Redis开发规范

Redis集群申请

Redis架构组推荐的redis 配置

func DefaultRedisClient(consulName string) (\*RedisClient, error) {

   opt := goredis.NewOption()

   opt.IdleTimeout = time.*Minute*

opt.SetMaxRetries(maxRetry)

   return NewRedisClientWithOption(consulName, opt)

}

架构选型

1. 双机房还是单主架构

目前双机房架构存在数据不一致的场景：

1. 逐出 → redis自发逐出，不会同步到对端机房，可能造成一边有这个key，一边没有。
2. 过期 →  一边过期了，另一边还未过期，造成一边有这个key，一边没有。
3. update操作（如incr、zadd等）异步写失败，可能造成两边都有，但value不一致

**所以：**

* 1. 如果系统对数据的一致性要求比较高，建议单主架构
  2. 如果服务对缓存的压力并没有太大的时候，可以选择单主架构
  3. 如果一个单主架构无法承担主数据访问压力，可以考虑对数据拆分，将数据分向不同实例

1. 存储还是缓存
   1. 缓存和存储的区别在于，存储是会做数据存储的，缓存也没有做数据存储。缓存实例重启后redis中是没有数据的，存储实例重启后会加载之前已经持久化的数据。当采用存储实例后，公司的配置是只采用AOF一种持久化方式。根据cache组同学的反馈，ev组目前的业务数据对redis实例的压力不大，所以采取缓存型和存储型差别不大。但是从长远角度，还是应该根据自己的业务采取不同的实例类型
   2. 目前ev的redis实例是和其他业务的redis实例共用机器，这样存在一个隐患，如果其他业务出现io密集的情况，如果ev的redis实例采用的是存储型，恰好这个节点出现并发高峰，是可能出现超时的情况的。

Key-Value设计

**1. key名设计**

* (1)【建议】: 可读性和可管理性

以业务名(或数据库名)为前缀(防止key冲突)，用冒号分隔，比如业务名:表名:id

ugc:video:1

* (2)【建议】：简洁性

保证语义的前提下，控制key的长度，当key较多时，内存占用也不容忽视，例如：

user:{uid}:friends:messages:{mid}简化为u:{uid}:fr:m:{mid}。

* (3)【强制】：不要包含特殊字符

反例：包含空格、换行、单双引号以及其他转义字符

**2. value设计**

* (1)【强制】：拒绝bigkey(防止网卡流量、慢查询)

string类型控制在10KB以内，hash、list、set、zset元素个数不要超过5000。

反例：一个包含200万个元素的list。

非字符串的bigkey，不要使用del删除，使用hscan、sscan、zscan方式渐进式删除，同时要注意防止bigkey过期时间自动删除问题(例如一个200万的zset设置1小时过期，会触发del操作，造成阻塞，而且该操作不会不出现在慢查询中(latency可查))，查找方法和删除方法

* (2)【推荐】：选择适合的数据类型。

例如：实体类型(要合理控制和使用数据结构内存编码优化配置,例如ziplist，但也要注意节省内存和性能之间的平衡)

反例：

set user:1:name tomset user:1:age 19set user:1:favor football

正例:

hmset user:1 name tom age 19 favor football

3.【推荐】：控制key的生命周期，redis不是垃圾桶。

建议使用expire设置过期时间(条件允许可以打散过期时间，防止集中过期)，不过期的数据重点关注idletime。

命令使用

1.【推荐】 O(N)命令关注N的数量

例如hgetall、lrange、smembers、zrange、sinter等并非不能使用，但是需要明确N的值。有遍历的需求可以使用hscan、sscan、zscan代替。

【建议】 单个请求的返回值控制100KB之内；

2.【推荐】：禁用命令

禁止已禁用使用keys、flushall、flushdb等；有变量需求通过使用scan的方式渐进式处理。

3.【推荐】不要使用select

redis的多数据库较弱，使用数字进行区分，很多客户端支持较差，同时多业务用多数据库实际还是单线程处理，会有干扰。

当前提供的集群不支持多个数据库；

4.【推荐】使用批量操作提高效率

可以使用pipeline提高效率。有mget场景尽量改用pipeline（mget对proxy的消耗比较大）

但要注意控制一次批量操作的元素个数(例如500以内，实际也和元素字节数有关)。

5.【建议】Redis事务功能较弱，不建议使用

我们提供了proxy不支持redis事务功能；Redis的事务功能较弱(不支持回滚)，如果非得使用只能直连使用；

6.【建议】必要情况下使用monitor命令时，要注意不要长时间使用。（在平台上有支持，平台管理页面的faina统计使用的是monitor命令）

服务使用

1.【推荐】

避免多个应用使用一个Redis实例

正例：不相干的业务拆分，公共数据做服务化。

2.【建议】

根据自身业务类型，选好maxmemory-policy(最大内存淘汰策略)，设置好过期时间。

默认策略是volatile-lru，即超过最大内存后，在过期键中使用lru算法进行key的剔除，保证不过期数据不被删除，但是可能会出现OOM问题。

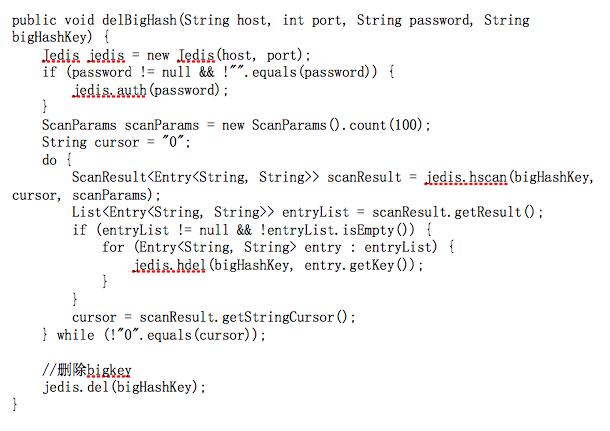
其他策略如下：

* allkeys-lru：根据LRU算法删除键，不管数据有没有设置超时属性，直到腾出足够空间为止。
* allkeys-random：随机删除所有键，直到腾出足够空间为止。
* volatile-random:随机删除过期键，直到腾出足够空间为止。
* volatile-ttl：根据键值对象的ttl属性，删除最近将要过期数据。如果没有，回退到noeviction策略。
* noeviction：不会剔除任何数据，拒绝所有写入操作并返回客户端错误信息"(error) OOM command not allowed when used memory"，此时Redis只响应读操作。

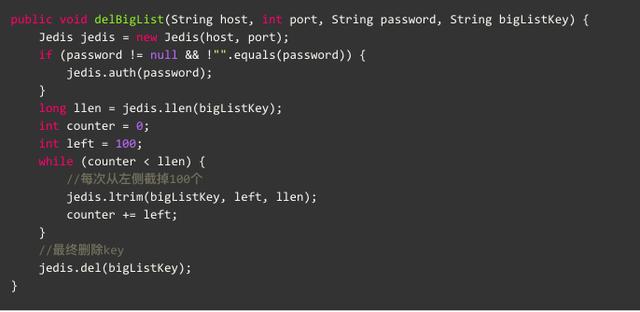
附录：删除bigkey

下面操作可以使用pipeline加速。

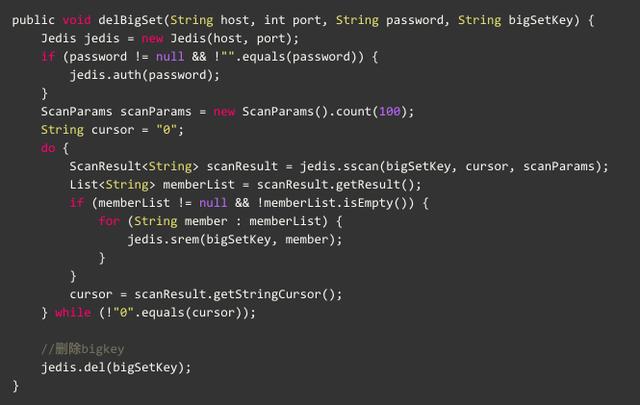
1. Hash删除: hscan + hdel



2. List删除: ltrim



3. Set删除: sscan + srem



4. SortedSet删除: zscan + zrem



Redis更新

监控报警

2020.1.15 架构侧沟通结论同步

[Abase介绍](https://docs.google.com/presentation/d/1M5JoD-sNlSJUGfx-_mIy9buSUtD2XQEFO-wI7gCKOtk/edit" \l "slide=id.p" \t "/Users/yangwenshuo/Documents\\x/_blank)

* redis的基础数据结构都支持，value建议不超过5M
* 支持scan row
* 存储型数据库，支持大容量、大吞吐量，面向最终一致性场景
* 支持abase->redis(正在做）abase->hive的同步
* 支持快照回档
* 正常网络情况下，主从延迟5ms内

Abase和Redis选型

* Abase是存储型场景，和存储型redis支持的操作基本相同，具体的选择根据细节需求，支持的操作等细化选择。如果需要的是一个持久化的存储，那么abase应该要考虑下。
* Abase可支持小容量场景，1T以下
* 缓存需求用缓存型Redis（缓存场景尽量不要申请存储型Redis)

Redis偶现的失败，怎么定位和解决

* ​[Redis Client SDK的报错](https://bytedance.feishu.cn/docs/doccn2pl6Qj8um199MEb8rBoPkb" \t "/Users/yangwenshuo/Documents\\x/_blank)​
* 架构侧会考虑教育的业务场景，针对性优化
* 从业务侧角度，建议所有的请求都加上重试。（EV场景请求加重试对架构来说没有风险）
* 因为EV流量不均匀，一段时间没有请求时，会导致redis的连接池中的连接失效，建议进程有一个日常的set操作，持续的set值到redis中，相当于keep\_alive （这个具体看下是否方便实现，或者写在common库中也行）

RC架构下，hscan有问题

* 后续有问题再反馈给redis同学，他们继续追查。

数据强一致性需求，分布式锁等场景

* redis/abase都是最终一致性，所以这种场景不能用redis，可以选择bytekv，zk，etcd

Redis数据迁移，比如拆分集群等

* 目前正在做，可能这个双月支持。后续abase/redis之间的数据同步组件都会支持

Redis开发规范

Redis集群申请

Redis架构组推荐的redis 配置

func DefaultRedisClient(consulName string) (\*RedisClient, error) {

   opt := goredis.NewOption()

   opt.IdleTimeout = time.*Minute*

opt.SetMaxRetries(maxRetry)

   return NewRedisClientWithOption(consulName, opt)

}

架构选型

1. 双机房还是单主架构

目前双机房架构存在数据不一致的场景：

1. 逐出 → redis自发逐出，不会同步到对端机房，可能造成一边有这个key，一边没有。
2. 过期 →  一边过期了，另一边还未过期，造成一边有这个key，一边没有。
3. update操作（如incr、zadd等）异步写失败，可能造成两边都有，但value不一致

**所以：**

* 1. 如果系统对数据的一致性要求比较高，建议单主架构
  2. 如果服务对缓存的压力并没有太大的时候，可以选择单主架构
  3. 如果一个单主架构无法承担主数据访问压力，可以考虑对数据拆分，将数据分向不同实例

1. 存储还是缓存
   1. 缓存和存储的区别在于，存储是会做数据存储的，缓存也没有做数据存储。缓存实例重启后redis中是没有数据的，存储实例重启后会加载之前已经持久化的数据。当采用存储实例后，公司的配置是只采用AOF一种持久化方式。根据cache组同学的反馈，ev组目前的业务数据对redis实例的压力不大，所以采取缓存型和存储型差别不大。但是从长远角度，还是应该根据自己的业务采取不同的实例类型
   2. 目前ev的redis实例是和其他业务的redis实例共用机器，这样存在一个隐患，如果其他业务出现io密集的情况，如果ev的redis实例采用的是存储型，恰好这个节点出现并发高峰，是可能出现超时的情况的。

Key-Value设计

**1. key名设计**

* (1)【建议】: 可读性和可管理性

以业务名(或数据库名)为前缀(防止key冲突)，用冒号分隔，比如业务名:表名:id

ugc:video:1

* (2)【建议】：简洁性

保证语义的前提下，控制key的长度，当key较多时，内存占用也不容忽视，例如：

user:{uid}:friends:messages:{mid}简化为u:{uid}:fr:m:{mid}。

* (3)【强制】：不要包含特殊字符

反例：包含空格、换行、单双引号以及其他转义字符

**2. value设计**

* (1)【强制】：拒绝bigkey(防止网卡流量、慢查询)

string类型控制在10KB以内，hash、list、set、zset元素个数不要超过5000。

反例：一个包含200万个元素的list。

非字符串的bigkey，不要使用del删除，使用hscan、sscan、zscan方式渐进式删除，同时要注意防止bigkey过期时间自动删除问题(例如一个200万的zset设置1小时过期，会触发del操作，造成阻塞，而且该操作不会不出现在慢查询中(latency可查))，查找方法和删除方法

* (2)【推荐】：选择适合的数据类型。

例如：实体类型(要合理控制和使用数据结构内存编码优化配置,例如ziplist，但也要注意节省内存和性能之间的平衡)

反例：

set user:1:name tomset user:1:age 19set user:1:favor football

正例:

hmset user:1 name tom age 19 favor football

3.【推荐】：控制key的生命周期，redis不是垃圾桶。

建议使用expire设置过期时间(条件允许可以打散过期时间，防止集中过期)，不过期的数据重点关注idletime。

命令使用

1.【推荐】 O(N)命令关注N的数量

例如hgetall、lrange、smembers、zrange、sinter等并非不能使用，但是需要明确N的值。有遍历的需求可以使用hscan、sscan、zscan代替。

【建议】 单个请求的返回值控制100KB之内；

2.【推荐】：禁用命令

禁止已禁用使用keys、flushall、flushdb等；有变量需求通过使用scan的方式渐进式处理。

3.【推荐】不要使用select

redis的多数据库较弱，使用数字进行区分，很多客户端支持较差，同时多业务用多数据库实际还是单线程处理，会有干扰。

当前提供的集群不支持多个数据库；

4.【推荐】使用批量操作提高效率

可以使用pipeline提高效率。有mget场景尽量改用pipeline（mget对proxy的消耗比较大）

但要注意控制一次批量操作的元素个数(例如500以内，实际也和元素字节数有关)。

5.【建议】Redis事务功能较弱，不建议使用

我们提供了proxy不支持redis事务功能；Redis的事务功能较弱(不支持回滚)，如果非得使用只能直连使用；

6.【建议】必要情况下使用monitor命令时，要注意不要长时间使用。（在平台上有支持，平台管理页面的faina统计使用的是monitor命令）

服务使用

1.【推荐】

避免多个应用使用一个Redis实例

正例：不相干的业务拆分，公共数据做服务化。

2.【建议】

根据自身业务类型，选好maxmemory-policy(最大内存淘汰策略)，设置好过期时间。

默认策略是volatile-lru，即超过最大内存后，在过期键中使用lru算法进行key的剔除，保证不过期数据不被删除，但是可能会出现OOM问题。

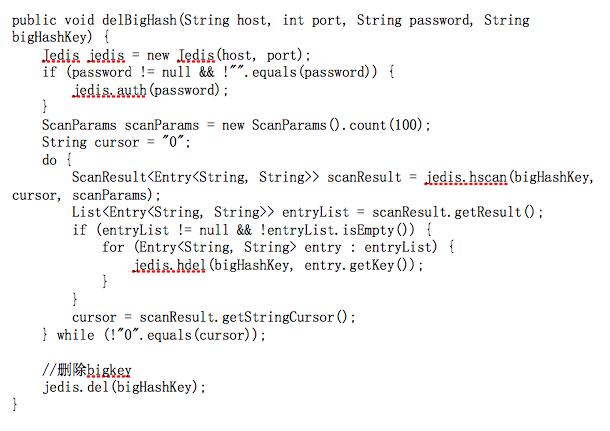
其他策略如下：

* allkeys-lru：根据LRU算法删除键，不管数据有没有设置超时属性，直到腾出足够空间为止。
* allkeys-random：随机删除所有键，直到腾出足够空间为止。
* volatile-random:随机删除过期键，直到腾出足够空间为止。
* volatile-ttl：根据键值对象的ttl属性，删除最近将要过期数据。如果没有，回退到noeviction策略。
* noeviction：不会剔除任何数据，拒绝所有写入操作并返回客户端错误信息"(error) OOM command not allowed when used memory"，此时Redis只响应读操作。

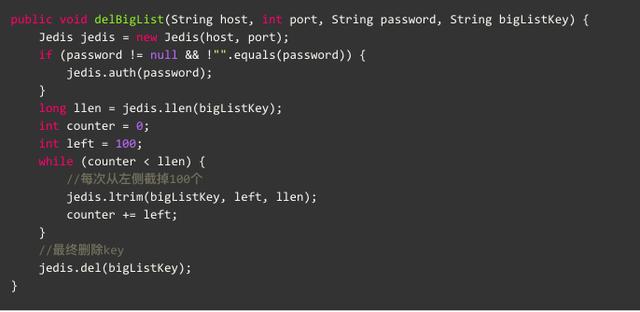
附录：删除bigkey

下面操作可以使用pipeline加速。

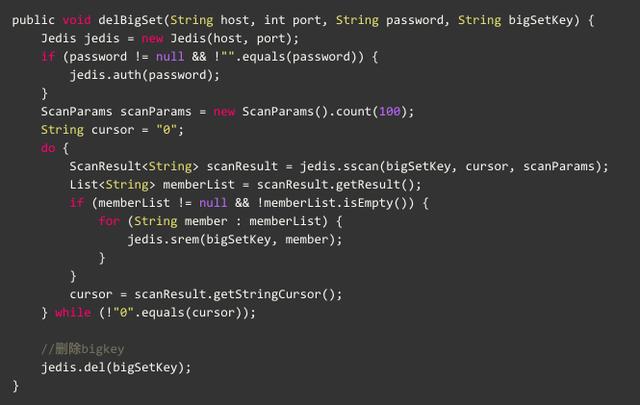
1. Hash删除: hscan + hdel



2. List删除: ltrim



3. Set删除: sscan + srem



4. SortedSet删除: zscan + zrem



Redis更新

监控报警

2020.1.15 架构侧沟通结论同步

[Abase介绍](https://docs.google.com/presentation/d/1M5JoD-sNlSJUGfx-_mIy9buSUtD2XQEFO-wI7gCKOtk/edit" \l "slide=id.p" \t "/Users/yangwenshuo/Documents\\x/_blank)

* redis的基础数据结构都支持，value建议不超过5M
* 支持scan row
* 存储型数据库，支持大容量、大吞吐量，面向最终一致性场景
* 支持abase->redis(正在做）abase->hive的同步
* 支持快照回档
* 正常网络情况下，主从延迟5ms内

Abase和Redis选型

* Abase是存储型场景，和存储型redis支持的操作基本相同，具体的选择根据细节需求，支持的操作等细化选择。如果需要的是一个持久化的存储，那么abase应该要考虑下。
* Abase可支持小容量场景，1T以下
* 缓存需求用缓存型Redis（缓存场景尽量不要申请存储型Redis)

Redis偶现的失败，怎么定位和解决

* ​[Redis Client SDK的报错](https://bytedance.feishu.cn/docs/doccn2pl6Qj8um199MEb8rBoPkb" \t "/Users/yangwenshuo/Documents\\x/_blank)​
* 架构侧会考虑教育的业务场景，针对性优化
* 从业务侧角度，建议所有的请求都加上重试。（EV场景请求加重试对架构来说没有风险）
* 因为EV流量不均匀，一段时间没有请求时，会导致redis的连接池中的连接失效，建议进程有一个日常的set操作，持续的set值到redis中，相当于keep\_alive （这个具体看下是否方便实现，或者写在common库中也行）

RC架构下，hscan有问题

* 后续有问题再反馈给redis同学，他们继续追查。

数据强一致性需求，分布式锁等场景

* redis/abase都是最终一致性，所以这种场景不能用redis，可以选择bytekv，zk，etcd

Redis数据迁移，比如拆分集群等

* 目前正在做，可能这个双月支持。后续abase/redis之间的数据同步组件都会支持