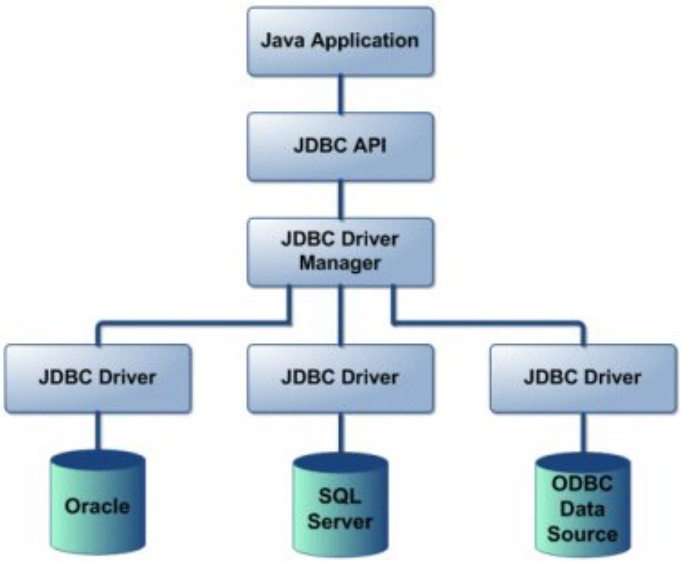
01、Mybatis的介绍和基本使用

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.Statment 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。

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

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xs
    <modelVersion>4.0.0</modelVersion>

    <groupId>cn.tulingxueyuan</groupId>
```

```
    <artifactId>mybatis_helloworld</artifactId>
    <version>1.0-SNAPSHOT</version>

    <dependencies>
        <dependency>
            <groupId>org.mybatis</groupId>
            <artifactId>mybatis</artifactId>
            <version>3.5.4</version>
        </dependency>
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>5.1.47</version>
        </dependency>
    </dependencies>

</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

```
package cn.tulingxueyuan.pojo;

/***
 * @Author 徐庶   QQ:1092002729
 * @Slogan 致敬大师，致敬未来的你
 */
public class Emp {
    private Integer id;
    private String username;

    public Integer getId() {
        return id;
    }
```

```
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" ?>
```

```
<!DOCTYPE mapper
        PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
        "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="cn.tulingxueyuan.mapper.EmpMapper">

    <!--根据id查询Emp实体-->
    <select id="selectEmp" resultType="cn.tulingxueyuan.pojo.Emp">
        select * from Emp where id = #{id}
    </select>

    <insert id="insertEmp">
        INSERT INTO
        `mybatis`.`emp` ( `username`)
        VALUES (#{username});
    </insert>

    <update id="updateEmp">
        UPDATE EMP
        SET username=#{username}
        WHERE id=#{id}
    </update>

    <delete id="deleteEmp">
        DELETE FROM emp
        WHERE id=#{id}
    </delete>

</mapper>
```

7、编写测试类

MyTest.java

```

/***
 * @Author 徐庶   QQ:1092002729
 * @Slogan 致敬大师，致敬未来的你
 *
 * MyBatis 搭建步骤：
 * 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）
 * 2.新建数据库和表
```

```
 * 3.添加mybatis全局配置文件 （可以从官网中复制）
 * 4.修改mybatis全局配置文件中的 数据源配置信息
 * 5.添加数据库表对应的POJO对象（相当于我们以前的实体类）
 * 6.添加对应的PojoMapper.xml （里面就维护所有的sql）
 *      修改namespace:  如果是StatementId没有特殊的要求
 *                      如果是接口绑定的方式必须等于接口的完整限定名
 *      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
 * 7.修改mybatis全局配置文件：修改Mapper
 */
public class MybatisTest {

    SqlSessionFactory sqlSessionFactory;
    @Before
    public void before(){
        // 从 XML 中构建 SqlSessionFactory
        String resource = "mybatis.xml";
        InputStream inputStream = null;
        try {
            inputStream = Resources.getResourceAsStream(resource);
        } catch (IOException e) {
            e.printStackTrace();
        }
        sqlSessionFactory = new SqlSessionFactoryBuilder().build(inputStream);
    }

    /**
     * 基于StatementId的方式去执行SQL
     *      <mapper resource="EmpMapper.xml"/>
     * @throws IOException
     */
    @Test
    public void test01() {
        try (SqlSession session = sqlSessionFactory.openSession()) {
            Emp emp = (Emp) session.selectOne("cn.tulingxueyuan.pojo.EmpMapper.selectEmp
            System.out.println(emp);
        }
    }

    /**
     * 基于接口绑定的方式
```

```
     *  1.新建数据访问层的接口：   POJOMapper
     *  2.添加mapper中对应的操作的方法
     *      1.方法名要和mapper中对应的操作的节点的id要一致
     *      2.返回类型要和mapper中对应的操作的节点的resultType要一致
     *      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
     *  3.Mapper.xml 中的namespace必须要和接口的完整限定名要一致
     *  4.修改mybatis全局配置文件中的mappers,采用接口绑定的方式:
     *        <mapper class="cn.tulingxueyuan.mapper.EmpMapper"></mapper>
     *  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
     *
     * @throws IOException
     */
    @Test
    public void test02(){
        try (SqlSession session = sqlSessionFactory.openSession()) {
            EmpMapper mapper = session.getMapper(EmpMapper.class);
            Emp emp = mapper.selectEmp(1);
            System.out.println(emp);
        }
    }


    /**
     * 基于注解的方式
     * 1.在接口方法上面写上对应的注解
     *@Select("select * from emp where id=#{id}")
     * 注意:
     *      注解可以和xml共用，  但是不能同时存在方法对应的xml的id
     *
     */
    @Test
    public void test03(){
        try (SqlSession session = sqlSessionFactory.openSession()) {
            EmpMapper mapper = session.getMapper(EmpMapper.class);
            Emp emp = mapper.selectEmp(1);
            System.out.println(emp);
        }
    }

}
```

```
89
```

3、增删改查的基本操作

EmpDao.java

```java
package cn.tulingxueyuan.dao;

import cn.tulingxueyuan.bean.Emp;

public interface EmpDao {

    public Emp findEmpByEmpno(Integer empno);

    public int updateEmp(Emp emp);

    public int deleteEmp(Integer empno);

    public int insertEmp(Emp emp);

}
```

EmpDao.xml

```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">
<!--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-->
<mapper namespace="cn.tulingxueyuan.dao.EmpDao">
    <!--
    select:表示这个操作是一个查询操作
    id表示的是要匹配的方法的名称
    resultType:表示返回值的类型，查询操作必须要包含返回值的类型
    #{属性名}：表示要传递的参数的名称
    -->
    <select id="findEmpByEmpno" resultType="cn.tulingxueyuan.bean.Emp">
        select * from emp where empno = #{empno}
    </select>
    <!--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-->
```

```
    <insert id="insertEmp">
        insert into emp(empno,ename) values(#{empno},#{ename})
    </insert>
    <update id="updateEmp">
        update emp set ename=#{ename} where empno = #{empno}
    </update>
    <delete id="deleteEmp">
        delete from emp where empno = #{empno}
    </delete>
</mapper>
```

MyTest.java

```
package cn.tulingxueyuan.test;

import cn.tulingxueyuan.bean.Emp;
import cn.tulingxueyuan.dao.EmpDao;
import org.apache.ibatis.io.Resources;
import org.apache.ibatis.session.SqlSession;
import org.apache.ibatis.session.SqlSessionFactory;
import org.apache.ibatis.session.SqlSessionFactoryBuilder;
import org.junit.Before;
import org.junit.Test;

import java.io.IOException;
import java.io.InputStream;

public class MyTest {
    SqlSessionFactory sqlSessionFactory = null;
    @Before
    public void init(){
        // 根据全局配置文件创建出SqlSessionFactory
        // SqlSessionFactory:负责创建SqlSession对象的工厂
        // SqlSession:表示跟数据库建议的一次会话
        String resource = "mybatis-config.xml";
        InputStream inputStream = null;
        try {
            inputStream = Resources.getResourceAsStream(resource);
            sqlSessionFactory= new SqlSessionFactoryBuilder().build(inputStream);
        } catch (IOException e) {
```

```
            e.printStackTrace();
        }
    }
    @Test
    public void test01() {

        // 获取数据库的会话
        SqlSession sqlSession = sqlSessionFactory.openSession();
        Emp empByEmpno = null;
        try {
            // 获取要调用的接口类
            EmpDao mapper = sqlSession.getMapper(EmpDao.class);
            // 调用方法开始执行
            empByEmpno = mapper.findEmpByEmpno(7369);
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            sqlSession.close();
        }
        System.out.println(empByEmpno);
    }

    @Test
    public void test02(){
        SqlSession sqlSession = sqlSessionFactory.openSession();
        EmpDao mapper = sqlSession.getMapper(EmpDao.class);
        int zhangsan = mapper.insertEmp(new Emp(1111, "zhangsan"));
        System.out.println(zhangsan);
        sqlSession.commit();
        sqlSession.close();
    }

    @Test
    public void test03(){
        SqlSession sqlSession = sqlSessionFactory.openSession();
        EmpDao mapper = sqlSession.getMapper(EmpDao.class);
        int zhangsan = mapper.updateEmp(new Emp(1111, "lisi"));
        System.out.println(zhangsan);
        sqlSession.commit();
        sqlSession.close();
```

```
    }

    @Test
    public void test04(){
        SqlSession sqlSession = sqlSessionFactory.openSession();
        EmpDao mapper = sqlSession.getMapper(EmpDao.class);
        int zhangsan = mapper.deleteEmp(1111);
        System.out.println(zhangsan);
        sqlSession.commit();
        sqlSession.close();
    }
}
```

EmpDaoAnnotation.java

```
package cn.tulingxueyuan.dao;

import cn.tulingxueyuan.bean.Emp;
import org.apache.ibatis.annotations.Delete;
import org.apache.ibatis.annotations.Insert;
import org.apache.ibatis.annotations.Select;
import org.apache.ibatis.annotations.Update;

public interface EmpDaoAnnotation {

    @Select("select * from emp where id= #{id}")
    public Emp findEmpByEmpno(Integer empno);

    @Update("update emp set ename=#{ename} where id= #{id}")
    public int updateEmp(Emp emp);

    @Delete("delete from emp where id= #{id}")
    public int deleteEmp(Integer empno);

    @Insert("insert into emp(id,user_name) values(#{id},#{username})")
    public int insertEmp(Emp emp);

}
```