# 在线学习之 tair 简介

jamescao(曹孝卿)*

2017年08月02日
V0.1

*jamescao@tencent.com
本文整理自网络博客

# 目录

# 1 概述

tair是一个类似于 map 的 key/value 结构存储系统（也就是缓存系统），是淘宝的一个开源项目，于 2010 年 6 月 30 号在淘宝开源平台上正式对外开源，不过目前这个项目好像已经没有维护了，主页已经没有任何东西了。tair 具备标准的特性是：高性能、高扩展、高可靠，也就是传说中的三高产品，支持分布式集群部署。官网说目前支持java 和c 这两个版本。适用场景是轻量级缓存应用，是为小文件和零碎文件、固定数据文件做的存储优化。

一般一个 tair 集群主要包括 3 个必选模块：config server、data server 和 client，一个可选模块：invalid server。通常情况下，一个集群中包含 2 台中心控制节点 config server 及多台服务节点 data server。其中的 config server 负责管理所有的 data server，并维护 data server 的状态信息，构建数据在集群中的分布信息（对照表）。为了保证高可用（High Available），config server 可通过 hear beat 以一主一备形式提供服务。data server 则对外提供各种数据存储服务，并以心跳的形式将自身状况汇报给 config server，并按照 config server 的指示完成数据的复制和迁移工作，所有的 data server 地位都是等价的。client 在启动的时候，从 config server 获取数据分布信息，根据数据分布信息和相应的 data server 交互完成用户的请求。invalid server 主要负责对等集群的删除和隐藏操作，保证对等集群的数据一致。

从架构上看，config server 的角色类似于传统应用系统的中心节点，整个集群服务依赖于 config server 的正常工作。但实际上相对来说，tair 的 config server 是非常轻量级的，当正在工作的服务器宕机的时候另外一台会在秒级别时间内自动接管。而且，如果出现两台服务器同时宕机的最恶劣情况，只要应用服务器没有新的变化，tair 依然服务正常。而有了 config server 这个中心节点，带来的好处就是应用在使用的时候只需要配置 config server 的地址（现在可以直接配置 Diamond key），而不需要知道内部节点的情况。如图 1 所示为 tair 集群的架构图。

## 1.1 各模块功能介绍

**config server 的功能**

1) 通过维护和 data server 心跳来获知集群中存活节点的信息；2) 根据存活节点的信息来构建数据在集群中的分布表；3) 提供数据分布表的查询服务；4) 调度 data server 之间的数据迁移、复制。

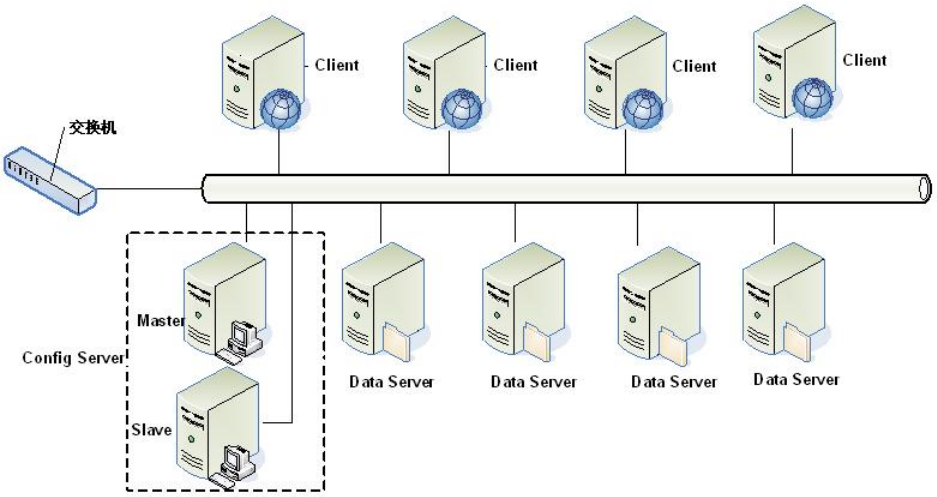


图 1: tair 集群架构图

**data server 的功能**

1) 提供存储引擎；2) 接受 client 的 put/get/remove 等操作；3) 执行数据迁移，复制等；4) 插件：在接受请求的时候处理一些自定义功能；5) 访问统计。

**invalid server 的功能**

1) 接收来自 client 的 invalid/hide 等请求后，对属于同一组的集群（双机房独立集群部署方式）做 delete/hide 操作，保证同一组集群的一致；2) 集群断网之后的，脏数据清理；3) 访问统计。

**client 的功能**

1) 在应用端提供访问 tair 集群的接口；2) 更新并缓存数据分布表和 invalid server 地址等；3) Local Cache，避免过热数据访问影响 tair 集群服务；4) 流控。

## 1.2 一些基本概念

（1）configID：唯一标识一个 tair 集群，每个集群都有一个对应的 configID，在当前的大部分应用情况下 configID 是存放在 diamond 中的，对应了该集群的 config server 地址和 groupname。业务在初始化 tair client 的时候需要配置此 configID。

（2）namespace：又称 area，是 tair 中分配给应用的一个内存或者持久化

存储区域，可以认为应用的数据存在自己的 namespace 中。同一集群（同一个 configID）中 namespace 是唯一的。通过引入 namespace，我们可以支持不同的应用在同集群中使用相同的 key 来存放数据，也就是 key 相同，但 value 不会冲突。一个 namespace 下是如果存放相同的 key，那么 value 会受到影响，在简单 K/V 形式下会被覆盖，rdb 等带有数据结构的存储引擎内容会根据不同的接口发生不同的变化。

（3）quota 配额：对应了每个 namespace 储存区的大小限制，超过配额后数据将面临最近最少使用（LRU）的淘汰。持久化引擎（ldb）本身没有配额，ldb 由于自带了 mdb cache，所以也可以设置 cache 的配额。超过配额后，在内置的 mdb 内部进行淘汰。

（4）expireTime：数据的过期时间。当超过过期时间之后，数据将对应用不可见，不同的存储引擎有不同的策略清理掉过期的数据。调用接口时，expiredTime 单位是秒，可以是相对时间（比如：30s），也可以是绝对时间（比如：当天 23 时，转换成距 1970-1-1 00:00:00 的秒数）。小于 0，不更改之前的过期时间；如果不传或者传入 0，则表示数据永不过期；大于 0 小于当前时间戳是相对时间过期；大于当前时间戳是绝对时间过期。

## 2 存储引擎

tair 分为持久化和非持久化两种使用方式：

（1）非持久化的 tair 可以看成是一个分布式缓存；

（2）持久化的 tair 将数据存放于磁盘中，为了解决磁盘损坏导致数据丢失，tair 可以配置数据的备份数目。tair 自动将一份数据的不同备份放到不同的主机上，当有主机发生异常，无法正常提供服务的时候，其余的备份会继续提供服务。

tair 的存储引擎有一个抽象层，只要满足存储引擎需要的接口，便可以很方便的替换 tair 底层的存储引擎。比如你可以很方便的将 mdb、tc、redis、leveldb 甚至 MySQL 作为 tair 的存储引擎，而同时使用 tair 的分布方式、同步等特性。tair主要有下面三种存储引擎：

（1）mdb：一个高效率的关系型缓存存储数据库，定位于 cache 缓存，类似于 memcache，都存在一样的内存管理方式。支持 k/v 存取和 prefix 操作。在实际业务应用场景中，大部分当缓存用（后端有 DB 之类的数据源），也可

用做大访问少量临时数据的存储（例如 session 登录，防攻击统计等）。阿里巴巴集团内部多数 cache 服务都是采用的 tair mdb。为了减少系统 down 的影响，mdb 采用共享内存的方式，这样当系统 down 后，重新启动后数据还存在。

（2）rdb，定位于 cache 缓存，采用了 redis 的内存存储结构。支持 k/v, list, hash, set, sortedset 等数据结构。适用于需要高速访问某些数据结构的应用，例如 SNS 中常见的的粉丝存储就可以采用 set 等结构，或者存储一个商品的多个属性（hashmap），高效的消息队列（list）等。

（3）ldb，定位于高性能存储，采用了 levelDB 作为引擎，并可选择内嵌 mdb cache 加速，这种情况下 cache 与持久化存储的数据一致性由 tair 进行维护。支持 k/v，prefix 等数据结构。今后将支持 list，hash，set，sortedset 等 redis 支持的数据结构。应用场景为存储，里面可以细分如下场景：1) 持续大数据量的存入读取，类似淘宝交易快照；2) 高频度的更新读取，例如计数器，库存等；3) 离线大批量数据导入后做查询（参见fastdump）。也可以用作 cache，数据量大，响应时间敏感度不高的 cache 需求可以采用，例如天猫实时推荐（why? james ask）。

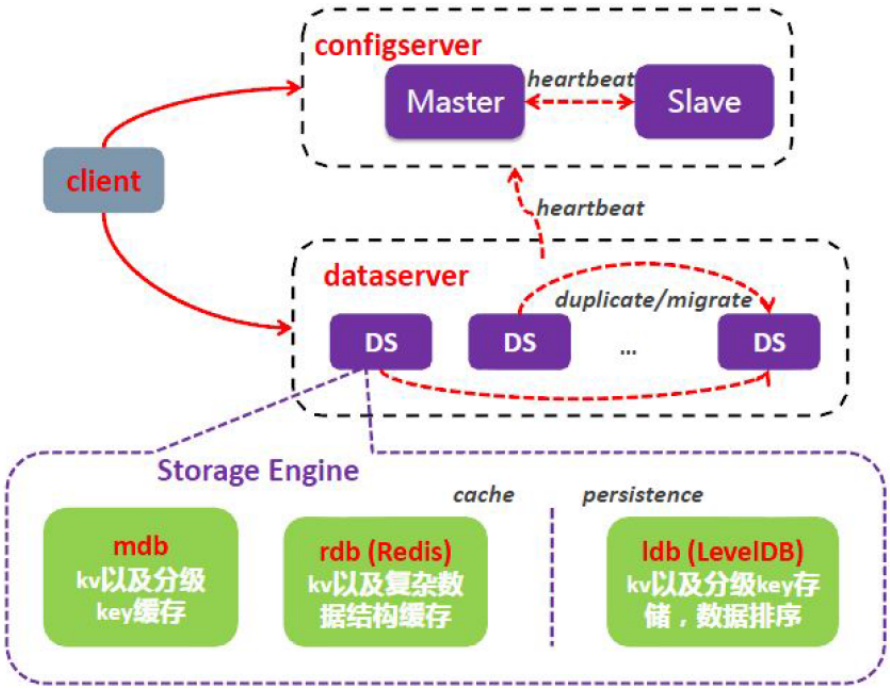


图 2: tair 存储引擎的位置

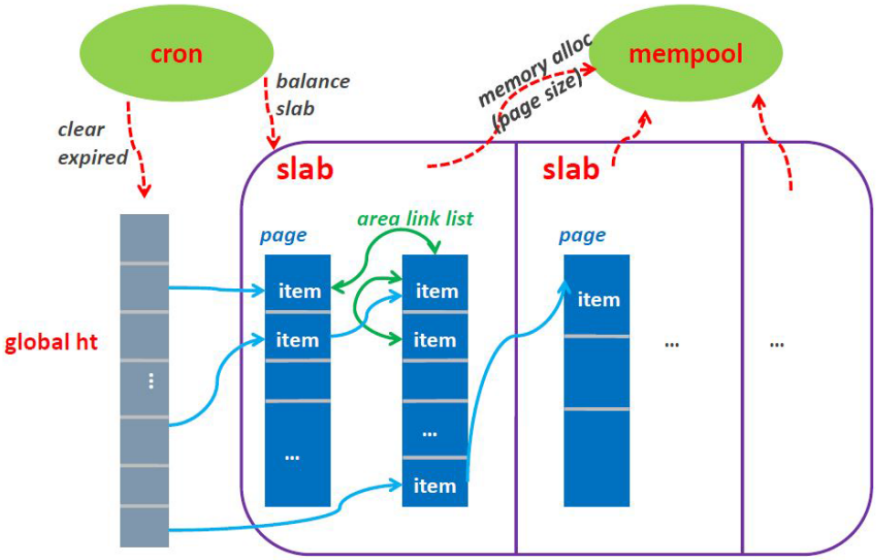


图 3: tair mdb 存储引擎的流程

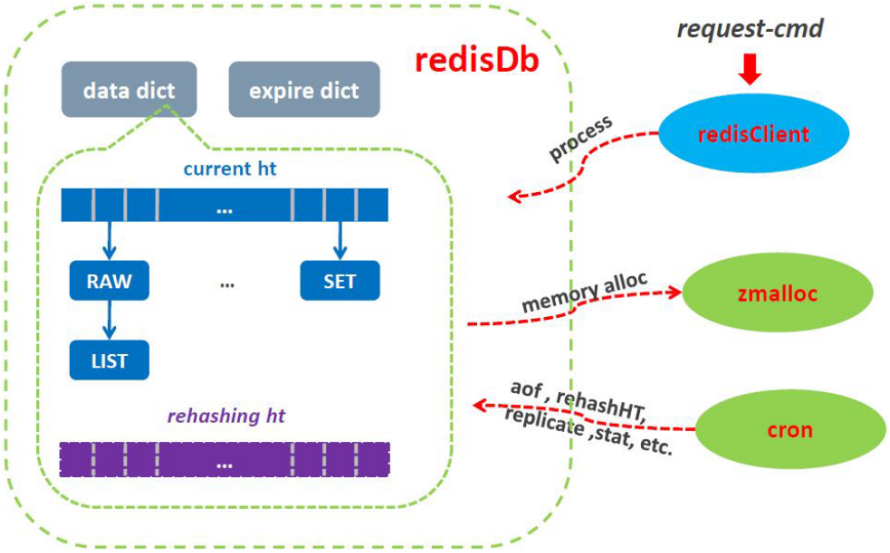


图 4: tair rdb 存储引擎的流程-1

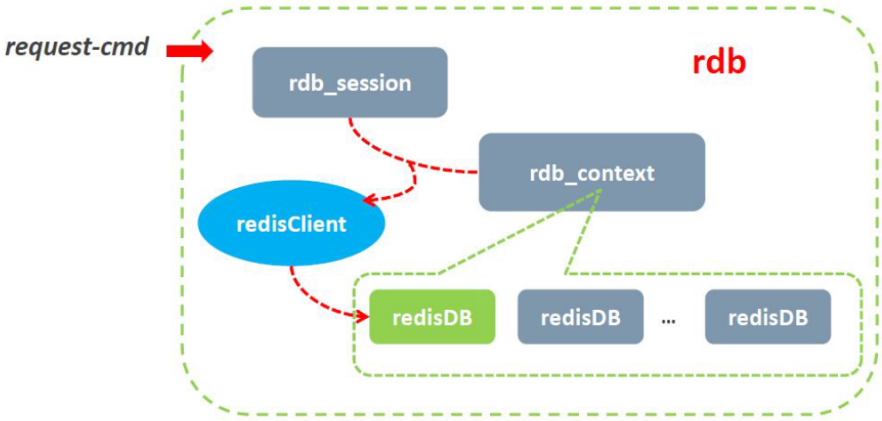


图 5: tair rdb 存储引擎的流程-2

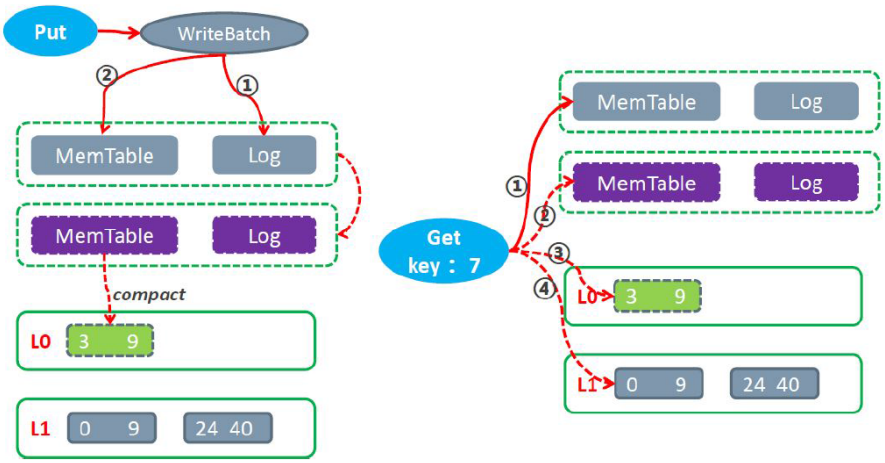


图 6: tair ldb 存储引擎的流程-1

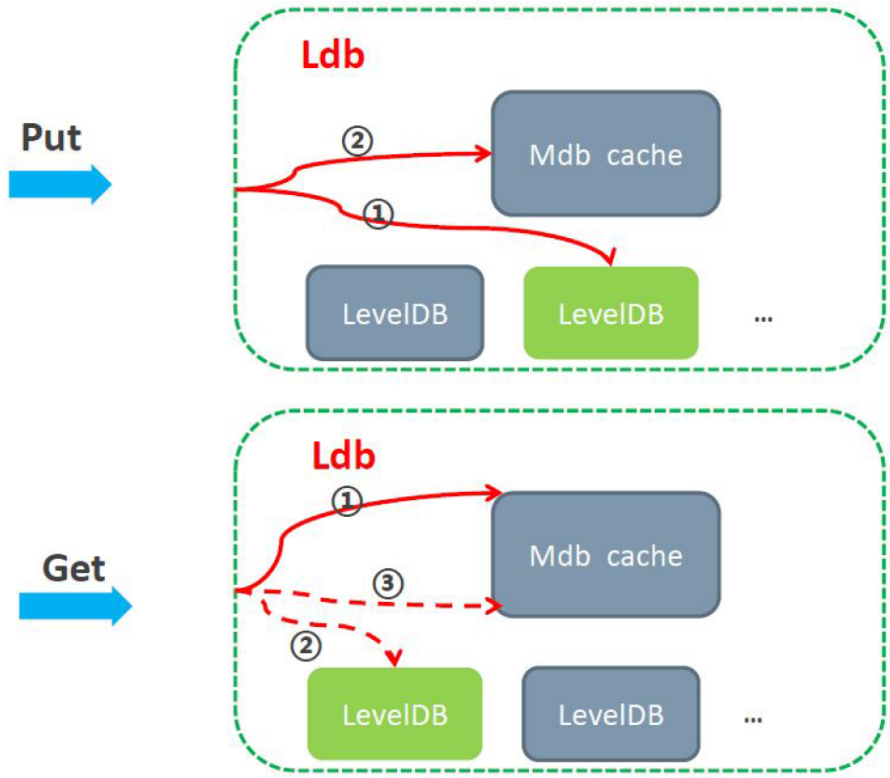


图 7: tair ldb 存储引擎的流程-2

## 3 分布式策略

tair 的分布采用的是一致性哈希算法，用来解决分布式中的平衡性、分散性和一致性。对于所有的 key，分到 Q 个桶中，桶是负载均衡和数据迁移的基本单位。config server 根据一定的策略把每个桶指派到不同的 data server 上，因为数据按照 key 做 hash 算法，所以可以认为每个桶中的数据基本是平衡的，保证了桶分布的均衡性, 就保证了数据分布的均衡性。

具体说，首先计算 Hash(key)，然后通过一些运算，比如取模运算，得到 key 所对应的 bucket，然后再去 config server 查找该 bucket 对应的 data server，再与相应的 data server 进行通信。也就是说，config server 维护了一张由 bucket 映射到 data server 的对照表，如：

```
bucket   data server
0    192.168.10.1
1    192.168.10.2
2    192.168.10.1
3    192.168.10.2
```

```
4     192.168.10.1
5     192.168.10.2
```

这里共 6 个 bucket，由两台机器负责，每台机器负责 3 个 bucket。客户端将 key hash 后，对 6 取模，找到负责的数据节点，然后和其直接通信。表的大小（行数）通常会远大于集群的节点数，这和 consistent hash 中的虚拟节点很相似。假设我们加入了一台新的机器: 192.168.10.3，tair 会自动调整对照表，将部分 bucket 交由新的节点负责，比如新的表很可能类似下表：

```
bucket    data server
0     192.168.10.1
1     192.168.10.2
2     192.168.10.1
3     192.168.10.2
4     192.168.10.3
5     192.168.10.3
```

在老的表中，每个节点负责 3 个桶，当扩容后，每个节点将负责 2 个桶，数据被均衡的分布到所有节点上。如果有多个备份，那么对照表将包含多列，比如备份是为 3，则表有 4 列，后面的 3 列都是数据存储的节点。为了增强数据的安全性，tair 支持配置数据的备份数（COPY COUNT）。比如你可以配置备份数为 3，则每个 bucket 都会写在不同的 3 台机器上。当数据写入一个节点（通常我们称其为主节点）后，主节点会根据对照表自动将数据写入到其他备份节点，整个过程对用户是透明的。

当有新节点加入或者有节点不可用时，config server 会根据当前可用的节点，重新 build 一张对照表。数据节点同步到新的对照表时，会自动将在新表中不由自己负责的数据迁移到新的目标节点。迁移完成后，客户端可以从 config server 同步到新的对照表，完成扩容或者容灾过程。整个过程对用户是透明的，服务不中断。

为了更进一步的提高数据的安全性，tair 的 config server 在 build 对照表的时候，可以配置考虑机房和机架信息。比如你配置备份数为 3，集群的节点分布在两个不同的机房 A 和 B，则 tair 会确保每个机房至少有一份数据。当 A 机房包含两份数据时，tair 会确保这两份数据会分布在不同机架的节点上。这可以防止整个机房发生事故和某个机架发生故障的情况。这里提到的特性需要节

点物理分布的支持，当前是通过可配置的 IP 掩码来区别不同机房和机架的节点。

tair 提供了两种生成对照表的策略：

1、负载均衡优先，config server 会尽量的把桶均匀的分布到各个 data server 上，所谓尽量是指在不违背下面的原则的条件下尽量负载均衡：每个桶必须有 COPY COUNT 份数据以及一个桶的各份数据不能在同一台主机上；

2、位置安全优先，一般我们通过控制 _pos_mask（tair的一个配置项）来使得不同的机房具有不同的位置信息，一个桶的各份数据不能都位于相同的一个位置（不在同一个机房）。

位置优先策略还有一个问题，假如只有两个机房，机房 1 中有 100 台 data server，机房 2 中只有 1 台 data server。这个时候，机房 2 中 data server 的压力必然会非常大，于是这里产生了一个控制参数 _build_diff_ratio（参见安装部署文档），当机房差异比率大于这个配置值时，config server 也不再 build 新表，机房差异比率是如何计出来的呢？首先找到机器最多的机房，不妨设机房 1 的 data server 数量是 $SA$，那么其余的 data server 的数量记做 $SB$，则机房差异比率=$|SA - SB|/SA$，因为一般我们线上系统配置的 COPY COUNT = 3，在这个情况下，不妨设只有两个机房 RA 和 RB，那么两个机房什么样的 data server 数量是均衡的范围呢? 当差异比率小于 0.5 的时候是可以做到各台 data server 负载都完全均衡的。这里有一点要注意，假设 RA 机房有机器 6 台，RB 有机器 3 台，那么差异比率= 6 - 3 / 6 = 0.5，这个时候如果进行扩容，在机房 RA 增加一台 data server，扩容后的差异比率= 7 - 3 / 7 = 0.57，也就是说，只在机器数多的机房增加 data server 会扩大差异比率。如果我们的 _build_diff_ratio配置值是 0.5，那么进行这种扩容后，config server 会拒绝再继续 build 新表。

## 4 一致性和可靠性

分布式系统中的可靠性和一致性是无法同时保证的，因为我们必须允许网络错误的发生。tair 采用复制技术来提高可靠性，并且为了提高效率做了一些优化。事实上在没有错误发生的时候，tair 提供的是一种强一致性，但是在有 data server 发生故障的时候，客户有可能在一定时间窗口内读不到最新的数据，甚至发生最新数据丢失的情况。

## 4.1 version

tair 中的每个数据都包含版本号，版本号在每次更新后都会递增。这个特性可以帮助防止数据的并发更新导致的问题。

那如何获取到当前 key 的 version？

get 接口返回的是 DataEntry 对象，该对象中包含 get 到的数据的版本号，可以通过 getVersion() 接口获得该版本号。

在 put 时，将该版本号作为 put 的参数即可。如果不考虑版本问题，则可设置 version 参数为 0 ，系统将强行覆盖数据，即使版本不一致。

很多情况下，更新数据是先 get，然后修改 get 回来的数据，再 put 回系统。如果有多个客户端 get 到同一份数据，都对其修改并保存，那么先保存的修改就会被后到达的修改覆盖，从而导致数据一致性问题，在大部分情况下应用能够接受，但在少量特殊情况下，这个是我们不希望发生的。

比如系统中有一个值 ”1”, 现在 A 和 B 客户端同时都取到了这个值。之后 A 和 B 客户端都想改动这个值，假设 A 要改成 12，B 要改成 13，如果不加控制的话，无论 A 和 B 谁先更新成功，它的更新都会被后到的更新覆盖。tair 引入的 version 机制避免了这样的问题。刚刚的例子中，假设 A 和 B 同时取到数据，当时版本号是 10，A 先更新，更新成功后，值为 12，版本为 11。当 B 更新的时候，由于其基于的版本号是 10，此时服务器会拒绝更新，返回 VersionError，从而避免 A 的更新被覆盖。B 可以选择 get 新版本的 value，然后在其基础上修改，也可以选择强行更新。

version 改变的逻辑如下：

1、如果 put 新数据且没有设置版本号，会自动将版本设置成 1；

2、如果 put 是更新老数据且没有版本号，或者 put 传来的参数版本与当前版本一致，版本号自增 1；

3、如果 put 是更新老数据且传来的参数版本与当前版本不一致，更新失败，返回 VersionError；

4、如果 put 时传入的 version 参数为 0，则强制更新成功，版本号自增 1。

version 具体使用案例，如果应用有 10 个 client 会对 key 进行并发 put，那么操作过程如下：

1、get key，如果成功，则进入步骤 2；如果数据不存在，则进入步骤 3；

2、在调用 put 的时候将 get key 返回的 verison 重新传入 put 接口，服务端根据 version 是否匹配来返回 client 是否 put 成功；

3、get key 数据不存在，则新 put 数据。此时传入的 version 必须不是 0 和 1，其他的值都可以（例如 1000，要保证所有 client 是一套逻辑）。因为传入 0，tair 会认为强制覆盖；而传入 1，第一个 client 写入会成功，但是新写入时服务端的 version 以 0 开始计数，所以此时 version 也是 1，所以下一个到来的 client 写入也会成功，这样造成了冲突。

### 4.2 version 分布式锁

tair 中存在该 key，则认为该 key 所代表的锁已被 lock；不存在该 key，则未加锁。操作过程和上面相似。可以在 put 的时候增加 expire，以避免该锁被长期锁住。当然在选择这种策略的情况下需要考虑并处理 tair 宕机带来的锁丢失的情况。

## 5 config server

client 和 config server 的交互主要是为了获取数据分布的对照表，当 client 启动时获取到对照表后，会 cache 这张表，然后通过查这张表决定数据存储的节点，所以请求不需要和 config server 交互，这使得 tair 对外的服务不依赖 config server，所以它不是传统意义上的中心节点，也并不会成为集群的瓶颈。

config server 维护的对照表有一个版本号，每次新生成表，该版本号都会增加。当有 data server 状态发生变化（比如新增节点或者有节点不可用了）时，config server 会根据当前可用的节点重新生成对照表，并通过数据节点的心跳，将新表同步给 data server。当 client 请求 data server 时，后者每次都会将自己的对照表的版本号放入 response 中返回给 client，client 接收到 response 后，会将 data server 返回的版本号和自己的版本号比较，如果不相同，则主动和 config server 通信，请求新的对照表。

这使得在正常的情况下，client 不需要和 config server 通信，即使 config server 不可用了，也不会对整个集群的服务造成大的影响。有了 config server，client 不需要配置 data server 列表，也不需要处理节点的的状态变化，这使

得 tair 对最终用户来说使用和配置都很简单。

## 5.1 容灾

当有某台 data server 故障不可用的时候，config server 会发现这个情况，config server 负责重新计算一张新的桶在 data server 上的分布表，将原来由故障机器服务的桶的访问重新指派到其它有备份的 data server 中。这个时候，可能会发生数据的迁移，比如原来由 data server A 负责的桶，在新表中需要由 B 负责，而 B 上并没有该桶的数据，那么就将数据迁移到 B 上来。同时，config server 会发现哪些桶的备份数目减少了，然后根据负载情况在负载较低的 data server 上增加这些桶的备份。

## 5.2 扩容

当系统增加 data server 的时候，config server 根据负载，协调 data server 将他们控制的部分桶迁移到新的 data server上，迁移完成后调整路由。

注意：

不管是发生故障还是扩容，每次路由的变更，config server 都会将新的配置信息推给 data server。在 client 访问 data server 的时候，会发送 client 缓存的路由表的版本号，如果 data server 发现 client 的版本号过旧，则会通知 client 去 config server 取一次新的路由表。如果 client 访问某台 data server 发生了不可达的情况（该 data server 可能宕机了），客户端会主动去 config server 取新的路由表。

## 5.3 迁移

当发生迁移的时候，假设 data server A 要把桶 3, 4, 5 迁移给 data server B。因为迁移完成前，client 的路由表没有变化，因此对 3, 4, 5 的访问请求都会路由到 A。现在假设 3 还没迁移，4 正在迁移中，5 已经迁移完成，那么：

1、如果是对 3 的访问，则没什么特别，跟以前一样；

2、如果是对 5 的访问，则 A 会把该请求转发给 B，并且将 B 的返回结果返回给 client；

3、如果是对 4 的访问，在 A 处理，同时如果是对 4 的修改操作，会记录

修改 log，桶 4 迁移完成的时候，还要把 log 发送到 B，在 B 上应用这些 log，最终 A，B 上对于桶 4 来说，数据完全一致才是真正的迁移完成。

# 6 tair更多功能

## 6.1 客户端

tair 的 server 端是 C++ 写的，因为 server 和 client 之间使用 socket 通信，理论上只要可以实现 socket 操作的语言都可以直接实现成 tair client。目前淘宝实际提供的开源客户端有 java 和 C++，client 只需要知道 config server 的位置信息就可以享受 tair 集群提供的服务了，不过 tair 如果作为存储层，前端肯定还需部署 Nginx 这样的 web 服务器，以 Nginx 为例，淘宝似乎还没有开源其 tair 模块，春哥（agentzh）也没有公布 tair 的 lua 插件，如果想在 Nginx 里面访问 tair，目前似乎还没有什么办法了，除非自己去开发一个模块。

## 6.2 plugin支持

tair 还内置了一个插件容器，可以支持热插拔插件。插件由 config server 配置，config server 会将插件配置同步给各个数据节点，数据节点会负责加载/卸载相应的插件。 插件分为 request 和 response 两类，可以分别在 request 和 response 时执行相应的操作，比如在 put 前检查用户的 quota 信息等。插件容器也让 tair 在功能方便具有更好的灵活性。

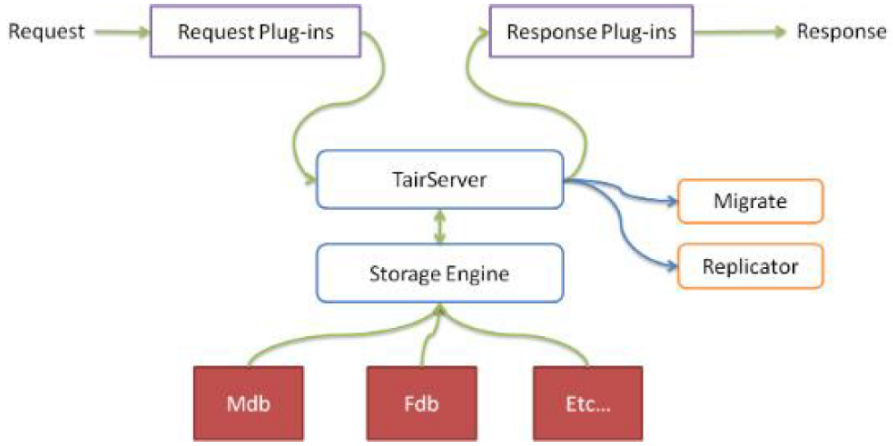


图 8: tair 热插拔插件

### 6.3 原子计数支持

tair 从服务器端支持原子的计数器操作，这使得 tair 成为一个简单易用的分布式计数器。

### 6.4 item 支持

tair 还支持将 value 视为一个 item 数组，对 value 中的部分 item 进行操作。比如有一个 key 的 value 为 [1,2,3,4,5]，我们可以只获取前两个 item，返回 [1,2]，也可以删除第一个 item。还支持将数据删除，并返回被删除的数据，通过这个接口可以实现一个原子的分布式 FIFO 的队列。

## 7 业内缓存框架优缺点对比

业内轻量级的缓存框架有 ehcache/redis/tair，重量级别的缓存框架 memcached，这里对这些缓存框架的优缺点做一个梳理给大家做参考。

**ehcache**

优点：易用性特别强、高性能缓存、版本迭代特别快、缓存策略支持多种、可以通过 rmi 可插入 api 实现分布式缓存、具备缓存监听、支持多缓存实例、提供 hibernate 的缓存实现、支持非持久化和持久化缓存数据。

缺点：使用磁盘空间做 cache 时非常占用磁盘空间，kill 掉 java进程时不能保证数据安全有可能会导致缓存数据丢失或者数据冲突。

适用场景：轻量级应用

**redis**

优点：非常丰富的数据结构而且都是原子性操作、高速读写、支持事务、支持命令行输入操作性好。在单节点的性能比较方面，redis 是性能比 tair 高大概 1/5，redis 目前研究人员比较多，社区比较活跃。在支持多种数据结构方面 tair 没有 redis 支持的全面。

缺点：没有自己的内存池在内存分配时存在的碎片会导致性能问题、对内存消耗剧烈（虽然有压缩方法但还是高）、持久化时定时快照每次都是写全量数据，代价有些高、aof 追加快照只追加增量数据但写入 log 特别大。目前发布版不支持分布式，测试版支持，测试版对分布式支持比较弱，是用主备支持高可能，没有副本。容灾性相比 tair 弱，原因是 redis 的分布式不支持多副本。

适用场景：轻量级应用

**memcached**

优点：协议简单，基于 libevent 的事件处理，内置内存存储方式，不互相通信的分布式.

缺点：数据是保存在内存当中的，一旦服务进程重启，数据会全部丢失。Memcached 以 root 权限运行，而且 memcached 本身没有任何权限管理和认证功能，安全性不足。内存消耗很大影响性能，memcached 进程运行之后，会预申请一块较大的内存空间，自己进行管理，用完之后再申请，不是每次需要的时候去向操作系统申请。

适用场景：分布式应用、数据库前段缓存、服务器间数据共享等。

不适合场景：不需要分布式的应用、单机服务、轻量级工程。

**tair 的优势**

(1) 在分布式集群支持方面 tair 支持副本，支持多种集群结构，如：一机房一个集群、双机房单集群单份、双机房独立集群、双机房单集群双份、双机房主备集群；

(2) 对于读写性能根据结点添加对线性上升，原因是各个结点之间是没有关系，节点增多相应性能提升；

(3) 高可用比较强，任何小于副本数结点挂掉，不会影响正常业务；

(4) 淘宝在多个实际应用场景应用，满足不同业务需求；

(5) 支持跨机房数据分布。

**tair 的弱势**

为了保证上面的高性能、高扩展和高可靠等特性，那么 tair 势必会消耗其他资源，比如查询效率有时候可能就会被降低。当然这个也是需要跟分布式的机器缓存空间、内存等等因素有关系，当缓存系统规模越大，业务系统规模越大时查询效率可能就会被降低，当然如果这样又有其他方案来解决这个规模大的问题了。

(1) 在单节点的性能比较方面，redis 性能比 tair 高大概 1/5；

(2) tair 目前研究人员比较少，社区不是很活跃。

# 8 参考资料

http://www.cnblogs.com/chenny7/p/4875396.html

http://blog.csdn.net/farphone/article/details/53522383

https://www.lvtao.net/database/tair.html

tair: https://github.com/alibaba/tair

tair java 客户端: https://github.com/alibaba/tair-java-client