## 一、代码生成插件：Mybatis Generator

**MybatisGenerator详解**：

<http://git.oschina.net/free/Mybatis_Utils/tree/master/MybatisGeneator/MybatisGeneator.md>

插件代码在包：tk.mybatis.mapper.generator，一共包含两个类：

MapperCommentGenerator：该类用于生成数据库备注字段的注释，以及实体类字段的注解。

MapperPlugin：插件的实现类，该类默认使用上面这个注释生成器，插件屏蔽了一般的CRUD方法（保留了Example），插件可以生成实体的@Table注解。

两种方式自动生成代码：编码方式+maven插件运行方式

1. 使用Maven执行Mybatis Generator

1）配置pom.xml文件

2）配置MBG配置文件：generatorConfig.xml

3）配置数据库连接等：config.properties

4）在pom.xml所在目录的命令行执行下面命令：

mvn mybatis-generator:generate

5）或者在eclipse，右键pom.xml->run as ->maven build

在goals中输入：mybatis-generator:generate

2、pom.xml配置

参见文件：[pom.xml](mapper-generator/pom.xml)

3、generatorConfig.xml配置

参见文件：[generatorConfig.xml](mapper-generator/generatorConfig.xml)

4、config.properties配置

参见文件：[config.properties](mapper-generator/config.properties)

## 二、Mybatis分页插件

资料：

<http://git.oschina.net/free/Mybatis_PageHelper/blob/master/wikis/HowToUse.markdown>

### 1、引入分页插件

通过maven引入分页插件，具体的依赖如下：

<dependency>

<groupId>com.github.pagehelper</groupId>

<artifactId>pagehelper</artifactId>

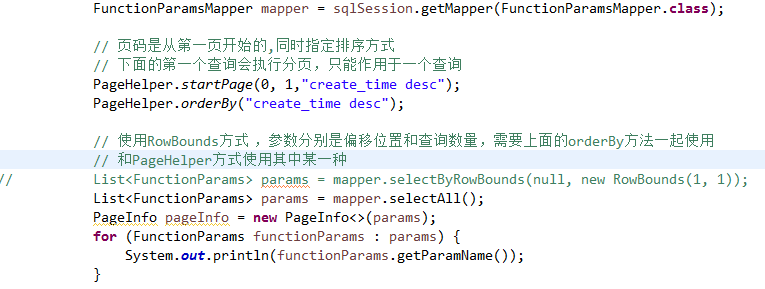
<version>4.1.6</version>

</dependency>

### 2、在mybatis-config.xml配置分页拦截器插件



### 3、调用



## 三、Mybatis返回值

Mybatis提供多种元素来匹配不同类型的语句，如：select、insert、update、delete.每种类型的元素需要提供一个id属性，id的值与mapper接口的对应方法名称一致。

Mybatis中的返回值可以通过resultType或resultMap指定。

**Remark**：

1) 当使用resultType时，指定的是JavaBean的全类型，默认的映射规则是数据库查询出来的列名与JavaBean的属性名称一致(不区分大小写)；

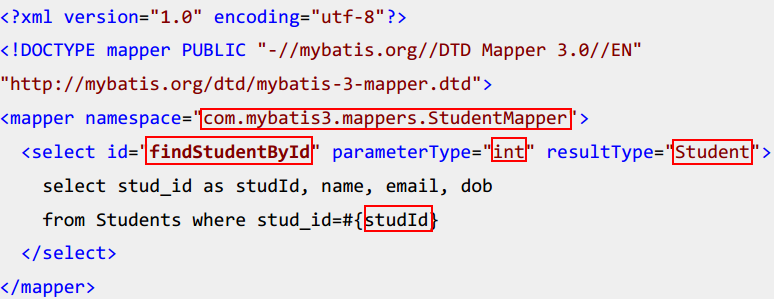
由于大多数数据库设置不区分大小写，因此下划线方式的命名很常见，如：user\_name,user\_email，而在Java代码中，一般使用驼峰式命名：userName,userEmail，因此Mybatis提供了全局属性：mapUnderscoreToCamelCase用于支持将以下划线方式命名的数据库列映射到Java对象的驼峰式命名属性中，**默认该配置为false**。

2) 使用resultMap时，可以指定数据库的column与实体类的property的映射关系，这个比较灵活，可以随便指定，如指定数据库的A列映射到实体的B属性。

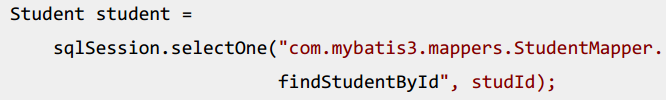
**3.1 Insert语句**

## 四、XML配置SQL映射器

### 4.1 配置文件与映射器接口



上面定义了一个查询映射器，可以通过下面的方式调用该映射器中定义的sql语句：



调用时通过配置文件所在的包名namespace+语句的id可以调用映射器中的SQL语句，这样做容易出错，需要检查映射器配置文件中的定义，以保证输入参数类型和结果返回值是有效的。

mybatis可以通过mapper接口来调用映射器语句的方法，定义一个与配置文件完全对应的接口，接口名与映射器文件名一致，接口所在的包与映射器文件的namespace一样；接口中的方法名称是映射器的id，方法参数类型是parameterType对应的值，方法返回值是returnType对应的值。

* 在mapper.xml中，使namespace等于mapper接口的地址（完全限定名）
* mapper.java接口中的方法名和mapper.xml中statement的id一致
* mapper.java接口中方法的输入参数类型和mapper.xml中statement的parameterType指定的类型一致
* mapper.java接口中方法返回值类型和mapper.xml中statement的resultType指定的类型一致

**使用接口进行调用是类型安全的(明确的知道输入参数的类型和返回值的类型)**

### 4.2 映射器语句

mybatis提供下面四种元素来定义不同类型的语句：select、insert、update、delete。

**insert语句**

<!-- 添加用户 -->

<insert id=*"insertUser"* parameterType=*"com.mybatis.model.NewUser"*>

insert into user(username,birthday,sex,address) values(#{username},#{birthday},#{sex},#{address})

<!-- 将插入数据的主键返回，返回到user对象中 -->

<selectKey keyProperty=*"id"* order=*"AFTER"* resultType=*"java.lang.Integer"*>

select @@identity

</selectKey>

<!--

<selectKey keyProperty="id" order="BEFORE" resultType="java.lang.String">

select uuid()

</selectKey>

-->

</insert>

* <insert>标签：用于执行数据库查询的，所有关于查询的都使用该标签。
* parameterType：要传入一个具体的pojo（包括用户信息）
* #{}中指定pojo的属性名，接收到pojo对象的属性值，mybatis通过OGNL获取对象的属性值。
* <selectKey>标签：用来返回插入数据的主键的，实际中如果有获得主键的需要就可以使用它。
* select @@identity：是sql函数，表示得到刚刚insert进去记录的主键值，只适用于自增主键。
* keyProperty：表示将查询到主键值设置到上面parameterType指定的对象的哪个属性。
* order：表示select last\_insert\_id()的执行顺序，是相对于insert语句来说的。
* resultType：表示select last\_insert\_id()的结果类型。

**select语句**

<select id=*"findUserById"* parameterType=*"int"* resultType=*"com.mybatis.model.NewUser"*>

select \* from tb\_user where id = #{abc}

</select>

* <select>标签：用于执行数据库查询的，所有关于查询的都使用该标签。
* id属性：标识映射文件中的sql，将sql语句封装到mappedStatement对象中，即statement的id，下面执行数据库的时候需要用到这个id。
* #{}：表示一个占位符，用来接收输入参数的。
* #{id}：id标识接收输入的参数，参数名称就是id，如果输入参数是简单类型，那么#{}中的参数名可以任意，可以value或者其他名称。
* parameterType：指定输入参数的类型，这个要和数据库中的保持一致。
* resultType：指定输出结果的类型，即查询结果所映射的java对象。
* ${}：表示拼接sql串，将接收到的参数的内容不加任何的修饰拼接在sql中,${}中只能使用value，但是使用${}来拼接sql，可能会引起sql注入，所以不建议使用这种方法。

**update语句**

<!-- 更新用户 -->

<update id=*"updateUser"* parameterType=*"com.mybatis.model.NewUser"*>

update tb\_user set username=#{userName},birthday=#{birthday},sex=#{sex},address=#{address} where id=#{id}

</update>

**delete语句**

<!-- 删除用户 -->

<delete id=*"deleteUser"* parameterType=*"java.lang.Integer"*>

delete from tb\_user where id=#{id}

</delete>

### 4.3 动态SQL

动态sql即对sql语句进行灵活操作，通过表达式进行判断，对sql进行灵活拼接、组装

<select id="findUserList" parameterType="mybatis.po.UserQueryVo" resultType="mybatis.po.User">

select \* from user

<!-- where可以自动去掉条件中的第一个and -->

<where>

<if test="user!=null">

<if test="user.sex!=null and user.sex!=''">

and user.sex = #{user.sex}

</if>

<if test="user.username!=null and user.username!=''">

and user.username like '%${user.username}%'

</if>

</if>

</where>

</select>

### 4.4 SQL片段

<sql id="query\_user\_where">

<if test="user!=null">

<if test="user.sex!=null and user.sex!=''">

and user.sex = #{user.sex}

</if>

<if test="user.username!=null and user.username!=''">

and user.username like '%${user.username}%'

</if>

</if>

</sql>

<select id="findUserList" parameterType="mybatis.po.UserQueryVo" resultType="mybatis.po.User">

select \* from user

<where>

<!-- 引用sql片段的id，如果refid指定的id不在本mapper文件中，需要在前面加上namespace -->

<include refid="query\_user\_where"></include>

<!-- 还可以引用其他sql片段 -->

</where>

</select>

### 4.5 foreach语句

SELECT \* FROM USER WHERE id=1 OR id=12 OR id=17形式

<sql id="query\_user\_where">

<if test="user!=null">

<if test="user.sex!=null and user.sex!=''">

and user.sex = #{user.sex}

</if>

<if test="user.username!=null and user.username!=''">

and user.username like '%${user.username}%'

</if>

</if>

<if test="ids!=null">

<!-- 使用右边的sql拼接：AND (id=1 OR id=12 OR id=17) -->

<foreach collection="ids" item="user\_id" open="AND (" close=")" separator="OR">

id=#{user\_id}

</foreach>

</if>

</sql>

SELECT \* FROM USER WHERE id IN(1,12,17)形式

<foreach collection="ids" item="user\_id" open="AND id IN(" close=")" separator=",">

#{user\_id}

</foreach>

* collection：指定输入对象中的集合属性，这里就是这个ids。
* item：表示每个遍历生成的对象，自己起个名儿，在foreach体中使用。
* open：开始遍历时拼接的sql串。
* close：结束遍历时拼接的sql串。
* separator：遍历的两个对象中需要拼接的sql串。
* 注意 你可以传递一个 List 实例或者数组作为参数对象传给 MyBatis。当你这么做的时 候,MyBatis 会自动将它包装在一个 Map 中,用名称在作为键。List 实例将会以“list” 作为键,而数组实例将会以“array”作为键。

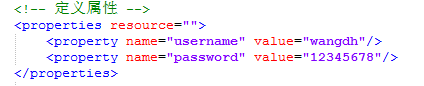
## 五、Mybatis配置文件

学习资料：<http://www.mybatis.org/mybatis-3/zh/configuration.html>

模板参见：[mybatis-config-template.xml](Mybatis/mybatis-config-template.xml)

### 5.1 properties属性

该元素用来配置一下可替换的数据，如数据库配置等。在该元素中定义的配置，可以在整个配置文件中进行使用。配置的格式如下：



可以使用resource或url属性配置外部配置文件。mybatis将按照下面的顺序来加载配置数据(当读取到相应配置后不会再继续往下)：

1. properties元素体内指定的属性
2. 通过resource或url指定的外部配置
3. 通过方法参数传递的属性

### 5.2 settings元素

该配置将会改变mybatis的运行行为，具体的配置项如下：

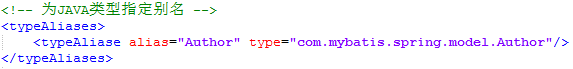
|  |  |  |  |
| --- | --- | --- | --- |
| **参数名** | **说明** | **有效值** | **默认值** |
| cacheEnabled | 所有映射器中配置的缓存的全局开关 | true/false | true |
| lazyLoadingEnabled | 延迟加载关联对象的全局开关 | true/false | false |
| aggressiveLazyLoading | 当启用时，对任意延迟属性的调用会使带有延迟加载属性的对象完整加载；反之，每种属性将会按需加载 | true/false | true |
| multipleResultSetsEnabled | 是否允许单一语句返回多结果集（需要兼容驱动） | true/false | true |
| useColumnLabel | 使用列标签代替列名 | true/false | true |
| useGeneratedKeys | 允许 JDBC 支持自动生成主 | true/false | false |
| autoMappingBehavior | 指定 MyBatis 应如何自动映射列到字段或属性。 NONE 表示取消自动映射；PARTIAL 只会自动映射没有定义嵌套结果集映射的结果集。 FULL 会自动映射任意复杂的结果集 | NONE/PARTIAL/ FULL | PARTIAL |
| autoMappingUnknownColumnBehavior |  | NONE/WARNING/ FAILING | NONE |
| defaultExecutorType | 配置默认的执行器。SIMPLE 就是普通的执行器；REUSE 执行器会重用预处理语句（prepared statements）； BATCH 执行器将重用语句并执行批量更新。 | SIMPLE/REUSE/ BATCH | SIMPLE |
| defaultStatementTimeout | 设置超时时间，它决定驱动等待数据库响应的秒数 | 任何正整数 | null |
| defaultFetchSize | 返回结果的数量，会被查询语句覆盖 | 任何正整数 | null |
| safeRowBoundsEnabled | 允许在嵌套语句中使用分页（RowBounds） | true/false | false |
| safeResultHandlerEnabled | 允许在嵌套语句中使用分页（ResultHandler） | true/false | false |
| mapUnderscoreToCamelCase | 否开启自动驼峰命名规则（camel case）映射，即从数据库列名 A\_COLUMN到 Java 属性名 aColumn 的类似映射。 | true/false | false |
| localCacheScope | MyBatis 利用本地缓存机制（Local Cache）防止循环引用（circular references）和加速重复嵌套查询。 默认值为 SESSION，这种情况下会缓存一个会话中执行的所有查询。 若设置值为 STATEMENT，本地会话仅用在语句执行上，对相同 SqlSession 的不同调用将不会共享数据。 | SESSION  STATEMENT | SESSION |
| logPrefix | 指定 MyBatis 增加到日志名称的前缀。 | 任何字符串 | null |

配置实例：



### 5.3 typeAliases元素

该元素用于为java类型设置一个较短的名字，以消除完全限定名的冗余，配置示例：



Author可以用在任何使用com.mybatis.spring.model.Author的地方

也可以指定一个包名，mybatis会自动扫描需要的Java bean，这种情况下，当该bean没有注解时，会使用首字母小写的非限定名作为别名。



### 5.4 typeHandlers元素

### 5.5 mappers元素

mapper文件包含SQL映射语句，每个mapper元素的配置有下面4种方式：



**resource**属性用来指定在classpath中的mapper.xml文件；

**url**属性用来通过文件系统路径或web URL地址指定mapper文件

**class**属性用来指向一个mapper接口

**package**属性用来指向可以找到mapper接口的包名

## 六、Mybatis集成通用mapper(非spring)

### 6.1 获取通用mapper和mybatis及数据库驱动jar包依赖

<!-- SqlServer 驱动 -->

<dependency>

<groupId>com.microsoft.sqlserver</groupId>

<artifactId>sqljdbc4</artifactId>

<version>${sqlserver.version}</version>

</dependency>

<!-- mybatis -->

<dependency>

<groupId>org.mybatis</groupId>

<artifactId>mybatis</artifactId>

<version>3.3.0</version>

</dependency>

<!-- 通用mapper -->

<dependency>

<groupId>org.mybatis.generator</groupId>

<artifactId>mybatis-generator-core</artifactId>

<version>1.3.2</version>

<scope>compile</scope>

<optional>true</optional>

</dependency>

### 6.2 在代码中使用下面语句继承mapper

// 需要使用这段代码来集成通用mapper

MapperHelper mapperHelper = new MapperHelper();

mapperHelper.registerMapper(FunctionParamsMapper.class);;

mapperHelper.processConfiguration(sqlSession.getConfiguration());

6.3 和使用mybatis一样正常使用了

## 七、Mybatis打印SQL语句

新建log4j.properties文件，添加下面内容：

log4j.rootLogger=debug,stdout

#

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

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

log4j.appender.stdout.layout.ConversionPattern=%d %p [%c] -%m%n

#

#

log4j.logger.com.ibatis=debug

#print package soc.dao sql

log4j.logger.soc.dao=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

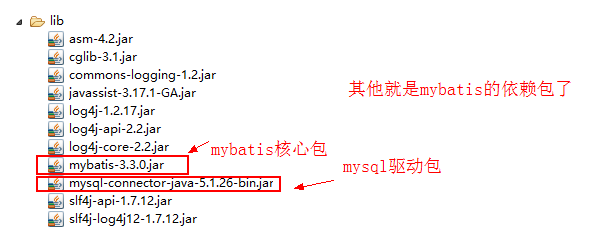
log4j.logger.java.sql.PreparedStatement=debug,stdout

## 八、Mybatis 入门

### 8.1、mybatis执行流程



### 8.2、mybatis环境准备



### 8.3 mybatis日志配置

# Global logging configuration

# developer-->DEBUG productor-->INFO or ERROR

log4j.rootLogger=DEBUG, stdout

# MyBatis logging configuration...

log4j.logger.org.mybatis.example.BlogMapper=TRACE

# 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

## 九、学习资料

<http://blog.csdn.net/eson_15/article/details/51592608>

<http://blog.csdn.net/owen_william/article/details/51815605>

<http://legend2011.blog.51cto.com/3018495/973579>

## 十、Mybatis缓存

缓存的作用是减轻数据库的压力，提高数据库的性能的。mybatis中提供了一级缓存和二级缓存。**只有在配置中将缓存开启了，才会自动进行缓存**

* 一级缓存是SqlSession级别的缓存。在操作数据库时需要构造sqlSession对象，在对象中有一个数据结构（HashMap）用于存储缓存数据。不同的sqlSession之间的缓存数据区域（HashMap）是互相不影响的。
* 二级缓存是mapper级别的缓存，多个SqlSession去操作同一个Mapper的sql语句，多个SqlSession可以共用二级缓存，二级缓存是跨SqlSession的。

**一级缓存的原理**：第一次发起查询用户id为1的用户信息，先去找缓存中是否有id为1的用户信息，如果没有，从数据库查询用户信息。得到用户信息，将用户信息存储到一级缓存中。 如果中间sqlSession去执行commit操作（执行插入、更新、删除），则会清空SqlSession中的一级缓存，这样做的目的为了让缓存中存储的是最新的信息，避免脏读。第二次发起查询用户id为1的用户信息，先去找缓存中是否有id为1的用户信息，缓存中有，直接从缓存中获取用户信息。(**缓存的HashMap的key规则是啥？**)

如果两次查找的中间，由其他的SQLSession进行过数据的提交，则在当前的sqlsession查询中数据还是不是最新的。

**禁止一级缓存：在对应的查询中，加上flushCache="true"**

**二级缓存的原理**：sqlSession1去查询用户id为1的用户信息，查询到用户信息会将查询数据存储到该UserMapper的二级缓存中；如果SqlSession3去执行相同 mapper下sql，执行commit提交，则会清空该UserMapper下二级缓存区域的数据；sqlSession2去查询用户id为1的用户信息，去缓存中找是否存在数据，如果存在直接从缓存中取出数据。

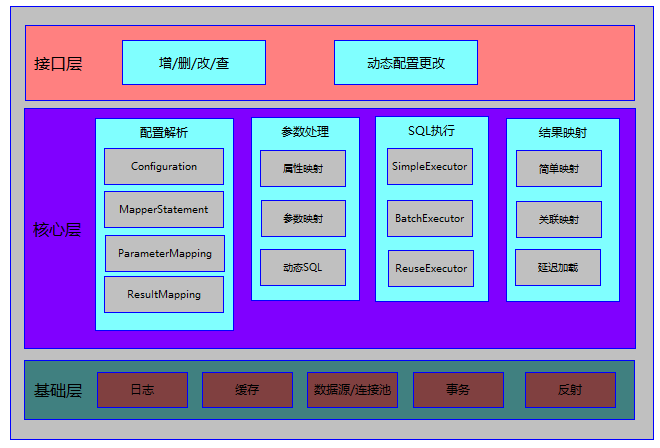
二级缓存与一级缓存区别在于二级缓存的范围更大，多个sqlSession可以共享一个mapper中的二级缓存区域。mybatis是按照不同mapper有不同的namespace来区分二级缓存区域的，也就是说，如果两个mapper的namespace相同，即使是两个mapper，那么这两个mapper中执行sql查询到的数据也将存在相同的二级缓存区域中。

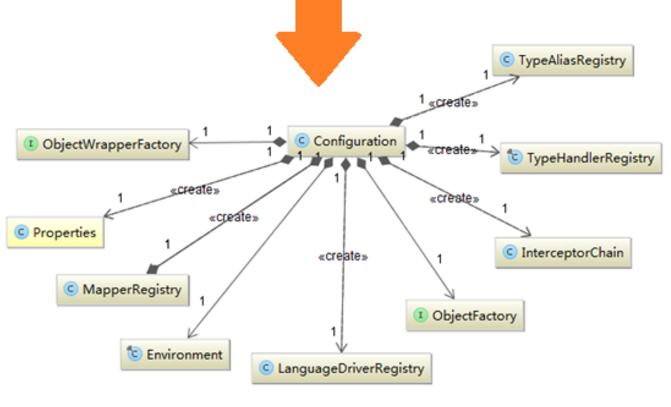
**二级缓存的开启，除了在全局配置文件中开启缓存，还需要在对应的mapper.xml文件中开启缓存配置。并且需要缓存的POJO类还需要继承Serializable接口**。因为二级缓存数据存储介质多种多样，不一定只存在内存中，有可能存在硬盘中，如果我们要再取这个缓存的话，就需要反序列化了。

当某个mapper.xml中配置了二级缓存，则默认这个xml下的所有查询结果都会进行缓存，但如果某个查询不想进行缓存，则可在对应的select标签中加上useCache="false"来禁用二级缓存。

在mapper的同一个namespace中，如果有其它insert、update、delete操作数据后需要刷新缓存，如果不执行刷新缓存会出现脏读。 设置statement配置中的flushCache=”true” 属性，默认情况下为true，即刷新缓存，如果改成false则不会刷新。使用缓存时如果手动修改数据库表中的查询数据会出现脏读。

## 十一、Spring和mybatis的整合





## 十二、Mybatis参数说明

当只有**一个参数**的时候(基本类型或TypeHandler配置的类型)，可以不使用@Param注解，因为这种情况下(除集合和数组外)，Mybatis**不关心这个参数叫什么名字**，就会直接把这个唯一的参数值拿来使用。

当有多个参数的时候，默认会使用arg0，arg1…或param1，param2…进行名称参数。

一般不推荐使用默认的规则，而是使用@Param注解执行参数名称。给参数配置@Param注解后，Mybatis就会自动将参数封装成Map类型，@Param注解值会作为Map的key，也是作为参数名称。

**Remark**：

1) 当只有一个参数时，SQL中的参数名称可以随便指定。

2) 当需要有多个参数时，可以封装成JavaBean或Map，SQL中的参数名称就是JavaBean的属性名称或Map的key。

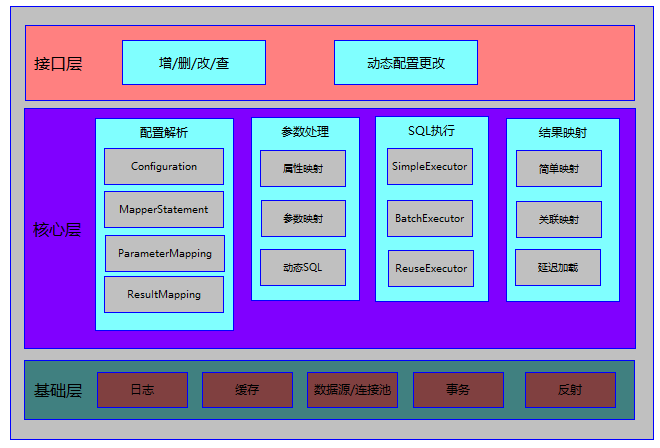
3) 多个参数时，也可以指定@Param参数指定SQL中参数的名称。

## 十三、源码分析

<http://www.importnew.com/25005.html>

<http://www.importnew.com/25836.html>

<http://www.cnblogs.com/xrq730/category/994185.html>



1.Configuration类是一个很重的类，包含的配置的一切信息，以及一些从配置构造出来的对象，如Executor、Statement等

2.Mybatis大量使用了代理模式

3.每个Mybatis的Mapper接口会经过MapperProxy代理，接口中包含的每个方法对应到一个MapperMethod对象，该对象包含了每个执行的SQL id和类型(增删改查)

4.程序启动时，Configuration会含有每个方法对象需要执行的详细SQL信息，封装在了MappedStatement对象中，而MapperMethod中的SqlCommand对象只含有MappedStatement对象的部分数据(id和类型)，之间是一一对应的。

执行一个SQL的大致流程：

Mapper对象 -- >获取对应的代理类Proxy -- > 找到方法对应的MapperMethod --> 将方法对应的参数转成SQL参数并和MapperMethod中的SqlCommad id一起传递给SqlSession -- > 从configuration中根据SQL id获取对应的MappedStatement对象 --> 和参数一起交给对应的Executor进行执行