Atitit 提升性能架构 之cache

目录

[1. .1缓存的原理 更快存储，更近的位置 2](#_Toc10971)

[1.1. 局部性原理 2](#_Toc4530)

[1.2. 金字塔型的存储器体系结构 2](#_Toc6377)

[2. cache具有的功能 2](#_Toc9027)

[2.1. TTL定时功能 2](#_Toc27303)

[2.2. 丰富的数据结构 2](#_Toc19920)

[2.3. LRU剔除算法等 换页算法 2](#_Toc29254)

[3. Cache的层次 3](#_Toc27975)

[3.1. 高速缓存 低速缓存 3](#_Toc3209)

[3.2. L1 L2 L3 三级缓存体系 3](#_Toc24996)

[3.3. Cpu 内存 磁盘 三级cache体系 3](#_Toc24226)

[3.4. 读缓存 vs 写缓存批量 3](#_Toc29676)

[3.5. 缓存分类 应用内缓存 vs 外部缓存 3](#_Toc19377)

[3.6. 常见的缓存形式 内存缓存。文件缓存。 3](#_Toc2088)

[3.7. 技术大体可分为三大类： 1. 缓存内容（Caching content）2. 内存缓存（Memory Cache）3. 数据库缓存（Database Cache） 4](#_Toc10078)

[4. 场景cache 4](#_Toc11294)

[4.1. 程序中间件提供的缓存比如Srping cache 4](#_Toc5673)

[4.2. 数据库中间件提供的缓存，比如Mybatis cache 5](#_Toc20689)

[4.3. Ehcache等磁盘cache Guava Cache 5](#_Toc20843)

[4.4. 工厂模式等单例模式 内存cache 5](#_Toc11543)

[4.5. CDN静态缓存 nginx缓存 5](#_Toc29718)

[4.6. 静态化页面 h5 缓存 store api 5](#_Toc31170)

[5. 数据库提供的cache工具 5](#_Toc24243)

[5.1. 内存表 临时表 物化视图 Cte缓存机制 5](#_Toc31793)

[5.2. 延迟写入 缓存机制 批量插入 5](#_Toc29755)

[5.3. 专用的cache sql 关键字，强制指定缓存 5](#_Toc14913)

[5.4. 集群模式 读写分离 从库缓存法 5](#_Toc16052)

[5.5. 数据库调整query cache参数 更大性能提升 5](#_Toc19192)

[5.6. 数据库连接池 驱动长连接 6](#_Toc16830)

[5.7. 负载均衡法 复制 读写分离 分区模式 6](#_Toc4455)

[6. 同步到nosql 作为cache 6](#_Toc12910)

[7. es mongodb等 redis memecache缓存 6](#_Toc23428)

[8. 应用方法 6](#_Toc18223)

[8.1. 换用性能更高的组件 大型数据库 大型mq等 6](#_Toc27931)

[8.2. 尽可能使用标准协议 方便跨越 6](#_Toc10112)

[9. 缓存的分片机制 6](#_Toc32518)

[9.1. 上云 提供虚拟化 分散压力 运维方便 6](#_Toc710)

[9.2. User key,date key ,hash key三种模式的key取值法 6](#_Toc15681)

[9.3. 客户端分片 负载均衡法 6](#_Toc20110)

[9.4. 服务端分片 6](#_Toc25248)

[10. 一些设计技巧 7](#_Toc28652)

[10.1. 4.2 反向Cache 7](#_Toc18222)

[10.2. 4.3 缓存Fail-Fast (快速失败) 8](#_Toc28901)

[10.3. 4.4 缓存无过期(Cache is Storage) 8](#_Toc7802)

[10.4. 4. 7避免雪崩 流控(具体应该放量多少) 8](#_Toc10749)

[11. Ref 9](#_Toc19807)

# .1缓存的原理 更快存储，更近的位置

（1） 将数据写入/读取速度更快的存储（设备）；

（2） 将数据缓存到离应用最近的位置；

（3） 将数据缓存到离用户最近的位置。

缓存的内容：文件，数据，对象；

## 局部性原理

由于程序的局部性原理的存在，小容量的cache在工作时能轻易达到90%以上的读写命中率。局部性原理分为****时间局部性****和****空间局部性****，这

## 金字塔型的存储器体系结构

# cache具有的功能

## TTL定时功能

## 丰富的数据结构

## LRU剔除算法等 换页算法

缓存淘汰算法

FIFO/LFU/LRU/过期时间/随机

FIFO：最先进入缓存的数据，在缓存空间不足时被清除，为了保证最新数据可用，保证实时性

LFU(Least Frequently Used)：最近最不常用，基于访问次数，去除命中次数最少的元素，保证高频数据有效性

LRU(Least Recently Used)：最近最少使用，基于访问时间，在被访问过的元素中去除最久未使用的元素，保证热点数据的有效性

# Cache的层次

## 高速缓存 低速缓存

## L1 L2 L3 三级缓存体系

## Cpu 内存 磁盘 三级cache体系

## 读缓存 vs 写缓存批量

## **缓存分类** 应用内缓存 vs 外部缓存

    缓存大致可以分为两类，一种是应用内缓存，比如Map(简单的数据结构)，以及EH Cache(Java第三方库)，另一种就是缓存组件，比如Memached，Redis。

## **常见的缓存形式** 内存缓存。文件缓存。

## 技术大体可分为三大类： 1. 缓存内容（Caching content）2. 内存缓存（Memory Cache）3. 数据库缓存（Database Cache）

**1.2缓存分类** （1） CDN缓存；（2） 反向代理缓存；

在分布式系统中，缓存的应用非常广泛，从部署角度有以下几个方面的缓存应用。

（1） CDN缓存；（2） 反向代理缓存；

（3） 分布式Cache；

（4） 本地应用缓存；

# 场景cache

## 程序中间件提供的缓存比如Srping cache

## 数据库中间件提供的缓存，比如Mybatis cache

## Ehcache等磁盘cache Guava Cache

## 工厂模式等单例模式 内存cache

## CDN静态缓存 nginx缓存

## 静态化页面 h5 缓存 store api

# 数据库提供的cache工具

## 内存表 临时表 物化视图 Cte缓存机制

## 延迟写入 缓存机制 批量插入

## 专用的cache sql 关键字，强制指定缓存

## 集群模式 读写分离 从库缓存法

## 数据库调整query cache参数 更大性能提升

**query\_cache\_type =1**

**query\_cache\_limit = 1M**

**query\_cache\_size =16M**

## 数据库连接池 驱动长连接

## 负载均衡法 复制 读写分离 分区模式

# 同步到nosql 作为cache

# es mongodb等 redis **memecache缓存**

# 应用方法

## 换用性能更高的组件 大型数据库 大型mq等

## 尽可能使用标准协议 方便跨越

# 缓存的分片机制

## 上云 提供虚拟化 分散压力 运维方便

## User key,date key ,hash key三种模式的key取值法

使用用户id作为分片key，扩容容易，其次日期key，hash key会面临扩容问题。。

## 客户端分片 负载均衡法

## 服务端分片

客户端实现数据分片

*优点：*简单，容易实现

*缺点：*扩缩容需要重新上线，手动数据迁移；（使用日期缓存法或uuserid取key法则扩容容易。。不要使用hash法，会面临扩容问题。。）

* proxy实现：twemproxy，codis，微博内部实现了CacheService

    通过引入一层代理，将数据分片策略放在代理层实现，客户端通过代理来访问数据。

*优点：*逻辑在proxy实现，客户端使用简单，支持多语言

*缺点：*数据访问多一跳，有一定的性能损耗

* 服务端实现：redis 3.x，cassandra

    由缓存组件本身，实现数据分片机制。

*优点：*扩缩容方便，自动数据迁移

*缺点：*数据存储与分布式逻辑耦合在一起，服务端复杂

# 一些设计技巧

## 4.2 反向Cache

    反向Cache就是将一个不存在的key放在缓存中，也就是在缓存中存一个空值。在某些场景下，比如微博维度的计数场景，若采用cache+DB的存储方式，由于大多数的微博并不存在转发、评论计数，这种场景下，就会出现由于大量访问不存在计数的mid，导致DB压力居高不下的情况。通过在cache中存一个null值，可减少对DB的穿透。当然这也存在潜在的风险或问题：

* 如果每次都是不同的mid，缓存效果可能不明显
* 需要更多的缓存容量

## 4.3 缓存Fail-Fast (快速失败)

    当缓存层某个节点出现故障时，会导致请求持续穿透到存储层，使请求响应时间长(需要等到读写故障缓存节点超时)，并且存储层负载居高不下。这就需要在使用缓存时考虑快速失败机制。快速失败指的是：当出现故障节点时，标识故障节点为不可用节点（策略举例：连续N次请求都出现超时，标识M时间段内为不可用)，读写不可用节点快速返回。通过快速失败策略，解决请求响应时间长问题，保证SLA。

## 4.4 缓存无过期(Cache is Storage)

    缓存无过期是指缓存中存储全量数据，不存在数据穿透的情况。 相比于缓存+DB的访问模型，使用内存存储简单可靠，但相应的内存成本也较高。选择内存缓存还是内存存储，需要结合具体的业务场景做权衡，比如单纯为解决Dog-Pile Effect而采用内存存储的话，内存成本可能就无法接受。通常情况下，内存存储模式，适合总体数据量很小，但是访问量巨大的业务场景，比如微博应用(来自weibo.com，weico等)列表。

**4.5 dog-pile effect (狗桩效应)**

    狗桩效应是由于极热访问的缓存数据失效，大量请求发现没有缓存，进而穿透至DB，导致数据库load瞬间飙高甚至宕机。这是一个典型的并发访问穿透问题，理想情况下缓存失效对数据库应该只有一次穿透

## 4. 7避免雪崩 流控(具体应该放量多少)

     雪崩效应是由于缓存服务器宕机等原因导致命中率降低，大量的请求穿透到数据库，导致数据库被冲垮，业务系统出现故障，服务很难再短时间内回复。避免雪崩主要从以下几方面考虑：

* 缓存高可用

    避免单点故障，保证缓存高命中率

* 降级和流控

    故障期间通过降级非核心功能来保证核心功能可用性

    故障期间通过拒掉部分请求保证有部分请求还能正常响应

# Ref

Atitit 数据库提升性能的机制总结