数据库缓存技术

(128条消息) 数据库缓存_无香菜不欢的博客-CSDN博客
数据库缓存的第一个技术特点就是提高性能，数据库缓存的数据基本上都是存储在内存中，相比io读写的速度，数据访问快速返回。而且在mysql 5.6的版本开始，已经把memcache这种跟数据库缓存直接挂钩的中间件集成进去了，已经不需要我们自己去单独部署对应数据库缓存的中间件了。

因为在常见的应用中，数据库层次的压力有80%的是查询，20%的才是数据的变更操作。

缓存中的数据和数据库中的数据的一致性问题。

用于数据库缓存场景的开源技术，好像memcache和redis这两个中间件比较有名。因为都是专注于内存缓存领域，memcache和redis向来都有争议。有持久化需求或者对数据结构和处理有高级要求的应用，选择redis。其他简单的key/value存储，选择memcache。所以根据自身业务特性，数据库缓存来选择适合自己的技术。

`redis`/`memcache`缓存

数据库缓存技术 数据库的优化 - 知乎 (zhihu.com)

- Redis,依赖客户端来实现分布式读写
- Memcache本身没有数据冗余机制
- Redis支持(RDB快照、AOF),依赖快照进行持久化,aof增强了可靠性的同时,对性能有所影响
- Memcache不支持持久化,通常做缓存,提升性能;
- Memcache在并发场景下,用cas保证一致性, redis事务支持比较弱,只能保证事务中的每个操作连续执行
- Redis支持多种类的数据类型
- Redis用于数据量较小的高性能操作和运算上
- Memcache用于在动态系统中减少数据库负载,提升性能;适合做缓存,提高性能

1. 数据结构：Memcache只支持key value存储方式，Redis支持更多的数据类型，比如Key value、hash、list、set、zset;
2. 多线程：Memcache支持多线程，Redis支持单线程；CPU利用方面Memcache优于Redis;
3. 持久化：Memcache不支持持久化，Redis支持持久化;
4. 内存利用率：Memcache高，Redis低（采用压缩的情况下比Memcache高）；
5. 过期策略：Memcache过期后，不删除缓存，会导致下次取数据时候的问题，Redis有专门线程，清除缓存数据;

缓存可能问题

1. 缓存穿透： DB 承受了没有必要的查询流量，意思就是查到空值的时候没有做缓存处理，再次查询的时候继续读库了
2. 缓存击穿：热点 Key，大量并发读请求引起的小雪崩， 就是缓存在某个时间点过期的时候，恰好在这个时间点对这个 Key 有大量的并发请求过来，这些请求发现缓存过期一般都会从后端 DB 加载数据并回设到缓存，这个时候大并发的请求可能会瞬间把后端 DB 压垮
3. 缓存雪崩：缓存设置同一过期时间，引发的大量的读取数据库操作

也就是说对于每一个key的缓存我们应当根据情况定制该缓存的过期时间。

Redis 是一个键值对数据库服务器，服务器中通常包含着任意个非空数据库，而每个非空数据库中有可以包含任意个键值对，为了方便起见，我们**将服务器中的非空数据库以及它们的键值对统称数据库状态**。

AOF 持久化功能则提供了一种更为可靠的持久化方式。 每当 Redis 接受到会修改数据集的命令时，就会把**命令**追加到 AOF 文件里，当你重启 Redis 时，AOF 文件里的命令会被重新执行一次，重建数据。

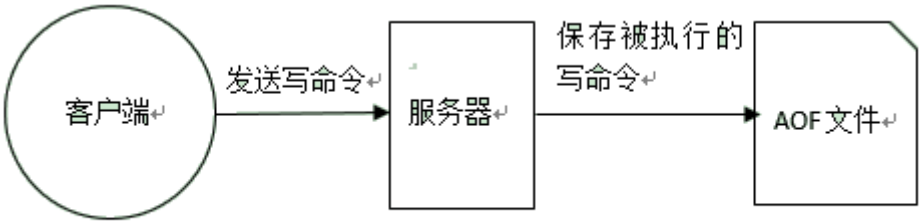


图 1-1-1 AOF 持久化

RDB持久化：简单来说就是会存储当前数据库状态的快照。

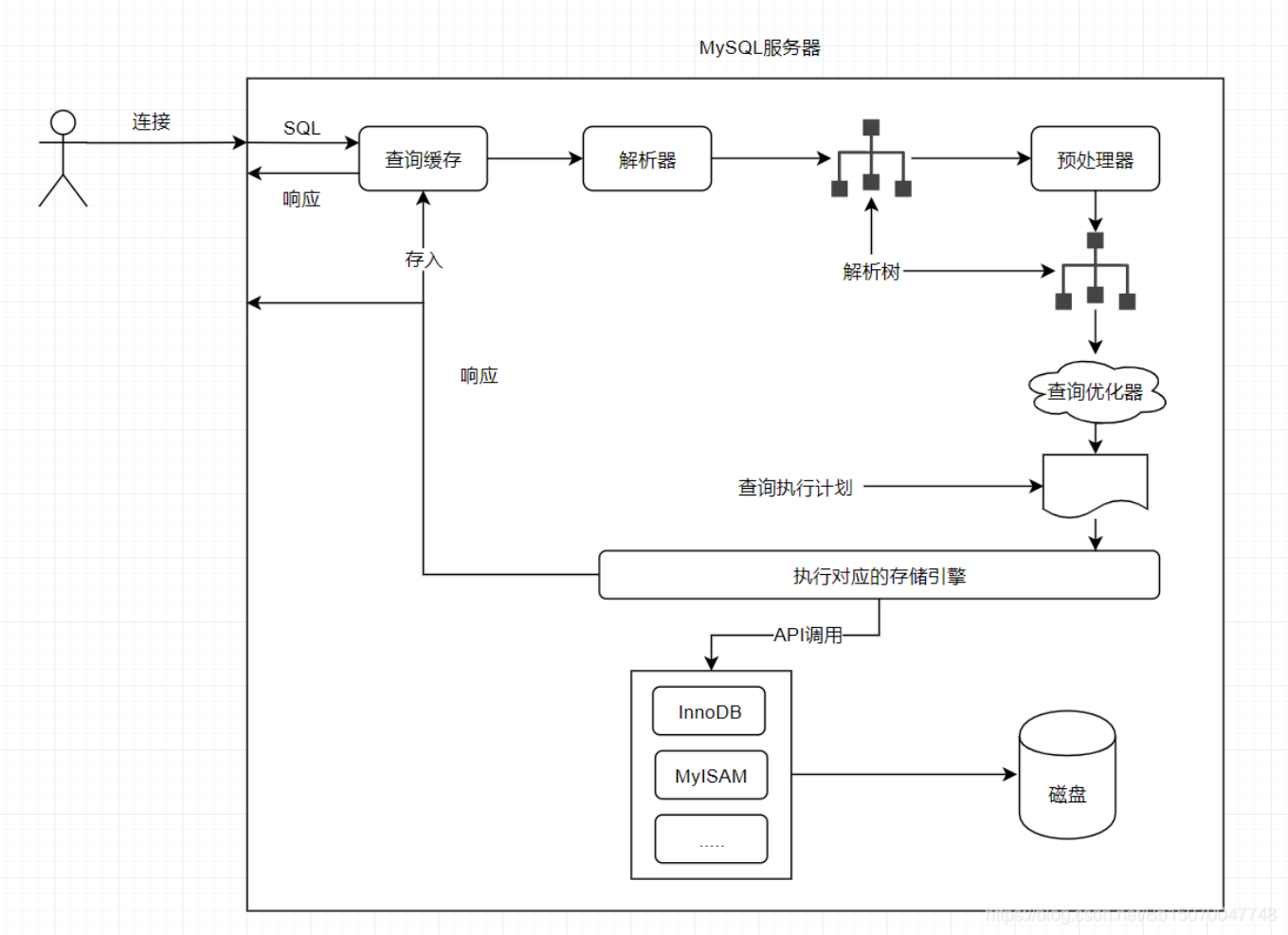
MySQL缓存机制

(129条消息) MySQL缓存深入理解（全网最深、最全、最实用）_查询缓存 最全 最深绿水長流*z的博客-CSDN博客

事务管理（ACID）：**原子性（Atomicity），一致性（Consistency），隔离性（Isolation），持久性（Durability）**

Mybatis

Mybatis：动态SQL：动态SQL就是指根据不同环境生成不同的sql语句



MySQL缓存机制就是缓存SQL文本及缓存结果，用**KV形式**保存再服务器内存中，如果运行相同的sql,服务器直接从缓存中去获取结果，不需要在再去解析、优化、执行sql。 如果这个表修改了，那么使用这个表中的 **所有缓存** 将不再有效，查询缓存值得相关条目将被清空。

MySQL在实现Query Cache的具体技术细节上类似典型的KV存储，就是将SELECT语句和该查询语句的结果集做了一个HASH映射并保存在一定的内存区域中，其中细节包含select语句包含的各个字段。 要完全一模一样：缓存存在一个hash表中，通过查询SQL，查询数据库，客户端协议等作为key。在判断是否命中前，MySQL不会解析SQL，而是直接使用SQL去查询缓存，SQL任何字符上的不同，如空格，注释，都会导致缓存不命中。

表中的任何改变是值表中任何数据或者是结构的改变，包括insert,update,delete,truncate,alter table,drop table或者是drop database 包括那些映射到改变了的表的使用merge表的查询，显然，对于频繁更新的表，查询缓存不合适，对于一些不变的数据且有大量相同sql查询的表，查询缓存会节省很大的性能。

对于wlock这种是不是缓存还会存在和lock被锁住的数据表的一些问题。

对MySQL表的任意DML操作都会导致有关于这张表的**所有缓存全部清空**。

缓存命中条件

缓存存在一个hash表中，通过查询SQL，查询数据库，客户端协议等作为key，在判断命中前，MySQL不会解析SQL，而是使用SQL去查询缓存，SQL上的任何字符的不同，如空格，注释，都会导致缓存不命中。

如果查询有不确定的数据current_date()（例如当前的日期这种，时间会一直改变），那么查询完成后结果者不会被缓存，包含不确定的数的是不会放置到缓存中。

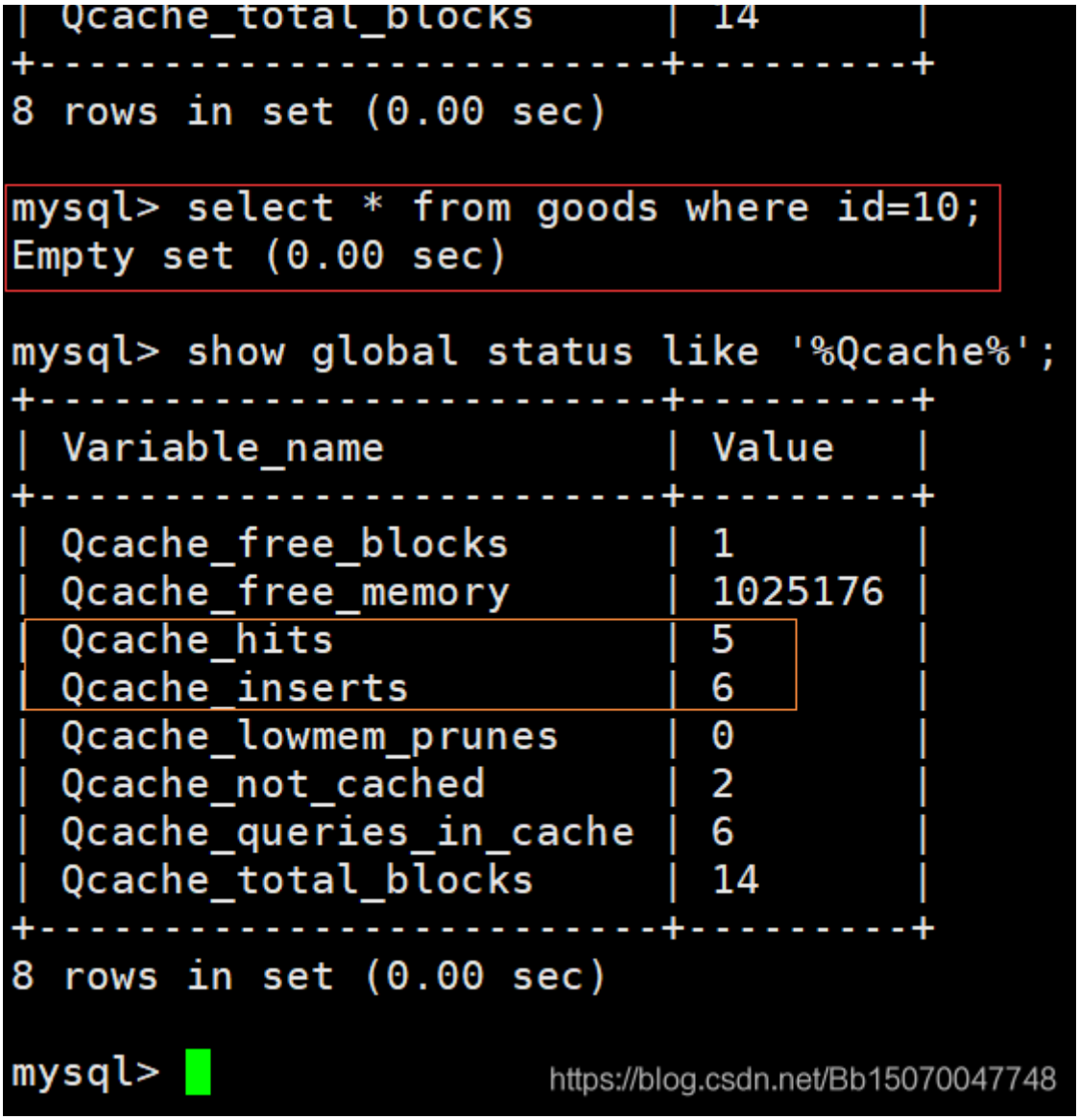
这一部分结束

在MySQL的机制之中：发现即使有的SQL没有查询到结果集，也会写入缓存，并且再次查询也会命中缓存：

```
mysql> show global status like '%Qcache%';
+-------------------------+---------+
| Variable_name           | Value   |
+-------------------------+---------+
| Qcache_free_blocks      | 1       |
| Qcache_free_memory      | 1026200 |
| Qcache_hits             | 4       |
| Qcache_inserts          | 5       |
| Qcache_lowmem_prunes    | 0       |
| Qcache_not_cached       | 2       |
| Qcache_queries_in_cache | 5       |
| Qcache_total_blocks     | 12      |
+-------------------------+---------+
8 rows in set (0.00 sec)

mysql> select * from goods where id=10;
Empty set (0.00 sec)

mysql> show global status like '%Qcache%';
+-------------------------+---------+
| Variable_name           | Value   |
+-------------------------+---------+
| Qcache_free_blocks      | 1       |
| Qcache_free_memory      | 1025176 |
| Qcache_hits             | 4       |
| Qcache_inserts          | 6       |
| Qcache_lowmem_prunes    | 0       |
| Qcache_not_cached       | 2       |
| Qcache_queries_in_cache | 6       |
```



使用SQL Hint选择缓存

我们知道MySQL的查询缓存一旦开启，会将本次SQL语句的结果集全部放入缓存中，这样其实是非常不友好的，因为我们知道，对于表的任何DML操作都会导致这张表的缓存全部清空。因此我们可以指定哪些SQL语句存入缓存，哪些不存。

- SQL_CACHE：将此次SQL语句的结果集存入缓存（前提是当前MySQL服务器时开启缓存的）
- SQL_NO_CACHE：此次SQL语句的结果集不存入缓存

设置合适的`query_cache_min_res_unit`值：与设置合适大小也就是4K的内存页的逻辑差不多，为的就是能够减小内存碎片的数量。

在MySQL8.0及以上版本，MySQL的缓存功能已经被删除了。。。主要原因就是缓存经常失效，但是对于安全多方数据库而言，十分大的安全多方计算代价，好像这个东西又很好用。
（查询缓存的失效非常频繁，只要有对一个表的更新，这个表上的所有的查询缓存都会被清空。

因此很可能你费劲地把结果存起来，还没使用呢，就被一个更新全清空了。对于更新压力大的数据库来说，查询缓存的命中率会非常低。**因此读写非常频繁的业务场景，缓存开启还不如关闭效率高。**）这种情况主要就是一次改动全部都要失效，粒度太大了。

发现开启缓存第一次查询明显比关闭缓存查询效率低多了，因为**当缓存开启后，MySQL需要将本次查询的结果集全部放入缓存中，这个过程是需要时间的**，如果后期能用上还好，如果用不上，或者创建了缓存里面又给情况了，那么无疑是在做无用功。缓存这个就有点像双刃剑的意思了。

一般来说，我们的数据库都是单独部署在一台服务器中，我们应该尽可能的减少这台服务器的压力，必要时还要进行扩容（搭建集群、读写分离、数据分片等），这些操作都是来提高我们单台MySQL的处理能力的，而不是把缓存和数据库放在一起，增加MySQL服务器的压力。如果真的需要缓存来提高响应速度，那么应该把缓存和数据库独立分开部署。（这也是为什么紧接着对应的Redis和Mencache中间件的出现）

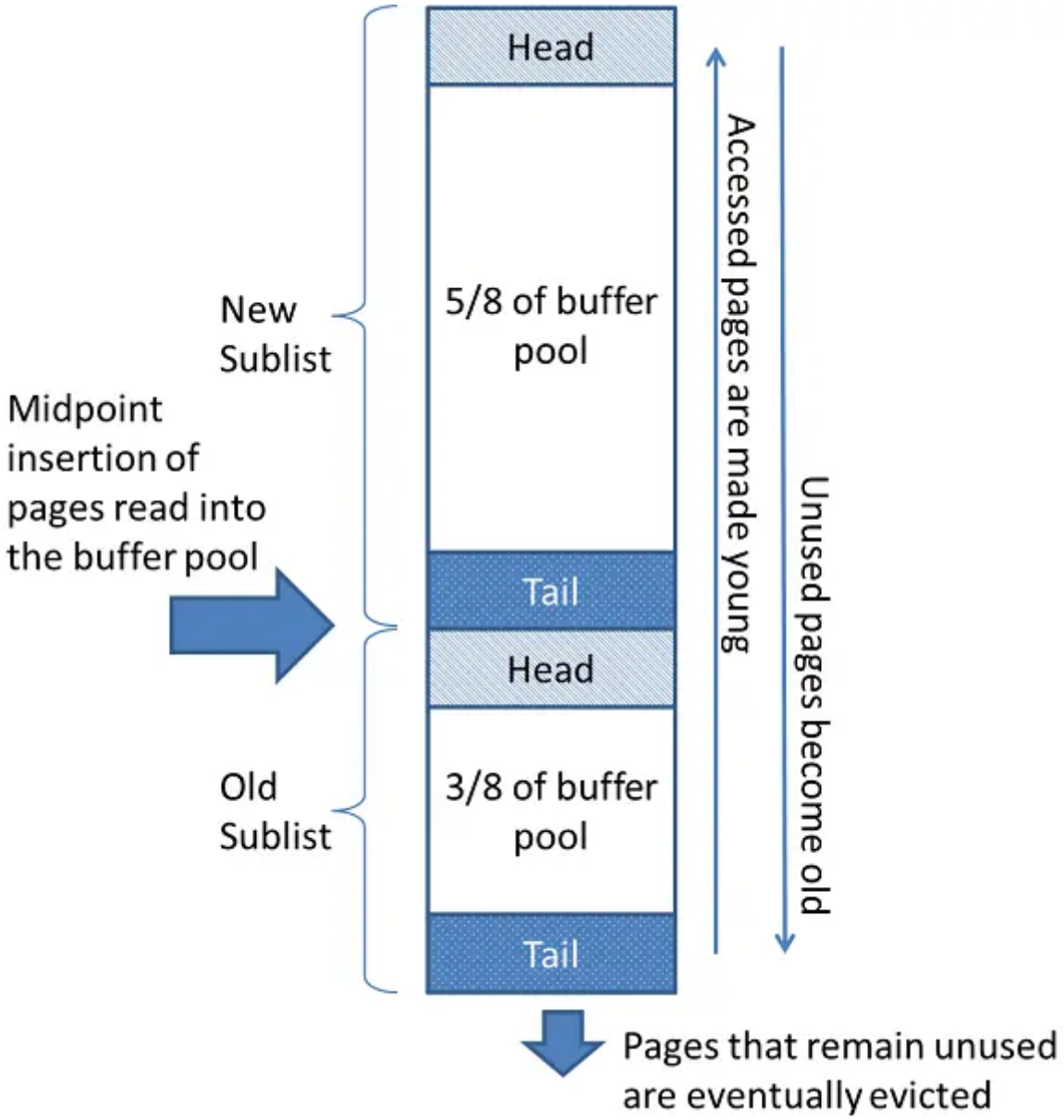
MySQL自带的并发压力测试工具：`mysqlsalp`

会出现一种情况，**通过SQL语句筛选出来的结果集少，会存入缓存，但是并发量非常高，加上又经常修改。** 执行SQL脚本里面都是创建一次缓存然后里面就修改了数据（清空了缓存）在处理小的结果集存入查询缓存的时间是可以忽略不计的，即使是**小规模的高并发**情况下。

Query Cache因MySql的存储引擎不同而实现略有差异，比如MyISAM，缓存的结果集存储在OS Cache中，而InnoDB则放在Buffer Pool中。MySQL查询缓存详解（总结） - 范仁义 - 博客园 (cnblogs.com)

如果是OS Cache也就是对应的OS操作系统决定的一系列的cache的策略与机制，例如OS中的LRU等cache策略。

下图展示的为InnoDB的Buffer Pool结构：



可以看出，Buffer Pool由1个大链表组成。因为Buffer Pool既然为缓存，肯定有对应的数据淘汰策略，不可能将所有的数据都拿到内存中存储，常见的缓存策略有LRU、LFU等，都是基于链表实现，Buffer Pool采用了基于LRU的自定义淘汰策略，用于解决标准LRU策略中的一些问题。
Buffer Pool链表一分为二为new sublist与old sublist，两个链表首尾相连。默认情况下，old list占用总链表3/8，里面存储的是最近很少访问的数据，即将要淘汰的数据；剩下的为new list，里面存储的是最近才被访问的数据。

Buffer Pool的处理：

- 当数据被读取的时候会存入Buffer Pool，这里的读取包含两个操作，一个是用户触发的读取，例如SQL query；或者是InnoDB自动触发的预读读取。
- 线性预读【预测在buffer pool中被访问到的数据它临近的页也会很快被访问到，可以使用innodb_read_ahead_threashold配置设置当连续读到多少个page时触发一次预读】
- 随机预读【根据已经在buffer pool中存在的page来猜测哪些数据将会被读取，如果随机预读开启，InnoDB如果发现Buffer Pool中有13个来自一个extent的page，那么将会发起异步请求读取

该extent剩余的page】

被读取的page将会被插入old list的头部，读取old list中的page会使其变“young”，会将其移动到new list的头部。如果page由于用户行为被读取，page第一次被访问的时候就会使其变“young”。这样随着page不断被访问，new list数据越来越多，超过限制后就会存入old list中，若old list也满了之后，就会从old list的尾部进行page的驱逐。

采用old list与new list这样的结构可以有效避免预读的数据不被访问的情况，如果只是原始的LRU算法，那么预读的数据也会被加入到链表的头部，会把真正的热数据往下“挤”，而假设预读的数据又不会被访问，那么Buffer Pool中会存储大量的冷数据，失去了做缓存的作用。

Buffer Pool如何解决大数据量读取的情况？

假设用户执行了一条全表查询的SQL，恰好这个表数据又很大，这就导致Buffer Pool中大量热数据被这个全表查询所替换，而这个数据可能就只会被快速的再访问几次，之后就不再被访问了。
使用innodb_old_blocks_time配置一个时间窗口参数，在第一次访问该页面之后，如果在该窗口时间内访问该page，该page也不会移动到new list的头部了

对于Buffer Pool的部分主要参考自 InnoDB Buffer Pool详解（缓冲池） - 简书 (jianshu.com)

`Redis` / `Memcache` 缓存技术详细探究

严格来说MySQL是包含有自身的缓存机制的，但是其自身的缓存机制无法满足需求与场景，所以现在最流行的两种组件就是Redis和Memcache。

`Redis`

接下来我们先来介绍一下最流行的Redis：

Redis的过期策略以及内存淘汰机制

Redis原理和机制详解 - 知乎 (zhihu.com)

Redis采用的是**定期删除+惰性删除策略**。

为什么不用定时删除策略?
定时删除，用一个定时器来负责监视key，过期则自动删除。虽然内存及时释放，但是十分消耗CPU资源。在大并发请求下，CPU要将时间应用在处理请求，而不是删除key，因此没有采用这一策略。

定期删除+惰性删除是如何工作的呢?
定期删除，Redis默认每隔100ms检查，是否有过期的key，有过期key则删除。需要说明的是，

Redis不是每个100ms将所有的key检查一次，而是随机抽取进行检查(如果每隔100ms，全部key进行检查，Redis岂不是卡死)。因此，如果只采用定期删除策略，会导致很多key到时间没有删除。
于是，惰性删除派上用场。也就是说在你获取某个key的时候，Redis会检查一下，这个key如果设置了过期时间那么是否过期了？如果过期了此时就会删除。

采用定期删除+惰性删除就没其他问题了么?
不是的，如果定期删除没删除key。然后你也没及时去请求key，也就是说惰性删除也没生效。这样，redis的内存会越来越高。那么就应该采用**内存淘汰机制**。

在redis.conf中有一行配置：

```
# maxmemory-policy allkeys-lru
```

上述这个就是设置的内存淘汰机制。

1）noeviction：当内存不足以容纳新写入数据时，新写入操作会报错。**应该没人用吧。**
2）allkeys-lru：当内存不足以容纳新写入数据时，在键空间中，移除最近最少使用的key。**推荐使用。**
3）allkeys-random：当内存不足以容纳新写入数据时，在键空间中，随机移除某个key。**应该也没人用吧，你不删最少使用key，去随机删。**
4）volatile-lru：当内存不足以容纳新写入数据时，在设置了过期时间的键空间中，移除最近最少使用的key。**这种情况一般是把redis既当缓存，又做持久化存储的时候才用。不推荐**
5）volatile-random：当内存不足以容纳新写入数据时，在设置了过期时间的键空间中，随机移除某个key。**依然不推荐**
6）volatile-ttl：当内存不足以容纳新写入数据时，在设置了过期时间的键空间中，有更早过期时间的key优先移除。**不推荐**
ps：如果没有设置 expire 的key，不满足先决条件(prerequisites)；那么 volatile-lru，volatile-random 和 volatile-ttl 策略的行为，和 noeviction(不删除) 基本上一致。

Redis的数据过期策略

1. Redis配置项hz定义了serverCron任务的执行周期，默认为10，即cpu空闲时每秒执行10次。
2. 每次过期key清理的视觉不超过cpu时间的25%，即若hz=1，则一次清理时间最大为250ms，若hz=10，则一次清理时间最大为25ms。
3. 清理时依次遍历所有的db。
4. 从db中随机取20个key，判断是否过期，若过期则清理。
5. 若有5个以上key过期，则重复步骤4，否则遍历下一个db。
6. 在清理过程中，若达到了25% cpu时间，则退出清理过程。

Redis的数据淘汰策略

1. volatile-lru：从已设置过期时间的数据集（server.db[i].expires）中挑选最近最少使用的数据淘汰
2. volatile-ttl：从已设置过期时间的数据集（server.db[i].expires）中挑选将要过期的数据淘汰
3. volatile-random：从已设置过期时间的数据集（server.db[i].expires）中任意选择数据淘汰
4. allkeys-lru：从数据集（server.db[i].dict）中挑选最近最少使用的数据淘汰
5. allkeys-random：从数据集（server.db[i].dict）中任意选择数据淘汰

no-enviction（驱逐）：禁止驱逐数据

Redis的到期时间（失效时间）设置

在使用 Redis 存储数据的时候，有些数据可能在某个时间点之后就不再有用了，用户可以用 DEL 命令显式地删除这些无用的数据，也可以通过 Redis 的过期时间（expiration）特性来让一个键在给定的时限（timeout）之后自动删除。

Redis可以为每个key设置过期时间，会将每个设置了过期时间的key放入一个独立的字典中。dict 用于维护一个 Redis 数据库中包含的所有 Key-Value 键值对，expires则用于维护一个 Redis 数据库中设置了失效时间的键(即key与失效时间的映射)。

Redis有四个不同的命令可以用于设置键的生存时间（键可以生存多久）或过期时间（键什么时候会被删除）：

1. expire 命令用于将键key的生存时间设置为ttl**秒**
2. pexpire 命令用于将键key的生存时间设置为ttl**毫秒**
3. expireat 命令用于将键key的过期时间设置为timestamp所指定的**秒**数时间戳
4. pexpireat 命令用于将键key的过期时间设置为timestamp所指定的**毫秒**数时间戳

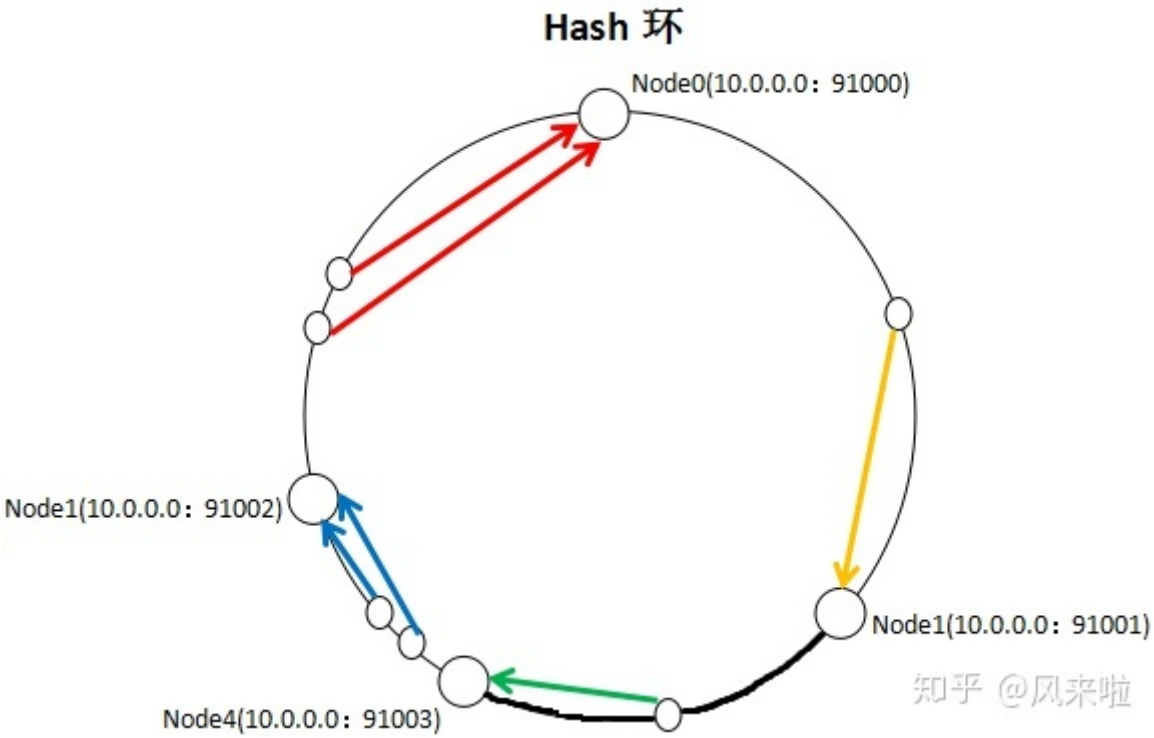
还有一种情况就是如果遇到了高并发的情形，并且还有不少的数据更新，那么缓存就会失效，如果更新的频率低可以选择先更新缓存，如果数据更新多，可以选择先令缓存失效然后通过消息队列进行顺序处理。（这些可以自己定制策略）

`Memcache`

MemCache原理超详细解读（仅学习） - 知乎 (zhihu.com)

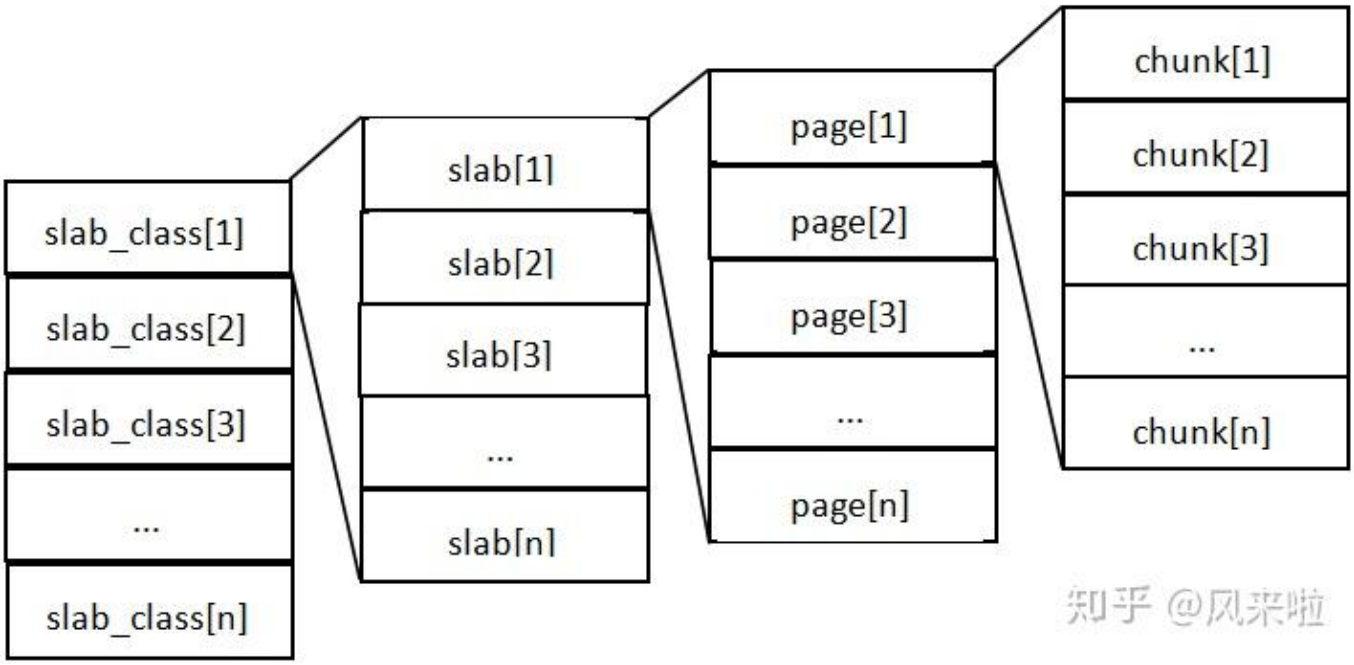
MemCache是一个自由、源码开放、高性能、分布式的分布式内存对象缓存系统，用于动态Web应用以减轻数据库的负载。它通过在内存中缓存数据和对象来减少读取数据库的次数，从而提高了网站访问的速度。MemCaChe是一个存储键值对的HashMap，在内存中对任意的数据（比如字符串、对象等）进行key-value的存储。

读缓存和写缓存一样，只要使用相同的路由算法和服务器列表，只要应用程序查询的是相同的Key，MemCache客户端总是访问相同的客户端去读取数据，只要服务器中还缓存着该数据，就能保证缓存命中。



上图为一致性hash算法用于负载均衡，分布式缓存的。（对于本探究倒是没啥用借鉴意义）

然后我们来看一下MemCache的内存分配原理，MemCache采用的内存分配方式是固定空间分配，下图来进行说明：



可以看到MemCache在内存分配的策略上基本和Linux操作系统里面的slab算法一致，也就是可以用售货商来模拟，这样的算法可以减轻内存碎片的情况。

如果这个slab中没有chunk可以分配了怎么办，如果MemCache启动没有追加-M（禁止LRU，这种情况下内存不够会报Out Of Memory错误），那么MemCache会把这个slab中最近最少使用的chunk中的数据清理掉，然后放上最新的数据。（Linux操作系统上会申请一个新的page追加）

一些缓存分配回收的要点：
1、MemCache的内存分配chunk里面会有内存浪费，88字节的value分配在128字节（紧接着大的用）的chunk中，就损失了30字节，但是这也避免了管理内存碎片的问题

**2、MemCache的LRU算法不是针对全局的，是针对slab的

3、应该可以理解为什么MemCache存放的value大小是限制的，因为一个新数据过来，slab会先以page为单位申请一块内存，申请的内存最多就只有1M，所以value大小自然不能大于1M了

MemCache设置添加某一个Key值的时候，传入expiry为0表示这个Key值永久有效，这个Key值也会在30天之后失效，见memcache.c的源代码：

```
#define REALTIME_MAXDELTA 60*60*24*30
static rel_time_t realtime(const time_t exptime) {
       if (exptime == 0) return 0;
       if (exptime > REALTIME_MAXDELTA) {
              if (exptime <= process_started)
                     return (rel_time_t)1;
              return (rel_time_t)(exptime - process_started);
       } else {
              return (rel_time_t)(exptime + current_time);
       }
}
```

这个失效的时间是memcache源码里面写的，开发者没有办法改变MemCache的Key值失效时间为30天这个限制

(130条消息) memcache 内存机制与缓存原理_yshir-phper的博客-CSDN博客
(130条消息) 【软件开发】Memcached（理论篇）_G皮T的博客-CSDN博客
接下来我们来看看Memcache的缓存失效机制和缓存替换机制：

- 当某个值过期后，并没有从内存删除，因此，stats 统计时，curr_item 有其信息
- 当某个新值去占用他的位置时，当成空 chunk 来占用。
- 当 get 值时，判断是否过期，如果过期，返回空，并且清空，curr_item 就减少了。

即 - 这个过期，只是让用户看不到这个数据而已，并没有在过期的瞬间立即从内存删除。这个称为 lazy expiration，惰性失效。好处：节省了 cpu 时间和检测的成本

在惰性失效上和Redis有一定相似。

Memcached 懒惰检测对象过期机制

Memcached 不会主动检测 item 对象是否过期，而是在进行 get 操作时检查 item 对象是否过期以及是否应该删除！

因为不会主动检测 item 对象是否过期，自然也就不会释放已分配给对象的内存空间了，除非为添加的数据设定过期时间或内存缓存满了，在数据过期后，它对应的value值不再可用，其存储空间将被重新利用。

Memcached 使用的这种策略为 懒惰检测对象过期策略，即自己不监控存入的 key / value 对是否过期，而是在获取 key 值时查看记录的时间戳（sed key flag exptime bytes），从而检查 key / value 对空间是否过期。这种策略不会在过期检测上浪费 CPU 资源。

Memcached 懒惰删除对象机制

当删除 item 对象时，一般不会释放内存空间，而是做删除标记，将指针放入 slot 回收插槽，下次分配的时候直接使用。

Memcached 在分配空间时，会优先使用已经过期的 key / value 对空间；若分配的内存空间占满，Memcached 就会使用 LRU 算法来分配空间，删除最近最少使用的 key / value 对，从而将其空间分配给新的 key / value 对。在某些情况下（完整缓存），如果不想使用 LRU 算法，那么可以通过 -M 参数来启动 Memcached，这样，Memcached 在内存耗尽时，会返回报错信息。

- 不主动检测 item 对象是否过期，而是在 get 时才会检查 item 对象是否过期以及是否应该删除。
- 当删除 item 对象时，一般不释放内存空间，而是做删除标记，将指针放入 slot 回收插槽，下次分配的时候直接使用。
- 当内存空间满的时候，将会根据 LRU 算法把最近最少使用的 item 对象删除。
- 数据存入可以设定过期时间，但是数据过期后不会被立即删除，而是在 get 时检查 item 对象是否过期以及是否应该删除。
- 如果不希望系统使用 LRU 算法清除数据，可以用使用 -M 参数。

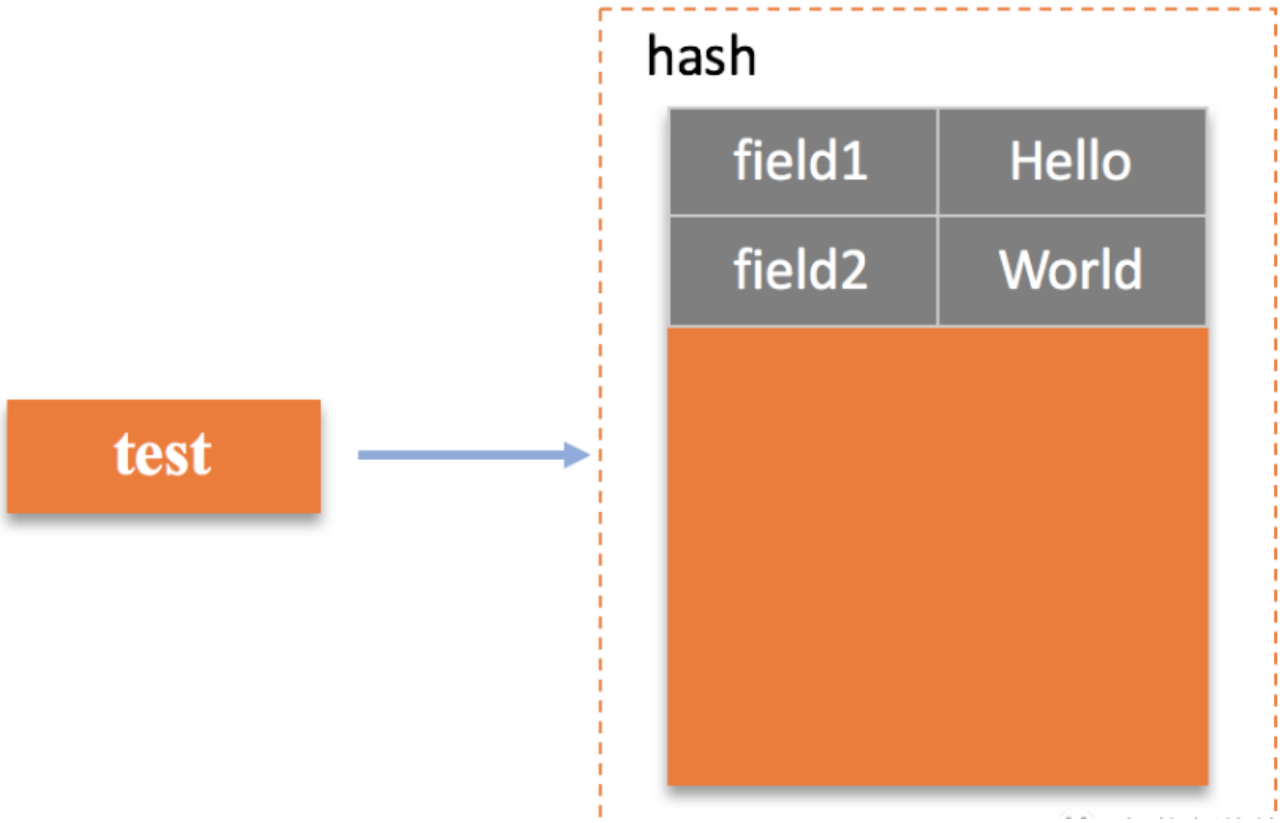
分布式数据之缓存技术，这次我是真的搞懂了 - 知乎 (zhihu.com)

Redis 是一个基于内存的 key-value 数据库，为了方便支持多应用的缓存，比如缓存文本类型、数据库的查询结果（字段与字段对应的值）等等，支持的数据结构不仅有简单的 k／v 类型，还可以支持 List、Set、Hash 等复杂类型的存储。

以 Hash 这种复杂类型的存储为例，Redis 将 Hash 视作一个整体当作数据库的 value（可以是一个对象，比如结构体对象）进行存储。如果把 Hash 结构的整体看作对象的话，Hash 结构里的 key-value 相当于该对象的属性名和属性值。

比如，插入 Hash 数据类型的命令：HMSET test field1 “Hello” field2 “World”中，如下图所示，test 为 key 值， field1 “Hello” field2 “World” 为 value 值，如果把整个 Hash 结构看做对象的话，则 field1、field2 类似于对象中的属性名，“Hello” “World”类似于对象中的属性值。

我感觉对于其中Hash类型的数据存储，有点类似于密文的存储。


hash
field1
Hello
field2
World
test