

01、Mybatis的介绍和基本使用

01、Mybatis的介绍和基本使用

1、数据库操作框架的历程

(1) JDBC

(2) DBUtils

(3) Hibernate

(4) JdbcTemplate

2、什么是Mybatis?

3、快速搭建Mybatis项目

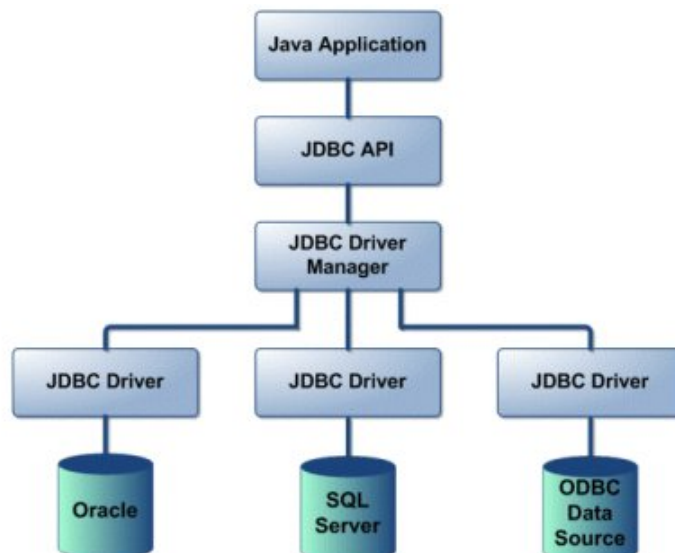
5、配置文件详解

1、数据库操作框架的历程

(1) JDBC

JDBC(Java Data Base Connection,java数据库连接)是一种用于执行SQL语句的Java API,可以为多种关系数据库提供统一访问,它由一组用Java语言编写的类和接口组成.JDBC提供了一种基准,据此可以构建更高级的工具和接口,使数据库开发人员能够编写数据库应用程序

- 优点：运行期：快捷、高效
- 缺点：编辑期：代码量大、繁琐异常处理、不支持数据库跨平台



jdbc核心api

- 1.DriverManager 连接数据库
- 2.Connection 连接数据库的抽象
- 3.Statement 执行SQL
- 4.ResultSet 数据结果集

(2) DBUtils

DBUtils是Java编程中的数据库操作实用工具，小巧简单实用。

DBUtils封装了对JDBC的操作，简化了JDBC操作，可以少写代码。

DBUtils三个核心功能介绍

- 1、QueryRunner中提供对sql语句操作的API
- 2、ResultSetHandler接口，用于定义select操作后，怎样封装结果集

- 3、DBUtils类，它就是一个工具类，定义了关闭资源与事务处理

的方法

(3)Hibernate

ORM 对象关系映射

object java对象

relational 关系型数据

mapping 映射

Hibernate 是由 Gavin King 于 2001 年创建的开放源代码的对象关系框架。它强大且高效的构建具有关系对象持久性和查询服务的 Java 应用程序。

Hibernate 将 Java 类映射到数据库表中，从 Java 数据类型中映射到 SQL 数据类型中，并把开发人员从 95% 的公共数据持续性编程工作中解放出来。

Hibernate 是传统 Java 对象和数据库服务器之间的桥梁，用来处理基于 O/R 映射机制和模式的那些对象。



Java Objects
Hibernate 优势

- Hibernate 使用 XML 文件来处理映射 Java 类别到数据库表格中，并且不用编写任何代码。
- 为在数据库中直接储存和检索 Java 对象提供简单的 APIs。
- 如果在数据库中或任何其它表格中出现变化，那么仅需要改变 XML 文件属性。
- 抽象不熟悉的 SQL 类型，并为我们提供工作中所熟悉的 Java 对象。
- Hibernate 不需要应用程序服务器来操作。
- 操控你数据库中对象复杂的关联。
- 最小化与访问数据库的智能提取策略。
- 提供简单的数据询问。

Hibernate 劣势

- hibernate 的完全封装导致无法使用数据的一些功能。
- Hibernate 的缓存问题。
- Hibernate 对于代码的耦合度太高。
- Hibernate 寻找 bug 困难。
- Hibernate 批量数据操作需要大量的内存空间而且执行过程中需要的对象太多

(4) JdbcTemplate

JdbcTemplate 针对数据查询提供了多个重载的模板方法,你可以根据需要选用不同的模板方法.如果你的查询很简单, 仅仅是传入相应 SQL 或者相关参数, 然后取得一个单一的结果, 那么你可以选择如下一组便利的模板方法。

优点：运行期：高效、内嵌 Spring 框架中、支持基于 AOP 的声明式事务

缺点：必须于 Spring 框架结合在一起使用、不支持数据库跨平台、默认没有缓存

2、什么是 Mybatis?

MyBatis 是一款优秀的持久层框架/半自动的ORM，它支持自定义 SQL、存储过程以及高级映射。MyBatis 免除了几乎所有的 JDBC 代码以及设置参数和获取结果集的工作。MyBatis 可以通过简单的 XML 或注解来配置和映射原始类型、接口和 Java POJO（Plain Old Java Objects，普通老式 Java 对象）为数据库中的记录。

优点：

- 1、与JDBC相比，减少了50%的代码量
 - 2、最简单的持久化框架，简单易学
 - 3、SQL代码从程序代码中彻底分离出来，可以重用
 - 4、提供XML标签，支持编写动态SQL
 - 5、提供映射标签，支持对象与数据库的ORM字段关系映射
- 支持缓存、连接池、数据库移植....

缺点：

- 1、SQL语句编写工作量大，熟练度要高
- 2、数据库移植性比较差，如果需要切换数据库的话，SQL语句会有很大的差异

3、快速搭建Mybatis项目

- 1、创建普通的maven项目
- 2、导入相关的依赖

pom.xml

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0"
3   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
5   <modelVersion>4.0.0</modelVersion>
6
7   <groupId>cn.tulingxueyuan</groupId>
8   <artifactId>mybatis_helloworld</artifactId>
9   <version>1.0-SNAPSHOT</version>
10
11   <dependencies>
12     <dependency>
13       <groupId>org.mybatis</groupId>
```

```

14 <artifactId>mybatis</artifactId>
15 <version>3.5.4</version>
16 </dependency>
17 <dependency>
18 <groupId>mysql</groupId>
19 <artifactId>mysql-connector-java</artifactId>
20 <version>5.1.47</version>
21 </dependency>
22 </dependencies>
23
24 </project>

```

驱动请按照数据库版本进行对应<https://dev.mysql.com/doc/relnotes/connector-j/5.1/en/>

Connector/J version	JDBC version	MySQL Server version	JRE Required	JDK Required for Compilation
5.1	3.0, 4.0, 4.1, 4.2	5.6 ¹ , 5.7 ¹ , 8.0 ¹	JRE 5 or higher ¹	JDK 5.0 AND JDK 8.0 or higher ^{2, 3}
8.0	4.2	5.6, 5.7, 8.0	JRE 8 or higher	JDK 8.0 or higher ²

3、创建对应的数据表

4、创建与表对应的实体类对象

emp.java

```

1 package cn.tulingxueyuan.pojo;
2
3 /**
4  * @Author 徐庶 QQ:1092002729
5  * @Slogan 致敬大师，致敬未来的你
6  */
7 public class Emp {
8     private Integer id;
9     private String username;
10
11     public Integer getId() {
12         return id;
13     }
14
15     public void setId(Integer id) {
16         this.id = id;
17     }
18
19     public String getUsername() {

```

```

20  return username;
21  }
22
23  public void setUsername(String username) {
24  this.username = username;
25  }
26
27  @Override
28  public String toString() {
29  return "Emp{" +
30  "id=" + id +
31  ", username='" + username + '\'' +
32  '}';
33  }
34  }
35

```

5、创建对应的Mapper接口

EmpMapper.java

```

1  package cn.tulingxueyuan.mapper;
2
3  import cn.tulingxueyuan.pojo.Emp;
4  import org.apache.ibatis.annotations.Select;
5
6  /**
7   * @Author 徐庶 QQ:1092002729
8   * @Slogan 致敬大师，致敬未来的你
9   */
10 public interface EmpMapper {
11     // 根据id查询Emp实体
12     // @Select("select * from emp where id=#{id}")
13     Emp selectEmp(Integer id);
14
15     // 插入
16     Integer insertEmp(Emp emp);
17
18     // 更新
19     Integer updateEmp(Emp emp);
20
21     // 删除
22     Integer deleteEmp(Integer id);

```

```
23 }  
24
```

6、编写配置文件

mybatis-config.xml

```
1 <?xml version="1.0" encoding="UTF-8" ?>  
2 <!DOCTYPE configuration  
3 PUBLIC "-//mybatis.org//DTD Config 3.0//EN"  
4 "http://mybatis.org/dtd/mybatis-3-config.dtd">  
5 <configuration>  
6   <environments default="development">  
7     <environment id="development">  
8       <transactionManager type="JDBC"/>  
9       <dataSource type="POOLED">  
10        <property name="driver" value="com.mysql.jdbc.Driver"/>  
11        <property name="url" value="jdbc:mysql://localhost:3306/mybatis"/>  
12        <property name="username" value="root"/>  
13        <property name="password" value="123456"/>  
14      </dataSource>  
15    </environment>  
16  </environments>  
17  <mappers>  
18    <!--<mapper resource="EmpMapper.xml"/-->  
19    <mapper class="cn.tulingxueyuan.mapper.EmpMapper"></mapper>  
20  </mappers>  
21 </configuration>
```

EmpMapper.xml

```
1 <?xml version="1.0" encoding="UTF-8" ?>  
2 <!DOCTYPE mapper  
3 PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"  
4 "http://mybatis.org/dtd/mybatis-3-mapper.dtd">  
5 <mapper namespace="cn.tulingxueyuan.mapper.EmpMapper">  
6  
7   <!--根据id查询Emp实体-->  
8   <select id="selectEmp" resultType="cn.tulingxueyuan.pojo.Emp">  
9     select * from Emp where id = #{id}  
10  </select>  
11  
12  <insert id="insertEmp">  
13    INSERT INTO
```

```

14  `mybatis`.`emp` ( `username`)
15  VALUES ({username});
16  </insert>
17
18  <update id="updateEmp">
19  UPDATE EMP
20  SET username=#{username}
21  WHERE id=#{id}
22  </update>
23
24  <delete id="deleteEmp">
25  DELETE FROM emp
26  WHERE id=#{id}
27  </delete>
28
29  </mapper>

```

7、编写测试类

MyTest.java

```

1
2  /**
3   * @Author 徐庶 QQ:1092002729
4   * @Slogan 致敬大师，致敬未来的你
5   *
6   * MyBatis 搭建步骤:
7   * 1.添加pom依赖 （mybatis的核心jar包和数据库版本对应版本的驱动jar包）
8   * 2.新建数据库和表
9   * 3.添加mybatis全局配置文件 （可以从官网中复制）
10  * 4.修改mybatis全局配置文件中的 数据源配置信息
11  * 5.添加数据库表对应的POJO对象（相当于我们以前的实体类）
12  * 6.添加对应的PojoMapper.xml（里面就维护所有的sql）
13  * 修改namespace：如果是StatementId没有特殊的要求
14  * 如果是接口绑定的方式必须等于接口的完整限定名
15  * 修改对应的id(唯一)、resultType 对应返回的类型如果是POJO需要制定完整限定名
16  * 7.修改mybatis全局配置文件：修改Mapper
17  */
18  public class MybatisTest {
19
20      SqlSessionFactory sqlSessionFactory;
21      @Before
22      public void before(){

```



```

23 // 从 XML 中构建 SqlSessionFactory
24 String resource = "mybatis.xml";
25 InputStream inputStream = null;
26 try {
27     inputStream = Resources.getResourceAsStream(resource);
28 } catch (IOException e) {
29     e.printStackTrace();
30 }
31 sqlSessionFactory = new SqlSessionFactoryBuilder().build(inputStream);
32 }
33
34 /**
35  * 基于StatementId的方式去执行SQL
36  * <mapper resource="EmpMapper.xml"/>
37  * @throws IOException
38  */
39 @Test
40 public void test01() {
41     try (SqlSession session = sqlSessionFactory.openSession()) {
42         Emp emp = (Emp) session.selectOne("cn.tulingxueyuan.pojo.EmpMapper.selectEmp", 1);
43         System.out.println(emp);
44     }
45 }
46
47 /**
48  * 基于接口绑定的方式
49  * 1.新建数据访问层的接口： POJOMapper
50  * 2.添加mapper中对应的操作的方法
51  * 1.方法名要和mapper中对应的操作的节点的id要一致
52  * 2.返回类型要和mapper中对应的操作的节点的resultType要一致
53  * 3.mapper中对应的操作的节点的参数必须要在方法的参数中声明
54  * 3.Mapper.xml 中的namespace必须要和接口的完整限定名要一致
55  * 4.修改mybatis全局配置文件中的mappers,采用接口绑定的方式:
56  * <mapper class="cn.tulingxueyuan.mapper.EmpMapper"></mapper>
57  * 5.一定要将mapper.xml和接口放在同一级目录中,只需要在resources新建和接口同样结构的文件夹就行了,生成就会合并在一起
58  *
59  * @throws IOException
60  */
61 @Test

```

```

62 public void test02(){
63     try (SqlSession session = sqlSessionFactory.openSession()) {
64         EmpMapper mapper = session.getMapper(EmpMapper.class);
65         Emp emp = mapper.selectEmp(1);
66         System.out.println(emp);
67     }
68 }
69
70
71 /**
72  * 基于注解的方式
73  * 1.在接口方法上面写上对应的注解
74  * @Select("select * from emp where id=#{id}")
75  * 注意:
76  * 注解可以和xml共用，但是不能同时存在方法对应的xml的id
77  *
78  */
79 @Test
80 public void test03(){
81     try (SqlSession session = sqlSessionFactory.openSession()) {
82         EmpMapper mapper = session.getMapper(EmpMapper.class);
83         Emp emp = mapper.selectEmp(1);
84         System.out.println(emp);
85     }
86 }
87
88 }
89

```

3、增删改查的基本操作

EmpDao.java

```

1 package cn.tulingxueyuan.dao;
2
3 import cn.tulingxueyuan.bean.Emp;
4
5 public interface EmpDao {
6
7     public Emp findEmpByEmpno(Integer empno);
8
9     public int updateEmp(Emp emp);
10

```

```

11 public int deleteEmp(Integer empno);
12
13 public int insertEmp(Emp emp);
14
15 }

```

EmpDao.xml

```

1 <?xml version="1.0" encoding="UTF-8" ?>
2 <!DOCTYPE mapper
3 PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
4 "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
5 <!--namespace:编写接口的全类名，就是告诉要实现该配置文件是哪个接口的具体实现-->
6 <mapper namespace="cn.tulingxueyuan.dao.EmpDao">
7 <!--
8 select:表示这个操作是一个查询操作
9 id表示的是要匹配的方法的名称
10 resultType:表示返回值的类型，查询操作必须要包含返回值的类型
11 #{属性名}: 表示要传递的参数名称
12 -->
13 <select id="findEmpByEmpno" resultType="cn.tulingxueyuan.bean.Emp">
14 select * from emp where empno = #{empno}
15 </select>
16 <!--增删改查操作不需要返回值，增删改返回的是影响的行数，mybatis会自动做判断-->
17 <insert id="insertEmp">
18 insert into emp(empno,ename) values(#{empno},#{ename})
19 </insert>
20 <update id="updateEmp">
21 update emp set ename=#{ename} where empno = #{empno}
22 </update>
23 <delete id="deleteEmp">
24 delete from emp where empno = #{empno}
25 </delete>
26 </mapper>

```

MyTest.java

```

1 package cn.tulingxueyuan.test;
2
3 import cn.tulingxueyuan.bean.Emp;
4 import cn.tulingxueyuan.dao.EmpDao;
5 import org.apache.ibatis.io.Resources;
6 import org.apache.ibatis.session.SqlSession;

```

```
7 import org.apache.ibatis.session.SqlSessionFactory;
8 import org.apache.ibatis.session.SqlSessionFactoryBuilder;
9 import org.junit.Before;
10 import org.junit.Test;
11
12 import java.io.IOException;
13 import java.io.InputStream;
14
15 public class MyTest {
16     SqlSessionFactory sqlSessionFactory = null;
17     @Before
18     public void init(){
19         // 根据全局配置文件创建出SqlSessionFactory
20         // SqlSessionFactory:负责创建SqlSession对象的工厂
21         // SqlSession:表示跟数据库建议的一次会话
22         String resource = "mybatis-config.xml";
23         InputStream inputStream = null;
24         try {
25             inputStream = Resources.getResourceAsStream(resource);
26             sqlSessionFactory= new SqlSessionFactoryBuilder().build(inputStream);
27         } catch (IOException e) {
28             e.printStackTrace();
29         }
30     }
31     @Test
32     public void test01() {
33
34         // 获取数据库的会话
35         SqlSession sqlSession = sqlSessionFactory.openSession();
36         Emp empByEmpno = null;
37         try {
38             // 获取要调用的接口类
39             EmpDao mapper = sqlSession.getMapper(EmpDao.class);
40             // 调用方法开始执行
41             empByEmpno = mapper.findEmpByEmpno(7369);
42         } catch (Exception e) {
43             e.printStackTrace();
44         } finally {
45             sqlSession.close();
46         }
```

```

47  System.out.println(empByEmpno);
48  }
49
50  @Test
51  public void test02(){
52      SqlSession sqlSession = sqlSessionFactory.openSession();
53      EmpDao mapper = sqlSession.getMapper(EmpDao.class);
54      int zhangsan = mapper.insertEmp(new Emp(1111, "zhangsan"));
55      System.out.println(zhangsan);
56      sqlSession.commit();
57      sqlSession.close();
58  }
59
60  @Test
61  public void test03(){
62      SqlSession sqlSession = sqlSessionFactory.openSession();
63      EmpDao mapper = sqlSession.getMapper(EmpDao.class);
64      int zhangsan = mapper.updateEmp(new Emp(1111, "lisi"));
65      System.out.println(zhangsan);
66      sqlSession.commit();
67      sqlSession.close();
68  }
69
70  @Test
71  public void test04(){
72      SqlSession sqlSession = sqlSessionFactory.openSession();
73      EmpDao mapper = sqlSession.getMapper(EmpDao.class);
74      int zhangsan = mapper.deleteEmp(1111);
75      System.out.println(zhangsan);
76      sqlSession.commit();
77      sqlSession.close();
78  }
79  }

```

EmpDaoAnnotation.java

```

1  package cn.tulingxueyuan.dao;
2
3  import cn.tulingxueyuan.bean.Emp;
4  import org.apache.ibatis.annotations.Delete;
5  import org.apache.ibatis.annotations.Insert;

```

```
6 import org.apache.ibatis.annotations.Select;
7 import org.apache.ibatis.annotations.Update;
8
9 public interface EmpDaoAnnotation {
10
11     @Select("select * from emp where id= #{id}")
12     public Emp findEmpByEmpno(Integer empno);
13
14     @Update("update emp set ename=#{ename} where id= #{id}")
15     public int updateEmp(Emp emp);
16
17     @Delete("delete from emp where id= #{id}")
18     public int deleteEmp(Integer empno);
19
20     @Insert("insert into emp(id,user_name) values(#{id},#{username})")
21     public int insertEmp(Emp emp);
22
23 }
```