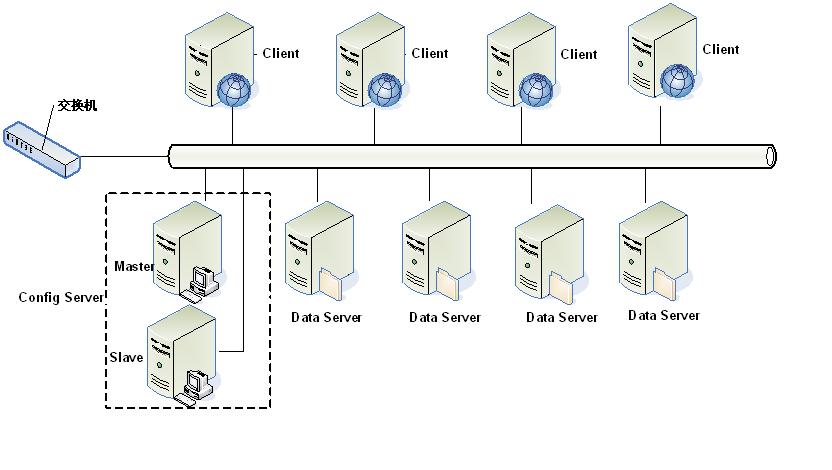
# Taobao Tair



Tair是由一个中心控制节点（config server）和若干个服务节点（Data server）组成的。

* config server负责管理所有的data server，维护data server的状态信息，config server目前采用一主一备的形式来保证可靠性；
* data server对外提供各种数据服务，并以心跳的形式将自身状况汇报给config server。所有的data server地位都是相同的。

## 基本概念

configID，唯一标识一个tair集群，每个集群有一个对应的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。

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，但内容不会冲突**。一个namespace下是如果存放相同的key，那么内容会受到影响，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。

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。

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。

## 存储引擎

tair分为持久化和非持久化两种使用方式：

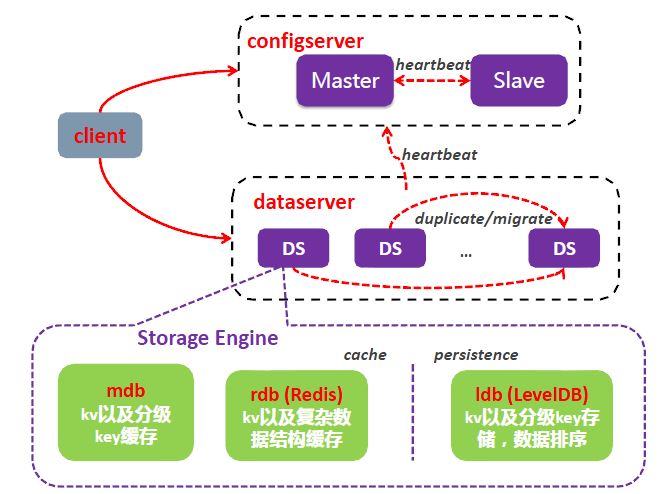
非持久化的tair可以看成是一个分布式缓存；

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bdb、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：

* mdb，定位于cache缓存，类似于memcache。支持k/v存取和prefix操作；
* 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；
* 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。



## 关键问题

### 分布式策略

Tair的分布采用的是一致性哈希算法，根据数据的主键计算哈希值后，分布到Q个桶中，桶是负载均衡和数据迁移的基本单位。Config server按照一定的策略把每个桶指派到不同的Data server上，因为数据按照主键哈希值，所以可以认为每个桶中的数据基本是平衡的，只要保证桶分布的均衡性，就能够保证数据分布的均衡性。Q的取值要远大于集群物理机器的数量。具体说，首先计算Hash(key)，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的路由表。

|  |  |
| --- | --- |
| bucket | data server |
| 0 | 192.168.10.1 |
| 1 | 192.168.10.2 |
| 2 | 192.168.10.1 |
| 3 | 192.168.10.2 |
| 4 | 192.168.10.1 |
| 5 | 192.168.10.2 |

这里有6个bucket，有两个data server负责维护。客户端Hash(key)之后mod 6得到bucket的编号，并向config server请求bucket对应的data server，然后客户端直接和data server通信。表的行数通常远大于集群节点数，这个和一致性哈希中的虚拟节点很相似。

如果有新的机器假如到集群，tair会自动调整路由表，会将部分bucket交给新的机器负责。

|  |  |
| --- | --- |
| 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 |

如果每个桶有多个备份，那么路由表就是有多个列存放data server的地址。

为了更进一步的提高数据的安全性，Tair的config server在build路由表的时候，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 提供了两种生成路由表的策略：

* 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； 一个桶的各份数据不能在同一台主机上；
* 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，不妨设使RA，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A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。

### 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。

### 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。

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version error，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的改变逻辑：**

* 如果put新数据且没有设置版本号，会自动将版本设置成1；
* 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；
* 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；
* put时传入的version参数为0，则强制更新成功，版本号自增1。

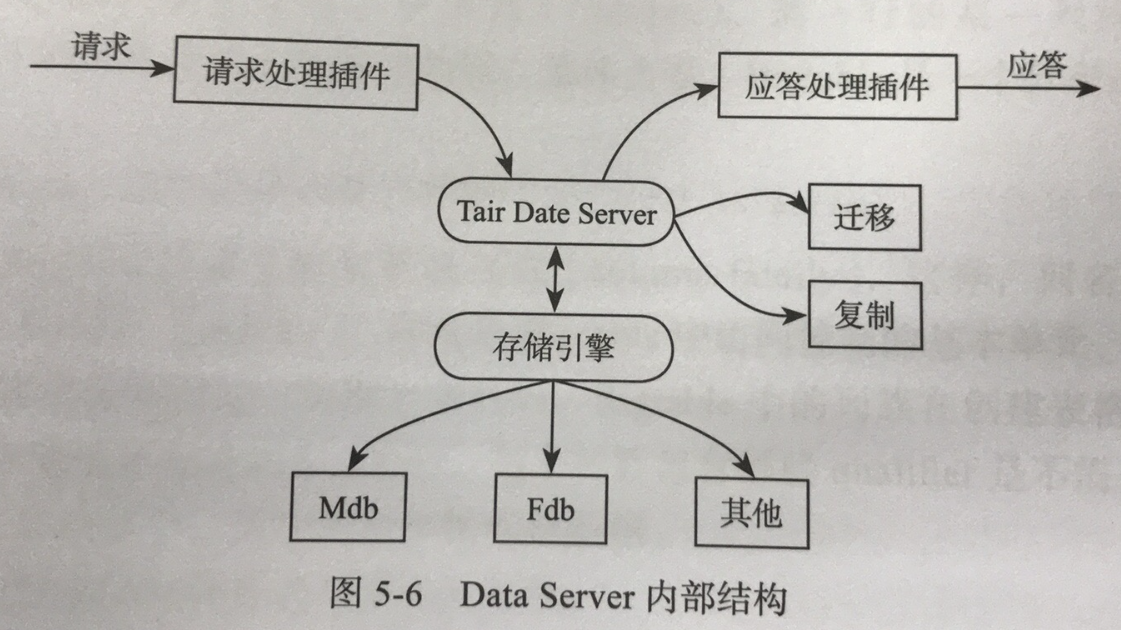
### config server

client和config server的交互是为了获取数据分布的路由表，client第一次获得路由表后会缓存路由表，大多数情况下，客户端不需要访问config server，即使config server故障了也不影响客户端正常访问。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，也不会成为系统的瓶颈。

config server维护的路由表有一个版本号，每次新生成表，该版本号都会增加。当有data server状态发生变化（新增节点或者节点不可用）时，config server会根据当前可用的节点重新生成路由表，并通过心跳把新表同步给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。

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。

### data server



data server具备抽象的存储引擎层，可以方便添加新的存储引擎。

### 容错

当某台data server故障不可用时，config server会检测到，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。每个哈希桶在tair中存储多个备份，如果是备副本，那么config server会重新为其指定一台data server。如果是主副本，那么config server首先将某个正常的备副本提升为主副本，对外提供服务。接着选择另外一台data server增加一个备副本。

### 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**。

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。

### 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：

* 如果是对3的访问，则没什么特别，跟以前一样；
* 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；
* 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。