尚马教育 JAVA 课程

Mybatis

文档编号：C01

创建日期： 2017-07-07

最后修改日期：2021-03-10

版 本 号：V3.6

电子版文件名：尚马教育-第三阶段-1.mybatis 基础.docx

文档修改记录：

更新日期	更新作者	更新说明	版本号
2017-07-30	张元林	初始版本	V1.0
2018-08-01	王绍成	Mybatis 版本更新	V2.0
2019-08-09	冯勇涛	课件格式以及课程深度加深	V3.0
2019-11-11	冯勇涛	格式调整以及多参数	V3.1
2021-01-18	冯勇涛	课程顺序调整，细节	V3.5
2021-03-10	冯勇涛	Mysql 连接 url 完整	V3.6

目录

1. Mybaits 介绍

1.1. 认识 mybatis

- MyBatis 本是 apache 的一个开源项目 iBatis, 2010 年这个项目由 apache software foundation 迁移到了 google code，并且改名为 MyBatis 。2013 年 11 月迁移到 Github，通俗说法 Ibatis3 = MyBatis
- iBATIS 一词来源于“internet”和“abatis”的组合，是一个基于 Java 的持久层框架。iBATIS 提供的持久层框架包括 SQL Maps 和 Data Access Objects（DAO）
- MyBatis 是一个数据持久层(ORM)框架。把实体类和 SQL 语句之间建立了映射关系，是一种半自动化的 ORM 实现
- MyBatis 的优点：
  - 减少代码量
  - 基于 SQL 语法，简单易学
  - 能了解底层组装过程
  - SQL 语句封装在配置文件中，便于统一管理与维护，降低了程序的耦合度
  - 程序调试方便

1.2. jdbc 与 mybatis 对比

- Jdbc 代码：

```
Class.forName(“com.mysql.jdbc.Driver”);

Connection conn =DriverManager.getConnection(“url”,”username”,”password”);

String sql = “select * from m_user where userId=?”;

PrepareStatement pst = conn.prepareStatement(sql);

Pst.setInt(1,1);

ResultSet rs = pst.executeQuery();

while(rs.next()){

      System.out.println(rs.getString(1));

}
```

- Mybatis 代码：

```
<select id="getUser" parameterType="int"     resultType="com.javasm.entity.UserInfo">

  select * from m_user where id=#{id}

</select>
```

2. 快速入门 mybatis

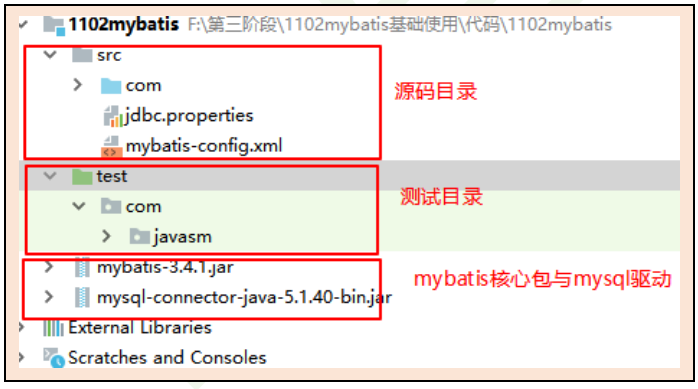
2.1. 步骤

1. 创建 java 工程
2. 添加 mybatis 与 mysql 数据库驱动包。
3. 在 mysql 数据库下创建数据库表。
4. 在 entity 包下创建数据库表对应的实体类。
5. 在 mapper 包下创建实体类对应的 xml 映射文件。
6. 在类路径下创建 mybatis 的 xml 核心配置文件,注意引入映射文件。
7. 使用 mybatis 的核心 api 对象加载配置文件。

具体细节步骤如下：

2.1.1. 添加 Mybaits 与数据库驱动包

- 创建 java 工程，并加入 mybatis 的 jar 包以及数据库驱动包。



2.1.2. 创建数据库表

系统用户表 DDL 语句：

```
CREATE TABLE `sysuser` (
    `id` int(11) NOT NULL,
    `uname` varchar(30) NOT NULL COMMENT '用户名',
    `upwd` varchar(20) NOT NULL COMMENT '密码',
    `uphone` char(11) NOT NULL COMMENT '手机号',
    `uimg` varchar(50) DEFAULT NULL COMMENT '头像',
    `uwechat` varchar(20) DEFAULT NULL COMMENT '微信号',
    `uemail` varchar(20) DEFAULT NULL COMMENT '邮箱',
    `ubirthday` date DEFAULT NULL COMMENT '生日',
    `deleted` char(1) DEFAULT '0' COMMENT '是否删除',
    `create_time` timestamp NULL DEFAULT CURRENT_TIMESTAMP,
    `update_time` timestamp NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,
    PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COMMENT='系统用户表';
```

系统用户表设计图：

名		类型	长度	小数点	允许空值 (	
id	用户id	int	11	0	☐	1
uname	用户名	varchar	30	0	☐	
upwd	密码	varchar	20	0	☐	
uphone	手机号	char	11	0	☐	
uimg	头像	varchar	50	0	☑	
uwechat	微信号	varchar	20	0	☑	
uemail	邮箱	varchar	20	0	☑	
ubirthday	生日	date	0	0	☑	
deleted	是否删除	char	1	0	☑	
create_time	创建时间	timestamp	0	0	☑	
update_time	最后修改时间	timestamp	0	0	☑	

2.1.3. 创建数据库表对应的实体类

```
package com.javasm.sys.entity;
public class Sysuser {
    private Integer id;
    private String uname;
    private String upwd;
    private String uphone;
    private String uimg;
    private String uwechat;
    private String uemail;
    private String ubirthday;
    private String deleted;
    private String createTime;
    private String updateTime;
    //TODO 生成 setter,getter,toString
}
```

2.1.4. 创建实体类对应的 xml 映射文件

创建 package 包：com.javasm.sys.mapper，在该包下创建映射文件 sysuserMapper.xml。

```
<mapper namespace="userMapper">
    <select id="getUserById"
parameterType="java.lang.Integer"
resultType="com.javasm.sys.entity.Sysuser">
        select * from sysuser where id=#{id}
    </select>
</mapper>
```

映射文件中的注意事项：

- 注意点 1：namespace:表示当前映射文件的唯一命名空间,确保不同映射文件之间的 namespace 不要重复。
- 注意点 2：进行查询的 sql 语句，标签名必须是 select 标签。
- 注意点 3：select 标签的 id，在当前映射文件中必须唯一。
- 注意点 4：parameterType 表示查询参数类型；resultType 表示数据库查询结果的每行记录封装类型。
- 注意点 5：#{}括号中的内容在当前查询参数是 int 时，可以随意定义。

2.1.5. 类路径下创建 xml 核心配置文件

- 在项目 src 目录中创建 mybatis-config.xml 文件
  - ✓ 配置当前数据库的数据源：

```
<?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>
    <!--配置数据库环境，支持多数据库环境-->
    <environments default="dev">
        <environment id="dev">
            <!--事务管理器,JDBC 表示启用 JDBC 事务管理-->
            <transactionManager type="JDBC"></transactionManager>
            <!--连接池：POOLED 表示初始化 PooledDataSource 连接池对象-->
            <dataSource type="POOLED">
                <property name="driver" value="com.mysql.jdbc.Driver"/>
                <property name="url"
value="jdbc:mysql://127.0.0.1:3306/720A?useUnicode=true&characterEncoding=utf8&useS
SL=true&serverTimezone=UTC"/>
                <property name="username" value="root"/>
```

```
            <property name="password" value="root"/>
        </dataSource>
    </environment>
</environments>
<!--引入 xml 映射文件-->
<mappers>
    <mapper resource="com/javasm/sys/mapper/sysuserMapper.xml"></mapper>
</mappers>
</configuration>
```

2.1.6. mybatis 的 api 调用

① 创建 SqlSessionFactoryUtil 工具类

```
public class SqlSessionFactoryUtil {
    private static SqlSessionFactory ssf=null;
    static{
        InputStream input = null;
        try {
            input =
Resources.getResourceAsStream("mybatis-config.xml");
        } catch (IOException e) {
            e.printStackTrace();
        }
        ssf = new
SqlSessionFactoryBuilder().build(input);
    }
    public static SqlSessionFactory
getSqlSessionFactory(){
        return ssf;
```

```
    }
}
```

② 创建 junit 测试类

```
public class TestStart {
    private static SqlSessionFactory ssf = null;
    private SqlSession session = null;
    @BeforeClass
    public static void initFactory() {
        ssf = SqlSessionFactoryUtil.getSqlSessionFactory();
    }
    @Before
    public void openSession() {
        session = ssf.openSession();
    }
    @Test
    public void testGetUserById() throws IOException {
        Object result = session.selectOne("userMapper.getUserById",
1);
        System.out.println(result);
    }
    @After
    public void closeSession() {
        session.close();
    }
    @AfterClass
    public static void closeFactory() {
        ssf = null;
```

```
    }
}
```

2.2. 重点

- SqlSessionFactoryBuilder 构建器对象生命周期很短暂，加载 xml 完成后即销毁。
- SqlSessionFactory 会话工厂对象生命周期全局唯一。
- SqlSession 会话对象生命周期很短暂，需要操作数据库则打开，操作完成后必须关闭释放连接。
- SqlSession 的 selectOne 方法的第一个参数是 namespace.id 组成的字符串，第二个参数是查询参数，且当前只能传入一个参数。
- 映射文件的 sql 语句动态参数使用#{}表达式获取，查询参数是 String，int，double 等简单类型时，括号中内容随意定义，一般写为数据库字段名。

3. SqlSession 对象的 selectList 方法

3.1. 修改映射文件

在 sysuserMapper.xml 映射文件中添加新的 select 标签。

```
<select id="selectUsers"
resultType="com.javasm.sys.entity.Sysuser">
    select * from sysuser order by update_time
</select>
```

3.2. 添加测试方法

在 junit 测试类中添加新的测试方法

```
@Test
public void testSelectUsers() throws IOException {
    List<Object> objects =
session.selectList("userMapper.selectUsers");
    System.out.println(objects);
}
```

3.3. 重点

注意点 1：当 selectUsers 查询无参数时，parameterType 可以省略(其实一直可以省略).

注意点 2：selectUsers 下的 sql 语句查询出列表数据，但 resultType 仍然是实体类，绝对不会出现 list，切记 resultType 表示查询结果单行记录封装类型。

4. SqlSession 对象的增删改方法

4.1. 修改映射文件

```
<insert id="addUser"
parameterType="com.javasm.sys.entity.Sysuser"
useGeneratedKeys="true" keyProperty="id">
  insert into sysuser(uname, upwd, uphone, uemail)
values (#{uname},#{upwd},#{uphone},#{uemail})
</insert>

<update id="updateUser"
parameterType="com.javasm.sys.entity.Sysuser">
    update sysuser set
uname=#{uname},upwd=#{upwd},uphone=#{uphone},uemail=#{ue
mail} where id=#{id}
</update>
```

```
<delete id="delUserByKey"
parameterType="java.lang.Integer">
    delete from sysuser where id=#{id}
</delete>
```

注意点：在 addUser 与 `updateUser` 标签内，传入的参数是 Sysuser 实体类，则#{}括号的内容不能随意定义，必须是实体类的属性名。

4.2. 添加测试方法

在 junit 测试类中添加新的测试方法

```
@Test
public void test1_addUser(){
    Sysuser suser = new Sysuser();
    suser.setUname("尚马学生李某");
    suser.setUemail("liXxx@163.com");
    suser.setUpwd("123456");
    suser.setUphone("15516122134");
    int rows=
session.insert("userMapper.addUser",suser);
    System.out.println("受影响的行数：  "+rows);
    System.out.println("获取自增 id:"+suser.getId());
}

@Test
public void test3_updateUser(){
    int id = 1;
```

```
    Sysuser suser = new Sysuser();
    suser.setId(id);
    suser.setUname("尚马学生李某人");
    suser.setUemail("liXxx@163.com");
    suser.setUpwd("123456");
    suser.setUphone("15516122134");
    int rows =
session.update("userMapper.updateUser",suser);
    System.out.println("结果: "+rows);
}

@Test
public void test4_delUserByKey(){
    int rows =
session.delete("userMapper.delUserByKey",1);
    System.out.println("结果: "+rows);
}
```

4.3. 重点

重点 1：insert，update，delete 标签对应的是数据库增删改操作，不能混用。

重点 2：insert 添加语句，如果主键 id 是自增，则可以使用 useGeneratedKeys="true" keyProperty="id"得到数据库生成的自增主键值，keyProperty 的值是**实体类的 id 属性名**。

重点 3：当 parameterType 的值是实体类自定义对象时，#{}内容必须是实体类属性名。

重点 4：insert，delete，update 标签没有 resultType 返回值。默认返回 int 受影响行数。

5. SqlSession 对象的 getMapper 方法(重要)

5.1. 创建 mapper 接口

```
package com.javasm.sys.mapper;


import com.javasm.sys.entity.Sysuser;
import java.util.List;


public interface SysuserMapper {
    public Sysuser getUserById(Integer uid);
}
```

5.2. 修改 sysuserMapper.xml

修改 mapper 标签的 namespace 为 mapper 接口名称

```
<?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="com.javasm.sys.mapper.SysuserMapper">
    <select id="getUserById"
parameterType="java.lang.Integer"
resultType="com.javasm.sys.entity.Sysuser" >
        select * from sysuser where id=#{id}
    </select>
</mapper>
```

5.3. 添加测试方法

在以上 junit 测试类中加入新的测试方法。

```
@Test
public void testGetUserByKey() throws IOException {
    SysuserMapper mapper = session.getMapper(SysuserMapper.class);
    Sysuser sysuser = mapper.getUserById(1);
    System.out.println(sysuser);
}
```

5.4. 重点

重点 1：xml 映射文件中的 namespace 不再随意，必须是 mapper 接口的名称（含包名）。

注意点 2：调用 SqlSession 会话对象方法时，通过 getMapper 方法返回 mapper 接口类型的对象是当前接口的代理对象，代理对象的细节后续分析。

6. 小单元总结

6.1. MyBatis 的核心对象

① SqlSessionFactoryBuilder：

SqlSessionFactoryBuilder 用过即丢，其生周期只存在于方法体内，可重用。

SqlSessionFactoryBuilder 来创建多个 SqlSessionFactory 实例。

② SqlSessionFactory：

SqlSessionFactory 是每个 MyBatis 应用的核心；

单例，存在于整合应用运行时，并且同时只存在一个对象实例；

③ SqlSession

包含了执行 sql 所需的所有方法，可以通过 SqlSession 实例直接运行映射的 sql 语句；

SqlSession 对应着一次数据库会话，它的生命周期不是永久的，在每次访问数据库时都需要创建它每个线程都有自己的 SqlSession 实例，SqlSession 实例是不能被共享，也不

是线程安全的。

调用 insert,update,selectList,selectOne,delete 等方法执行增、删、查、改等操作；

调用 getMapper(xxx.class)来实例化接口执行增删查改；

注意：并不是说在一个 SqlSession 会话中只能执行一次 Sql，可以执行多次，当关闭了 SqlSession 就需要重新创建。同时关闭是非常重要，必须确保正常关闭。

7. xml 核心配置认识

以上配置文件中配置了数据库连接信息，引入了映射文件，该文件中可以有其他标签。详细如下：

7.1. properties 标签

7.1.1. 在类路径下定义 jdbc.properties 文件

```
jdbc.user = root
jdbc.password = root
jdbc.url = jdbc:mysql://127.0.0.1:3306/704a?characterEncoding=utf8&useSSL=true
jdbc.driver = com.mysql.jdbc.Driver
```

7.1.2. 引入 properties 资源文件

在 xml 核心配置文件中通过 properties 标签引入 jdbc.properties

```
<properties resource="jdbc.properties"></properties>
```

修改原有的配置信息,采用变量引入式声明

```
<dataSource type="POOLED">
<property name="driver" value="${jdbc.driver}" />
```

```
<property name="url" value="${jdbc.url}" />
<property name="username" value="${jdbc.username}" />
<property name="password" value="${jdbc.password}" />
</dataSource>
```

7.2. Settings 标签

用来修改 MyBatis 在运行时的行为方式，主要是 Mybatis 的一些全局配置属性的设置。

设置项	描述	允许值	默认值
cacheEnabled	对在此配置文件下的所有cache 进行全局性开/关设置。	true | false	true
lazyLoadingEnabled	全局性设置懒加载。如果设为‘false’，则所有相关联的都会被初始化加载。	true | false	true
……(9个)	……	…….	……

```
<settings>
    <!--把数据库中的_分割的列名，映射到实体类的驼峰属性名 -->
    <setting name="mapUnderscoreToCamelCase" value="true"/>
    <!--mybatis 的运行日志输出位置-->
    <setting name="logImpl" value="STDOUT_LOGGING"></setting>
</settings>
```

7.3. typeAliases 标签

类型别名是 Java 类型的简称,它仅仅只是关联到 XML 配置，简写冗长的 JAVA 类名。同时

没有 mybatis 对常用简单类型内建了别名，比如 java.lang.Integer-int,java.util.String-String

```
<typeAliases>
    <typeAlias alias="User" type="com.javasm.entity.User"/>
</typeAliases>
```

```
<typeAliases>
        <package name="com.javasm"></package>
</typeAliases>
```

加入 typeAlias 配置后，映射文件中出现的类名可以写别名：

```
<select id="getUserById" parameterType="int"
resultType="Sysuser" >
    select * from sysuser where uid=#{aa}
</select>
```

Mybatis 内嵌别名

别名	映射的类型
string	String
long	Long
Integer	Integer
Int	Integer
double	Double
List	ArrayList
....	....

注意点 1：要求不同包下类不能同名，类名做别名，忽略大小写。

注意点 2：以上两个 typeAlias 一般选择配置包名。

注意点 3：别名影响的是映射文件中的 parameterType 与 resultType 属性。

7.4. Environments 标签

MyBatis 可以配置多套运行环境，将 SQL 映射到多个数据库上。

虽然可以配置多个运行环境，但是每个 SqlSessionFactory 实例只能选择一个运行环境。即：每个数据库对应一个 SqlSessionFactory 实例。


```
<environments default="development">
    <environment id="development">
        <transactionManager type="JDBC"/>
        <dataSource type="POOLED">
            <property name="driver" value="${driver}"/>
            <property name="url" value="${url}"/>
            <property name="username" value="${username}"/>
            <property name="password" value="${password}"/>
        </dataSource>
    </environment>
    <environment id="test">
    ……
    </environment>
</environments>
```

dataSource

dataSource 元素使用基本的 JDBC 数据源接口来配置 JDBC 连接对象的资源。有三种内建的数据源类型 <dataSource type=“[UNPOOLED|POOLED|JNDI]"/>

UNPOOLED 每次创建新的连接

POOLED 创建连接池,用完归还

JNDI 获取 JNDI 连接池

7.5. mappers 标签

Sql 映射语句一般定义在各持久类的 Mapper.xml 文件中，需要在配置中引用这些映射文件

```
<!-- 将 mapper 文件加入到配置文件中 -->
<mappers>
    <mapper resource="com/javasm/dao/UserMapper.xml"/>
</mappers>
```

8. xml 映射文件认识

8.1. mapper 元素

Mapper 元素的 namespace 属性一般都与对应的 dao 接口全名称保持一致；

```
<mapper namespace="com.javasm.dao.UserInfoDao"></mapper >
```

8.2. select 子元素

表示数据库查询，注意属性：id，parameterType，resultType

- Id 与 mapper 接口中方法名保持一致；
- parameterType 与 mapper 接口中方法形参类型一致，一般是 String,Map, 实体类,Integer,Double 等；
- resultType 与 mapper 接口中方法返回值类型一致,resultType 表示 mybatis 对查询结果的单行记录封装类型,一般写为实体类,Map,String,Integer,Double 等简单类型；

```
<select id="getUser" parameterType="int" resultType="Sysuser">
   select * from sysuser where id=#{id}
</select>
```

8.3. Update 子元素

update,insert,delete 子元素都没有 resultType 属性,默认返回受影响的行数.

```
<update id="updateUser" parameterType="Sysuser">
    update sysuser set
uname=#{uname},uphone=#{uphone},uemail=#{uemail} where uid=#{uid}
</update>
```

```
public int updateUser(Sysuser user);
```

Dao 接口的方法返回值默认是 int

8.4. Insert 子元素

```
<insert id="addUser" parameterType="Sysuser" useGeneratedKeys="true"
keyProperty="uid">
    insert into sysuser(uname, upwd, uphone, uemail) values
(#{uname},#{upwd},#{uphone},#{uemail})
</insert>
```

```
public int addUser(Sysuser user);
```

Dao 接口的方法返回值默认是 int

8.5. Delete 子元素

```
<delete id="delUser" parameterType="int">
    delete from sysuser where uid=#{uid}
</delete>
```

```
public int delUser(Integer uid);
```

8.6. #{}与${}表达式

两者都是用来获取查询参数,用于 sql 语句中.

8.6.1. #{}表达式

mybatis 内对#{}解析为?参数占位符,更加安全,因此对于条件查询参数必须使用#{}来获取数据.避免 sql 注入攻击.

1) 如果 parameterType 是简单类型的话,#{随便写}
2) 如果 parameterType 是实体类,#{类的成员变量名}
3) 如果 parameterType 是 map,#{map 的 key}

8.6.2. ${}表达式

mybatis 内对${}不解析?占位符,直接进行 sql 拼接,因此不适合做条件查询.如果有动态指定字段排序的场景,可以使用${}来获取排序字段以及排序方式.如下:

```
<select id="selectUsers" parameterType="map" resultType="sysuser">
    select * from sysuser order by ${soreField} ${sorted}
</select>
```

```
public List<Sysuser> selectUsers(Map<String,Object> map);
```

```
@Test
public void test2_selectUsers() {
    Map<String,Object> args = new HashMap<>();
    args.put("soreField","update_time");
    args.put("sorted","asc");
    SysuserDao ud = session.getMapper(SysuserDao.class);
    List<Sysuser> sysusers = ud.selectUsers(args);
    System.out.println(sysusers);
}
```

9. 多参数传递

mybatis 默认 api 只能够接收一个动态参数，因此当需要传入多个数据变量时，需要把多个变量封装到一个对象中，比如实体对象或 Map

9.1. 封装实体

Dao接口：

```
public Sysuser selectUser(Sysuser u);
```

Xml映射文件：

```
    <select id="selectUser" parameterType="sysuser"
resultType="sysuser">
        select * from sysuser where uname=#{uname} and
uphone=#{uphone}
    </select>
```

测试类：

```
ISysuserDao ud = session.getMapper(ISysuserDao.class);
Sysuser u = new Sysuser();
u.setUname("aaa");
u.setUphone("123123123");
Sysuser selectUser = ud.selectUser(u);
```

注意点：传入实体类时，#{}内容为实体类属性名

9.2. 封装 map

Mapper接口：

```
public Sysuser selectUser2(Map<String,Object> m);
```

Xml映射文件：

```
<!-- #{map的key} -->
    <select id="selectUser2" parameterType="map" resultType="sysuser">
        select * from sysuser where uname=#{unameKey} and
uphone=#{uphoneKey}
    </select>
```

测试类：

```
ISysuserDao ud = session.getMapper(ISysuserDao.class);
```

```
Map<String,Object> u = new HashMap<>();
u.put("unameKey", "aaa");
u.put("uphoneKey", "1231231232");Sysuser selectUser =
ud.selectUser2(u);
```

注意点：传入 map 时，#{}内容为 map 的 key 值

9.3. @Param 注解

接口的方法形参保持多参数,由 mybatis 底层把多参数封装到 Map 对象中,通过@Param 注解指定 key.

```
mapper接口：
public Sysuser selectUserByNameAndPhone(@Param("un")String
uname,@Param("up")String uphone);
```

```
Xml映射文件：
    <select id="selectUserByNameAndPhone" resultType="Sysuser">
       select * from sysuser where uname=#{un} and uphone=#{up}
    </select>
```

```
测试类：
ISysuserDao ud = session.getMapper(ISysuserDao.class);
Sysuser u = ud.selectUserByNameAndPhone("aaa", "123123123");
```

注意点：使用@Param 注解形参时，#{}内容是 Param 注解的 value 属性值。

10. 复杂查询返回 Map

如果进行多表连接查询，查询结果列无法直接封装到实体类，可以封装到一个 Map 集合中。

10.1. Mysql 创建 sysrole 表

```
CREATE TABLE `sysrole` (
  `id` int(11) NOT NULL,
  `rname` varchar(255) DEFAULT NULL COMMENT '角色名',
  `rdesc` varchar(255) DEFAULT NULL COMMENT '角色描述',
  `deleted` char(1) DEFAULT NULL COMMENT '是否删除',
  `create_time` timestamp NULL DEFAULT CURRENT_TIMESTAMP,
  `update_time` timestamp NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE
```

```
CURRENT_TIMESTAMP,
   PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

手动添加测试数据(人事部经理，人事部职员角色)

10.2. 修改 sysuser 表的数据

10.3. 映射文件中添加 select 标签

```
<select id="selectUserAndRoleByUID" parameterType="int"
resultType="map">
  select u.*,r.rname from sysuser u,sysrole r where
u.roleid=r.id and u.id=#{id}
</select>
```

注意：resultType 属性是 map，不再是一个实体类。

10.4. mapper 接口添加方法

```
public Map<String,Object> selectUserAndRoleByUID(Integer
uid);
```

11. 模糊查询

```
<select id="selectUsers" parameterType="Sysuser" resultType="Sysuser">
        select * from sysuser where uname like "%"#{uname}"%"
</select>
```

注意：模糊查询的%使用双引号括起来即可。

12. 总结

MyBatis 是一个数据持久层(ORM)框架。把实体类和 SQL 语句之间建立了映射关系，是一种半自动化的 ORM 实现，MyBatis 的优点：基于 SQL 语法，简单易学，能了解底层组装过程，SQL 语句封装在配置文件中，便于统一管理与维护，降低了程序的耦合度，程序调试方便。

MyBatis 环境搭建，实现对数据表简单的查询操作。

Mybatis 的 getMapper 方法原理，动态代理模式。

MyBatis 的核心对象：SqlSessionFactoryBuilder，SqlSessionFactory，SqlSession

mybatis-config.xml 全局配置文件常用标签。

Mybatis 映射文件常用标签。