Redis只能做缓存？太out了！

漫话编程 3天前

以下文章来源于小姐姐味道 ， 作者小姐姐养的狗

小姐姐味道
不羡鸳鸯不羡仙，一行代码调半天

大多数数据库，由于经常和磁盘打交道，在高并发场景下，响应会非常的慢。为了解决这种速度差异，大多数系统都习惯性的加入一个缓存层，来加速数据的读取。redis由于它优秀的处理能力和丰富的数据结构，已经成为了事实上的分布式缓存标准。

但是，如果你以为redis只能做缓存的话，那就太小看它了。

redis丰富的数据结构，使得它的业务使用场景非常广泛，加上rdb的持久化特性，它甚至能够被当作落地的数据库使用。在这种情况下，redis能够撑起大多数互联网公司，尤其是社交、游戏、直播类公司的半壁江山。

1. Redis能够胜任存储工作

redis提供了非常丰富的集群模式： `主从` 、 `哨兵` 、 `cluster` ，满足服务高可用的需求。同时，redis提供了两种持久化方式： `aof` 和 `rdb` ，常用的是rdb。

通过 `bgsave` 指令，主进程会fork出新的进程，回写磁盘。bgsave相当于做了一个快照，由于它并没有WAL日志和checkpoint机制，是无法做到实时备份的。如果机器突然断电，那就很容易丢失数据。

幸运的是，redis是内存型的数据库，主从同步的速度是非常快的。如果你的集群维护的好，内存分配的合理，那么除非机房断电，否则redis的SLA，会一直保持在非常高的水平。

img

听起来不是绝对可靠啊，有丢失数据的可能！这在一般CRUD的业务中，是无法忍受的。但为什么redis能够满足大多数互联网公司的需求？这也是由业务属性所决定的。

在决定最大限度拥抱redis之前，你需要确认你的业务是否有以下特点：

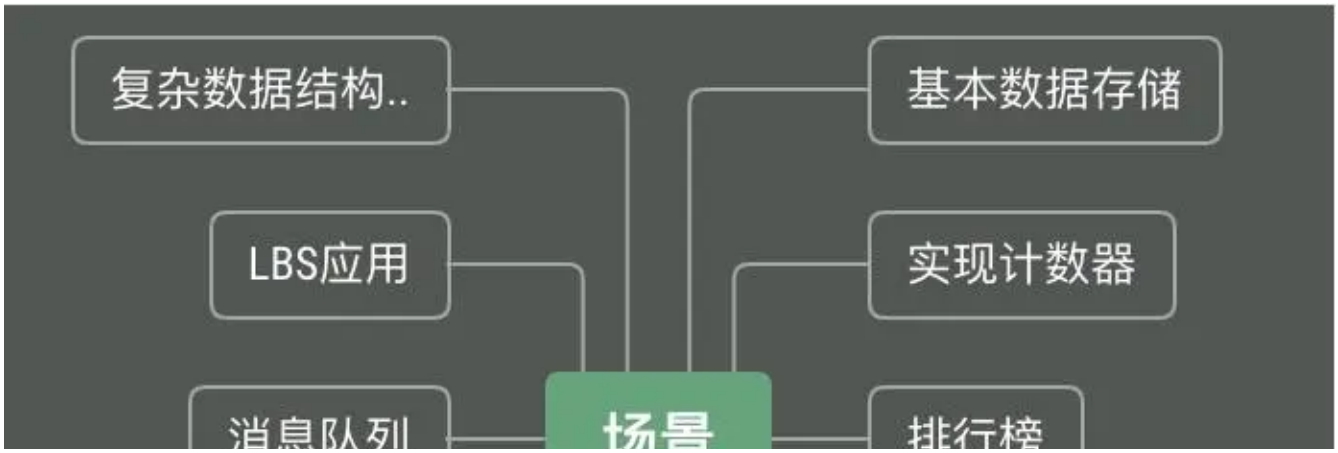
除了核心业务，是否大多数业务对于数据的可靠性要求较低，丢失一两条数据是可以忍受的？

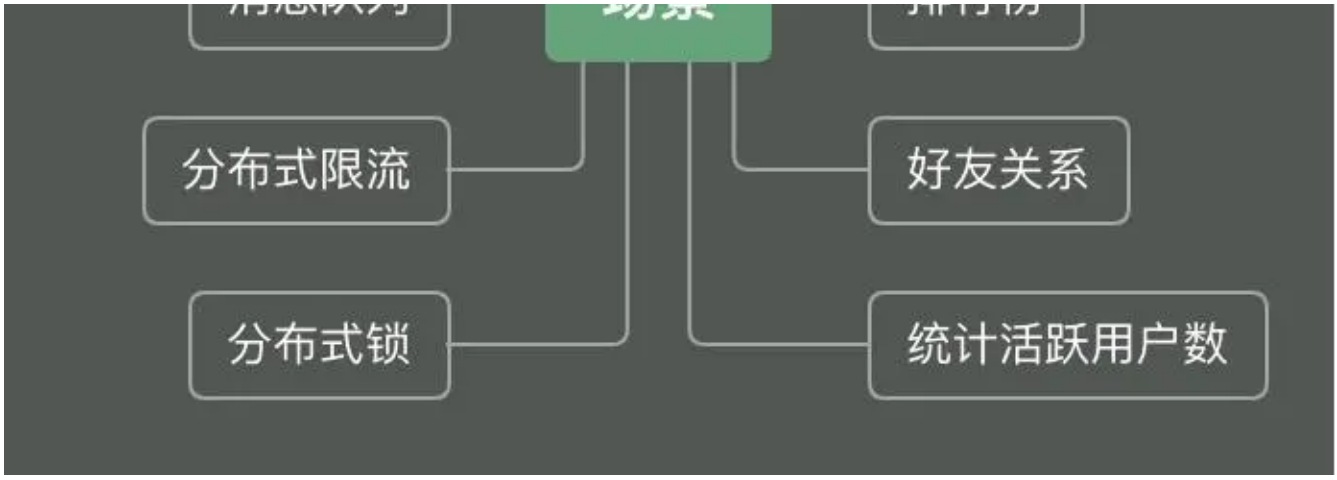
1. 面对的是C端用户，可根据用户ID快速定位到一类数据，数据集合普遍较小？无大量范围查询需求？
2. 是否能忍受内存型数据的成本需求？
3. 是否业务几乎不需要事务操作？

很幸运的是，这类业务需求特别的多。比如常见的社交，游戏、直播、运营类业务，都是可以完全依赖Redis的。

2. Reids应用场景

Redis具有松散的文档结构，丰富的数据类型，能够适应千变万化的scheme变更需求，接下来我将介绍Redis除缓存外的大量的应用场景。





img

2.1 基本用户数据存储

在传统的数据库设计中，用户表是非常难以设计的，变更的时候会伤筋动骨。使用Redis的 `hash` 结构，可以实现松散的数据模型设计。某些不固定，验证型的功能属性，可以以JSON接口直接存储在hash的value中。使用hash结构，可以采用HGET和HMGET等指令，只获取自己所需要的数据，在使用上也是非常便捷的。

```
>HSET user:199929 sex m
>HSET user:199929 age 22
>HGETALL user:199929
1) "sex"
2) "m"
3) "age"
4) "22"
```

这种非统计型的、读多写少的场景，是非常适合使用KV结构进行存储的。Redis的hash结构提供了非常丰富的指令，某个属性也可以使用 `HINCRBY` 进行递增递减，非常的方便。

2.2 实现计数器

上面稍微提了一下HINCRBY指令，而对于Redis的Key本身来说，也有 `INCRBY` 指令，实现 某个值 的递增递减。

比如以下场景：统计某个帖子的点赞数；存放某个话题的关注数；存放某个标签的粉丝数；存储一个大体的评论数；某个帖子热度；红点消息数；点赞、喜欢、收藏数等。

```
> INCRBY feed:e3kk38j4kl:like 1
> INCRBY feed:e3kk38j4kl:like 1
```

```
> GET feed:e3kk38j4kl:like
"2"
```

像微博这样容易出现热点的业务，传统的数据库，肯定是撑不住的，就要借助于内存数据库。由于Redis的速度非常快，就不用再采用传统DB非常慢的 `count` 操作，所有这种递增操作都是毫秒级别的，而且效果都是实时的。

2.3 排行榜

排行榜能提高参与者的积极性，所以这项业务非常常见，它本质上是一个topn的问题。

Redis中有一个叫做zset的数据结构，使用跳表实现的有序列表，可以很容易实现排行榜一类的问题。当存入zset中的数据，达到千万甚至是亿的级别，依然能够保持非常高的并发读写，且拥有非常棒的平均响应时间（5ms以内）。

使用 `zadd` 可以添加新的记录，我们会使用排行相关的分数，作为记录的score值，然后使用 `zrevrange` 指令即可获取实时的排行榜数据，而 `zrevrank` 则可以非常容易的获取用户的实时排名。

```
>ZADD sorted:xjjdog:2021-07  55 dog0
>ZADD sorted:xjjdog:2021-07  89 dog1
>ZADD sorted:xjjdog:2021-07  32 dog2
>ZCARD sorted:xjjdog:2021-07
>3
> ZREVRANGE  sorted:xjjdog:2021-07  0 -10 WITHSCORES # top10排行榜
1) "dog1"
2) "89"
3) "dog0"
4) "55"
5) "dog2"
6) "32"
```

2.4 好友关系

`set` 结构，是一个没有重复数据的集合，你可以将某个用户的关注列表、粉丝列表、双向关注列表、黑名单、点赞列表等，使用独立的zset进行存储。

使用 `ZADD` 、 `ZRANK` 等，将用户的黑名单使用ZADD添加，ZRANK使用返回的sorce值判断是否存在黑名单中。使用 `sinter` 指令，可以获取A和B的共同好友。

除了好友关系，有着明确黑名单、白名单业务场景的数据，都可以使用set结构进行存储。这种业务场景还有很多，比如某个用户上传的通讯录，计算通讯录的好友关系等等。

在实际使用中，使用zset存储这类关系的更多一些。zset同set一样，都不允许有重复值，但zset多了一个score字段，我们可以存储一个时间戳，用来标明关系建立所发生的时间，有更明确的业务含义。

2.5 统计活跃用户数

类似统计每天的活跃用户、用户签到、用户在线状态，这种零散的需求，实在是太多了。如果为每一个用户存储一个bool变量，那占用的空间就太多了。这种情况下，我们可以使用 `bitmap` 结构，来节省大量的存储空间。

```
>SETBIT online:2021-07-23 3876520333 1
>SETBIT online:2021-07-24 3876520333 1
>GETBIT online:2021-07-23 3876520333
1
>BITOP AND active online:2021-07-23 online:2021-07-24
>GETBIT active 3876520333
1
>DEBUG OBJECT online:2021-07-23
Value at:0x7fdfde438bf0 refcount:1 encoding:raw serializedlength:5506446 lru:16410558 lru
(0.96s)
```

注意，如果你的id很大，你需要先进行一次预处理，否则它会占用非常多的内存。

bitmap包含一串连续的2进制数字，使用1bit来表示真假问题。在bitmap上，可以使用and、or、xor等位操作(`bitop`)。

2.6 分布式锁

Redis的分布式锁，是一种轻量级的解决方案。虽然它的可靠性比不上Zookeeper之类的系统，但Redis分布式锁有着极高的吞吐量。

一个最简陋的加锁动作，可以使用redis带nx和px参数的set指令去完成。下面是一小段简单的分布式样例代码。

```
public String lock(String key, int timeOutSecond) {
    for (; ; ) {
        String stamp = String.valueOf(System.nanoTime());
        boolean exist = redisTemplate.opsForValue().setIfAbsent(key, stamp, timeOutSecon
        if (exist) {
            return stamp;
        }
    }
}
public void unlock(String key, String stamp) {
    redisTemplate.execute(script, Arrays.asList(key), stamp);
}
```

删除操作的lua为。

```
local stamp = ARGV[1]
local key = KEYS[1]
local current = redis.call("GET",key)
if stamp == current then
    redis.call("DEL",key)
    return "OK"
end
```

redisson的RedLock，是使用最普遍的分布式锁解决方案，有读写锁的差别，并处理了多redis实例情况下的异常问题。

2.7 分布式限流

使用计数器去实现简单的限流，在Redis中是非常方便的，只需要使用incr配合expire指令即可。

```
incr key
expire key 1
```

这种简单的实现，通常来说不会有问题，但在流量比较大的情况下，在时间跨度上会有流量突然飙升的风险。根本原因，就是这种时间切分方式太固定了，没有类似滑动窗口这种平滑的过度方案。

同样是redisson的RRateLimiter，实现了与 `guava` 中类似的分布式限流工具类，使用非常便捷。下面是一个简短的例子：

```
RRateLimiter limiter = redisson.getRateLimiter("xjjdogLimiter");
// 只需要初始化一次
// 每2秒钟5个许可
limiter.trySetRate(RateType.OVERALL, 5, 2, RateIntervalUnit.SECONDS);

// 没有可用的许可，将一直阻塞
limiter.acquire(3);
```

2.8 消息队列

redis可以实现简单的队列。在生产者端，使用LPUSH加入到某个列表中；在消费端，不断的使用RPOP指令取出这些数据，或者使用阻塞的BRPOP指令获取数据，适合小规模的抢购需求。

Redis还有PUB/SUB模式，不过pubsub更适合做消息广播之类的业务。

在Redis5.0中，增加了stream类型的数据结构。它比较类似于Kafka，有主题和消费组的概念，可以实现多播以及持久化，已经能满足大多数业务需求了。

2..9 LBS应用

早早在Redis3.2版本，就推出了GEO功能。通过 `GEOADD` 指令追加lat、lng经纬数据，可以实现坐标之间的距离计算、包含关系计算、附近的人等功能。

关于GEO功能，最强大的开源方案是基于PostgreSQL的PostGIS，但对于一般规模的GEO服务，redis已经足够用了。

2.10 更多扩展应用场景

要看redis能干什么，就不得不提以下java的客户端类库redisson。redisson包含丰富的分布式数据结构，全部是基于redis进行设计的。

redisson提供了比如Set、 SetMultimap、 ScoredSortedSet、 SortedSet, Map、 ConcurrentMap、 List、 ListMultimap、 Queue、BlockingQueue等非常多的数据结构，使得基于redis的编程更加的方便。在github上，可以看到有上百个这样的数据结构：https://github.com/redisson/redisson/tree/master/redisson/src/main/java/org/redisson/api。

对于某个语言来说，基本的数组、链表、集合等api，配合起来能够完成大部分业务的开发。Redis也不例外，它拥有这些基本的api操作能力，同样能够组合成分布式的、线程安全的高并发应用。

由于Redis是基于内存的，所以它的速度非常快，我们也会把它当作一个中间数据的存储地去使用。比如一些公用的配置，放到redis中进行分享，它就充当了一个配置中心的作用；比如把JWT的令牌存放到Redis中，就可以突破JWT的一些限制，做到安全登出。

3. 一站式Redis面临的挑战

redis的数据结构丰富，一般不会在功能性上造成困扰。但随着请求量的增加，SLA要求的提高，我们势必会对Redis进行一些改造和定制性开发。

3.1 高可用挑战

redis提供了主从、哨兵、cluster等三种集群模式，其中cluster模式为目前大多数公司所采用的方式。

但是，redis的cluster模式，有不少的硬伤。redis cluster采用虚拟槽的概念，把所有的key映射到 0～16383个整数槽内，属于无中心化的架构。但它的维护成本较高，slave也不能够参与读取操作。

它的主要问题，在于一些批量操作的限制。由于key被hash到多台机器上，所以mget、hmset、sunion等操作就非常的不友好，经常发生性能问题。

redis的主从模式是最简单的模式，但无法做到自动failover，通常在主从切换后，还需要修改业务代码，这是不能忍受的。即使加上haproxy这样的负载均衡组件，复杂性也是非常高的。

哨兵模式在主从数量比较多的时候，能够显著的体现它的价值。一个哨兵集群，能够监控成百上千个集群，但是哨兵集群本身的维护是比较困难的。幸运的是，redis的文本协议非常简单，在netty中，甚至直接提供了redis的codec。自研一套哨兵系统，加强它的功能，是可行的。

3.2 冷热数据分离

redis的特点是，不管什么数据，都一股脑地搞到内存里做计算，这对于有时间序列概念，有冷热数据之分的业务，造成了非常大的成本考验。为什么大多数开发者喜欢把数据存放在MySQL中，而不是Redis中？除了事务性要求以外，很大原因是历史数据的问题。

通常，这种冷热数据的切换，是由中间件完成的。我们上面也谈到了，Redis是一个文本协议，非常简单。做一个中间件，或者做一个协议兼容的Redis模拟存储，是比较容易的。

比如我们Redis中，只保留最近一年的活跃用户。一个好几年不活跃的用户，突然间访问了系统，这时候我们获取数据的时候，就需要中间件进行转换，从容量更大，速度更慢的存储中查找。

这个时候，Redis的作用，更像是一个热库，更像是一个传统cache层做的事情，发生在业务已经上规模的时候。但是注意，直到此时，我们的业务层代码，一直都是操作的redis的api。它们使用这众多的函数指令，并不关心数据到底是真正存储在redis中，还是在ssdb中。

3.3 功能性需求

redis还能玩很多花样。举个例子，全文搜索。很多人都会首选es，但redis生态就提供了一个模块：RediSearch，可以做查询，可以做filter。

但我们通常还会有更多的需求，比如统计类、搜索类、运营效果分析等。这类需求与大数据相关，即使是传统的DB也不能胜任。这时候，我们当然要把redis中的数据，导入到其他平台进行计算啦。

如果你选择的是redis数据库，那么dba打交道的，就是rdb，而不是binlog。有很多的rdb解析工具(比如redis-rdb-tools)，能够定期把rdb解析成记录，导入到hadoop等其他平台。

此时，rdb成为所有团队的中枢，成为基本的数据交换格式。导入到其他db后的业务，该怎么玩怎么玩，完全不会因为业务系统选用了redis就无法运转。

4. 总结

大多数业务系统，跑在redis上，这是很多一直使用MySQL做业务系统的同学所不能想象的。看完了上面的介绍，相信你能够对redis能够实现的存储功能有个大体的了解。打开你的社交app、游戏app、视频app，看一下它们的功能，能够涵盖多少呢？

我这里要强调的是，某些数据，并不是一定要落地到RDBMS才算安全，它们并不是一个强需求。

那既然redis这么厉害，为什么还要有mysql、tidb这样的存储呢？关键还在于业务属性上。

如果一个业务系统，每次交互的数据，都是一个非常大的结果集，并涉及到非常复杂的统计、过滤工作，那么RDBMS是必须的；但如果一个系统，能够通过某个标识，快速定位到一类数据，这一类数据在可以预见的未来，是有限的，那就非常适合Redis 存储 。

一个电商系统，选用redis做存储就是作死，但一个社交系统就快活的多。在合适的场景选用合适的工具，才是我们应该做的。

但是一个系统，能否在产品验证期，就能快速的响应变化，快速开发上线，才是成功的关键。这也是使用redis做数据库，所能够带来的最大好处。千万别被那概率极低的丢数据场景，给吓怕了。比起产品成功，你的系统即使是牢如钢铁，也一文不值。

推荐阅读：

- 活久见！国内首款 18 禁游戏，来了！
- B 站 CEO 的身份证被上传到 GitHub 了？
- 刚刚！996违法！人社部、最高法最新官宣
- 干掉 Postman？测试接口直接生成API文档，这个工具我爱了

❤️喜欢我可以给我设为星标哦❤️