一 MyBatis 概述

1 概述

MyBatis 是最近几年非常流行的数据访问层(DAO)框架，能够简单高效的实现对数据层访问。

2 常见数据访问层方式比较

2.A JDBC

java 原生的关系型数据库访问方式

- 每次操作数据库都需要获取连接关闭连接，在大量访问数据库时，频繁的开关连接消耗性能。
- 需要手动编写 sql，有学习成本
- 查询出的结果需要手动进行封装到 bean
- 没有缓存处理机制
- sql 语句写死在程序中，需要修改 sql 必须修改源文件

2.B Hibernate

基于面向对象理念设计的 DAO 层框架，基本理念就是维护对象到表的映射关系，通过操作对象操作表中的数据，从而可以减少甚至杜绝 sql 的使用

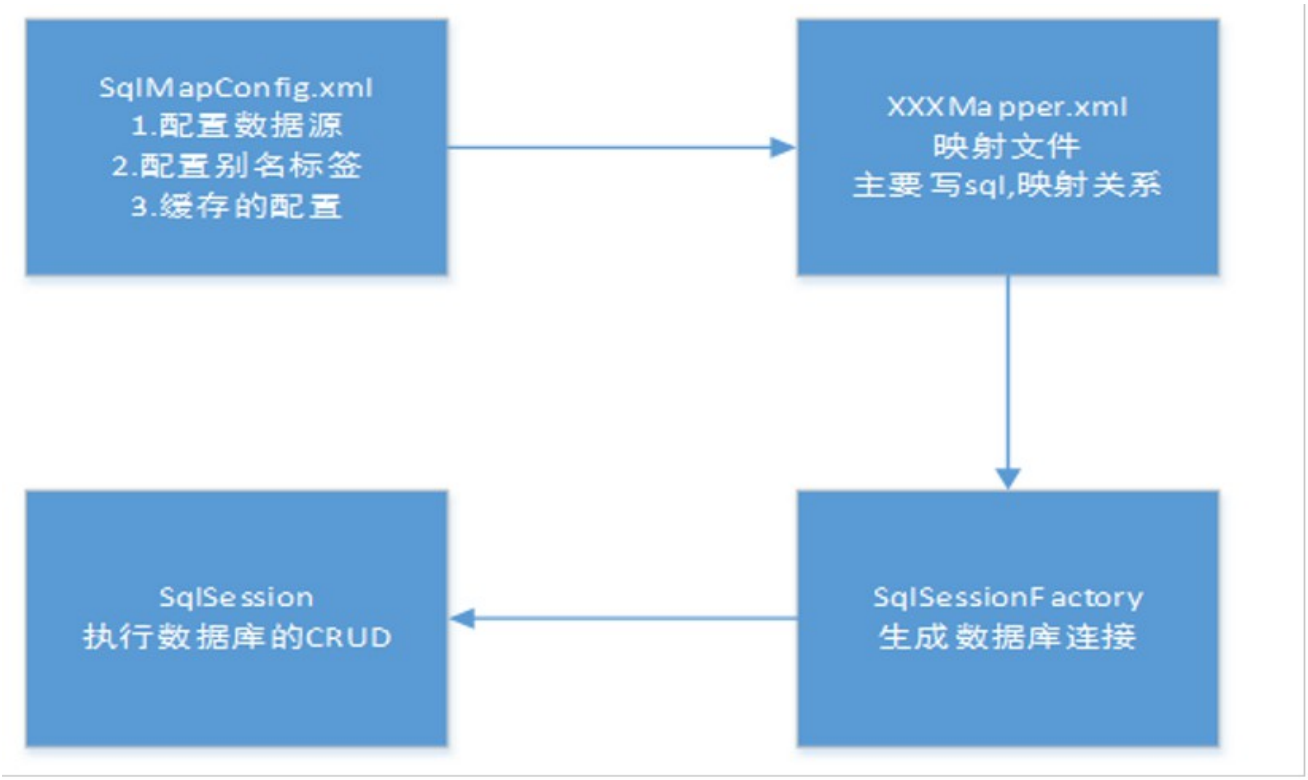
- 相对比较沉重，效率不好
- 当涉及到比较复杂的查询时 Hibernate 的操作对象的方式用起来非常麻烦，甚至无法实现，只能用 sql 操作
- 底层需要频繁的拼接 sql，产生大量冗余的 sql

2.C MyBatis

是一种半自动对象-表映射关系的 DAO 层框架，可以自动的进行对象的封装，但是 sql 仍然需要自己来写。

- 结合了 JDBC 和 Hibernate 的优点
- 可以手写 sql 灵活实现数据访问
- 自动封装数据
- 减少冗余代码
- 执行效率比 JDBC 低

二 MyBatis 的结构



三 MyBatis 结构的创建

1 创建 XXXMapper.xml 文件

在该文件下配置 sql 语句，供程序调用

例如：创建一个 user 相关的 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 命 名 空 间
<mapper namespace="cn.shu.blog.dao.UserDaoInter">
  <!--用户登录-->
        <!— 这 里 的 #{} 代 表 可 以 接 收 参 数 -->
        <select id="userLogin" resultType="cn.shu.blog.beans.User">
        SELECT * FROM users WHERE account=#{userName} and
password=#{password}
                                                    </select>
</mapper>
```

2 创建 SqlMapConfig.xml 文件

Mybatis 的核心配置文件,在 Src 目录下创建该文件

注意：根标签<configuration>

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE configuration
        PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
        "http://mybatis.org/dtd/mybatis-3-config.dtd">
<configuration>
    <!-- 可 以 配 置 多 个 数 据 源 ， 这 里 指 定 默 认 值 -->
        <environments default="MySQL">
            <environment id="MySQL">
                <transactionManager type="JDBC"/>
                    <dataSource type="">
                <property name="driver" value="com.mysql.jdbc.Driver"/>
                    <property name="url" value="jdbc:mysql:///myblog"/>
                        <property name="username" value="admin"/>
                            <property name="password" value="pws"/>
                    </dataSource>
                </environment>
            </environments>
                <mappers>
            <!-- 指 定 SQL 映 射 文 件 可 以 写 多 个 -->
                    <mapper resource="mappers/userMapper.xml"/>
                </mappers>
</configuration>
```

3 创建连接工厂

4 获取连接

```
public static void main(String[] args) throws IOException {
        //1、加载配置文件
    InputStream inputStream = Resources.getResourceAsStream("sqlMapConfig.xml");
        //2、创建连接工厂
    SqlSessionFactory build = new SqlSessionFactoryBuilder().build(inputStream);
        //3、获取连接
    SqlSession sqlSession = build.openSession();
        //4、操作数据库
    Map<String,String> map=new HashMap<>();
    map.put("userName","shuxinsheng");// 多个参数用map传递
    map.put("password","123");
    sqlSession.selectOne("cn.shu.blog.dao.UserDaoInter.userLogin",map);
        //5、关闭连接
    sqlSession.close();
}
```

SqlSession 有很多方法，第一个参数为 SQL 的命名空间 . SQL ID

第二个参数为传递给 SQL 的参数

四 MyBatis 接口方式使用

在上面的例子中，调用查询方法，至少需要传递一个 String 参数(SQL 语句的命名空间 . SQL ID),另一个可选参数即为传递给 SQL 的参数:

```
sqlSession.selectOne("cn.shu.blog.dao.UserDaoInter.userLogin",map);
```

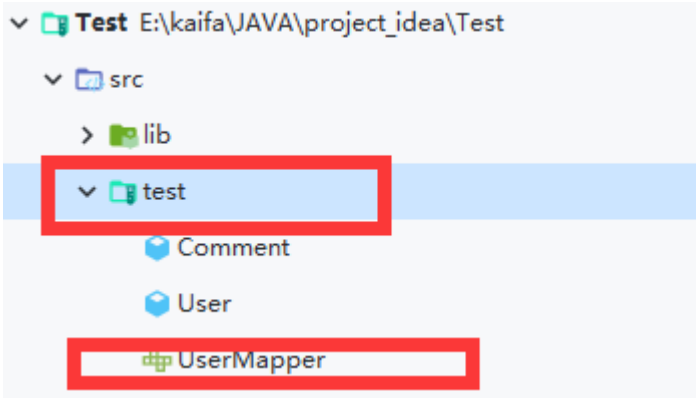
他们的缺点还是挺明显的，命名空间.SQLID 名字太长，且容易写错，不易维护，可读性差。其次是传递的参数类型有限制。

为了简化 MyBatis 的使用，MyBatis 提供了接口方式自动生成调用过程的机制，可以大大简化 MyBatis 的开发。

1 创建 SQL 映射文件

```
<?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">
<mapper namespace="test.UserMapper">
<select id="userLogin" resultType="test.User" >
SELECT * FROM users where account =#{account}
</select>
</mapper>
```

2 创建接口(接口的全路径名与命名空间一致)



3 创建方法(方法名与 SQL 的 ID 一致)

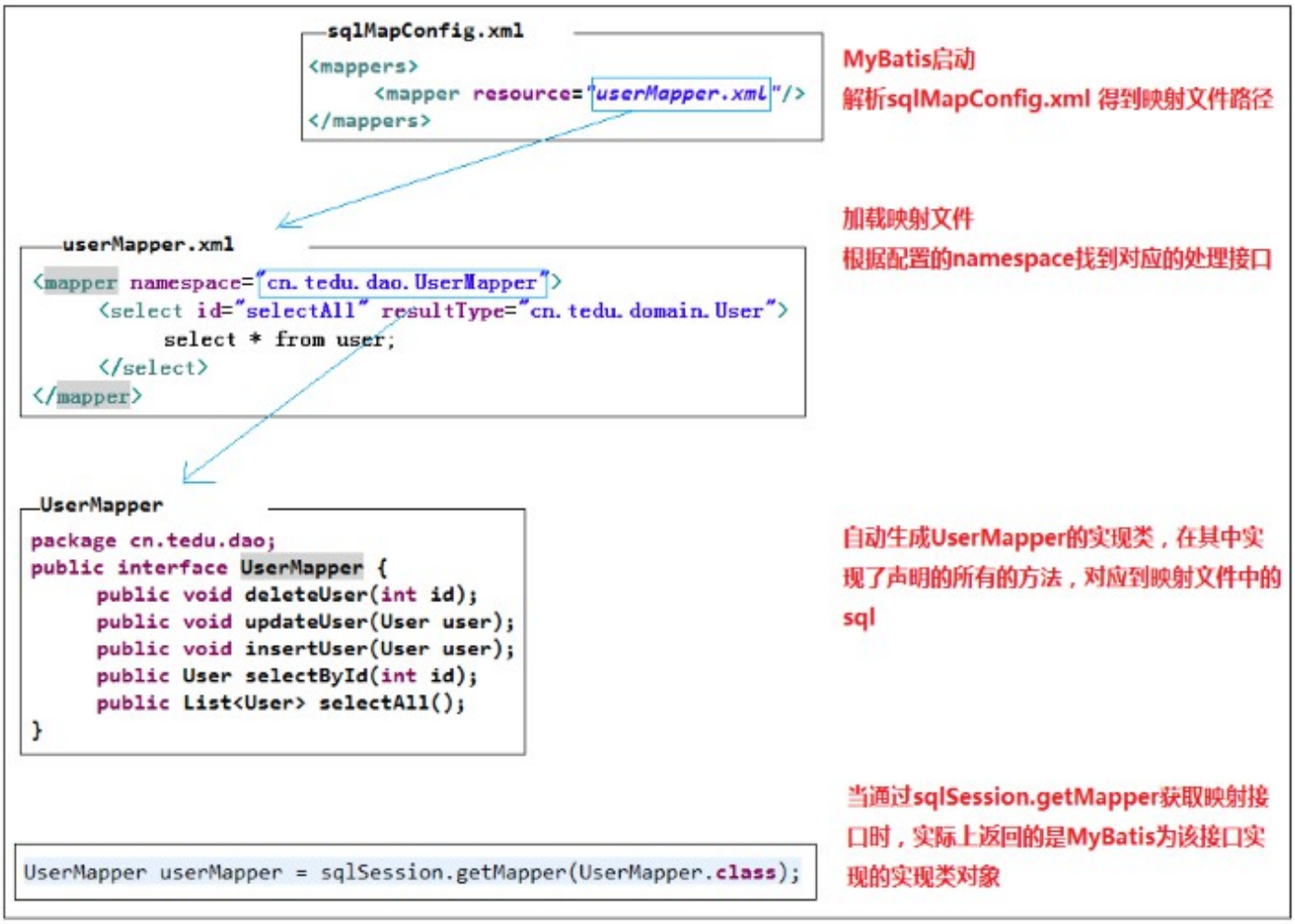
4 声明方法参数(方法参数与 SQL 中的参数一致)

5 声明方法返回值（方法返回值与 SQL 中声明的返回值一致)

```
public interface UserMapper {
    //接口方法
    public User userLogin( String account);
}
```

6 使用

```
//1、加载配置文件
InputStream inputStream = Resources.getResourceAsStream("sqlMapConfig.xml");
//2、创建连接工厂
SqlSessionFactory build = new SqlSessionFactoryBuilder().build(inputStream);
//3、获取连接
SqlSession sqlSession = build.openSession();
//4、获取对象 MyBatis 会自动创建该类的实现类并返回
UserMapper userMapper=sqlSession.getMapper(UserMapper.class);
//5、执行 SQL
userMapper.userLogin("123");
//6、关闭连接
sqlSession.close();
```



五 MyBatis 的增删改查

1 Insert

通过 Trim 拼接，prefix 表示拼接完成后以什么开头，suffix 以什么结尾, suffixOverrides 表示什么分隔，他会去掉指定的最后一个参数的逗号

Test 指定表达式，值为 True 则拼接

```
<insert id="myInsert">
insert into user
<trim prefix="(" suffix=")" suffixOverrides=",">
id,
<if test="name!=null">name,</if>
</trim>
values
<trim prefix="(" suffix=")" suffixOverrides=",">
id,
<if test="name!=null">#{name},</if>
</trim>
</insert>
```

2 Delete

```
<delete id="myDelete">
   delete from myblog.users
   <where>
      <if test="name!=null">name=#{name}</if>
   </where>
</delete>
```

也可通过 foreach 遍历 然后拼接 where

Collection 传递的集合名

Open 以什么开头

Close 以什么结尾

Separator 以什么分隔

最终效果 in（ “shu”，” xin” “sheng”）

```
<delete id="myDelete">
   delete from myblog.users where name in
   <foreach collection="names" open="(" close=")" separator="," item="name">
      #{name}
   </foreach>
</delete>
```

3 Update

Set 标签会自动去掉最后一个逗号

```
<update id="myUpdate">
   update user
   <set>
      <if test="name!=null">name=#{name},</if>
   </set>
   <where>
      <if test="name!=null">name=#{name}</if>
   </where>
</update>
```

4 Select

查询结果可以封装到 Bean 中，需要通过 resultType 指定类的全路径名

```
<select id="userLogin" resultType="test.User" >
   SELECT * FROM users
   <where>
      <if test="account!=null">account =#{account}</if>
   </where>
</select>
```

如果查询结构的列名和 Bean 属性不一致，可以手动指定映射关系

```
<resultMap id="re" type="test.User">
   <!--主键列 必须配置-->
   <id property="id" column="userId"/>
   <!--其它列-->
   <result property="name" column="userName"/>
</resultMap>
```

```
<select id="userLogin" resultMap="re">
  SELECT * FROM users
  <where>
    <if test="account!=null">account =#{account}</if>
  </where>
</select>
```

5 注意事项

- **上面的 foreach set where 等标签可以根据情况在不同类型的 SQL 语句中使用**
- **普通方式的 Insert、Delete、Update，操作完成记得提交事务**

六 参数传递方式

上面的例子中传递了 Map 参数，参数传递的方式有多种。
首先是在 sql 语句中用#{}或${}指定接收的参数名，然后在调用方法时传递参数即可

1 #{}与${}传值的区别

1.A #{}

- 相当于 PreParedStatement
- #{}作为方法参数，替代 SQL 中的?
- #{}预编译，效率高、防止 sql 注入
- #{}会在字符串类型二边添加单引号

1.B ${}

- 会原样拼接到 SQL 上
- 没有预编译、效率低、不能防止 SQL 注入

- 字符串类型二边不会拼接单引号
- 在 order by 等其它语句中指定字段名，不需要单引号，所以就必须用这个

2 单值传递

2.A XML:resultType 中指定返回的数据类型为 User 对象

```
<!--用户登录-->
<select id="userLogin" resultType="User">
   SELECT * FROM users where account =#{account}
</select>
```

2.B JAVA：传递参数 1 赋值给 sql 语句的#{}位置

```
User user=sqlSession.selectOne("cn.shu.blog.dao.test.userLogin","1");
System.out.println(user);
```

2.C 注意，单值传递如果需要 if 判断，不能用参数名

```
   <!--用户登录-->
<select id="userLogin" resultType="User">
   SELECT * FROM users
   <if test="account!=null">//报错
      where account =#{account}
   </if>
</select>
```

这里会报错，因为单值传递没有指定参数名

```
There is no getter for property named 'account' in 'class java.lang.String'
```

！！！ 需要用 _parameter

```
   <!--用户登录-->
<select id="userLogin" resultType="User">
   SELECT * FROM users
   <if test="_parameter!=null">
      where account =#{account}
   </if>
</select>
```

3 顺序传参

这种方式只能通过 MyBatis 的接口方式使用，普通方式只能接收一个 Object 参数
即 Mapper 接口的参数有多个，无需@param 指定，然后再 xml 文件中通过序号接收

3.A 接口方法

```
public User userLogin(String account,String password);
```

3.B Xml 文件

```
<select id="userLogin" resultType="test.User">
   SELECT * FROM users where account=#{account}
   and password=#{password}
</select>
```

3.C 使用问题

因为按顺序传递，xml 文件的 SQL 语句只能通过序号接收，或者在接口方法用 @Param（“name”）注解指定参数名,所以报错

```
Parameter 'account' not found. Available parameters
UserMapper mapper = sqlSession.getMapper(UserMapper.class);
//报错 Parameter 'account' not found. Available parameters
User user = mapper.userLogin("123", "123");
```

3.D 问题解决：

3.D.a @Param 修饰

```
public User userLogin(@Param("account") String account,
@Param("password")String password);
```

```
<select id="userLogin" resultType="test.User">
   SELECT * FROM users where account=#{account} and
password=#{password}
</select>
```

3.D.b 序号(0 1 ...)

```
<select id="userLogin" resultType="test.User">
   SELECT * FROM users where account=#{0} and password=#{1}
</select>
```

3.D.c 参数序号(param1...)

```
<select id="userLogin" resultType="test.User">
   SELECT * FROM users where account=#{param1} and
password=#{param2}
</select>
```

4 Map 传参

在 SQL 语句中通过，Map 写的的 KEY 来接收参数

创建 Map

```
Map<String,String> map=new HashMap<>();
map.put("account","123");
map.put("password","123");
```

4.A 普通方式

```
User user = sqlSession.selectOne("test.UserMapper.userLogin", map);
```

4.B 接口方式

接口方法:

```
//接口方法
public User userLogin(Map map);
```

sql:

```
<select id="userLogin" resultType="test.User">
```

```
<!—account 和 password 为 map 的 key-->
   SELECT * FROM users where account=#{account} and
password=#{password}
</select>
```

调用:

```
UserMapper mapper = sqlSession.getMapper(UserMapper.class);
User user = mapper.userLogin(map);
```

5 bean 传参

即传递一个 Bean 对象，SQL 语句中通过指定每个属性名获取参数

经测试，无需 get 或 set 方法一样可以传递参数

5.A 普通方式

```
User loginUser = new User("guest","guest");
User user = sqlSession.selectOne("test.UserMapper.userLogin", loginUser);
```

5.B 接口方式

接口方法:

```
//接口方法
public User userLogin(User user);
```

使用:

```
//调用
User loginUser = new User("guest","guest");
User user=sqlSession.getMapper(UserMapper.class).userLogin(loginUser);
```

6 JSON 传参

SQL 语句通过 JSON 的 key 查找对应值，没找到为 null，查询结果应该也为空

需要引入 JSON 相关 JAR 包

```
// 生 成 JSON
String json="{\"account\":123,\"password\":123}";
JSONObject jsonObject = JSONObject.parseObject(json);
```

6.A 普通方式传参

```
User user1 = sqlSession.selectOne("test.UserMapper.userLogin", jsonObject);
```

6.B 接口方式传参

```
//接口方法
public User userLogin(JSONObject jsonObject);
```

使用:

```
User user2=sqlSession.getMapper(UserMapper.class).userLogin(jsonObject);
```

7 集合类型传递参数(List、Set、Array)

在 SQL 中 foreach 遍历，读取

```
//创建集合
List<String> arrayList = new ArrayList<>();
arrayList.add("1");
arrayList.add("123");
arrayList.add("123456");
```

SQL:

```
<!—-SQL 语句-->
<select id="userLogin" resultType="test.User" >
   SELECT * FROM users where account in
   <foreach collection="list" item="ac" separator="," open="(" close=")">
      #{ac}
   </foreach>
</select>
```

7.A 普通方式

```
List<User> user8 =
sqlSession.selectList("test.UserMapper.userLogin",arrayList);
```

7.B 接口方式

```
//接口方法
public List<User> userLogin(List<String> list);
```

```
//使用
List<User>
user9=sqlSession.getMapper(UserMapper.class).userLogin(arrayList);
```

7.C Foreach

foreach 元素的属性主要有 item，index，collection，open，separator，close。
item 表示集合中每一个元素进行迭代时的别名，
index 指定一个名字，用于表示在迭代过程中，每次迭代到的位置，
open 表示该语句以什么开始，
separator 表示在每次进行迭代之间以什么符号作为分隔符，
close 表示以什么结束

在使用 foreach 的时候最关键的也是最容易出错的就是 collection 属性，该属性是必须指定的，但是在不同情况下，该属性的值是不一样的，主要有一下 3 种情况：

1.如果传入的是单参数且参数类型是一个 List 的时候，collection 属性值为 list
2.如果传入的是单参数且参数类型是一个 array 数组的时候，collection 的属性值为 array
3.如果传入的参数是多个的时候，我们就需要把它们封装成一个 Map 或者 Object

8 (对象+集合)传参或者(对象+对象)

Bean 对象:

```
public class User {
    private Comment comment;
```

```
}
public class Comment {
    private String comm="1";
}
```

这种方式应该只能接口方式，因为需要@Param 指定参数名

```
//接口方法
public User userLogin(@Param("user") User user8);
```

同理如果 user 对象的 comment 对象是集合，同样可以 foreach 遍历

```
    <!--
    user.comment.comm
    获取 bean 参数 user 对象
    再获取其属性 comment 对象
    最后获取 comment 对象的 comm 属性
    -->
<select id="userLogin" resultType="test.User" >
    SELECT * FROM users where account =#{user.comment.comm}
</select>
```

七　返回值的类型

1　返回一般数据类型

```
<!--
    指定 resultType 返回值类型时 String 类型的，
    string 在这里是一个别名，代表的是 java.lang.String

    对于引用数据类型，都是将大写字母转小写，比如 HashMap 对应的别名是 'hashmap'
    基本数据类型考虑到重复的问题，会在其前面加上 '_'，比如 byte 对应的别名是 '_byte'
  -->
  <select id="getEmpNameById" resultType="string">
    select username from t_employee where id = #{id}
```

```
</select>
```

八 MyBatis 多表查询

1 一对一

Comments:

```
public class Comments {
  private long id;
  private String comment;
  private Users users;
}
```

Users:

```
public class Users {
  private long id;
  private String nickname;
}
```

在通过 MyBatis 实现一对一查询时，需要通过 resultMaps 指定如何将结果集中的列名对应到目标 bean 中，在一对一的 bean 中，如果包含了另一个表的对应对象，则可以在 resultMap 标签中通过 association 标签来声明映射方式.

```
<!--获取文章评论-->
<resultMap id="comments" type="cn.shu.bean.Comments">
    <!--把数据库返回字段的 commentId 赋值给 Comments 对象的 id 属性-->
    <id property="id" column="commentId"/>
    <!--把数据库返回字段的 commentId 赋值给 Comments 对象的 id 属性-->
    <!--如果不写，数据库返回的字段不会赋值给对象的属性，会是 null-->
    <result column="comment" property="comment"/>

    <!-- property="users" 即 comments 中的 users 属性-->
```

```
    <association property="users" javaType="cn.shu.bean.Users">
        <!--把数据库返回字段的 userId 赋值给 Comments 对象下 Users 对象的 id 属性--
>
        <id column="userId" property="id"/>
        <!--把数据库返回字段的 nickname 赋值给 Comments 对象下 Users 对象的
nickname 属性-->
        <result column="nickname" property="nickname"/>
    </association>
</resultMap>
<select id="getArticleComments" resultMap="comments">
    select comments.id as commentId,users.id as userId,comment,nickname from
comments,users
    where  comments.userId=users.id
</select>
```

这里本质上不是数据库的一对一关系，但是一个评论只对应一个用户，可以用这种方式查询

2 一对多

文章的**分类**与**文章**是**一对多**关系，一个分类对应多篇文章，一篇文章只能一个分类

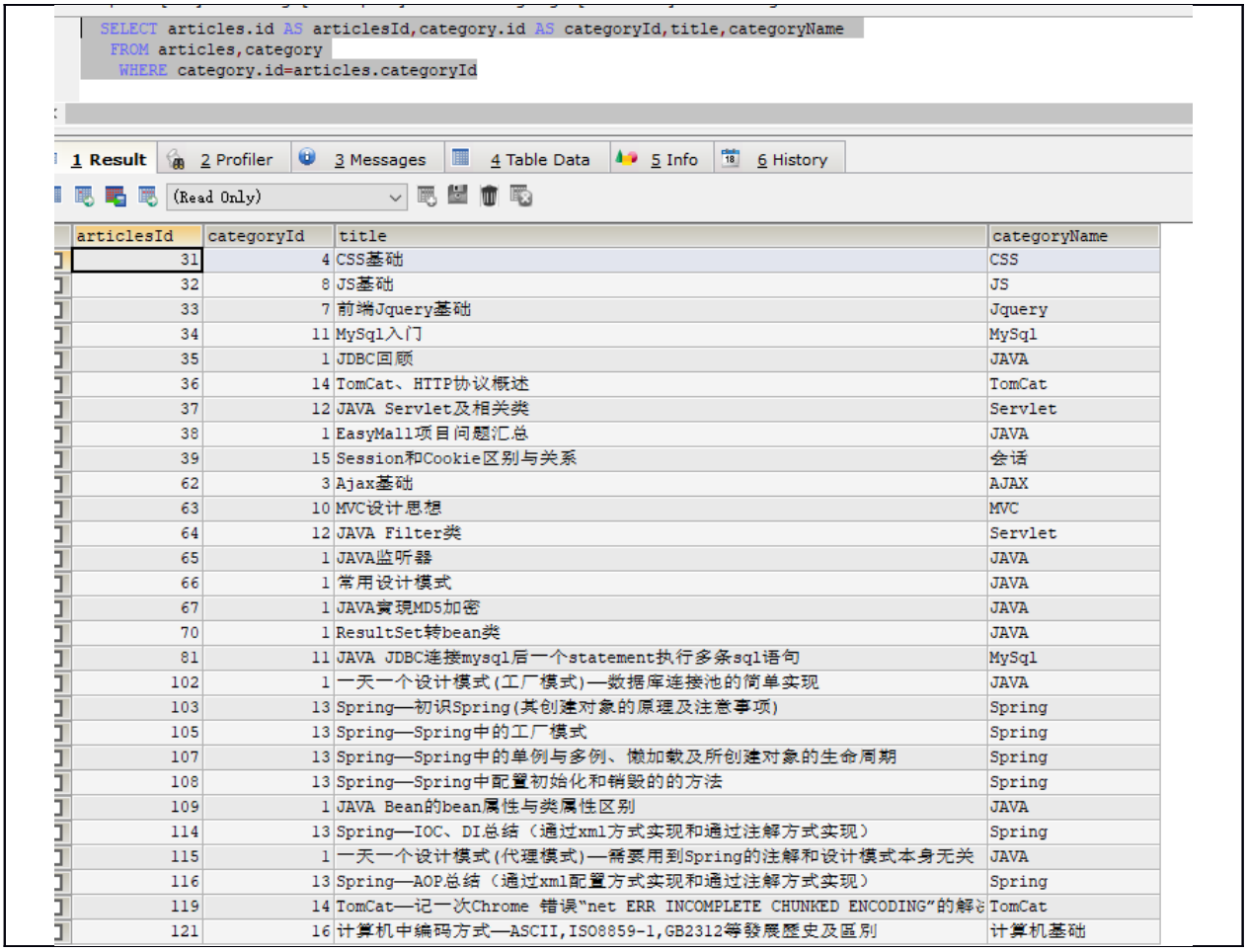
Category：

```
public class Category {
  private long id;
  private String categoryName;
  //一个分类 对应多篇文章
  private List<Articles> articlesList;
}
```

Articles：

```
public class Articles {
  private long id;
  private String title;
  //一篇文章 只有一个分类
  private Category category;
}
```

数据库查询结果：



```
SELECT articles.id AS articlesId,category.id AS categoryId,title,categoryName
 FROM articles,category
  WHERE category.id=articles.categoryId
```

1 Result | 2 Profiler | 3 Messages | 4 Table Data | 5 Info | 6 History

(Read Only)

articlesId	categoryId	title	categoryName
31	4	CSS基础	CSS
32	8	JS基础	JS
33	7	前端Jquery基础	Jquery
34	11	MySql入门	MySql
35	1	JDBC回顾	JAVA
36	14	TomCat、HTTP协议概述	TomCat
37	12	JAVA Servlet及相关类	Servlet
38	1	EasyMall项目问题汇总	JAVA
39	15	Session和Cookie区别与关系	会话
62	3	Ajax基础	AJAX
63	10	MVC设计思想	MVC
64	12	JAVA Filter类	Servlet
65	1	JAVA监听器	JAVA
66	1	常用设计模式	JAVA
67	1	JAVA实现MD5加密	JAVA
70	1	ResultSet转bean类	JAVA
81	11	JAVA JDBC连接mysql后一个statement执行多条sql语句	MySql
102	1	一天一个设计模式(工厂模式)—数据库连接池的简单实现	JAVA
103	13	Spring—初识Spring(其创建对象的原理及注意事项)	Spring
105	13	Spring—Spring中的工厂模式	Spring
107	13	Spring—Spring中的单例与多例、懒加载及所创建对象的生命周期	Spring
108	13	Spring—Spring中配置初始化和销毁的的方法	Spring
109	1	JAVA Bean的bean属性与类属性区别	JAVA
114	13	Spring—IOC、DI总结（通过xml方式实现和通过注解方式实现）	Spring
115	1	一天一个设计模式(代理模式)—需要用到Spring的注解和设计模式本身无关	JAVA
116	13	Spring—AOP总结（通过xml配置方式实现和通过注解方式实现）	Spring
119	14	TomCat—记一次Chrome 错误"net ERR INCOMPLETE CHUNKED ENCODING"的解决	TomCat
121	16	计算机中编码方式—ASCII,ISO8859-1,GB2312等发展历史及区别	计算机基础

这里以 Category 为视角查询，则一个 Category 下面包含多个 Articles 对象(一对多)

此时需要用 collection 标签指定 articlesList 属性为 Articles 类型，并设置对应关系

指定类型时用 ofType,注意: association 用的 javaType

```
<resultMap id="resultMapId" type="cn.shu.bean.Category">
    <id property="id" column="categoryId"/>
    <result property="categoryName" column="categoryName" />
    <!--这里用的 ofType-->
    <collection property="articlesList" ofType="cn.shu.bean.Articles">
        <id column="articlesId" property="id"/>
        <result column="title" property="title"/>
    </collection>

</resultMap>
<select id="getArticleComments" resultMap="resultMapId">
    SELECT articles.id as articlesId,category.id as categoryId,title,categoryName
    FROM articles,category
```

```
    WHERE category.id=articles.categoryId
</select>
```

如果以**文章**视角查询，那么一个**文章**对应一个**分类**，那么就回到了**一对一的**查询方式

3 多对多

无论从哪一方的**视角**查询，分开来看，都可以按照**一对多**的方式查询

九 MyBatis 的其它标签

1 别名标签

映射文件中大量使用类名长的 JAVA 类型，可以建立别名，需要的地方引用即可

sqlMapConfig 文件中：

注意:该标签要放到最前面，configuration 内的前面,但不能在 settings 前面

```
<typeAliases>
    <typeAlias type="cn.shu.blog.beans.Article" alias="article"/>
</typeAliases>
```

映射文件中：代替长串包名

```
<select id="asd" resultType="article">
    select * from user
</select>
```

2 Sql 的复用

某段 SQL 重复出现，可以提取出来，到处引用

```
<sql id="testSql">
    select * from user
```

```
</sql>
<select id="asd" resultType="article">
   <include refid="testSql"/>  where id=10
</select>
```

十　MyBatis 的缓存

缓存机制可以减轻数据库的压力，原理是在第一次查询时，将查询结果保存起来，后续查询相同的 SQL，不是真的去查数据库，而是直接返回的第一次查询后保存的值。

虽然降低了数据库的压力，但同时不能及时获取最新的数据。

第三方工具 Redis 也可实现缓存

1　一级缓存

只在一个事务中有效：即在一个事务中先后执行多次同一查询，只有第一次去查。

而如果是不同事务执行相同 SQL 并不会共用缓存。

MyBatis 一级缓存默认开启，且无法手动关闭

2　二级缓存

缓存全局有效，一个事务查询一个 SQL 得到结果，其它事务查询相同 SQL，得到之前缓存的结果

作用范围大，时间长，可能造成危害更大，开发中很少使用。

MyBatis 二级缓存默认关闭

2.A 开启二级缓存

a 可在 SQL 映射文件中开启

```
<mapper>
    <cache/>
</mapper>
```

B 也可在 MyBatis 配置文件 sqlMapConfig.xml 中开启

注意：该方式需放在 configuration 标签下，且必须位于第一行，不然报错

```
<configuration>
    <!—开启二级缓存 -->
    <settings>
        <setting name="cacheEnabled" value="true"/>
    </settings>
</configuration>
```