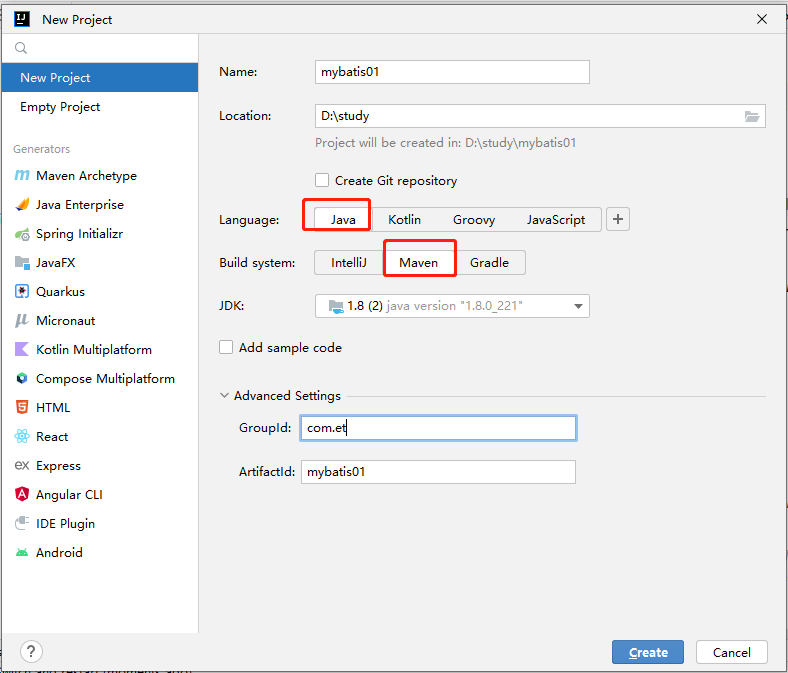
MYBATIS3

# DAY1

## 搭建项目

### 创建maven项目



### 添加mybatis支持

pom.xml

|  |
| --- |
| <dependencies>  <!-- 添加mybatis支持 -->  <dependency>  <groupId>org.mybatis</groupId>  <artifactId>mybatis</artifactId>  <version>3.3.0</version>  </dependency>  <!-- jdbc驱动包 -->  <dependency>  <groupId>mysql</groupId>  <artifactId>mysql-connector-java</artifactId>  <version>5.1.37</version>  </dependency>  </dependencies> |

### 创建mybatis配置文件

mybatis-config.xml

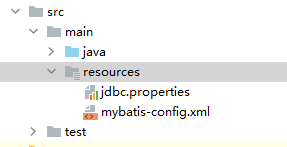
|  |
| --- |
| <?xml version="1.0" encoding="UTF-8" ?>  <!DOCTYPE configuration  PUBLIC "-//mybatis.org//DTD Config 3.0//EN"  "http://mybatis.org/dtd/mybatis-3-config.dtd">  <configuration>  <properties resource="jdbc.properties"/>  <!-- 别名 -->  <typeAliases>  <!-- type 指定需要映射的实体类 包名+类名  alias 别名  -->  <typeAlias type="com.etjava.model.Student" alias="Student"/>  </typeAliases>  <!-- 配置环境 例如 development开发环境-->  <environments default="development">  <environment id="development">  <!-- 事务管理 -->  <transactionManager type="JDBC"/>  <!-- 数据源 POOLED连接池 -->  <dataSource type="POOLED">  <property name="driver" value="${jdbc.driverClassName}"/>  <property name="url" value="${jdbc.url}"/>  <property name="username" value="${jdbc.username}"/>  <property name="password" value="${jdbc.password}"/>  </dataSource>  </environment>  </environments>  <!-- 映射器 映射实体类与数据库SQL -->  <mappers>  <mapper resource="com/etjava/mappers/StudentMapper.xml"/>  </mappers>  </configuration> |

### 添加数据库连接配置文件

jdbc.properties

|  |
| --- |
| jdbc.driverClassName=com.mysql.jdbc.Driver  jdbc.url=jdbc:mysql://localhost:3306/db\_mybatis?characterEncoding=utf-8  jdbc.username=root  jdbc.password=Karen@1234 |

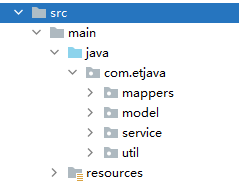
### 目录结构



## HelloWorld

### 创建包

src目录下创建如下包结构



### 创建数据库及表

|  |
| --- |
| CREATE TABLE `t\_student` (    `id` int(11) NOT NULL AUTO\_INCREMENT,    `stuName` varchar(32) DEFAULT NULL,    `age` int(11) DEFAULT NULL,    PRIMARY KEY (`id`)  ) ENGINE=InnoDB AUTO\_INCREMENT=2 DEFAULT CHARSET=latin1 |

### 创建SqlSessionFactoryUtil工具类

该工具类用来封装连接数据库的 存放在util包下

|  |
| --- |
| package com.etjava.util;  import java.io.InputStream;  import org.apache.ibatis.io.Resources; import org.apache.ibatis.session.SqlSession; import org.apache.ibatis.session.SqlSessionFactory; import org.apache.ibatis.session.SqlSessionFactoryBuilder;  */\*\*  \* 获取数据库连接 操作数据库工具类  \*  \*/* public class SqlSessionFactoryUtil {   private static SqlSessionFactory *sqlSessionFactory*;   *// 创建SqlSessionFactory* private static SqlSessionFactory getSqlSessionFactory() {  if (*sqlSessionFactory* == null) {  InputStream is = null;  try {  is = Resources.*getResourceAsStream*("mybatis-config.xml");  *sqlSessionFactory* = new SqlSessionFactoryBuilder().build(is);  } catch (Exception e) {  e.printStackTrace();  }  }  return *sqlSessionFactory*;  }   *// 打开连接* public static SqlSession openSession() {  return *getSqlSessionFactory*().openSession();  }   public static void main(String[] args) {  *openSession*();  } } |

### model包下创建实体类

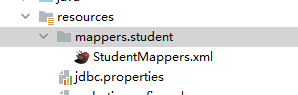
|  |
| --- |
| package com.etjava.model;  public class Student {  private Integer id;  private String stuName;  private Integer age;   public Student() {  super();  }  public Student(String stuName, Integer age) {  super();  this.stuName = stuName;  this.age = age;  }  public Integer getId() {  return id;  }  public void setId(Integer id) {  this.id = id;  }  public String getStuName() {  return stuName;  }  public void setStuName(String stuName) {  this.stuName = stuName;  }  public Integer getAge() {  return age;  }  public void setAge(Integer age) {  this.age = age;  } } |

### mapper包下创建StudentMapper接口

|  |
| --- |
| package com.etjava.mappers;  import com.etjava.model.Student;  */\*\*  \* Student Mapper接口  \*/* public interface StudentMappers {  */\*\*  \* 添加学生信息  \* @param stu  \* @return  \*/* public int add(Student stu); } |

### resources目录下创建接口的映射文件

StudentMapper.xml



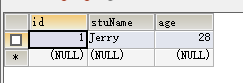
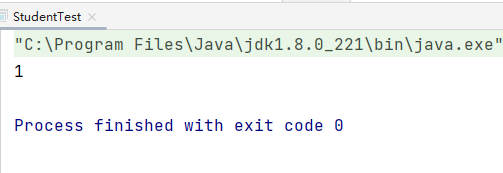
### StudentMappers.xml文件内容

|  |
| --- |
| *<?*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 namespace 对应的是接口的完整名称-->* <mapper namespace="com.etjava.mappers.StudentMappers">  *<!-- id 需要与接口中的方法名保持一直  parameterType 入参  Student 对应的是mybatis-config中的别名 如果那边没有配置别名  这里需要写完整的类名 包名+类名  -->* <insert id="add" parameterType="Student">  insert into t\_student values (null,#{stuName},#{age})  </insert> </mapper> |

### service包下添加测试类

|  |
| --- |
| package com.etjava.service;  import com.etjava.mappers.StudentMappers; import org.apache.ibatis.session.SqlSession;  import com.etjava.model.Student; import com.etjava.util.SqlSessionFactoryUtil;  public class StudentTest {   public static void main(String[] args) {  *// 获取SqlSession* SqlSession sqlSession = SqlSessionFactoryUtil.*openSession*();  *// 获取Mapper mybatis帮我们做的映射接口* StudentMappers studentMapper = sqlSession.getMapper(StudentMappers.class);  *// 添加数据到数据库* Student stu = new Student("Jerry",28);  int res = studentMapper.add(stu);  *// 执行完成需要手动提交事务* sqlSession.commit();  System.*out*.println(res);  } } |

执行测试结果



# DAY2

## Mybatis多环境配置

|  |
| --- |
| <!-- 配置环境 例如 development开发环境-->  <environments default="development">  <environment id="development">  <!-- 事务管理 -->  <transactionManager type="JDBC"/>  <!-- 数据源 POOLED连接池 -->  <dataSource type="POOLED">  <property name="driver" value="${jdbc.driverClassName}"/>  <property name="url" value="${jdbc.url}"/>  <property name="username" value="${jdbc.username}"/>  <property name="password" value="${jdbc.password}"/>  </dataSource>  </environment>  <!-- 测试环境 -->  <environment id="test">  <!-- 事务管理 -->  <transactionManager type="JDBC"/>  <!-- 数据源 POOLED连接池 -->  <dataSource type="POOLED">  <property name="driver" value="${jdbc.driverClassName}"/>  <property name="url" value="${jdbc.url}"/>  <property name="username" value="${jdbc.username}"/>  <property name="password" value="${jdbc.password}"/>  </dataSource>  </environment>  </environments> |

### transactionManager 事务管理器

MyBatis支持两种事务管理 JDBC和MANAGED(托管)

JDBC 是由应用程序负责管理数据库连接的生命周期

MANAGED 是由应用服务器负责管理数据库连接的声明周期（只有商业版的应用服务器才有此功能，例如Jboss ，WebLogic等）

配置transactionManager 需要在environment具体的某个环境下进行配置

### dataSource数据源配置

dataSource是用来配置数据库连接相关的信息 类型有 UNPOOLED POOLED JNDI druid

UNPOOLED 没有连接池，每次操作数据库 MyBatis都会新建一个连接，用完后关闭，适合小并发量项目

POOLED 有连接池

JNDI 使用应用服务器的配置 通过JNDI获取数据库连接

druid 阿里巴巴连接池

dataSource的使用同样需要配置在每个具体的环境下(environment)

### properties 配置属性

可以直接用来关联外部的properties文件也可以进行配置

除了上述方式外 还可以直接在配置文件中进行配置

|  |
| --- |
| <!--<properties resource="jdbc.properties"/>-->  <!-- 配置属性 -->  <properties>  <property name="jdbc.driverClassName" value="com.mysql.jdbc.Driver"/>  <property name="jdbc.url" value="jdbc:mysql://localhost:3306/db\_mybatis?characterEncoding=utf-8"/>  <property name="jdbc.username" value="root"/>  <property name="jdbc.password" value="Karen@1234"/>  </properties> |

### typeAlias 给类的完整名称取别名 方便使用

单个类取别名时 需要指定到类 另外 别名可以自定义

扫描包下的所有类进行取别名时 别名就是类的名字

|  |
| --- |
| 单个类取别名  <typeAliases>  <!-- type 指定需要映射的实体类 包名+类名 alias 别名 -->  <typeAlias type="com.etjava.model.Student" alias="Student"/>  </typeAliases>  扫描指定包下的所有类取别名  <typeAliases>  <!--  扫描该包下的所有类进行取别名  -->  <package name="com.etjava.model"/>  </typeAliases> |

### mappers 引入映射文件

所有的ORM框架都会有对应的映射

mappers中可以配置单个接口的映射文件 也可以通过扫描的形式配置多个

|  |
| --- |
| 单个映射文件配置  <!-- 映射器 映射实体类与数据库SQL -->  <mappers>  <mapper resource="mappers/student/StudentMapper.xml"/>  </mappers>  扫描指定包下的多个映射文件  <!-- 映射器 映射实体类与数据库SQL -->  <mappers>  // 接口映射文件需要与接口在同一个包下才可以被扫描到  <package name="com.etjava.mappers"/>  <!--  扫描包时 如果是放在resources目录下的话 必须与mapper接口报名字保持一致 否则编译后无法将其放入到 src目录下对应的包中  如果你的接口映射文件与接口文件在同一个包下 直接扫描即可  Eclipse中不会存在这些问题 直接使用即可  -->  <package name="com.etjava.mappers"/>  </mappers> |

### Log4J配置

#### 依赖

|  |
| --- |
| <dependency>  <groupId>log4j</groupId>  <artifactId>log4j</artifactId>  <version>1.2.17</version>  </dependency> |

#### 添加Log4J的配置文件

resources目录下

log4j.properties

|  |
| --- |
| log4j.rootLogger=debugger,appender1,appender2 log4j.appender.appender1.encoding=UTF-8 log4j.appender.File.encoding=UTF-8 log4j.appender.appender1=org.apache.log4j.ConsoleAppender log4j.appender.appender2=org.apache.log4j.FileAppender log4j.appender.appender2.File=D:/mybatis.log  log4j.appender.appender1.layout=org.apache.log4j.TTCCLayout log4j.appender.appender2.layout=org.apache.log4j.TTCCLayout |

#### 测试log4j的使用

|  |
| --- |
| package com.etjava.service;  import org.apache.ibatis.session.SqlSession;  import com.etjava.mappers.StudentMapper;  import com.etjava.model.Student;  import com.etjava.util.SqlSessionFactoryUtil;  import org.apache.log4j.Logger;  public class StudentTest {  private static Logger logger = Logger.getLogger(StudentTest.class);  public static void main(String[] args) {  // 获取SqlSession  SqlSession sqlSession = SqlSessionFactoryUtil.openSession();  // 获取Mapper mybatis帮我们做的映射接口  StudentMapper studentMapper = sqlSession.getMapper(StudentMapper.class);  // 添加数据到数据库  Student stu = new Student("Jerry",21);  int res = studentMapper.add(stu);  // 执行完成需要手动提交事务  sqlSession.commit();  logger.info("添加成功");  System.out.println(res);  }  } |

# DAY3

## XML方式配置SQL映射器

### INSERT映射语句

#### StudentMapper接口中添加add方法

|  |
| --- |
| public interface StudentMapper {  /\*\*  \* 添加学生信息  \* @param stu  \* @return  \*/  public int add(Student stu);  } |

#### StudentMapper.xml映射文件中添加对应的映射SQL

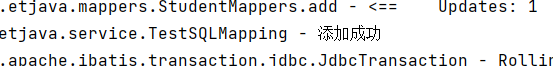
|  |
| --- |
| <?xml version="1.0" encoding="UTF-8" ?>  <!DOCTYPE mapper  PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"  "http://mybatis.org/dtd/mybatis-3-mapper.dtd">  <!-- 映射 namespace 对应的是接口的完整名称-->  <mapper namespace="com.etjava.mappers.StudentMapper">  <!-- id 需要与接口中的方法名保持一直  parameterType 入参  Student 对应的是mybatis-config中的别名 如果那边没有配置别名  这里需要写完整的类名 包名+类名  -->  <insert id="add" parameterType="Student">  insert into t\_student values (null,#{stuName},#{age})  </insert>  </mapper> |

#### 添加JUnit单元测试依赖

|  |
| --- |
| <dependency>  <groupId>junit</groupId>  <artifactId>junit</artifactId>  <version>4.12</version> </dependency> |

#### 测试

|  |
| --- |
| package com.etjava.service;  import com.etjava.mappers.StudentMappers; import com.etjava.model.Student; import com.etjava.util.SqlSessionFactoryUtil; import org.apache.ibatis.session.SqlSession; import org.apache.log4j.Logger; import org.junit.After; import org.junit.Before; import org.junit.Test;  *// 测试SQL映射器* public class TestSQLMapping {  private static Logger *logger* = Logger.*getLogger*(TestSQLMapping.class);  *// 定义SQLSession 用来操作数据库* SqlSession sqlSession = null;  *// 获取Student映射文件* StudentMappers studentMapper = null;  *// 测试方法执行之前* @Before  public void setUp(){  sqlSession = SqlSessionFactoryUtil.*openSession*();  studentMapper = sqlSession.getMapper(StudentMappers.class);  }   *// 测试方法执行之后* @After  public void tearDown(){  sqlSession.close();  }   *// 添加学生* @Test  public void teseAdd(){  int res = studentMapper.add(new Student("Jerry", 13));  if(res>0){  *logger*.info("添加成功");  }else{  *logger*.info("添加失败");  }  } } |



### UPDATE映射语句

#### StudentMapper接口中添加update方法

|  |
| --- |
| package com.etjava.mappers;  import com.etjava.model.Student;  */\*\*  \* Student Mapper接口  \*/* public interface StudentMappers {  */\*\*  \* 添加学生信息  \* @param stu  \* @return  \*/* public int add(Student stu);   */\*\*  \* 修改学生信息  \* @param stu  \* @return  \*/* public int update(Student stu); } |

#### StudentMapper.xml中添加SQL映射

|  |
| --- |
| *<?*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 namespace 对应的是接口的完整名称-->* <mapper namespace="com.etjava.mappers.StudentMappers">  *<!-- id 需要与接口中的方法名保持一直  parameterType 入参  Student 对应的是mybatis-config中的别名 如果那边没有配置别名  这里需要写完整的类名 包名+类名  -->* <insert id="add" parameterType="Student">  insert into t\_student values (null,#{stuName},#{age})  </insert>   *<!--修改学生信息SQL映射-->* <update id="update" parameterType="Student">  update t\_student set stuName=#{stuName},age=#{age} where id=#{id}  </update> </mapper> |

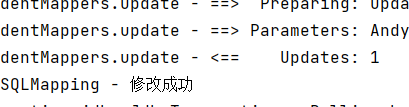
#### Student实体类中添加三个参数的构造方法

|  |
| --- |
| package com.etjava.model;  public class Student {  private Integer id;  private String stuName;  private Integer age;   public Student() {  super();  }   public Student(Integer id, String stuName, Integer age) {  this.id = id;  this.stuName = stuName;  this.age = age;  }   public Student(String stuName, Integer age) {  super();  this.stuName = stuName;  this.age = age;  }  public Integer getId() {  return id;  }  public void setId(Integer id) {  this.id = id;  }  public String getStuName() {  return stuName;  }  public void setStuName(String stuName) {  this.stuName = stuName;  }  public Integer getAge() {  return age;  }  public void setAge(Integer age) {  this.age = age;  } } |

#### 测试

TestSQLMapping

|  |
| --- |
| package com.etjava.service;  import com.etjava.mappers.StudentMappers; import com.etjava.model.Student; import com.etjava.util.SqlSessionFactoryUtil; import org.apache.ibatis.session.SqlSession; import org.apache.log4j.Logger; import org.junit.After; import org.junit.Before; import org.junit.Test;  *// 测试SQL映射器* public class TestSQLMapping {  private static Logger *logger* = Logger.*getLogger*(TestSQLMapping.class);  *// 定义SQLSession 用来操作数据库* SqlSession sqlSession = null;  *// 获取Student映射文件* StudentMappers studentMapper = null;  *// 测试方法执行之前* @Before  public void setUp(){  sqlSession = SqlSessionFactoryUtil.*openSession*();  studentMapper = sqlSession.getMapper(StudentMappers.class);  }   *// 测试方法执行之后* @After  public void tearDown(){  sqlSession.close();  }   *// 添加学生* @Test  public void teseAdd(){  int res = studentMapper.add(new Student("Jerry", 13));  if(res>0){  *logger*.info("添加成功");  }else{  *logger*.info("添加失败");  }  }   *// 修改学生信息* @Test  public void teseUpdate(){  int res = studentMapper.update(new Student(1, "Andy", 12));  if(res>0){  *logger*.info("添加成功");  }else{  *logger*.info("添加失败");  }  } } |



### DELETE映射语句

#### StudentMapper接口中添加删除方法

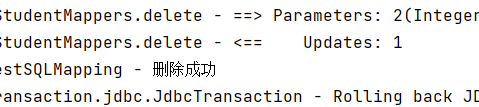
|  |
| --- |
| package com.etjava.mappers;  import com.etjava.model.Student;  */\*\*  \* Student Mapper接口  \*/* public interface StudentMappers {  */\*\*  \* 添加学生信息  \* @param stu  \* @return  \*/* public int add(Student stu);   */\*\*  \* 修改学生信息  \* @param stu  \* @return  \*/* public int update(Student stu);   */\*\*  \* 删除学生信息  \* @param id  \* @return  \*/* public int delete(Integer id); } |

#### StudentMapper.xml中添加删除的映射语句

|  |
| --- |
| *<?*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 namespace 对应的是接口的完整名称-->* <mapper namespace="com.etjava.mappers.StudentMappers">  *<!-- id 需要与接口中的方法名保持一直  parameterType 入参  Student 对应的是mybatis-config中的别名 如果那边没有配置别名  这里需要写完整的类名 包名+类名  -->* <insert id="add" parameterType="Student">  insert into t\_student values (null,#{stuName},#{age})  </insert>   *<!--修改学生信息SQL映射-->* <update id="update" parameterType="Student">  update t\_student set stuName=#{stuName},age=#{age} where id=#{id}  </update>  *<!--删除学生信息-->* <delete id="delete" parameterType="Integer">  delete from t\_student where id=#{id}  </delete> </mapper> |

#### 测试

|  |
| --- |
| package com.etjava.service;  import com.etjava.mappers.StudentMappers; import com.etjava.model.Student; import com.etjava.util.SqlSessionFactoryUtil; import org.apache.ibatis.session.SqlSession; import org.apache.log4j.Logger; import org.junit.After; import org.junit.Before; import org.junit.Test;  *// 测试SQL映射器* public class TestSQLMapping {  private static Logger *logger* = Logger.*getLogger*(TestSQLMapping.class);  *// 定义SQLSession 用来操作数据库* SqlSession sqlSession = null;  *// 获取Student映射文件* StudentMappers studentMapper = null;  *// 测试方法执行之前* @Before  public void setUp(){  sqlSession = SqlSessionFactoryUtil.*openSession*();  studentMapper = sqlSession.getMapper(StudentMappers.class);  }   *// 测试方法执行之后* @After  public void tearDown(){  sqlSession.close();  }   *// 添加学生* @Test  public void teseAdd(){  int res = studentMapper.add(new Student("Jerry", 13));  if(res>0){  *logger*.info("添加成功");  }else{  *logger*.info("添加失败");  }  }   *// 修改学生信息* @Test  public void teseUpdate(){  int res = studentMapper.update(new Student(1, "Andy", 12));  if(res>0){  *logger*.info("修改成功");  }else{  *logger*.info("修改失败");  }  }  *// 删除学生信息* @Test  public void testDelete(){  int res = studentMapper.delete(2);  if(res>0){  *logger*.info("删除成功");  }else{  *logger*.info("删除失败");  }  } } |



# DAY4

## SELECT映射语句

### 根据ID查询学生信息 返回学生对象

#### StudentMapper接口中添加根据ID查询的方法

|  |
| --- |
| package com.etjava.mappers;  import com.etjava.model.Student;  */\*\*  \* Student Mapper接口  \*/* public interface StudentMappers {  */\*\*  \* 添加学生信息  \* @param stu  \* @return  \*/* public int add(Student stu);   */\*\*  \* 修改学生信息  \* @param stu  \* @return  \*/* public int update(Student stu);   */\*\*  \* 删除学生信息  \* @param id  \* @return  \*/* public int delete(Integer id);   */\*\*  \* 根据ID查询学生信息  \* @param id  \* @return  \*/* public Student findById(Integer id); } |

#### StudentMapper映射文件中添加映射语句

|  |
| --- |
| *<?*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 namespace 对应的是接口的完整名称-->* <mapper namespace="com.etjava.mappers.StudentMappers">   *<!--根据ID查询学生信息-->* <select id="findById" parameterType="Integer" resultType="Student">  select \* from t\_student where id=#{id}  </select>   *<!-- id 需要与接口中的方法名保持一直  parameterType 入参  Student 对应的是mybatis-config中的别名 如果那边没有配置别名  这里需要写完整的类名 包名+类名  -->* <insert id="add" parameterType="Student">  insert into t\_student values (null,#{stuName},#{age})  </insert>   *<!--修改学生信息SQL映射-->* <update id="update" parameterType="Student">  update t\_student set stuName=#{stuName},age=#{age} where id=#{id}  </update>  *<!--删除学生信息-->* <delete id="delete" parameterType="Integer">  delete from t\_student where id=#{id}  </delete> </mapper> |

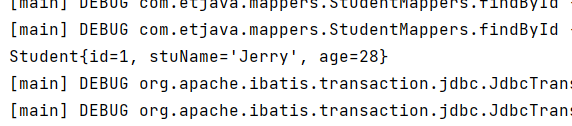
#### Student实体类添加toString方法

方便测试

|  |
| --- |
| package com.etjava.model;  public class Student {  private Integer id;  private String stuName;  private Integer age;   public Student() {  super();  }   public Student(Integer id, String stuName, Integer age) {  this.id = id;  this.stuName = stuName;  this.age = age;  }   public Student(String stuName, Integer age) {  super();  this.stuName = stuName;  this.age = age;  }  public Integer getId() {  return id;  }  public void setId(Integer id) {  this.id = id;  }  public String getStuName() {  return stuName;  }  public void setStuName(String stuName) {  this.stuName = stuName;  }  public Integer getAge() {  return age;  }  public void setAge(Integer age) {  this.age = age;  }   @Override  public String toString() {  return "Student{" +  "id=" + id +  ", stuName='" + stuName + '\'' +  ", age=" + age +  '}';  } } |

#### 测试

|  |
| --- |
| package com.etjava.service;  import com.etjava.mappers.StudentMappers; import com.etjava.model.Student; import com.etjava.util.SqlSessionFactoryUtil; import org.apache.ibatis.session.SqlSession; import org.apache.log4j.Logger; import org.junit.After; import org.junit.Before; import org.junit.Test;  *// 测试SQL映射器* public class TestSQLMapping {  private static Logger *logger* = Logger.*getLogger*(TestSQLMapping.class);  *// 定义SQLSession 用来操作数据库* SqlSession sqlSession = null;  *// 获取Student映射文件* StudentMappers studentMapper = null;  *// 测试方法执行之前* @Before  public void setUp(){  sqlSession = SqlSessionFactoryUtil.*openSession*();  studentMapper = sqlSession.getMapper(StudentMappers.class);  }   *// 测试方法执行之后* @After  public void tearDown(){  sqlSession.close();  }   *// 添加学生* @Test  public void teseAdd(){  int res = studentMapper.add(new Student("Jerry", 13));  if(res>0){  *logger*.info("添加成功");  }else{  *logger*.info("添加失败");  }  }   *// 修改学生信息* @Test  public void teseUpdate(){  int res = studentMapper.update(new Student(1, "Andy", 12));  if(res>0){  *logger*.info("修改成功");  }else{  *logger*.info("修改失败");  }  }  *// 删除学生信息* @Test  public void testDelete(){  int res = studentMapper.delete(2);  if(res>0){  *logger*.info("删除成功");  }else{  *logger*.info("删除失败");  }  }   *// 根据ID查询学生信息* @Test  public void testFindById(){  Student stu = studentMapper.findById(1);  System.*out*.println(stu);  } } |



### 模糊查询 返回集合数据

需要注意 返回的是集合数据

#### StudentMapper接口中定义根据姓名查询

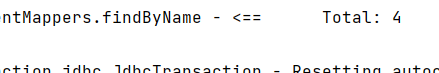
|  |
| --- |
| package com.etjava.mappers;  import com.etjava.model.Student;  import java.util.List;  */\*\*  \* Student Mapper接口  \*/* public interface StudentMappers {  */\*\*  \* 添加学生信息  \* @param stu  \* @return  \*/* public int add(Student stu);   */\*\*  \* 修改学生信息  \* @param stu  \* @return  \*/* public int update(Student stu);   */\*\*  \* 删除学生信息  \* @param id  \* @return  \*/* public int delete(Integer id);   */\*\*  \* 根据ID查询学生信息  \* @param id  \* @return  \*/* public Student findById(Integer id);   *// 根据姓名模糊匹配学生信息* public List<Student> findByName(String name); } |

#### StudentMapper.xml中定义查询SQL

|  |
| --- |
| *<?*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 namespace 对应的是接口的完整名称-->* <mapper namespace="com.etjava.mappers.StudentMappers">   *<!--定义返回的结果集-->* <resultMap id="StudentMap" type="Student">  <id property="id" column="id"/>  <result property="stuName" column="stuName"/>  <result property="age" column="age"/>  </resultMap>   *<!--根据姓名模糊匹配学生信息-->* <select id="findByName" parameterType="String" resultMap="StudentMap">  select \* from t\_student where stuName like concat('%',#{name},'%');  </select>   *<!--根据ID查询学生信息-->* <select id="findById" parameterType="Integer" resultType="Student">  select \* from t\_student where id=#{id}  </select>   *<!-- id 需要与接口中的方法名保持一直  parameterType 入参  Student 对应的是mybatis-config中的别名 如果那边没有配置别名  这里需要写完整的类名 包名+类名  -->* <insert id="add" parameterType="Student">  insert into t\_student values (null,#{stuName},#{age})  </insert>   *<!--修改学生信息SQL映射-->* <update id="update" parameterType="Student">  update t\_student set stuName=#{stuName},age=#{age} where id=#{id}  </update>  *<!--删除学生信息-->* <delete id="delete" parameterType="Integer">  delete from t\_student where id=#{id}  </delete> </mapper> |

#### 测试

|  |
| --- |
| package com.etjava.service;  import com.etjava.mappers.StudentMappers; import com.etjava.model.Student; import com.etjava.util.SqlSessionFactoryUtil; import org.apache.ibatis.session.SqlSession; import org.apache.log4j.Logger; import org.junit.After; import org.junit.Before; import org.junit.Test;  import java.util.List;  *// 测试SQL映射器* public class TestSQLMapping {  private static Logger *logger* = Logger.*getLogger*(TestSQLMapping.class);  *// 定义SQLSession 用来操作数据库* SqlSession sqlSession = null;  *// 获取Student映射文件* StudentMappers studentMapper = null;  *// 测试方法执行之前* @Before  public void setUp(){  sqlSession = SqlSessionFactoryUtil.*openSession*();  studentMapper = sqlSession.getMapper(StudentMappers.class);  }   *// 测试方法执行之后* @After  public void tearDown(){  sqlSession.close();  }   *// 添加学生* @Test  public void teseAdd(){  int res = studentMapper.add(new Student("Jerry", 13));  if(res>0){  *logger*.info("添加成功");  }else{  *logger*.info("添加失败");  }  }   *// 修改学生信息* @Test  public void teseUpdate(){  int res = studentMapper.update(new Student(1, "Andy", 12));  if(res>0){  *logger*.info("修改成功");  }else{  *logger*.info("修改失败");  }  }  *// 删除学生信息* @Test  public void testDelete(){  int res = studentMapper.delete(2);  if(res>0){  *logger*.info("删除成功");  }else{  *logger*.info("删除失败");  }  }   *// 根据ID查询学生信息* @Test  public void testFindById(){  Student stu = studentMapper.findById(1);  System.*out*.println(stu);  }   *// 根据姓名模糊匹配学生信息* @Test  public void testFindByName(){  List<Student> stuList = studentMapper.findByName("Jerry");  System.*out*.println(stuList.size());  } } |



# DAY5

## 分页查询返回集合数据

### StudentMapper接口中定义查询所有学生信息方法 带分页

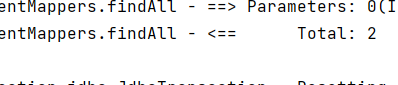
|  |
| --- |
| package com.etjava.mappers;  import com.etjava.model.Student;  import java.util.List;  */\*\*  \* Student Mapper接口  \*/* public interface StudentMappers {  */\*\*  \* 添加学生信息  \* @param stu  \* @return  \*/* public int add(Student stu);   */\*\*  \* 修改学生信息  \* @param stu  \* @return  \*/* public int update(Student stu);   */\*\*  \* 删除学生信息  \* @param id  \* @return  \*/* public int delete(Integer id);   */\*\*  \* 根据ID查询学生信息  \* @param id  \* @return  \*/* public Student findById(Integer id);   *// 根据姓名模糊匹配学生信息* public List<Student> findByName(String name);   *// 带分页的查询* public List<Student> findAll(Map<String,Object> map); } |

### StudentMapper.xml中定义带有分页的SQL语句

|  |
| --- |
| *<?*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 namespace 对应的是接口的完整名称-->* <mapper namespace="com.etjava.mappers.StudentMappers">   *<!--定义返回的结果集-->* <resultMap id="StudentMap" type="Student">  <id property="id" column="id"/>  <result property="stuName" column="stuName"/>  <result property="age" column="age"/>  </resultMap>   *<!--带分页条件的查询-->* <select id="findAll" parameterType="Map" resultMap="StudentMap">  select \* from t\_student limit #{start},#{size}  </select>   *<!--根据姓名模糊匹配学生信息-->* <select id="findByName" parameterType="String" resultMap="StudentMap">  select \* from t\_student where stuName like concat('%',#{name},'%');  </select>   *<!--根据ID查询学生信息-->* <select id="findById" parameterType="Integer" resultType="Student">  select \* from t\_student where id=#{id}  </select>   *<!-- id 需要与接口中的方法名保持一直  parameterType 入参  Student 对应的是mybatis-config中的别名 如果那边没有配置别名  这里需要写完整的类名 包名+类名  -->* <insert id="add" parameterType="Student">  insert into t\_student values (null,#{stuName},#{age})  </insert>   *<!--修改学生信息SQL映射-->* <update id="update" parameterType="Student">  update t\_student set stuName=#{stuName},age=#{age} where id=#{id}  </update>  *<!--删除学生信息-->* <delete id="delete" parameterType="Integer">  delete from t\_student where id=#{id}  </delete> </mapper> |

### 测试

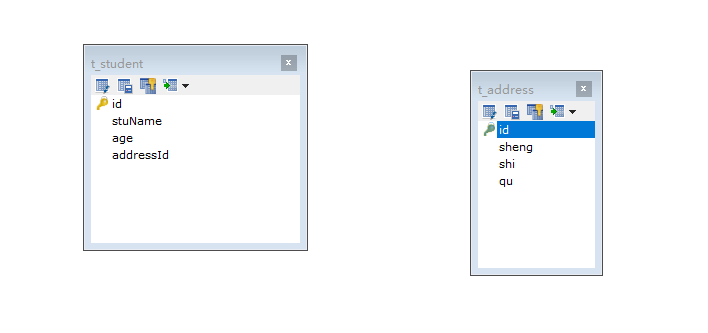
|  |
| --- |
| package com.etjava.service;  import com.etjava.mappers.StudentMappers; import com.etjava.model.Student; import com.etjava.util.SqlSessionFactoryUtil; import org.apache.ibatis.session.SqlSession; import org.apache.log4j.Logger; import org.junit.After; import org.junit.Before; import org.junit.Test;  import java.util.HashMap; import java.util.List; import java.util.Map;  *// 测试SQL映射器* public class TestSQLMapping {  private static Logger *logger* = Logger.*getLogger*(TestSQLMapping.class);  *// 定义SQLSession 用来操作数据库* SqlSession sqlSession = null;  *// 获取Student映射文件* StudentMappers studentMapper = null;  *// 测试方法执行之前* @Before  public void setUp(){  sqlSession = SqlSessionFactoryUtil.*openSession*();  studentMapper = sqlSession.getMapper(StudentMappers.class);  }   *// 测试方法执行之后* @After  public void tearDown(){  sqlSession.close();  }   *// 添加学生* @Test  public void teseAdd(){  int res = studentMapper.add(new Student("Jerry", 13));  if(res>0){  *logger*.info("添加成功");  }else{  *logger*.info("添加失败");  }  }   *// 修改学生信息* @Test  public void teseUpdate(){  int res = studentMapper.update(new Student(1, "Andy", 12));  if(res>0){  *logger*.info("修改成功");  }else{  *logger*.info("修改失败");  }  }  *// 删除学生信息* @Test  public void testDelete(){  int res = studentMapper.delete(2);  if(res>0){  *logger*.info("删除成功");  }else{  *logger*.info("删除失败");  }  }   *// 根据ID查询学生信息* @Test  public void testFindById(){  Student stu = studentMapper.findById(1);  System.*out*.println(stu);  }   *// 根据姓名模糊匹配学生信息* @Test  public void testFindByName(){  List<Student> stuList = studentMapper.findByName("Jerry");  System.*out*.println(stuList.size());  }   *// 查询全部学生信息 带分页* @Test  public void testFindAll(){  Map<String,Object> map = new HashMap<>();  map.put("start",0);  map.put("size",2);  List<Student> stuList = studentMapper.findAll(map);  System.*out*.println(stuList.size());  } } |



## 一对一查询

添加地址表及修改学生表

|  |
| --- |
| CREATE TABLE `t\_address` (    `id` int(11) NOT NULL AUTO\_INCREMENT,    `sheng` varchar(32) COLLATE utf8\_unicode\_ci DEFAULT NULL,    `shi` varchar(32) COLLATE utf8\_unicode\_ci DEFAULT NULL,    `qu` varchar(32) COLLATE utf8\_unicode\_ci DEFAULT NULL,    PRIMARY KEY (`id`)  ) ENGINE=InnoDB AUTO\_INCREMENT=4 DEFAULT CHARSET=utf8 COLLATE=utf8\_unicode\_ci |



### 添加地址表的实体类

Address

|  |
| --- |
| package com.etjava.model;  public class Address {  private Integer id;  private String sheng;  private String shi;  private String qu;   public void setId(Integer id) {  this.id = id;  }   public void setSheng(String sheng) {  this.sheng = sheng;  }   public void setShi(String shi) {  this.shi = shi;  }   public void setQu(String qu) {  this.qu = qu;  }   public Integer getId() {  return id;  }   public String getSheng() {  return sheng;  }   public String getShi() {  return shi;  }   public String getQu() {  return qu;  }   @Override  public String toString() {  return "Address{" +  "id=" + id +  ", sheng='" + sheng + '\'' +  ", shi='" + shi + '\'' +  ", qu='" + qu + '\'' +  '}';  } } |

### 修改学生的实体类

查询学生时需要带地址

|  |
| --- |
| package com.etjava.model;  public class Student {  private Integer id;  private String stuName;  private Integer age;   private Address address;*// 查询学生是可以直接将地址信息一并获取* public Student() {  super();  }   public Student(Integer id, String stuName, Integer age) {  this.id = id;  this.stuName = stuName;  this.age = age;  }   public Student(String stuName, Integer age) {  super();  this.stuName = stuName;  this.age = age;  }  public Integer getId() {  return id;  }  public void setId(Integer id) {  this.id = id;  }  public String getStuName() {  return stuName;  }  public void setStuName(String stuName) {  this.stuName = stuName;  }  public Integer getAge() {  return age;  }  public void setAge(Integer age) {  this.age = age;  }   public Address getAddress() {  return address;  }   public void setAddress(Address address) {  this.address = address;  }   @Override  public String toString() {  return "Student{" +  "id=" + id +  ", stuName='" + stuName + '\'' +  ", age=" + age +  ", address=" + address +  '}';  } } |

### StudentMapper接口中添加查询方法

|  |
| --- |
| package com.etjava.mappers;  import com.etjava.model.Student;  import java.util.List; import java.util.Map;  */\*\*  \* Student Mapper接口  \*/* public interface StudentMappers {  */\*\*  \* 添加学生信息  \* @param stu  \* @return  \*/* public int add(Student stu);   */\*\*  \* 修改学生信息  \* @param stu  \* @return  \*/* public int update(Student stu);   */\*\*  \* 删除学生信息  \* @param id  \* @return  \*/* public int delete(Integer id);   */\*\*  \* 根据ID查询学生信息  \* @param id  \* @return  \*/* public Student findById(Integer id);   *// 根据姓名模糊匹配学生信息* public List<Student> findByName(String name);   *// 带分页的查询* public List<Student> findAll(Map<String,Object> map);   *// 根据学生ID查询学生信息 带地址* public Student findStudentWithAddress(Integer stuId); } |

#### 方式1 直接使用对象级联

StudentMapper.xml中需要在resultMap中添加外键映射

这种方式可重用性不高

|  |
| --- |
| *<?*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 namespace 对应的是接口的完整名称-->* <mapper namespace="com.etjava.mappers.StudentMappers">   *<!--定义返回的结果集-->* <resultMap id="StudentMap" type="Student">  <id property="id" column="id"/>  <result property="stuName" column="stuName"/>  <result property="age" column="age"/>  </resultMap>     *<!--测试1对1查询 定义返回的结果集 这种方式重用性低-->* <resultMap id="StudentMap2" type="Student">  <id property="id" column="id"/>  <result property="stuName" column="stuName"/>  <result property="age" column="age"/>  *<!--address.id 对象级联方式-->* <result property="address.id" column="addressId"/>  <result property="address.sheng" column="sheng"/>  <result property="address.shi" column="shi"/>  <result property="address.qu" column="qu"/>   </resultMap>   *<!--根据ID查询学生信息 带地址-->* <select id="findStudentWithAddress" parameterType="Integer" resultMap="StudentMap2">  SELECT \* FROM t\_student t1,t\_address t2 WHERE t1.addressId=t2.id AND t1.id=#{id}  </select>   *<!--带分页条件的查询-->* <select id="findAll" parameterType="Map" resultMap="StudentMap">  select \* from t\_student limit #{start},#{size}  </select>   *<!--根据姓名模糊匹配学生信息-->* <select id="findByName" parameterType="String" resultMap="StudentMap">  select \* from t\_student where stuName like concat('%',#{name},'%');  </select>   *<!--根据ID查询学生信息-->* <select id="findById" parameterType="Integer" resultType="Student">  select \* from t\_student where id=#{id}  </select>   *<!-- id 需要与接口中的方法名保持一直  parameterType 入参  Student 对应的是mybatis-config中的别名 如果那边没有配置别名  这里需要写完整的类名 包名+类名  -->* <insert id="add" parameterType="Student">  insert into t\_student values (null,#{stuName},#{age})  </insert>   *<!--修改学生信息SQL映射-->* <update id="update" parameterType="Student">  update t\_student set stuName=#{stuName},age=#{age} where id=#{id}  </update>  *<!--删除学生信息-->* <delete id="delete" parameterType="Integer">  delete from t\_student where id=#{id}  </delete> </mapper> |

#### 测试

|  |
| --- |
| package com.etjava.service;  import com.etjava.mappers.StudentMappers; import com.etjava.model.Student; import com.etjava.util.SqlSessionFactoryUtil; import org.apache.ibatis.session.SqlSession; import org.apache.log4j.Logger; import org.junit.After; import org.junit.Before; import org.junit.Test;  import java.util.HashMap; import java.util.List; import java.util.Map;  *// 测试SQL映射器* public class TestSQLMapping {  private static Logger *logger* = Logger.*getLogger*(TestSQLMapping.class);  *// 定义SQLSession 用来操作数据库* SqlSession sqlSession = null;  *// 获取Student映射文件* StudentMappers studentMapper = null;  *// 测试方法执行之前* @Before  public void setUp(){  sqlSession = SqlSessionFactoryUtil.*openSession*();  studentMapper = sqlSession.getMapper(StudentMappers.class);  }   *// 测试方法执行之后* @After  public void tearDown(){  sqlSession.close();  }   *// 添加学生* @Test  public void teseAdd(){  int res = studentMapper.add(new Student("Jerry", 13));  if(res>0){  *logger*.info("添加成功");  }else{  *logger*.info("添加失败");  }  }   *// 修改学生信息* @Test  public void teseUpdate(){  int res = studentMapper.update(new Student(1, "Andy", 12));  if(res>0){  *logger*.info("修改成功");  }else{  *logger*.info("修改失败");  }  }  *// 删除学生信息* @Test  public void testDelete(){  int res = studentMapper.delete(2);  if(res>0){  *logger*.info("删除成功");  }else{  *logger*.info("删除失败");  }  }   *// 根据ID查询学生信息* @Test  public void testFindById(){  Student stu = studentMapper.findById(1);  System.*out*.println(stu);  }   *// 根据姓名模糊匹配学生信息* @Test  public void testFindByName(){  List<Student> stuList = studentMapper.findByName("Jerry");  System.*out*.println(stuList.size());  }   *// 查询全部学生信息 带分页* @Test  public void testFindAll(){  Map<String,Object> map = new HashMap<>();  map.put("start",0);  map.put("size",2);  List<Student> stuList = studentMapper.findAll(map);  System.*out*.println(stuList.size());  }   @Test  public void testFindStudentWithAddress(){  Student stu = studentMapper.findStudentWithAddress(1);  System.*out*.println(stu.getStuName()+"==="+stu.getAddress().getSheng());  } } |

#### 方式2 多结果集引用

将地址实体对象独立出来 然后在学生对象中引用

这种方式可以提高resultMap的可重用性

##### 修改mybatis配置文件

在取别名的地方将指的的类改为扫描包下的类

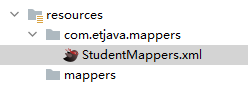
顺便把映射文件的配置也改为扫描形式

|  |
| --- |
| *<?*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>  <properties resource="jdbc.properties"/>  *<!-- 别名 -->* <typeAliases>  *<!-- type 指定需要映射的实体类 包名+类名  alias 别名  -->  <!--<typeAlias type="com.etjava.model.Student" alias="Student"/>-->* <package name="com.etjava.model"/>  </typeAliases>  *<!-- 配置环境 例如 development开发环境-->* <environments default="development">  <environment id="development">  *<!-- 事务管理 -->* <transactionManager type="JDBC"/>  *<!-- 数据源 POOLED连接池 -->* <dataSource type="POOLED">  <property name="driver" value="${jdbc.driverClassName}"/>  <property name="url" value="${jdbc.url}"/>  <property name="username" value="${jdbc.username}"/>  <property name="password" value="${jdbc.password}"/>  </dataSource>  </environment>  *<!-- 测试环境 -->* <environment id="test">  *<!-- 事务管理 -->* <transactionManager type="JDBC"/>  *<!-- 数据源 POOLED连接池 -->* <dataSource type="POOLED">  <property name="driver" value="${jdbc.driverClassName}"/>  <property name="url" value="${jdbc.url}"/>  <property name="username" value="${jdbc.username}"/>  <property name="password" value="${jdbc.password}"/>  </dataSource>  </environment>  </environments>  *<!-- 映射器 映射实体类与数据库SQL -->* <mappers>  *<!--<mapper resource="mappers/student/StudentMappers.xml"/>-->  <!--  扫描包时 如果是放在resources目录下的话 必须与mapper接口报名字保持一致 否则编译后无法将其放入到src目录下对应的包中  如果你的接口映射文件与接口文件在同一个包下 直接扫描即可  Eclipse中不会存在这些问题 直接使用即可  -->* <package name="com.etjava.mappers"/>   </mappers> </configuration> |

##### 更改映射文件存放目录

这是由于idea工具自身原因导致的 如果不与接口的完整名称保持一致 在编译后会出现找不到映射文件的问题

将其改为与接口的完整名称一致



|  |
| --- |
| *<?*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 namespace 对应的是接口的完整名称-->* <mapper namespace="com.etjava.mappers.StudentMappers">   *<!--定义返回的结果集-->* <resultMap id="StudentMap" type="Student">  <id property="id" column="id"/>  <result property="stuName" column="stuName"/>  <result property="age" column="age"/>  </resultMap>     *<!--测试1对1查询 定义返回的结果集 这种方式重用性低-->* <resultMap id="StudentMap2" type="Student">  <id property="id" column="id"/>  <result property="stuName" column="stuName"/>  <result property="age" column="age"/>  *<!--address.id 对象级联方式-->* <result property="address.id" column="addressId"/>  <result property="address.sheng" column="sheng"/>  <result property="address.shi" column="shi"/>  <result property="address.qu" column="qu"/>   </resultMap>   *<!--引入其他的结果集 引入其他结果集-->* <resultMap id="StudentMap3" type="Student">  <id property="id" column="id"/>  <result property="stuName" column="stuName"/>  <result property="age" column="age"/>  *<!--引用其他的resultmap-->* <association property="address" resultMap="AddressMap"/>  </resultMap>   *<!--定义地址实体类映射-->* <resultMap id="AddressMap" type="Address">  <id property="id" column="id"/>  <result property="sheng" column="sheng"/>  <result property="shi" column="shi"/>  <result property="qu" column="qu"/>  </resultMap>    *<!--根据ID查询学生信息 带地址-->* <select id="findStudentWithAddress" parameterType="Integer" resultMap="StudentMap3">  SELECT \* FROM t\_student t1,t\_address t2 WHERE t1.addressId=t2.id AND t1.id=#{id}  </select>   *<!--带分页条件的查询-->* <select id="findAll" parameterType="Map" resultMap="StudentMap">  select \* from t\_student limit #{start},#{size}  </select>   *<!--根据姓名模糊匹配学生信息-->* <select id="findByName" parameterType="String" resultMap="StudentMap">  select \* from t\_student where stuName like concat('%',#{name},'%');  </select>   *<!--根据ID查询学生信息-->* <select id="findById" parameterType="Integer" resultType="Student">  select \* from t\_student where id=#{id}  </select>   *<!-- id 需要与接口中的方法名保持一直  parameterType 入参  Student 对应的是mybatis-config中的别名 如果那边没有配置别名  这里需要写完整的类名 包名+类名  -->* <insert id="add" parameterType="Student">  insert into t\_student values (null,#{stuName},#{age})  </insert>   *<!--修改学生信息SQL映射-->* <update id="update" parameterType="Student">  update t\_student set stuName=#{stuName},age=#{age} where id=#{id}  </update>  *<!--删除学生信息-->* <delete id="delete" parameterType="Integer">  delete from t\_student where id=#{id}  </delete> </mapper> |

##### 测试

|  |
| --- |
| package com.etjava.service;  import com.etjava.mappers.StudentMappers; import com.etjava.model.Student; import com.etjava.util.SqlSessionFactoryUtil; import org.apache.ibatis.session.SqlSession; import org.apache.log4j.Logger; import org.junit.After; import org.junit.Before; import org.junit.Test;  import java.util.HashMap; import java.util.List; import java.util.Map;  *// 测试SQL映射器* public class TestSQLMapping {  private static Logger *logger* = Logger.*getLogger*(TestSQLMapping.class);  *// 定义SQLSession 用来操作数据库* SqlSession sqlSession = null;  *// 获取Student映射文件* StudentMappers studentMapper = null;  *// 测试方法执行之前* @Before  public void setUp(){  sqlSession = SqlSessionFactoryUtil.*openSession*();  studentMapper = sqlSession.getMapper(StudentMappers.class);  }   *// 测试方法执行之后* @After  public void tearDown(){  sqlSession.close();  }   *// 添加学生* @Test  public void teseAdd(){  int res = studentMapper.add(new Student("Jerry", 13));  if(res>0){  *logger*.info("添加成功");  }else{  *logger*.info("添加失败");  }  }   *// 修改学生信息* @Test  public void teseUpdate(){  int res = studentMapper.update(new Student(1, "Andy", 12));  if(res>0){  *logger*.info("修改成功");  }else{  *logger*.info("修改失败");  }  }  *// 删除学生信息* @Test  public void testDelete(){  int res = studentMapper.delete(2);  if(res>0){  *logger*.info("删除成功");  }else{  *logger*.info("删除失败");  }  }   *// 根据ID查询学生信息* @Test  public void testFindById(){  Student stu = studentMapper.findById(1);  System.*out*.println(stu);  }   *// 根据姓名模糊匹配学生信息* @Test  public void testFindByName(){  List<Student> stuList = studentMapper.findByName("Jerry");  System.*out*.println(stuList.size());  }   *// 查询全部学生信息 带分页* @Test  public void testFindAll(){  Map<String,Object> map = new HashMap<>();  map.put("start",0);  map.put("size",2);  List<Student> stuList = studentMapper.findAll(map);  System.*out*.println(stuList.size());  }   @Test  public void testFindStudentWithAddress(){  Student stu = studentMapper.findStudentWithAddress(1);  System.*out*.println(stu.getStuName()+"==="+stu.getAddress().getSheng());  } } |

#### 方式3 内嵌式

|  |
| --- |
| *<?*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 namespace 对应的是接口的完整名称-->* <mapper namespace="com.etjava.mappers.StudentMappers">   *<!--定义返回的结果集-->* <resultMap id="StudentMap" type="Student">  <id property="id" column="id"/>  <result property="stuName" column="stuName"/>  <result property="age" column="age"/>  </resultMap>     *<!--测试1对1查询 定义返回的结果集 这种方式重用性低-->* <resultMap id="StudentMap2" type="Student">  <id property="id" column="id"/>  <result property="stuName" column="stuName"/>  <result property="age" column="age"/>  *<!--address.id 对象级联方式-->* <result property="address.id" column="addressId"/>  <result property="address.sheng" column="sheng"/>  <result property="address.shi" column="shi"/>  <result property="address.qu" column="qu"/>   </resultMap>   *<!--引入其他的结果集 引入其他结果集-->* <resultMap id="StudentMap3" type="Student">  <id property="id" column="id"/>  <result property="stuName" column="stuName"/>  <result property="age" column="age"/>  *<!--引用其他的resultmap-->* <association property="address" resultMap="AddressMap"/>  </resultMap>   *<!--定义地址实体类映射-->* <resultMap id="AddressMap" type="Address">  <id property="id" column="id"/>  <result property="sheng" column="sheng"/>  <result property="shi" column="shi"/>  <result property="qu" column="qu"/>  </resultMap>   *<!--内嵌式-->* <resultMap id="StudentMap4" type="Student">  <id property="id" column="id"/>  <result property="stuName" column="stuName"/>  <result property="age" column="age"/>  *<!--直接内嵌一个对象进来-->* <association property="address" javaType="Address">  <id property="id" column="id"/>  <result property="sheng" column="sheng"/>  <result property="shi" column="shi"/>  <result property="qu" column="qu"/>  </association>  </resultMap>    *<!--根据ID查询学生信息 带地址-->* <select id="findStudentWithAddress" parameterType="Integer" resultMap="StudentMap4">  SELECT \* FROM t\_student t1,t\_address t2 WHERE t1.addressId=t2.id AND t1.id=#{id}  </select>   *<!--带分页条件的查询-->* <select id="findAll" parameterType="Map" resultMap="StudentMap">  select \* from t\_student limit #{start},#{size}  </select>   *<!--根据姓名模糊匹配学生信息-->* <select id="findByName" parameterType="String" resultMap="StudentMap">  select \* from t\_student where stuName like concat('%',#{name},'%');  </select>   *<!--根据ID查询学生信息-->* <select id="findById" parameterType="Integer" resultType="Student">  select \* from t\_student where id=#{id}  </select>   *<!-- id 需要与接口中的方法名保持一直  parameterType 入参  Student 对应的是mybatis-config中的别名 如果那边没有配置别名  这里需要写完整的类名 包名+类名  -->* <insert id="add" parameterType="Student">  insert into t\_student values (null,#{stuName},#{age})  </insert>   *<!--修改学生信息SQL映射-->* <update id="update" parameterType="Student">  update t\_student set stuName=#{stuName},age=#{age} where id=#{id}  </update>  *<!--删除学生信息-->* <delete id="delete" parameterType="Integer">  delete from t\_student where id=#{id}  </delete> </mapper> |

##### 测试

|  |
| --- |
| package com.etjava.service;  import com.etjava.mappers.StudentMappers; import com.etjava.model.Student; import com.etjava.util.SqlSessionFactoryUtil; import org.apache.ibatis.session.SqlSession; import org.apache.log4j.Logger; import org.junit.After; import org.junit.Before; import org.junit.Test;  import java.util.HashMap; import java.util.List; import java.util.Map;  *// 测试SQL映射器* public class TestSQLMapping {  private static Logger *logger* = Logger.*getLogger*(TestSQLMapping.class);  *// 定义SQLSession 用来操作数据库* SqlSession sqlSession = null;  *// 获取Student映射文件* StudentMappers studentMapper = null;  *// 测试方法执行之前* @Before  public void setUp(){  sqlSession = SqlSessionFactoryUtil.*openSession*();  studentMapper = sqlSession.getMapper(StudentMappers.class);  }   *// 测试方法执行之后* @After  public void tearDown(){  sqlSession.close();  }   *// 添加学生* @Test  public void teseAdd(){  int res = studentMapper.add(new Student("Jerry", 13));  if(res>0){  *logger*.info("添加成功");  }else{  *logger*.info("添加失败");  }  }   *// 修改学生信息* @Test  public void teseUpdate(){  int res = studentMapper.update(new Student(1, "Andy", 12));  if(res>0){  *logger*.info("修改成功");  }else{  *logger*.info("修改失败");  }  }  *// 删除学生信息* @Test  public void testDelete(){  int res = studentMapper.delete(2);  if(res>0){  *logger*.info("删除成功");  }else{  *logger*.info("删除失败");  }  }   *// 根据ID查询学生信息* @Test  public void testFindById(){  Student stu = studentMapper.findById(1);  System.*out*.println(stu);  }   *// 根据姓名模糊匹配学生信息* @Test  public void testFindByName(){  List<Student> stuList = studentMapper.findByName("Jerry");  System.*out*.println(stuList.size());  }   *// 查询全部学生信息 带分页* @Test  public void testFindAll(){  Map<String,Object> map = new HashMap<>();  map.put("start",0);  map.put("size",2);  List<Student> stuList = studentMapper.findAll(map);  System.*out*.println(stuList.size());  }   @Test  public void testFindStudentWithAddress(){  Student stu = studentMapper.findStudentWithAddress(1);  System.*out*.println(stu.getStuName()+"==="+stu.getAddress().getSheng());  } } |

#### 方式4 高可用的结果集引入

##### 定义Address的Mapper接口

根据ID查询

|  |
| --- |
| package com.etjava.mappers;  import com.etjava.model.Address;  public interface AddressMapper {  *// 根据ID查询地址信息* public Address findById(Integer id); } |

##### 定义AddressMapper接口的映射文件

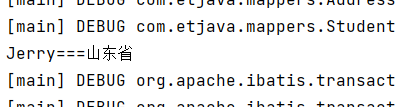
|  |
| --- |
| *<?*xml version="1.0" encoding="UTF-8" *?>* <!DOCTYPE mapper  PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"  "http://mybatis.org/dtd/mybatis-3-mapper.dtd"*> <!-- 映射 namespace 对应的是接口的完整名称-->* <mapper namespace="com.etjava.mappers.AddressMapper">    <resultMap id="AddressMap" type="Address">  <id property="id" column="id"/>  <result property="sheng" column="sheng"/>  <result property="shi" column="shi"/>  <result property="qu" column="qu"/>  </resultMap>   <select id="findById" parameterType="Integer" resultMap="AddressMap">  select \* from t\_address where id=#{id}  </select> </mapper> |

##### 在StudentMapper.xml中添加映射

|  |
| --- |
| *<?*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 namespace 对应的是接口的完整名称-->* <mapper namespace="com.etjava.mappers.StudentMappers">   *<!--定义返回的结果集-->* <resultMap id="StudentMap" type="Student">  <id property="id" column="id"/>  <result property="stuName" column="stuName"/>  <result property="age" column="age"/>  </resultMap>     *<!--测试1对1查询 定义返回的结果集 这种方式重用性低-->* <resultMap id="StudentMap2" type="Student">  <id property="id" column="id"/>  <result property="stuName" column="stuName"/>  <result property="age" column="age"/>  *<!--address.id 对象级联方式-->* <result property="address.id" column="addressId"/>  <result property="address.sheng" column="sheng"/>  <result property="address.shi" column="shi"/>  <result property="address.qu" column="qu"/>   </resultMap>   *<!--引入其他的结果集 引入其他结果集-->* <resultMap id="StudentMap3" type="Student">  <id property="id" column="id"/>  <result property="stuName" column="stuName"/>  <result property="age" column="age"/>  *<!--引用其他的resultmap-->* <association property="address" resultMap="AddressMap"/>  </resultMap>   *<!--定义地址实体类映射-->* <resultMap id="AddressMap" type="Address">  <id property="id" column="id"/>  <result property="sheng" column="sheng"/>  <result property="shi" column="shi"/>  <result property="qu" column="qu"/>  </resultMap>   *<!--内嵌式-->* <resultMap id="StudentMap4" type="Student">  <id property="id" column="id"/>  <result property="stuName" column="stuName"/>  <result property="age" column="age"/>  *<!--直接内嵌一个对象进来-->* <association property="address" javaType="Address">  <id property="id" column="id"/>  <result property="sheng" column="sheng"/>  <result property="shi" column="shi"/>  <result property="qu" column="qu"/>  </association>  </resultMap>   <resultMap id="StudentMap5" type="Student">  <id property="id" column="id"/>  <result property="stuName" column="stuName"/>  <result property="age" column="age"/>  *<!--级联查询  property 当前实体类中的属性  column 对应的是当前类中的外键属性 addressId  select 指定要执行的方法  -->* <association property="address" column="addressId" select="com.etjava.mappers.AddressMapper.findById"></association>  </resultMap>    *<!--根据ID查询学生信息 带地址-->* <select id="findStudentWithAddress" parameterType="Integer" resultMap="StudentMap5">  SELECT \* FROM t\_student t1,t\_address t2 WHERE t1.addressId=t2.id AND t1.id=#{id}  </select>   *<!--带分页条件的查询-->* <select id="findAll" parameterType="Map" resultMap="StudentMap">  select \* from t\_student limit #{start},#{size}  </select>   *<!--根据姓名模糊匹配学生信息-->* <select id="findByName" parameterType="String" resultMap="StudentMap">  select \* from t\_student where stuName like concat('%',#{name},'%');  </select>   *<!--根据ID查询学生信息-->* <select id="findById" parameterType="Integer" resultType="Student">  select \* from t\_student where id=#{id}  </select>   *<!-- id 需要与接口中的方法名保持一直  parameterType 入参  Student 对应的是mybatis-config中的别名 如果那边没有配置别名  这里需要写完整的类名 包名+类名  -->* <insert id="add" parameterType="Student">  insert into t\_student values (null,#{stuName},#{age})  </insert>   *<!--修改学生信息SQL映射-->* <update id="update" parameterType="Student">  update t\_student set stuName=#{stuName},age=#{age} where id=#{id}  </update>  *<!--删除学生信息-->* <delete id="delete" parameterType="Integer">  delete from t\_student where id=#{id}  </delete> </mapper> |

##### 测试

|  |
| --- |
| package com.etjava.service;  import com.etjava.mappers.StudentMappers; import com.etjava.model.Student; import com.etjava.util.SqlSessionFactoryUtil; import org.apache.ibatis.session.SqlSession; import org.apache.log4j.Logger; import org.junit.After; import org.junit.Before; import org.junit.Test;  import java.util.HashMap; import java.util.List; import java.util.Map;  *// 测试SQL映射器* public class TestSQLMapping {  private static Logger *logger* = Logger.*getLogger*(TestSQLMapping.class);  *// 定义SQLSession 用来操作数据库* SqlSession sqlSession = null;  *// 获取Student映射文件* StudentMappers studentMapper = null;  *// 测试方法执行之前* @Before  public void setUp(){  sqlSession = SqlSessionFactoryUtil.*openSession*();  studentMapper = sqlSession.getMapper(StudentMappers.class);  }   *// 测试方法执行之后* @After  public void tearDown(){  sqlSession.close();  }   *// 添加学生* @Test  public void teseAdd(){  int res = studentMapper.add(new Student("Jerry", 13));  if(res>0){  *logger*.info("添加成功");  }else{  *logger*.info("添加失败");  }  }   *// 修改学生信息* @Test  public void teseUpdate(){  int res = studentMapper.update(new Student(1, "Andy", 12));  if(res>0){  *logger*.info("修改成功");  }else{  *logger*.info("修改失败");  }  }  *// 删除学生信息* @Test  public void testDelete(){  int res = studentMapper.delete(2);  if(res>0){  *logger*.info("删除成功");  }else{  *logger*.info("删除失败");  }  }   *// 根据ID查询学生信息* @Test  public void testFindById(){  Student stu = studentMapper.findById(1);  System.*out*.println(stu);  }   *// 根据姓名模糊匹配学生信息* @Test  public void testFindByName(){  List<Student> stuList = studentMapper.findByName("Jerry");  System.*out*.println(stuList.size());  }   *// 查询全部学生信息 带分页* @Test  public void testFindAll(){  Map<String,Object> map = new HashMap<>();  map.put("start",0);  map.put("size",2);  List<Student> stuList = studentMapper.findAll(map);  System.*out*.println(stuList.size());  }   @Test  public void testFindStudentWithAddress(){  Student stu = studentMapper.findStudentWithAddress(1);  System.*out*.println(stu.getStuName()+"==="+stu.getAddress().getSheng());  } } |



# DAY6

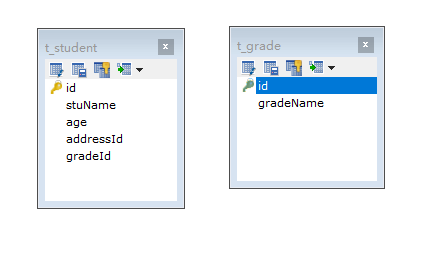
## 一对多

查询年级带学生信息

新建年级表

|  |
| --- |
| CREATE TABLE `t\_grade` (    `id` int(11) NOT NULL AUTO\_INCREMENT,    `gradeName` varchar(32) DEFAULT NULL,    PRIMARY KEY (`id`)  ) ENGINE=InnoDB AUTO\_INCREMENT=4 DEFAULT CHARSET=latin1 |

修改学生表 添加年级外键



需求 查询年级时讲所有学生查询出来

### 创建年级的实体类

|  |
| --- |
| package com.etjava.model;  import java.util.List;  public class Grade {  private Integer id;  private String gradeName;   private List<Student> studentList;   public Integer getId() {  return id;  }   public void setId(Integer id) {  this.id = id;  }   public String getGradeName() {  return gradeName;  }   public void setGradeName(String gradeName) {  this.gradeName = gradeName;  }   public List<Student> getStudentList() {  return studentList;  }   public void setStudentList(List<Student> studentList) {  this.studentList = studentList;  }   @Override  public String toString() {  return "Grade{" +  "id=" + id +  ", gradeName='" + gradeName + '\'' +  ", studentList=" + studentList +  '}';  } } |

### 修改学生实体类

添加年级外键

|  |
| --- |
| package com.etjava.model;  public class Student {  private Integer id;  private String stuName;  private Integer age;   private Address address;*// 查询学生是可以直接将地址信息一并获取* private Integer gradeId;  public Student() {  super();  }   public Student(Integer id, String stuName, Integer age) {  this.id = id;  this.stuName = stuName;  this.age = age;  }   public Student(String stuName, Integer age) {  super();  this.stuName = stuName;  this.age = age;  }  public Integer getId() {  return id;  }  public void setId(Integer id) {  this.id = id;  }  public String getStuName() {  return stuName;  }  public void setStuName(String stuName) {  this.stuName = stuName;  }  public Integer getAge() {  return age;  }  public void setAge(Integer age) {  this.age = age;  }   public Address getAddress() {  return address;  }   public void setAddress(Address address) {  this.address = address;  }   public Integer getGradeId() {  return gradeId;  }   public void setGradeId(Integer gradeId) {  this.gradeId = gradeId;  }   @Override  public String toString() {  return "Student{" +  "id=" + id +  ", stuName='" + stuName + '\'' +  ", age=" + age +  ", address=" + address +  '}';  } } |

### StudentMapper接口中添加根据年级ID查询的方法

|  |
| --- |
| package com.etjava.mappers;  import com.etjava.model.Student;  import java.util.List; import java.util.Map;  */\*\*  \* Student Mapper接口  \*/* public interface StudentMappers {  */\*\*  \* 添加学生信息  \* @param stu  \* @return  \*/* public int add(Student stu);   */\*\*  \* 修改学生信息  \* @param stu  \* @return  \*/* public int update(Student stu);   */\*\*  \* 删除学生信息  \* @param id  \* @return  \*/* public int delete(Integer id);   */\*\*  \* 根据ID查询学生信息  \* @param id  \* @return  \*/* public Student findById(Integer id);   *// 根据姓名模糊匹配学生信息* public List<Student> findByName(String name);   *// 带分页的查询* public List<Student> findAll(Map<String,Object> map);   *// 根据学生ID查询学生信息 带地址* public Student findStudentWithAddress(Integer stuId);   *// 根据年级查询学生信息* public Student findByGradeId(Integer gradeId); } |

### StudentMapper映射文件中添加SQL

|  |
| --- |
| *<?*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 namespace 对应的是接口的完整名称-->* <mapper namespace="com.etjava.mappers.StudentMappers">   *<!--定义返回的结果集-->* <resultMap id="StudentMap" type="Student">  <id property="id" column="id"/>  <result property="stuName" column="stuName"/>  <result property="age" column="age"/>  </resultMap>     *<!--测试1对1查询 定义返回的结果集 这种方式重用性低-->* <resultMap id="StudentMap2" type="Student">  <id property="id" column="id"/>  <result property="stuName" column="stuName"/>  <result property="age" column="age"/>  *<!--address.id 对象级联方式-->* <result property="address.id" column="addressId"/>  <result property="address.sheng" column="sheng"/>  <result property="address.shi" column="shi"/>  <result property="address.qu" column="qu"/>   </resultMap>   *<!--引入其他的结果集 引入其他结果集-->* <resultMap id="StudentMap3" type="Student">  <id property="id" column="id"/>  <result property="stuName" column="stuName"/>  <result property="age" column="age"/>  *<!--引用其他的resultmap-->* <association property="address" resultMap="AddressMap"/>  </resultMap>   *<!--定义地址实体类映射-->* <resultMap id="AddressMap" type="Address">  <id property="id" column="id"/>  <result property="sheng" column="sheng"/>  <result property="shi" column="shi"/>  <result property="qu" column="qu"/>  </resultMap>   *<!--内嵌式-->* <resultMap id="StudentMap4" type="Student">  <id property="id" column="id"/>  <result property="stuName" column="stuName"/>  <result property="age" column="age"/>  *<!--直接内嵌一个对象进来-->* <association property="address" javaType="Address">  <id property="id" column="id"/>  <result property="sheng" column="sheng"/>  <result property="shi" column="shi"/>  <result property="qu" column="qu"/>  </association>  </resultMap>   <resultMap id="StudentMap5" type="Student">  <id property="id" column="id"/>  <result property="stuName" column="stuName"/>  <result property="age" column="age"/>  *<!--级联查询  property 当前实体类中的属性  column 对应的是当前类中的外键属性 addressId  select 指定要执行的方法  -->* <association property="address" column="addressId" select="com.etjava.mappers.AddressMapper.findById"></association>  </resultMap>   *<!--根据年级ID查询学生信息-->* <select id="findByGradeId" parameterType="Integer" resultMap="StudentMap">  select \* from t\_student where gradeId=#{gradeId}  </select>   *<!--根据ID查询学生信息 带地址-->* <select id="findStudentWithAddress" parameterType="Integer" resultMap="StudentMap5">  SELECT \* FROM t\_student t1,t\_address t2 WHERE t1.addressId=t2.id AND t1.id=#{id}  </select>   *<!--带分页条件的查询-->* <select id="findAll" parameterType="Map" resultMap="StudentMap">  select \* from t\_student limit #{start},#{size}  </select>   *<!--根据姓名模糊匹配学生信息-->* <select id="findByName" parameterType="String" resultMap="StudentMap">  select \* from t\_student where stuName like concat('%',#{name},'%');  </select>   *<!--根据ID查询学生信息-->* <select id="findById" parameterType="Integer" resultType="Student">  select \* from t\_student where id=#{id}  </select>   *<!-- id 需要与接口中的方法名保持一直  parameterType 入参  Student 对应的是mybatis-config中的别名 如果那边没有配置别名  这里需要写完整的类名 包名+类名  -->* <insert id="add" parameterType="Student">  insert into t\_student values (null,#{stuName},#{age})  </insert>   *<!--修改学生信息SQL映射-->* <update id="update" parameterType="Student">  update t\_student set stuName=#{stuName},age=#{age} where id=#{id}  </update>  *<!--删除学生信息-->* <delete id="delete" parameterType="Integer">  delete from t\_student where id=#{id}  </delete> </mapper> |

### 创建GradeMapper接口

添加根据ID查询的方法

|  |
| --- |
| package com.etjava.mappers;  import com.etjava.model.Grade;  public interface GradeMapper {  *// 根据ID查询年级* Grade findById(Integer id); } |

### GradeMapper映射文件中添加根据ID查询的方法

|  |
| --- |
| *<?*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 namespace 对应的是接口的完整名称-->* <mapper namespace="com.etjava.mappers.GradeMapper">    <resultMap id="GradeMap" type="Grade">  <id property="id" column="id"/>  <result property="gradeName" column="gradeName"/>  *<!--  collection 一对多查询映射关系  property 当前Grade中的属性  column 对应的是Student类中的主键ID  select 指向StudentMapper中根据年级ID查询的语句  -->* <collection property="studentList" column="id" select="com.etjava.mappers.StudentMappers.findByGradeId"></collection>  </resultMap>   <select id="findById" parameterType="Integer" resultMap="GradeMap">  select \* from t\_grade where id=#{id}  </select> </mapper> |

### 测试

|  |
| --- |
| package com.etjava.service; import com.etjava.mappers.GradeMapper; import com.etjava.model.Grade; import com.etjava.model.Student; import com.etjava.util.SqlSessionFactoryUtil; import org.apache.ibatis.session.SqlSession; import org.apache.log4j.Logger; import org.junit.After; import org.junit.Before; import org.junit.Test;  import java.util.List;  *// 测试SQL映射器* public class TestGradeSQLMapping {  private static Logger *logger* = Logger.*getLogger*(TestGradeSQLMapping.class);  *// 定义SQLSession 用来操作数据库* SqlSession sqlSession = null;  *// 获取Student映射文件* GradeMapper gradeMapper = null;  *// 测试方法执行之前* @Before  public void setUp(){  sqlSession = SqlSessionFactoryUtil.*openSession*();  gradeMapper = sqlSession.getMapper(GradeMapper.class);  }   *// 测试方法执行之后* @After  public void tearDown(){  sqlSession.close();  }   @Test  public void testFindByGradeId(){  Grade grade = gradeMapper.findById(1);  System.*out*.println(grade);  } } |

# DAY7

动态SQL

## if条件

当查询条件可能不存在时 需要通过if进行判断是否需要添加到查询语句中

### 新建教师表

|  |
| --- |
| CREATE TABLE `t\_teacher` (  `id` int(11) NOT NULL AUTO\_INCREMENT,  `teaName` varchar(32) COLLATE utf8\_unicode\_ci DEFAULT NULL,  `teaAge` int(11) DEFAULT NULL,  `curriculum` varchar(32) COLLATE utf8\_unicode\_ci DEFAULT NULL COMMENT '课程',  PRIMARY KEY (`id`)  ) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8\_unicode\_ci |

### 新建老师的实体类

|  |
| --- |
| package com.etjava.model;  public class Teacher {  private Integer id;  private String teaName;  private Integer teaAge;  private String curriculum;   public Integer getId() {  return id;  }   public void setId(Integer id) {  this.id = id;  }   public String getTeaName() {  return teaName;  }   public void setTeaName(String teaName) {  this.teaName = teaName;  }   public Integer getTeaAge() {  return teaAge;  }   public void setTeaAge(Integer teaAge) {  this.teaAge = teaAge;  }   public String getCurriculum() {  return curriculum;  }   public void setCurriculum(String curriculum) {  this.curriculum = curriculum;  }   @Override  public String toString() {  return "Teacher{" +  "id=" + id +  ", teaName='" + teaName + '\'' +  ", teaAge=" + teaAge +  ", curriculum='" + curriculum + '\'' +  '}';  } } |

### 新建Mapper接口

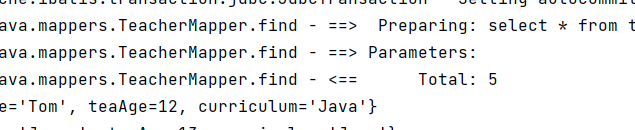
|  |
| --- |
| package com.etjava.mappers;  import com.etjava.model.Teacher;  import java.util.List; import java.util.Map;  public interface TeacherMapper {   List<Teacher> find(Map<String,Object> map); } |

### 创建接口的映射文件

|  |
| --- |
| *<?*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"*> <!-- 映射 namespace 对应的是接口的完整名称-->* <mapper namespace="com.etjava.mappers.TeacherMapper">  <resultMap id="TeacherMap" type="Teacher">  <id property="id" column="id"/>  <result property="teaName" column="teaName"/>  <result property="teaAge" column="teaAge"/>  <result property="curriculum" column="curriculum"/>  </resultMap>   <select id="find" parameterType="Integer" resultMap="TeacherMap">  select \* from t\_teacher where 1=1  <if test="teaName!=null ">  and teaName=#{teaName}  </if>  <if test="teaAge!=null">  and teaAge=#{teaAge}  </if>  <if test="curriculum!=null">  and curriculum=#{curriculum}  </if>  </select> </mapper> |

### 测试类

|  |
| --- |
| package com.etjava.service;  import com.etjava.mappers.TeacherMapper; import com.etjava.model.Teacher; import com.etjava.util.SqlSessionFactoryUtil; import org.apache.ibatis.session.SqlSession; import org.apache.log4j.Logger; import org.junit.After; import org.junit.Before; import org.junit.Test;  import java.util.HashMap; import java.util.List; import java.util.Map;  public class TeacherTest {  private static Logger *logger* = Logger.*getLogger*(TeacherTest.class);  *// 定义SQLSession 用来操作数据库* SqlSession sqlSession = null;  *// 获取Student映射文件* TeacherMapper teacherMapper = null;  *// 测试方法执行之前* @Before  public void setUp(){  sqlSession = SqlSessionFactoryUtil.*openSession*();  teacherMapper = sqlSession.getMapper(TeacherMapper.class);  }   *// 测试方法执行之后* @After  public void tearDown(){  sqlSession.close();  }   *// 测试多条件查询 if判断* @Test  public void testFind(){  Map<String,Object> map = new HashMap<>();  *//map.put("teaName","Tom");* List<Teacher> list = teacherMapper.find(map);  for (Teacher teacher : list) {  System.*out*.println(teacher);  }  } } |



## choose when otherwhise

类似if else if条件 但只有一个条件 例如 页面中用户在下拉框中选择一个查询条件时 动态获取对应的数据

### TeacherMapper接口中添加查询方法

|  |
| --- |
| package com.etjava.mappers;  import com.etjava.model.Teacher;  import java.util.List; import java.util.Map;  public interface TeacherMapper {   List<Teacher> find(Map<String,Object> map);   List<Teacher> find2(Map<String,Object> map); } |

### TeacherMapper.xml中添加映射语句

|  |
| --- |
| *<?*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 namespace 对应的是接口的完整名称-->* <mapper namespace="com.etjava.mappers.TeacherMapper">  <resultMap id="TeacherMap" type="Teacher">  <id property="id" column="id"/>  <result property="teaName" column="teaName"/>  <result property="teaAge" column="teaAge"/>  <result property="curriculum" column="curriculum"/>  </resultMap>   <select id="find" parameterType="Integer" resultMap="TeacherMap">  select \* from t\_teacher where 1=1  <if test="teaName!=null ">  and teaName=#{teaName}  </if>  <if test="teaAge!=null">  and teaAge=#{teaAge}  </if>  <if test="curriculum!=null">  and curriculum=#{curriculum}  </if>  </select>   <select id="find2" parameterType="Map" resultMap="TeacherMap">  select \* from t\_teacher  <choose>  *<!-- 相当于if -->* <when test="searchBy=='name'">  where teaName=#{teaName}  </when>  <when test="searchBy=='age'">  where teaAge=#{teaAge}  </when>  *<!--相当于else-->* <otherwise>  where curriculum='Java'  </otherwise>  </choose>  </select> </mapper> |

### 测试

|  |
| --- |
| package com.etjava.service;  import com.etjava.mappers.TeacherMapper; import com.etjava.model.Teacher; import com.etjava.util.SqlSessionFactoryUtil; import org.apache.ibatis.session.SqlSession; import org.apache.log4j.Logger; import org.junit.After; import org.junit.Before; import org.junit.Test;  import java.util.HashMap; import java.util.List; import java.util.Map;  public class TeacherTest {  private static Logger *logger* = Logger.*getLogger*(TeacherTest.class);  *// 定义SQLSession 用来操作数据库* SqlSession sqlSession = null;  *// 获取Student映射文件* TeacherMapper teacherMapper = null;  *// 测试方法执行之前* @Before  public void setUp(){  sqlSession = SqlSessionFactoryUtil.*openSession*();  teacherMapper = sqlSession.getMapper(TeacherMapper.class);  }   *// 测试方法执行之后* @After  public void tearDown(){  sqlSession.close();  }   *// 测试多条件查询 if判断* @Test  public void testFind(){  Map<String,Object> map = new HashMap<>();  *//map.put("teaName","Tom");* List<Teacher> list = teacherMapper.find(map);  for (Teacher teacher : list) {  System.*out*.println(teacher);  }  }   */\*  choose when otherwhise  多条件选择  \*/* @Test  public void testFind2(){  Map<String,Object> map = new HashMap<>();  map.put("searchBy","name");  map.put("teaName","Tom");  List<Teacher> list = teacherMapper.find2(map);  for (Teacher teacher : list) {  System.*out*.println(teacher);  }  } } |

# DAY8

动态SQL

接上讲

## where条件

如果只使用if 我们必须有一个固定条件 如果写where 1=1这样是非常不好的 因为会全表扫描

自动添加where关键字，如果where子句以and或or开头会自动删除第一个and或or

### TeacherMapper接口中添加查询方法

|  |
| --- |
| List<Teacher> find3(Map<String,Object> map); |

### TeacherMapper.xml中添加映射语句

|  |
| --- |
| <select id="find3" parameterType="Map" resultMap="TeacherMap">  select \* from t\_teacher  <where>  <if test="teaName!=null">  and teaName=#{teaName}  </if>  <if test="teaAge!=null">  and teaAge=#{teaAge}  </if>  </where>  </select> |

### 测试

|  |
| --- |
| @Test  public void testFind3(){  Map<String,Object> map = new HashMap<>();  map.put("teaName","Tom");  List<Teacher> list = teacherMapper.find3(map);  for (Teacher teacher : list) {  System.out.println(teacher);  }  } |

## foreach

SQL中传入集合时用来编辑集合中数据的

### TeacherMapper接口中添加查询方法

|  |
| --- |
| List<Teacher> find5(Map<String,Object> map); |

### TeacherMapper.xml中添加SQL映射语句

|  |
| --- |
| <select id="find5" parameterType="Map" resultMap="TeacherMap">  select \* from t\_teacher  <if test="ids!=null">  <!--  collection="ids" 传入的集合  item="id" 每次变量获取的数据存放到id中  open="(" 添加前缀 小括号  separator="," 添加分隔符  close=")" 添加后缀小括号  -->  <where>  id in  <foreach item="id" collection="ids" open="(" separator="," close=")">  #{id}  </foreach>  </where>  </if>  </select> |

### 测试

|  |
| --- |
| @Test  public void testFind5(){  Map<String,Object> map = new HashMap<>();  map.put("teaName","Tom");  List<Integer> ids = new ArrayList<>();  Collections.addAll(ids,1,2);  map.put("ids",ids);  List<Teacher> list = teacherMapper.find5(map);  for (Teacher teacher : list) {  System.out.println(teacher);  }  } |

## set条件

自动添加set关键字 并自动剔除最后一个逗号

### TeacherMapper接口中添加更新数据的方法

|  |
| --- |
| void update(Teacher teacher); |

### TeacherMapper.xml中添加映射SQL语句

|  |
| --- |
| <update id="update" parameterType="Teacher">  update t\_teacher  <set>  <if test="teaName!=null">  teaName=#{teaName},  </if>  <if test="teaAge!=null">  teaAge=#{teaAge},  </if>  </set>  where id=#{id}  </update> |

### 测试

|  |
| --- |
| @Test  public void testUpdate(){  Teacher teacher = new Teacher();  teacher.setId(1);  teacher.setTeaAge(22);  teacher.setTeaName("Tom2");  teacherMapper.update(teacher);  sqlSession.commit();// 更新操作 junti测试时需要手动提交  } |

# DAY9

## 处理CLOB、BLOG类型数据

CLOB 文本类型 但数据流较大 mysql中使用longtext类型 因为text存放的数据较小

BLOB 媒体文件 mysql中使用longblob类型 因为blob存放的数据较小

### 修改教员表

新增pic教员图片 remark教员简介字段

|  |
| --- |
| CREATE TABLE `t\_teacher` (  `id` int(11) NOT NULL AUTO\_INCREMENT,  `teaName` varchar(32) COLLATE utf8\_unicode\_ci DEFAULT NULL,  `teaAge` int(11) DEFAULT NULL,  `curriculum` varchar(32) COLLATE utf8\_unicode\_ci DEFAULT NULL COMMENT '课程',  `pic` longblob,  `remark` longtext COLLATE utf8\_unicode\_ci,  PRIMARY KEY (`id`)  ) ENGINE=InnoDB AUTO\_INCREMENT=5 DEFAULT CHARSET=utf8 COLLATE=utf8\_unicode\_ci |

### Teacher实体类中新增两个字段对应的属性

|  |
| --- |
| public class Teacher {  private Integer id;  private String teaName;  private Integer teaAge;  private String curriculum;  private byte[] pic;// blob 字节形式  private String remark;// clob 字符串  } |

### TeacherMapper接口中新增添加和修改方法

|  |
| --- |
| int addTeacher(Teacher teacher);  Teacher findById(Integer id); |

### TeacherMapper.xml新增添加和修改的映射SQL

|  |
| --- |
| <insert id="addTeacher" parameterType="Teacher">  insert into t\_teacher values(null,#{teaName},#{teaAge},#{curriculum},#{pic},#{remark})  </insert>  <select id="findById" parameterType="Integer" resultMap="TeacherMap">  select \* from t\_teacher where id=#{id}  </select> |

### 测试添加

|  |
| --- |
| @Test  public void testAdd(){  Teacher teacher = new Teacher();  teacher.setId(1);  teacher.setTeaAge(22);  teacher.setTeaName("Tom2");  teacher.setRemark("这是一个很长的文本内容...........");  byte[] pic = null;  try (FileInputStream fis = new FileInputStream("d:/a.jpg");){  pic = new byte[fis.available()];// available 文件的长度  fis.read(pic);// 将文件存放到字节数组中  }catch (Exception e){  e.printStackTrace();  }  teacher.setPic(pic);  int res = teacherMapper.addTeacher(teacher);  System.out.println(res);  sqlSession.commit();  } |



### 测试读取

|  |
| --- |
| @Test  public void testFindById(){  Teacher tea = teacherMapper.findById(6);  System.out.println(tea);// 普通属性直接输出  // 图片数据需要保存到本地磁盘  byte[] pic = tea.getPic();  try(FileOutputStream fos = new FileOutputStream("d:/1.jpg")){  fos.write(pic);  } catch (IOException e) {  throw new RuntimeException(e);  }  } |

## 多参数查询

多参数查询时通常使用Map或List 参考动态SQL内容的实现

这里在讲一个不推荐的方式

例如 一个查询方法中有两个参数 此时需要使用param1 param2方式实现

### TeacherMapper接口中添加查询方法

|  |
| --- |
| Teacher findA3(String teaName,int age); |

### TeacherMapper.xml中添加映射SQL

|  |
| --- |
| <!-- 如果入参是多个 这里就不能直接指定参数类型 而改为#{param1} #{param2} 方式实现-->  <select id="findA3" resultMap="TeacherMap">  select \* from t\_teacher where teaName=#{param1} and teaAge=#{param2}  </select> |

### 测试

|  |
| --- |
| @Test  public void test(){  Teacher tom = teacherMapper.findA3("Tom", 12);  System.out.println(tom);  } |

## MyBatis分页

MyBatis分页分为逻辑分页和物理分页

逻辑分页是在内存中进行分页 例如一百条数据 每页显示十条 那么每次都会获取这一百条数据 然后在从内存中把需要的数据单独拿出来 这种性能较低

物理分页 我们实际怎么查询就怎么返回数据

### 逻辑分页[不推荐]

逻辑分页也叫内存分页

#### TeacherMapper接口添加查询方法

|  |
| --- |
| // 逻辑分页 RowBounds  List<Teacher> findA4(RowBounds rowBounds); |

#### TeacherMapper.xml中添加映射SQL

|  |
| --- |
| <!-- 逻辑分页 参数RowBounds可以不用定义数据类型-->  <select id="findA4" resultMap="TeacherMap" >  select \* from t\_teacher  </select> |

#### 测试

|  |
| --- |
| @Test  public void test2(){  // RowBounds 构造方法有两个参数  // 第一个相当于start 起始页  // 第二个相当于截止页码  // 相当于 limit 0,3 注意 这是在内存中进行分页的  List<Teacher> list = teacherMapper.findA4(new RowBounds(3, 3));  for (Teacher teacher : list) {  System.out.println(teacher);  }  } |

### 物理分页

实际需要多少就直接取多少数据[推荐]

#### TeacherMapper接口中添加分页查询方法

|  |
| --- |
| // 物理分页  List<Teacher> findA4(Map<String,Object> map); |

#### TeacherMapper.xml中添加分页SQL映射

|  |
| --- |
| <!-- 物理分页 真正的分页-->  <select id="findA5" resultMap="TeacherMap" >  select \* from t\_teacher  <if test="start!=null and size!=null">  limit #{start},#{size}  </if>  </select> |

#### 测试

|  |
| --- |
| @Test  public void test3(){  Map<String,Object> map = new HashMap<>();  map.put("start",0);  map.put("size",2);  List<Teacher> list = teacherMapper.findA5(map);  for (Teacher teacher : list) {  System.out.println(teacher);  }  } |

# DAY10

## MyBatis缓存

MyBatis的缓存机制是用来提高查询效率的，默认启用的是一级缓存，即同一个sqlsession接口对象调用了相同的select语句 则直接在缓存中返回结果 而不是在查询一次数据库

我们可以配置二级缓存，二级缓存是全局的

默认情况下只有 select语句使用缓存，insert，update，delete不使用缓存

### 配置二级缓存

在接口的Mapper映射文件中添加如下内容

TeacherMapper.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.etjava.mappers.TeacherMapper">  <resultMap id="TeacherMap" type="Teacher">  <id property="id" column="id"/>  <result property="teaName" column="teaName"/>  <result property="teaAge" column="teaAge"/>  <result property="curriculum" column="curriculum"/>  </resultMap>  *<!--  size="1024" 表示cache中能容纳最大元素的数量 默认1024  flushInterval="60000" 定义缓存刷新周期 单位毫秒  eviction="LRU" 定义缓存的移除机制，默认LRU(Least Recently Used) 最近最少使用  还有FIFO(First In First Out) 先进先出  readOnly="false" 是否只读 默认false 可读可写 true 缓存只能读  使用缓存必须实现Serializable接口  -->* <cache size="1024" flushInterval="60000" eviction="LRU" readOnly="false"/>  *<!--  然后在select中添加使用缓存配置  useCache="true" 定义使用缓存 默认就是true  flushCache="false" 定义是否清空缓存 false为不清空缓存  insert 语句中默认是清空缓存的  -->* <select id="find" parameterType="Map" resultMap="TeacherMap" useCache="true" flushCache="false">  select \* from t\_teacher  <where>  <if test="teaName!=null and teaName!='' ">  and teaName=#{teaName}  </if>  </where>  </select>  *<!--添加修改删除语句默认清空缓存的-->* <insert id="addTeacher" parameterType="Teacher" flushCache="true">  insert into t\_teacher values(null,#{teaName},${teaAge},#{curriculum},#{pic},#{remark})  </insert>  </mapper> |

### 测试

|  |
| --- |
| @Test public void testFind(){  Map<String,Object> map = new HashMap<>();  map.put("teaName","Jerry");  List<Teacher> list = teacherMapper.find(map);  for (Teacher teacher : list) {  System.*out*.println(teacher);  } }  @Test public void testAdd(){  Teacher teacher = new Teacher();  teacher.setId(1);  teacher.setTeaAge(22);  teacher.setTeaName("Tom3");  teacher.setRemark("这是一个很长的文本内容...........");  byte[] pic = null;  try (FileInputStream fis = new FileInputStream("d:/a.jpg");){  pic = new byte[fis.available()];*// available 文件的长度* fis.read(pic);*// 将文件存放到字节数组中* }catch (Exception e){  e.printStackTrace();  }  teacher.setPic(pic);  int res = teacherMapper.addTeacher(teacher);  System.*out*.println(res);  sqlSession.commit(); } |

## 使用注解配置SQL映射器

### 新建测试表 t\_user

|  |
| --- |
| CREATE TABLE `t\_user` (  `id` int(11) NOT NULL AUTO\_INCREMENT,  `name` varchar(32) COLLATE utf8\_unicode\_ci DEFAULT NULL,  `age` int(11) DEFAULT NULL,  PRIMARY KEY (`id`)  ) ENGINE=InnoDB AUTO\_INCREMENT=5 DEFAULT CHARSET=utf8 COLLATE=utf8\_unicode\_ci |

### 新建实体类

|  |
| --- |
| @Data @ToString @NoArgsConstructor @AllArgsConstructor public class User {  private Integer id;  private String name;  private Integer age; } |

### 新建Mapper接口

|  |
| --- |
| public interface UserMapper {  Integer add(User user);  Integer update(User user);  Integer delete(Integer id);  List<User2> list();  } |

### 基本映射注解

所有注解都在Mapper接口中添加

#### @Insert

|  |
| --- |
| @Insert("insert into t\_user values(null,#{name},#{age})")  Integer add(User user); |

##### 测试

|  |
| --- |
| @Test  public void testInsert(){  User u = new User();  u.setName("Judy");  u.setAge(16);  Integer res = userMapper.add(u);  System.out.println(res);  sqlSession.commit();  } |

#### @Update

|  |
| --- |
| @Update("update t\_user set name=#{name},age=#{age} where id=#{id}")  Integer update(User user); |

##### 测试

|  |
| --- |
| @Test public void testUpdate(){  User u = new User();  u.setName("Judy3");  u.setAge(16);  u.setId(6);  Integer res = userMapper.update(u);  System.*out*.println(res);  sqlSession.commit(); } |

#### @Delete

|  |
| --- |
| @Delete("delete from t\_user where id=#{id}")  Integer delete(Integer id); |

##### 测试

|  |
| --- |
| @Test public void testDelete(){  Integer res = userMapper.delete(4);  System.*out*.println(res);  sqlSession.commit(); } |

#### @Select

##### 基本映射

|  |
| --- |
| @Select("select \* from t\_user where id=#{id}")  User findById(Integer id); |

###### 测试

|  |
| --- |
| @Test public void testQuery(){  User user = userMapper.findById(1);  System.*out*.println(user); } |

##### 结果集映射

|  |
| --- |
| @Select("select \* from t\_user")  @Results(// 映射结果集 相当于配置文件中的resultMap  {  @Result(id = true,column = "id",property = "id"),  @Result(column = "name",property = "name"),  @Result(column = "age",property = "age")  }  )  List<User> list(); |

###### 测试

|  |
| --- |
| @Test public void testQuery2(){  List<User> list = userMapper.list();  for (User user2 : list) {  System.*out*.println(user2);  } } |

### 关系映射

#### 一对一

更改user表 添加地址外键

|  |
| --- |
| CREATE TABLE `t\_user` (  `id` int(11) NOT NULL AUTO\_INCREMENT,  `name` varchar(32) COLLATE utf8\_unicode\_ci DEFAULT NULL,  `age` int(11) DEFAULT NULL,  `addressId` int(11) DEFAULT NULL,  PRIMARY KEY (`id`)  ) ENGINE=InnoDB AUTO\_INCREMENT=6 DEFAULT CHARSET=utf8 COLLATE=utf8\_unicode\_ci |

地址表

|  |
| --- |
| CREATE TABLE `t\_address` (  `id` int(11) NOT NULL AUTO\_INCREMENT,  `sheng` varchar(32) COLLATE utf8\_unicode\_ci DEFAULT NULL,  `shi` varchar(32) COLLATE utf8\_unicode\_ci DEFAULT NULL,  `qu` varchar(32) COLLATE utf8\_unicode\_ci DEFAULT NULL,  PRIMARY KEY (`id`)  ) ENGINE=InnoDB AUTO\_INCREMENT=4 DEFAULT CHARSET=utf8 COLLATE=utf8\_unicode\_ci |

##### 修改实体

|  |
| --- |
| public class User {  private Integer id;  private String name;  private Integer age;  private Address address;  } |

##### UserMapper接口中添加查询语句

|  |
| --- |
| @Select("select \* from t\_user")  @Results(// 映射结果集 相当于配置文件中的resultMap  {  @Result(id = true,column = "id",property = "id"),  @Result(column = "name",property = "name"),  @Result(column = "age",property = "age"),  // one = @One 表示一对一 需要关联其他Mapper接口中的查询方法  // column 表中的外键 property实体类中的属性  // 如果AddressMapper中可以使用xml映射文件 也可以使用注解方式根据ID查询  @Result(column = "addressId",property = "address"  ,one = @One(select = "com.etjava.mappers.AddressMapper.findById"))  }  )  List<User> findUserWithAddress(); |

##### 测试

|  |
| --- |
| @Test public void testQuery3(){  List<User> u = userMapper.findUserWithAddress();  for (User user2 : u) {  System.*out*.println(user2);  } } |

#### 一对多

查询部门 需要将部门下所有员工信息获取到

##### 部门表

|  |
| --- |
| CREATE TABLE `t\_dept` (  `id` int(11) NOT NULL AUTO\_INCREMENT,  `deptName` varchar(32) COLLATE utf8\_unicode\_ci DEFAULT NULL,  PRIMARY KEY (`id`)  ) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8\_unicode\_ci |

修改user表 添加部门外键

|  |
| --- |
| CREATE TABLE `t\_user` (  `id` int(11) NOT NULL AUTO\_INCREMENT,  `name` varchar(32) COLLATE utf8\_unicode\_ci DEFAULT NULL,  `age` int(11) DEFAULT NULL,  `addressId` int(11) DEFAULT NULL,  `deptId` int(11) DEFAULT NULL,  PRIMARY KEY (`id`)  ) ENGINE=InnoDB AUTO\_INCREMENT=6 DEFAULT CHARSET=utf8 COLLATE=utf8\_unicode\_ci |

##### 新建部门实体

|  |
| --- |
| @Data @ToString public class Dept {  private Integer id;  private String deptName;  private List<User> userList; } |

##### 修改user实体 添加关联属性

|  |
| --- |
| @Data @ToString  public class User {  private Integer id;  private String name;  private Integer age;  private Address address;  private Dept dept;  } |

##### 新建DeptMapper

|  |
| --- |
| package com.etjava.mappers;  import com.etjava.model.Dept; import org.apache.ibatis.annotations.Many; import org.apache.ibatis.annotations.Result; import org.apache.ibatis.annotations.Results; import org.apache.ibatis.annotations.Select;  public interface DeptMapper {  @Select("select *\** from t\_dept where id=#{id}")  Dept findById(Integer id);   @Select("select *\** from t\_dept where id=#{id}")  @Results(  {  @Result(id = true,column = "id",property = "id"),  @Result(column = "deptName",property = "deptName"),  *// 关联查询 需要带入当前表的主键 到User表中根据外键查询* @Result(column = "id",property = "userList"  ,many = @Many(select = "com.etjava.mappers.UserMapper.findUserWithDeptId"))  }  )  Dept findById2(Integer id); } |

##### 测试

|  |
| --- |
| package com.etjava.service;  import com.etjava.mappers.DeptMapper; import com.etjava.model.Dept; import com.etjava.util.SqlSessionFactoryUtil; import org.apache.ibatis.session.SqlSession; import org.apache.log4j.Logger; import org.junit.After; import org.junit.Before; import org.junit.Test;   public class DeptTest {  private static Logger *logger* = Logger.*getLogger*(DeptTest.class);  *// 定义SQLSession 用来操作数据库* SqlSession sqlSession = null;  *// 获取Student映射文件* DeptMapper deptMapper = null;  *// 测试方法执行之前* @Before  public void setUp(){  sqlSession = SqlSessionFactoryUtil.*openSession*();  deptMapper = sqlSession.getMapper(DeptMapper.class);  }   *// 测试方法执行之后* @After  public void tearDown(){  sqlSession.close();  }    @Test  public void testQuery3(){  Dept d = deptMapper.findById2(1);  System.*out*.println(d);  } } |