1. MyBatis
2. 原始JDBC操作的分析
3. 原始JDBC开发存在的问题：
4. 数据库连接创建，释放频繁造成系统资源浪费从而影响系统性能。
5. sql语句在代码中硬编码，造成代码不易维护，实际应用SQL的改动需要改变java代码
6. 查询操作时，需要手动将结果集中的数据封装到实体中；插入操作时，需要手动将实体的数据设置到sql语句的占位符的位置。
7. 解决方案
8. 使用数据库连接池初始化连接资源，解决资源浪费问题。
9. 将SQL语句抽取到xml配置文件中。
10. 使用反射，内省等技术，自动将实体与表进行属性与字段的自动映射。
11. Mybatis简介

Mybatis是一个优秀的基于java的持久层框架，它内部封装了jdbc，使开发者只需要关注sql语句本身，而不需要花费精力去处理加载驱动，创建连接，创建statement等繁杂的过程。

Mybatis通过xml或注解的方式将要执行的各种statement配置起来，并通过java对象和statement中sql的动态参数进行映射生成最终执行的sql语句。Mybatis框架执行完sql语句后，将结果映射为java对象返回，这是采用了ORM（Object Relation Mapping）对象关系映射思想来解决实体对象和数据库的映射问题。

Mybatis对jdbc进行了封装，屏蔽了jdbc api底层访问的细节，使得我们不用与jdbc api打交道，即可完成对数据库的持久化操作。

1. Mybatis的开发步骤
2. 添加Mybatis的坐标

在pom.xml中添加Mybatis的坐标，数据库驱动，测试，日志。

1. 创建数据库表user

为该数据表添加三个字段，id，username，password。

1. 编写实体类User

根据数据表，创建一个实体类User，属性有id，username，password。

1. 编写映射文件UserMapper.xml

命名规则由实体对象+Mapper组成，位置在实体类的resources目录下。这个文件中写的是SQL语句。

<?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>  
 <select id="findAll" resultType="com.mybatis.domain.User">  
 select \* from user  
 </select>  
</mapper>

1. 编写核心文件SqlMapConfig.xml

该文件用于配置Mybatis。

<?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>  
  
<!-- 数据源环境-->  
 <environments default="developement">  
 <environment id="developement">  
 <transactionManager type="JDBC"></transactionManager>  
 <dataSource type="POOLED">  
 <property name="driver" value="com.mysql.cj.jdbc.Driver"/>  
 <property name="url" value="jdbc:mysql://localhost:3306/test"/>  
 <property name="username" value="root"/>  
 <property name="password" value="123"/>  
 </dataSource>  
 </environment>  
 </environments>  
</configuration>

1. 编写测试类

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.Test;  
  
import java.io.IOException;  
import java.io.InputStream;  
import java.util.List;  
  
public class MyBatisTest {  
  
 @Test  
 public void test1() throws IOException {  
 //获得核心配置文件  
 InputStream resourceAsStream = Resources.*getResourceAsStream*("sqlMapConfig.xml");  
 //获得session工厂对象  
 SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(resourceAsStream);  
 //获得session会话对象  
 SqlSession sqlSession = sqlSessionFactory.openSession();  
 //执行操作， 参数是 namespace + id  
 List<Object> userList = sqlSession.selectList("userMapper.findAll");  
 System.*out*.println(userList);  
 //释放资源  
 sqlSession.close();  
 }  
}

1. MyBatis的映射文件

映射文件即为上面的UserMapper.xml配置文件，下面来详解该配置文件。

1. 约束头

给配置文件中的约束头既能够起约束作用，也会有提示作用。

<?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">

1. 根标签mapper

<mapper namespace=”userMapper”>

根标签为mapper，有一个属性是namespace命名空间，与下面的sql语句的id一起组成调用语句的标识。

1. 操作子标签

在根标签mapper内部有操作的子标签，如同SQL语句的增删改查操作一样，子标签使用的就是insert(增)，delete(删)，update（改），select（查）。属性id是该SQL语句的id名称，与根标签的命名空间一起组成查询的标识，如userMapper.findAll。属性resultType是和操作的数据表的数据进行映射的实体类的全类名，操作数据表后能够将数据封装到该实体类中。

再下面就是一个SQL语句。

如，查询操作

<mapper namespace=”userMapper”>

<select id=”findAll” resultType=”com.mybatis.domain.User”>

select \* from User

</mapper>

1. 查询数据操作

要从数据库中查询数据，并封装到对象实体中。因此，在映射配置文件UserMapper.xml中，我们需要在select子标签中设置resultType属性，值为实体对象的全类名。

<mapper namespace =”userMapper”>

<select id=”findAll” resultType=”domain.User”>

select \* from user

</select>

</mapper>

1. 插入数据操作

不同于查询数据操作，插入数据操作需要从实体对象中抽取属性的值到sql语句中，进行插入操作。那么，如何从实体对象中抽取属性值呢？

在映射文件UserMapper.xml中设置insert子标签，并设置属性parameterType，值为要抽取的实体对象的全类名。

下面的sql语句如何写呢？如何将实体对象中的每个属性写到SQL语句中？

通过#(属性名)的方式，代表着抽取该对象的属性名。

<mapper namespace =”userMapper”>

<insert id=”save” parameterType=”domain.User”>

insert into user values(#{id}, #{username}, #{password})

</insert>

</mapper>

插入操作使用的API是sqlSession.insert(“命名空间.id”，实体对象);

插入操作设计是数据库数据的变化，所以要使用sqlSession对象显式地提交事务，如sqlSession.commit()，数据才会被提交到数据库中，一般更新数据库的操作都需要显式地提交事务。

1. 修改数据操作

修改操作和插入操作类似，在映射文件UserMapper.xml中都需要抽取实体类对象的数据，因此都需要parameterType属性。同样，在测试类中使用时最后都需要提交事务。

<insert id=”save” parameterType=”domain.User”>

<update id=”update” parameterType=”domain.User”>

update user set username=#{username}, password=#{password} where id=#{id}

</update>

</insert>

修改操作使用的API是sqlSession.update(“命名空间.id”, 实体对象)

1. 删除数据操作

删除操作，使用delete子标签，属性parameterType值为要删除的值的类型全类名，如int类型的全类名为java.lang.Integer。其他的使用方式是一样的。

如，根据id值来删除：

<mapper namespace=”userMapper”>

<delete id=”delete” parameterType=”java.lang,Integer”>

delete from user where id=#{id}

</mapper>

调用时，使用delete函数，7是要删除的int类型值：

sqlSession.delete(“userMapper.delete”, 7);

1. MyBatis核心配置文件
2. 约束头

每个MyBatis核心配置文件上都有一个约束头。

<?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">

1. configuration根标签

跟在约束头后面的就是configuration根标签。

<configuration></configuration>

1. environments标签

environments标签是对数据库环境的配置，在该标签内还可以有多个environment子标签，即支持多个环境的配置。

1. default属性

在environments标签的default属性中，可以指定默认的环境名称。

如，指定id名为 development的环境：

<environments default=”development”>

1. environment 标签

每个环境的id属性是自己的标识，指定当前环境的名称：

<environment id=”development”>

1. transactionManager标签

该标签为事务管理， type属性可以指定事务管理的类型。

有两种类型，一种是JDBC，这个配置就是直接使用了JDBC的提交和回滚设置，依赖于从数据源得到的连接来管理事务的作用域。

第二种是MANAGED，这个配置的意思是让容器来管理事务的整个过程，他不会提交或回滚一个连接。默认情况下它会关闭连接，通常我们需要将closeConnection属性设置为false来阻止它。

如，指定类型为JDBC:

<transactionManager type=”JDBC”>

1. dataSource标签

该标签用于设置当前的数据源，type属性可以指定数据源的类型。

数据源的类型共有三种：

1. UNPOOLED

这个数据源不是连接池类型，每次被请求都要打开和关闭连接，效率低。

1. POOLED

数据源为连接池类型。

1. JDNI

这个数据源的实现是为了能在EJB服务器这类容器中使用，容器可以集中火灾外部配置数据源，然后放置一个JNDI上下文的引用。

如连接池类型：

<dataSource type=”POOLED”>

1. property标签

property标签是dataSource标签的子标签，用于配置数据源的基本信息，如驱动，url，用户名，密码：

<property name=”driver” value=”${jdbc.driver}”/>

<property name=”url” value=”${jdbc.url}”/>

<property name=”username” value=”${jdbc.username}”/>

<property name=”password” value=”${jdbc.password}”/>

一个environments标签体内部的整体结构如下所示：

<environments default=”development”>

<environment id=”development”>

<transactionManager type=”JDBC”/>

<dataSource type=”POOLED”>

<property name=”driver” value=”${jdbc.driver}”/>

<property name=”url” value=”${jdbc.url}”/>

<property name=”username” value=”${jdbc.username}”/>

<property name=”password” value=”${jdbc.password}”/>

</dataSource>

</environment>

</environments>

1. mappers标签

我们在测试MyBatis的测试类中，只是调用了MyBatis核心配置文件，并没有调用映射文件，因为在核心配置文件的mappers标签就是为了加载映射文件。加载方式又如下几种：

1. 使用相对类路径的资源引用

即映射文件的全包名。

如：

<mapper resource=”org/mybatis/userMapper.xml”/>

1. 使用完全限定资源定位符（URL）

这种就是直接指定映射文件所在的位置。

如：

<mapper url=”<file://var/mappers/userMapper.xml>” />

1. 使用映射器接口实现类的完全限定类名

使用注解时会接触到，class属性为全类名。

如：

<mapper class=”org.mybatis.builer.userMapper”/>

1. 将包内的映射器接口实现全部注册为映射器

这与上面的不同在于，可以将包内的所有映射器都注册为映射器，而上面只能使用一个映射器。

如：

<package name=”org.mybatis”/>

1. properties标签

将配置信息抽取到一个properties文件中，是一个非常高效的方法。在MyBatista的核心配置文件中，数据库环境的配置就需要使用外部的properties文件。在使用前设置properties标签，resource属性为properties文件的位置。

如：

<properties resource=”jdbc.properties”></properties>

然后，在使用properties文件中的键值对时，通过EL表达式的形式取出键值对的值。

如：

<property name=”driver” value=”${jdbc.driver}”/>

1. typeAliases标签

此标签可以为一个类型设置一个别名，使开发更加高效。一些简单类型MyBatis已经为它们设置了别名，如：

数据类型 别名

String string

Long long

Integer int

Double double

Boolean Boolean

那没有预定义别名的如何自定义别名呢？

通过typeAliases标签来定义，可以在其中用typeAlias子标签定义多个别名type属性是类名，alias属性是别名：

<typeAliases>

<typeAlias type=”com.mybatis.domain.User” alias=”user”></typeAlias>

</typeAliases>

这样，我们在使用该类时就可以使用别名了。

1. MyBatis相应的API
2. SqlSessionFactory构建器SqlSessionFactoryBuilder

我们要通过SqlSessionFactoryBuilder这个工厂构建器来创建SqlSessionFactory。这是通过工厂构建器的build函数实现的：

SqlSessionFactory build(InputStream inputStream);

首先通过Resources工具类将MyBatis的核心文件生成输入流的格式，再创建一个SqlSessionFactoryBuilder，通过这个构建器的build函数生成一个SqlSessionFactory工厂。

Resources工具类在org.apache.ibatis.io 包中，它能帮助从类路径下，文件系统或一个web url中加载资源文件。

String resources = “org/mybatis/builder/mybatis-confg.xml”;

InputStream inputStream = Resources.getResourceAsStream(resource);

SqlSessionFactoryBuilder builder = new SqlSessionFactoryBuilder();

SqlSessionFactory factory = builder.build(inputStream);

1. SqlSessionFactory工厂

如名字的意思，SqlSessionFactory是SqlSession的工厂对象，能够创建SqlSession对象，创建的方式有两种：

1. openSession()

SqlSessionFactory对象调用函数openSession()，没有任何参数，会创建一个SqlSession对话实例，但同时也会默认开启一个事务，且这个事务不会自动提交，需要我们手动提交这个事务后，更新操作的数据才会持久化到数据库中。

1. openSession(boolean autoCommit)

有一个参数autoCommit，可设置是否自动提交事务，若为true，则无需手动提交事务。

1. SqlSession会话对象

SqlSession会话对象可以用于执行SQL语句，提交或回滚事务，获取映射器实例。

1. 执行SQL语句

<T> T selectOne(String statement, Object parameter);

<E> List<E> selectList(String statement, Object parameter);

int insert(String statement, Object parameter);

int update(String statement, Object parameter);

int delete(String statement, Object parameter);

1. 操作事务

void commit();

void rollback();