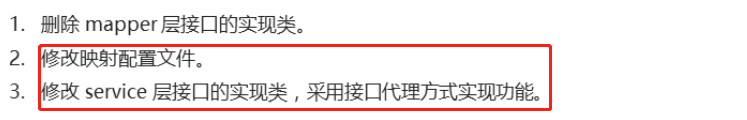
02 MyBatis进阶

# 1（掌握）MyBatis接口代理方式实现Dao层

## 1.1（掌握）实现规则的介绍



## 1.2（掌握）代码的实现



### 源码

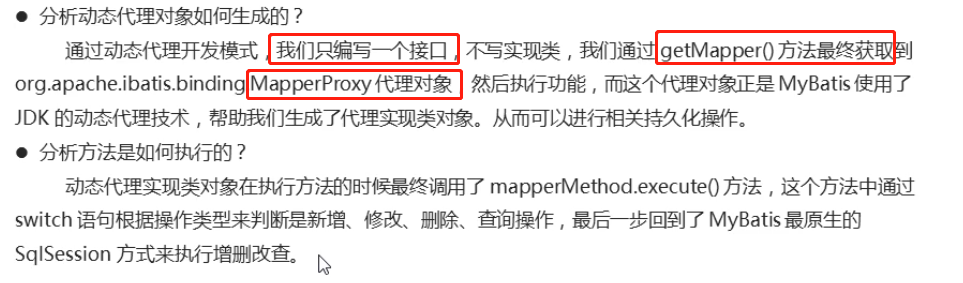
StudengMapper.xml

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8" ?>  <!--MyBatis的DTD约束-->  <!DOCTYPE mapper  PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"  "http://mybatis.org/dtd/mybatis-3-mapper.dtd">  <!--  mapper：核心根标签  namespace属性：名称空间  -->  <mapper namespace="com.itheima.mapper.StudentMapper">  <!--  select：查询功能的标签  id属性：唯一标识  resultType属性：指定结果映射对象类型  parameterType属性：指定参数映射对象类型  -->  <select id="selectAll" resultType="student">  SELECT \* FROM student  </select>  <select id="selectById" resultType="student" parameterType="int">  SELECT \* FROM student WHERE id = #{id}  </select>  <insert id="insert" parameterType="student">  INSERT INTO student VALUES (#{id},#{name},#{age})  </insert>  <update id="update" parameterType="student">  UPDATE student SET name = #{name},age = #{age} WHERE id = #{id}  </update>  <delete id="delete" parameterType="int">  DELETE FROM student WHERE id = #{id}  </delete>  </mapper> |

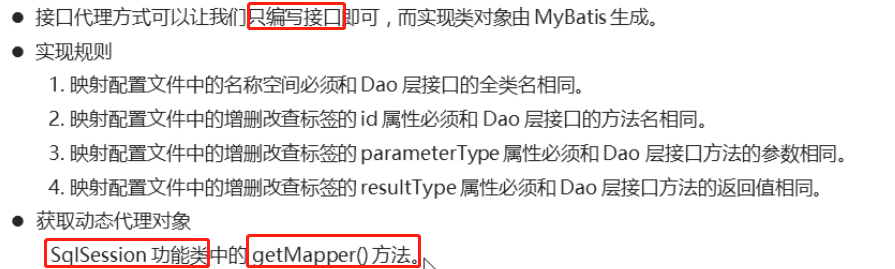
StudentServiceImpl.java

|  |
| --- |
| /\*  业务层实现类  \*/  public class StudentServiceImpl implements StudentService {  @Override  public List<Student> selectAll() {  List<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StudentMapper接口的实现类对象  StudentMapper mapper = sqlSession.getMapper(StudentMapper.class);  // StudentMapper mapper = new StudentMapperImpl();  //5.通过实现类对象调用方法，接收结果  list = mapper.selectAll();  } catch (Exception e) {  } 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 selectById(Integer id) {  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StudentMapper接口的实现类对象  StudentMapper mapper = sqlSession.getMapper(StudentMapper.class); // StudentMapper mapper = new StudentMapperImpl();  //5.通过实现类对象调用方法，接收结果  stu = mapper.selectById(id);  } catch (Exception e) {  } 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 Integer insert(Student stu) {  Integer result = 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StudentMapper接口的实现类对象  StudentMapper mapper = sqlSession.getMapper(StudentMapper.class); // StudentMapper mapper = new StudentMapperImpl();  //5.通过实现类对象调用方法，接收结果  result = mapper.insert(stu);  } catch (Exception e) {  } 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 result;  }  @Override  public Integer update(Student stu) {  Integer result = 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StudentMapper接口的实现类对象  StudentMapper mapper = sqlSession.getMapper(StudentMapper.class); // StudentMapper mapper = new StudentMapperImpl();  //5.通过实现类对象调用方法，接收结果  result = mapper.update(stu);  } catch (Exception e) {  } 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 result;  }  @Override  public Integer delete(Integer id) {  Integer result = 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StudentMapper接口的实现类对象  StudentMapper mapper = sqlSession.getMapper(StudentMapper.class); // StudentMapper mapper = new StudentMapperImpl();  //5.通过实现类对象调用方法，接收结果  result = mapper.delete(id);  } catch (Exception e) {  } 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 result;  }  } |

## 1.3（了解）源码分析

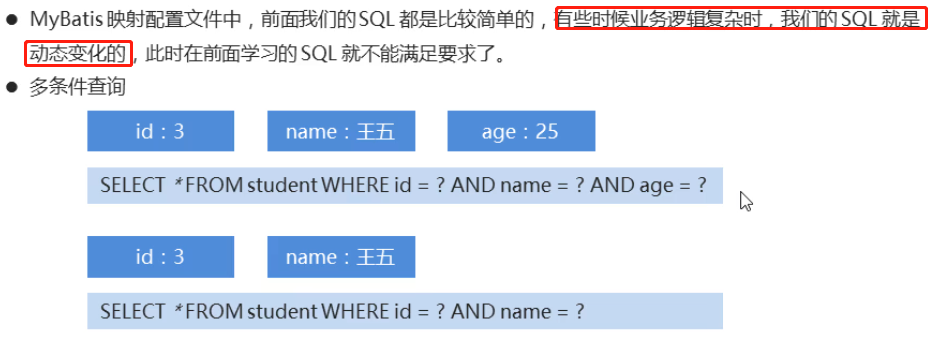


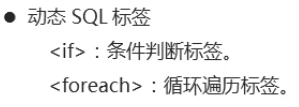
## 1.4（掌握）小结



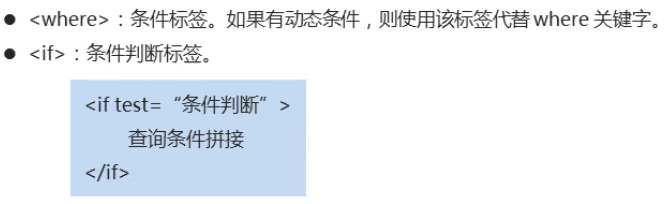
# 2（掌握）MyBatis映射配置文件-动态SQL

## 2.1（掌握）动态SQL的介绍





## 2.2（掌握）if标签的使用



### 源码

StudentMapper.java

|  |
| --- |
| /\*  持久层接口  \*/  public interface StudentMapper {  //查询全部  public abstract List<Student> selectAll();  //根据id查询  public abstract Student selectById(Integer id);  //新增数据  public abstract Integer insert(Student stu);  //修改数据  public abstract Integer update(Student stu);  //删除数据  public abstract Integer delete(Integer id);  //多条件查询  public abstract List<Student> selectCondition(Student stu);  } |

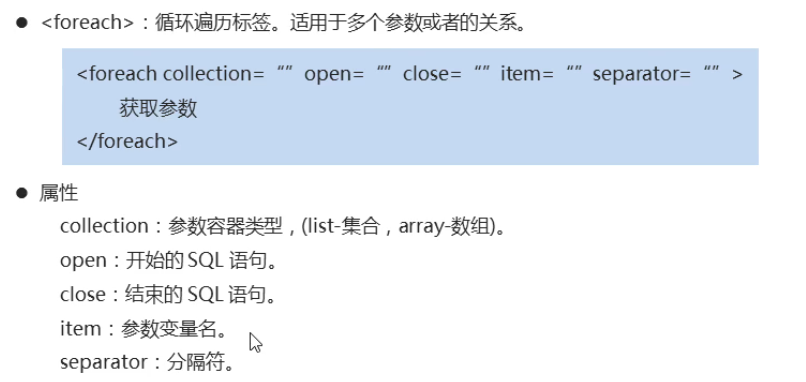
StudentMapper.xml

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8" ?>  <!--MyBatis的DTD约束-->  <!DOCTYPE mapper  PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"  "http://mybatis.org/dtd/mybatis-3-mapper.dtd">  <!--  mapper：核心根标签  namespace属性：名称空间  -->  <mapper namespace="com.itheima.mapper.StudentMapper">  <!--  select：查询功能的标签  id属性：唯一标识  resultType属性：指定结果映射对象类型  parameterType属性：指定参数映射对象类型  -->  <select id="selectAll" resultType="student">  SELECT \* FROM student  </select>  <select id="selectById" resultType="student" parameterType="int">  SELECT \* FROM student WHERE id = #{id}  </select>  <insert id="insert" parameterType="student">  INSERT INTO student VALUES (#{id},#{name},#{age})  </insert>  <update id="update" parameterType="student">  UPDATE student SET name = #{name},age = #{age} WHERE id = #{id}  </update>  <delete id="delete" parameterType="int">  DELETE FROM student WHERE id = #{id}  </delete>  <select id="selectCondition" resultType="student" parameterType="student">  SELECT \* FROM student  <where>  <if test="id != null">  id = #{id}  </if>  <if test="name != null">  AND name = #{name}  </if>  <if test="age != null">  AND age = #{age}  </if>  </where>  </select>  </mapper> |

Test01.java

|  |
| --- |
| public class Test01 {  @Test  public void selectCondition() 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);  Student stu = new Student();  stu.setId(2);  stu.setName("李四");  //stu.setAge(24);  //5.调用实现类的方法，接收结果  List<Student> list = mapper.selectCondition(stu);  //6.处理结果  for (Student student : list) {  System.out.println(student);  }  //7.释放资源  sqlSession.close();  is.close();  }  } |

## 2.3（掌握）foreach标签的使用



### 源码

StudentMapper.java

|  |
| --- |
| /\*  持久层接口  \*/  public interface StudentMapper {  //查询全部  public abstract List<Student> selectAll();  //根据id查询  public abstract Student selectById(Integer id);  //新增数据  public abstract Integer insert(Student stu);  //修改数据  public abstract Integer update(Student stu);  //删除数据  public abstract Integer delete(Integer id);  //多条件查询  public abstract List<Student> selectCondition(Student stu);  //根据多个id查询  public abstract List<Student> selectByIds(List<Integer> ids);  } |

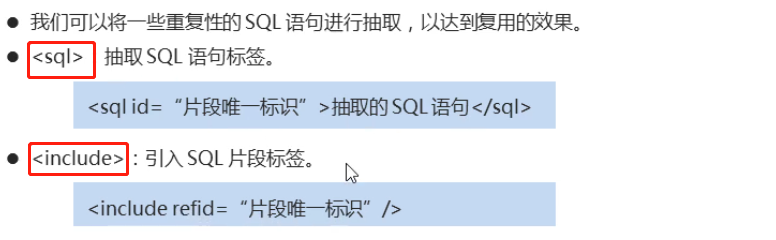
StudentMapper.xml

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8" ?>  <!--MyBatis的DTD约束-->  <!DOCTYPE mapper  PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"  "http://mybatis.org/dtd/mybatis-3-mapper.dtd">  <!--  mapper：核心根标签  namespace属性：名称空间  -->  <mapper namespace="com.itheima.mapper.StudentMapper">  <!--  select：查询功能的标签  id属性：唯一标识  resultType属性：指定结果映射对象类型  parameterType属性：指定参数映射对象类型  -->  <select id="selectAll" resultType="student">  SELECT \* FROM student  </select>  <select id="selectById" resultType="student" parameterType="int">  SELECT \* FROM student WHERE id = #{id}  </select>  <insert id="insert" parameterType="student">  INSERT INTO student VALUES (#{id},#{name},#{age})  </insert>  <update id="update" parameterType="student">  UPDATE student SET name = #{name},age = #{age} WHERE id = #{id}  </update>  <delete id="delete" parameterType="int">  DELETE FROM student WHERE id = #{id}  </delete>  <select id="selectCondition" resultType="student" parameterType="student">  SELECT \* FROM student  <where>  <if test="id != null">  id = #{id}  </if>  <if test="name != null">  AND name = #{name}  </if>  <if test="age != null">  AND age = #{age}  </if>  </where>  </select>  <select id="selectByIds" resultType="student" parameterType="list">  SELECT \* FROM student  <where>  <foreach collection="list" open="id IN (" close=")" item="id" separator=",">  #{id}  </foreach>  </where>  </select>  </mapper> |

Test01.java

|  |
| --- |
| public class Test01 {  @Test  public void selectByIds() 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);  List<Integer> ids = new ArrayList<>();  ids.add(1);  ids.add(2);  ids.add(3);  //5.调用实现类的方法，接收结果  List<Student> list = mapper.selectByIds(ids);  //6.处理结果  for (Student student : list) {  System.out.println(student);  }  //7.释放资源  sqlSession.close();  is.close();  }  } |

## 2.4（掌握）sql片段的抽取

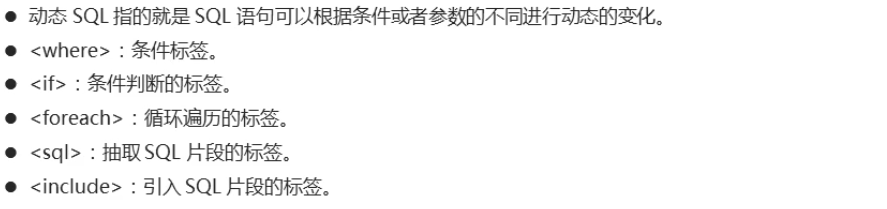


### 源码

StudentMapper.xml

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8" ?>  <!--MyBatis的DTD约束-->  <!DOCTYPE mapper  PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"  "http://mybatis.org/dtd/mybatis-3-mapper.dtd">  <!--  mapper：核心根标签  namespace属性：名称空间  -->  <mapper namespace="com.itheima.mapper.StudentMapper">  <sql id="select" >SELECT \* FROM student</sql>  <!--  select：查询功能的标签  id属性：唯一标识  resultType属性：指定结果映射对象类型  parameterType属性：指定参数映射对象类型  -->  <select id="selectAll" resultType="student">  <include refid="select"/>  </select>  <select id="selectById" resultType="student" parameterType="int">  <include refid="select"/> WHERE id = #{id}  </select>  <insert id="insert" parameterType="student">  INSERT INTO student VALUES (#{id},#{name},#{age})  </insert>  <update id="update" parameterType="student">  UPDATE student SET name = #{name},age = #{age} WHERE id = #{id}  </update>  <delete id="delete" parameterType="int">  DELETE FROM student WHERE id = #{id}  </delete>  <select id="selectCondition" resultType="student" parameterType="student">  <include refid="select"/>  <where>  <if test="id != null">  id = #{id}  </if>  <if test="name != null">  AND name = #{name}  </if>  <if test="age != null">  AND age = #{age}  </if>  </where>  </select>  <select id="selectByIds" resultType="student" parameterType="list">  <include refid="select"/>  <where>  <foreach collection="list" open="id IN (" close=")" item="id" separator=",">  #{id}  </foreach>  </where>  </select>  </mapper> |

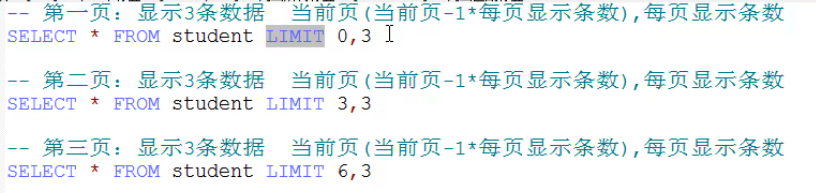
## 2.5（掌握）动态SQL小结

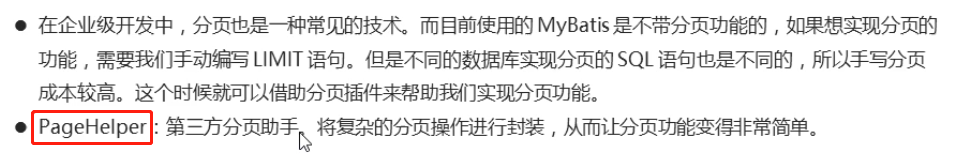


# 3（掌握）MyBatis核心配置文件-分页插件

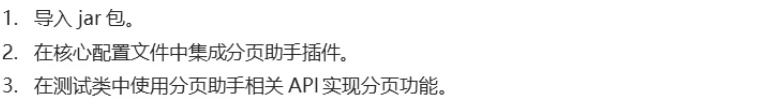
## 3.1（掌握）分页插件的介绍







## 3.2（掌握）分页插件的使用





### 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>  <typeAlias type="com.itheima.bean.Student" alias="student"/>  <!--<package name="com.itheima.bean"/>-->  </typeAliases>  <!--集成分页助手插件-->  <plugins>  <plugin interceptor="com.github.pagehelper.PageInterceptor"></plugin>  </plugins>  <!--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>  <!-- mappers引入映射配置文件 -->  <mappers>  <!-- mapper 引入指定的映射配置文件 resource属性指定映射配置文件的名称 -->  <mapper resource="StudentMapper.xml"/>  </mappers>  </configuration> |

Test01.java

|  |
| --- |
| public class Test01 {  @Test  public void selectPaging() 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);  //通过分页助手来实现分页功能  // 第一页：显示3条数据  //PageHelper.startPage(1,3);  // 第二页：显示3条数据  //PageHelper.startPage(2,3);  // 第三页：显示3条数据  PageHelper.startPage(3,3);  //5.调用实现类的方法，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();  }  } |

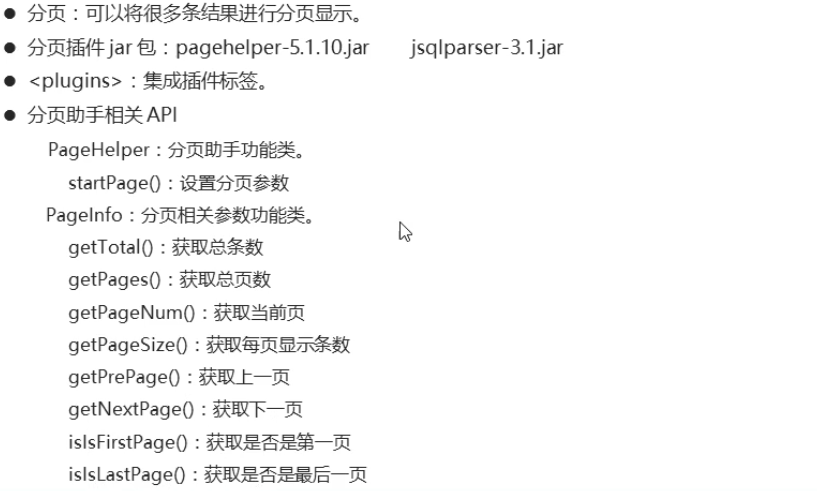
## 3.3（掌握）分页参数的获取



### 源码

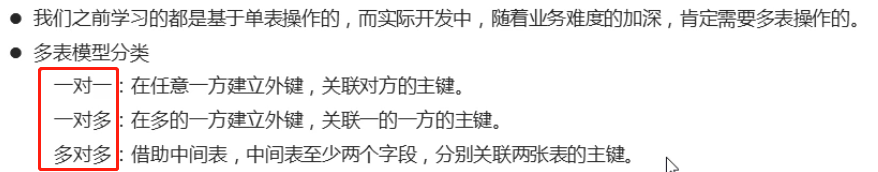
|  |
| --- |
| public class Test01 {  @Test  public void selectPaging() throws Exception{  //1.加载核心配置文件  InputStream is = Resources.getResourceAsStream("MyBatisConfig.xml");  //2.获取SqlSession工厂对象  SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(is);  //3.通过工厂对象获取SqlSession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);  //通过分页助手来实现分页功能  // 第一页：显示3条数据  //PageHelper.startPage(1,3);  // 第二页：显示3条数据  //PageHelper.startPage(2,3);  // 第三页：显示3条数据  PageHelper.startPage(3,3);  //5.调用实现类的方法，接收结果  List<Student> list = mapper.selectAll();  //6.处理结果  for (Student student : list) {  System.out.println(student);  }  //获取分页相关参数  PageInfo<Student> info = new PageInfo<>(list);  System.out.println("总条数：" + info.getTotal());  System.out.println("总页数：" + info.getPages());  System.out.println("当前页：" + info.getPageNum());  System.out.println("每页显示条数：" + info.getPageSize());  System.out.println("上一页：" + info.getPrePage());  System.out.println("下一页：" + info.getNextPage());  System.out.println("是否是第一页：" + info.isIsFirstPage());  System.out.println("是否是最后一页：" + info.isIsLastPage());  //7.释放资源  sqlSession.close();  is.close();  }  } |

## 3.4（掌握）分页插件的小结

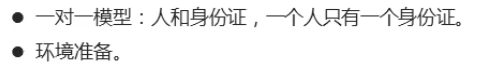


# 4（掌握）MyBatis多表操作

## 4.1（掌握）多表模型介绍



## 4.2（掌握）一对一数据的准备



### 数据准备

|  |
| --- |
| CREATE DATABASE db2;  USE db2;  CREATE TABLE person(  id INT PRIMARY KEY AUTO\_INCREMENT,  NAME VARCHAR(20),  age INT  );  INSERT INTO person VALUES (NULL,'张三',23);  INSERT INTO person VALUES (NULL,'李四',24);  INSERT INTO person VALUES (NULL,'王五',25);  CREATE TABLE card(  id INT PRIMARY KEY AUTO\_INCREMENT,  number VARCHAR(30),  pid INT,  CONSTRAINT cp\_fk FOREIGN KEY (pid) REFERENCES person(id)  );  INSERT INTO card VALUES (NULL,'12345',1);  INSERT INTO card VALUES (NULL,'23456',2);  INSERT INTO card VALUES (NULL,'34567',3); |

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>  <!-- mappers引入映射配置文件 -->  <mappers>  <mapper resource="com/itheima/one\_to\_one/OneToOneMapper.xml"/>  </mappers>  </configuration> |

jdbc.properties

|  |
| --- |
| driver=com.mysql.jdbc.Driver  url=jdbc:mysql://192.168.59.143:3306/db2  username=root  password=itheima |

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 |

## 4.3（掌握）一对一数据的功能实现

### 源码

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 +  '}';  }  } |

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 +  '}';  }  } |

OneToOneMapper.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.itheima.table01.OneToOneMapper">  <!--配置字段和实体对象属性的映射关系-->  <resultMap id="oneToOne" type="card">  <id column="cid" property="id" />  <result column="number" property="number" />  <!--  association：配置被包含对象的映射关系  property：被包含对象的变量名  javaType：被包含对象的数据类型  -->  <association property="p" javaType="person">  <id column="pid" property="id" />  <result column="name" property="name" />  <result column="age" property="age" />  </association>  </resultMap>  <select id="selectAll" resultMap="oneToOne">  SELECT c.id cid,number,pid,NAME,age FROM card c,person p WHERE c.pid=p.id  </select>  </mapper> |

OneToOneMapper.java

|  |
| --- |
| public interface OneToOneMapper {  //查询全部  public abstract List<Card> 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OneToOneMapper接口的实现类对象  OneToOneMapper mapper = sqlSession.getMapper(OneToOneMapper.class);  //5.调用实现类的方法，接收结果  List<Card> list = mapper.selectAll();  //6.处理结果  for (Card c : list) {  System.out.println(c);  }  //7.释放资源  sqlSession.close();  is.close();  }  } |

## 4.4（掌握）一对多数据的准备

### 数据准备

|  |
| --- |
| CREATE TABLE classes(  id INT PRIMARY KEY AUTO\_INCREMENT,  NAME VARCHAR(20)  );  INSERT INTO classes VALUES (NULL,'黑马一班');  INSERT INTO classes VALUES (NULL,'黑马二班');  CREATE TABLE student(  id INT PRIMARY KEY AUTO\_INCREMENT,  NAME VARCHAR(30),  age INT,  cid INT,  CONSTRAINT cs\_fk FOREIGN KEY (cid) REFERENCES classes(id)  );  INSERT INTO student VALUES (NULL,'张三',23,1);  INSERT INTO student VALUES (NULL,'李四',24,1);  INSERT INTO student VALUES (NULL,'王五',25,2);  INSERT INTO student VALUES (NULL,'赵六',26,2); |

## 4.5（掌握）一对多数据的功能实现

### 源码

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 +  '}';  }  } |

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 +  '}';  }  } |

OneToManyMapper.java

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

OneToManyMapper.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.itheima.table02.OneToManyMapper">  <resultMap id="oneToMany" type="classes">  <id column="cid" property="id"/>  <result column="cname" property="name"/>  <!--  collection：配置被包含的集合对象映射关系  property：被包含对象的变量名  ofType：被包含对象的实际数据类型  -->  <collection property="students" ofType="student">  <id column="sid" property="id"/>  <result column="sname" property="name"/>  <result column="sage" property="age"/>  </collection>  </resultMap>  <select id="selectAll" resultMap="oneToMany">  SELECT c.id cid,c.name cname,s.id sid,s.name sname,s.age sage FROM classes c,student s WHERE c.id=s.cid  </select>  </mapper> |

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OneToManyMapper接口的实现类对象  OneToManyMapper mapper = sqlSession.getMapper(OneToManyMapper.class);  //5.调用实现类的方法，接收结果  List<Classes> classes = mapper.selectAll();  //6.处理结果  for (Classes cls : classes) {  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();  }  } |

## 4.6（掌握）多对多数据的准备

### 数据准备

|  |
| --- |
| CREATE TABLE course(  id INT PRIMARY KEY AUTO\_INCREMENT,  NAME VARCHAR(20)  );  INSERT INTO course VALUES (NULL,'语文');  INSERT INTO course VALUES (NULL,'数学');  CREATE TABLE stu\_cr(  id INT PRIMARY KEY AUTO\_INCREMENT,  sid INT,  cid INT,  CONSTRAINT sc\_fk1 FOREIGN KEY (sid) REFERENCES student(id),  CONSTRAINT sc\_fk2 FOREIGN KEY (cid) REFERENCES course(id)  );  INSERT INTO stu\_cr VALUES (NULL,1,1);  INSERT INTO stu\_cr VALUES (NULL,1,2);  INSERT INTO stu\_cr VALUES (NULL,2,1);  INSERT INTO stu\_cr VALUES (NULL,2,2); |

## 4.7（掌握）多对多数据的功能实现

### 源码

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; // 学生所选择的课程集合  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 +  '}';  }  } |

ManyToManyMapper.java

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

ManyToManyMapper.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.itheima.table03.ManyToManyMapper">  <resultMap id="manyToMany" type="student">  <id column="sid" property="id"/>  <result column="sname" property="name"/>  <result column="sage" property="age"/>  <collection property="courses" ofType="course">  <id column="cid" property="id"/>  <result column="cname" property="name"/>  </collection>  </resultMap>  <select id="selectAll" resultMap="manyToMany">  SELECT sc.sid,s.name sname,s.age sage,sc.cid,c.name cname FROM student s,course c,stu\_cr sc WHERE sc.sid=s.id AND sc.cid=c.id  </select>  </mapper> |

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ManyToManyMapper接口的实现类对象  ManyToManyMapper mapper = sqlSession.getMapper(ManyToManyMapper.class);  //5.调用实现类的方法，接收结果  List<Student> students = mapper.selectAll();  //6.处理结果  for (Student student : students) {  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();  }  } |

## 4.8（掌握）多表操作的小结

