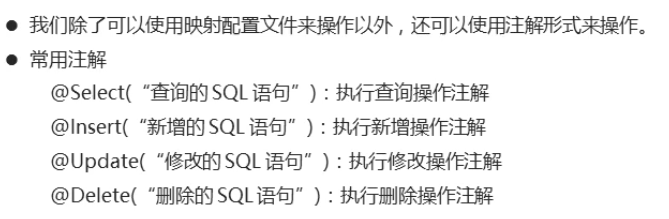
03 MyBatis高级

# 1（掌握）MyBatis注解开发

## 1.1（掌握）注解开发的介绍



## 1.2（掌握）注解实现查询操作



### 源码

MyBatisConfig.xml

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8" ?>  <!--MyBatis的DTD约束-->  <!DOCTYPE configuration PUBLIC "-//mybatis.org//DTD Config 3.0//EN" "http://mybatis.org/dtd/mybatis-3-config.dtd">  <!--configuration 核心根标签-->  <configuration>  <!--引入数据库连接的配置文件-->  <properties resource="jdbc.properties"/>  <!--配置LOG4J-->  <settings>  <setting name="logImpl" value="log4j"/>  </settings>  <!--起别名-->  <typeAliases>  <package name="com.itheima.bean"/>  </typeAliases>  <!--environments配置数据库环境，环境可以有多个。default属性指定使用的是哪个-->  <environments default="mysql">  <!--environment配置数据库环境 id属性唯一标识-->  <environment id="mysql">  <!-- transactionManager事务管理。 type属性，采用JDBC默认的事务-->  <transactionManager type="JDBC"></transactionManager>  <!-- dataSource数据源信息 type属性 连接池-->  <dataSource type="POOLED">  <!-- property获取数据库连接的配置信息 -->  <property name="driver" value="${driver}" />  <property name="url" value="${url}" />  <property name="username" value="${username}" />  <property name="password" value="${password}" />  </dataSource>  </environment>  </environments>  <!--配置映射关系-->  <mappers>  <package name="com.itheima.mapper"/>  </mappers>  </configuration> |

Student.java

|  |
| --- |
| public class Student {  private Integer id;  private String name;  private Integer age;  public Student() {  }  public Student(Integer id, String name, Integer age) {  this.id = id;  this.name = name;  this.age = age;  }  public Integer getId() {  return id;  }  public void setId(Integer id) {  this.id = id;  }  public String getName() {  return name;  }  public void setName(String name) {  this.name = name;  }  public Integer getAge() {  return age;  }  public void setAge(Integer age) {  this.age = age;  }  @Override  public String toString() {  return "Student{" +  "id=" + id +  ", name='" + name + '\'' +  ", age=" + age +  '}';  }  } |

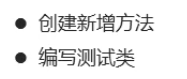
StudentMapper.java

|  |
| --- |
| public interface StudentMapper {  //查询全部  @Select("SELECT \* FROM student")  public abstract List<Student> selectAll();  } |

Test01.java

|  |
| --- |
| public class Test01 {  @Test  public void selectAll() throws Exception{  //1.加载核心配置文件  InputStream is = Resources.getResourceAsStream("MyBatisConfig.xml");  //2.获取SqlSession工厂对象  SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(is);  //3.通过工厂对象获取SqlSession对象  SqlSession sqlSession = sqlSessionFactory.openSession(true);  //4.获取StudentMapper接口的实现类对象  StudentMapper mapper = sqlSession.getMapper(StudentMapper.class);  //5.调用实现类对象中的方法，接收结果  List<Student> list = mapper.selectAll();  //6.处理结果  for (Student student : list) {  System.out.println(student);  }  //7.释放资源  sqlSession.close();  is.close();  }  } |

## 1.3（掌握）注解实现新增操作



### 源码

StudentMapper.java

|  |
| --- |
| public interface StudentMapper {  //查询全部  @Select("SELECT \* FROM student")  public abstract List<Student> selectAll();  //新增操作  @Insert("INSERT INTO student VALUES (#{id},#{name},#{age})")  public abstract Integer insert(Student stu);  } |

Test01.java

|  |
| --- |
| public class Test01 {  @Test  public void insert() throws Exception{  //1.加载核心配置文件  InputStream is = Resources.getResourceAsStream("MyBatisConfig.xml");  //2.获取SqlSession工厂对象  SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(is);  //3.通过工厂对象获取SqlSession对象  SqlSession sqlSession = sqlSessionFactory.openSession(true);  //4.获取StudentMapper接口的实现类对象  StudentMapper mapper = sqlSession.getMapper(StudentMapper.class);  //5.调用实现类对象中的方法，接收结果  Student stu = new Student(4,"赵六",26);  Integer result = mapper.insert(stu);  //6.处理结果  System.out.println(result);  //7.释放资源  sqlSession.close();  is.close();  }  } |

## 1.4（掌握）注解实现修改操作



### 源码

StudentMapper.java

|  |
| --- |
| public interface StudentMapper {  //查询全部  @Select("SELECT \* FROM student")  public abstract List<Student> selectAll();  //新增操作  @Insert("INSERT INTO student VALUES (#{id},#{name},#{age})")  public abstract Integer insert(Student stu);  //修改操作  @Update("UPDATE student SET name=#{name},age=#{age} WHERE id=#{id}")  public abstract Integer update(Student stu);  } |

Test01.java

|  |
| --- |
| public class Test01 {  @Test  public void update() throws Exception{  //1.加载核心配置文件  InputStream is = Resources.getResourceAsStream("MyBatisConfig.xml");  //2.获取SqlSession工厂对象  SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(is);  //3.通过工厂对象获取SqlSession对象  SqlSession sqlSession = sqlSessionFactory.openSession(true);  //4.获取StudentMapper接口的实现类对象  StudentMapper mapper = sqlSession.getMapper(StudentMapper.class);  //5.调用实现类对象中的方法，接收结果  Student stu = new Student(4,"赵六",36);  Integer result = mapper.update(stu);  //6.处理结果  System.out.println(result);  //7.释放资源  sqlSession.close();  is.close();  }  } |

## 1.5（掌握）注解实现删除操作



### 源码

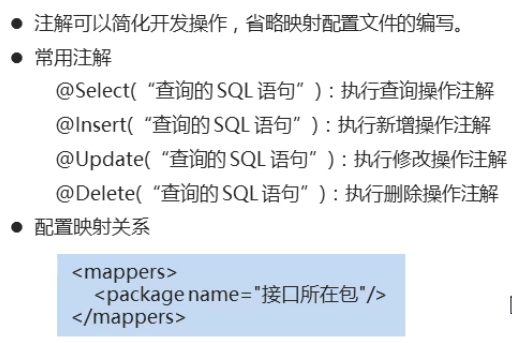
StudentMapper.java

|  |
| --- |
| public interface StudentMapper {  //查询全部  @Select("SELECT \* FROM student")  public abstract List<Student> selectAll();  //新增操作  @Insert("INSERT INTO student VALUES (#{id},#{name},#{age})")  public abstract Integer insert(Student stu);  //修改操作  @Update("UPDATE student SET name=#{name},age=#{age} WHERE id=#{id}")  public abstract Integer update(Student stu);  //删除操作  @Delete("DELETE FROM student WHERE id=#{id}")  public abstract Integer delete(Integer id);  } |

Test01.java

|  |
| --- |
| public class Test01 {  @Test  public void delete() throws Exception{  //1.加载核心配置文件  InputStream is = Resources.getResourceAsStream("MyBatisConfig.xml");  //2.获取SqlSession工厂对象  SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(is);  //3.通过工厂对象获取SqlSession对象  SqlSession sqlSession = sqlSessionFactory.openSession(true);  //4.获取StudentMapper接口的实现类对象  StudentMapper mapper = sqlSession.getMapper(StudentMapper.class);  //5.调用实现类对象中的方法，接收结果  Integer result = mapper.delete(4);  //6.处理结果  System.out.println(result);  //7.释放资源  sqlSession.close();  is.close();  }  } |

## 1.6（掌握）注解开发小结



# 2（掌握）MyBatis注解实现多表操作

## 2.1（了解）一对一的环境介绍

### 数据准备

Person.java

|  |
| --- |
| public class Person {  private Integer id; //主键id  private String name; //人的姓名  private Integer age; //人的年龄  public Person() {  }  public Person(Integer id, String name, Integer age) {  this.id = id;  this.name = name;  this.age = age;  }  public Integer getId() {  return id;  }  public void setId(Integer id) {  this.id = id;  }  public String getName() {  return name;  }  public void setName(String name) {  this.name = name;  }  public Integer getAge() {  return age;  }  public void setAge(Integer age) {  this.age = age;  }  @Override  public String toString() {  return "Person{" +  "id=" + id +  ", name='" + name + '\'' +  ", age=" + age +  '}';  }  } |

Card.java

|  |
| --- |
| public class Card {  private Integer id; //主键id  private String number; //身份证号  private Person p; //所属人的对象  public Card() {  }  public Card(Integer id, String number, Person p) {  this.id = id;  this.number = number;  this.p = p;  }  public Integer getId() {  return id;  }  public void setId(Integer id) {  this.id = id;  }  public String getNumber() {  return number;  }  public void setNumber(String number) {  this.number = number;  }  public Person getP() {  return p;  }  public void setP(Person p) {  this.p = p;  }  @Override  public String toString() {  return "Card{" +  "id=" + id +  ", number='" + number + '\'' +  ", p=" + p +  '}';  }  } |

CardMapper.java

|  |
| --- |
| public interface CardMapper {  //查询全部  public abstract List<Card> selectAll();  } |

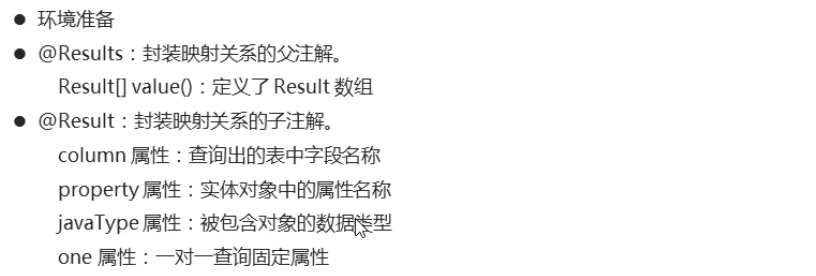
PersonMapper.java

|  |
| --- |
| public interface PersonMapper {  //根据id查询  @Select("SELECT \* FROM person WHERE id=#{id}")  public abstract Person selectById(Integer id);  } |

Test01.java

|  |
| --- |
| public class Test01 {  @Test  public void selectAll() throws Exception{  //1.加载核心配置文件  InputStream is = Resources.getResourceAsStream("MyBatisConfig.xml");  //2.获取SqlSession工厂对象  SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(is);  //3.通过工厂对象获取SqlSession对象  SqlSession sqlSession = sqlSessionFactory.openSession(true);  //4.获取CardMapper接口的实现类对象  CardMapper mapper = sqlSession.getMapper(CardMapper.class);  //5.调用实现类对象中的方法，接收结果  List<Card> list = mapper.selectAll();  //6.处理结果  for (Card card : list) {  System.out.println(card);  }  //7.释放资源  sqlSession.close();  is.close();  }  } |

## 2.2（掌握）一对一的实现



### 源码

CardMapper.java

|  |
| --- |
| public interface CardMapper {  //查询全部  @Select("SELECT \* FROM card")  @Results({  @Result(column = "id",property = "id"),  @Result(column = "number",property = "number"),  @Result(  property = "p", // 被包含对象的变量名  javaType = Person.class, // 被包含对象的实际数据类型  column = "pid", // 根据查询出的card表中的pid字段来查询person表  /\*  one、@One 一对一固定写法  select属性：指定调用哪个接口中的哪个方法  \*/  one = @One(select = "com.itheima.one\_to\_one.PersonMapper.selectById")  )  })  public abstract List<Card> selectAll();  } |

## 2.3（了解）一对多的环境介绍

### 数据准备

Student.java

|  |
| --- |
| public class Student {  private Integer id; //主键id  private String name; //学生姓名  private Integer age; //学生年龄  public Student() {  }  public Student(Integer id, String name, Integer age) {  this.id = id;  this.name = name;  this.age = age;  }  public Integer getId() {  return id;  }  public void setId(Integer id) {  this.id = id;  }  public String getName() {  return name;  }  public void setName(String name) {  this.name = name;  }  public Integer getAge() {  return age;  }  public void setAge(Integer age) {  this.age = age;  }  @Override  public String toString() {  return "Student{" +  "id=" + id +  ", name='" + name + '\'' +  ", age=" + age +  '}';  }  } |

Classes.java

|  |
| --- |
| public class Classes {  private Integer id; //主键id  private String name; //班级名称  private List<Student> students; //班级中所有学生对象  public Classes() {  }  public Classes(Integer id, String name, List<Student> students) {  this.id = id;  this.name = name;  this.students = students;  }  public Integer getId() {  return id;  }  public void setId(Integer id) {  this.id = id;  }  public String getName() {  return name;  }  public void setName(String name) {  this.name = name;  }  public List<Student> getStudents() {  return students;  }  public void setStudents(List<Student> students) {  this.students = students;  }  @Override  public String toString() {  return "Classes{" +  "id=" + id +  ", name='" + name + '\'' +  ", students=" + students +  '}';  }  } |

ClassesMapper.java

|  |
| --- |
| public interface ClassesMapper {  //查询全部  public abstract List<Classes> selectAll();  } |

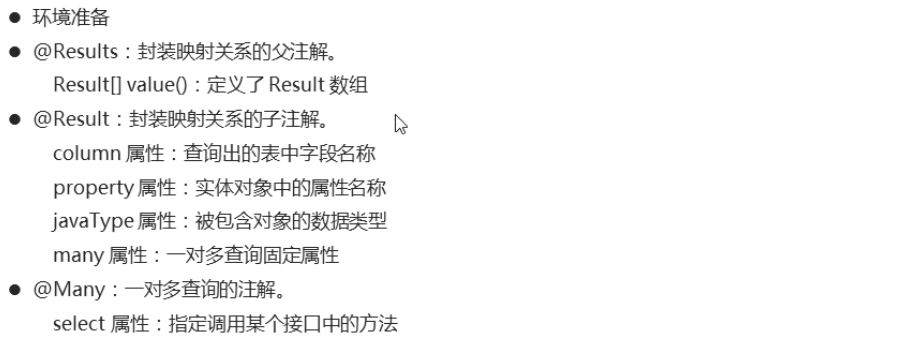
StudentMapper.java

|  |
| --- |
| public interface StudentMapper {  //根据cid查询student表  @Select("SELECT \* FROM student WHERE cid=#{cid}")  public abstract List<Student> selectByCid(Integer cid);  } |

Test01.java

|  |
| --- |
| public class Test01 {  @Test  public void selectAll() throws Exception{  //1.加载核心配置文件  InputStream is = Resources.getResourceAsStream("MyBatisConfig.xml");  //2.获取SqlSession工厂对象  SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(is);  //3.通过工厂对象获取SqlSession对象  SqlSession sqlSession = sqlSessionFactory.openSession(true);  //4.获取ClassesMapper接口的实现类对象  ClassesMapper mapper = sqlSession.getMapper(ClassesMapper.class);  //5.调用实现类对象中的方法，接收结果  List<Classes> list = mapper.selectAll();  //6.处理结果  for (Classes cls : list) {  System.out.println(cls.getId() + "," + cls.getName());  List<Student> students = cls.getStudents();  for (Student student : students) {  System.out.println("\t" + student);  }  }  //7.释放资源  sqlSession.close();  is.close();  }  } |

## 2.4（掌握）一对多的实现



### 源码

ClassesMapper.java

|  |
| --- |
| public interface ClassesMapper {  //查询全部  @Select("SELECT \* FROM classes")  @Results({  @Result(column = "id",property = "id"),  @Result(column = "name",property = "name"),  @Result(  property = "students", // 被包含对象的变量名  javaType = List.class, // 被包含对象的实际数据类型  column = "id", // 根据查询出的classes表的id字段来查询student表  /\*  many、@Many 一对多查询的固定写法  select属性：指定调用哪个接口中的哪个查询方法  \*/  many = @Many(select = "com.itheima.one\_to\_many.StudentMapper.selectByCid")  )  })  public abstract List<Classes> selectAll();  } |

|  |
| --- |
|  |

## 2.5（了解）多对多的环境介绍

### 数据准备

Course.java

|  |
| --- |
| public class Course {  private Integer id; //主键id  private String name; //课程名称  public Course() {  }  public Course(Integer id, String name) {  this.id = id;  this.name = name;  }  public Integer getId() {  return id;  }  public void setId(Integer id) {  this.id = id;  }  public String getName() {  return name;  }  public void setName(String name) {  this.name = name;  }  @Override  public String toString() {  return "Course{" +  "id=" + id +  ", name='" + name + '\'' +  '}';  }  } |

Student.java

|  |
| --- |
| public class Student {  private Integer id; //主键id  private String name; //学生姓名  private Integer age; //学生年龄  private List<Course> courses; //学生所选择的课程对象  public Student() {  }  public Student(Integer id, String name, Integer age, List<Course> courses) {  this.id = id;  this.name = name;  this.age = age;  this.courses = courses;  }  public List<Course> getCourses() {  return courses;  }  public void setCourses(List<Course> courses) {  this.courses = courses;  }  public Integer getId() {  return id;  }  public void setId(Integer id) {  this.id = id;  }  public String getName() {  return name;  }  public void setName(String name) {  this.name = name;  }  public Integer getAge() {  return age;  }  public void setAge(Integer age) {  this.age = age;  }  @Override  public String toString() {  return "Student{" +  "id=" + id +  ", name='" + name + '\'' +  ", age=" + age +  '}';  }  } |

StudentMapper.java

|  |
| --- |
| public interface StudentMapper {  //查询全部  public abstract List<Student> selectAll();  } |

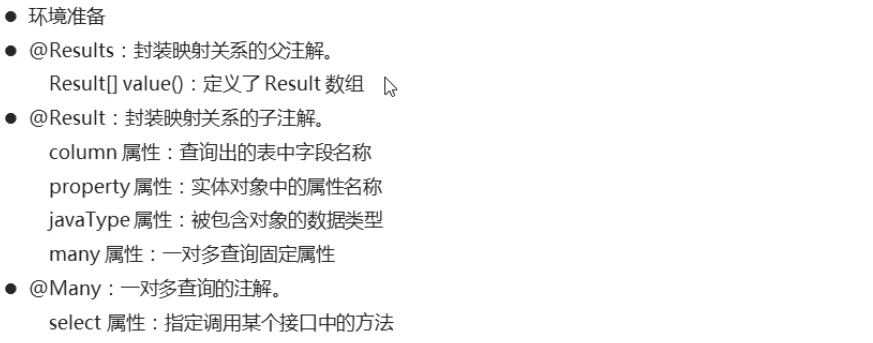
CourseMapper.java

|  |
| --- |
| public interface CourseMapper {  //根据学生id查询所选课程  @Select("SELECT c.id,c.name FROM stu\_cr sc,course c WHERE sc.cid=c.id AND sc.sid=#{id}")  public abstract List<Course> selectBySid(Integer id);  } |

Test01.java

|  |
| --- |
| public class Test01 {  @Test  public void selectAll() throws Exception{  //1.加载核心配置文件  InputStream is = Resources.getResourceAsStream("MyBatisConfig.xml");  //2.获取SqlSession工厂对象  SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(is);  //3.通过工厂对象获取SqlSession对象  SqlSession sqlSession = sqlSessionFactory.openSession(true);  //4.获取StudentMapper接口的实现类对象  StudentMapper mapper = sqlSession.getMapper(StudentMapper.class);  //5.调用实现类对象中的方法，接收结果  List<Student> list = mapper.selectAll();  //6.处理结果  for (Student student : list) {  System.out.println(student.getId() + "," + student.getName() + "," + student.getAge());  List<Course> courses = student.getCourses();  for (Course cours : courses) {  System.out.println("\t" + cours);  }  }  //7.释放资源  sqlSession.close();  is.close();  }  } |

## 2.6（掌握）多对多的实现

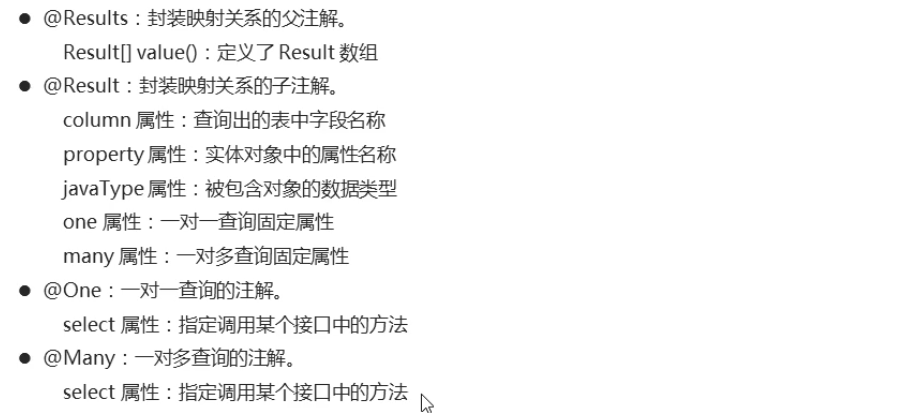


### 源码

StudentMapper.java

|  |
| --- |
| public interface StudentMapper {  //查询全部  @Select("SELECT DISTINCT s.id,s.name,s.age FROM student s,stu\_cr sc WHERE sc.sid=s.id")  @Results({  @Result(column = "id",property = "id"),  @Result(column = "name",property = "name"),  @Result(column = "age",property = "age"),  @Result(  property = "courses", // 被包含对象的变量名  javaType = List.class, // 被包含对象的实际数据类型  column = "id", // 根据查询出student表的id来作为关联条件，去查询中间表和课程表  /\*  many、@Many 一对多查询的固定写法  select属性：指定调用哪个接口中的哪个查询方法  \*/  many = @Many(select = "com.itheima.many\_to\_many.CourseMapper.selectBySid")  )  })  public abstract List<Student> selectAll();  } |

## 2.7（掌握）注解多表操作的小结



# 3（了解）MyBatis构建SQL语句

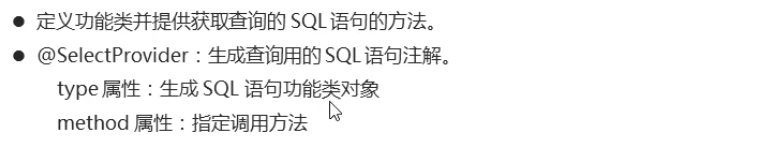
## 3.1（了解）SQL功能类的简介



### 源码

|  |
| --- |
| public class SqlTest {  public static void main(String[] args) {  String sql = getSql();  System.out.println(sql);  }  //定义方法，获取查询student表的sql语句  /\*public static String getSql() {  String sql = "SELECT \* FROM student";  return sql;  }\*/  public static String getSql() {  String sql = new SQL(){  {  SELECT("\*");  FROM("student");  }  }.toString();  return sql;  }  } |

## 3.2（了解）查询功能的实现



### 源码

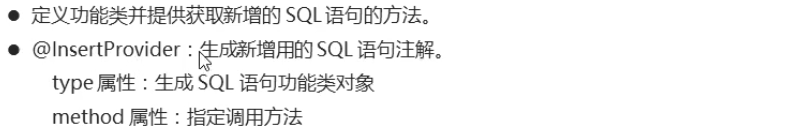
ReturnSql.java

|  |
| --- |
| public class ReturnSql {  //定义方法，返回查询的sql语句  public String getSelectAll() {  return new SQL() {  {  SELECT("\*");  FROM("student");  }  }.toString();  }  } |

StudentMapper.java

|  |
| --- |
| public interface StudentMapper {  //查询全部  //@Select("SELECT \* FROM student")  @SelectProvider(type = ReturnSql.class , method = "getSelectAll")  public abstract List<Student> selectAll();  } |

## 3.3（了解）新增功能的实现



### 源码

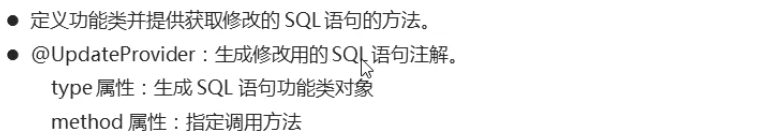
ReturnSql.java

|  |
| --- |
| public class ReturnSql {  //定义方法，返回新增的sql语句  public String getInsert(Student stu) {  return new SQL() {  {  INSERT\_INTO("student");  INTO\_VALUES("#{id},#{name},#{age}");  }  }.toString();  }  } |

StudentMapper.java

|  |
| --- |
| public interface StudentMapper {  //新增功能  //@Insert("INSERT INTO student VALUES (#{id},#{name},#{age})")  @InsertProvider(type = ReturnSql.class , method = "getInsert")  public abstract Integer insert(Student stu);  } |

## 3.4（了解）修改功能的实现



### 源码

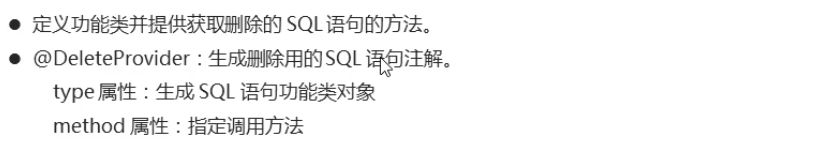
ReturnSql.java

|  |
| --- |
| public class ReturnSql {  //定义方法，返回修改的sql语句  public String getUpdate(Student stu) {  return new SQL() {  {  UPDATE("student");  SET("name=#{name}","age=#{age}");  WHERE("id=#{id}");  }  }.toString();  }  } |

StudentMapper.java

|  |
| --- |
| public interface StudentMapper {  //修改功能  //@Update("UPDATE student SET name=#{name},age=#{age} WHERE id=#{id}")  @UpdateProvider(type = ReturnSql.class , method = "getUpdate")  public abstract Integer update(Student stu);  } |

## 3.5（了解）删除功能的实现



### 源码

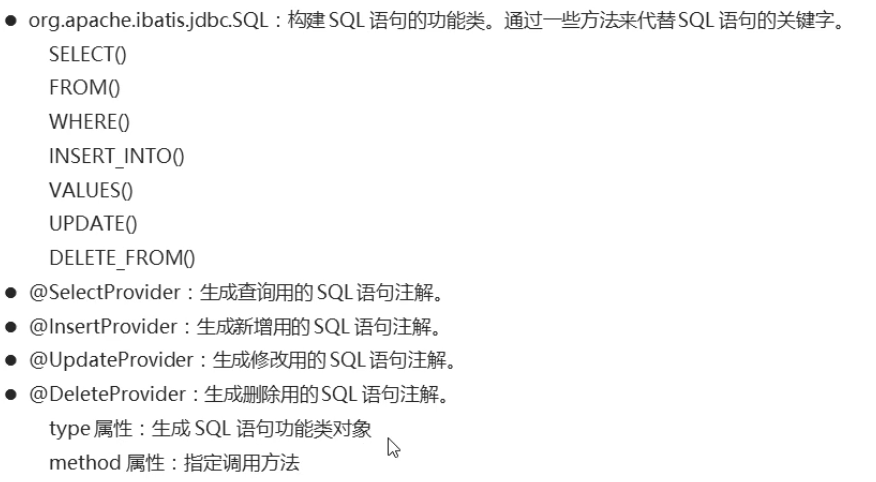
ReturnSql.java

|  |
| --- |
| public class ReturnSql {  //定义方法，返回删除的sql语句  public String getDelete(Integer id) {  return new SQL() {  {  DELETE\_FROM("student");  WHERE("id=#{id}");  }  }.toString();  }  } |

StudentMapper.java

|  |
| --- |
| public interface StudentMapper {  //删除功能  //@Delete("DELETE FROM student WHERE id=#{id}")  @DeleteProvider(type = ReturnSql.class , method = "getDelete")  public abstract Integer delete(Integer id);  } |

## 3.6（了解）构建SQL小结



# 4（掌握）MyBatis学生管理系统

## 4.1（掌握）学生管理系统的介绍和环境搭建

导入JDBC基础网页版

添加log4j的jar包

### 源码

log4j.properties

|  |
| --- |
| # Global logging configuration  # ERROR WARN INFO DEBUG  log4j.rootLogger=DEBUG, stdout  # Console output...  log4j.appender.stdout=org.apache.log4j.ConsoleAppender  log4j.appender.stdout.layout=org.apache.log4j.PatternLayout  log4j.appender.stdout.layout.ConversionPattern=%5p [%t] - %m%n |

MyBatisConfig.xml

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8" ?>  <!--MyBatis的DTD约束-->  <!DOCTYPE configuration PUBLIC "-//mybatis.org//DTD Config 3.0//EN" "http://mybatis.org/dtd/mybatis-3-config.dtd">  <!--configuration 核心根标签-->  <configuration>  <!--引入数据库连接的配置文件-->  <properties resource="config.properties"/>  <!--配置LOG4J-->  <settings>  <setting name="logImpl" value="log4j"/>  </settings>  <!--environments配置数据库环境，环境可以有多个。default属性指定使用的是哪个-->  <environments default="mysql">  <!--environment配置数据库环境 id属性唯一标识-->  <environment id="mysql">  <!-- transactionManager事务管理。 type属性，采用JDBC默认的事务-->  <transactionManager type="JDBC"></transactionManager>  <!-- dataSource数据源信息 type属性 连接池-->  <dataSource type="POOLED">  <!-- property获取数据库连接的配置信息 -->  <property name="driver" value="${driver}" />  <property name="url" value="${url}" />  <property name="username" value="${username}" />  <property name="password" value="${password}" />  </dataSource>  </environment>  </environments>  <!--配置映射关系-->  <mappers>  <package name="com.itheima"/>  </mappers>  </configuration> |

StudentDao.java

|  |
| --- |
| public interface StudentDao {  //查询所有学生信息  @Select("SELECT \* FROM student")  public abstract ArrayList<Student> findAll();  //条件查询，根据id获取学生信息  @Select("SELECT \* FROM student WHERE sid=#{sid}")  public abstract Student findById(Integer sid);  //新增学生信息  @Insert("INSERT INTO student VALUES (#{sid},#{name},#{age},#{birthday})")  public abstract int insert(Student stu);  //修改学生信息  @Update("UPDATE student SET name=#{name},age=#{age},birthday=#{birthday} WHERE sid=#{sid}")  public abstract int update(Student stu);  //删除学生信息  @Delete("DELETE FROM student WHERE sid=#{sid}")  public abstract int delete(Integer sid);  } |

StudentServiceImpl.java

|  |
| --- |
| public class StudentServiceImpl implements StudentService {  @Override  public List<Student> findAll() {  }  @Override  public Student findById(Integer sid) {  }  @Override  public void save(Student student) {  }  }  @Override  public void update(Student student) {  }  @Override  public void delete(Integer sid) {  }  } |

## 4.2（掌握）学生管理系统的实现

### 源码

StudentServiceImpl.java

|  |
| --- |
| public class StudentServiceImpl implements StudentService {  @Override  public List<Student> findAll() {  ArrayList<Student> list = null;  SqlSession sqlSession = null;  InputStream is = null;  try{  //1.加载核心配置文件  is = Resources.getResourceAsStream("MyBatisConfig.xml");  //2.获取SqlSession工厂对象  SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(is);  //3.通过工厂对象获取SqlSession对象  sqlSession = sqlSessionFactory.openSession(true);  //4.获取StudentDao接口的实现类对象  StudentDao mapper = sqlSession.getMapper(StudentDao.class);  //5.调用实现类对象的方法，接收结果  list = mapper.findAll();  } catch (Exception e) {  e.printStackTrace();  } finally {  //6.释放资源  if(sqlSession != null) {  sqlSession.close();  }  if(is != null) {  try {  is.close();  } catch (IOException e) {  e.printStackTrace();  }  }  }  //7.返回结果  return list;  }  @Override  public Student findById(Integer sid) {  Student stu = null;  SqlSession sqlSession = null;  InputStream is = null;  try{  //1.加载核心配置文件  is = Resources.getResourceAsStream("MyBatisConfig.xml");  //2.获取SqlSession工厂对象  SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(is);  //3.通过工厂对象获取SqlSession对象  sqlSession = sqlSessionFactory.openSession(true);  //4.获取StudentDao接口的实现类对象  StudentDao mapper = sqlSession.getMapper(StudentDao.class);  //5.调用实现类对象的方法，接收结果  stu = mapper.findById(sid);  } catch (Exception e) {  e.printStackTrace();  } finally {  //6.释放资源  if(sqlSession != null) {  sqlSession.close();  }  if(is != null) {  try {  is.close();  } catch (IOException e) {  e.printStackTrace();  }  }  }  //7.返回结果  return stu;  }  @Override  public void save(Student student) {  SqlSession sqlSession = null;  InputStream is = null;  try{  //1.加载核心配置文件  is = Resources.getResourceAsStream("MyBatisConfig.xml");  //2.获取SqlSession工厂对象  SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(is);  //3.通过工厂对象获取SqlSession对象  sqlSession = sqlSessionFactory.openSession(true);  //4.获取StudentDao接口的实现类对象  StudentDao mapper = sqlSession.getMapper(StudentDao.class);  //5.调用实现类对象的方法，接收结果  mapper.insert(student);  } catch (Exception e) {  e.printStackTrace();  } finally {  //6.释放资源  if(sqlSession != null) {  sqlSession.close();  }  if(is != null) {  try {  is.close();  } catch (IOException e) {  e.printStackTrace();  }  }  }  }  @Override  public void update(Student student) {  SqlSession sqlSession = null;  InputStream is = null;  try{  //1.加载核心配置文件  is = Resources.getResourceAsStream("MyBatisConfig.xml");  //2.获取SqlSession工厂对象  SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(is);  //3.通过工厂对象获取SqlSession对象  sqlSession = sqlSessionFactory.openSession(true);  //4.获取StudentDao接口的实现类对象  StudentDao mapper = sqlSession.getMapper(StudentDao.class);  //5.调用实现类对象的方法，接收结果  mapper.update(student);  } catch (Exception e) {  e.printStackTrace();  } finally {  //6.释放资源  if(sqlSession != null) {  sqlSession.close();  }  if(is != null) {  try {  is.close();  } catch (IOException e) {  e.printStackTrace();  }  }  }  }  @Override  public void delete(Integer sid) {  SqlSession sqlSession = null;  InputStream is = null;  try{  //1.加载核心配置文件  is = Resources.getResourceAsStream("MyBatisConfig.xml");  //2.获取SqlSession工厂对象  SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(is);  //3.通过工厂对象获取SqlSession对象  sqlSession = sqlSessionFactory.openSession(true);  //4.获取StudentDao接口的实现类对象  StudentDao mapper = sqlSession.getMapper(StudentDao.class);  //5.调用实现类对象的方法，接收结果  mapper.delete(sid);  } catch (Exception e) {  e.printStackTrace();  } finally {  //6.释放资源  if(sqlSession != null) {  sqlSession.close();  }  if(is != null) {  try {  is.close();  } catch (IOException e) {  e.printStackTrace();  }  }  }  }  } |

|  |
| --- |
|  |