MooseFs/tfs研究

网站数据存储需要做好前期的规划，不然数据量上来以后无论是管理还是性能上都带来很多问题，对于需要存储大量文件、图片的网站来说是个重点的话题，通常网页文字内容和网页图片内容 所占用的存储空间相比较的话，显然是图片存储占用的空间要远远超过网站文字内容所需要存储的空间，从另外一个角度上来看访问图片所占用的网络带宽要远远超过文字内容所占用的带宽。  
在系统运行上的瓶颈对于互联网用户来说往往不是落在系统内部计算的I/O瓶颈上，而是落在网络带宽的瓶颈上。把存储图片的服务器和访问图片服务器的网络入口独立出来，在条件允许的情况下给予更的高网络带宽和独立的域名，有利于扩展性和整体性能的发挥，将计算、存储的扩展性和带宽的资源做到合理分配，互不影响，是我们当前的目标。

# Tfs

通用性方面。TFS目前只支持小文件的应用，大文件应用是不支持的。对小图片、网页等几十KB内的数据存储非常适用，但对视频点播VOD、文件下载等应用暂时无法适用。

性能方面。Client写文件是同步处理的，需要等所有dataserver写成功后才能返回，这很是影响性能。

用户接口。TFS没有提供POSIX接口，提供的API也与标准接口不一致。另外，TFS有自己的文件命名规则，如果用户使用自定义的文件名，则需要自已维护文件名与TFS文件名之间的映射关系。

代码方面。使用了C++实现，感觉相对臃肿一点，如果用纯C实现应该会简洁不少(可能我C中毒太深了)。代码注释基本没有，代码质量也不是很好。

技术文档。官方有一些文档，但显然非常不够深入和全面。

小文件优化。官方称针对海量小文件的随机读写访问性能做了特殊优化，现在只看到把众多小文件存放与一个Block中，这与Squid中的COSS原理相似。其他特殊优化措施未知，LOFS(Lost of small files)是个难点问题。

# Mfs

特性：

1、 实施起来简单。MFS的安装、部署、配置相对于其他几种工具来说，要简单和容易得多。

2、 不停服务扩容。MFS框架做好后，随时增加服务器扩充容量；扩充和减少容量皆不会影响现有的服务。注：hadoop也实现了这个功能。

3、 恢复服务容易。除了MFS本身具备高可用特性外，手动恢复服务也是非常快捷.

4、使用廉价机器(老机器)构建分布式网络存储环境，支持150TB以上的存储容量

5、提高海量小文件的读取效率，

6、提高随机读写的效率

7、可以设置删除文件的空间回收时间

常见问题：

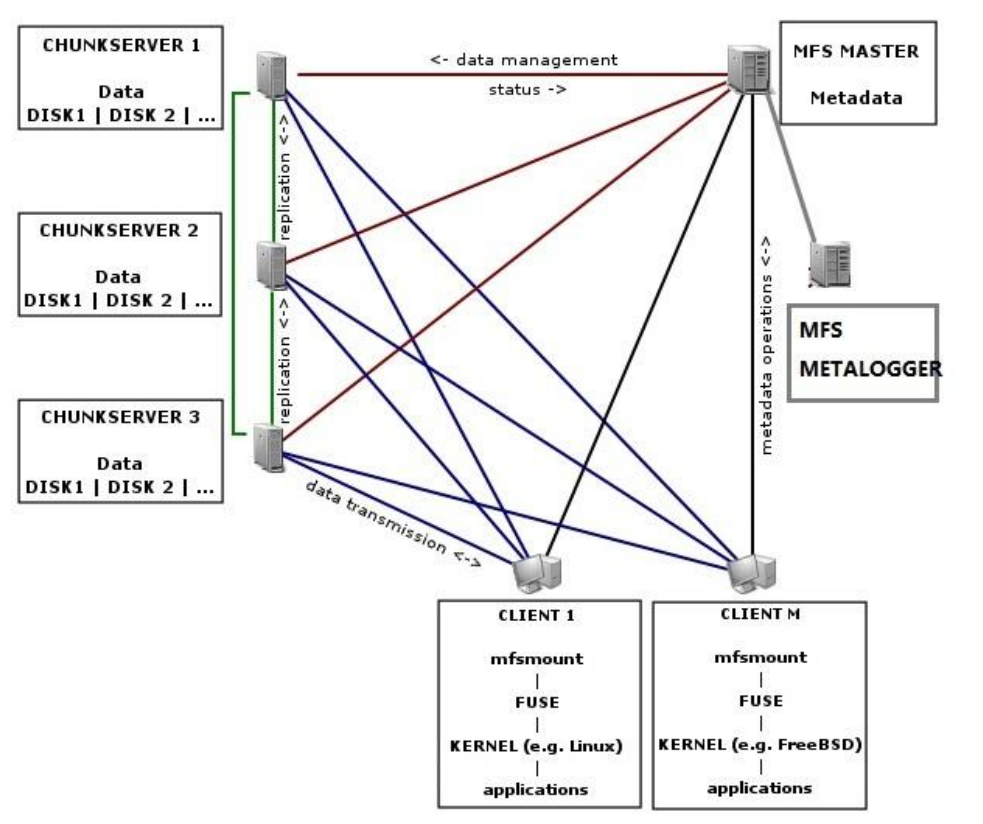
1、master性能瓶颈，主要是可扩展性不强

2、体系架构存储文件总数的瓶颈，mfs把文件系统的结构缓存到master内存中，这样文件越多，master的内存消耗越大，8G对应2500kw文件数，2亿就的64G内存

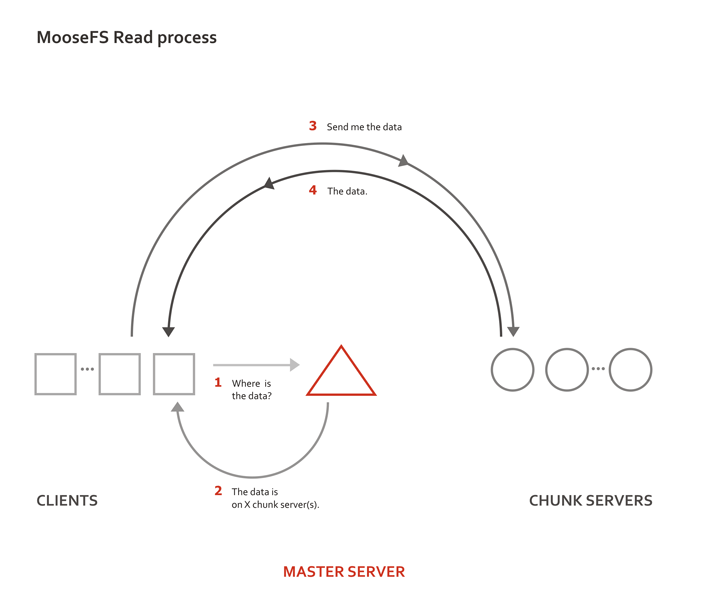
3、单点故障解决方案的健壮性

4、垃圾回收   把默认的86400改为300秒，这样可以免的垃圾还没回收完，你的存储容量就暴掉了。

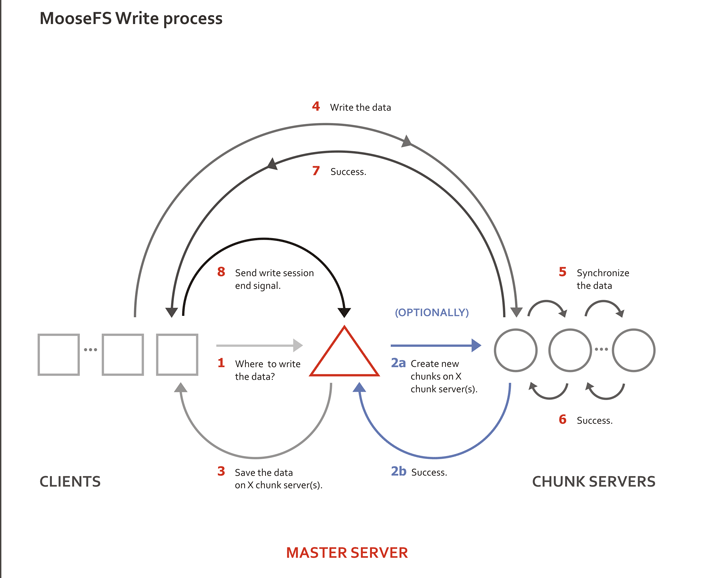
## 运行原理



## 读进程工作机制



## 写进程工作机制



**Chunkserver** —— 数据存储地，文件以chunk大小存储，每chunk最大为64M，小于64M的，该chunk的大小即为该文件大小，超过64M的文件将被均分，每一份（chunk）的大小以不超过64M为原则；文件可以有多份copy，即除了原始文件以外，该文件还存储的份数，当goal为1时，表示只有一份copy，这份copy会被随机存到一台chunkserver上，当goal的数大于1时，每一份copy会被分别保存到每一个chunkserver上，goal的大小不要超过chunkserver的数量，否则多出的copy，不会有chunkserver去存，goal设置再多实际上也就没有意义的。Copy的份数，一般设为大于1份，这样如果有一台chukserver坏掉后，至少还有一份copy，当这台又被加进来后，会将失去的那份copy补回来，始终保持原有的copy数，而如果goal设为1copy，那么当存储该copy的chunkserver坏掉，之后又重新加入回来，copy数将始终是0，不会恢复到之前的1个copy。

**Chunkserver**上的剩余存储空间要大于1GB（Reference Guide有提到），新的数据才会被允许写入，否则，你会看到No space left on device的提示，实际中，测试发现当磁盘使用率达到95%左右的时候，就已经不能写入了，当时可用空间为1.9GB。

多个chunker并不影响写的速度，但是能加快读的速度，在原来的基础上增加一个chunker时，数据会自动同步到新增的chunker上以达到数据的平衡和均衡。

## 组成角色

1、元数据服务器-----在整个体系中负责管理管理文件系统

比如：文件大小，存储的位置，份数等，和innodb中共享空间（ibdata）中存储的信息类似，这些信息被记录到metadata.mfs中，当该文件被载入内存后，改文件会重命名为metadata.mfs.back，当chunkserver上有更新时，master会定期将获得的新的信息回写到metadata.mfs.back中，保重元数据的可靠。

**2、** Metalogger ——用来安装 metalogger 的主机，在性能上应该比 master 强大（至少有更多的内存）。一旦主控服务器 master 失效，只要导入 changelogs 到元数据文件,备份服务器 metalogger 将能接替发生故障的 master，行使管理服务器的职能

3、  数据存储服务器chunkserver。真正存储用户数据的服务器。存储文件时，首先把文件分成块，然后这些块在数据服务器chunkserver之间复制（复制份数可以手工指定，建议设置副本数为3）。数据服务器可以是多个，并且数量越多，可使用的“磁盘空间”越大，可靠性也越高。

4、  客户端。使用MFS文件系统来存储和访问的主机称为MFS的客户端。凡是使用mfs文件系统的机器都可以被称为client。client是mfs系统的使用者。当client把mfs文件系统挂载到本机以后，它可以像使用一个普通的磁盘分区一样，来使用mfs。

## 使用的建议

负载均衡：HAproxy采用RoundRobin负载均衡算法，分载前端用户请求的压力到每个web图片服务器上，  
  2.web服务：采用Nginx-0.9.6 做图片的web服务器，对网站的大、中、小图片进行读取，加上**[Nginx的Redis模块](http://wiki.nginx.org/HttpRedis" \t "_blank)**对缓存中的微型(头像)图片进行读取，  
  3. 缓存服务器：存储网站的 微型图片，签名照，小头像，表情图片，通过Nginx的Redis模块直接读取，通过调用Redis的java API程序对数据进行写入，  
  4.存储单元：采用Moosefs 存储  大、中、小图片，并且提供监控管理界面，查看存储空间运行状态，  
  5.图片索引：将图片名和图片url路径作为键值对(Key/Value)，放入nosql 中存储，并且进行数据查询，避免图片重复存储，便于将来管理，  
  6.应用服务器：对图片写入的操作全部由Java应用服务器完成。

# 安装

升级系统内核为2.6.18-164版本 yum install kernel

2、使用fuse 2.7x版本编译安装 版本的fuse中包含了linux内核需要的fuse模块，配置编译选项时指定 –enable-kernel-module 选项，make的时候就会编译相应的fuse模块，make intall会将fuse.ko复制到 /lib/modules/`uname -r`/kernel/fs/fuse/ 目录下 安装后利用

# modprobe -l | grep fuse

/lib/modules/2.6.18-128.el5xen/kernel/fs/fuse/fuse.ko

利用yum安装当前内核版本的fuse模块

## Master安装

groupadd mfs

useradd -g mfs mfs

cd /usr/src

tar -zxvf mfs-1.6.15.tar.gz

cd mfs-1.6.15

./configure --prefix=/usr --sysconfdir=/etc --localstatedir=/var/lib --with-default-user=mfs --with-default-group=mfs --disable-mfschunkserver --disable-mfsmount

Configure失败：configure: error: zlib development library not found

wget <http://zlib.net/zlib-1.2.5.tar.gz>

tar -zxvf zlib-1.2.5.tar.gz

./conifgure

make && make install

示例配置文件将会建立在/etc下扩展名为.dist,我们使用这些作用我们最终的配置文件：

cd /etc

cp mfsmaster.cfg.dist mfsmaster.cfg

#cp mfsmetalogger.cfg.dist mfsmetalogger.cfg

#cp mfsexports.cfg.dist mfsexports.cfg

mfsmaster.cfg 文件包含master server的设置,这里我们没有做任何改变，你能在手册页（man mfsmaster.cfg）了解这个文件的更多信息

fsexports.cfg文件指定哪个客户端可以挂载MooseFS文件系统并且拥有什么权限.在我们的例子中指定仅仅192.168.2.x的机器能读写MooseFS的整个架构.该文件的第一行，并有这行没有注释我们把星号(\*)改变为192.168.2.0/24 因此像下面这样：

192.168.2.0/24        /       rw,alldirs,maproot=0

二进制metadata文件和changelog文本文件存放在一个文件夹，该文件夹是在编译期间localstatedir所设置-在我们的例子中这个文件夹是:/var/lib/mfs.首次安装会建立一个名字为metadata.mfs.empty的空的metadata文件.我们把这个空文件改变成metadata.mfs：

我们也需要在/etc/hosts 指定主机名mfsmaster和主机IP地址192.168.1.1相对应(译者注：为了在配置文件中用主机名mfsmaster代替IP，做了一次简单DNS解析)：

192.168.1.1     mfsmaster

现在可以运行master server(server将要使用安装时给定的用户运行，在我们的实例中它是mfs)：

/usr/sbin/mfsmaster start

在生产环境中应该设置mfsmaster进程在操作系统启动时自动运行

我们现在也可以运行CGI监控在浏览器中查看当前MooseFS系统的状态

/usr/sbin/mfscgiserv

现在可以在http://192.168.1.118:9425/看到一些信息(暂时还没有关于chunk servers的数据)

## **Backup server(metalogger) 的安装**

用来安装metalogger 的主机，在性能上应该比master 强大（至少有更多的内存）。一旦主控服务器master 失效，只要导入changelogs 到元数据文件,备份服务器metalogger 将能接替发生故障的master，行使管理服务器的职能.

1. groupadd mfs
2. #useradd -g mfs mfs
3. #cd /usr/src
4. #tar -zxvf mfs-1.6.15.tar.gz
5. #cd mfs-1.6.15
6. #./configure --prefix=/usr --sysconfdir=/etc
7. --localstatedir=/var/lib --with-default-user=mfs
8. --with-default-group=mfs --disable-mfschunkserver --disable-mfsmount
9. #make
10. #make install
11. #cd /etc
12. #cp mfsmetalogger.cfg.dist mfsmetalogger.cfg

## Chunkserver 安装

1. ./configure --prefix=/usr --sysconfdir=/etc
2. --localstatedir=/var/lib --with-default-user=mfs
3. --with-default-group=mfs --disable-mfsmaster

挂载文件系统  
MooseFS chunk 以独占方式使用专门磁盘分区是非常必要的--这样做的好处是便于管理剩余空间。  
MooseFS 并不考虑其剩余空间能被另作他用。如果没有单独创建文件系统的条件，可以在文件中创建一个文件系统.  
  
以下是具体操作步骤：

创建目录  
mkdir -p /storage/mfschunks

创建镜像文件mfschunks1  
dd if=/dev/zero of=/storage/mfschunks/mfschunks1 bs=1024 count=1 seek=$((3\*1024\*1024-1))

创建文件系统  
mkfs -t ext3 /storage/mfschunks/mfschunks1

创建挂接点  
mkdir -p /mnt/mfschunks1

挂接文件系统  
mount -t ext3 -o loop /storage/mfschunks/mfschunks1 /mnt/mfschunks1

在启动chunk server 前，需确保用户mfs 有权限读写将要被挂接的分区（因为chunk server 运  
行时要在此创建一个.lock 的文件）:  
chown -R mfs:mfs /mnt/mfschunks1

1. #mkdir -p /storage/mfschunks
2. #dd if=/dev/zero of=/storage/mfschunks/mfschunks1 bs=1024 count=1
3. seek=$((2\*1024\*1024-1))
4. #mkfs -t ext3 /storage/mfschunks/mfschunks1
5. #mkdir -p /mnt/mfschunks1
6. #mount -t ext3 -o loop /storage/mfschunks/mfschunks1 /mnt/mfschunks1
7. #dd if=/dev/zero of=/storage/mfschunks/mfschunks2 bs=1024 count=1
8. seek=$((2\*1024\*1024-1))
9. #mkfs -t ext3 /storage/mfschunks/mfschunks2
10. #mkdir -p /mnt/mfschunks2
11. #mount -t ext3 -o loop /storage/mfschunks/mfschunks2 /mnt/mfschunks2
12. #chown -R mfs:mfs /mnt/mfschunks1
13. #chown -R mfs:mfs /mnt/mfschunks2

## Client 安装

（一） 安装MFS客户端

◆Mfsmount需要依赖FUSE,因此需要先安装好fuse，这里我选用 fuse-2.7.4.tar.gz。

1、解包 tar zxvf fuse-2.7.4.tar.gz

2、切换目录 cd fuse-2.7.4.

3、配置 ./configure

4、编译安装 make；make install

如果系统已经安装了fuse,则跳过这个步骤。

◆安装MFS客户端程序

1、修改环境变量文件/etc/profile ,追加下面的行，然后再执行命令source /etc/profile使修改生效。

|  |
| --- |
| export PKG\_CONFIG\_PATH=/usr/local/lib/pkgconfig:$PKG\_CONFIG\_PATH |

如果不执行这个操作，在后面安装MFS的过程中，执行命令

./configure --enable-mfsmount时可能出现"checking for FUSE... no  configure: error: mfsmount build was forced, but fuse development package is not installed"这样的错误，而不能正确安装MFS客户端程序。

./configure --prefix=/usr --sysconfdir=/etc --localstatedir=/var/lib --with-default-user=mfs --with-default-group=mfs --disable-mfsmaster --disable-mfschunkserver

configure: error: mfsmount build was forced, but fuse library is too old or not installed

解决方法：yum -y install fuse-devel。

更改安装目录：

./configure --prefix=/usr/local/mfs --with-default-user=mfs --with-default-group=mfs --sysconfdir=/etc/mfs/ --localstatedir=/var/lib/ --disable-mfsmaster --enable-mfsmount --disable-mfschunkserver

现在挂载系统到/mnt/mfs：

/usr/bin/mfsmount /mnt/mfs -H mfsmaster

mfssetglal -r 命令设置文件夹的副本数：

mfssetgoal -r 1 /mnt/mfs/folder1

安全停止 MooseFS 集群步骤:

umount -l /mnt/mfs                                                #客户端卸载MooseFS 文件系统

/usr/local/mfs/bin/mfschunkserver stop                            #停止 chunk server 进程

/usr/local/mfs/bin/mfsmetalogger stop                             #停止 metalogger 进程

/usr/local/mfs/bin/mfsmaster stop

安全启动 MooseFS 集群步骤：

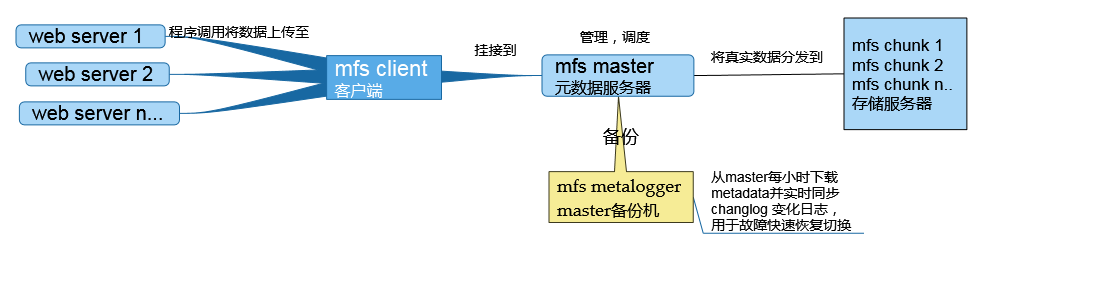
/usr/local/mfs/bin/mfsmaster start                                #启动 master 进程

/usr/local/mfs/bin/mfschunkserver start                           #启动 chunkserver 进程

/usr/local/mfs/bin/mfsmetalogger start                            #启动 metalogger 进程

/usr/local/mfs/bin/mfsmount /mnt/mfs -H 192.168.128.10

## mfs应用



1. mfs client客户端数据由mount创建或web服务器通过程序调用将要上传的内容至mfs client(相当于原先的nfs图片服务器)。
2. master元数据服务器负责管理，调度，只保留元数据日志（此日志非日志文件中的日志），将真实数据分发至chunk存储服务器。  
   目前不支持多个master存在单点故障，可增设一台metalogger　server作为master的备份，定期下载metadata并实时同步changlog日志变化文件（默认24小时同步一次，最小可设为每1小时同步一次）。
3. master调度将真实数据分发至各chunk server，文件分成块以副本形式在各chunk间复制存储。  
   后期可以不停服务增加chunk服务器或磁盘空间进行扩容。

## 后期维护

1.可以不停服务进行在线扩容，在原来基础上增加一个chunk时，数据会自动同步到新增的chunk server上以达到数据均衡。master会自动调度，数据会在chunk间重新分配。  
2.master的主备切换  
master的主备切换分为两个步骤：一是由metalogger恢复master；二是chunk和client端进行响应的处理。  
2.1　metalogger恢复master  
1）metalogger定期从master下载metadata文件，并实时记录changelog，但是这个“实时”究竟有多么的实时，还得再看看。这个下载metadata和记录changelog的工作有点类似sfrd客户端每天下载基准和导入增量。  
2）master挂掉之后，使用metarestore命令将metalogger中的基准和增量变成master需要的metadata，然后启动mfsmaster。master和metalogger可以部署在同一台机器，也可以部署在不同机器。  
3）metalogger恢复master时使用的命令：  
 cd /home/XXXX/local/mfs/sbin  
 ./metarestore –a  
 ./mfsmaster

4）说明：  
      metalogger服务器中需要备份master的两个配置文件，由于配置文件不是经常变化，因此通过定时脚本进行文件同步即可。  
      当metalogger没有下载metadata之前，不能使用期接管master。此时metarestore程序会运行失败。  
metarestore程序是根据metalogger中定期下载的metadata和changelog来恢复master挂掉时刻master所记录的整个mfs的信息。  
        
   2.2  chunk和client相应的修改  
       1)对于client，需要umount掉mfs分区后，重启mfsmount新的master的IP地址。如果master挂掉之后，经过(1)重启服务器(2)使用同一台机器中metalogger恢复master数据(3)启动master；则client端不需要重新手动进行mfsmount，因为mfsmount会自动重试。  
       2)对于chunk，可以逐个chunk修改配置文件中master的IP地址，然后进行重启。如果master挂掉之后，经过(1)重启服务器(2)使用同一台机器中metalogger恢复master数据(3)启动master；则chunk不需要重启，master会在自动检测chunk的时候检测到它.  
  
 2.3 metalogger的注意事项  
     1)部署metalogger的服务器的open files数目必须大于或者等于5000，  
     2)metalogger在启动时不会下载metadata，而是等到第一个下载周期的下载时间点时再去下载，metalogger下载metadata的时间点是每小时的10分30秒，时间间隔是1小时的整数倍。  
     3) metalogger不是在启动的时候就取下载metadata，而是必须等到下载时间点(2中所述)才会去下载。也就是说，metalogger为确保正确性，要在启动后最少一个小时以内master和metalogger都要保持良好的状态。

Mfs写入：

 1.客户端在mount目录下面，对一个regular file调用write, 这一步是在用户空间执行

       2.write内部会调用虚拟文件系统提供的一致性接口vfs\_write

       3.根据FUSE模块注册的file\_operations信息，vfs\_write会调用fuse\_file\_aio\_write，将写请求放入fuse connection的request pending queue, 随后进入睡眠等待应用程序reply

       4.用户空间的libfuse有一个守护进程通过函数fuse\_session\_loop轮询杂项设备/dev/fuse, 一旦request queue有请求即通过fuse\_kern\_chan\_receive接收

       5.fuse\_kern\_chan\_receive通过read读取request queue中的内容，read系统调用实际上是调用的设备驱动接口fuse\_dev\_read

       6.在用户空间读取并分析数据，执行用户定义的write操作，将状态通过fuse\_reply\_write返回给kernel

       7.fuse\_reply\_write调用VFS提供的一致性接口vfs\_write

       8.vfs\_write最终调用fuse\_dev\_write将执行结果返回给第3步中等待在waitq的进程，此进程得到reply 后，write返回

# 图片写操作

由于安装了fuse,像操作本机一样操作moosefs

BufferedOutputStream buffer = new BufferedOutputStream(new FileOutputStream(path + filename));

buffer.write(bytes);

buffer.flush();

buffer.close();

# 图片读操作

由于安装了fuse,读取本机一样从moosefs读取数据

通过nginx来进行访问即可

server {

listen 8000;

server\_name imgServer;

location /test{

alias /mnt/mfs/folder1;

}

}

关于nginx、以及nginx作为图片服务器这块，请参考nginx的相关内容