Maven项目管理和构建工具

项目生命周期管理

Jar包的依赖管理

本地仓库

中央仓库

私服

Groupid

Artificid

Version

使用maven，解压

配置环境变量 MAVEN\_HOME， %MAVEN\_HOME%\bin

Mvn -v

eclipse中配置maven插件

Settings.xml

配置本地仓库路径

配置镜像

如果需要进行发布，配置server

Maven project

使用arctype，目录结构不全

跳过arctype，注意选择jar或者war

Maven插件

编译插件

Tomcat插件

tomcat7:run

tomcat7:deploy

tomcat7:redeploy

SSM

Spring 粘合剂，核心功能，创建对象

Spring mvc 控制层

Mybatis 数据库操作，dao层

# 一 mybatis的基本介绍

ORM框架

ORM 对象关系映射模型

数据库的表和相关实体类进行映射处理

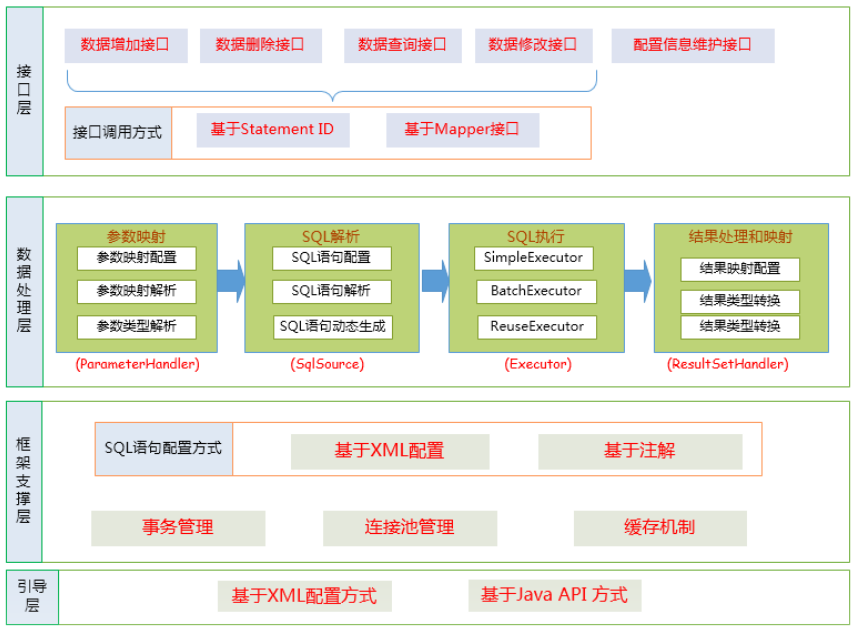
## 1 概述

mybatis是一个封装自jdbc的持久层框架，它和hibernate都属于ORM框架

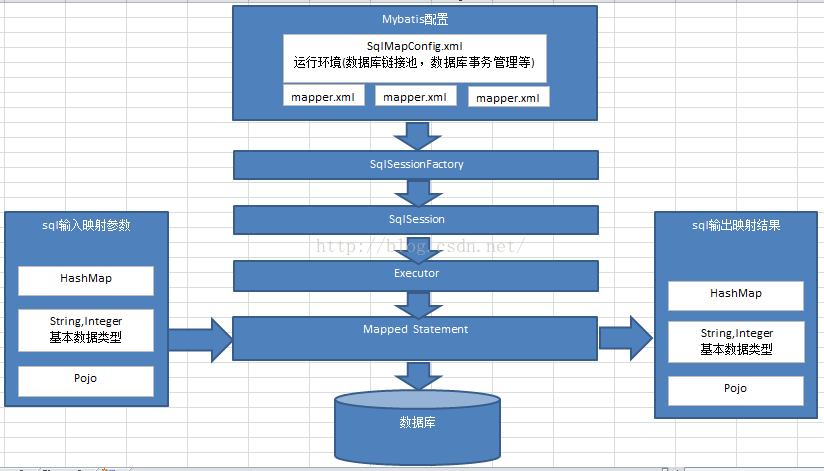
mybatis让程序员只关注sql本身，而不需要去关注如连接的创建、statement的创建等操作。

mybatis可以使用简单的 XML 或注解来配置和映射信息，将接口和 Java 的 POJO(Plain Old Java Objects,普通的 Java对象)映射成数据库中的记录

## 2 功能框架



## 3 mybatis框架的体系结构



1）mybatis配置

SqlMapConfig.xml，此文件作为mybatis的全局配置文件，配置了mybatis的运行环境等信息。

mapper.xml文件即sql映射文件，文件中配置了操作数据库的sql语句。此文件需要在SqlMapConfig.xml中加载。

2）通过mybatis环境等配置信息构造SqlSessionFactory即会话工厂

3）由会话工厂创建sqlSession即会话，操作数据库需要通过sqlSession进行。

4）mybatis底层自定义了Executor执行器接口操作数据库，Executor接口有两个实现，一个是基本执行器、一个是缓存执行器。

5）Mapped Statement也是mybatis一个底层封装对象，它包装了mybatis配置信息及sql映射信息等。mapper.xml文件中一个sql对应一个Mapped Statement对象，sql的id即是Mapped statement的id。

6）Mapped Statement对sql执行输入参数进行定义，包括HashMap、基本类型、实体类对象，Executor通过Mapped Statement在执行sql前将输入的java对象映射至sql中，输入参数映射就是jdbc编程中对preparedStatement设置参数。

MappedStatement对sql执行输出结果进行定义，包括HashMap、基本类型、实体类对象，Executor通过Mapped Statement在执行sql后将输出结果映射至java对象中，输出结果映射过程相当于jdbc编程中对结果的解析处理过程。

## 4 简单执行流程

1）读取配置文件(通常是XML配置文件中)得到，SqlSessionFactory对象.

2）通过SqlSessionFactory对象，获取SqlSession对象

3）通过SqlSession对象，完成对数据的增删改查和事务提交等.

4）用完之后关闭session 。

# 二 mybatis的基本用法

## 1 入门

使用mybatis需要全局配置文件和映射文件；

在Dao层，加载配置文件，获取SqlSession对象，进行操作

1)需要导入的jar

mybatis-3.4.5.jar

连接数据库的驱动的jar包

如果需要打印日志，需要导入：log4j-1.2.17.jar

还需要log4j.properties文件（src目录下）

<!-- 引入mybatis的jar包 -->

<dependency>

<groupId>org.mybatis</groupId>

<artifactId>mybatis</artifactId>

<version>3.4.5</version>

</dependency>

<!-- mysql -->

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<version>5.1.39</version>

</dependency>

<!-- 日志处理 -->

<dependency>

<groupId>log4j</groupId>

<artifactId>log4j</artifactId>

<version>1.2.17</version>

</dependency>

2)mybatis的核心配置文件

<?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>

<!-- 引入数据库配置的属性文件 -->

<properties resource=*"db.properties"*></properties>

<environments default=*"mysqlEnv"*>

<!-- 配置数据库的连接 -->

<environment id=*"mysqlEnv"*>

<!-- 事务 -->

<transactionManager type=*"JDBC"* />

<!-- 配置数据源 -->

<dataSource type=*"POOLED"*>

<property name=*"driver"* value=*"${driver}"* />

<property name=*"url"* value=*"${url}"* />

<property name=*"username"* value=*"${username}"* />

<property name=*"password"* value=*"${password}"* />

</dataSource>

</environment>

</environments>

<!-- 加载映射文件 -->

<mappers>

<mapper resource=*"com/rr/first/PersonMapper.xml"* />

</mappers>

</configuration>

3)针对实体类的映射文件

<?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">

<!-- 命名空间 -->

<mapper namespace=*"personNameSpace"*>

<!-- 表示实体与数据库表的映射关系

type表示实体类的全名 包名+类名

id 标识

-->

<resultMap type=*"com.rr.first.Person"* id=*"personMap"*>

<!-- 主键 -->

<id property=*"id"* column=*"id"*/>

<!-- 其他字段 -->

<result property=*"name"* column=*"name"*/>

</resultMap>

<!-- sql语句相关配置 -->

<insert id=*"add"*>

insert into person(name) values("hahaha");

</insert>

<!-- parameterType 参数类型 ，

可以使用int、double、string等基础类型

自定义类的类型

map 等类型

-->

<insert id=*"add1"* parameterType=*"com.rr.first.Person"*>

<!-- #表示获取参数的值 ， 参数类型是自定义对象时，参数名相当于对象中的get方法-->

insert into person(name) values(#{name});

</insert>

</mapper>

4)加载配置文件

**private** **static** SqlSessionFactory *ssf* = **null**;

**static**{

**try** {

//加载主配置文件，生成Reader对象

Reader reader = Resources.*getResourceAsReader*("mybatis.xml");

*ssf* = **new** SqlSessionFactoryBuilder().build(reader);

} **catch** (IOException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

5)执行操作

**public** **void** add(){

//获得SqlSession对象

SqlSession session = *ssf*.openSession();

//通过 命名空间.对应sql的id 找到需要执行的sql语句

session.insert("personNameSpace.add");

session.commit();

session.close();

}

## 2 增删改查

<!-- 给类定义别名 -->

<typeAliases>

<typeAlias type=*"com.rr.pojo.Person"* alias=*"person"*/>

<typeAlias type=*"com.rr.pojo.Student"* alias=*"student"*/>

</typeAliases>

映射文件：

<?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">

<!-- 命名空间 ,习惯上写为 包名+类名的形式-->

<mapper namespace=*"com.rr.pojo.Person"*>

<!-- 表示实体与数据库表的映射关系

type表示实体类的全名 包名+类名

type 可以使用别名

id 标识

-->

<resultMap type=*"person"* id=*"personMap"*>

<!-- 主键 -->

<id property=*"id"* column=*"id"*/>

<!-- 其他字段 -->

<result property=*"name"* column=*"name"*/>

</resultMap>

<!-- parameterType 参数类型 ，可以使用别名

可以使用int、double、string等基础类型

自定义类的类型

map 等类型

-->

<insert id=*"add1"* parameterType=*"person"*>

<!-- #表示获取参数的值 ， 参数类型是自定义对象时，参数名相当于对象中的get方法-->

insert into person(name) values(#{name});

</insert>

<!-- 表示查询的sql -->

<!-- 如果使用以前的方式，返回值List<Person>

查询结果返回集合时，mybatis的映射文件中，只需要使用集合中对象的类型

resultType 查询返回结果的类型

-->

<select id=*"findAll"* resultType=*"person"*>

select \* from person;

</select>

<!-- 查询单个数据

参数为基本类型，通过#{} 获取参数时，参数名字任意

基本类型还可以使用java中类型表示

-->

<select id=*"findById2"* parameterType=*"int"* resultType=*"person"*>

SELECT ID,NAME FROM PERSON WHERE ID=#{id};

<!-- SELECT ID,NAME FROM PERSON WHERE ID=#{aaa}; -->

</select>

<select id=*"findById"* parameterType=*"java.lang.Integer"* resultType=*"person"*>

SELECT ID,NAME FROM PERSON WHERE ID=#{id};

</select>

<!-- 删除 -->

<delete id=*"deleteById"* parameterType=*"int"*>

delete from person where id=#{id};

</delete>

<!-- 更新 -->

<update id=*"update"* parameterType=*"person"*>

update person set name=#{name} where id=#{id};

</update>

<!-- resultMap的值对应 <resultMap>配置的id -->

<select id=*"findByPage"* parameterType=*"map"* resultMap=*"personMap"*>

<!-- start/rows是map中的key值 -->

select \* from person limit #{start},#{rows};

</select>

</mapper>

resultType和resultMap：

进行select映射的时候，返回类型可以用resultType，也可以用resultMap，resultType直接表示返回类型，而resultMap则是对外部resultMap的引用，resultType和resultMap不能同时存在。  
 在MyBatis进行查询映射时，其实查询出来的每一个属性都是放在一个对应的Map里面的，其中键是属性名，值则是其对应的值。  
1）当提供的返回类型属性是resultType时，MyBatis会将Map里面的键值对取出赋给resultType所指定的对象对应的属性。所以其实MyBatis的每一个查询映射的返回类型都是ResultMap，只是当提供的返回类型属性是resultType的时候，MyBatis对自动的给把对应的值赋给resultType所指定对象的属性。  
2）当提供的返回类型是resultMap时，因为Map不能很好表示领域模型，就需要自己再进一步的把它转化为对应的对象，这常常在复杂查询中很有作用。

#{}与${}区别：

#{}在预编译时，相当于一个参数占位符“?”，用来补全预编译语句，能够防止注入式攻击。

${}表示内容的原样输出，相当于单纯的内容替换，拼接完成后才会对SQL进行编译、执行。

关于特殊符号的处理：

xml中，<、 <=、& 属于特殊符号

比如在映射配置中，select \* from person where id<4会报错

解决方案：

1）<![CDATA[]>，xml中，CDATA括起来的内容，会进行原样输出

比如<![CDATA[select \* from person where id<4]]>

select \* from person where id<![CDATA[<=]]>4

2）使用实体符号

select \* from person where id &lt;= 4

# 三 mapper代理开发

Dao实现类，是传统的操作数据库的方式

Mybatis提供了另外的方式，即映射代理，执行sql语句操作数据库，相当于原来的dao

## 1 规范

1)mapper接口的全限定名要和mapper映射文件的namespace值一致。

2)mapper方法名称接口的要和mapper映射文件的statement的id一致。

3)mapper接口的方法参数类型要和mapper映射文件的statement的parameterType的值一致，而且它的参数是一个。

4)mapper接口的方法返回值类型要和mapper映射文件的statement的resultType的值一致

## 2 用法

1）映射接口

IPersonMapper

//相当于Dao层

**public** **interface** IPersonMapper {

**public** **void** add(Person p);

**public** List<Person> findAll();

**public** Person findById(**int** id);

}

2）映射文件

<?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">

<!-- 命名空间 ,必须与mapper代理的接口的全路径名一致-->

<mapper namespace=*"com.rr.mapper.IPersonMapper"*>

<resultMap type=*"person"* id=*"personMap"*>

<!-- 主键 -->

<id property=*"id"* column=*"id"*/>

<!-- 其他字段 -->

<result property=*"name"* column=*"name"*/>

</resultMap>

<!-- mapper方法名称接口的要和mapper映射文件的statement的id一致

-->

<insert id=*"add"* parameterType=*"person"*>

<!-- #表示获取参数的值 ， 参数类型是自定义对象时，参数名相当于对象中的get方法-->

insert into person(name) values(#{name});

</insert>

<!-- 表示查询的sql -->

<!-- 如果使用以前的方式，返回值List<Person>

查询结果返回集合时，mybatis的映射文件中，只需要使用集合中对象的类型

resultType 查询返回结果的类型

-->

<select id=*"findAll"* resultType=*"person"*>

select \* from person;

</select>

<select id=*"findById"* parameterType=*"java.lang.Integer"* resultType=*"person"*>

SELECT ID,NAME FROM PERSON WHERE ID=#{id};

</select>

</mapper>

3）调用

SqlSession session = *ssf*.openSession();

//获取mapper对象

IPersonMapper mapper = session.getMapper(IPersonMapper.**class**);

Person p = mapper.findById(2);

System.***out***.println(p.getName());

List<Person> list = mapper.findAll();

System.***out***.println(list.get(1).getName());

session.close();

# 四 动态sql

如果我们不知道sql查询的条件具体有哪些，sql语句该怎么写？

Select \* from user where 1=1 and name=’’ and age>10;

Mybatis中提供了一些关键词用来进行动态sql操作

1. if 语句 (简单的条件判断)  
2. choose (when,otherwize)  
3. trim (对包含的内容加上 prefix,或者 suffix 等，前缀，后缀)  
4. where (主要是用来简化sql语句中where条件判断的，能智能的处理 and 、or ,不必担心多余导致语法错误)  
5. set (主要用于更新时)  
6. foreach (在实现 mybatis in 语句查询时特别有用)

<?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">

<!-- 命名空间 ,习惯上写为 包名+类名的形式-->

<mapper namespace=*"com.rr.pojo.Student"*>

<resultMap type=*"student"* id=*"stuMap"*>

<!-- 主键 -->

<id property=*"num"* column=*"num"*/>

<!-- 其他字段 -->

<result property=*"name"* column=*"name"*/>

<result property=*"age"* column=*"age"*/>

<result property=*"score1"* column=*"score1"*/>

<result property=*"score2"* column=*"score2"*/>

</resultMap>

<!-- insert into student(num,name,age) values(?,?,?) -->

<insert id=*"add"* parameterType=*"student"*>

insert into student(

<!-- trim prefix/suffix设置前后缀，

prefixOverrides/suffixOverrides去除前后的内容

-->

<trim suffixOverrides=*","*>

<!-- if 进行条件判断 -->

<if test=*"num!=null"*>

num,

</if>

<if test=*"name!=null"*>

name,

</if>

<if test=*"age!=null"*>

age,

</if>

<if test=*"score1!=null"*>score1,</if>

<if test=*"score2!=null"*>score2</if>

</trim>

) values(

<trim suffixOverrides=*","*>

<if test=*"num!=null"*>

#{num},

</if>

<if test=*"name!=null"*>

#{name},

</if>

<if test=*"age!=null"*>

#{age},

</if>

<if test=*"score1!=null"*>#{score1},</if>

<if test=*"score2!=null"*>#{score2}</if>

</trim>

);

</insert>

<!-- select \* from student where name=? and/or age=? -->

<select id=*"findAll"* parameterType=*"student"* resultType=*"student"*>

select \* from student

<!-- 相当于sql语句中的where ，通过<where>会忽略不需要的or/and-->

<where>

<if test=*"num!=null"*>

num=#{num}

</if>

<if test=*"name!=null"*>

or name=#{name}

</if>

<if test=*"age!=null"*>

or age=#{age}

</if>

<if test=*"score1!=null"*>

or score1=#{score1}

</if>

<if test=*"score2!=null"*>

or score2=#{score2}

</if>

</where>

</select>

<select id=*"findAll1"* parameterType=*"map"* resultType=*"student"*>

select \* from student

<!-- 相当于sql语句中的where ，通过<where>会忽略不需要的or/and-->

<where>

<if test=*"num!=null"*>

num=#{num}

</if>

<if test=*"name!=null"*>

or name=#{name}

</if>

<if test=*"age!=null"*>

or age=#{age}

</if>

<if test=*"score1!=null"*>

or score1=#{score1}

</if>

<if test=*"score2!=null"*>

or score2=#{score2}

</if>

</where>

</select>

<!-- update student set name=?,age=? where num=? -->

<update id=*"update"* parameterType=*"student"*>

update student

<!-- <set>相当于sql语句中的set，会忽略最后的"," -->

<set>

<if test=*"name!=null"*>

name=#{name},

</if>

<if test=*"age!=null"*>

age=#{age},

</if>

<if test=*"score1!=null"*>

score1=#{score1},

</if>

<if test=*"score2!=null"*>

score2=#{score2}

</if>

</set>

where num=#{num};

</update>

<!--

对于mybatis，传过来的参数本质上都封装为map

如果是基础类型，key值随便写

如果是实体类对象，key值是对象的属性（本质上是get方法中get后面的内容）

如果是数组，key值是array

如果是List，key值是list

如果是map，key/value是程序设置的值

如果传过来的参数是数组，映射文件中使用map类型,key值是array -->

<!-- delete from student where num in (？,？,？) -->

<delete id=*"deletes"* parameterType=*"map"*>

delete from student where num in

<!-- foreach 循环

collection 要遍历的数组或者列表，值表示map中的key值

item 遍历的元素的变量名

open 表示以什么符号开始，本例中给出的"(" 相当于sql语句中的 "in (" 位置的"("

close 表示以什么符号结束

separator 设置分隔符

index 设置索引变量

-->

<foreach collection=*"array"* item=*"n"* open=*"("* close=*")"* separator=*","*>

#{n}

</foreach>

</delete>

<delete id=*"deleteList"* parameterType=*"map"*>

delete from student where num in

<foreach collection=*"list"* item=*"n"* open=*"("* close=*")"* separator=*","*>

#{n}

</foreach>

</delete>

<!-- 也可以使用list类型 -->

<delete id=*"deleteList1"* parameterType=*"list"*>

delete from student where num in

<foreach collection=*"list"* item=*"n"* open=*"("* close=*")"* separator=*","*>

#{n}

</foreach>

</delete>

</mapper>

附录：

log4j配置文件的说明：

#代表root节点整个工程下所有输出日志的地方配置的输出的级别和输出的位置。

log4j.rootLogger=debug,stdout

#代表配置控制台的类

log4j.appender.stdout=org.apache.log4j.ConsoleAppender

#log4j.appender.stdout.Target=System.err

#代表配置布局

log4j.appender.stdout.layout=org.apache.log4j.SimpleLayout

#代表自定义输出内容的格式。其中ConversionPattern设置输出格式的参数解释如下

%d 产生日志的时间

%t 产生日志所处的线程名称

%-5p 输出日志的级别，将占5位字符，不足5位用空格填补，-指的是在右边补齐，没有则在左边

%c 输出日志的包以及类的全名

%m 附加的信息

%n 换行

log4j.appender.Console.layout.ConversionPattern=%d [%t] %-5p [%c] - %m%n

#代表特定包下的特定级别。log4j把日志分为ALL、TRACE&（跟踪）、DEBUG（调试）、INFO（信息）、WARNING（警告）、ERROR（错误）、FITAL（致命）、OFF等几个级别，级别依次升高。整个配置中，级别高的Level会屏蔽级别低的信息。

log4j.logger.com.ibatis=DEBUG

log4j.logger.com.ibatis.common.jdbc.SimpleDataSource=DEBUG

log4j.logger.com.ibatis.common.jdbc.ScriptRunner=DEBUG

log4j.logger.com.ibatis.sqlmap.engine.impl.SqlMapClientDelegate=DEBUG

log4j.logger.java.sql.Connection=DEBUG

log4j.logger.java.sql.Statement=DEBUG

log4j.logger.java.sql.PreparedStatement=DEBUG