Redis

说了那么多AVL树，B树，B+树，红黑树什么之类的，优化来优化去只不过是常数上的变化，对性能不会有太大的影响。因为这个时候我们已经遇到了另一个严重的瓶颈，就是IO瓶颈，为了解决这样的问题，Redis是如何破局的?

简介

百科上说，这玩意是一个性能极高的key-value数据库。我们说的什么Oracle数据库，MySQL数据库，SQL Server不香吗，丰富而且严谨的关系型数据库不好吗，为啥非得搞这样一个限制很大的数据库呢?存储的东西只能是键值对，而且还不能范围查询，这是什么辣鸡玩意?

没错，我一开始也是这么以为的，因为我觉得Redis的限制太大了，你一个Key就只能有一个value，那我想像关系型数据库一样存那么多字段怎么办，难道我每次存取都要序列化和反序列化，这样不是没事找事?而且最麻烦的是，范围查询直接就GG，哈希表里何来的范围查询，这对于我们这种查询密集的业务场景很明显是不好的嘛。好了，吐槽到此结束，下面我们好好来解释一下为什么Redis好。

Redis的底层实现

源码级的东西我就不多说了，往大了说，Redis依赖的数据库就是一个巨大的哈希表。众所周知，哈希表的查询复杂度是O(1)的，那么理论上，无论数据库里存了多少的数据，性能表现应该差异不会太大。事实上，Redis官方给出的性能是，每秒11万次查询和8万次写入。wdnmd的太厉害了吧?一秒钟几万次查询，你他妈逗我呢?MySQL能一秒钟几百次都已经不得了了，你一个这么轻量的玩意能扛住一秒几万次?对不起，哈希表就真的可以为所欲为。

说到哈希，没办法忽略哈希碰撞的问题嘛，我们之前说到有非常多的解决方案，比如说二次探测，或者在节点处延伸一个List出去等等。其实我们可以先从源头做起，我们如何选择一个算法，使得它哈希之后更难碰撞?

首先先得说一下哈希之后的结果是什么，理论上，为了让计算机不会发生溢出和超出寻址空间，计算出的哈希值应该是一个unsigned int类型（这个叫做一致性哈希算法，默认会对2^32-1取模，这里不铺开来说），这玩意有42亿个，足足够用了，没谁那么无聊把这么多的数据存进去数据库，什么电脑都得炸。我们这个时候来挖一下HashCode的计算方法:

```
public static int hashCode(byte[] value) {
        int h = 0;
        int length = value.length >> 1;
        for (int i = 0; i < length; i++) {
            h = 131 * h + getChar(value, i);
        }
        return h;
    }
```

其实简单一句话说就是将字符串里面的字符以131进制求和，即hash=hash*131+value[i]。关于其他的字符串哈希的算法，可以去看看这个博客，还是讲的很清楚的。

```
https://www.cnblogs.com/zl1991/p/11820922.html
```

然而，Redis在发生哈希碰撞之后，Redis的处理办法是将它们延伸出一个链表，然后当链表长到一定地步之后，触发rehash过程。这一点上，Redis并没有借用Java HashMap的实现方法（哈希表+红黑树，当碰撞数量大于8并且哈希表中的节点数多于64时，会触发将列表转化为红黑树）Redis的解决办法简单粗暴，这样其实更有利于持久化的维护，如果哈希方法不当导致红黑树长得很大之后，这样的哈希表性能也非常不好，这样子长痛不如短痛，直接rehash结束会更好。如果想看更多的底层设计，可以看看这两篇博客：

```
https://www.cnblogs.com/gaopengfirst/p/10062980.html
https://www.jianshu.com/p/658365f0abfc
```

当然，有了良好的哈希策略和处理方式还不够，Redis真正甩开其他非关系型数据库的地方在于，它的数据是储存在内存里的。学过计算机组成原理都知道，内存的读写性能远远比磁盘快，所以这也就是为啥Redis的性能可以如此恐怖。但是，内存终究是易失性的，到底数据库如何保证安全和数据持久化？后面说……这里不剧透了。

对比传统的关系型数据库？

传统的关系型数据库主要依靠B树和B+树来建立。至于为什么选用的不是红黑树之类的二叉树，我之前在数据库那篇文章里说过，这里就不多说了。

Redis除了性能强速度快之外，它还兼有非关系型数据库的所有优点。其中一个很典型的就是，数据的组织会非常灵活。因为它不会再受制于任何的表间关系和数据类型等等。

就举我自己的例子吧，第一次动摇了我对关系型数据库的看法是在一个游戏后台设计中，以吃鸡为例子，一个用户他含有这些字段：金币数量，XX皮肤，XX枪械，最高击杀记录，爆头率，KD，段位，分数……你要我说再讲上百个不是问题。假设我在一张表里想存下来这些数据，那岂不是需要几百个字段？这个时候，我发现MySQL的最大限制是255个字段，实在没办法，另外开一张表来存，然后一个外键连过来父表。其实，这两张表在逻辑上有没有必要呢？完全没有。

而且问题更加大的是，张三拥有火箭少女皮肤，李四没有，那么我就要建立一个字段叫做“火箭少女皮肤”，张三的该字段为true，李四为false。试想一下，李四没有的东西，还有必要存进来吗？你可能觉得，每个人多一个字节存一个boolean而已嘛，多大回事呢。然而，如果有几十亿用户呢？有好几十个这样的字段呢？这就是巨大的空间浪费了。

最后一个致命的东西，就是新增与删除字段了。比如说新出了八神庵和魔法少女皮肤，我就得在表里再加两个字段，然后这个时候发现，我第二张表也够255个字段了，又TM要新开一张表。这还没完，你居然要修改生产环境中的数据库，这个是企业开发的大忌！就算你可以修改，一个有几十亿数据，并且还带有外键的表，没个几天你弄得完？（粗略估计，这样的数据规模下，一秒钟能更新几百千把条数据已经不错了，你自己算算得等多少秒）而且更新数据库的时候，所有的服务肯定要降级或者停止的，造成的损失怎么承担？

在这样的应用场景下，传统的数据库已经没有丝毫用武之地了，被按在地上狠狠摩擦。当然，我不是说传统的数据库就该死，只是他们的应用场景不同而已，如果需要范围查询，Redis你再牛也得扑街。当然，后面我会说到两种数据库如何取长补短，各自发挥优势。

怎么用？

Redis中有五种数据类型，分别是字符串（其实就是字节流，这个String二进制安全）、哈希类型、列表、集合、有序集合。至于这五种数据结构具体是什么，代码怎么用，我贴一个链接出来，可以看视频学。再说下去……我估计又可以出一篇文章了。

```
https://www.bilibili.com/video/BV1Cb411j7RA
https://www.bilibili.com/video/BV1DV411o7Ny
```

我前面吐槽的传统数据库不灵活的问题，可以被一个哈希类型横扫（第一个链接里有讲），每个用户可以拥有不同的字段，不同的数据，其灵活程度极高，我当时学到这个的时候惊叹于这样优美的解决方案，实在厉害。

问题？解决策略？

Redis的问题也是很显而易见的，咱们挨个来谈谈怎么解决。

1，内存？

我前面说过，Redis把数据存在了内存里，企图用内存强大的读写能力来换取性能。这样的危害是明显的，第一个，内存占用很恐怖，第二个，一旦服务器宕机，数据全部扑街。

内存占用有多离谱，我就给一个例子吧，我现在运行的服务里，大概几十万行数据，依靠Redis做一个缓存，就要消耗接近0.5G的内存，这还是在业务不密集的时候，一旦业务密集，服务器直接报警。那么也就是说，如果内存不宽裕，基本上Redis就别想，即便内存很宽裕，Redis仍然还需要调校才可以使用，否则一样会出现溢出的问题。那么这个时候我们必须做一件事情，对内存里的对象进行处理，热点对象继续留在内存里，冷对象就塞进去磁盘里，不要在内存里霸地方，所以说这里又涉及到数据的甄别和标定了。这里不展开讨论。

服务器宕机之后，内存里的数据岂不是全部完蛋？没错，确实是完蛋了，所以这个时候，我们需要让Redis做持久化，也就是把数据写入磁盘，一般而言，可以使用定时快照或者当修改量多到一定程度的时候，触发一次持久化。然而，终究这样是有时间差的，还不是很安全，所以这个时候我们已经不可以单纯依靠一个实例来解决问题了，这个时候就需要Redis集群和异地容灾来帮助我们了。

2，KV键值对的模型是不是太局限了？

说得太对了，一个key只有一个value，不能重复。那么如果对于有多个信息的人或者模型，Redis就很难受了。好比一个学生，今天在学校犯了两个违纪，被记录下来了，那么我还能用学生id做主键吗？显然后来的数据会覆盖前面的数据，这就变得很麻烦了。

所以说……其实我们也应该公正看待关系型和非关系型数据库，现在没有绝对好的技术，也没有绝对完美的东西。Redis有它的灵活和高性能，关系型数据库有它的查询优势，两个东西应该互相补充而不是说XX吊打XX（我承认我前面的说辞有误导你们认为非关系数据库就无敌的嫌疑，但是还是希望大家客观对待）

3，缓存穿透、缓存雪崩

其实直接拿Redis做数据库是不太好的，就单单因为它易失性的问题，就足以枪毙它了，但是，Redis如此强大的性能，让我们不免想到一个妙用，就是缓存。比如说我要查询一个数据，如果这个数据在Redis里面有，那直接从Redis里面拿肯定更快，如果Redis里面没有，就先去MySQL里查，查到了之后再放进去Redis里面，这样下次查就快了。

看着很完美，我们不得不解决几个问题：1，不常用的数据也在占内存，是不是该想好后面怎么处理？2，怎么保证MySQL里的数据和Redis的数据是一致的？3，老是查询到Redis里没有的数据，咋办？4，Redis炸了之后，所有的请求冲进去MySQL里，把MySQL也整炸了，咋办？

前两个问题咱们不考虑，因为其实都可以在应用端解决，和数据库本身没什么太大关系。我们考虑第3，4个问题，它们分别对应着我的标题“缓存穿透”和“缓存雪崩”。

缓存穿透很好理解，就是老是查到Redis里没缓存的数据，导致这个缓存没什么卵用的问题。当然，大部分时候下，我们可以通过优化筛选热点数据的算法，或者加入启动时冷加载（就是说在应用启动的时候，就先往Redis里填充一点数据，避免刚开始的时候Redis命中率非常低的问题）的办法来优化。但是有一个东西是很要命的——空值。假设我们在一堆用户里查询一个并不存在的用户，Redis没找到，于是进去MySQL里查，MySQL也没查到，所以并不可以更新Redis里面的缓存，然后下次如果还是查这个不存在的用户，那就还是那么慢，甚至乎，无效的查询可能会触发数据库全表扫描，然后这么多请求丢过去，直接把数据库炸了，这可怎么办？当然，有一个很粗暴的办法，MySQL没查到的时候，把这个null值写进去Redis里面，这样就解决问题了呗！但是，有一堆的key存着空值，这不又是空间浪费吗？这个时候我们需要引入一个新的概念叫做布隆过滤器。布隆过滤器可以一定程度上，帮我们筛掉这些无效的查询，具体布隆过滤器是什么东西，请看这些链接：

```
https://www.jianshu.com/p/2104d11ee0a2
https://www.cnblogs.com/CodeBear/p/10911177.html
https://www.cnblogs.com/qdhxhz/p/11237246.html
```

缓存雪崩其实算是一种灾难，它的成因主要就是Redis缓存意外失效。由于Redis很可能处理着巨量的查询和写入请求，一旦这个缓存失效，所有的查询直接冲进去MySQL，然后瞬间MySQL就瘫痪了。往往缓存雪崩的时候，主库都会跟着炸，因为关系型数据库中没有任何一个有能力负担如此巨大的查询量。解决这个问题，往往我们也需要依赖分布式技术，集群技术等等，这个已经不在我们的讨论范围里了。

应用场景?

单纯拿Redis做数据库是不现实的……除非这个公司真的够头铁，毕竟我们说过，光是断电丢数据这一条就足够枪毙它了。那么Redis能拿来干嘛？其实我上面在说缓存穿透和缓存雪崩的时候已经说了出来了，没错，就是缓存。

Redis用作缓存已经是一个非常非常广泛的应用了，比如说阿里P8的技术，腾讯云，京东云等等，全部都是重度依赖Redis（或者它的衍生产物，比如说Tair数据库）不然，它们根本就没法承受像双十一，限时秒杀之类的巨量访问。当然，我做的产品里，也有使用Redis来加速访问。

所以，现在的大型项目，架构都大致如此：Redis做缓存，用于快速查询；关系型数据库负责保证数据一致性与普通查询，同时，关系型数据库还可以完成很多的范围查询的请求。当然，一切设计都得从功能出发。

总结

夸了Redis一整篇了，我们是时候客观地说说总结了。其实，我们不应该在这个时候说Redis无敌之类的话，只能说，不同的数据库应用的场景并不相同，我们并不能说谁就一定好或者不好。比如传统的关系型数据库，就可以很方便地处理范围查询，加和，平均，最大最小值的问题，而Redis在这个时候就会扑街，因为两个本来相邻的元素，被哈希之后，早就已经不知道去哪了，我想查个2020-04-28至2020-04-29的交易记录都必须要全表扫描。但是关系型数据库不灵活的问题也时时刻刻阻碍着业务的调整，这一点上非关系型数据库的优势又体现出来了。

所以，针对不同的需求，使用不同的技术，才是最最正确的解决方式。本没有最优秀的技术一说，只有合不合适的说法。