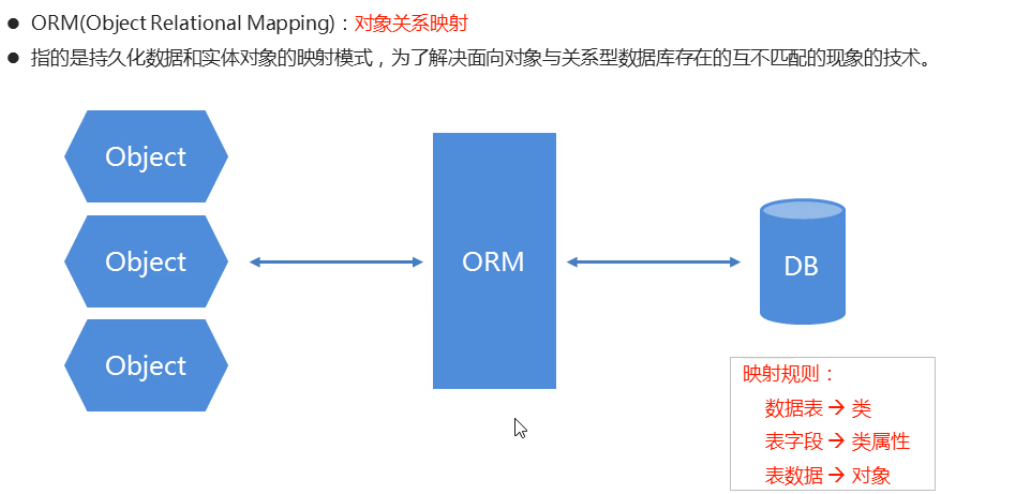
01 MyBatis基础

# 1（掌握）MyBatis快速入门

## 1.1（掌握）框架的介绍



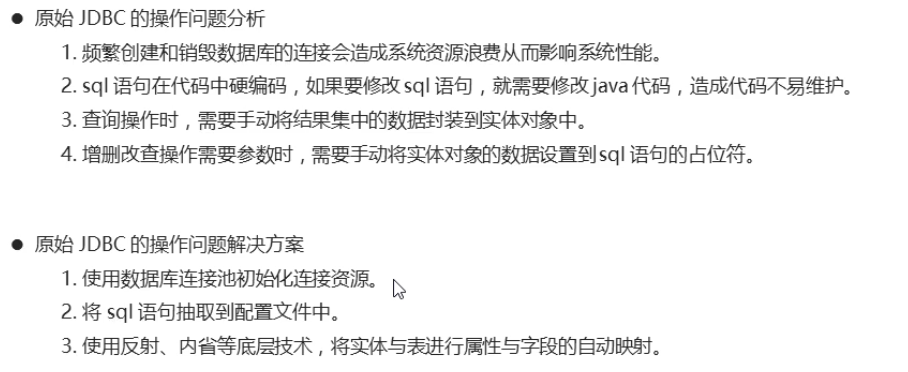
## 1.2（掌握）ORM介绍



## 1.3（掌握）MyBatis的介绍

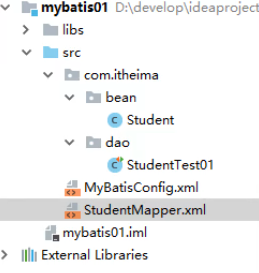


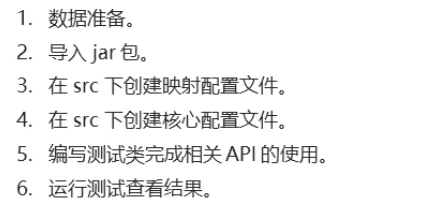






## 1.4（掌握）MyBatis入门程序





### 源码

/bean/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.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 namespace="StudentMapper">  <select id="selectAll" resultType="com.itheima.bean.Student">  SELECT \* FROM student  </select>  </mapper> |

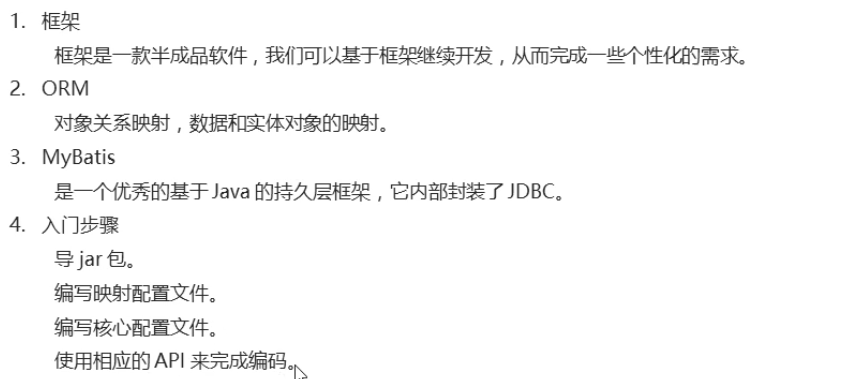
MyBatisConfig.xml

|  |
| --- |
| <?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>  <environments default="mysql">  <environment id="mysql">  <transactionManager type="JDBC"></transactionManager>  <dataSource type="POOLED">  <property name="driver" value="com.mysql.jdbc.Driver" />  <property name="url" value="jdbc:mysql://192.168.59.143:3306/db1 />  <property name="username" value="root" />  <property name="password" value="itheima" />  </dataSource>  </environment>  </environments>  <mappers>  <mapper resource="StudentMapper.xml"/>  </mappers>  </configuration> |

/dao/StudentTest01.java

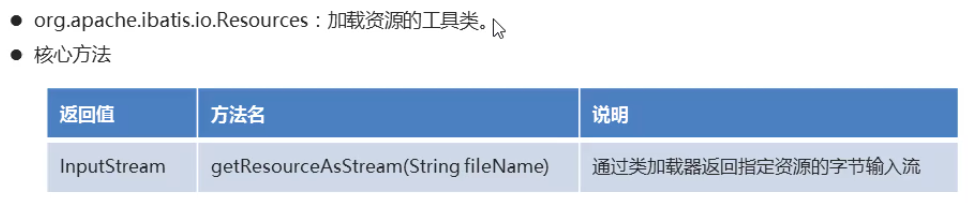
|  |
| --- |
| public class StudentTest01 {  /\*  查询全部  \*/  @Test  public void selectAll() throws Exception{  //1.加载核心配置文件  InputStream is = Resources.getResourceAsStream("MyBatisConfig.xml");  //InputStream is = StudentTest01.class.getClassLoader().getResourceAsStream("MyBatisConfig.xml");  //2.获取SqlSession工厂对象  SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(is);  //3.通过SqlSession工厂对象获取SqlSession对象  SqlSession sqlSession = sqlSessionFactory.openSession();  //4.执行映射配置文件中的sql语句，并接收结果  List<Student> list = sqlSession.selectList("StudentMapper.selectAll");  //5.处理结果  for (Student stu : list) {  System.out.println(stu);  }  //6.释放资源  sqlSession.close();  is.close();  }  } |

## 1.5（掌握）快速入门小结

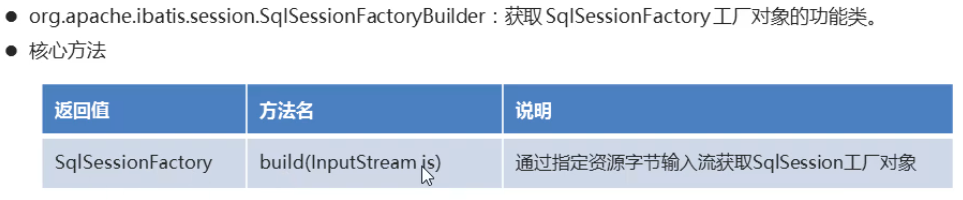


# 2（掌握）MyBatis相关API

## 2.1（掌握）Resources的介绍



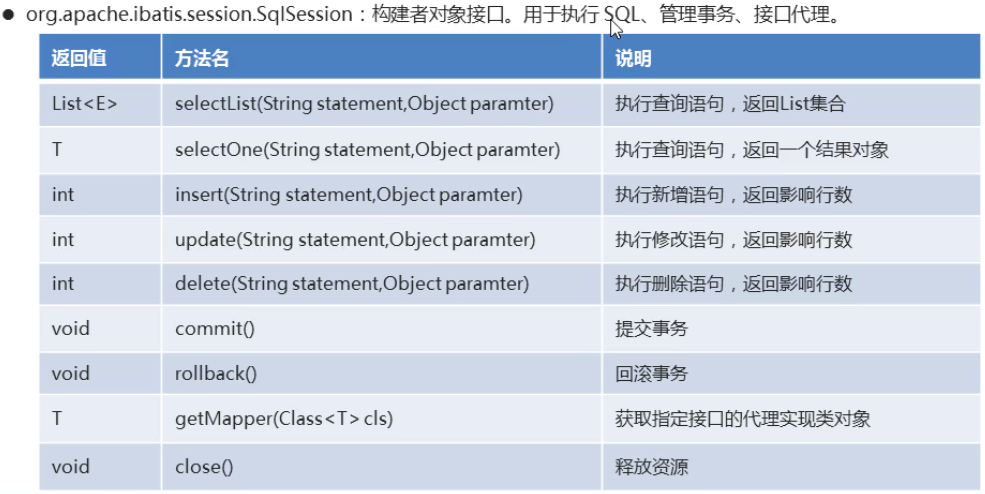
## 2.2（掌握）SqlSessionFactoryBuilder的介绍



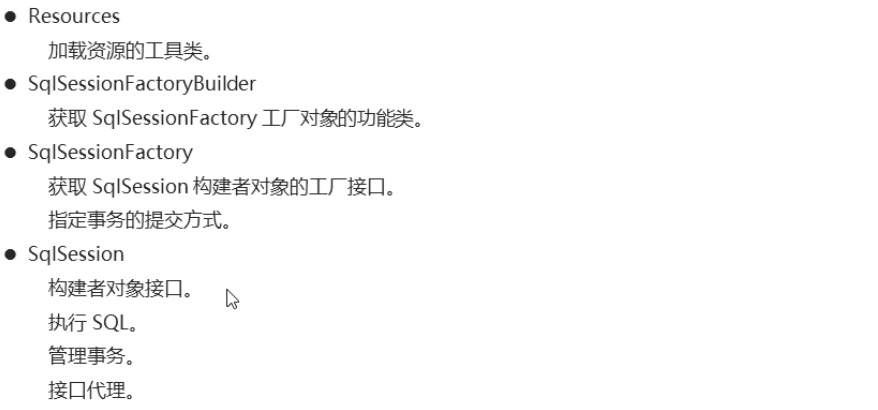
## 2.3（掌握）SqlSessionFactory的介绍



## 2.4（掌握）SqlSession的介绍



## 2.5（掌握）相关API的小结



# 3（掌握）MyBatis映射配置文件

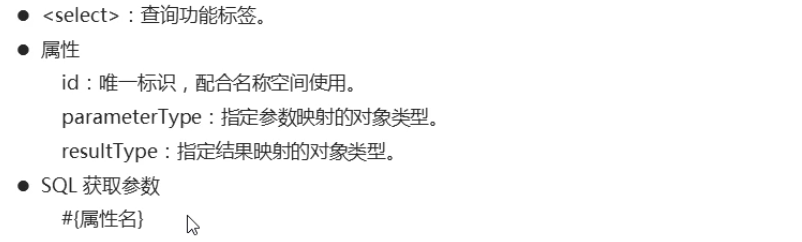
## 3.1（掌握）映射配置文件的介绍



### 源码

|  |
| --- |
| <?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="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=" com.itheima.bean.Student">  SELECT \* FROM student  </select>  </mapper> |

## 3.2（掌握）查询功能的使用



### 源码

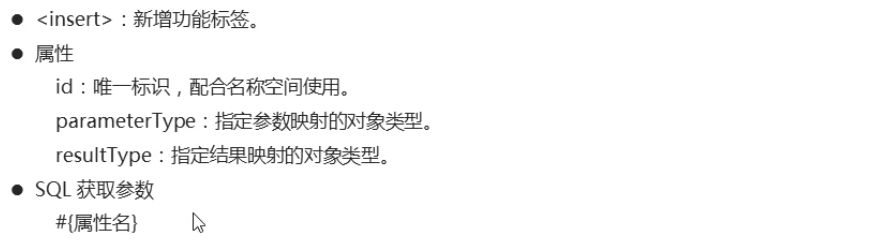
StudengMapper.xml

|  |
| --- |
| <?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="StudentMapper">  <select id="selectById" resultType=" com.itheima.bean.Student" parameterType="java.lang.Integer">  SELECT \* FROM student WHERE id = #{id}  </select>  </mapper> |

StudentTest01.java

|  |
| --- |
| public class StudentTest01 {  /\*  根据id查询  \*/  @Test  public void selectById() 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();  //4.执行映射配置文件中的sql语句，并接收结果  Student stu = sqlSession.selectOne("StudentMapper.selectById", 3);  //5.处理结果  System.out.println(stu);  //6.释放资源  sqlSession.close();  is.close();  }  } |

## 3.3（掌握）新增功能的使用



### 源码

StudengMapper.xml

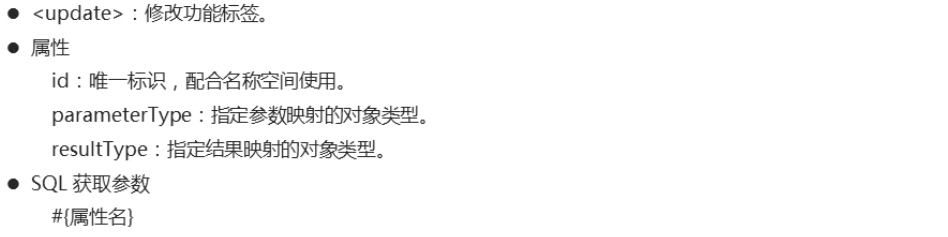
|  |
| --- |
| <?xml version="1.0" encoding="UTF-8" ?>  <!--MyBatis的DTD约束-->  <!DOCTYPE mapper  PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"  "http://mybatis.org/dtd/mybatis-3-mapper.dtd">  <!--  mapper：核心根标签  namespace属性：名称空间  -->  <mapper namespace="StudentMapper">  <insert id="insert" parameterType=" com.itheima.bean.Student">  INSERT INTO student VALUES (#{id},#{name},#{age})  </insert>  </mapper> |

StudentTest01.java

|  |
| --- |
| public class StudentTest01 {  /\*  新增功能  \*/  @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();  SqlSession sqlSession = sqlSessionFactory.openSession(true);  //4.执行映射配置文件中的sql语句，并接收结果  Student stu = new Student(5,"周七",27);  int result = sqlSession.insert("StudentMapper.insert", stu);  //5.提交事务  //sqlSession.commit();  //6.处理结果  System.out.println(result);  //7.释放资源  sqlSession.close();  is.close();  }  } |

**注意：插入的数据的id已经存在会报错的。**

## 3.4（掌握）修改功能的使用



### 源码

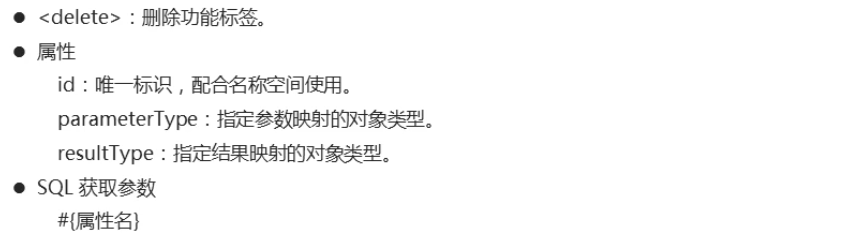
StudengMapper.xml

|  |
| --- |
| <?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="StudentMapper">  <update id="update" parameterType=" com.itheima.bean.Student">  UPDATE student SET name = #{name},age = #{age} WHERE id = #{id}  </update>  </mapper> |

StudentTest01.java

|  |
| --- |
| public class StudentTest01 {  /\*  修改功能  \*/  @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();  //SqlSession sqlSession = sqlSessionFactory.openSession(true);  //4.执行映射配置文件中的sql语句，并接收结果  Student stu = new Student(5,"周七",37);  int result = sqlSession.update("StudentMapper.update",stu);  //5.提交事务  sqlSession.commit();  //6.处理结果  System.out.println(result);  //7.释放资源  sqlSession.close();  is.close();  }  } |

## 3.5（掌握）删除功能的使用



### 源码

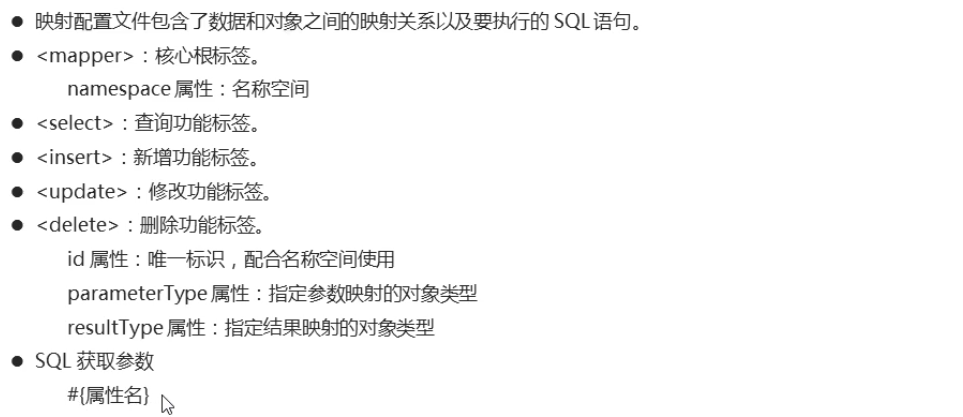
StudengMapper.xml

|  |
| --- |
| <?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="StudentMapper">  <delete id="delete" parameterType="java.lang.Integer">  DELETE FROM student WHERE id = #{id}  </delete>  </mapper> |

StudentTest01.java

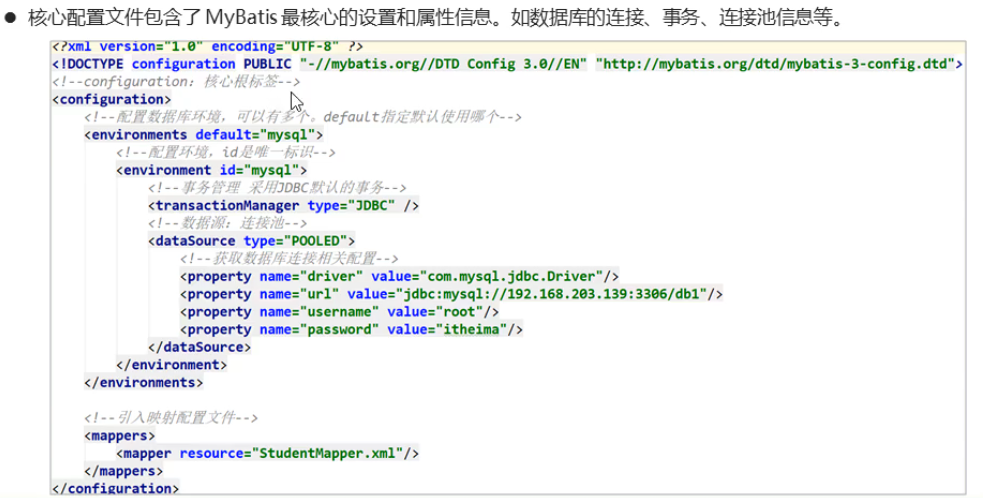
|  |
| --- |
| public class StudentTest01 {  /\*  删除功能  \*/  @Test  public void delete() throws Exception{  //1.加载核心配置文件  InputStream is = Resources.getResourceAsStream("MyBatisConfig.xml");  //2.获取SqlSession工厂对象  SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(is);  //3.通过工厂对象获取SqlSession对象  SqlSession sqlSession = sqlSessionFactory.openSession();  //SqlSession sqlSession = sqlSessionFactory.openSession(true);  //4.执行映射配置文件中的sql语句，并接收结果  int result = sqlSession.delete("StudentMapper.delete",5);  //5.提交事务  sqlSession.commit();  //6.处理结果  System.out.println(result);  //7.释放资源  sqlSession.close();  is.close();  }  } |

## 3.6（掌握）映射配置文件的小结



# 4（掌握）MyBatis核心配置文件

## 4.1（掌握）核心配置文件的介绍



### 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>  <!--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="com.mysql.jdbc.Driver" />  <property name="url" value="jdbc:mysql://192.168.59.143:3306/db1 />  <property name="username" value="root" />  <property name="password" value="itheima" />  </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> |

## 4.2（掌握）properties标签的使用

### 源码

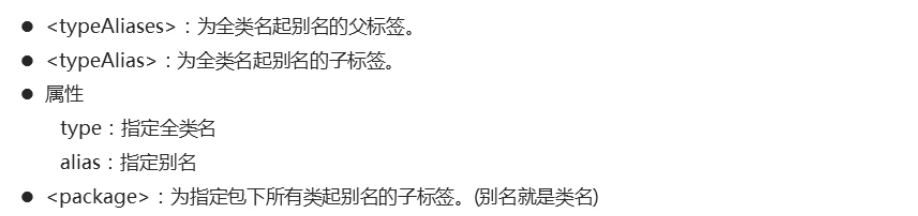
jdbc.properties

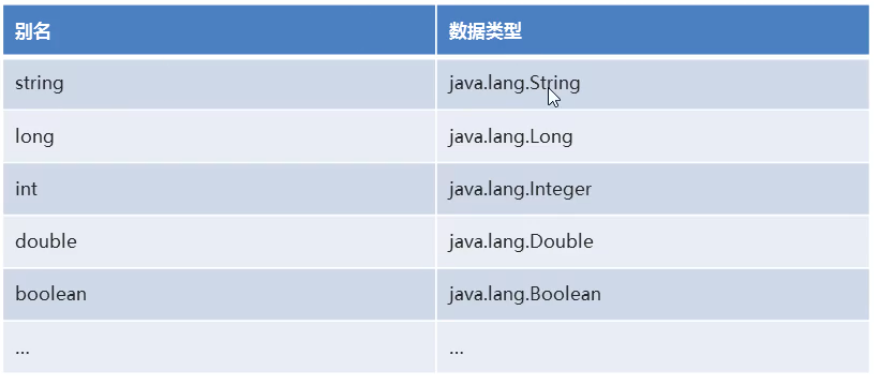
|  |
| --- |
| driver=com.mysql.jdbc.Driver  url=jdbc:mysql://192.168.59.143:3306/db1  username=root  password=itheima |

MyBastisConfig.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"/>  <!--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> |

## 4.3（掌握）起别名的使用





### 源码

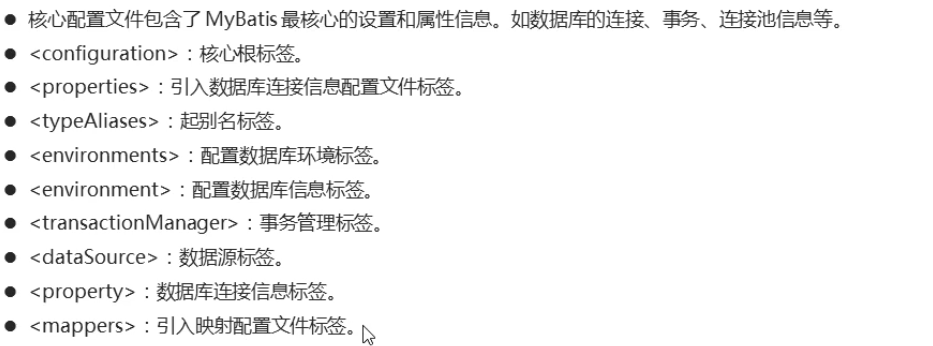
MyBastisConfig.xml

|  |
| --- |
| <?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"/>  <!--起别名-->  <typeAliases>  <typeAlias type="com.itheima.bean.Student" alias="student"/>  <!--<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 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> |

StudentMapper.xml

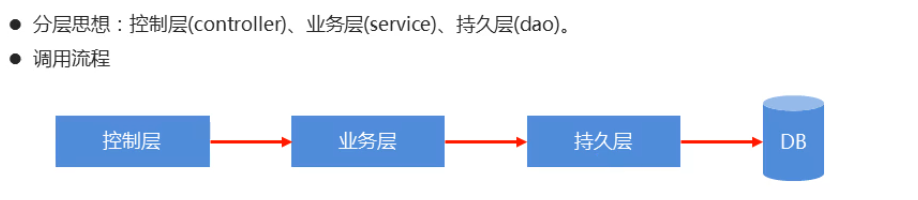
|  |
| --- |
| <?xml version="1.0" encoding="UTF-8" ?>  <!--MyBatis的DTD约束-->  <!DOCTYPE mapper  PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"  "http://mybatis.org/dtd/mybatis-3-mapper.dtd">  <!--  mapper：核心根标签  namespace属性：名称空间  -->  <mapper namespace="StudentMapper">  <!--  select：查询功能的标签  id属性：唯一标识  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> |

## 4.4（掌握）核心配置文件小结



# 5（掌握）MyBatis传统方式实现Dao层

## 5.1（掌握）环境介绍



### 源码

src

com.itheima

bean

controller

dao

mapper

service

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);  } |

StudentMapperImpl.java

|  |
| --- |
| /\*  持久层实现类  \*/  public class StudentMapperImpl implements StudentMapper {  /\*  查询全部  \*/  @Override  public List<Student> selectAll() {  return null;  }  /\*  根据id查询  \*/  @Override  public Student selectById(Integer id) {  return null;  }  /\*  新增功能  \*/  @Override  public Integer insert(Student stu) {  return 0;  }  /\*  修改功能  \*/  @Override  public Integer update(Student stu) {  return 0;  }  /\*  删除功能  \*/  @Override  public Integer delete(Integer id) {  return 0;  }  } |

StudentService.java

|  |
| --- |
| /\*  业务层接口  \*/  public interface StudentService {  //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);  } |

StudentServiceImpl.java

|  |
| --- |
| /\*  业务层实现类  \*/  public class StudentServiceImpl implements StudentService {  //创建持久层对象  private StudentMapper mapper = new StudentMapperImpl();  @Override  public List<Student> selectAll() {  return mapper.selectAll();  }  @Override  public Student selectById(Integer id) {  return mapper.selectById(id);  }  @Override  public Integer insert(Student stu) {  return mapper.insert(stu);  }  @Override  public Integer update(Student stu) {  return mapper.update(stu);  }  @Override  public Integer delete(Integer id) {  return mapper.delete(id);  }  } |

StudentController.java

|  |
| --- |
| /\*  控制层测试类  \*/  public class StudentController {  //创建业务层对象  private StudentService service = new StudentServiceImpl();  //查询全部功能测试  @Test  public void selectAll() {  List<Student> students = service.selectAll();  for (Student stu : students) {  System.out.println(stu);  }  }  //根据id查询功能测试  @Test  public void selectById() {  Student stu = service.selectById(3);  System.out.println(stu);  }  //新增功能测试  @Test  public void insert() {  Student stu = new Student(4,"赵六",26);  Integer result = service.insert(stu);  System.out.println(result);  }  //修改功能测试  @Test  public void update() {  Student stu = new Student(4,"赵六",16);  Integer result = service.update(stu);  System.out.println(result);  }  //删除功能测试  @Test  public void delete() {  Integer result = service.delete(4);  System.out.println(result);  }  } |

## 5.2（掌握）dao层功能的实现

### 源码

StudentMapperImpl.java

|  |
| --- |
| /\*  持久层实现类  \*/  public class StudentMapperImpl implements StudentMapper {  /\*  查询全部  \*/  @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.执行映射配置文件中的sql语句，并接收结果  list = sqlSession.selectList("StudentMapper.selectAll");  } catch (Exception e) {  e.printStackTrace();  } finally {  //5.释放资源  if(sqlSession != null) {  sqlSession.close();  }  if(is != null) {  try {  is.close();  } catch (IOException e) {  e.printStackTrace();  }  }  }  //6.返回结果  return list;  }  /\*  根据id查询  \*/  @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.执行映射配置文件中的sql语句，并接收结果  stu = sqlSession.selectOne("StudentMapper.selectById",id);  } catch (Exception e) {  e.printStackTrace();  } finally {  //5.释放资源  if(sqlSession != null) {  sqlSession.close();  }  if(is != null) {  try {  is.close();  } catch (IOException e) {  e.printStackTrace();  }  }  }  //6.返回结果  return stu;  }  /\*  新增功能  \*/  @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.执行映射配置文件中的sql语句，并接收结果  result = sqlSession.insert("StudentMapper.insert",stu);  } catch (Exception e) {  e.printStackTrace();  } finally {  //5.释放资源  if(sqlSession != null) {  sqlSession.close();  }  if(is != null) {  try {  is.close();  } catch (IOException e) {  e.printStackTrace();  }  }  }  //6.返回结果  return result;  }  /\*  修改功能  \*/  @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.执行映射配置文件中的sql语句，并接收结果  result = sqlSession.update("StudentMapper.update",stu);  } catch (Exception e) {  e.printStackTrace();  } finally {  //5.释放资源  if(sqlSession != null) {  sqlSession.close();  }  if(is != null) {  try {  is.close();  } catch (IOException e) {  e.printStackTrace();  }  }  }  //6.返回结果  return result;  }  /\*  删除功能  \*/  @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.执行映射配置文件中的sql语句，并接收结果  result = sqlSession.delete("StudentMapper.delete",id);  } catch (Exception e) {  e.printStackTrace();  } finally {  //5.释放资源  if(sqlSession != null) {  sqlSession.close();  }  if(is != null) {  try {  is.close();  } catch (IOException e) {  e.printStackTrace();  }  }  }  //6.返回结果  return result;  }  } |

## 5.3（掌握）LOG4J的使用



MyBatisConfig.xml

|  |
| --- |
| <!--配置LOG4J-->  <settings>  <setting name="logImpl" value="log4j"/>  </settings> |

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 |