# 概述

## 架构示意图



Figure 1 mybatis架构示意图

# 代码生成器

参考MybatisDemo项目中generationConfig.xml文件。

# 分层结构

## 配置文件

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，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>

## 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 2 API模式



Figure 3 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 4 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 5 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 6 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 7 U操作源码

## 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



## SqlSession

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。

每一个sqlsession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。

## Executor

执行器，用于执行所有业务端调用的数据库操作SQL

# 执行过程

## 初始化

Mybatis初始化的过程其实就是创建Configuration容器的过程，最终返回的是SqlSessionFactory，其中SqlSessionFactory就包含一个Configuration对象。

Mybatis初始化有两种方式：

* 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 8 时序图

# 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

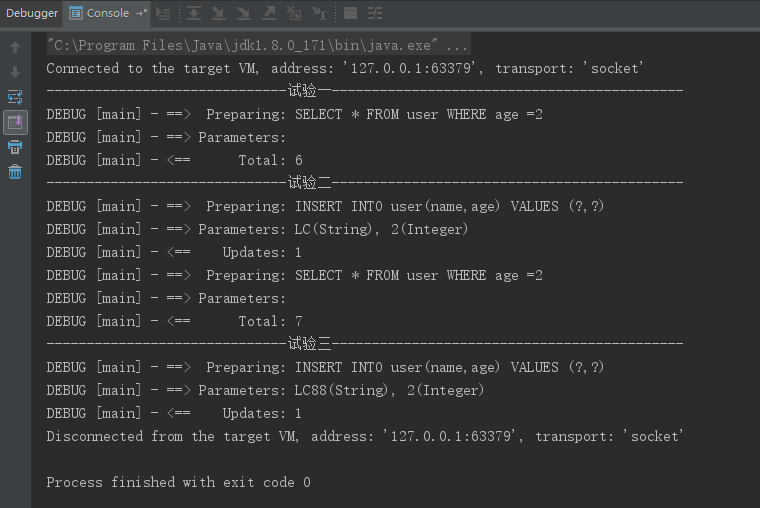


Figure 9 一级缓存试验结果

重点是第三次结果，再次查询获取的是脏数据。

## 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>

### PerpetualCache类

HashMap的封装类，类图如下

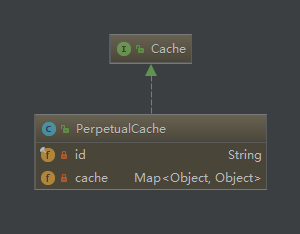


Figure 一级缓存类PerpetualCache

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

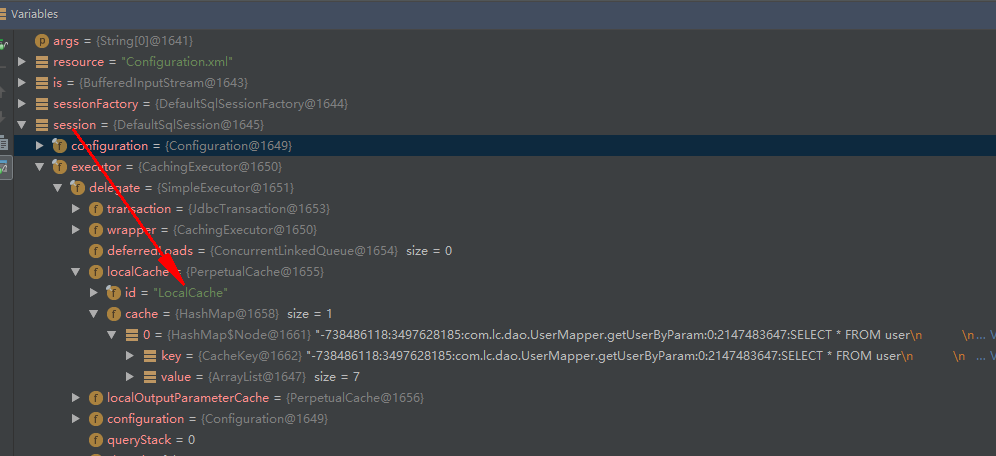


Figure 一级缓存缓存数据

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：

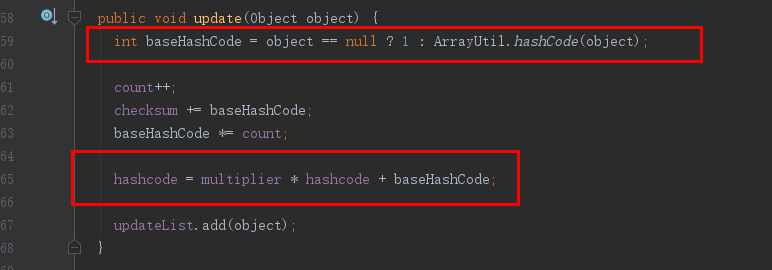


Figure cachekey的hashcode计算算法

### 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 缓存命中数据流



Figure 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

# 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 15 类包含关系

数据库表格设计如下：



Figure 16 Cars表格数据库结构

Mybatis中Mapper文件的映射关系：



Figure 17 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 18 一次查询数据库



Figure 19 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 20 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 21 一对多类定义

#### 第一种方式

对应Mapper文件



Figure 22 多次查询数据库的Mapper

#### 第二种方式

对应Mapper样式



Figure 23 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