maven

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>nuc.wcy</groupId>
    <artifactId>frist</artifactId>
    <packaging>pom</packaging>
    <version>1.0-SNAPSHOT</version>
    <modules>
        <module>mybatis2</module>
    </modules>
```

```
    <build>
        <finalName>springmvc-study</finalName>
        <resources>
            <resource>
                <directory>${basedir}/src/main/java</directory>
                <includes>
                    <include>a.xml</include>
                    <include>**/*.xml</include>
<!--                    <include>personMapper.xml</include>-->
                </includes>
            </resource>
            <resource>
                <directory>${basedir}/src/main/resources</directory>
                <includes>
                    <include>**/*.xml</include>
                    <include>personMapper.xml</include>
                </includes>
            </resource>
        </resources>
    </build>
    <dependencies>
        <dependency>
            <groupId>org.mybatis</groupId>
            <artifactId>mybatis</artifactId>
            <version>3.2.2</version>
        </dependency>
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>6.0.6</version>
        </dependency>
    </dependencies>

</project>
```

开始

0.装包

1.配置conf.xml(配置数据库)

```
<?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"><!--运行环境（写环境的id）-->
       <!-- 环境一 -->
        <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/test"/>
                <property name="username" value="root"/>
                <property name="password" value="654321"/>
            </dataSource>
        </environment>
    </environments>
    <mappers>
        <!--        加载映射文件-->
        <mapper resource="personMapper.xml"/>
    </mappers>
</configuration>
```

2.配置数据（mapper.xml）使数据一一对应

```
<?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">

<!-- 映射路径（可以依附到某个文件） -->
<mapper namespace="xxxx.Mapper">
    <!--   resultType:返回类型
    id地址
    parameterType：输入参数的类型
    -->
     <select id="queryStudentByStuno"
parameterType="java.lang.Integer"
resultType="com.wcy.mybatis.entity.Student">
    select * from student where stuno= #{stuno}
  </select>

  <select id="queryAllStudent"
 resultType="com.wcy.mybatis.entity.Student">
    select * from student
```

```
  </select>

</mapper>
```

3.做数组类，测试

```
public class TestMyBatis {
    public static void main(String[] args) throws IOException {
        //加载配置文件
        Reader reader = Resources.getResourceAsReader("conf.xml");
        //SqlSessionFactory -connection
        SqlSessionFactory sessionFactory = new
SqlSessionFactoryBuilder().build(reader);
        //
        SqlSession session = sessionFactory.openSession();
        String statement =
"com.lanjiao.entiy.personMapper.queryPerson";
        Person person = session.selectOne(statement);
        System.out.println(person);
        session.close();
    }
}
```

接口建设

- 使用xml配置
    1. 事务设置

    ```
    <?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.wcy.dao.IUserDao">
        <select id="findAll" resultType="com.wcy.domain.User" >
            select * from user
        </select>

    </mapper>
    ```

    2. 映射接口

    ```
    findAll()
    ```

    3. 编写测试

```
 public class MybatisTest {
     public static void main(String[] args) throws Exception {
         InputStream in =
Resources.getResourceAsStream("SqlMapConfig.xml");
         SqlSessionFactoryBuilder builder = new
SqlSessionFactoryBuilder();
         SqlSessionFactory factory = builder.build(in);
         SqlSession sqlSession = factory.openSession();
         IUserDao userDao =
sqlSession.getMapper(IUserDao.class);
         List<User> users = userDao.findAll();
         for (User user:users){
             System.out.println(user);
         }
     }
```

- 使用注解配置
  1. 在mybatis配置文件

```
     <mappers>
         <mapper class="com.wcy.dao.IUserDao"/>
     </mappers>
```

  2. 在映射接口配置

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

附一：properties配置

```
#db.properties
driver=com.mysql.jdbc.Driver
url=jdbc:mysql://localhost:3306/textjdbc
username = root
password = 654321
```

```
<!-- conf.xml -->
<properties resource="db.properties"/>
    <environment id="development">
    <!-- 事务提交方式 -->
      <transactionManager type="JDBC"/>
      <!-- 数据源类型 -->
      <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>
```

基本使用

增删改查

若输入类型是8中基本类型则使用&{随便写}来记录数据

插入

```
  <insert id="addStudent"
parameterType="com.wcy.mybatis.entity.Student">
  INSERT student(stuno,stuname,stuage,graname) value(#{stuNo},#
{stuName},#{stuAge},#{graName})
  </insert>
```

```
    public static void addStudent() throws IOException {
        Reader reader = Resources.getResourceAsReader("conf.xml");
        SqlSessionFactory sessionFactory = new
SqlSessionFactoryBuilder().build(reader);
        SqlSession session = sessionFactory.openSession();
        String statement = "com.wcy.mybatis.entity.addStudent";
        Student student = new Student(3,"ww",25,"s1");
        int count = session.insert(statement,student);
        System.out.println("增加"+ count+"个学生");
        session.commit();//提交事务
        System.out.println(student);
        session.close();
    }
```

删除

```
  <delete id="deleteStudentByStuno" parameterType="int">
  delete from student where stuno=#{stuno}
  </delete>
```

```
public static void deleteStudent() throws IOException {
        Reader reader = Resources.getResourceAsReader("conf.xml");
        SqlSessionFactory sessionFactory = new
SqlSessionFactoryBuilder().build(reader);
        SqlSession session = sessionFactory.openSession();
        String statement =
"com.wcy.mybatis.entity.deleteStudentByStuno";
        Student student = new Student(3,"ww",25,"s1");
        int count = session.delete(statement,3);
        System.out.println("删除"+ count+"个学生");
        session.commit();//提交事务
        System.out.println(student);
        session.close();
    }
```

修改

```
  <update id ="updateStudentByStuno"
parameterType="com.wcy.mybatis.entity.Student">
  update student set stuname = #{stuName},stuage=#{stuAge},graname=#
{graName} where stuno=#{stuNo}
  </update>
```

```
public static void updateStudent() throws IOException {
        Reader reader = Resources.getResourceAsReader("conf.xml");
        SqlSessionFactory sessionFactory = new
SqlSessionFactoryBuilder().build(reader);
        SqlSession session = sessionFactory.openSession();
        String statement =
"com.wcy.mybatis.entity.updateStudentByStuno";
//      修改人
        Student student = new Student(3,"wwqw",25,"s2");
        int count = session.delete(statement,student);
        System.out.println("修改"+ count+"个学生");
        session.commit();//提交事务
        System.out.println(student);
        session.close();
    }
```

路径的映射(接口的使用)

> - 1.方法名和mapper.xml文件标签的id值相同

> - 2.方法的输入参数和mapper.xml文件中的标签的parameterTyper一致
> - 3.方法的返回值和mapper.xmld的resultType相同
> -
> - namespace的值就是接口的全类名

例:

```
<select id="queryStudentByStuno" parameterType="int"
resultType="student">
    select * from student where stuno= #{stuno}
  </select>
```

对应

```
Student queryStudentByStuno(int stuno);
```

使用

```
    public static void queryStudent() throws IOException {
        Reader reader = Resources.getResourceAsReader("conf.xml");
        SqlSessionFactory sessionFactory = new
SqlSessionFactoryBuilder().build(reader);
        SqlSession session = sessionFactory.openSession();
//      String statement =
"com.wcy.mybatis.entity.studentMapper.queryStudentByStuno";
//      Student student = (Student) session.selectOne(statement,1);
        StudentMapp studentMapper =
session.getMapper(StudentMapp.class);
        Student student = studentMapper.queryStudentByStuno(1);

        System.out.println(student);
        session.close();
    }
```

别名（可以不写类的全名）

使用时（大小写任意）

> conf.xml文件中

```
    <typeAliases>
        <!-- 单个别名 -->
        <typeAlias type="com.nuc.frist.Person" alias="student"/>
        <!--多个定义-->
<package name="com.nuc.frist"/>
    </typeAliases>
```

最后

```
  <select id="queryStudentByStuno" parameterType="int"
resultType="com.wcy.mybatis.entity.Student">
    select * from student where stuno= #{stuno}
  </select>
```

改为

```
<select id="queryStudentByStuno" parameterType="int"
resultType="student">
    select * from student where stuno= #{stuno}
  </select>
```

类型转换

1.做转换器

继承 `BaseTypeHandler` 类

```
public class BooleanAndInConvernt extends BaseTypeHandler{

    /**
     * ps:PreparedStatement对象
     * i：PreparedStatement操作位置
     * para：java值
     * JdbcType：jdbc操作的数据库类型
     */
        //java（boolean）->db(number)
    @Override
    public void setNonNullParameter(PreparedStatement ps, int i,
Boolean parameter, JdbcType jdbcType)
            throws SQLException {
        // TODO Auto-generated method stub
        if(parameter) {
            //1
            ps.setInt(i, 1);
        }else {
            //0
```

```
            ps.setInt(i, 0);
        }
    }

    //通过列拿值
    @Override
    public Boolean getNullableResult(ResultSet rs, String columnName)
throws SQLException {
        // TODO Auto-generated method stub
        int sexNum = rs.getInt(columnName);
//      if(sexNum==1) {
//          return true;
//      }else {
//          return false;
//      }
        return sexNum==1?true:false;
    }

    //通过下标拿值
    @Override
    public Boolean getNullableResult(ResultSet rs, int columnIndex)
throws SQLException {
        // TODO Auto-generated method stub
        int sexNum = rs.getInt(columnIndex);
        return sexNum==1?true:false;
    }

    @Override
    public Boolean getNullableResult(CallableStatement cs, int
columnIndex) throws SQLException {
        // TODO Auto-generated method stub
        int sexNum = cs.getInt(columnIndex);
        return sexNum==1?true:false;
    }


}
```

2.conf.xml使得代码知道转化器在哪

```
    <typeHandlers>
        <typeHandler
handler="com.wcy.mybatis.converter.BooleanAndNumber"
javaType="Boolean" jdbcType="INTEGER"/>
    </typeHandlers>
```

3.在mybatis-config.xml中

```
<!-- 如果使用了类型转换器
  如果类中的属性和数据库的字段类型能够合理识别（String-varchar），则可以使用resultType;否则使用resultMap
  如果类中的属性名和数据库的字段名能够合理识别（stuno-stuNo），则可以使用resultType;否则使用resultMap
   -->
<!-- 查询的转换器 -->
    <select id="queryStudentWithConverter" parameterType="int" resultMap="studentResult">
    select * from student where stuno= #{stuno}
  </select>

   <resultMap type="student" id="studentResult">
        <!-- 分为主键 id和非主键 result-->
        <id property="stuNo" column="stuno"/>
        <result property="stuName" column="stuname" />
        <result property="stuAge" column="stuage" />
        <result property="graName" column="graname" />
        <result property="stuSex" column="stusex" javaType="boolean" jdbcType="INTEGER" />
   </resultMap>

   <!-- 带转换器的增加 -->
   <insert id="addStudentWithConverter" parameterType="student">
   INSERT student(stuno,stuname,stuage,graname,stusex) value(#{stuNo},#{stuName},#{stuAge},#{graName},#{stuSex,javaType=boolean,jdbcType=INTEGER })
   </insert>
```

在面对类与数据库的字段不一致的问题

1.在大小写不一致的情况下mysql不用管

2.sql语句改为(效率高)

```
select 字段 as 属性名 ,..... from user
```

3.映射（须有主键）

例：类中的id对数据库的stuno

```
<resultMap type="student" id="studentmapping">
    <!-- 主键映射 -->
   <id property="id" column="stuno"/>
    <!-- 成员映射 -->
    <result property="stuName" column="stuname" />
        <result property="stuAge" column="stuage" />
        <result property="graName" column="graname" />
        <result property="stuSex" column="stusex" javaType="boolean"
jdbcType="INTEGER" />
   </resultMap>
```

关联查询

一对多，多对一

一对一：

```
//Student
    private int stuNo;
    private String stuName;
    private int stuAge;
    private String graName;
    private boolean stuSex;
    private StudentCard card;
```

```
<!-- studentmapper.xml -->
<select id="queryStudentByNoWith00" parameterType="int"
        resultMap="student_card_map">
  select s.*,c.*  from student s inner join studentcard c
  on s.cardid=c.cardid
  where s.stuno = 1;
    </select>

    <resultMap type="student" id="student_card_map">
        <result property="stuName" column="stuname" />
        <result property="stuAge" column="stuage" />
        <result property="graName" column="graname" />
        <result property="stuSex" column="stusex" javaType="boolean"
            jdbcType="INTEGER" />
        <!-- -->
        <association property="card" javaType="StudentCard">
```

```
            <id property="cardId" column="cardid" />
            <result property="cardinfo" column="cardinfo" />
        </association>
    </resultMap>
```

一对多

```
<!-- studentmapper.xml -->
<select id="queryClassAndStudents" parameterType="int"
        resultMap="calss_student_map">
        select c.*,s.* from student s
        inner join studentclass c
        on
        c.classid = s.classid
        where c.classid = #{classid}
    </select>

    <resultMap type="studentClass" id="calss_student_map">
        <id property="classid" column="classid" />
        <result property="className" column="className" />
        <!-- 配置成员属性学生，一对多 ;属性类型javaType，属性元素类型:
ofType -->
        <collection property="students" ofType="Student">
            <result property="stuName" column="stuname" />
            <result property="stuAge" column="stuage" />
            <result property="graName" column="graname" />
            <result property="stuSex" column="stusex"
javaType="boolean"
                jdbcType="INTEGER" />
            <!--一对一加载学生证，对象成员使用assciation映射 -->
            <association property="card" javaType="StudentCard">
                <id property="cardId" column="cardid" />
                <result property="cardinfo" column="cardinfo" />
            </association>
        </collection>
    </resultMap>
```

延迟加载

注意:记得将其他配置文件放入conf.xml

adfsafasd

一对一

```
<!-- studentmapper.xml -->
<!-- 延迟加载 -->
    <select id="queryStudentByNoWith002"
        resultMap="student_card_lazyLoad_map">
        <!-- 1.获取学生表 -->
        select * from student
    </select>

    <resultMap type="student" id="student_card_lazyLoad_map">
        <result property="stuName" column="stuname" />
        <result property="stuAge" column="stuage" />
        <result property="graName" column="graname" />
        <result property="stuSex" column="stusex" javaType="boolean"
            jdbcType="INTEGER" />
        <!-- 延迟加载 在需要的时候再查学生证-->
        <association property="card" javaType="StudentCard"
select="com.wcy.mybatis.mapper.studentCardMapper.queryCardById"
column="cardid">
        <!-- 立即加载区 -->
            <!-- <id property="cardId" column="cardid" />
            <result property="cardinfo" column="cardinfo" /> -->
        </association>
    </resultMap>
```

```
<!-- StudentCardMapper.xml -->
<mapper namespace="com.wcy.mybatis.mapper.StudentCardMapper">
    <!-- 查询学生证 -->
    <select id="queryCardById" parameterType="int"
resultType="studentCard">
    <!-- 查询学生对应的学生证 -->
    select * from studentCard where cardid = #{cardid}
    </select>
</mapper>
```

缓存（未完成）

mybatis会自动存储一级缓存

日志

log4j.jar

开启日志

```
#log4j.properties
log4j.rootLogger=DEBUG, Console
#Console
log4j.appender.Console=org.apache.log4j.ConsoleAppender
log4j.appender.Console.layout=org.apache.log4j.PatternLayout
log4j.appender.Console.layout.ConversionPattern=%d [%t] %-5p [%c] -
%m%n
log4j.logger.java.sql.ResultSet=INFO
log4j.logger.org.apache=INFO
log4j.logger.java.sql.Connection=DEBUG
log4j.logger.java.sql.Statement=DEBUG
log4j.logger.java.sql.PreparedStatement=DEBUG
```

日志级别：

DEBUG<INFO<WARN<ERROR

若设置为info，则只显示info以上的级别的信息

使用

```
<!-- 日志 -->
    <settings>
        <setting name="logImpl" value="LOG4J"/>
        <!-- 开启延迟 -->
        <setting name="lazyLoadingEnabled" value="true"/>
        <!-- 关闭立即 -->
        <setting name="aggressiveLazyLoading" value="false"/>
    </settings>
```

其他

8个基本类型

输入参数：parameterType

1.类型为 简单类型（8个基本类型+Srting）

- `#{任意值}`

  `${value}`标识符只能是value

- 如果类型是string, `#{}` 会自动加 `''` , `${}` 原样输出，一般用于动态排序 要是想加用 `'${}'`

  例： `select * from student order ${value} asc` 根据什么排序自己设

- #{}可以防止SQL注入，${}不可以防止SQL注入

  比如正则

2.对象类型

- `#{属性名}`

  `` `${属性名} ``