Mybatis学习笔记

资源

官网

https://mybatis.org/mybatis-3/zh/index.html

项目地址

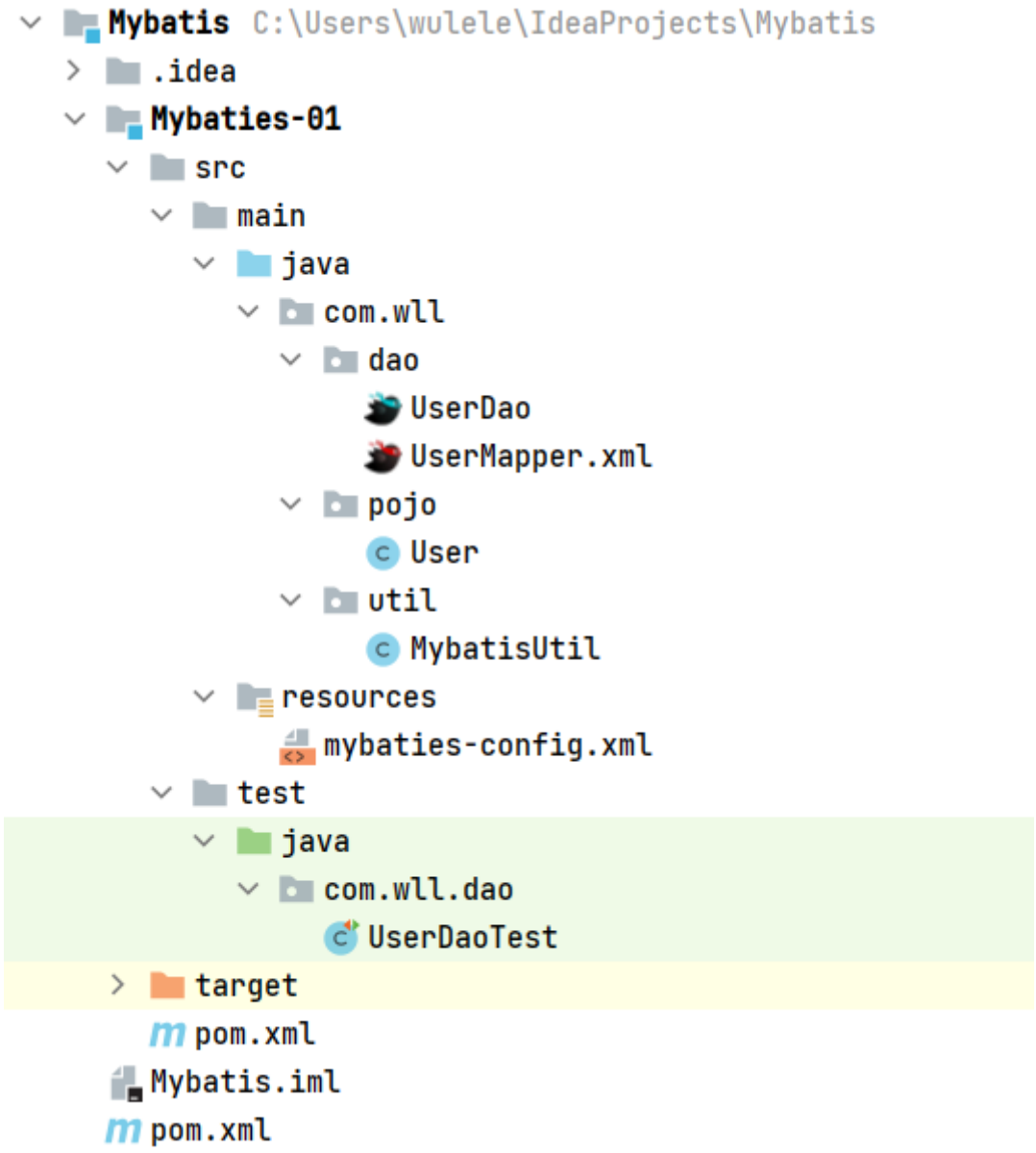
https://github.com/mybatis/mybatis-3/releases

Maven配置

```
<dependency>
    <groupId>org.mybatis</groupId>
    <artifactId>mybatis</artifactId>
    <version>3.5.6</version>
</dependency>
```

第一个Mybatis程序

文件结构



1. sql

```
create table user
(
    id   int(10)     not null
    primary key,
    name varchar(15) null,
    pwd  varchar(15) null
);
```

2. pom.xml

```
<dependencies>
    <dependency>
        <groupId>org.mybatis</groupId>
        <artifactId>mybatis</artifactId>
        <version>3.5.6</version>
    </dependency>
    <dependency>
```

```
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>4.13</version>
    </dependency>
    <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <version>8.0.23</version>
    </dependency>
    <dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
        <version>1.18.18</version>
    </dependency>
</dependencies>

<build>
    <!--在build中配置resources，来防止我们资源导出失败的问题-->
    <resources>
        <resource>
            <directory>src/main/resources</directory>
            <includes>
                <include>**/*.properties</include>
                <include>**/*.xml</include>
            </includes>
        </resource>
        <resource>
            <directory>src/main/java</directory>
            <includes>
                <include>**/*.properties</include>
                <include>**/*.xml</include>
            </includes>
            <filtering>true</filtering>
        </resource>
    </resources>
</build>
```

3. Mybatis**核心配置文件**

```
<?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>
    <environments default="development">
        <environment id="development">
            <transactionManager type="JDBC"/>
            <dataSource type="POOLED">
```

```
                <property name="driver"
value="com.mysql.jdbc.Driver"/>
                <property name="url"
value="jdbc:mysql://localhost:3306/mybatis?
useSSL=true&amp;useUnicode=true&amp;characterEncoding=utf-
8"/>
                <property name="username" value="root"/>
                <property name="password" value="123456"/>
            </dataSource>
        </environment>
    </environments>
    <!-- 每一个Mapper文件都需要在核心配置文件中配置！ -->
    <mappers>
        <mapper resource="com/wll/dao/UserMapper.xml"/>
    </mappers>
</configuration>
```

4. util**工具类**

```
/**
 * @author wulele
 */
public class MybatisUtil {
    /**
     * 每个基于 MyBatis 的应用都是以一个 SqlSessionFactory 的实例为
核心的。SqlSessionFactory 的实例
     * 可以通过 SqlSessionFactoryBuilder 获得。而
SqlSessionFactoryBuilder 则可以从 XML 配置文件
     * 或一个预先配置的 Configuration 实例来构建出
SqlSessionFactory 实例
     */
    private static SqlSessionFactory sqlSessionFactory;

    static {
        try {
            String resource = "mybaties-config.xml";
            InputStream inputStream =
Resources.getResourceAsStream(resource);
            sqlSessionFactory = new
SqlSessionFactoryBuilder().build(inputStream);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    /**
     * SqlSession 提供了在数据库执行 SQL 命令所需的所有方法。
     * @return SqlSession
     */
```

```
    public SqlSession getSqlSession() {
        return sqlSessionFactory.openSession();
        //可以设置autocommit，设置为自动提交事务，不用
sqlsession.commit()
        //return sqlSessionFactory.openSession(true);
    }
}
```

5. pojo层

```
/**
 * @author wulele
 */
@Getter
@Setter
@NoArgsConstructor
@AllArgsConstructor
@ToString
public class User {
    private int id;
    private String name;
    private String pwd;
}
```

6. UserDao——>UserMapper

```
/**
 * @author wulele
 */
public interface UserDao {
    /**
     * User list
     * @return User list
     */
    public List<User> getUerList();
}
```

7. UserDaoImpl——>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">
<mapper namespace="com.wll.dao.UserDao">
    <select id="getUerList" resultType="com.wll.pojo.User">
        select *
        from mybatis.user
    </select>
</mapper>
```

8. test

```java
public class UserDaoTest {
    @Test
    public void test(){
        //建议写法，保证sqlSession能在finally被关闭
        SqlSession sqlSession = null;
        try {
            sqlSession = new MybatisUtil().getSqlSession();
            UserDao mapper =
sqlSession.getMapper(UserDao.class);
            List<User> userList = mapper.getUerList();
            //不推荐使用
            //List<User> userList =
sqlSession.selectList("com.wll.dao.UserDao.getUerList");
            for (User user : userList) {
                System.out.println(user);
            }
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            sqlSession.close();
        }
    }
}
```

9. **注意点！**

- util工具类

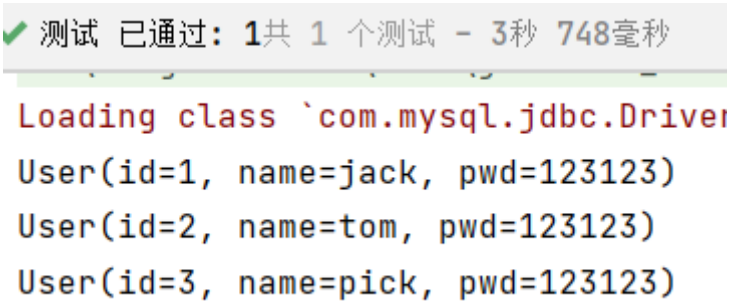
  String resource = "mybaties-config.xml";

  不要写全路径

- 如果自己写的xml在sources文件夹中，不能被打包到发布文件夹，记得pom.xml配置build的resources
- Mapper.xml文件的namespace、resultType写完整包名，id为namespace对应包下的方法

- 每一个Mapper文件都需要在核心配置文件中配置！

10. **运行结果**



增删改查

注意：增删改需要提交事务！

传递多个参数使用Map或者注解

增

```
/**
     * add user
     * @param user
     */
public void addUser(User user);
```

```
<insert id="addUser" parameterType="com.wll.pojo.User">
    insert into mybatis.user(id, name, pwd) value (#{id},#{name},#
{pwd})
</insert>
```

```
@Test
public void addUser(){
    SqlSession sqlSession = new MybatisUtil().getSqlSession();
    UserMapper mapper = sqlSession.getMapper(UserMapper.class);
    mapper.addUser(new User(4,"bob","1234"));
    //提交事务！！！
    sqlSession.commit();
    sqlSession.close();
}
```

删

```
<delete id="delUserById" parameterType="int">
    delete
    from mybatis.user
    where id = #{id}
</delete>
```

改

```
<update id="updateUser" parameterType="com.wll.pojo.User">
    update mybatis.user
    set name = #{name},
    pwd  = #{pwd}
    where id = #{id}
</update>
```

查

```
<select id="getUserById" parameterType="int"
resultType="com.wll.pojo.User">
    select *
    from mybatis.user
    where id = #{id}
</select>
```

配置进阶

配置顺序

配置有先后顺序

environments

可以配置多个`environment`但只能使用一个

使用场景：正式+测试

```
<environments default="test">
    <environment id="development">
        <transactionManager type="JDBC"/>
        <dataSource type="POOLED">
            <property name="driver"
value="com.mysql.jdbc.Driver"/>
            <property name="url"

 value="jdbc:mysql://192.144.231.70:3306/mybatis?
useSSL=true&amp;useUnicode=true&amp;characterEncoding=utf-8"/>
            <property name="username" value="root"/>
            <property name="password" value="123123"/>
        </dataSource>
    </environment>
    <environment id="test">
        <transactionManager type="JDBC"/>
        <dataSource type="POOLED">
            <property name="driver"
value="com.mysql.jdbc.Driver"/>
            <property name="url"

 value="jdbc:mysql://192.144.231.70:3306/mybatis?
useSSL=true&amp;useUnicode=true&amp;characterEncoding=utf-8"/>
            <property name="username" value="root"/>
            <property name="password" value="123123"/>
        </dataSource>
    </environment>
</environments>
```

properties

```
<!--假如外部配置文件也含有username属性，优先使用外部配置文件-->
<properties resource="db.properties">
    <property name="username" value="root"/>
</properties>

<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>
</environments>
```

settings

设置名	描述	有效值	默认值
cacheEnabled	全局性地开启或关闭所有映射器配置文件中已配置的任何缓存。	true | false	true
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。 特定关联关系中可通过设置 `fetchType` 属性来覆盖该项的开关状态。	true | false	false
logImpl	指定 MyBatis 所用日志的具体实现，未指定时将自动查找。	SLF4J | LOG4J | LOG4J2 | JDK_LOGGING | COMMONS_LOGGING | STDOUT_LOGGING | NO_LOGGING	未设置

别名（typeAliases）

1. 指定实体类取别名

```
<typeAliases>
    <typeAlias type="com.wll.pojo.User" alias="User"/>
</typeAliases>
```

2. 扫描包

   扫描实体类所在包，如果不使用注解，其默认别名就是类首字母小写

```
<typeAliases>
    <package name="com.wll.pojo"/>
</typeAliases>
```

   对比，第一种方式可以自定义别名，第二种需要在实体类加注解才可以自定义别名 `@Alias("hello")`

映射器（mappers）

1. 类路径资源引用

```xml
<!-- 使用相对于类路径的资源引用 -->
<mappers>
  <mapper resource="org/mybatis/builder/AuthorMapper.xml"/>
  <mapper resource="org/mybatis/builder/BlogMapper.xml"/>
  <mapper resource="org/mybatis/builder/PostMapper.xml"/>
</mappers>
```

2. mapper**文件完全限定类名**

```xml
<!-- 使用映射器接口实现类的完全限定类名 -->
<mappers>
  <mapper class="org.mybatis.builder.AuthorMapper"/>
  <mapper class="org.mybatis.builder.BlogMapper"/>
  <mapper class="org.mybatis.builder.PostMapper"/>
</mappers>
```

3. mapper**文件包名**

```xml
<!-- 将包内的映射器接口实现全部注册为映射器 -->
<mappers>
  <package name="org.mybatis.builder"/>
</mappers>
```

注意点：第一种使用无特殊要求，第二三种使用需要 接口和mapper**配置文件同名，并且在一个文件夹下！**

结果映射（ResultMap）

如果pojo**层对象名不与数据库列名一一对应，可以使用**ResultMap**来解决。**

```xml
<resultMap id="userResultMap" type="User">
  <id property="id" column="user_id" />
  <result property="username" column="user_name"/>
  <result property="password" column="hashed_password"/>
</resultMap>
```

然后在引用它的语句中设置 `resultMap` 属性就行了（注意我们去掉了 `resultType` 属性）。比如:

```xml
<select id="selectUsers" resultMap="userResultMap">
  select user_id, user_name, hashed_password
  from some_table
  where id = #{id}
</select>
```

日志

标准日志配置

```xml
<settings>
    <setting name="logImpl" value="STDOUT_LOGGING"/>
</settings>
```

```
Created connection 464064894.
Setting autocommit to false on JDBC Connection [com.mysql.cj.jdbc.ConnectionImpl@1ba9117e]
==>  Preparing: select * from mybatis.user where id = ?
==> Parameters: 1(Integer)
<==    Columns: id, name, pwd
<==        Row: 1, jack, 123123
<==      Total: 1
User(id=1, name=jack, pwd=123123)
Resetting autocommit to true on JDBC Connection [com.mysql.cj.jdbc.ConnectionImpl@1ba9117e]
Closing JDBC Connection [com.mysql.cj.jdbc.ConnectionImpl@1ba9117e]
Returned connection 464064894 to pool.
```

LOG4J配置

1. **导包**

```xml
<dependency>
    <groupId>log4j</groupId>
    <artifactId>log4j</artifactId>
    <version>1.2.17</version>
</dependency>
```

2. log4j.properties

```
#将等级为DEBUG的日志信息输出到console和file这两个目的地，console和file的定义在下面的代码
log4j.rootLogger=DEBUG,console,file

#控制台输出的相关设置
log4j.appender.console = org.apache.log4j.ConsoleAppender
log4j.appender.console.Target = System.out
log4j.appender.console.Threshold=DEBUG
log4j.appender.console.layout =
org.apache.log4j.PatternLayout
log4j.appender.console.layout.ConversionPattern=【%c】-%m%n

#文件输出的相关设置
log4j.appender.file = org.apache.log4j.RollingFileAppender
log4j.appender.file.File=./log/wll.log
log4j.appender.file.MaxFileSize=10mb
log4j.appender.file.Threshold=DEBUG
log4j.appender.file.layout=org.apache.log4j.PatternLayout
```

```
log4j.appender.file.layout.ConversionPattern=【%p】【%d{yy-MM-dd}】【%c】%m%n

#日志输出级别
log4j.logger.org.mybatis=DEBUG
log4j.logger.java.sql=DEBUG
log4j.logger.java.sql.Statement=DEBUG
log4j.logger.java.sql.ResultSet=DEBUG
log4j.logger.java.sql.PreparedStatement=DEBUG

# 输出消息编码
log4j.appender.LOGFILE.encoding=UTF-8
```

3. **核心配置文件**

```
<settings>
    <setting name="logImpl" value="LOG4J"/>
</settings>
```

4. **补充**

```
Logger logger = Logger.getLogger(UserMapper.class);
logger.info("开始了");
logger.debug("哦哦哦");
logger.error("错错错");
```

```
"C:\Program Files\Java\jdk1.8.0_
[com.wll.dao.UserMapper]-开始了
[com.wll.dao.UserMapper]-哦哦哦
[com.wll.dao.UserMapper]-错错错
```

5. **注意**

   mybatis**核心配置文件中不要使用或中使用**package**来指定要扫描的包，否则**log**文件打不开**

分页

limit

1. limit**格式**

```
select * from user limit startIndex,pageSize;
```

`example`

```
--从0开始，一页三行
select * from user limit 0,3;
```

2. UserMapper

```
/**
* limit user list
* @param map
* @return List
*/
List<User> limitUserList(Map<String, Integer> map);
```

3. UserMapper.xml

```
<select id="limitUserList" parameterType="map"
resultType="User">
    select *
    from mybatis.user
    limit #{startIndex},#{pageSize}
</select>
```

4. Test

```
@Test
public void limitUserList(){
    SqlSession sqlSession = new
MybatisUtil().getSqlSession();
    UserMapper mapper =
sqlSession.getMapper(UserMapper.class);
    HashMap<String, Integer> map = new HashMap<String,
Integer>();
    map.put("startIndex",0);
    map.put("pageSize",2);
    List<User> users = mapper.limitUserList(map);
    for (User user : users) {
        System.out.println(user);
    }
    sqlSession.close();
}
```

RowBounds

1. UserMapper

```
/**
* rowBounds user list
* @return List
*/
List<User> rowBoundsUserList();
```

2. UserMapper.xml

```
<select id="rowBoundsUserList" resultType="User">
    select *
    from mybatis.user
</select>
```

3. Test

```
@Test
public void rowBoundsUserList(){
    RowBounds rowBounds = new RowBounds(0,2);
    SqlSession sqlSession = new
MybatisUtil().getSqlSession();
    List<User> userList =
sqlSession.selectList("com.wll.dao.UserMapper.rowBoundsUserLi
st", null, rowBounds);
    for (User user : userList) {
        System.out.println(user);
    }
    sqlSession.close();
}
```

Mybatis PageHelper插件

地址：https://pagehelper.github.io/

注解开发

`example`

UserMapper.java

```
@Select("select * from user")
List<User> getUerList();
```

本质：反射机制

底层：动态代理

使用场景：sql**语句较为简单**

增删改查

```
/**
* get User By id
*
* @param id
* @return
*/
@Select("select * from user where id = #{uid}")
User getUserById(@Param("uid") int id);

/**
* add user
*
* @param user
*/
@Insert("insert into user(id,name,pwd) values(#{id},#{name},#
{pwd})")
void addUser(User user);

/**
* update user
*
* @param user
*/
@Update("update user set name = #{name}, pwd = #{pwd} where id = #
{id}")
void updateUser(User user);

/**
* delete user
*
* @param id
*/
@Delete("delete from user where id = #{uid}")
void delUser(@Param("uid") int id);
```

#{}与${}的区别：

#{}**方式能够很大程度防止**sql**注入(安全)，**${}**方式无法防止**Sql**注入**

多对一操作

```
create table student
(
    id   int(10)     not null
    primary key,
    name varchar(30) null,
    tid  int(10)     null,
    constraint fktid
    foreign key (tid) references teacher (id)
);

create table teacher
(
    id   int(10)     not null
    primary key,
    name varchar(30) null
);
```

sql

```
select s.id,s.name,t.name from student s, teacher t where s.tid =
t.id;
```

	id	s.name	t.name
1	1	小明	吴老师
2	2	小红	吴老师
3	3	小张	吴老师
4	4	小李	吴老师
5	5	小王	吴老师

```
public class Student {
    private int id;
    private String name;
    //一个学生对应多个老师
    private Teacher teacher;
}

public class Teacher {
    private int id;
    private String name;
}
```

查询嵌套

StudentMapper

```
/**
* student list
* @return List
*/
List<Student> studentList();
```

StudentMapper.xml

```
<!--查询全部学生信息，学生tid对应老师id-->
<select id="studentList" resultMap="StudentTeacher">
    select * from student
</select>
<!--map集结果映射-->
<resultMap id="StudentTeacher" type="com.wll.pojo.Student">
    <id property="id" column="id"/>
    <id property="name" column="name"/>
    <!--对象处理association，映射老师对象到学生tid属性-->
    <!--集合处理collection-->
    <association property="teacher" column="tid"
select="teacherList" javaType="com.wll.pojo.Teacher"/>
</resultMap>
<!--查询全部老师信息-->
<select id="teacherList" resultType="com.wll.pojo.Teacher">
    select * from teacher where id = #{tid}
</select>
```

结果：

```
Student(id=1, name=小明, teacher=Teacher(id=1, name=吴老师))
Student(id=2, name=小红, teacher=Teacher(id=1, name=吴老师))
Student(id=3, name=小张, teacher=Teacher(id=1, name=吴老师))
Student(id=4, name=小李, teacher=Teacher(id=1, name=吴老师))
Student(id=5, name=小王, teacher=Teacher(id=1, name=吴老师))
```

结果嵌套

```
<select id="studentList2" resultMap="StudentTeacher2">
    select s.id sid, s.name sname, t.name tname, t.id tid
    from student s,
    teacher t
    where s.tid = t.id;
</select>
```

```
<resultMap id="StudentTeacher2" type="com.wll.pojo.Student">
    <id property="id" column="sid"/>
    <id property="name" column="sname"/>
    <!--嵌套结果-->
    <association property="teacher"
javaType="com.wll.pojo.Teacher">
        <!--返回teacher的id和name-->
        <id property="id" column="tid"/>
        <id property="name" column="tname"/>
    </association>
</resultMap>
```

结果：

```
Student(id=1, name=小明, teacher=Teacher(id=1, name=吴老师))
Student(id=2, name=小红, teacher=Teacher(id=1, name=吴老师))
Student(id=3, name=小张, teacher=Teacher(id=1, name=吴老师))
Student(id=4, name=小李, teacher=Teacher(id=1, name=吴老师))
Student(id=5, name=小王, teacher=Teacher(id=1, name=吴老师))
```

一对多

实体类：

```
public class Student {
    private int id;
    private String name;
    private int tid;
}

public class Teacher {
    private int id;
    private String name;
    private List<Student> students;
}
```

结果嵌套

```java
/**
* get Teacher
* @param id
* @return
*/
Teacher getTeacher(@Param("tid") int id);

/**
* get teacher
* @param id
* @return
*/
Teacher getTeacher2(@Param("tid")int id);
```

```xml
<select id="getTeacher" resultMap="TeacherStudent">
    select s.id sid, s.name sname, t.id tid, t.name tname
    from student s,
    teacher t
    where s.tid = t.id
    and t.id = #{tid}
</select>
<resultMap id="TeacherStudent" type="teacher">
    <result property="id" column="tid"/>
    <result property="name" column="tname"/>
    <!--javaType: 指定属性的类型
        集合中的泛型信息，我们使用ofType-->
    <collection property="students" ofType="student">
        <result property="id" column="sid"/>
        <result property="name" column="sname"/>
        <result property="tid" column="tid"/>
    </collection>
</resultMap>
```

结果：

```
Teacher(id=1, name=吴老师,
students=[Student(id=1, name=小明, tid=1),
          Student(id=2, name=小红, tid=1),
          Student(id=3, name=小张, tid=1),
          Student(id=4, name=小李, tid=1),
          Student(id=5, name=小王, tid=1)])
```

查询嵌套

```
<select id="getTeacher2" resultMap="TeacherStudent2">
    select *
    from teacher
    where id = #{tid}
</select>

<resultMap id="TeacherStudent2" type="teacher">
    <result property="id" column="id"/>
    <collection property="students" javaType="ArrayList"
ofType="Student" select="getStudentByTeacherId"
                column="id"/>
</resultMap>

<select id="getStudentByTeacherId" resultType="student">
    select *
    from student
    where tid = #{tid}
</select>
```

动态sql

学习环境

```
create table blog
(
    id          varchar(50)  not null comment '博客id',
    title       varchar(100) not null comment '博客标题',
    author      varchar(30)  not null comment '博客作者',
    create_time datetime     not null comment '创建时间',
    views       int(30)      not null comment '浏览量'
);
```

mybaties-config.xml

```
<!—开启数据库映射到实体类驼峰命名—>
<setting name="mapUnderscoreToCamelCase" value="true"/>
```

pojo

```java
public class Blog {
    private String id;
    private String title;
    private String author;
    private Date createTime;
    private int views;
}
```

BlogMapper

```java
/**
* add blog
* @param blog
*/
void addBlog(Blog blog);
```

BlogMapper.xml

```xml
<insert id="addBlog" parameterType="blog">
    insert into blog(id, title, author, create_time, views)
    values (#{id}, #{title}, #{author}, #{createTime}, #{views})
</insert>
```

UuidUtil

```java
public class UuidUtil {
    public String getId(){
        return UUID.randomUUID().toString().replaceAll("-","");
    }
}
```

Test

```java
public void test(){
    SqlSession sqlSession = new MybatisUtil().getSqlSession();
    BlogMapper mapper = sqlSession.getMapper(BlogMapper.class);
    Blog blog = new Blog();
    blog.setId(new UuidUtil().getId());
    blog.setTitle("study");
    blog.setAuthor("wll");
    blog.setCreateTime(new Date());
    blog.setViews(1000);
    mapper.addBlog(blog);

    blog.setId(new UuidUtil().getId());
```

```
        blog.setTitle("mybatis");
        blog.setCreateTime(new Date());
        mapper.addBlog(blog);

        blog.setId(new UuidUtil().getId());
        blog.setTitle("servlet");
        blog.setCreateTime(new Date());
        mapper.addBlog(blog);

        sqlSession.commit();
        sqlSession.close();
    }
```

结果：

	id	title	author	create_time	views
1	3d14b89488284b0892bfdc51938d1d31	study	wll	2021-04-16 10:45:27	1000
2	7e734923aded40a79f680c19c6e99fb0	mybatis	wll	2021-04-16 10:45:30	1000
3	4c181d0a843c431193922cae6d20a193	servlet	wll	2021-04-16 10:45:30	1000

if

BlogMapper

```
    /**
    * IF find blog
    * @param map
    * @return
    */
    List<Blog> findBlogIF(Map map);
```

BlogMapper.xml

```
    <select id="findBlogIF" parameterType="map" resultType="blog">
        select * from blog where 1=1
        <if test="title != null">
            title = #{title}
        </if>
        <if test="author != null">
            and author = #{author}
        </if>
    </select>
```

Test

```
public void test(){
    SqlSession sqlSession = new MybatisUtil().getSqlSession();
    BlogMapper mapper = sqlSession.getMapper(BlogMapper.class);
    HashMap map = new HashMap();
    map.put("author","wll");
    List<Blog> blogList = mapper.findBlogIF(map);
    for (Blog blog : blogList) {
        System.out.println(blog);
    }
    sqlSession.close();
}
```

结果：

```
Blog(id=3d14b89488284b0892bfdc51938d1d31, title=study, author=wll, createTime=Fri Apr 16 10:45:27 CST 2021, views=1000)
Blog(id=7e734923aded40a79f680c19c6e99fb0, title=mybatis, author=wll, createTime=Fri Apr 16 10:45:30 CST 2021, views=1000)
Blog(id=4c181d0a843c431193922cae6d20a193, title=servlet, author=wll, createTime=Fri Apr 16 10:45:30 CST 2021, views=1000)
```

where & trim & sql片段

消除第一个查询条件为空的情况下第二个查询条件前的AND**或**OR**，类似于**where 1=1

BlogMapper.xml & where

```
<select id="findBlogIF" parameterType="map" resultType="blog">
    select * from blog
    <where>
        <if test="title != null">
            title = #{title}
        </if>
        <if test="author != null">
            and author = #{author}
        </if>
    </where>
</select>
```

BlogMapper.xml & trim

```
<select id="findBlogIF" parameterType="map" resultType="blog">
    select * from blog
    <trim prefix="where" prefixOverrides="AND|OR">
        <if test="title != null">
            title = #{title}
        </if>
        <if test="author != null">
            and author = #{author}
        </if>
    </trim>
</select>
```

BlogMapper.xml & sql片段

`提取公用部分sql`

1. **最好基于单表来定义sql片段**
2. **不要存在where标签**

这个元素可以用来定义可重用的 SQL 代码片段，以便在其它语句中使用。 参数可以静态地（在加载的时候）确定下来，并且可以在不同的 include 元素中定义不同的参数值。比如：

```
<sql id="userColumns"> ${alias}.id,${alias}.username,${alias}.password </sql>
```

这个 SQL 片段可以在其它语句中使用，例如：

```
<select id="selectUsers" resultType="map">
  select
    <include refid="userColumns"><property name="alias" value="t1"/></include>,
    <include refid="userColumns"><property name="alias" value="t2"/></include>
  from some_table t1
    cross join some_table t2
</select>
```

```
<sql id="IF-title-author">
    <if test="title != null">
        title = #{title}
    </if>
    <if test="author != null">
        and author = #{author}
    </if>
</sql>
<select id="findBlogIF" parameterType="map" resultType="blog">
    select * from blog
    <trim prefix="where" prefixOverrides="AND|OR">
        <include refid="IF-title-author"></include>
    </trim>
</select>
```

choose

条件成立就执行，第一个成立就退出后面不执行，以此类推

BlogMapper

```
/**
* choose find blog
* @param map
* @return
*/
List<Blog> findBlogChoose(Map map);
```

BlogMapper.xml & where

```
<select id="findBlogChoose" parameterType="map" resultType="blog">
    select * from blog
    <where>
        <choose>
```

```
            <when test="title != null">
                title = #{title}
            </when>
            <when test="author != null">
                and author = #{author}
            </when>
            <otherwise>
                and views = ${views}
            </otherwise>
        </choose>
    </where>
</select>
```

BlogMapper.xml & trim

```
<select id="findBlogChoose" parameterType="map" resultType="blog">
    select * from blog
    <trim prefix="where" prefixOverrides="AND|OR">
        <choose>
            <when test="title != null">
                title = #{title}
            </when>
            <when test="author != null">
                and author = #{author}
            </when>
            <otherwise>
                and views = ${views}
            </otherwise>
        </choose>
    </trim>
</select>
```

Test

```
public void test(){
    SqlSession sqlSession = new MybatisUtil().getSqlSession();
    BlogMapper mapper = sqlSession.getMapper(BlogMapper.class);
    HashMap map = new HashMap();
    map.put("title","study");
    map.put("author","wll");
    map.put("views",1000);
    List<Blog> blogList = mapper.findBlogChoose(map);
    for (Blog blog : blogList) {
        System.out.println(blog);
    }
    sqlSession.close();
}
```

结果：

```
-==>  Preparing: select * from blog WHERE title = ?
-==> Parameters: study(String)
-<==      Total: 1
```

set

BlogMapper

```
/**
* update blog
* @param map
*/
void updateBlog(Map map);
```

BlogMapper.xml & set

```
<update id="updateBlog" parameterType="map">
    update blog
    <set>
        <if test="title != null">
            title = #{title},
        </if>
        <if test="author != null">
            author = #{author}
        </if>
    </set>
    <where>
        id = #{id}
    </where>
</update>
```

BlogMapper.xml & trim

```
<update id="updateBlog" parameterType="map">
    update blog
    <trim prefix="set" suffixOverrides=",">
        <if test="title != null">
            title = #{title},
        </if>
        <if test="author != null">
            author = #{author}
        </if>
    </trim>
    <where>
        id = #{id}
    </where>
```

```
</update>
```

Test

```
public void test(){
    SqlSession sqlSession = new MybatisUtil().getSqlSession();
    BlogMapper mapper = sqlSession.getMapper(BlogMapper.class);
    HashMap map = new HashMap();
    map.put("title","studying");
    map.put("author","wll");
    map.put("id","3d14b89488284b0892bfdc51938d1d31");
    mapper.updateBlog(map);
    sqlSession.commit();
    sqlSession.close();
}
```

Foreach

	id	title	author	create_time	views
1	1	study	wll	2021-04-16 10:45:27	1000
2	2	mybatis	wll	2021-04-16 10:45:30	1000
3	3	servlet	wll	2021-04-16 10:45:30	1000

sql

```
--两种方式查询id是1，2，3中的信息
select * from blog where id in (1,2,3);
select * from blog where 1 = 1 and (id = 1 or id = 2 or id =3);
```

BlogMapper

```
/**
* dynamic sql
* @param map
* @return
*/
List<Blog> DynamicSql(Map map);
```

BlogMapper.xml

```xml
<select id="DynamicSql" parameterType="map" resultType="blog">
    select * from blog
    <where>
        <!--方式一-->
        <!-- <foreach collection="ids" item="id" open="id in("
separator="," close=")">-->
        <!-- #{id}-->
        <!-- </foreach>-->
        <!--方式二-->
        <foreach collection="ids" item="id" open="and ("
separator="or" close=")">
            id = #{id}
        </foreach>
    </where>
</select>
```

Test

```java
public void test(){
    SqlSession sqlSession = new MybatisUtil().getSqlSession();
    BlogMapper mapper = sqlSession.getMapper(BlogMapper.class);

    HashMap map = new HashMap();
    ArrayList<Integer> ids = new ArrayList<>();
    ids.add(1);
    ids.add(2);
    map.put("ids",ids);

    List<Blog> blogIF = mapper.DynamicSql(map);
    for (Blog blog : blogIF) {
        System.out.println(blog);
    }
    sqlSession.close();
}
```

Mybatis缓存

My Batis**系统中默认定义了两级缓存:一级缓存和二级缓存**

- **默认情况下,只有一级缓存开启。**(Sqlsession**级别的缓存,也称为本地缓存**)
- **二级缓存需要手动开启和配置,他是基于** namespace**级别的缓存**

  为了提高扩展性, MyBatis**定义了缓存接口** Cache**。我们可以通过实现** Cache**接口来自定义二级缓存**

可用的清除策略有：

- `LRU` – 最近最少使用：移除最长时间不被使用的对象。
- `FIFO` – 先进先出：按对象进入缓存的顺序来移除它们。
- `SOFT` – 软引用：基于垃圾回收器状态和软引用规则移除对象。
- `WEAK` – 弱引用：更积极地基于垃圾收集器状态和弱引用规则移除对象。

一级缓存

```
public void myTest(){
    SqlSession sqlSession = new MybatisUtil().getSqlSession();
    UserMapper mapper = sqlSession.getMapper(UserMapper.class);

    User user = mapper.getUserById(1);
    System.out.println(user);
    User user2 = mapper.getUserById(1);
    System.out.println(user2);

    sqlSession.close();
}
```

查询两次id都为1的user，发现之查询了一次

```
[com.wll.dao.UserMapper.getUserById]-==>  Preparing: select * from mybatis.user where id = ?
[com.wll.dao.UserMapper.getUserById]-==> Parameters: 1(Integer)
[com.wll.dao.UserMapper.getUserById]-<==      Total: 1
User(id=1, name=jack, pwd=123123)                                    一次
User(id=1, name=jack, pwd=123123)
```

查询一次id为1，一次为2的user，发现查询了两次

```
[com.wll.dao.UserMapper.getUserById]-==>  Preparing: select * from mybatis.user where id = ?
[com.wll.dao.UserMapper.getUserById]-==> Parameters: 1(Integer)
[com.wll.dao.UserMapper.getUserById]-<==      Total: 1
User(id=1, name=jack, pwd=123123)                                    两次
[com.wll.dao.UserMapper.getUserById]-==>  Preparing: select * from mybatis.user where id = ?
[com.wll.dao.UserMapper.getUserById]-==> Parameters: 2(Integer)
[com.wll.dao.UserMapper.getUserById]-<==      Total: 1
User(id=2, name=tom, pwd=123123)
```

```java
public void myTest(){
    SqlSession sqlSession = new MybatisUtil().getSqlSession();
    UserMapper mapper = sqlSession.getMapper(UserMapper.class);
    User user = mapper.getUserById(1);
    System.out.println(user);
    mapper.updateUser(new User(2,"kkk","123"));
    User user2 = mapper.getUserById(1);
    System.out.println(user2);
    sqlSession.close();
}
```

查询两次同一个人过程中更新另一个人信息，发现查询两次

```
[com.wll.dao.UserMapper.getUserById]-==>  Preparing: select * from mybatis.user where id = ?
[com.wll.dao.UserMapper.getUserById]-==> Parameters: 1(Integer)
[com.wll.dao.UserMapper.getUserById]-<==      Total: 1
User(id=1, name=jack, pwd=123123)
[com.wll.dao.UserMapper.updateUser]-==>  Preparing: update mybatis.user set name = ?, pwd = ? where id = ?
[com.wll.dao.UserMapper.updateUser]-==> Parameters: kkk(String), 123(String), 2(Integer)
[com.wll.dao.UserMapper.updateUser]-<==    Updates: 1
[com.wll.dao.UserMapper.getUserById]-==>  Preparing: select * from mybatis.user where id = ?
[com.wll.dao.UserMapper.getUserById]-==> Parameters: 1(Integer)
[com.wll.dao.UserMapper.getUserById]-<==      Total: 1
User(id=1, name=jack, pwd=123123)
```

总结：

1. **映射语句文件中的所有** select **语句的结果将会被缓存。**
2. **映射语句文件中的所有** insert、update **和** delete **语句会刷新缓存。**
3. **缓存会使用最近最少使用算法（**LRU, Least Recently Used**）算法来清除不需要的缓存。**
4. **缓存不会定时进行刷新（也就是说，没有刷新间隔）。**
5. **主动清理缓存（**sqlSession.clearCache()**）**

二级缓存

mybaties-config.xml

```xml
<!—默认是开启的，显示地表达开启缓存—>
<setting name="cacheEnabled" value="true"/>
```

在想开启二级缓存地Mapper.xml**文件中**

```xml
<cache/>
```

也可扩展

创建了一个 FIFO **缓存，每隔** 60 **秒刷新，最多可以存储结果对象或列表的** 512 **个引用，而且返回的对象被认为是只读的，因此对它们进行修改可能会在不同线程中的调用者产生冲突**

```
<cache
       eviction="FIFO"
       flushInterval="60000"
       size="512"
       readOnly="true"/>
```

注意：若是不配置readOnly="true"**，且**pojo**没有序列化，会报错**Cause: java.io.NotSerializableException: com.wll.pojo.User

Test

```
public void myTest(){
    SqlSession sqlSession = new MybatisUtil().getSqlSession();
    SqlSession sqlSession2 = new MybatisUtil().getSqlSession();
    UserMapper mapper = sqlSession.getMapper(UserMapper.class);
    UserMapper mapper2 = sqlSession2.getMapper(UserMapper.class);

    User user = mapper.getUserById(1);
    System.out.println(user);
    sqlSession.close();

    User user2 = mapper2.getUserById(1);
    System.out.println(user2);
    sqlSession2.close();

}
```

两个sqlSession**，第一个关闭后第二个查询，发现只查询一次（一级缓存存到二级缓存中去了）**

```
[com.wll.dao.UserMapper.getUserById]-==>  Preparing: select * from mybatis.user where id = ?
[com.wll.dao.UserMapper.getUserById]-==> Parameters: 1(Integer)
[com.wll.dao.UserMapper.getUserById]-<==      Total: 1
User(id=1, name=jack, pwd=123123)
[org.apache.ibatis.transaction.jdbc.JdbcTransaction]-Resetting autocommit to true on JDBC Conne
[org.apache.ibatis.transaction.jdbc.JdbcTransaction]-Closing JDBC Connection [com.mysql.cj.jdbc
[org.apache.ibatis.datasource.pooled.PooledDataSource]-Returned connection 1063980005 to pool.
[org.apache.ibatis.io.SerialFilterChecker]-As you are using functionality that deserializes obj
[com.wll.dao.UserMapper]-Cache Hit Ratio [com.wll.dao.UserMapper]: 0.5
User(id=1, name=jack, pwd=123123)
```

一次

Ehcache

pom.xml

```
<dependency>
    <groupId>org.mybatis.caches</groupId>
    <artifactId>mybatis-ehcache</artifactId>
    <version>1.2.1</version>
</dependency>
```

UserMapper.xml

```
<cache type="org.mybatis.caches.ehcache.EhcacheCache"/>
```

mybatis-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<ehcache xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:noNamespaceSchemaLocation="http://ehcache.org/ehcache.xsd"
         updateCheck="false">

    <diskStore path="./tmpdir/Tmp_EhCache/"/>

    <defaultCache
            eternal="false"
            maxElementsInMemory="10000"
            overflowToDisk="false"
            diskPersistent="false"
            timeToIdleSeconds="1800"
            timeToLiveSeconds="259200"
            memoryStoreEvictionPolicy="LRU"/>

    <cache
            name="cloud_user"
            eternal="false"
            maxElementsInMemory="5000"
            overflowToDisk="false"
            diskPersistent="false"
            timeToIdleSeconds="1800"
            timeToLiveSeconds="1800"
            memoryStoreEvictionPolicy="LRU"/>
</ehcache>
```