Redis 进阶 -- 企业级解决方案

作者：sirwsl
原文链接：https://ld246.com/article/1592967733093
来源网站：链滴
许可协议：署名-相同方式共享 4.0 国际 (CC BY-SA 4.0)

redis基础理论写的差不多了，今天来个最后的，用作之后的复习使用，之后开始学习其他的东西了，了开始搞起来，下文主要讲述有：**缓存预热、缓存雪崩、缓存击穿、缓存穿透、性能指标监控。随总结，**用作日后复习，失眠随便看看

一、缓存预热

现象： 服务器启动后快速宕机

原因： 请求数据量比较大、主从之间数据吞吐量较大、数据同步操作频度较高

分析： 由于刚刚启动服务器，redis中没有数据，这时候redis之间需要进行大量的同步信息，这时候外界频繁的请求数据，就很容易出现服务器宕机

解决方案：

- **启动前准备工作**
  - 日常进行统计数据访问记录，统计访问频度较高的热点数据利用LRU数据删除策略，构建数据留队列（eg：storm与kafka配合）
  - 手动去维护
  - **目的**：在重启之间可以保证哪些数据是属于比较高的热点数据
- **准备工作：**
  - 将统计结果的数据进行分类，更具级别，redis优先加载级别较高的热点数据
  - 利用分布式多服务器间进行数据读取，提取数据加载过程
  - **目的**：在服务器启动的时候提前将一些热点数据加载到redis中，保证服务器在加载数据时候有定的热点数据，避免应用服务器过来请求数据的时候没有数据或者是不经常用的数据，同时也能够使使得加载数据时候速度快一些。
- **实施：**
  - 使用脚本程序固定出发数据预热过程
  - 如果条件允许，使用了CDN（内容分发网络）效果会更好

总结： 缓存预热就是系统启动前将相关的缓存数据直接加载到缓存系统，避免在用户请求的时候，reds中数据为0，造成先查询数据库，然后再数据缓存，造成redis压力较大，使得cpu内存较快的达到峰。从而做到用户直接查询事先被预热的缓存数据！

二、缓存雪崩

现象： 系统平稳运行过程中，忽然数据库连接量激增（连接比较多），造成应用服务器无法及时处理求，出现大量408、500错误页面，客户反复刷新页面但是没有获取数据。从而造成数据库崩溃、应服务器奔溃，我们在重启应用服务器无效、Redis服务器崩溃、Redis集群奔溃、重启数据库后再次被间流量放倒。

原因： 在一个较短的时间内，缓存中较多的key集中过期、造成在一定周期内请求访问过期的数据，rdis未命中，redis向数据库获取数据。使得数据库同时接收到大量的请求无法进行及时处理，Redis大请求被积压，开始出现超时现象，数据库流量激增，数据库奔溃（重启数据库没有作用），重启后仍面对缓存中无数据可用造成Redis服务器资源被严重占用，Redis服务器奔溃，Redis集群呈现崩溃，群瓦解，应用服务器无法及时得到数据响应的请求，来自客户端的请求数量越来越多，使得服务器崩

，应用服务器，redis，数据库全部重启，效果不理想。

分析：短时间内大量的key过期

解决方案：

- 更多的页面进行静态化处理
  - 构建多级缓存架构：nginx缓存+redis缓存+ehcache缓存
  - 检测mysql严重耗时的业务进行优化
  - 对数据库的瓶颈排查：例如超时查询、耗时较高事务等
  - 灾难预警机制：监控redis服务器性能指标（ **CPU占用、CPU使用率、内存容量**、查询平均响时间、线程数）
  - 限流、降级：短时间内牺牲一些用户体验、限制一部分请求、降低应用服务器的压力，待业务速运转后再逐步放开访问
- **争对过期方案：**
  - LRU与LFU切换
  - 数据有限期策略调整：根据业务数据有效期进行分类错峰（eg：A类90分钟、B类80分类、C类7分钟）
  - 过期时间使用固定时间+随机值的心是，释放计中到期的key数量
  - 超热数据使用永久key
  - 定期维护（自动+人工）：对即将过期的数据做访问量分析，确认是否延迟，配合访问量统计，热点数据的延迟
  - 加锁（慎用）

总结：

缓存雪崩就是瞬间的过期数据量太大，导致对数据库的服务器造成压力，如果能够有效避免过期时间集中，就可以有效解决雪崩现象的出现（约40%）配合其他策略一起使用，并监控服务器的运行数据更具运行记录做出快速调整。

三、缓存击穿

现象： 数据库服务器奔溃

系统平稳运行的过程中，数据库连接量瞬间激增、Redis服务器无大量key过期，Redis内存平稳运行波动，同时Reids服务器CPU正常但数据库崩溃。

原因： Redis中某个key过期，改key访问量巨大。同时多个数据请求从服务器直接压到Redis后均未中，从而造成Redis再短时间引发大量对数据库中同一数据的访问。

分析： 单个key高热且redis过期

解决方案：

- 通过预先设定（eg：以电商为例，每个商家都会有自己的店铺等级、指定若干款主打商品，再双十时候，加大此类商品信息的过期时间）
- 现场调整：监控访问量，对自然流量激增的数据延长过期时间或者设置为永久key
- 后台刷新数据：定期启动任务，高峰期来临之间，刷新数据的有效期，确保不会丢失

- 二级缓存：设置不同的失效时间，保障不会被同时淘汰
- 加锁：分布式锁，防止被击穿，但要注意性能（慎用）

总结： 缓存击穿就是单个高热数据过期的瞬间，数据访问量较大，未命中Redis后，发起了大量对同数据的数据库访问，导致数据库的压力过大，应对策略应该在业务分析与预热方面进行，配合运行监测试与及时调整策略，毕竟单个key的过期监控难度较高。

四、缓存穿透

现象： 数据库服务器奔溃

系统在平稳运行中，应用服务器流浪随时间而增大，Redis服务器命中率随时间逐步降低，但是Redis存平稳运行，内存无压力，但Redis服务器CPU占用增加，数据库服务器压力增加，最终数据库服务崩溃。

原因： Redis中大面积出现未命中，出现非正常的URL访问

分析： 获取的数据在数据库中不存在，数据库查询未得到相应的数据，Redis获取到null数据未进行久化直接返回，造成下次此类数据重复这个过程（黑客攻击服务器）

解决方案：

- 缓存null：对数据库查询结果为null的数据库进行缓存（长期使用、定期清理），设定短时间30-12秒
- 白名单策略：提前预热各种分类数据id对应的bitmaps，id作为bitmaps的offset，相当于设置了数库的白名单，在正常加载数据库的时候放行，加载异常时直接拦截（每次访问都需要加载bitmaps，似于加了一个拦截器，造成效率偏低）
- 使用布隆过滤器（有关布隆过滤器的命中问题对当前的状况可以直接忽略（此处不做讲解））
- 实时监控：实时监控Redis命中率（业务范围内。通常会有一个波动值）于null数据的占比

非活动时段波动：通常检测3-5倍，超过5倍纳入重点排查对象

活动时段：通常检测5-10倍，超过50倍纳入重点排查对象

根据倍数的不同，启动不同的排查流程，然后使用黑名单进行防控

- key加密：问题出现后，启动临时防灾业务key，对key进行业务岑创数加密服务，设定校验层序，过来的key校验（eg：每天随机60个加密串，挑选2-3个，混淆在页面id中，发现访问的key不满足规时直接驳回访问）

总结：

缓存击穿范文不存在的数据，条幅哦了合法数据的redis数据缓存阶段，每次访问数据库导致对数据库务器压力变大，通常此类数据的出现量是一个较低的值，当出现此类情况以毒攻毒并及时报警，应对略应该在临时预防时多加注意。在实时监控时，无论是黑名单还是白名单，都是对整体提供的压力，报解除后要尽快移除。

五、性能指标监控

性能指标：

Name	Description
latency	Redis响应一个请求的时间

instantaneous_ops_sec	平均每秒处理请求总数
hit rate(calculated)	缓存命中率（计算出来的）

内存指标：

Name	Description
used_memory	已用内存
mem_fragmentation_ratio	内存碎片率
evicted_keys	由于最大内存限制倍移除的key的量
clocked_clients	由于BLPOP,BRPOP,BRPOPLPUH而备阻塞的客户端

基本活动指标：

Name	Description
connected_clients	客户端的连接数
connected_slaves	Slave数量
master_last_io_seconds_ago	最近一次主从交之后的秒数
keyspace	数据库中key值总数

持久性指标：

Name	Description
rdb_last_save_time	最后一次持久化保存到磁盘时间戳
rdb_changes_since_last_ave	自最后一次持久化赖数据库的更改数

错误指标：

Name	Description
rejected_connections	由于达到了maxclient限而被拒绝的链接数
keyspace_misses	Key值查找失败的次数
master_link_down_since_seconds	主从断开的续时间（单位：s）

监控方式：

- **工具：**

Cloud Insight Redis

Prometheus

Redis-stat

Redis-faina

RedisLive

zabbix

- **命令：**

benchmark 压测：

redis-cli monitor|showlog

redis-benchmark [-h] [-p] [-n <requests> [-k]]

eg:

redis-benchmark -c 100 -n 5000 #100个连接，5000次请求对应的性能

monitor打印服务器调试信息

```
redis-cli
monitor
```

slowlog 日志：

```
slowlog [operator]   #get：获取慢查询日志 len：获取慢查询日志条目  reset：重置慢查询日志
slowlog get          #eg:get
```

配置：

```
slowlog-log-slower-then 1000 #设置慢查询的时间下线。单位微秒
slowlog-max-len 100 #设置慢查询命令对应的日志显示长度，单位：命令数
```