# 启动过程描述

## 启动方法:

$CSDIR/chunkserver –f config\_file\_path

config\_file\_path是chunkserver的配置文件路径，配置文件的具体格式以及含义见下节。

## sstable数据存放位置

系统会建立一个/data目录(具体路径参见datadir\_path配置项),然后会在下面为每一个磁盘建立一个子目录，以数字命名标识磁盘号,从1开始。

将每一块磁盘mount到/[datadir\_path]/disk\_no节点。

所有的sstable files均匀分布到 /[datadir\_path]/disk[1..n]/[application]/sstable目录中，其中datadir\_path和application为配置项指定的名字，参见【配置文件】小节。

每个/[datadir\_path]/disk[1..n]/[application]/sstable目录中，还存放一个meta index文件，这个文件记录了这个目录下所有的tablet信息(包含range,sstable files)，如果一个目录下没有这个索引文件，则认为这个目录不可用。

## 启动过程

1. ChunkServer启动后会首先加载本地所有的tablet
   1. 先对[datadir\_path](见配置项)目录进行扫描，[datadir\_path]目录下应该存放了所有的sstable信息，并且按磁盘存放(见上节)，扫描每个磁盘目录下的最新的meta idx文件，如果找到idx文件，则加载idx文件。如果没有找到idx文件，忽略。
   2. 反序列化idx文件，读取其中的tablet信息，并根据其中记载的sstable信息，加载所有的sstable文件。(初次加载时，只是读入sstable trailer)
2. CS如果装载本地tablet成功，则开始向RS进行注册，CS会一直重复注册，直到成功注册为止。(如果此时RS尚未启动，则CS会不停尝试)
3. 注册成功后，CS会设置第一次租约的时常，并检查RS返回的注册状态
   1. 如果RS返回注册状态为1，说明是系统第一次启动，CS此时开始开始等待RS发起初次启动流程，此时，CS除了接收心跳延长租约，和等待初次启动消息以外，不接收任何服务。
   2. 如果RS返回注册状态为0，说明系统已经启动，CS则汇报本地HOLD的所有tablet信息，如果汇报成功，则进入服务状态，启动完毕。
4. 如果进入3.a,CS接收到RS发起的初次启动流程，CS开始汇报本地HOLD的所有tablet信息，进入服务状态，至此，启动完毕。

# 配置文件

使用ini file的形式

[root\_server]

vip = 10.1.1.1 // RootServer的vip address

port=3100 // RootServer的监听端口

[chunk\_server]

dev\_name = bond0 // ChunkServer监听的网络接口

port = 2900 // ChunkServer的监听端口

task\_queue\_size = 1000 // 任务处理队列的长度，待处理的任务积压超过这个数量，新接受到的任务将被返回系统繁忙的应答。

task\_thread\_count = 10 // 系统任务处理线程个数，配置与机器的CPU个数相关。

max\_migrate\_task\_count = 1 //最多用于迁移的线程个数

datadir\_path = /data // sstable文件所在的磁盘路径（对于多磁盘的机器，都mount到一个路径下的不同磁盘目录下）

application\_name = collect // 应用名称，比如collect为收藏夹

network\_timeout = 1000000 // 一般网络超时时间，单位微秒

lease\_check\_interval = 5000000 // 定时检查租约的间隔，单位微秒

retry\_times = 3 // 某些操作失败后需要重试，最大重试次数

max\_sstable\_size = 256 \* 1024 \* 1024 //sstable文件的最大尺寸

max\_sstable\_remain\_days = 3 // chunkserver上最多保存的老的sstable file的天数，超过这个天数的未被引用的文件，将被回收。

migrate\_band\_limit\_kbytes = 200 // 复制sstable时同步时的网络带宽限制，单位是KB per second

merge\_mem\_limit = 67108864 // 每日合并的时候每次最大处理多少数据，默认64M

merge\_thread\_per\_disk = 1 // 每日合并的线程数，以磁盘为单位，1表示为每个磁盘

// 启动一个线程来做每日合并，默认为1

reserve\_sstable\_copy = 3 // 要保留的sstable的版本数，默认为3

min\_drop\_cache\_wait\_second = 120 // tablets切换到新版本以后，等待一段时间后将老的cache清理掉，这个等待是为了避免还有线程在使用老的cache;

min\_merge\_interval\_second = 600 // 两个合并开始的最小间隔,单位秒。

[memory\_cache]

dicache\_max\_num = 32 \* 1024 // block数据索引的个数，计算方法：

// cache内存总量/每个block的压缩后的平均大小

// 现假设block压缩后的平均大小为32k，cache总量为1G

dicache\_min\_timeo = 1 \* 1000 \* 1000 // block数据索引的过期时间，单位微秒，默认1秒

mbcache\_max\_num = 1024 // 内存块的个数，每个内存块默认1M，总内存为1G

mbcache\_min\_timeo = 1 \* 1000 \* 1000 // 内存块过期时间，单位微秒，默认1秒

block\_cache\_memblock\_size = 1024 \* 1024 // block cache分配并管理的内存块大小为1M

block\_cache\_memalign\_size = 512 // 使用512字节对齐的方式，direct IO方式读取数据

block\_index\_cache\_memsize = 512 \* 1024 \* 1024 // block index cache的大小，默认512M

block\_index\_cache\_max\_no\_active\_usec = 0 // 默认按照lru进行淘汰，单位微秒

// >0 单个item最大允许不被访问的时间，超过该时// 间 item会被淘汰；在该时间范围内item不会被淘// 汰，此时cache\_mem\_size无含义，但是可以提示内// 存的使用

// =0 按照lru淘汰

// <0 不被淘汰，此时cache\_mem\_size无含义，但是// 可以提示内存的使用

block\_index\_cache\_hash\_slot\_num = 4 \* 1024 // 每个chunkserver上大约有4000个sstable

// 文件，每个文件都有一个block index

join\_cache\_memsize = 512 \* 1024 \* 1024 //chunkserver 合并时的join cache选项

join\_cache\_max\_no\_active\_usec = 0 //取值类似于上述的block index cache

join\_cache\_hash\_slot\_num = 4 \* 1024

file\_info\_cache\_max\_cache\_num = 1024 // 打开并缓存的文件句柄个数

file\_info\_cache\_min\_gc\_timeo = 0 // 文件句柄的过期时间，单位微秒