Atitit cache缓存分类

二级缓存是 一级 三级缓存

## **mybatis二级缓存的局限性**

细粒度缓存就是，针对某个商品，如果要修改其信息，只修改该商品的缓存数据，缓存区的其他数据不动（不会因为某个商品信息的修改就直接清空整个缓存区）

mybatis二级缓存对细粒度级别的缓存实现不好，比如如下需求：  
对商品信息进行缓存， 由于商品信息查询访问量大，但是要求用户每次都能查询到最新的商品信息，此时如果使用mybatis的二级缓存，就无法实现当一个商品信息变化时，只刷新该商品的缓存信息，而不刷新其它商品的信息。这是因为mybatis的二级缓存区域以mapper为单位进行划分，当一个商品信息变化时（使用这个mapper的任意一个sqlSession执行commt操作），会将所有商品信息的缓存数据全部清空。解决此问题，需要在业务层根据需求对数据进行针对性的缓存（三级缓存）。

Application级别的缓存 session级别 业务缓存

内存 文件缓存

本地 远程分布式缓存

-> 1000 ，最后一次联查实际上查询的是第一次查询结果的缓存，而不是从数据库中查询得到的值，这样就读到了脏数据。

**解决办法**

如果是两个mapper命名空间的话，可以使用 <cache-ref>来把一个命名空间指向另外一个命名空间，从而消除上述的影响，再次执行，就可以查询到正确的数据

**MyBatis缓存**

我们知道，频繁的数据库操作是非常耗费性能的（主要是因为对于DB而言，数据是持久化在磁盘中的，因此查询操作需要通过IO，IO操作速度相比内存操作速度慢了好几个量级），尤其是对于一些相同的查询语句，完全可以把查询结果存储起来，下次查询同样的内容的时候直接从内存中获取数据即可，这样在某些场景下可以大大提升查询效率。

MyBatis的缓存分为两种：

1. 一级缓存，一级缓存是**SqlSession级别**的缓存，对于相同的查询，会从缓存中返回结果而不是查询数据库
2. 二级缓存，二级缓存是**Mapper级别**的缓存，定义在Mapper文件的<cache>标签中并需要开启此缓存，多个Mapper文件可以共用一个缓存，依赖<cache-ref>标签配置

# Cachekey nmsps+sqlid+limit+sql

1 public <E> List<E> query(MappedStatement ms, Object parameter, RowBounds rowBounds, ResultHandler resultHandler) throws SQLException {2 BoundSql boundSql = ms.getBoundSql(parameter);3

CacheKey key = createCacheKey(ms, parameter, rowBounds, boundSql);

到了这里应当一目了然了，MyBastis从四组共五个条件判断两次查询是相同的：

1. **<select>标签所在的Mapper的Namespace+<select>标签的id属性**
2. **RowBounds的offset和limit属性，RowBounds是MyBatis用于处理分页的一个类，offset默认为0，limit默认为Integer.MAX\_VALUE**
3. **<select>标签中定义的sql语句**
4. **输入参数的具体参数值，一个int值就update一个int，一个String值就update一个String，一个List就轮询里面的每个元素进行update**

即只要两次查询满足以上三个条件且没有定义flushCache="true"，那么第二次查询会直接从MyBatis一级缓存PerpetualCache中返回数据，而不会走DB。

is一级缓存来看，它以单纯的HashMap做缓存，没有容量控制，而一次SqlSession中通常来说并不会有大量的查询操作，因此只适用于一次SqlSession，如果用到二级缓存的Mapper级别的场景，有可能缓存数据不断碰到而导致内存溢出。

还有一点，差点忘了写了，<insert>、<delete>、<update>最终都会转换为update方法，看一下BaseExecutor的update方法：

[IMG_256](https://www.cnblogs.com/xrq730/p/javascript:void(0);)

1 public int update(MappedStatement ms, Object parameter) throws SQLException {2 ErrorContext.instance().resource(ms.getResource()).activity("executing an update").object(ms.getId());3 if (closed) {4 throw new ExecutorException("Executor was closed.");5 }6 clearLocalCache();7 return doUpdate(ms, parameter);

第6行clearLocalCache()方法，这意味着**所有的增、删、改都会清空本地缓存**，这和是否配置了flushCache=true是无关的。

这很好理解，因为增、删、改这三种操作都可能会导致查询出来的结果并不是原来的结果，如果增、删、改不清理缓存，那么可能导致读取出来的数据是脏数据。

因此**MyBatis二级缓存的生命周期即整个应用的生命周期**，应用不结束，定义的二级缓存都会存在在内存中。

从这个角度考虑，为了避免MyBatis二级缓存中数据量过大导致内存溢出，MyBatis在配置文件中给我们增加了很多配置例如size（缓存大小）、flushInterval（缓存清理时间间隔）、eviction（数据淘汰算法）来保证缓存中存储的数据不至于太过庞大。

# 脏读问题 参照缓存

MyBatis中很少会同时使用Mapper接口注解方式和XML映射文件，所以参照缓存并不是为了解决这个问题而设计的。参照缓存除了能够通过引用其他缓存减少配置外，主要的作用是解决脏读（后面章节详细介绍）。

## MyBatis默认提供的缓存实现是基于Map实现的内存缓存

MyBatis默认提供的缓存实现是基于Map实现的内存缓存，已经可以满足基本的应用。但是当需要缓存大量的数据时，不能仅仅通过提高内存来使用MyBatis的二级缓存，还可以选择一些类似EhCache的缓存框架或Redis缓存数据库等工具来保存MyBatis的二级缓存数据。接下来两节，我们会介绍两个常见的缓存框架。

Redis是一个高性能的key-value 数据库。

MyBatis项目开发者提供了Redis的MyBatis二级缓存实现，该项目名为redis-cache，目前只有beta版本，项目地址是https：//github.com/mybatis/redis-cache。

这一节将使用MyBatis官方提供的redis-cache集成Redis数据库，步骤如下。

