherwise>
        </choose>
    </where>
</select>

<!-- 使用trim替代where元素 -->
<select id="findUserByNameOrAge" resultMap="userMap">
    select * from user
    <trim prefix="where" prefixOverrides="and|or">
        <choose>
            <when test="name != null and !name.equals('')">
                and user_name=#{name}
            </when>
            <when test="age neq 0">
                and age=#{age}
            </when>
            <otherwise>
                and age > 18
            </otherwise>
        </choose>
    </trim>
</select>

<!-- 使用foreach遍历数组类型 -->
<select id="findUserByIds" resultMap="userMap">
    select * from user
    where id in
    <foreach collection="array" item="n"
        open="(" close=")" separator=",">
        #{n}
    </foreach>
```

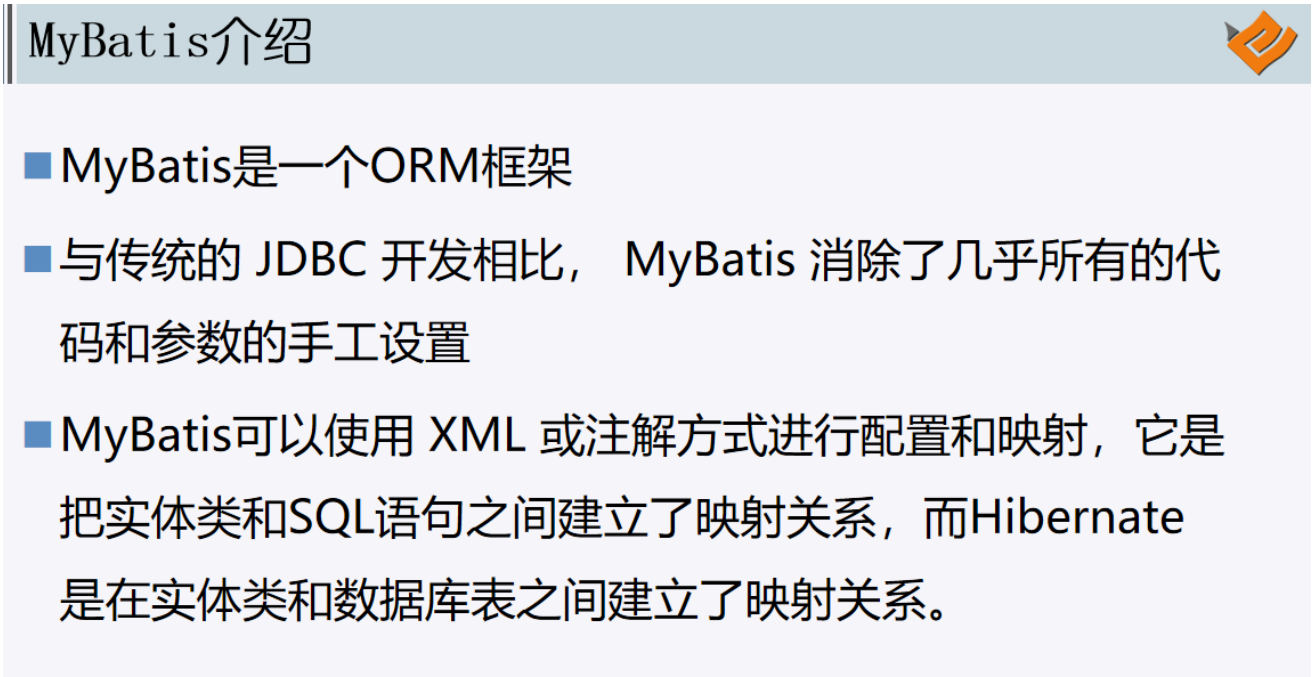
```
    </select>

    <!-- foreach实现批量插入 -->
    <insert id="insertUsers">
        insert into user(user_name,password,age)
        values
        <foreach collection="list" item="u" separator=",">
            (#{u.userName},#{u.password},#{u.age})
        </foreach>
    </insert>

    <!-- foreach实现动态列的更新（根据Map类型参数中key的值决定更新字段） -->
    <update id="updateUserByMap">
        update user set
        <foreach collection="_parameter" index="k" item="v"
            separator=",">
            ${k} = #{v}
        </foreach>
        where id = #{id}
    </update>
</mapper>
```

简答

对MyBatis的理解



传参方式

映射器接口中方法的参数情况

- 当根据单个条件查询时，可以直接以该条件为参数
- 当根据多个条件查询时，可以将JavaBean作为参数
- 当根据多个条件查询且多个条件不属于某一个JavaBean时，可以Map类型作为参数，且通过Map中的key值来映射XML中SQL使用的参数的名字
- 如果要使用多个参数，必须使用@Param注解指定参数名

#{}与${}的异同

MyBatis中$和#的异同

- 可以获取对象中的属性值，${userName}和#{userName}相同
- #可以防止SQL注入，解析时会把所有使用#的地方变成？占位符，再设置参数的值
- $在解析时，会直接使用传入的参数作为字符串直接填充到SQL语句中，会导致SQL注入
- #会把传入的参数使用引号括起来，$则不会
- $一般用传入数据库相关参数，如数据库表名、字段名