insert, update 和 delete

数据变更语句 insert，update 和 delete 的实现非常接近：

```
<insert

  id="insertAuthor"

  parameterType="domain.blog.Author"

  flushCache="true"

  statementType="PREPARED"

  keyProperty=""

  keyColumn=""

  useGeneratedKeys=""

  timeout="20">



<update

  id="updateAuthor"

  parameterType="domain.blog.Author"

  flushCache="true"

  statementType="PREPARED"

  timeout="20">



<delete

  id="deleteAuthor"

  parameterType="domain.blog.Author"
```

```
flushCache="true"

statementType="PREPARED"

timeout="20">
```

Insert, Update, Delete 元素的属性

属性	描述
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的类全限定名或别名。这个属性是可选的，因为 MyBatis 可以通过类型处理器（TypeHandler）推断出具体传入语句的参数，默认值为未设置（unset）。
~~parameterMap~~	~~用于引用外部 parameterMap 的属性，目前已被废弃。请使用行内参数映射和 parameterType 属性。~~
flushCache	将其设置为 true 后，只要语句被调用，都会导致本地缓存和二级缓存被清空，默认值：（对 insert、update 和 delete 语句）true。
timeout	这个设置是在抛出异常之前，驱动程序等待数据库返回请求结果的秒数。默认值为未设置（unset）（依赖数据库驱动）。
statementType	可选 STATEMENT，PREPARED 或 CALLABLE。这会让 MyBatis 分别使用 Statement，PreparedStatement 或 CallableStatement，默认值：PREPARED。
useGeneratedKeys	（仅适用于 insert 和 update）这会令 MyBatis 使用 JDBC 的 getGeneratedKeys 方法来取出由数据库内部生成的主键（比如：像 MySQL 和 SQL Server 这样的关系型数据库管理系统的自动递增字段），默认值：false。

Insert, Update, Delete 元素的属性

属性	描述
keyProperty	（仅适用于 insert 和 update）指定能够唯一识别对象的属性，MyBatis 会使用 getGeneratedKeys 的返回值或 insert 语句的 selectKey 子元素设置它的值，默认值：未设置（unset）。如果生成列不止一个，可以用逗号分隔多个属性名称。
keyColumn	（仅适用于 insert 和 update）设置生成键值在表中的列名，在某些数据库（像 PostgreSQL）中，当主键列不是表中的第一列的时候，是必须设置的。如果生成列不止一个，可以用逗号分隔多个属性名称。
databaseId	如果配置了数据库厂商标识（databaseIdProvider），MyBatis 会加载所有不带 databaseId 或匹配当前 databaseId 的语句；如果带和不带的语句都有，则不带的会被忽略。

下面是 insert，update 和 delete 语句的示例：

```
<insert id="insertAuthor">

  insert into Author (id,username,password,email,bio)

  values (#{id},#{username},#{password},#{email},#{bio})

</insert>



<update id="updateAuthor">

  update Author set

    username = #{username},

    password = #{password},

    email = #{email},
```

```
    bio = #{bio}
  where id = #{id}
</update>

<delete id="deleteAuthor">
  delete from Author where id = #{id}
</delete>
```

如前所述，插入语句的配置规则更加丰富，在插入语句里面有一些额外的属性和子元素用来处理主键的生成，并且提供了多种生成方式。

首先，如果你的数据库支持自动生成主键的字段（比如 MySQL 和 SQL Server），那么你可以设置 useGeneratedKeys=”true”，然后再把 keyProperty 设置为目标属性就 OK 了。例如，如果上面的 Author 表已经在 id 列上使用了自动生成，那么语句可以修改为：

```
<insert id="insertAuthor" useGeneratedKeys="true"
    keyProperty="id">
  insert into Author (username,password,email,bio)
  values (#{username},#{password},#{email},#{bio})
</insert>
```

如果你的数据库还支持多行插入，你也可以传入一个 `Author` 数组或集合，并返回自动生成的主键。

```
<insert id="insertAuthor" useGeneratedKeys="true"
    keyProperty="id">
  insert into Author (username, password, email, bio) values
  <foreach item="item" collection="list" separator=",">
    (#{item.username}, #{item.password}, #{item.email}, #{item.bio})
  </foreach>
```

```
</insert>
```

对于不支持自动生成主键列的数据库和可能不支持自动生成主键的 JDBC 驱动，MyBatis 有另外一种方法来生成主键。

这里有一个简单（也很傻）的示例，它可以生成一个随机 ID（不建议实际使用，这里只是为了展示 MyBatis 处理问题的灵活性和宽容度）：

```
<insert id="insertAuthor">

  <selectKey keyProperty="id" resultType="int" order="BEFORE">

    select CAST(RANDOM()*1000000 as INTEGER) a from SYSIBM.SYSDUMMY1

  </selectKey>

  insert into Author

    (id, username, password, email,bio, favourite_section)

  values

    (#{id}, #{username}, #{password}, #{email}, #{bio}, #{favouriteSection,jdbcType=VARC
HAR})

</insert>
```

在上面的示例中，首先会运行 selectKey 元素中的语句，并设置 Author 的 id，然后才会调用插入语句。这样就实现了数据库自动生成主键类似的行为，同时保持了 Java 代码的简洁。

selectKey 元素描述如下：

```
<selectKey

  keyProperty="id"

  resultType="int"

  order="BEFORE"

  statementType="PREPARED">
```

selectKey 元素的属性	
属性	描述

keyProperty	selectKey 语句结果应该被设置到的目标属性。如果生成列不止一个，可以用逗号分隔多个属性名称。
keyColumn	返回结果集中生成列属性的列名。如果生成列不止一个，可以用逗号分隔多个属性名称。
resultType	结果的类型。通常 MyBatis 可以推断出来，但是为了更加准确，写上也不会有什么问题。MyBatis 允许将任何简单类型用作主键的类型，包括字符串。如果生成列不止一个，则可以使用包含期望属性的 Object 或 Map。
order	可以设置为 BEFORE 或 AFTER。如果设置为 BEFORE，那么它首先会生成主键，设置 keyProperty 再执行插入语句。如果设置为 AFTER，那么先执行插入语句，然后是 selectKey 中的语句 - 这和 Oracle 数据库的行为相似，在插入语句内部可能有嵌入索引调用。
statementType	和前面一样，MyBatis 支持 STATEMENT，PREPARED 和 CALLABLE 类型的映射语句，分别代表 Statement, PreparedStatement 和 CallableStatement 类型。

sql

这个元素可以用来定义可重用的 SQL 代码片段，以便在其它语句中使用。 参数可以静态地（在加载的时候）确定下来，并且可以在不同的 include 元素中定义不同的参数值。比如：

```
<sql id="userColumns"> ${alias}.id,${alias}.username,${alias}.password </sql>
```

这个 SQL 片段可以在其它语句中使用，例如：

```
<select id="selectUsers" resultType="map">
  select
    <include refid="userColumns"><property name="alias" value="t1"/></include>,
    <include refid="userColumns"><property name="alias" value="t2"/></include>
  from some_table t1
    cross join some_table t2
</select>
```

也可以在 include 元素的 refid 属性或内部语句中使用属性值，例如：

```
<sql id="sometable">
```

```
  ${prefix}Table
</sql>

<sql id="someinclude">
  from
    <include refid="${include_target}"/>
</sql>

<select id="select" resultType="map">
  select
    field1, field2, field3
  <include refid="someinclude">
    <property name="prefix" value="Some"/>
    <property name="include_target" value="sometable"/>
  </include>
</select>
```