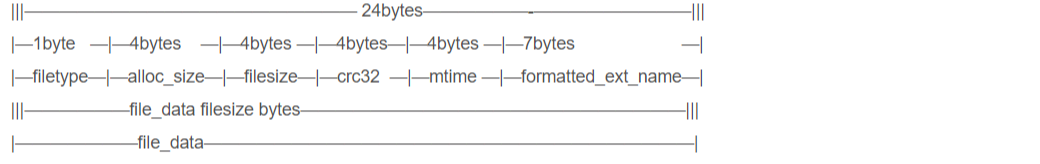
**FastDFS提供了合并存储功能的实现，所有的配置都在tracker.conf文件之中，具体摘录如下：**  
trunk功能启动与配置：通过tracker.conf文件启动与配置，个配置项如下：  
use\_trunk\_file = false#是否启用trunk存储  
slot\_min\_size = 256#trunk文件最小分配单元  
slot\_max\_size = 16MB#trunk内部存储的最大文件，超过该值会被独立存储  
trunk\_file\_size = 64MB#trunk文件大小  
trunk\_create\_file\_advance = false#是否预先创建trunk文件  
trunk\_create\_file\_time\_base = 02:00#预先创建trunk文件的基准时间  
trunk\_create\_file\_interval = 86400#预先创建trunk文件的时间间隔  
trunk\_create\_file\_space\_threshold = 20G#trunk创建文件的最大空闲空间  
trunk\_init\_check\_occupying = false#启动时是否检查每个空闲空间列表项已经被使用  
trunk\_init\_reload\_from\_binlog = false#是否纯粹从trunk-binlog重建空闲空间列表  
trunk\_compress\_binlog\_min\_interval = 0#对trunk-binlog进行压缩的时间间隔  
  
  
**合并存储文件命名与文件结构**  
    我们知道向FastDFS上传文件成功时，服务器返回该文件的存取ID叫做file\_id，当没有启动合并存储时该file\_id和磁盘上实际存储的文件一一对应，当采用合并存储时就不再一一对应而是多个file\_id对应的文件被存储成一个大文件。  
注：下面将采用合并存储后的大文件统称为Trunk文件，没有合并存储的文件统称为源文件；  
区分三个概念：  
    1）Trunk文件：storage服务器磁盘上存储的实际文件，默认大小为64MB  
    2）合并存储文件的file\_id：表示服务器启用合并存储后，每次上传返回给客户端的file\_id，注意此时该file\_id与磁盘上的文件没有一一对应关系；  
    3）没有合并存储的file\_id：表示服务器未启用合并存储时，Upload时返回的file\_id  
    Trunk文件文件名格式：fdfs\_storage1/data/00/00/000001 文件名从1开始递增，类型为int；  
 **1、在启动合并存储时服务返回给客户端的file\_id也会有所变化**  
具体如下：

1) 没有合并存储时file\_id：group1/M00/00/00/CmQPRlP0T4-AA9\_ECDsoXi21HR0.tar.gzCmQPRlP0T4-AA9\_ECDsoXi21HR0.tar.gz  
这个文件名中，除了.tar.gz 为文件后缀，CmQPRlP0T4-AA9\_ECDsoXi21HR0 这部分是一个base64编码缓冲区，组成如下：  
storage\_id（ip的数值型）  
timestamp（创建时间）  
file\_size（若原始值为32位则前面加入一个随机值填充，最终为64位）  
crc32（文件内容的检验码）

2)合并存储时file\_id：group1/M00/00/00/CgAEbFQWWbyIPCu1AAAFr1bq36EAAAAAQAAAAAAAAXH82.conf  
采用合并的文件ID更长，因为其中需要加入保存的大文件id以及偏移量，具体包括了如下信息：  
file\_size：占用大文件的空间（注意按照最小slot-256字节进行对齐）  
mtime：文件修改时间  
crc32：文件内容的crc32码  
formatted\_ext\_name：文件扩展名  
alloc\_size：文件大小与size相等  
id：大文件ID如000001  
offset：文件内容在trunk文件中的偏移量  
size：文件大小

**2、Trunk文件内部结构**  
    trunk内部是由多个小文件组成，每个小文件都会有一个trunkHeader(可认为是元数据)，以及紧跟在其后的真实数据，结构如下：



每个Trunk-Header从上图可以看到，占用了72字节。  
  
**合并存储空闲空间管理**  
**1、概述**  
    Trunk文件为64MB(默认)，因此每次创建一次Trunk文件总是会产生空余空间，比如为存储一个10MB文件，创建一个Trunk文件，那么就会剩下接近54MB的空间(TrunkHeader 会24字节，后面为了方便叙述暂时忽略其所占空间)，下次要想再次存储10MB文件时就不需要创建新的文件，存储在已经创建的Trunk文件中即可。另外当删除一个存储的文件时，也会产生空余空间。  
    在Storage内部会为每个store\_path构造一颗以空闲块大小作为关键字的空闲平衡树，相同大小的空闲块保存在链表之中。每当需要存储一个文件时会首先到空闲平衡树中查找大于并且最接近的空闲块，然后试着从该空闲块中分割出多余的部分作为一个新的空闲块，加入到空闲平衡树中。例如：要求存储文件为300KB，通过空闲平衡树找到一个350KB的空闲块，那么就会将350KB的空闲块分裂成两块，前面300KB返回用于存储，后面50KB则继续放置到空闲平衡树之中。  
    假若此时找不到可满足的空闲块，那么就会创建一个新的trunk文件64MB，将其加入到空闲平衡树之中，再次执行上面的查找操作(此时总是能够满足了)。  
  
**2、TrunkServer**  
    假若所有的Storage都具有分配空闲空间的能力(upload文件时自主决定存储到哪个TrunkFile之中)，那么可能会由于同步延迟导致数据冲突，例如：Storage-A：Upload一个文件A.txt 100KB，将其保存到000001这个TrunkFile的开头，与此同时，Storage-B也接受Upload一个文件B.txt 200KB，将其保存在000001这个TrunkFile文件的开头，当Storage-B收到Storage-A的同步信息时，他无法将A.txt 保存在000001这个trunk文件的开头，因此这个位置已经被B.txt占用。  
   为了处理这种冲突，引入了TrunkServer概念，只有TrunkServer才有权限分配空闲空间，决定文件应该保存到哪个TrunkFile的什么位置。TrunkServer由Tracker指定，并且在心跳信息中通知所有的Storage。  
    引入TrunkServer之后，一次Upload请求，Storage的处理流程图如下：

**3、TrunkFile同步**  
    开启了合并存储服务后，除了原本的源文件同步之外，TrunkServer还多了TrunkBinlog的同步（非TrunkServer没有TrunkBinlog同步）。源文件的同步与没有开启合并存储时过程完全一样，都是从binlog触发同步文件。  
    TrunkBinlog记录了TrunkServer所有分配与回收空闲块的操作，由TrunkServer同步给同组中的其他Storage。TrunkServer为同组中的其他Storage各创建一个同步线程，每秒将TrunkBinlog的变化同步出去。同组的Storage接收到TrunkBinlog只是保存到文件中，不做其他任何操作。  
TrunkBinlog文件文件记录如下：  
1410750754 A 0 0 0 1 0 67108864  
1410750754 D 0 0 0 1 0 67108864  
各字段含义如下：  
时间戳  
操作类型(A：增加，D：删除)  
store\_path\_index  
sub\_path\_high  
sub\_path\_low  
file.id（TrunkFile文件名，比如000001）  
offset（在TrunkFile文件中的偏移量）  
size（占用的大小，按照slot对齐）

**4、空闲平衡树重建**  
    当作为TrunkServer的Storage启动时可以从TrunkBinlog文件中中加载所有的空闲块分配与加入操作，这个过程就可以实现空闲平衡书的重建。  
    当长期运行时，随着空闲块的不断删除添加会导致TrunkBinlog文件很大，那么加载时间会很长，FastDFS引入了检查点文件storage\_trunk.dat，每次TrunkServer进程退出时会将当前内存里的空闲平衡树导出为storage\_trunk.dat文件，该文件的第一行为TrunkBinlog的offset，也就是该检查点文件负责到这个offset为止的TrunkBinlog。也就是说下次TrunkServer启动的时候，先加载storage\_trunk.dat文件，然后继续加载这个offset之后的TrunkBinlog文件内容。

下面为TrunkServer初始化的流程图：

**5、TrunkBinlog压缩**  
    上文提到的storage\_trunk.dat既是检查点文件，其实也是一个压缩文件，因为从内存中将整个空闲平衡树直接导出，没有了中间步骤，因此文件就很小。这种方式虽然实现了TrunkServer自身重启时快速加载空闲平衡树的目的，但是并没有实际上缩小TrunkBinlog文件的大小。假如这台TrunkServer宕机后，Tracker会选择另外一台机器作为新的TrunkServer，这台新的TrunkServer就必须从很庞大的TrunkBinlog中加载空闲平衡树，由于TrunkBinlog文件很大，这将是一个很漫长的过程。  
    为了减少TrunkBinlog，可以选择压缩文件，在TrunkServer初始化完成后，或者退出时，可以将storage\_trunk.dat与其负责偏移量之后的TrunkBinlog进行合并，产生一个新的TrunkBinlog。由于此时的TrunkBinlog已经从头到尾整个修改了，就需要将该文件完成的同步给同组内的其他Storage，为了达到该目的，FastDFS使用了如下方法：  
      1）TrunkServer将TrunkBinlog同步给组内其他Storage时会将同步的最后状态记录到一个mark文件之中，比如同步给A，则记录到A.mark文件(其中包括最后同步成功的TrunkBinlog偏移量)。  
      2）TrunkServer在将storage\_trunk.dat与TrunkBinlog合并之后，就将本地记录TrunkBinlog最后同步状态的所有mark文件删除，如一组有A、B、C，其中A为TrunkServer，则A此时删除B.mark、C.mark。  
      3）当下次TrunkServer要同步TrunkBinlog到B、C时，发现找不到B.mark、C.mark文件，就会自动从头转换成从头开始同步文件。  
      4）当TrunkServer判断需要从头开始同步TrunkBinlog，由于担心B、C已经有旧的文件，因此就需要向B、C发送一个删除旧的TrunkBinlog的命令。  
      5）发送删除命令成功之后，就可以从头开始将TrunkBinlog同步给B、C了。  
大家发现了么，这里的删除TrunkBinlog文件，会有一个时间窗口，就是删除B、C的TrunkBinlog文件之后，与将TrunkBinlog同步给他们之前，假如TrunkBinlog宕机了，那么组内的B、C都会没有TrunkBinlog可使用。  
流程图如下：

**Tracker-Leader选择TrunkServer**  
**1、描述**  
      在一个FastDFS集群之中，在开启合并存储时，为了分配空间引入了一个TrunkServer角色，该TrunkServer是该Group中的一个Storage，只是该Storage要负责为该组内的所有Upload操作分配空间。为了避免不同Tracker为该Group选择了不同的TrunkServer，此时引入了Tracker-Leader角色，也就是TrunkServer最终是由Tracker-Leader来选择的，然后通知给该组内的所有Storage。  
  
**2、 Tracker -Leader选择TrunkServer时机**  
    1）当组内的一个Storage状态变成Active时，并且该组还没有指定TrunkServer，在tracker\_mem\_active\_store\_server函数中触发。  
    2）某个Tracker在经过选择，被设置成Leader时，则为当前还没有指定TrunkServer的组选择TrunkServer，在relationship\_select\_leader函数中触发。  
    3）在定期指定的任务tracker\_mem\_check\_alive函数中，默认该函数100秒指定一次。会尝试着为每个当前还没有指定TrunkServer的组选择TrunkServer。对已经指定的组检查其TrunkServer是否还处于活动状态(根据TrunkServer与Tracker的最后心跳时间计算)，若不处于活动状态，则会尝试着给该组选择一个新的TrunkServer。在tracker\_mem\_check\_alive函数中触发。  
   
**3、Tracker-Leader选择TrunkServer的过程**  
    该过程由tracker\_mem\_find\_trunk\_server函数负责，具体操作步骤如下：  
    1）依次向组内当前状态为ACTIVE的Storage发送TRUNK\_GET\_BINLOG\_SIZE命令的消息，来查询每个Storage当前保存的Trunk-Binlog的文件大小。来找到Trunk-Binlog文件最大的Storage。  
    2）若该Group的最后一个TrunkServer与要设置的新的TrunkServer并非同一个，则像该新的TrunkServer发送TRUNK\_DELETE\_BINLOG\_MARKS命令，让其删除trunk-binlog同步的状态mark文件。(既然这个TrunkServer是新的，那么就要清除同步trunk-binlog的状态，使其从头同步trunk-binlog给组内的其他Storage)。  
    3）变更该组的TrunkServer，并将修改写入到storage\_groups\_new.dat文件之中，更新该组的最后TrunkServer设置，设置TrunkServer已经变更标志，该标志使得在与Storage的心跳中通知对方TrunkServer已经变更。