MyBatis基础

**2019年02月**

# 基础知识

## 概述

MyBatis是一个支持普通SQL查询、存储过程以及高级映射的持久层框架，它消除了几乎所有的JDBC代码和参数的手动设置以及对结果集的检索，并使用简单的XML或注解进行配置和原始映射，用以将接口和Java的POJO映射成数据库中的记录。

MyBatis也被称为ORM框架。ORM框架的工作原理如下图。



POJO：（Plain Ordinary Java Object）简单的Java对象。可以理解为一个类，这个类不包含任何业务逻辑，跟数据表字段一一对应，每个字段都有对应的get和set方法。在某些框架中，跟VO、Entity其实是差不多的概念。具体可以参考如下链接：

<https://baike.baidu.com/item/POJO/3311958?fr=aladdin>

ORM框架：ORM（Object Relational Mapping）框架采用[元数据](https://baike.baidu.com/item/%E5%85%83%E6%95%B0%E6%8D%AE/1946090)来描述对象——关系映射细节，元数据一般采用XML格式，并且存放在专门的对象一映射文件中。具体可以参考如下链接：

<https://baike.baidu.com/item/ORM%E6%A1%86%E6%9E%B6/15541111?fr=aladdin>

## MyBatis和Hibernate的比较

Hebernate和MyBatis都是常见的ORM框架，区别如下：

Hebernate：一个全表映射的框架。适合场景不太复杂且对性能要求不高的项目中使用。

优点：

开发者只需定义好持久化对象（即POJO）到数据库表的映射关系，就可以通过Hibername提供的方法完成持久层操作。

开发者不需要熟练地掌握SQL语句的编写，Hibername会根据制定的存储逻辑，自动生成对应的SQL，并调用JDBC接口来执行。

开发效率会高于MyBatis（当然这种说法是基于存在牛逼架构师和十分理解Hibernate使用方法的前提）。

跨数据库开发迁移方便。

缺点：

封装度强，学习门槛高。

封装度太强，多表关联时，对SQL查询的支持较差。

更新数据时，需要发送所有字段。

不支持存储过程。

不能通过优化SQL来优化性能。

MyBatis：一个半自动映射的框架。适合一些复杂的和需要优化性能的项目使用。

优点：

可以配置动态SQL并优化SQL。

可以通过配置决定SQL的映射规则。

支持存储过程。

学习门槛和Hibernate对比低。

缺点：

需要手动匹配提供POJO、SQL和映射关系，相对Hibernate多了手动配置SQL的操作。

开发工作量比较大。

## MyBatis的工作原理

原理如下图：



**mybatis-config.xml：**配置MyBatis的运行环境信息，其中主要内容是数据库连接。

**Mapper.xml：**Mapper.xml文件即SQL映射文件，该文件配置了各种操作数据库的SQL语句。这种文件需要在mybatis-config.xml中加载才能执行。mybatis-config.xml可以加载多个配置文件，每个配置文件对应数据库的一张表。

**SqlSessionFactory：**会话工厂是一个接口类，基本上方法的功能主要是获取mybatis-config.xml的环境配置信息。在源码中可以看到有一个DefaultSqlSessionFactory类和SqlSessionManager实现SqlSssionFactory类的方法。初步看DefaultSqlSessionFactory类单纯实现了SqlSessionFactory的方法，而SqlSessionManager类既实现了SqlSessionFactory类和SqlSession类的方法。

**SqlSession：**SqlSession对象由会话工厂创建，该对象包含了执行SQL的所有方法。包括增删查改、提交回退等等的方法。在源码中可以看到有一个DefaultSqlSession和SqlSessionManager实现SqlSession类的方法。其中有一个selectOne方法，源码能看到如果SQL查询的结果超过一条，是会报异常的。

**Executor：**MyBatis底层定义了一个Executor接口来操作数据库，它会根据SqlSession传递的参数动态地生成需要执行的SQL语句，同时负责查询缓存的维护。从源码看来，Executor也有增删查改的方法，不过没有SqlSession这么多元化，例如查这方面只有query方法，增删改主要是update方法，区别就在于接受SqlSession传递的参数执行数据库操作了。在源码中，初步能看到一个抽象类BaseExecutor和另一个类CachingExecutor实现Executor接口的方法。

**MappedStatement：**在Executor接口的执行方法中，包含一个MappedStatement类型的参数，该参数是对映射信息的封装，用于存储要映射的SQL语句的id、参数等。Mapper.xml文件中一个SQL对应一个MappedStatement对象，SQL的id即是MappedStatement的id。从源码中可以看到，MappedStatement这个类是一个final修饰的类，也就是说不允许存在子类的类。另外，里面存在一个Builder的内部类。

**输入参数映射：**在执行方法时，MappedStatement对象会对用户执行SQL语句输入参数进行定义（可以定义为Map、List类型、基本类型和POJO类型），Executor执行器会通过MappedStatement对象在执行SQL前，将输入的Java对象映射到SQL语句中。这里对输入参数的映射过程就类似于JDBC编程中preparedStatement对象设置参数的过程。

**输出参数映射：**在执行完SQL语句后，MappedStatement对象会对执行输出的结果进行定义（可以定义为Map、List类型、基本类型和POJO类型），Executor执行器会通过MappedStatement对象在执行SQL语句后，将输出结果映射到Java对象中。这种输出结果映射到Java对象的过程类似于JDBC编程中对结果的解析处理过程。

# MyBatis的核心对象

## SqlSessionFactory

### 概述

1. SqlSessionFactory是单个数据库映射关系经过编译后的内存映像。
2. SqlSeesionFactory通过SqlSessionFactoryBuilder和mybatis-config.xml配置文件创建。
3. SqlSessionFactory对象是线程安全的，一旦被创建，在整个应用执行期间都会存在。  
   （如果多次地创建同一个数据库的SqlSessionFactory，那么此数据库的资源将很容易被耗尽。所以每一个数据库都会只对应一个SqlSessionFactory。建议利用单例模式创建SqlSessionFactory。）  
   （详细单例的代码可以参考SqlSessionBuilder.java）
4. SqlSessionFactory的主要作用是创建SqlSession。

### 源码分析

SqlSessionFactoryBuilder结合mybatis-config.xml配置文件创建的SqlSessionFactory仅仅只是把配置内容放到应用内存中，具体存储配置的类是Configuration。

## SqlSession

### 概述

1. SqlSession是应用程序与持久层之间执行交互操作得一个单线程对象，其主要作用是执行持久化操作。  
   （这里所谓的持久化操作，可以狭义理解为数据库的增删查改）
2. SqlSeesion包含了数据库中所有执行SQL的操作方法，由于其底层封装了JDBC连接，所以可以直接使用其实例来执行已映射的SQL语句。具体方法可以参考如下链接的SqlSession部分  
   （<http://www.mybatis.org/mybatis-3/zh/java-api.html>）
3. SqlSession实例是线程不安全的，所以不能共享SqlSession实例，每一个线程都应该有一个自己的SqlSession实例。
4. 使用完SqlSession对象之后，要及时关闭，通常可以放在finally块中关闭。

### 源码分析

1. SqlSession通过SqlSessionFactory的openSession方法创建获取。获取的参数是SqlSessionFactory里面的Configuration对象，Executor对象（这个在基础知识有提及过）以及autoCommit布尔值。  
   Configuration对象：SqlSessionFactory创建的时候加载。  
   Executor对象：根据配置中的数据源信息创建的事务对象(Transaction)生成。  
   autoCommit布尔值：手动传入，默认为false。
2. 数据库连接的动作是在Transaction对象中进行的。
3. 当SqlSession执行close方法的时候，实则上是利用了Executor对象执行了Trasaction的close，也就是回归到最基本的jdbc的关闭连接过程。当然，如果是数据库连接池，jdbc的close则是回收连接的操作。

## Executor

### 概述

1. Executor是提供给SqlSession调用，一个相对底层，同时也是执行持久操作的类。
2. Executor提供的持久性操作没有SqlSession这么多，大概只有query、doUpadate、commit等。

# 配置文件mybatis-config.xml

## 主要元素

mybatis-config.xml中包含多个元素，如下图所示：



值得注意的是，图上的元素必须由上到下按照顺序进行配置，否则MyBatis在解析XML配置文件时会报错。因为Mybatis是适用DTD作为XML的校验文档，而XML规范中DTD顺序是有严格要求的。如果不按照顺序配置，那么，程序启动的时候，会出现报错。

### <properties>元素（强烈建议使用）

<properties>元素用于配置外在化的配置，用得最多就是配置jdbc基础配置文件jdbc.properties。

jdbc.propertis配置文件样例如下：

**jdbc.driver**=**org.mariadb.jdbc.Driver  
jdbc.url**=**jdbc:mariadb://192.168.81.130:3306/mybatis  
jdbc.username**=**Allan  
jdbc.password**=**123**

根据上述配置，配置数据源信息的时候就可以使用${jdbc.driver}进行替代。

### <settings>元素（禁用一二级缓存）

<settings>元素主要用于改变MyBatis运行时的行为，例如开启二级缓存、开启延时加载等。

不配置<settings>元素，也可以正常运行MyBatis。

|  |  |  |  |
| --- | --- | --- | --- |
| 设置参数名 | 描述 | 可选值 | 默认值 |
| cacheEnabled | 该配置影响的所有映射器中配置的缓存的全局开关。 | true | false | TRUE |
| lazyLoadingEnabled | 延迟加载的全局开关。当开启时，所有关联对象都会延迟加载。 特定关联关系中可通过设置fetchType属性来覆盖该项的开关状态。 | true | false | FALSE |
| aggressiveLazyLoading | 当开启时，任何方法的调用都会加载该对象的所有属性。否则，每个属性会按需加载（参考lazyLoadTriggerMethods). | true | false | false |
| 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。不同的驱动在这方面会有不同的表现， 具体可参考相关驱动文档或通过测试这两种不同的模式来观察所用驱动的结果。 | true | false | TRUE |
| useGeneratedKeys | 允许 JDBC 支持自动生成主键，需要驱动兼容。 如果设置为 true 则这个设置强制使用自动生成主键，尽管一些驱动不能兼容但仍可正常工作（比如 Derby）。 | 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（无论是否嵌套）。 | NONE, PARTIAL, FULL | PARTIAL |
| autoMappingUnknownColumnBehavior | 指定发现自动映射目标未知列（或者未知属性类型）的行为。NONE: 不做任何反应。WARNING: 输出提醒日志 (‘org.apache.ibatis.session.AutoMappingUnknownColumnBehavior’ 的日志等级必须设置为 WARN)。FAILING: 映射失败 (抛出 SqlSessionException) | 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。 | 任意正整数 | Not Set (null) |
| defaultFetchSize | 为驱动的结果集获取数量（fetchSize）设置一个提示值。此参数只可以在查询设置中被覆盖。 | 任意正整数 | Not Set (null) |
| safeRowBoundsEnabled | 允许在嵌套语句中使用分页（RowBounds）。 If allow, set the false. | true | false | FALSE |
| safeResultHandlerEnabled | 允许在嵌套语句中使用分页（ResultHandler）。 If allow, set the false. | true | false | TRUE |
| mapUnderscoreToCamelCase | 是否开启自动驼峰命名规则（camel case）映射，即从经典数据库列名 A\_COLUMN 到经典 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 |
| jdbcTypeForNull | 当没有为参数提供特定的 JDBC 类型时，为空值指定 JDBC 类型。 某些驱动需要指定列的 JDBC 类型，多数情况直接用一般类型即可，比如 NULL、VARCHAR 或 OTHER。 | JdbcType enumeration. Most common are: NULL, VARCHAR and OTHER | OTHER |
| lazyLoadTriggerMethods | 指定哪个对象的方法触发一次延迟加载。 | A method name list separated by commas | equals,clone,hashCode,toString |
| defaultScriptingLanguage | 指定动态 SQL 生成的默认语言。 | A type alias or fully qualified class name. | org.apache.ibatis.scripting.xmltags.XMLLanguageDriver |
| callSettersOnNulls | 指定当结果集中值为 null 的时候是否调用映射对象的 setter（map 对象时为 put）方法，这对于有 Map.keySet() 依赖或 null 值初始化的时候是有用的。注意基本类型（int、boolean等）是不能设置成 null 的。 | true | false | FALSE |
| returnInstanceForEmptyRow | 当返回行的所有列都是空时，MyBatis默认返回null。 当开启这个设置时，MyBatis会返回一个空实例。 请注意，它也适用于嵌套的结果集 (i.e. collectioin and association)。（从3.4.2开始） | true | false | FALSE |
| logPrefix | 指定 MyBatis 增加到日志名称的前缀。 | Any String | Not set |
| logImpl | 指定 MyBatis 所用日志的具体实现，未指定时将自动查找。 | SLF4J | LOG4J | LOG4J2 | JDK\_LOGGING | COMMONS\_LOGGING | STDOUT\_LOGGING | NO\_LOGGING | Not set |
| proxyFactory | 指定 Mybatis 创建具有延迟加载能力的对象所用到的代理工具。 | CGLIB | JAVASSIST | JAVASSIST (MyBatis 3.3 or above) |
| vfsImpl | 指定VFS的实现 | 自定义VFS的实现的类全限定名，以逗号分隔。 | Not set |
| useActualParamName | 允许使用方法签名中的名称作为语句参数名称。 为了使用该特性，你的工程必须采用Java 8编译，并且加上-parameters选项。（从3.4.1开始） | true | false | TRUE |
| configurationFactory | Specifies the class that provides an instance of Configuration. The returned Configuration instance is used to load lazy properties of deserialized objects. This class must have a method with a signature static Configuration getConfiguration(). (Since: 3.2.3) | A type alias or fully qualified class name. | Not set |

### <typeAliases>元素（使用时注意使用事项）

* 存在别名重复的问题，如果使用这个标签设置，代码真心不好维护，建议少用。
* 使用自动扫描包的形式定义别名，代码会自动扫描po包中的POJO类以首字母小写的非限定类名来作为它的别名，例如com.itheima.pl.Customer的别名为customer等。如下面代码：  
  <**typeAliases**>  
   <**package name="com.mybatis.po"** />  
  </**typeAliases**>
* 若不同的包里面类名出现重复，则可以通过配置文件配置指定、TypeAliasRegistry的registerAlias方法注册或者注解的方式解决：  
  **配置文件指定：**  
  <typeAlias alias="user" type="com.daily.pojo.User"></typeAlias>  
  <typeAlias alias="product" type="com.daily.pojo.Product"></typeAlias>  
  **TypeAliasRegistry的registerAlias方法注册：**  
  SqlSession sqlSession = getSqlSession();  
  TypeAliasRegistry typeAliasRegistry = sqlSession.getConfiguration().getTypeAliasRegistry();  
  //代码方式配置别名  
  typeAliasRegistry.registerAlias("user1",com.daily.pojo2.User.class);  
  **注解：**@Alias("user1")  
  public class User {}
* 除了可配置的别名，Mybatis本身还带了许多基本类型的别名，如下表所示：

|  |  |
| --- | --- |
| 别名 | 映射的类型 |
| \_byte | byte |
| \_long | long |
| \_short | short |
| \_int | int |
| \_integer | int |
| \_double | double |
| \_float | float |
| \_boolean | boolean |
| string | String |
| byte | Byte |
| long | Long |
| short | Short |
| int | Integer |
| integer | Integer |
| double | Double |
| float | Float |
| boolean | Boolean |
| date | Date |
| decimal | BigDecimal |
| bigdecimal | BigDecimal |

### <typeHandler>元素（有需要的时候可以使用）

* MyBatis在预处理语句(PreparedStatement)中设置一个参数或者从结果集(ResultSet)中取出一个值时，都会用其框架内部注册了的typeHandler(类型处理器)进行相关处理。
* typeHandler的作用就是将预处理语句中传入的参数从javaType(Java类型)转换为jdbcType(JDBC类型)，或者从数据库取出结果时将jdbcType转换为javaType。
* 常用的类型处理器如下表：

|  |  |
| --- | --- |
| 类型处理器 | 说明 |
| BooleanTypeHandler | 用于 java 类型 boolean，jdbc 类型 bit、boolean |
| ByteTypeHandler | 用于 java 类型 byte，jdbc 类型 TINYINT |
| ShortTypeHandler | 用于 java 类型 short，jdbc 类型 SMALLINT |
| IntegerTypeHandler | 用于 INTEGER 类型 |
| LongTypeHandler | 用于 long 类型 |
| FloatTypeHandler | 用于 FLOAT 类型 |
| DoubleTypeHandler | 用于 double 类型 |
| StringTypeHandler | 用于 java 类型 string，jdbc 类型 CHAR、VARCHAR |
| ArrayTypeHandler | 用于 jdbc 类型 ARRAY |
| BigDecimalTypeHandler | 用于 java 类型 BigDecimal，jdbc 类型 REAL、DECIMAL、NUMERIC |
| DateTypeHandler | 用于 java 类型 Date，jdbc 类型 TIMESTAMP |
| DateOnlyTypeHandler | 用于 java 类型 Date，jdbc 类型 DATE |
| TimeOnlyTypeHandler | 用于 java 类型 Date，jdbc 类型 TIME |

* 自定义类型处理器的使用，需要继承BaseTypeHandler类，并配合配置使用。

### <environments>元素（很重要，不过现在一般都用不着）

* Environments主要用于配置环境信息，包括事务，数据源等。
* MyBatis的事务管理分两种，一种是JDBC，另外一种是MANAGED。  
  JDBC直接使用JDBC的提交和回滚设置，它依赖于从数据源得到的连接来管理事务的作用域。  
  MANAGED从来不提交或回滚一个连接，而是让容器（例如Tomcat）来管理事务的整个生命周期。
* MyBatis的数据源配置分为3种，分别是UNPOOLED、POOLED和JNDI  
  UNPOOLED：最简单的连接，只需要配置driver、url、username、password以及defaultTransactionIsolationLevel（默认的连接事务隔离级别，可以不配置使用默认）即可  
  POOLED：此数据源利用“池”的给你按将JDBC连接对象组织起来，避免了再创建新的连接实例时所需要初始化和认证的时间。除了UNPOOLED的五个属性，POOLED还有其他类似于数据库连接池的额外属性可以配置。
* JNDI：此数据源可以在EJB或应用服务器等容器中使用。容器可以集中或在外部配置数据源，然后放置一个JNDI上下文的应用。

### <mappers>元素（很重要）

mappers元素的实际上就是配置映射文件，从而使MyBatis可以读取具体的sql配置。

* 使用类路径配置  
  <mappers>  
   <mapper resource=”com/itheima/mapper/UserMapper.xml”>  
  </mappers>
* 使用本地文件路径引入  
  <mappers>  
   <mapper url=”file:///D:/com/itheima/mapper/UserMapper.xml”>  
  </mappers>
* 使用接口类引入  
  <mappers>  
   <mapper class=”com.itheima.mapper.UserMapper”>  
  </mappers>
* 使用包名引入  
  <mappers>  
   <package name=”com.itheima.mapper”>  
  </mappers>

# 映射文件

映射文件中包括了select、insert、update、delete、sql、cache、cache-ref和resultMap元素。

每个元素都包含了多个属性，这里不一一描述，详细见如下链接：

https://blog.csdn.net/qq\_27008807/article/details/78975957

## <select>元素

<select>元素标签的常用属性如下：

|  |  |
| --- | --- |
| 属性 | 说明 |
| id | 表示命名空间中的唯一标识符，常与命名空间组合起来使用。组合后如果不唯一，MyBatis会抛出异常。 |
| parameterType | 该属性表示传入SQL语句的参数类的全限定名或者别名。它是一个可选属性，因为MyBatis可以通过TypeHandler推断出具体传入语句的参数。其默认值是unset（依赖于驱动）。如果参数类型parameterType为String类型，则变量名必须为value或者\_parameter。 |
| resultType | 从SQL语句中返回的类型的类的全限定名或者别名。如果是集合类型，那么返回的应该是集合可以包含的类型，而不是集合的本身。返回时可以使用resultType或resultMap之一。 |
| resultMap | 表示外部resultMap的命名引用。返回时可以使用resultType或resultMap之一。 |
| flushCache | 表示在调用SQL语句之后，是否需要MyBatis清空之前查询的本地缓存和二级魂村。其值为布尔类型（true|false），默认值为false。如果设置为true，则任何时候只要SQL语句被调用，都会清空本地缓存和二级缓存。 |
| useCache | 用于控制二级缓存的开启和关闭。其值为布尔类型（true|false），默认值为true，表示将查询结果存入二级缓存中。 |
| timeout | 用于设置超时参数，单位为秒。超时时将抛出异常。 |
| fetchSize | 获取记录的总条数设定，其默认值是unset（依赖于驱动） |
| statementType | 用于设置MyBatis使用哪个JDBC的Statement工作，其值为STATEMENT、PREPARED（默认值）或CALLABLE，分别对应JDBC中的Statement、PreparedStatement和CallableStatement。 |
| resultSetType | 表示结果集的类型，其值可设置为FORWARD\_ONLY、SCROLL\_SENSITIVE或SCROLL\_INSENSITIVE，它的默认值是unset（依赖于驱动） |

## <insert><update><delete>元素

<insert><update><delete>元素标签的常用属性与<select>元素标签的常用属性大致相同，但有三个特有属性不一样，如下：

|  |  |
| --- | --- |
| 属性 | 说明 |
| keyProperty | （仅对insert和update有用）此属性的作用是将插入或更新操作时的返回值赋值给PO类的某个属性，通常会设置为主键对应的属性。如果需要设置联合主键，可以在多个值之间用逗号隔开。 |
| keyColumn | （仅对insert和update有用）此属性用于设置第几列是主键，当主键列不是表中的第一列时需要设置。在需要主键联合时，值可以用逗号分开。 |
| useGeneratedKeys | （仅对insert和update有用）此属性会使MyBatis使用JDBC的getGeneratedKeys()方法来获取由数据库内部生产的主键，如MySQL和SQL Server等自动递增的字段，其默认值为false。但是有些数据库不支持自动增长，如oracle。 |

## <sql><property>元素

<sql>元素是用于封装公用的SQL语句。<select><insert><update><delete>可以通过在SQL语句中加入<include refied=”sqlId”>的方式替换。

<property>元素用于替换<sql>元素里面的${变量名}变量。一般可以在<sql>标签的语句加入变量，<select><insert><update><delete>标签在调用<sql>标签的时候，在<incluce>标签里面使用<property>标签进行变量替换即可。

## <resultMap>元素

<resultMap>元素是MyBatis最强大的元素，它可以将查询到的复杂数据（比如查询到几个表中的数据）映射到一个结果集中。

如果一个实体里面存在Map或者List的属性，用<resultMap>元素可以实现。

<!--column不做限制，可以为任意表的字段，而property须为type 定义的pojo属性-->

<resultMap id="唯一的标识" type="映射的pojo对象">

<id column="表的主键字段，或者可以为查询语句中的别名字段" jdbcType="字段类型" property="映射pojo对象的主键属性" />

<result column="表的一个字段（可以为任意表的一个字段）" jdbcType="字段类型" property="映射到pojo对象的一个属性（须为type定义的pojo对象中的一个属性）"/>

<association property="pojo的一个对象属性" javaType="pojo关联的pojo对象">

<id column="关联pojo对象对应表的主键字段" jdbcType="字段类型" property="关联pojo对象的主席属性"/>

<result column="任意表的字段" jdbcType="字段类型" property="关联pojo对象的属性"/>

</association>

<!-- 集合中的property须为oftype定义的pojo对象的属性-->

<collection property="pojo的集合属性" ofType="集合中的pojo对象">

<id column="集合中pojo对象对应的表的主键字段" jdbcType="字段类型" property="集合中pojo对象的主键属性" />

<result column="可以为任意表的字段" jdbcType="字段类型" property="集合中的pojo对象的属性" />

</collection>

</resultMap>

# MyBatis的缓存

详细介绍可以参考如下链接：

<https://www.jianshu.com/p/c553169c5921>

## 一级缓存

### 概述

* Mybatis一级缓存的生命周期和SqlSession一致。
* Mybatis的缓存是一个粗粒度的缓存，没有更新缓存和缓存过期的概念，同时只是使用了默认的hashmap，也没有做容量上的限定。
* Mybatis的一级缓存最大范围是SqlSession内部，有多个SqlSession或者分布式的环境下，有操作数据库写的话，会引起脏数据，建议是把一级缓存的默认级别设定为Statement，即不使用一级缓存。
* MyBatis一级缓存默认使用SESSION模式，所以一般开发的时候要把配置进行修改：  
  <**setting name="localCacheScope" value="STATEMENT"**/>

## 二级缓存

### 概述

* 二级缓存是多个SqlSession间共享的
* 二级缓存不适用于多表查询，因为二级缓存基于map配置文件中的cache-ref的命名空间进行缓存，不同命名空间的查询会出现脏读的情况。
* MyBatis必须commit后才能刷新缓存。
* MyBatis二级缓存的配置默认是true，开发的时候需要把配置进行修改：  
  <**setting name="cacheEnabled" value="false"**/>

# 学习过程中踩过的坑

1. Mybatis-config.xml里面，environments中可以有多个environment，每个environment都有自己的id，而environments的default属性用于选择用哪一个environment。用处是可以通过简单的配置切换开发、测试、UAT以及生产环境。
2. Mapper.xml中的SQL配置，如果参数类型parameterType为String类型，则变量名必须为value或者\_parameter，否则会报异常“Cause: org.apache.ibatis.reflection.ReflectionException: There is no getter for property named 'username' in 'class java.lang.String'”
3. Mapper.xml中的SQL配置，如果是模糊查询（即使用like），参数不能只用’%#{value}%’，会报错。网上很多方法，例如’%${value}%’这种非预编译的方式，但是这样不能解决SQL注入的问题；又例如CONCAT('%',#{name},'%')、'%'+#{name}+'%'或者'%'||#{name}||'%'，这种又会存在不同数据库存在不同语法的问题。比较好的方式是使用bind标签。Bind标签同样需要使用value或者\_parameter去支撑String类型的参数，SQL语句引用Bind标签的时候不能使用单引号。
4. 针对增删改的SQL，mybatis在执行完的时候基本都会返回变更的条数，如果SQL存在语法等问题导致无法执行，一般都会通过报异常的方式报错，而不会返回数值。
5. 针对增删改的SQL，经过测试，如果sqlSession不主动commit数据，SQL执行完毕的事务是会自动回退的
6. 如果mybatis-config.xml中的元素不按照顺序执行，那么会出现如下报错：  
   Cause: org.apache.ibatis.builder.BuilderException: Error creating document instance. Cause: org.xml.sax.SAXParseException: The content of element type "configuration" must match "(properties?,settings?,typeAliases?,typeHandlers?,objectFactory?,objectWrapperFactory?,plugins?,environments?,databaseIdProvider?,mappers?)".