本地缓存的局限

从上一篇我们已经了解了选择缓存需要考虑缓存吞吐量、缓存命中率、缓存扩展机制、缓存是否支持分布式几个维度， 还分别了解了几种Java 本地缓存。进程本地缓存因为就在进程的内存里面，不需要网络和对象拷贝的开销所以性能非常高，不过也正因为数据保存在进程的内存中也有了很多限制。

1、无法共享：本地缓存是无法多个进程共享。

2、不支持技术异构：另外进程级的缓存一般都和对应的开发语言绑定，无法提供给不同的开发语言使用。

3、无法扩展：缓存是绑定着进程共生共死的，其资源的分配和使用都限制其应用程序，也无法进行节点扩展。

4、没有持久化机制：同时因为数据都保存在内存中的不会进行持久化，所以一旦进程停止了缓存数据就都没有了。

所以综合来看，我们需要有一种支持多进程共享、数据持久化、技术异构的缓存，这种缓存也就是我们所熟知的分布式缓存。

分布式缓存的基本概念

1.1 分布式缓存的发展

1）本地缓存:数据存储在应用代码所在内存空间.优点是可以提供快速的数据访问;缺点是数据无法分布式共享,无容错处理.典型的,如Cache4j;

2）分布式缓存系统:数据在固定数目的集群节点间分布存储.优点是缓存容量可扩展(静态扩展);缺点是扩展过程中需要大量配置,无容错机制.典型的,如 Memcached;

3）弹性缓存平台:数据在集群节点间分布存储,基于冗余机制实现高可用性.优点是可动态扩展,具有容错能力;缺点是复制备份会对系统性能造成一定影响.典型的,如 Windows Appfabric Caching;

4）弹性应用平台:弹性应用平台代表了云环境下分布式缓存系统未来的发展方向.简单地讲,弹性应用平台是弹性缓存与代码执行的组合体,将业务逻辑代码转移到数据所在节点执行,可以极大地降低数据传输开销,提升系统性能.典型的,如 GigaSpaces XAP.

1.2 分布式缓存特性

1）高性能:当传统数据库面临大规模数据访问时,磁盘I/O 往往成为性能瓶颈,从而导致过高的响应延迟.分布式缓存将高速内存作为数据对象的存储介质,数据以key/value 形式存储,理想情况下可以获得DRAM级的读写性能

2）动态扩展性:支持弹性扩展,通过动态增加或减少节点应对变化的数据访问负载,提供可预测的性能与扩展性;同时,最大限度地提高资源利用率;

3）高可用性:可用性包含数据可用性与服务可用性两方面.基于冗余机制实现高可用性,无单点失效(single point of failure),支持故障的自动发现,透明地实施故障切换,不会因服务器故障而导致缓存服务中断或数据丢失.动态扩展时自动均衡数据分区,同时保障缓存服务持续可用;

4）易用性:提供单一的数据与管理视图;API接口简单,且与拓扑结构无关;动态扩展或失效恢复时无需人工配置;自动选取备份节点;多数缓存系统提供了图形化的管理控制台,便于统一维护;

5）分布式代码执行(distributed code execution):将任务代码转移到各数据节点并行执行,客户端聚合返回结果,从而有效避免了缓存数据的移动与传输.最新的Java 数据网格规范JSR-347中加入了分布式代码执行与Map/reduce 的API 支持,各主流分布式缓存产品,如IBM WebSphere eXtreme Scale,VMware GemFire,GigaSpaces XAP 和Red Hat Infinispan 等也都支持这一新的编程模型

1.3 分布式缓存与NoSQL

NoSQL 又称为Not Only Sql,主要是指非关系型、分布式、支持水平扩展的数据库设计模式.NoSQL 放弃了传统关系型数据库严格的事务一致性和范式约束,采用弱一致性模型.相对于NoSQL 系统,传统数据库难以满足云环境下应用数据的存储需求,具体体现在以下3 个方面:

1）根据CAP 理论,一致性(consistency)、可用性(availability)和分区容错(partition tolerance)这3 个要素最多同时满足两个,不可能三者兼顾.对云平台中部署的大量Web应用而言,数据可用性与分区容错的优先级通常更高,所以一般会选择适当放松一致性约束.传统数据库的事务一致性需求制约了其横向伸缩与高可用技术的实现;

2）传统数据库难以适应新的数据存储访问模式.Web 2.0 站点以及云平台中存在大量半结构化数据,如用户Session 数据、时间敏感的事务型数据、计算密集型任务数据等,这些状态数据更适合以Key/Value 形式存储,不需要RDBMS 提供的复杂的查询与管理功能;

3）NoSQL 提供低延时的读写速度,支持水平扩展,这些特性对拥有海量数据访问请求的云平台而言是至关重要的.传统关系型数据无法提供同样的性能,而内存数据库容量有限且不具备扩展能力.分布式缓存作为NoSQL 的一种重要实现形式,可为云平台提供高可用的状态存储与可伸缩的应用加速服务,与其他NoSQL 系统间并无清晰的界限.平台中应用访问与系统故障均具有不可预知性,为了更好地应对这些挑战,应用软件在架构时通常采用无状态设计,大量状态信息不再由组件、容器或平台来管理,而是直接交付给后端的分布式缓存服务或NoSQL 系统.

分布式缓存——redis介绍

一、五种数据类型

1、string 字符串（可以为整形、浮点型和字符串，统称为元素）

string类型的常用命令：

自加：incr

自减：decr

加： incrby

减： decrby

2、list 列表（实现队列,元素不唯一，先入先出原则）

Redis 列表是简单的字符串列表，按照插入顺序排序。你可以添加一个元素到列表的头部（左边）或者尾部（右边）。；

lpush:从左边推入

lpop:从左边弹出

rpush：从右变推入

rpop:从右边弹出

llen：查看某个list数据类型的长度

3、set 集合（无序，各不相同的元素，STring 类型）

Redis 的 Set 是 string 类型的无序集合。

集合是通过哈希表实现的，所以添加，删除，查找的复杂度都是 O(1)。

sadd 命令：sadd key member

sadd:添加数据

scard:查看set数据中存在的元素个数

sismember:判断set数据中是否存在某个元素

srem:删除某个set数据中的元素

hset:添加hash数据

hget:获取hash数据

hmget:获取多个hash数据

4、 hash hash散列值（ hash的key必须是唯一的 ）

Redis hash 是一个键值(key=>value)对集合。

Redis hash 是一个 string 类型的 field 和 value 的映射表，hash 特别适合用于存储对象。

HMSET

HGET

5、 zset （ 有序集合，不允许重复 ）

zadd:添加

zcard:查询

zrange:数据排序

类型	简介	特性	场景
String(字符串)	二进制安全	可以包含任何数据,比如jpg图片或者序列化的对象,一个键最大能存储512M	jpg图片或者序列化的对象
Hash(字典)	键值对集合,即编程语言中的Map类型	适合存储对象,并且可以像数据库中update一个属性一样只修改某一项属性值(Memcached中需要取出整个字符串反序列化成对象修改完再序列化存回去)	存储、读取、修改用户属性 存储对象
List(列表)	链表(双向链表)	增删快,提供了操作某一段元素的API	1,最新消息排行等功能(比如朋友圈的时间线) 2,消息队列
Set(集合)	哈希表实现,元素不重复	1、添加、删除,查找的复杂度都是O(1) 2、为集合提供了求交集、并集、差集等操作	1、共同好友 2、利用唯一性,统计访问网站的所有独立ip 3、好友推荐时,根据tag求交集,大于某个阈值就可以推荐
Sorted Set(有序集合)	将Set中的元素增加一个权重参数score,元素按score有序排列	数据插入集合时,已经进行天然排序	1、排行榜 2、带权重的消息队列

缓存哪些数据：

1、不需要实时更新但是又极其消耗数据库的数据。 比如网站上商品销售排行榜，这种数据一天统计一次就可以了，用户不会关注其是否是实时的。

2、需要实时更新，但是更新频率不高的数据。比如一个用户的订单列表，他肯定希望能够实时看到自己下的订单，但是大部分用户不会频繁下单。

3、在某个时刻访问量极大而且更新也很频繁的数据。这种数据有一个很典型的例子就是秒杀，在秒杀那一刻，可能有N倍于平时的流量进来，系统压力会很大。但是这种数据使用的缓存不能和普通缓存一样，这种缓存必须保证不丢失，否则会有大问题

单线程的redis为什么这么快

纯内存操作

单线程操作，避免了频繁的上下文切换

采用了非阻塞I/O多路复用机制

动态灾备切换，速度快不会导致数据丢失。

redis的优点

速度快

数据类型丰富

支持事务

持久化存储

主从同步，故障转移 集群

Redis持久化

Redis提供了两种持久化的方式：RDB（Redis DataBase）和AOFAppend Only File）。

RDB，简而言之，就是在不同的时间点，将redis存储的数据生成快照并存储到磁盘等介质上。

AOF，则是将redis执行过的所有写指令记录下来，在下次redis重新启动时，只要把这些写指令从前到后再重复执行一遍，就可以实现数据恢复了。

redis的缺点

由于 Redis 是内存数据库，短时间内大量增加数据，可能导致内存不够用。

redis是单线程的，单台服务器无法充分利用多核服务器的CPU

使用redis会遇到的问题

2.21 缓存穿透

（1）概念

查询一个数据库中不存在的数据，比如商品详情，查询一个不存在的ID，每次都会访问DB，如果有人恶意破坏，很可能直接对DB造成过大地压力

黑客故意去请求缓存中不存在的数据，导致所有的请求都怼到数据库上，从而数据库连接异常

（2）如何解决

a. 当通过某一个key去查询数据的时候，如果对应在数据库中的数据都不存在，我们将此key对应的value设置为一个默认的值，比如“NULL”，并设置一个缓存的失效时间，这时在缓存失效之前，所有通过此key的访问都被缓存挡住了。后面如果此key对应的数据在DB中存在时，缓存失效之后，通过此key再去访问数据，就能拿到新的value了

b. 采用异步更新策略，无论key是否取到值，都直接返回。value值中维护一个缓存失效时间，缓存如果过期，异步起一个线程去读数据库，更新缓存。需要做缓存预热(项目启动前，先加载缓存)操作

c. 提供一个能迅速判断请求是否有效的拦截机制，比如，利用布隆过滤器，内部维护一系列合法有效的key。迅速判断出，请求所携带的Key是否合法有效。如果不合法，则直接返回

2.22 缓存雪崩

（1）概念

a. 是指在我们设置缓存时采用了相同的过期时间，导致缓存在某一时刻同时失效，请求全部转发到DB，DB瞬时压力过重雪崩

b. 缓存同一时间大面积的失效，这个时候又来了一波请求，结果请求都怼到数据库上，从而导致数据库连接异常

（2）如何解决

a. 将系统中key的缓存失效时间均匀地错开，防止统一时间点有大量的key对应的缓存失效。比如我们可以在原有的失效时间基础上增加一个随机值，比如1-5分钟随机，这样每一个缓存的过期时间的重复率就会降低，就很难引发集体失效的事件

b.双缓存。我们有两个缓存，缓存A和缓存B。缓存A的失效时间为20分钟，缓存B不设失效时间。自己做缓存预热操作。然后细分以下几个小点

从缓存A读数据，有则直接返回

A没有数据，直接从B读数据，直接返回，并且异步启动一个更新线程

更新线程同时更新缓存A和缓存B

2.23 缓存击穿

（1）概念

缓存中的一个Key(比如一个促销商品)，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

（2）如何解决

对缓存查询加锁，如果KEY不存在，就加锁，然后查DB入缓存，然后解锁；其他进程如果发现有锁就等待，然后等解锁后返回数据或者进入DB查询

Redis 主从复制原理

一、什么是主从复制

在数据库语境下，复制(replication)就是将数据从一个数据库复制到另一个数据库中。主从复制，是将数据库分为主节点和从节点，主节点源源不断地将数据复制给从节点，保证主从节点中存有相同的数据。有了主从复制,数据可以有多份副本，这带来了多种好处：

第—，提升数据库系统的请求处理能力。单个节点能够支撑的读流量有限，部多个节点，并构成主从关系，用主从复制保持主从节点数据一致，如此主从节点一起提供服务。

第二,提升整个系统的可用性。因为从节点中有主节点数据的副本,当主节点宕机后，可以立刻提升其中一个从节点为主节点，继续提供服务

二、Redis 主从复制原理

主从复制，直观的做法是主节点产生一份数据的快照发送给从节点，以快照数据为基准，将之后增量的数据变更同步给从节点，如此就能保证主从数据的一致。总体来看，主从复制一般包含全量数据同步、增量同步两个阶段。

在 Redis的主从复制实现中，包含两个类似阶段：全量数据同步和命令传播。

全量数据同步：主节点产生一份全量数据的快照，即RDB文件，并将此快照发送给从节点。且从产生快照时刻起，记录新接收到的写命令。当快照发送完成后，将累积的写命令发送给从节点，从节点执行这些写命令。此时基准已经建立完成。

·命令传播：全量数据同步完成后，主节点将执行过的写命令源源不断地发送给从节点，从节点执行这些命令，保证主从节点中数据有相同的变更，如此保证主从节点数据的一致。

下图中给出了主从复制的整个过程：

1、主从关系建立后，从节点向主节点发送一个 SYNC 命令请求进行主从同步。

2、主节点收到 SYNC 命令后，执行 fork 创建一个子进程，子进程将所有的数据编码存储到 RDB（ Redis Database ）文件中，这就产生了数据库的快照。

3、主节点将此快照发送给从节点，从节点接收快照并载入。

4、主节点接着将生成快照、发送快照期间积压的命令发送给从节点。

5、此后，主节点源源不断地新执行的写命令同步到从节点，从节点执行传播来的命令，命令执行后，从库中的数据也就得到了更新。如此，主从数据保持一致。需要说明的是，命令传播存在时延的，所以任意时刻，不能保证主从节点间数据完全一致。

---

redis和memcached的区别

1、Redis和Memcache都是将数据存放在内存中，都是内存数据库。不过memcache还可用于缓存其他东西，例如图片、视频等等；

2、Redis不仅仅支持简单的k/v类型的数据，同时还提供list，set，hash等数据结构的存储；

3、虚拟内存–Redis当物理内存用完时，可以将一些很久没用到的value 交换到磁盘；

4、过期策略–memcache在set时就指定，例如set key1 0 0 8,即永不过期。Redis可以通过例如expire 设定，例如expire name 10；

5、分布式–设定memcache集群，利用magent做一主多从;redis可以做一主多从。都可以一主一从；

6、存储数据安全–memcache挂掉后，数据没了；redis可以定期保存到磁盘（持久化）；

7、灾难恢复–memcache挂掉后，数据不可恢复; redis数据丢失后可以通过aof恢复；

8、Redis支持数据的备份，即master-slave模式的数据备份；

	Memcache	Redis
数据类型	简答 key/value 结构	key/value，list，set，hash，sorted
持久性	不支持	支持
分布式存储	不支持	多种方式，主从、Sentinel、Cluster 等
多线程支持	支持	不支持
内存管理	有	无
事物支持	不支持	有限支持

Redis的启动流程

1.初始化server变量，设置redis相关的默认值

2.读入配置文件，同时接收命令行中传入的参数，替换服务器设置的默认值

3.初始化服务器功能模块。在这一步初始化了包括进程信号处理、客户端链表、共享对象、初始化数据、初始化网络连接等

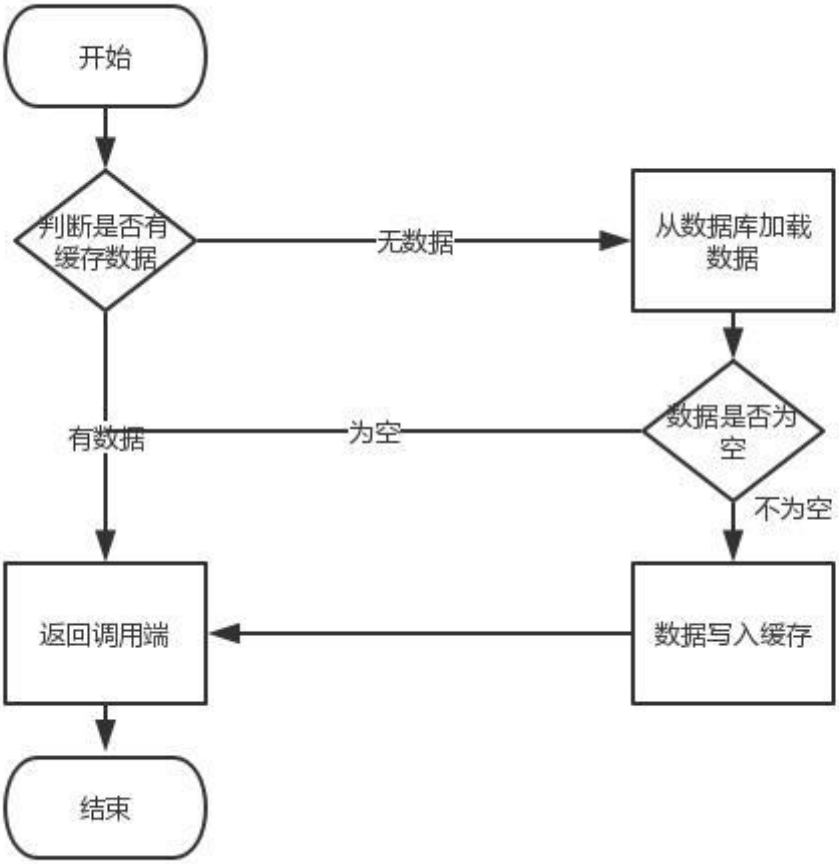
4.从RDB或AOF重载数据

5.网络监听服务启动前的准备工作

6.开启事件监听，开始接受客户端的请求

启动的部分过程通过查看下图，会更直观。

redis中缓存与数据库中的数据同步



这张图，大多数人的很多业务操作都是根据这个图来做缓存的。但是一旦设计到双写或者数据库和缓存更新等操作，就很容易出现数据一致性的问题。无论是先写数据库，在删除缓存，还是先删除缓存，在写入数据库，都会出现数据一致性的问题。列举两个小例子。

1、 先删除了redis缓存，但是因为其他什么原因还没来得及写入数据库，另外一个线程就来读取，发现缓存为空，则去数据库读取到之前的数据并写入缓存，此时缓存中为脏数据。

2、 如果先写入了数据库，但是在缓存被删除前，写入数据库的线程因为其他原因被中断了，没有删除掉缓存，就也会出现数据不一致的情况。

总的来说，写和读在多数情况下都是并发的，不能绝对保证先后顺序，就会很容易出现缓存和数据库数据不一致的情况，还怎么解决呢？

1、方案一：采用延时双删策略

基本思路： 在写库前后都进行删除缓存操作，并且设置合理的超时时间

基本步骤： 先删除缓存–再写数据库—休眠一段时间—再次删除缓存

注：休眠的时间是根据自己的项目的读数据业务逻辑的耗时来确定的。这样做主要是为了保证在写请求之前确保读请求结束，写请求可以删除读请求造成的缓存脏数据。

该方案的弊端： 集合双删策略+缓存超时策略设置，这样最差的结果就是在超时时间内数据存在不一致，又增加了写请求的耗时。

2、方案二：一步更新缓存（基于订阅Binlog的同步机制）

基本思路： mysql Binlog增强订阅消费+消息队列+增量数据更新到redis—读redis：热数据基本上都在redis—写mysql：增删改都是操作mysql—更新redis数据：mysql的数据操作Binlog，来更新redis

我们再来看看详细的过程

1、Redis更新

1）、数据操作主要分为两大块：

一个是全量，将全部数据写去redis；另一个就是增量（update、insert、delete），实时更新。

2）、读取binlog后分析 ，利用消息队列,推送更新各台的redis缓存数据。

这样一旦MySQL中产生了新的写入、更新、删除等操作，就可以把binlog相关的消息推送至Redis，Redis再根据binlog中的记录，对Redis进行更新。

其实这种机制，很类似MySQL的主从备份机制，因为MySQL的主备也是通过binlog来实现的数据一致性。

redis事务

1、Redis事务的概念：

Redis 事务的本质是一组命令的集合。事务支持一次执行多个命令，一个事务中所有命令都会被序列化。在事务执行过程，会按照顺序串行化执行队列中的命令，其他客户端提交的命令请求不会插入到事务执行命令序列中。

总结说：redis事务就是一次性、顺序性、排他性的执行一个队列中的一系列命令

2、Redis事务没有隔离级别：

批量操作在发送 EXEC 命令前被放入队列缓存，并不会被实际执行，也就不存在事务内的查询要看到事务里的更新，事务外查询不能看到。

3、Redis不保证原子性：

Redis中，单条命令是原子性执行的，但事务不保证原子性，且没有回滚。事务中任意命令执行失败，其余的命令仍会被执行。

4、Redis事务的三个阶段：

（1）开始事务

（2）命令入队

（3）执行事务