# 前言

本次源码阅读基于mybatis-3.4.5。

# 概述

本文档主要记录自己在学习Mybatis过程中的随笔，很多东西都是断断续续记录的，连贯性不是很强。



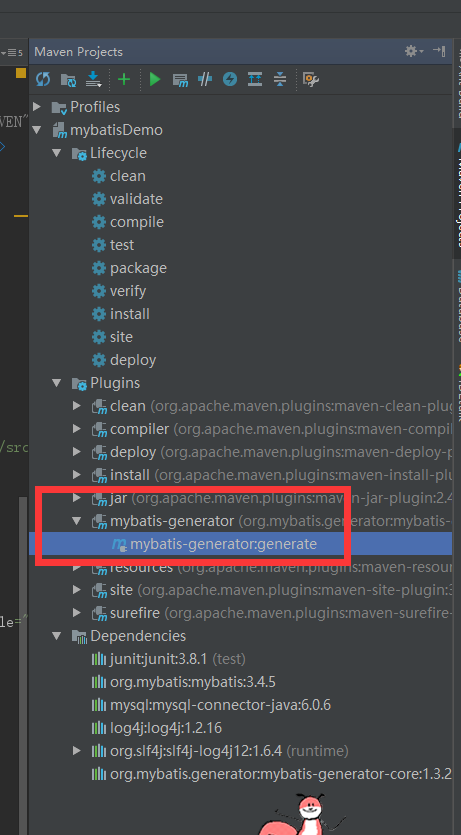
Figure 1 mybatis架构示意图

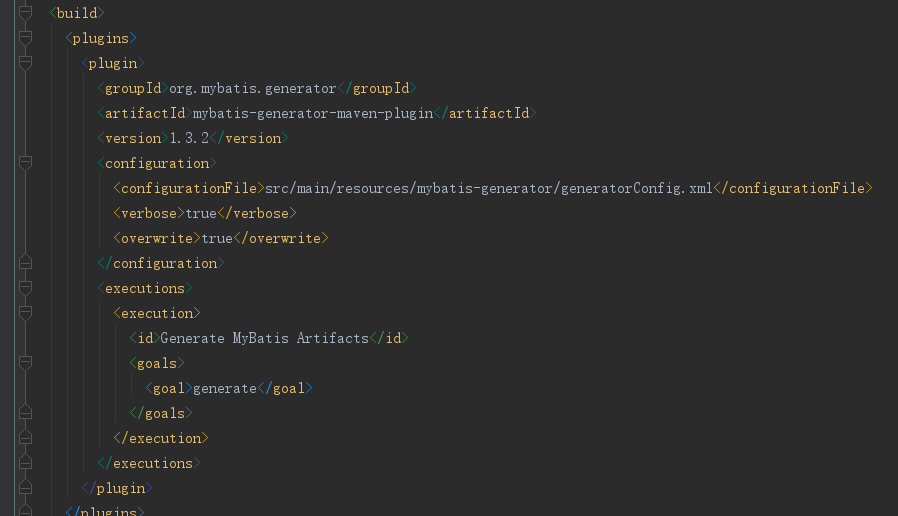
# 代码生成器

参考MybatisDemo项目中generationConfig.xml文件。

需要依赖的包：

<dependency>  
 <groupId>org.mybatis.generator</groupId>  
 <artifactId>mybatis-generator-core</artifactId>  
 <version>1.3.2</version>  
</dependency>





# 分层结构

## 配置文件

参考<https://www.jianshu.com/p/82f0875ac22f>

配置文件是mybatis的入口，其中包含很多可配置项，各个配置项含义参考<http://www.mybatis.org/mybatis-3/configuration.html>，翻译后的网站

<http://www.mybatis.org/mybatis-3/zh/configuration.html>，对于每个配置项。mybatis读取配置文件示意图

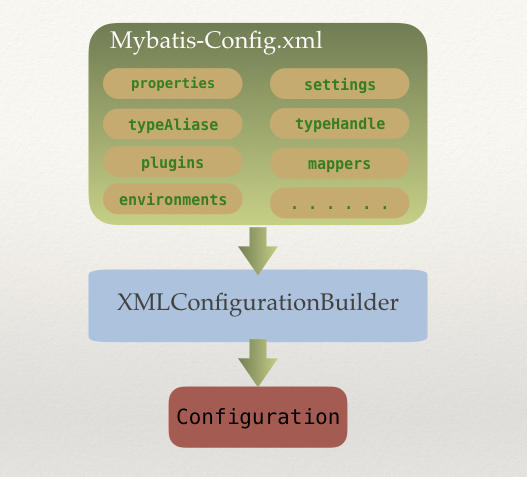


Figure 2 配置文件读取示意图

需要注意的事项包括如下：

### properties

设置Property包括三种方式：

* 通过property文件设置，然后将文件路径赋值给properties的resource属性；
* 通过property属性逐个设置；
* 代码调用mybatis的api时提供的参数；

比如

<properties resource="org/mybatis/example/config.properties">

<property name="username" value="dev\_user"/>

<property name="password" value="F2Fa3!33TYyg"/>

</properties>

注意，如果在编写配置文件时，一个属性不止一次被配置时，mybatis按照如下顺序进行加载：

* 在 properties 元素体内指定的属性首先被读取。
* 然后根据 properties 元素中的 resource 属性读取类路径下属性文件或根据 url 属性指定的路径读取属性文件，并覆盖已读取的同名属性。
* 最后读取作为方法参数传递的属性，并覆盖已读取的同名属性。

最先被加载的项，可能会被后续操作给覆盖，因此，通过方法参数传递的属性具有最高优先级，resource/url 属性中指定的配置文件次之，最低优先级的是 properties 属性中指定的属性。比如：

配置文件中设置了username，但是在代码中创建连接时传递了新的username，最后将以方法的参数值为准。

### settings

这一个标签用途很大，包含丰富的配置项，下面是一个示例，具体的使用参考官网

<settings>

<setting name="cacheEnabled" value="true"/>

<setting name="lazyLoadingEnabled" value="true"/>

<setting name="multipleResultSetsEnabled" value="true"/>

<setting name="useColumnLabel" value="true"/>

<setting name="useGeneratedKeys" value="false"/>

<setting name="autoMappingBehavior" value="PARTIAL"/>

<setting name="autoMappingUnknownColumnBehavior" value="WARNING"/>

<setting name="defaultExecutorType" value="SIMPLE"/>

<setting name="defaultStatementTimeout" value="25"/>

<setting name="defaultFetchSize" value="100"/>

<setting name="safeRowBoundsEnabled" value="false"/>

<setting name="mapUnderscoreToCamelCase" value="false"/>

<setting name="localCacheScope" value="SESSION"/>

<setting name="jdbcTypeForNull" value="OTHER"/>

<setting name="lazyLoadTriggerMethods" value="equals,clone,hashCode,toString"/>

</settings>

### typeAliases

顾名思义，为类型设置别名，举例：

<!-- 取别名 -->

<typeAliases>

<typeAlias type="com.lc.model.User" alias="user"/>

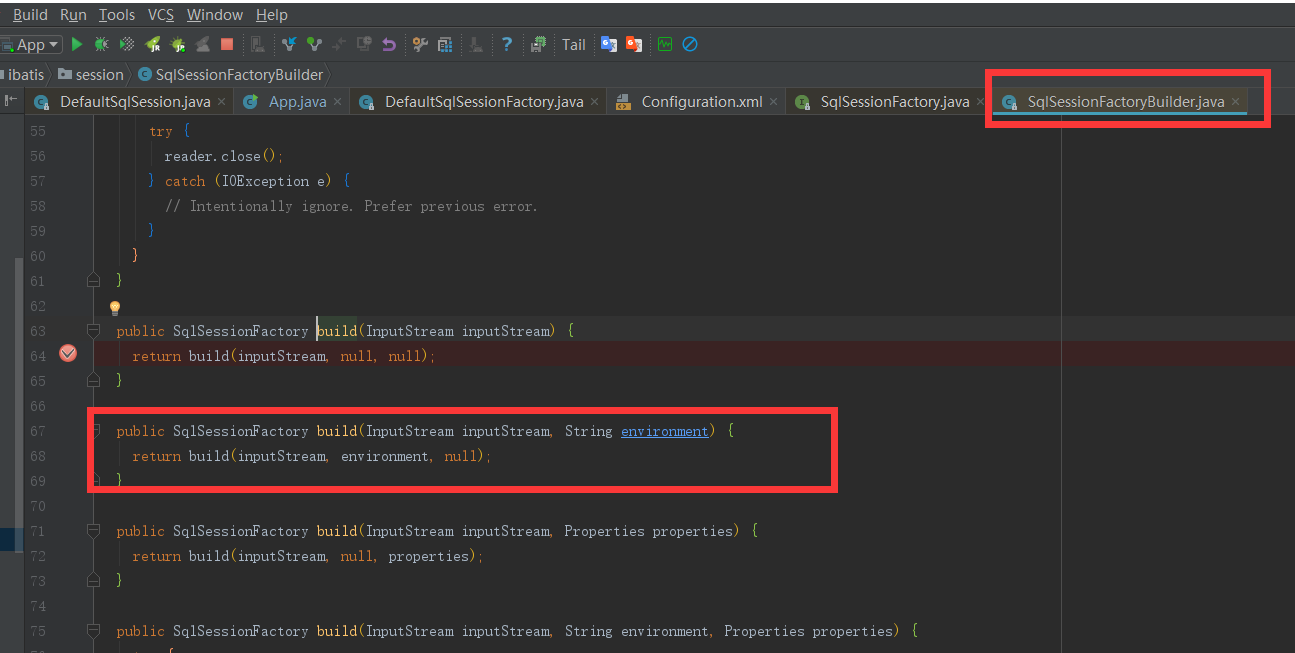
</typeAliases>

### environments

用于配置多个数据源，比如：

<environments default="HO"*>  
<*environment id="HD"*>  
 <*transactionManager type="JDBC"*/>  
 <*dataSource type="POOLED"*>  
 <*property name="driver" value="${hd.jdbc.driverClassName}"*/>  
 <*property name="url" value="${hd.jdbc.url}"*/>  
 <*property name="username" value="${hd.jdbc.username}"*/>  
 <*property name="password" value="${hd.jdbc.password}"*/>  
 </*dataSource*>  
</*environment*>  
<*environment id="HO"*>  
 <*transactionManager type="JDBC"*/>  
 <*dataSource type="POOLED"*>  
 <*property name="driver" value="${ho.jdbc.driverClassName}"*/>  
 <*property name="url" value="${ho.jdbc.url}"*/>  
 <*property name="username" value="${ho.jdbc.username}"*/>  
 <*property name="password" value="${ho.jdbc.password}"*/>  
 </*dataSource*>  
</*environment*>  
</*environments*>*

指定不同数据源的接口：



## 接口层

### 概述

* 掌握Mybatis使用的两种方式：
  + 一种是传统API方式，通过传递StatementId和Arguments访问mapper中的sql，参考App中的测试代码；
  + 一种是使用Mapper接口，这种方式不需要实现Mapper接口，但是方法名必须和Mapper XML中StatementId保持一致；

注意：通过Mapper方式访问数据库的底层仍然会调用sqlsession提供的CRUD接口，因此重点理解通过API方式的使用，然后理解Mapper接口方式如何转换为调用Api方式即可。

* 使用的基本步骤：
  + 构造Mybatis的构造文件，包括db.driver，username，password等等；
  + 创建SqlSessionFactory
  + 从SqlSessionFactory中获取一条session
  + 如果是Mapper接口方式，通过session.getMapper(mapper接口名)获取一个Mapper，比如UserMapper userMapper = session.getMapper(UserMapper.class);这种方式也可以通过注解的方式表明sql语句
  + 如果是传统API方式+StatementId，直接利用session.select|insert|update|delete（key，parameter）,其中key是Mapper XML文件中的StatementId，比如：userList = session.selectList("userMapper.getUserByParam",params);



Figure 3 API模式



Figure 4 Mapper接口模式

参考代码

MybatisDemo项目

### 接口

SqlSession提供的接口目录如下，提供数据库基本的CRUD操作：



Figure 5 SqlSession接口目录

CRUD操作也按照读写操作来划分，CUD操作都可以被认作为写操作，R操作为读操作，因此在Mybatis的Executor中，只提供了Update和Query操作，Update操作即对应到写操作，Query对应读操作。

#### C操作

C操作，即Create，Mybatis提供了两个Insert接口



Figure 6 C操作接口

查看Mybatis源码，可以得知Insert操作被转换为了Update操作，至于Update如何执行Insert的效果，详见3.2.2.3：



Figure 7 C操作代码

#### R操作

#### U操作

U操作，即Update操作，需要重点掌握，因为C操作、D操作都转换为U操作来执行相应的SQL。

#### D操作

D操作，即Delete操作，Mybatis源码提供的delete接口，最终也转换为Update操作：



Figure 8 U操作源码

### Mapper接口

## 数据处理层

## 引导层

# 组件

## Configuration

Configuration作为配置文件中所有信息的容器。

Configuration对象用于存放mybatis中配置文件中的所有信息，应用程序在加载完xml配置文件后会生成一个SqlSessionFactory，其中就包含一个Configuration对象。Configuration的结构与XML配置文件的结构几乎完全类似。

Configuration类中包含多个属性，重点理解如下几个：

* MappedStatement

## MappedStatement

Mapper.xml文件中所有的sql语句都会被映射为一个MappedStatement对象，一个MappedStatement对应两个key，一个是通过权限的名作为key，一个是通过sql的id作为key存放在一个map中，比如如下语句：



读入Configuration类中保存如下



注意，如果不同Mapper文件中定义的statement的id一样，在解析Mapper的时候会被覆盖。每一个MapperStatement读入到Configuration的mappstatement时，会创建两个key-value，其中一个key是通过全限定名来指定，一个只通过statement的id来指定，比如如下：

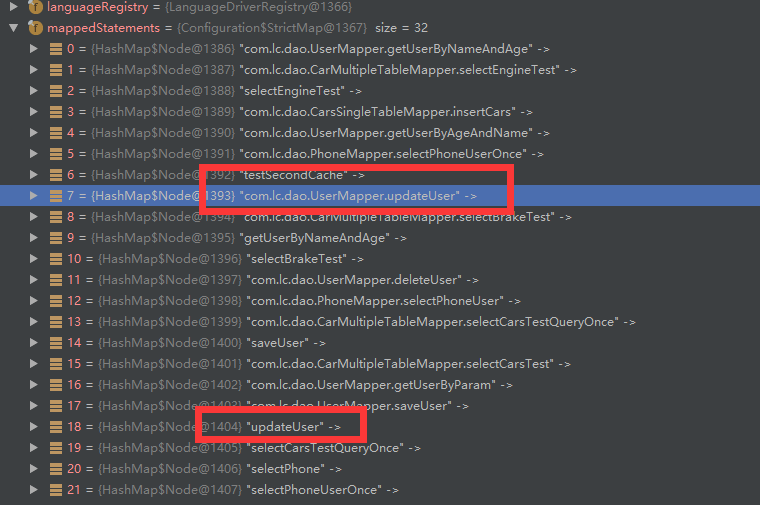


Figure 9 mappedstatement示例

## SqlSession

SqlSession接口、层级关系如下所示：

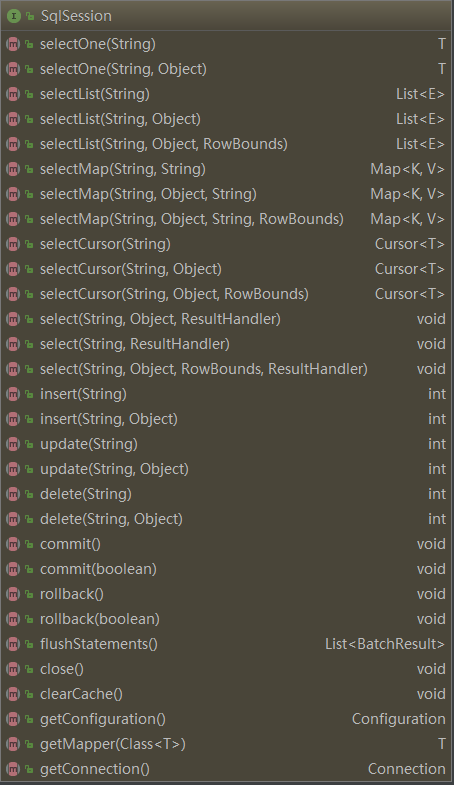


Figure 10 SqlSession接口

实现SqlSession接口的Class：

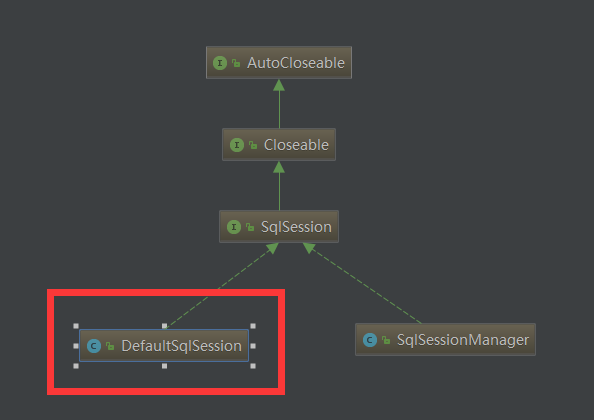


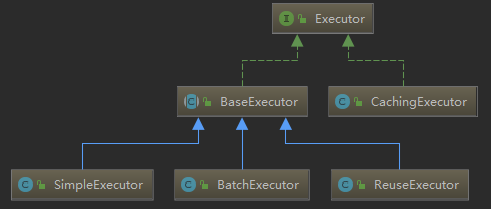
Figure 11 SqlSession的实现类

SqlSession作为应用程序与mybatis之间的中介对象，提供对数据库进行CRUD操作的接口，其中接口包括方法接口，另外一种就是Mapper接口。

DefaultSqlSession属性：

* executor：每一个DefaultSqlSession中都包含一个Executor执行器的代理对象。Sqlsession的所有操作都委托给executor执行。
* autoCommit：提交标志autoCommit=true表示事务执行完之后自动提交到数据库，否则需要应用端调用commit接口；
* configuration：配置信息被映射为的内存对象

## Executor(通过配置决定使用哪个executor)



配置文件中配置方式：

<setting name="defaultExecutorType" value="SIMPLE"/>

执行器，用于执行所有业务端调用的数据库操作SQL。Mybatis提供多种执行器并且可配置，通过配置文件的defaultExecutorType设置执行器类型，默认为SIMPLE类型。Executor执行器的类型包括如下几个：

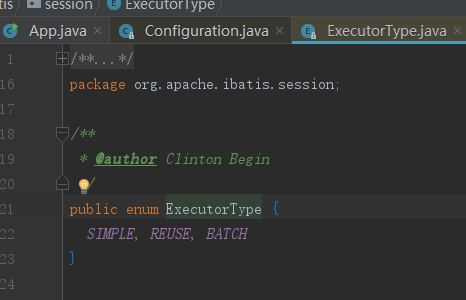


Figure 12 Executor执行器类型

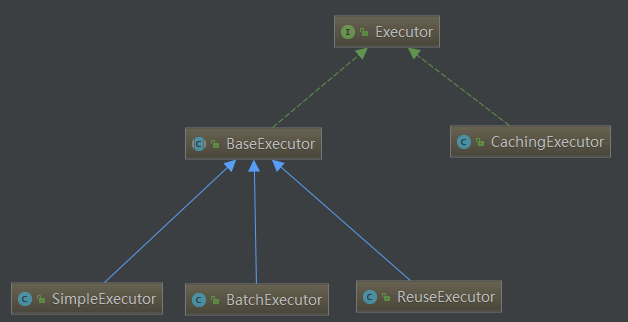


Figure 13 Executor类继承关系

如果mybatis开启了cache将使用CachingExecutor。创建Executor的示例代码如下：

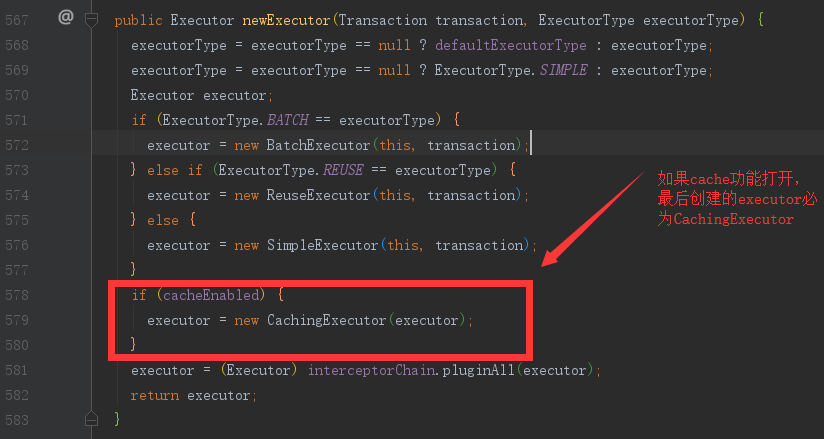


Figure 14 Executor创建示例

### SimpleExecutor

每执行一次update或select，就开启一个Statement对象，用完立刻关闭Statement对象（可以是Statement或PrepareStatement对象）。

### ReuseExecutor

执行update或select，以sql作为key查找Statement对象，存在就使用，不存在就创建，用完后，不关闭Statement对象，而是放置于Map<String, Statement>内，供下一次使用。（可以是Statement或PrepareStatement对象）。

### BatchExecutor

执行update（没有select，JDBC批处理不支持select），将所有sql都添加到批处理中（addBatch()），等待统一执行（executeBatch()），它缓存了多个Statement对象，每个Statement对象都是addBatch()完毕后，等待逐一执行executeBatch()批处理的；BatchExecutor相当于维护了多个桶，每个桶里都装了很多属于自己的SQL，就像苹果蓝里装了很多苹果，番茄蓝里装了很多番茄，最后，再统一倒进仓库。

### CachingExecutor(优先级最高)

默认是开启缓存。

先从缓存中获取查询结果，存在就返回，不存在，再委托给Executor delegate去数据库取，delegate可以是上面任一的SimpleExecutor、ReuseExecutor、BatchExecutor。

# 执行过程

## 初始化

初始化的过程可以抽象为如下：



Figure 15 Mybatis初始化抽象过程

### 加载配置

Mybatis加载配置过程将指定的配置文件读入内存，创建Configuration容器的过程。

Mybatis读取配置有两种方式：

* XML配置文件；
* 代码初始化（不推荐，因为如果修改配置，需要重新编译代码）；

因此本节内容主要分析通过配置文件进行初始化的过程。

读取配置文件有两种方式，一种是读取字符的输入流Reader，一种是读取字节流的InputStream：

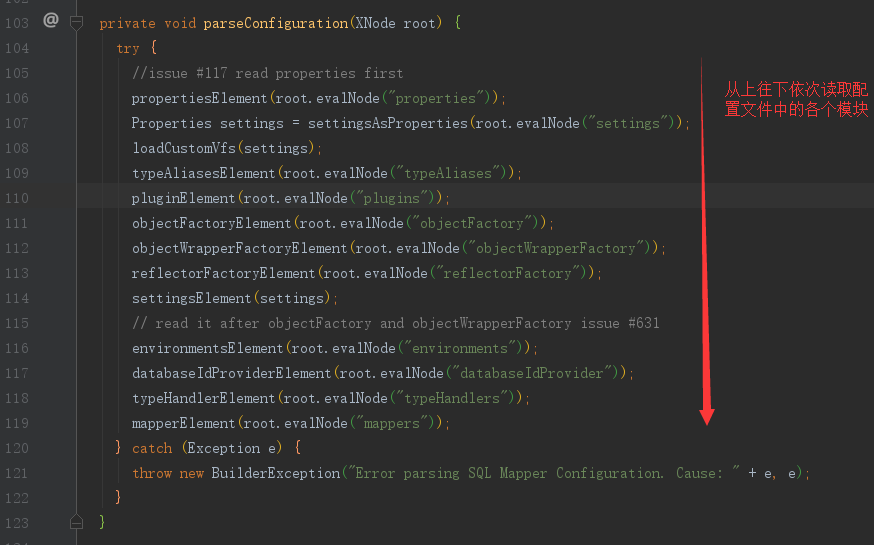
* Reader
* InputStream

初始化阶段的时序图大致如下，其中在创建SqlSessionFactory的过程中采用了Builder设计模式：



Figure 16 时序图

加载配置最重要的方法是XMLConfigBuilder类中定义的解析方法parseConfiguration，其代码如下：



### 创建SqlSessionFactory

SqlSessionFactory的创建通过SqlSessionFactoryBuilder执行，采用Builder模式，在创建SessionFactory时，需要根据配置文件中的Environment来创建连接不同数据源的SessionFactory。配置文件解析过程按照如下顺序进行，需要重点掌握的是mapper的加载。

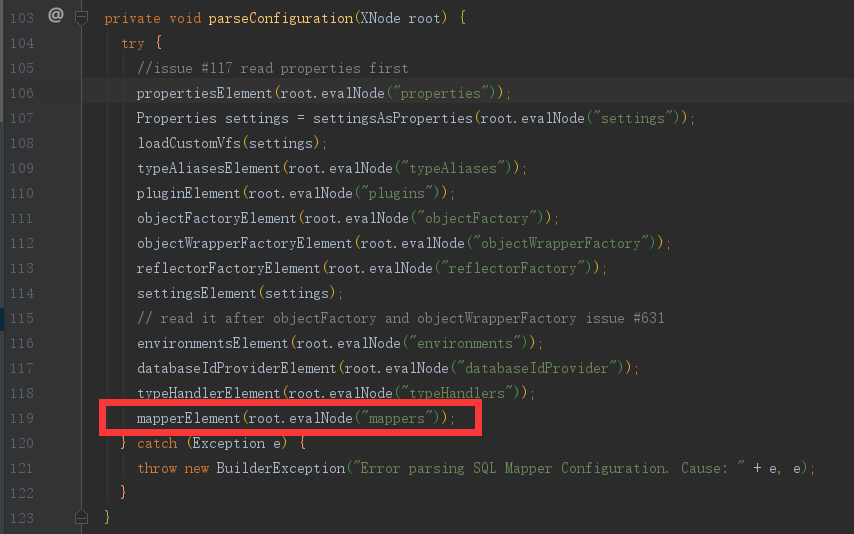


Figure 17 配置文件加载顺序

#### 解析Mapper文件

解析Mapper文件是创建SqlSessionFactory过程的一个子过程（Figure 13中标注的mapperElement过程）。

首先，获取配置文件中的<mappers>标签，并通过mapperElement方法逐个mapper的处理：

mapperElement*(*root.evalNode*(*"mappers"*))*;

mapperElement方法遍历所有mapper，每一个mapper文件都通过一个XMLMapperBuilder进行解析，解析的大致过程是先解析mapper文件中各个节点并保存在configuration对象中，如果该mapper有对应的Mapper接口，再获取对应的接口并保存在mapperRegistry中。

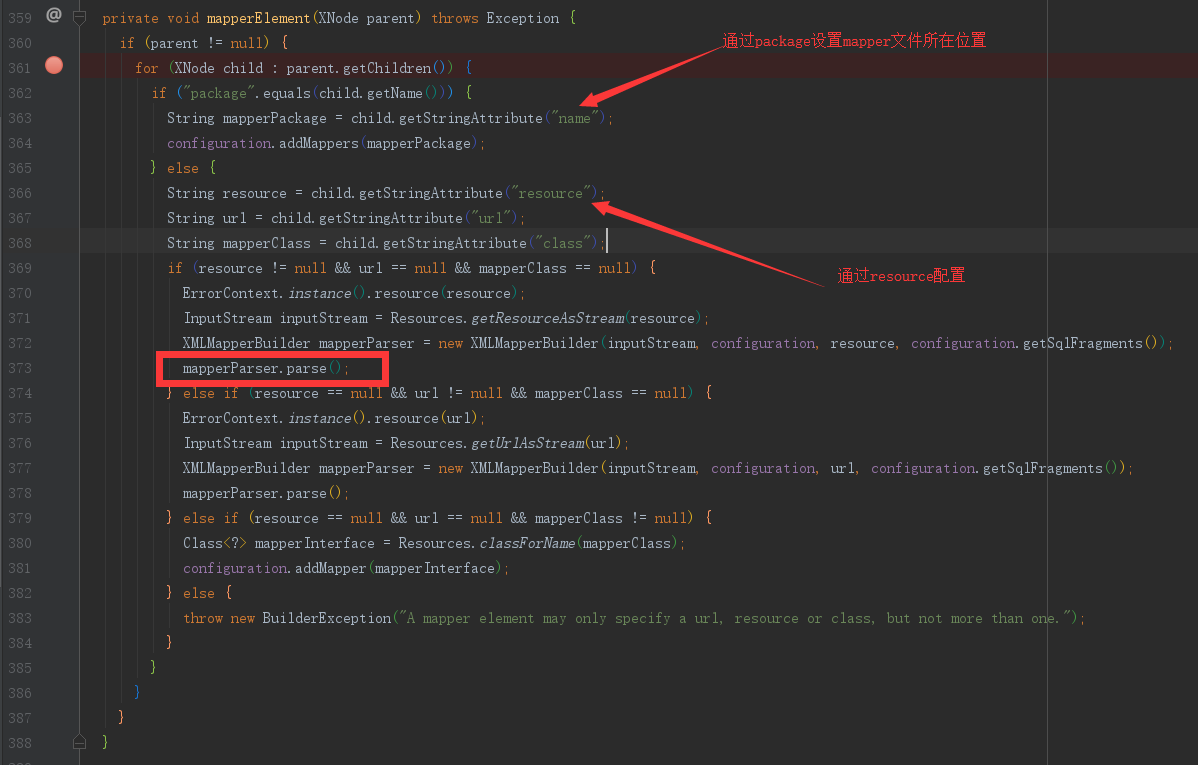


Figure 18 mapperElement方法

在XMLMapperBuilder的parser方法中，需要获取与该Mapper文件对应的Mapper接口（interface），在获取Mapper接口之前需要解析mapper文件中的所有节点：

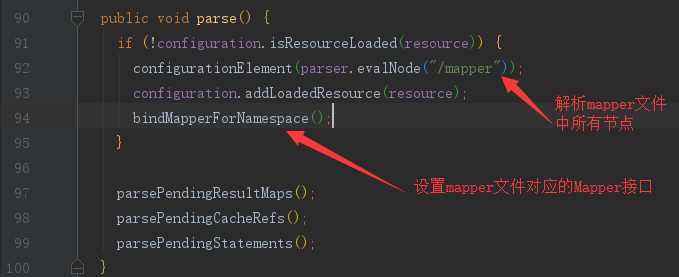


Figure 19 XMLMapperBuilder的parser方法

mapper文件所有节点解析：

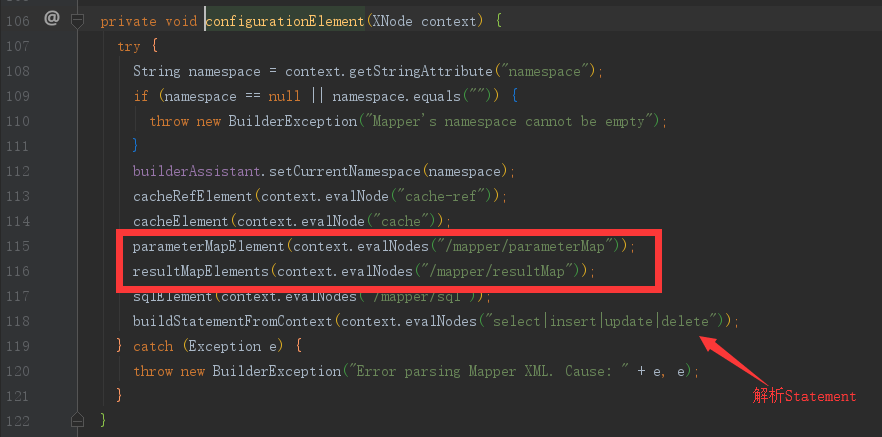


Figure 20 mapper文件节点解析

获取对应Mapper接口：

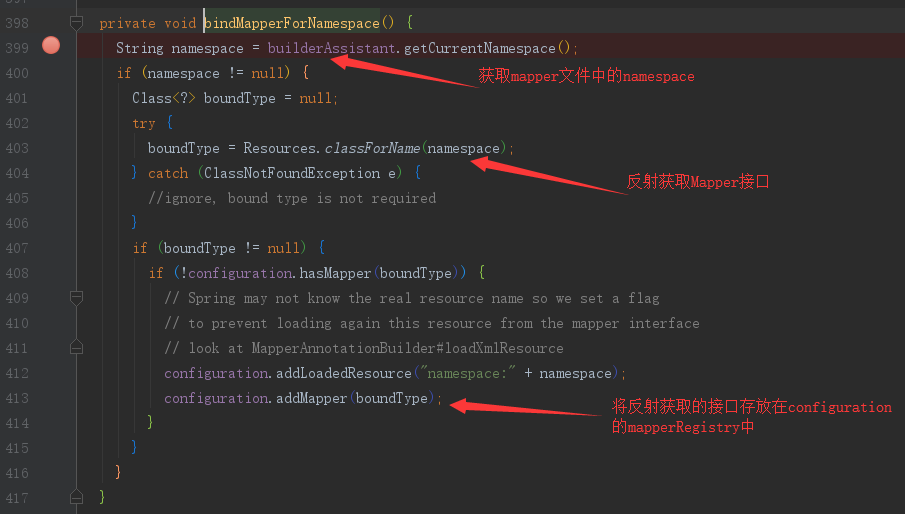


Figure 21 解析mapper文件中的namespace

解析mapper中的namespace时，先获取namespace指定的接口全限定名，并通过反射获取对应的Mapper接口，如果该接口不存在，则直接跳过，因此configuration的mapperRegistry中存放的一定是声明过的mapper接口。

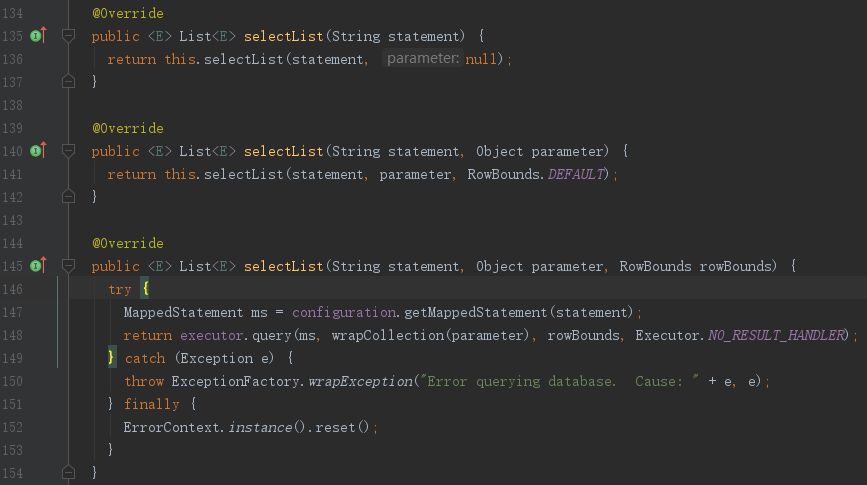
## 使用

### SqlSession获取

使用SqlSession前首先需要从SqlSessionFactory获取一个SqlSession（SqlSession的详细说明参见4.3），而SqlSession的使用方式有两种，一种是采用SqlSession提供的CRUD接口，一种是通过Mapper接口操作。

### SqlSession接口使用

这种方式很简单，通过SqlSessionFactory获取到一个SqlSession后，通过该SqlSession调用Figure 10中的接口方法进行CRUD操作。调用接口方法是只需要传递对应StatementId和查询参数即可。以Select接口为例：



Select接口必要的参数是statement，用于指明对应的SQL语句，接下来的执行动作交个sqlsession中的 executor。

### Mapper接口使用（理解原理）

使用这种方式进行数据库操作时，最终都会转换为sqlsession的接口调用，这个调用主要在MapperMethod的execute方法。

在理解这个原理前，需要理清几个类定义。

#### MapperProxy

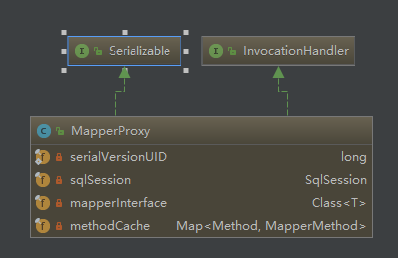


Figure 22 MapperProxy类定义

MapperProxy实现了InvocationHandler接口，包含一个Class<T>，该Class则是用户自定义的Mapper接口；methodCache是一个方法缓存，一开始缓存是空的，以后每次调用都将先判断之前是否调用过该方法缓存起来，无需再去构造。其中methodCache的Key是一个Method，Value是定义的MapperMethod。

#### MapperMethod

Mapper xml文件中各个SQLstatement的抽象。

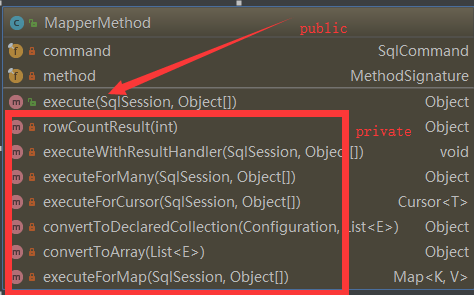


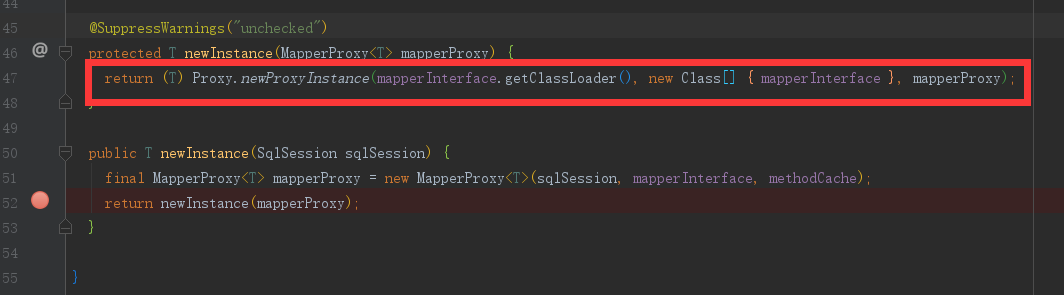
Figure 23 MapperMethod类定义

MapperMethod属性

* SqlCommand类型的command
* MethodSignature类型的method

#### MapperProxyFactory

用于创建MapperProxy的工厂，封装了通过Proxy反射获取MapperProxy的方法。

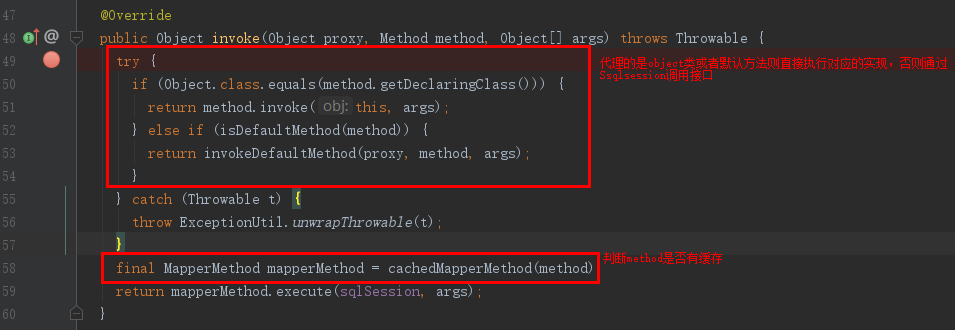


结合代理模式的类图，SqlSession、DefaultSqlSession、MapperProxy对应类图之间的关系可以映射为如下：

* Subject对应SqlSession接口；
* RealSubject对应DefaultSqlSession；
* Proxy对应MapperProxy，MapperProxy中包含的代理对象是DefaultSqlSession的一个实例；

通过Mapper调用自定义接口时：

* 传递参数；
* 调用mapperproxy中实现的invoke方法
* 调用 mappermethod的execute方法，其实质就是通过sqlsession执行对应的SQL并获取结果。



## SqlSession执行SQL

通过getMapper获取对应mapper来调用接口中的业务方法，最终都会执行SqlSession的接口方法。两种使用方式都殊途同归。为了学习SqlSession的执行过程，以SqlSession的Select方法为例浏览源码。

# 缓存

参考<https://tech.meituan.com/mybatis_cache.html>

<https://blog.csdn.net/luanlouis/article/details/41408341>

MybatisDemo中 CacheTest测试用例试验了缓存功能

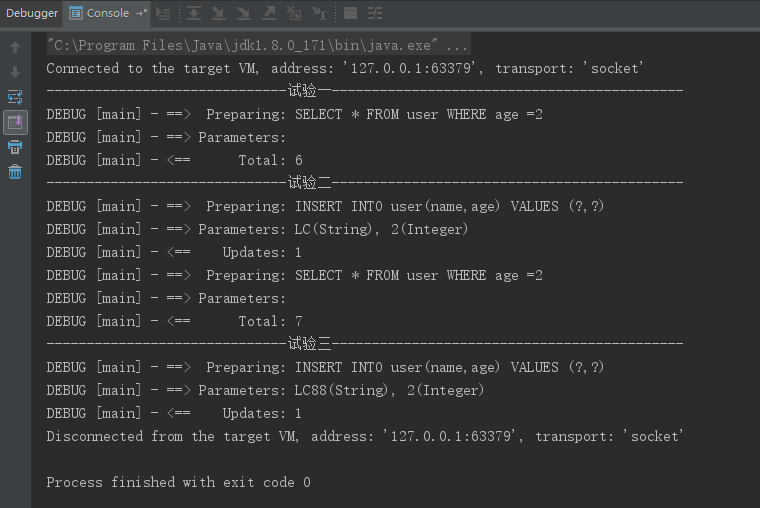


Figure 24 一级缓存试验结果

重点是第三次结果，再次查询获取的是脏数据。

结论：mybatis获取数据的顺序是

二级缓存---🡪一级缓存---🡪数据库

## 一级缓存

启用一级缓存的配置，首先要保证总开关cacheenabled为true，否则缓存功能无法启用。

总开关：<setting name="cacheEnabled" value="true"/>

默认localcachescope为session模式：

<setting name="localCacheScope" value="SESSION"/>

一级缓存是个进程内缓存，每个sqlsession都有一个单独的缓存，一级缓存其实就是一个HashMap结构，只是在mybatis中被封装为PerpetualCache，自底而上的包含关系如下文介绍。

参考：

<https://blog.csdn.net/luanlouis/article/details/41280959>

如果用户配置了"cacheEnabled=true"，那么MyBatis在为SqlSession对象创建Executor对象时，会对Executor对象加上一个装饰者：CachingExecutor

### PerpetualCache类

HashMap的封装类，类图如下

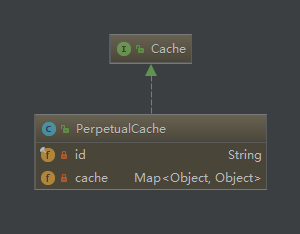


Figure 25 一级缓存类PerpetualCache

其中PerpetualCache的id一般为LocalCache。如图所示是一个本地缓存实例数据：

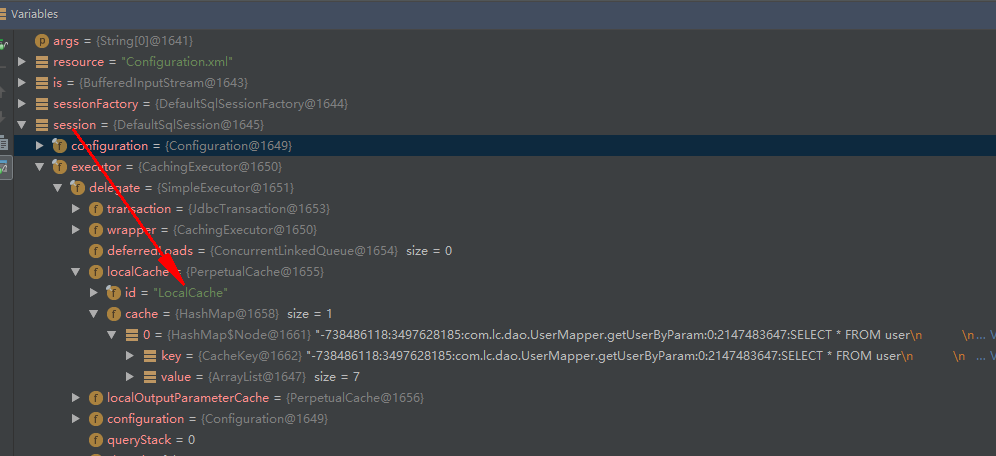


Figure 26 一级缓存缓存数据

到此已经知道一级缓存的结构实质就是HashMap，HashMap结构需要存放的key-value，由此就会对cachekey是如何构造的产生疑问。调试跟踪代码，可以发现一级缓存key的构造规则，构造缓存cachekey需要四个参数

* StatementId
* RowBound(offset,limit分页查询的参数)
* 传递给JDBC的SQL语句java.sql.Preparedstatement（通过boundSQL.getsql获取）
* java.sql.Preparedstatement的参数

注意第四点，所谓的参数是指传递给jdbc所需的参数，不是指用户端传递过来的参数，比如一条SQL语句只需两个参数，但是用户传递了一些无关的参数给mybatis的查询语句，在构造cachekey时，多余的参数会被忽略。

**MyBatis认为的完全相同的查询，不是指使用sqlSession查询时传递给算起来Session的所有参数值完完全全相同，你只要保证statementId，rowBounds,最后生成的SQL语句，以及这个SQL语句所需要的参数完全一致就可以了。**

通过这四个构造一个唯一的cachekey。Cachekey的hashcode计算算法：

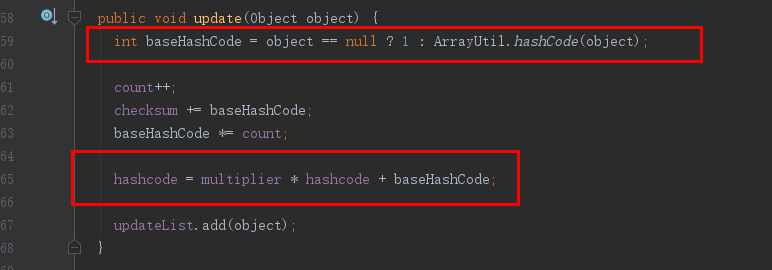


Figure 27 cachekey的hashcode计算算法

### 一级缓存工作流

数据流分两种情况，一种是client查询时，一级缓存中存在缓存结果，一种是缓存中无对应缓存，其时序图分别如下：



Figure 28 缓存命中数据流



Figure 29 缓存失败时序图

### 一级缓存生命期

当创建sqlsession时会创建一个executor，同时executor中 包含一个perpetualcache对象，因此一级缓存在创建sqlsession时就已经创建，销毁时期包含如下几个时刻：

* 当sqlsession对象被回收或者被销毁时，localcache也会被清空；
* 当sqlsession调用close方法时，会将localcache设置为null，因此清空了一级缓存，同时该对象也不存在了；
* 当 sqlsession调用clearcache时，会调用perpetualcache对象的clear操作（其实就是HashMap的clear操作）清空缓存，此时localcache对象还存在；
* 当通过sqlsession执行数据库写操作（包括update、delete、insert）时，会调用perpetualcache对象的clear操作（其实就是HashMap的clear操作）清空缓存，此时localcache对象还存在；

## 二级缓存

启用二级缓存，需要在配置文件中打开总开关。

### 二级缓存划分

* 每个Mapper分配一个cache对象（通过<cache>节点指定，创建cache时通过<Mapper>节点的namespace属性进行指定cache的名字，也便于后续cache-ref节点引用）
* 多个Mapper公用同一个cache对象（通过<cache-ref>指定）

注意，如果在公用同一个cache对象时，必须保证该cache对象在某个 Mapper中已经定义。

Mybatis开启二级缓存后，并不意味着一个Mapper中的所有查询语句都会缓存，必须显式地为查询语句指定使用缓存，通过useCache=”true”进行指定。

因此，一个查询语句启用 二级缓存的条件包括如下：

* 缓存总开关打开cacheenable=true；
* Mapper文件中使用了<cache>或者<cache-ref>节点，比如<cache namespace = “namespace1”，通过cache-ref指定缓存存放的名空间，这样少创建了几个namespace，实际上缓存仍然需要计算cachekey，因为statementid不一样
* Select语句启用缓存，通过usecache=true指定

二级缓存的选择包括如下：

* Mybatis自身提供的缓存实现
* 用户自定义的缓存实现，如果是自定义实现，必须实现mybatis提供的cache接口，并且在Mapper文件中<cache type=”自定义类型”>来指定
* 第三方缓存，比如memcache

### Mybatis内部cache实现

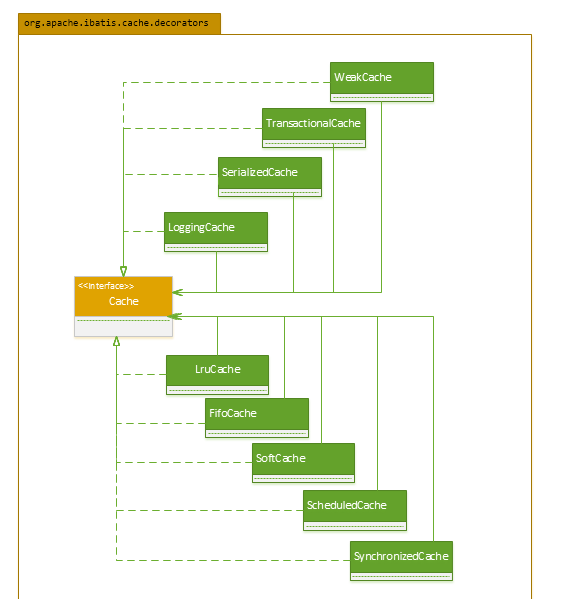


Figure 30 mybatis二级缓存内部实现

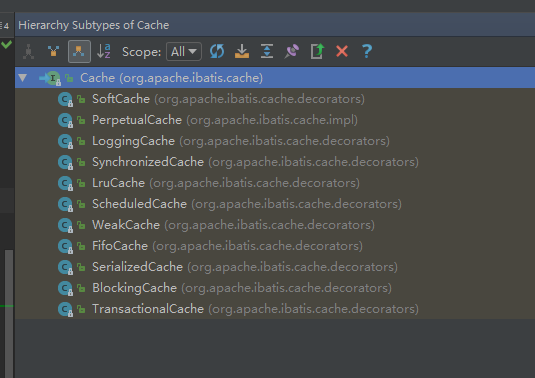


Figure 31 实现cache接口的类

### 二级缓存数据流

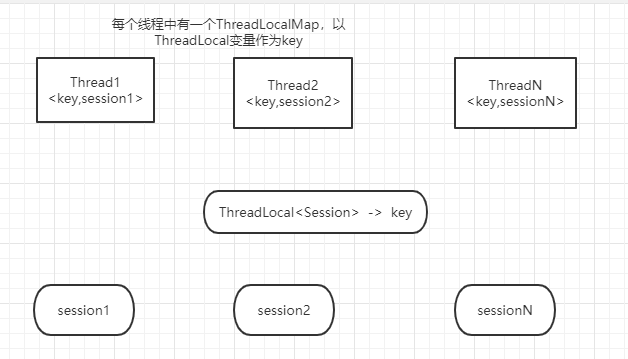
# 事务

# SqlsessionTemplate

线程安全实现原理是动态代理+ThreadLocal<SqlSession>，保证每个线程有独立的Sqlsession实例。动态代理是代理SqlSession接口，重新生成一个代理对象，而不是使用Mybatis提供的DefaultSqlSession类对象。

独立的Sqlsession实例是通过SqlSessionTemplate的内部类SqlSessionInterceptor代理实现的。

每次通过SqlSessionTemplate调用select，insert等方法时，通过代理拦截这些方法调用（实际调用SqlSessionTemplate的内部类SqlSessionInterceptor的invoke方法），这个方法会根据线程获取不同的sqlsession，然后使用完之后会释放该sqlsession。



# 示例

## 关联查询

### 一对一

针对一对一这种关联查询可分为两种情况，一种是数据库一张表格存放所有的字段，然后通过mybatis中的resultMap映射为不同对象a,b，c等，其中一个a对象包含b和c对象。

#### 单表结构

比如mybatisDemo项目中 的CarsTest例子，各个类之间的包含关系如下：



Figure 32 类包含关系

数据库表格设计如下：



Figure 33 Cars表格数据库结构

Mybatis中Mapper文件的映射关系：



Figure 34 Mapper映射关系

#### 多表结构

为每一个子对象设计一个数据库表格，这样便于管理。多表结构又分为一次查询和多次查询数据库。一次查询数据库主要通过SQL语句的join操作，将多个表格关联成一张大表哥，如此一来就和单表结构一样了；多次查询主要通过外键再次查询。

类结构和6.1.1.1中的类结构一样，只是数据库表格需要将原来的表格拆分为三个，分别是car\_test,engine\_test,brake\_tests。采用这种方式必须通过外键将多个表格管理起来，然后在sql语句中通过链接操作（join）管理。

一次查询数据库的Mapper如下：



Figure 35 一次查询数据库



Figure 36 Mapper查询关系

多次查询数据库的Mapper如下，这种方式没有多余的列：



Figure 37 多次查询的Mapper

### 一对多

参考<https://blog.csdn.net/reliveIT/article/details/45366867>

一对多需要涉及多张表格。实现一对多的方式有两种，一种是查询多次数据库，一种是通过数据库SQL的链接操作一把查询，然后再通过mybatis的resultMap映射为java类对象。

类定义：



Figure 38 一对多类定义

#### 第一种方式

对应Mapper文件



Figure 39 多次查询数据库的Mapper

#### 第二种方式

对应Mapper样式



Figure 40 一次查询数据库的Mapper

### 多对多

几乎无此情况，如若出现这种情况，说明数据库设计欠佳。

# 参考

《深入理解mybatis原理》 MyBatis的架构设计以及实例分析

<https://blog.csdn.net/luanlouis/article/details/40422941>

Mybatis示例（包括关联查询）

<https://blog.csdn.net/column/details/mybatis-sample.html>

<https://blog.csdn.net/column/details/23967.html?&page=1>