SQL映射文件有很少的几个顶级元素（按照他们应该被定义的顺序如下）。

* cache -  配置给定命名空间的缓存
* cache-ref –  从其他命名空间引用缓存配置。
* resultMap –  最复杂，也是最有力量的元素，用来描述如何从数据库结果集中来加载你的对象。
* parameterMap –  已经被废弃了！老式风格的参数映射。内联参数是首选，这个元素可能在将来被移除。
* sql –  可以重用的SQL块，也可以被其他语句引用。
* insert –  映射插入语句
* update –  映射更新语句
* delete -映射删除语句
* select - 映射查询语句

# SELECT

简单的SELECT 语句如下：

 写道

<select id=”selectPerson” parameterType=”int” resultType=”hashmap”>   
SELECT \* FROM PERSON WHERE ID = #{id}   
</select>

 这个语句被称作selectPerson，使用一个int （或Integer）类型的参数，并返回一个HashMap类型的对象

#{id}告诉mybatis创建了一个PreparedStatement(预处理语句)参数。在JDBC中，类似的代码如下

 写道

String selectPerson = “SELECT \* FROM PERSON WHERE ID=?”;   
PreparedStatement ps = conn.prepareStatement(selectPerson);   
ps.setInt(1,id);

 SELECT的属性还有很多的属性可以配置，具体的如下：

* id        在命名空间中唯一的标识符，可以被用来引用这条语句。
* parameterType    将会传入这条语句的参数类的完全限定名或别名。
* resultType  从这条语句中返回的期望类型的类的完全限定名或别名。注意集合情形，那应该是集合可以包含的类型，而不能是集合本身。使用resultType或resultMap，但不能同时使用。
* resultMap 命名引用外部的resultMap。返回map是MyBatis最具力量的特性，对其有一个很好的理解的话，许多复杂映射的情形就能被解决了。使用resultMap 或resultType，但不能同时使用
* flushCache  将其设置为true，无论语句什么时候被调用，都会导致缓存被清空。默认值：false。
* useCache  将其设置为true，将会导致本条语句的结果被缓存。默认值：true。
* timeout  这个设置驱动程序等待数据库返回请求结果，并抛出异常时间的最大等待值。默认不设置（驱动自行处理）。
* fetchSize 这是暗示驱动程序每次批量返回的结果行数。默认不设置（驱动自行处理）。
* statementType   STATEMENT,PREPARED或CALLABLE的一种。这会让MyBatis使用选择使用Statement，PreparedStatement或CallableStatement。默认值：PREPARED。
* resultSetType FORWARD\_ONLY|SCROLL\_SENSITIVE|SCROLL\_INSENSITIVE中的一种。默认是不设置（驱动自行处理）。

# insert，update，delete

数据修改语句insert，update和delete在它们的实现中非常相似。

写道

<insert   
id="insertAuthor"   
parameterType="domain.blog.Author"   
flushCache="true"   
statementType="PREPARED"   
keyProperty=""   
useGeneratedKeys=""   
timeout="20000">

# SQL

这个元素可以被用来定义可重用的SQL代码段，可以包含在其他语句中，例如

 写道

<sql id=”userColumns”> id,username,password </sql>   
这个SQL片段可以被包含在其他语句中，例如：   
<select id=”selectUsers” parameterType=”int” resultType=”hashmap”>   
select <include refid=”userColumns”/>   
from some\_table   
where id = #{id}   
</select>

# Parameters

这个元素说的直白点就是定义参数。注意一个语句中只能有一个参数。所以参数类型在以后的使用中，可能需要复杂的类型，比如hashmap，一个复杂的对象等。例如：

 写道

<insert id=”insertUser” parameterType=”User” >   
insert into users (id, username, password)   
values (#{id}, #{username}, #{password})   
</insert>

如果User类型的参数对象传递到了语句中，id、username和password属性将会被查找，

然后它们的值就被传递到预处理语句的参数中。

# resultMap

resultMap 元素是MyBatis中最重要最强大的元素。它就是让你远离90%的需要从结果集中取出数据的JDBC代码的那东西，而且在一些情形下允许你做一些JDBC不支持的事情。事实上，编写相似于对复杂语句联合映射这些等同的代码，也许可以跨过上千行的代码。ResultMap的设计就是简单语句不需要明确的结果映射，而很多复杂语句确实需要描述它们的关系。

 写道

<select id=”selectUsers” parameterType=”int” resultType=”hashmap”>   
select id, username, hashedPassword   
from some\_table   
where id = #{id}   
</select>

这个语句查询出来的所有列将会自动映射到HashMap的键上，这个是由resultType的属性来指定的。但是HashMap不能很好描述一个领域模型。那样你的应用程序将会使用JavaBeans或POJO 来作为领域模型。

例如一个javabean如下

**[java]** [view plain](http://blog.csdn.net/whatlookingfor/article/details/38381855" \o "view plain) [copy](http://blog.csdn.net/whatlookingfor/article/details/38381855" \o "copy)

1. **package** com.someapp.model;
2. **public** **class** User {
3. **private** **int** id;
4. **private** String username;
5. **private** String hashedPassword;
7. //set和get方法。。。。
8. }

 写道

<select id=”selectUsers” parameterType=”int”   
resultType=”com.some app.model.User”>   
select id, username, hashedPassword   
from some\_table   
where id = #{id}   
</select>

当然在使用的时候可以不用输入全名，只需在XML的配置文件中进行以下设置。

 写道

<!-- 在XML配置文件中-->   
<typeAlias type=”com.some app.model.User” alias=”User”/>   
<!-- 在SQL映射的XML文件中-->   
<select id=”selectUsers” parameterType=”int”   
resultType=”User”>   
select id, username, hashedPassword   
from some\_table   
where id = #{id}   
</select>

# 缓存

MyBatis包含一个非常强大的查询缓存特性，它可以非常方便地配置和定制。默认情况下是没有开启缓存的，除了局部的session 缓存。要开启二级缓存，你需要在你的SQL映射文件中添加一行： <cache/>

这个简单语句的效果如下：

1  映射语句文件中的所有select语句将会被缓存。

2  映射语句文件中的所有insert，update和delete语句会刷新缓存。

3  缓存会使用Least Recently Used（LRU，最近最少使用的）**[算法](http://lib.csdn.net/base/31" \o "算法与数据结构知识库" \t "http://blog.csdn.net/whatlookingfor/article/details/_blank)**来收回。

4  根据时间表（比如no  Flush  Interval，没有刷新间隔），缓存不会以任何时间顺序来刷新。

5  缓存会存储列表集合或对象（无论查询方法返回什么）的1024个引用。

6  缓存会被视为是read/write（可读/可写）的缓存，意味着对象检索不是共享的，而且可以安全地被调用者修改，而不干扰其他调用者或线程所做的潜在修改。

一个简单的实例如下

 写道

<cache   
eviction="FIFO"   
flushInterval="60000"   
size="512"   
readOnly="true"/>

 这个配置创建了一个FIFO缓存，并每隔60秒刷新，存数结果对象或列表的512个引用，而且返回的对象被认为是只读的，因此在不同线程中的调用者之间修改它们会导致冲突。

可用的收回策略有：

* LRU –  最近最少使用的：移除最长时间不被使用的对象。
* FIFO –  先进先出：按对象进入缓存的顺序来移除它们。
* SOFT –  软引用：移除基于垃圾回收器状态和软引用规则的对象。
* WEAK –  弱引用：更积极地移除基于垃圾收集器状态和弱引用规则的对象。 默认的是LRU。

flushInterval（刷新间隔）可以被设置为任意的正整数，而且它们代表一个合理的毫秒形式的时间段。默认情况是不设置，也就是没有刷新间隔，缓存仅仅调用语句时刷新。

size（引用数目）可以被设置为任意正整数，要记住你缓存的对象数目和你运行环境的可用内存资源数目。默认值是1024。

 readOnly（只读）属性可以被设置为true 或false。只读的缓存会给所有调用者返回缓存对象的相同实例。因此这些对象不能被修改。这提供了很重要的性能优势。可读写的缓存会返回缓存对象的拷贝（通过序列化）。这会慢一些，但是安全，因此默认是false。

# 动态SQL

动态SQL的元素有以下几个：

if  choose(when,otherwise)  trim(where,set)   foreach

## if

 写道

<select id=”findActiveBlogWithTitleLike”   
parameterType=”Blog” resultType=”Blog”>   
SELECT \* FROM BLOG   
WHERE state = „ACTIVE‟   
<if test=”title != null”>   
AND title like #{title}   
</if>   
</select>

## choose, when, otherwise

有时我们不想应用所有的条件，相反我们想选择很多情况下的一种。和**[Java](http://lib.csdn.net/base/17" \o "Java EE知识库" \t "http://blog.csdn.net/whatlookingfor/article/details/_blank)**中的switch语句相似，MyBatis提供choose元素。

 写道

<select id=”findActiveBlogLike”   
parameterType=”Blog” resultType=”Blog”>   
SELECT \* FROM BLOG WHERE state = „ACTIVE‟   
<choose>   
<when test=”title != null”>   
AND title like #{title}   
</when>   
<when test=”author != null and author.name != null”>   
AND title like #{author.name}   
</when>   
<otherwise>   
AND featured = 1   
</otherwise>   
</choose>   
</select>

## trim, where, set

 写道

<select id=”findActiveBlogLike”   
parameterType=”Blog” resultType=”Blog”>   
SELECT \* FROM BLOG   
WHERE   
<if test=”state != null”>   
state = #{state}   
</if>   
<if test=”title != null”>   
AND title like #{title}   
</if>   
<if test=”author != null and author.name != null”>   
AND title like #{author.name}   
</if>   
</select>

 如果这些条件都没有匹配上将会发生什么？这条SQL结束时就会成这样：

  SELECT \* FROM BLOG  WHERE，从而导致查询失败。

如果仅仅第二个条件匹配，这条SQL结束时就会是这样：

  SELECT \* FROM BLOG WHERE AND title like

MyBatis有一个简单的处理，这在90%的情况下都会有用。而在不能使用的地方，你可以自定义处理方式。加上一个简单的改变，所有事情都会顺利进行：

 写道

<select id=”findActiveBlogLike”   
parameterType=”Blog” resultType=”Blog”>   
SELECT \* FROM BLOG   
<where>   
<if test=”state != null”>   
state = #{state}   
</if>   
<if test=”title != null”>   
AND title like #{title}   
</if>   
<if test=”author != null and author.name != null”>   
AND title like #{author.name}   
</if>   
</ where >   
</select>

 where元素知道如果由被包含的标记返回任意内容，就仅仅插入“WHERE”。而且，如果以“AND”或“OR”开头的内容，那么就会跳过WHERE不插入。

## foreach

另外一个动态SQL通用的必要操作是迭代一个集合，通常是构建在IN条件中的

 写道

<select id="selectPostIn" resultType="domain.blog.Post">   
SELECT \*   
FROM POST P   
WHERE ID in   
<foreac h ite m="item" inde x="index" c ollection="list"   
open="(" separator="," close=")">   
#{item}   
</fore ach>   
</select>

 foreach 元素是非常强大的，它允许你指定一个集合，声明集合项和索引变量，它们可以用在元素体内。它也允许你指定开放和关闭的字符串，在迭代之间放置分隔符。