1、分布式存储

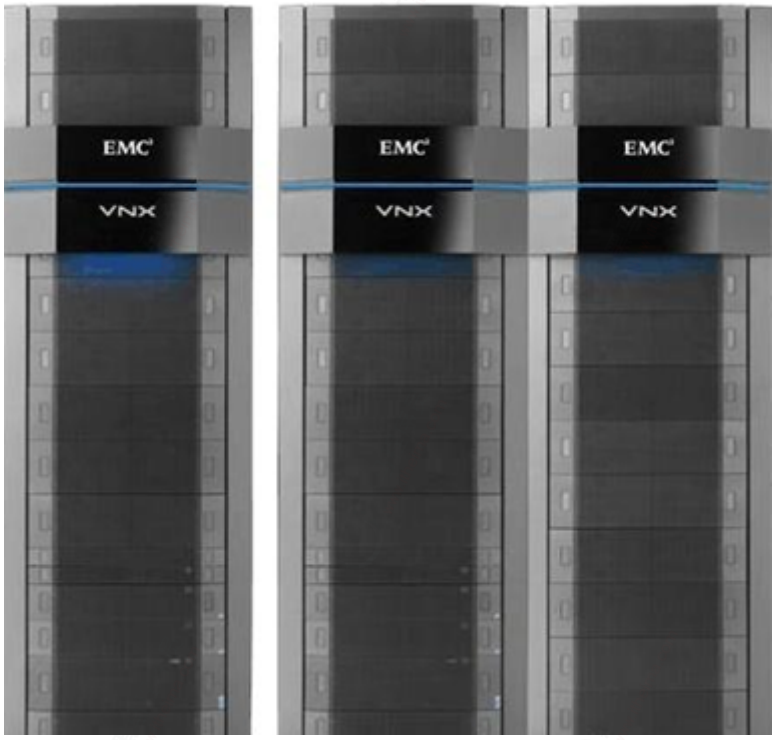
1.1 常用存储方案

> 在开始介绍分布式存储之前，先了解一下，常用的存储方案

1.1.1 集中存储结构

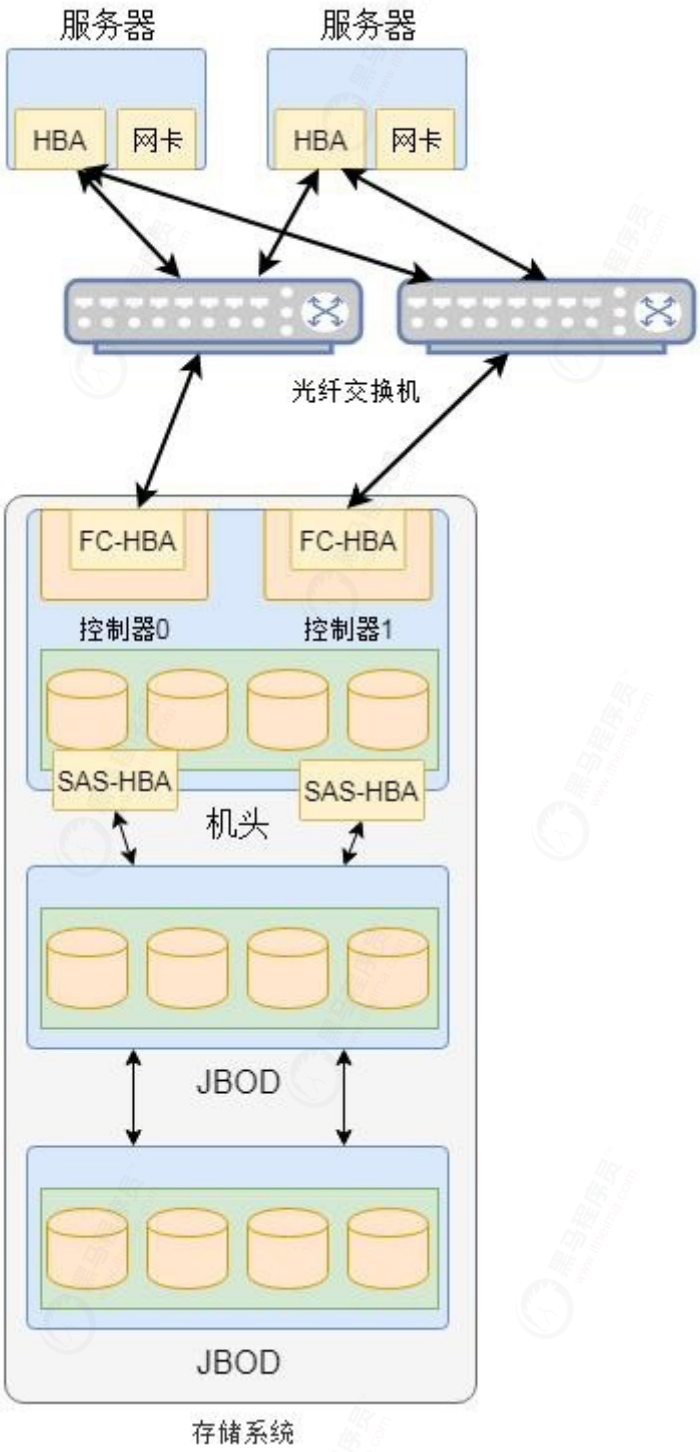
> 说到分布式存储，我们先来看一下传统的存储是怎么个样子。

传统的存储也称为集中式存储， 从概念上可以看出来是具有集中性的，也就是整个存储是集中在一个系统中的，但集中式存储并不是一个单独的设备，是集中在一套系统当中的多个设备，比如下图中的EMC 存储就需要几个机柜来存放。



在这个存储系统中包含很多组件，除了核心的机头(控制器)、磁盘阵列(JBOD)和交换机等设备外，还有管理设备等辅助设备。

结构下图所示：



1.1.2 分布式存储

分布式存储最早是由谷歌提出的，其目的是通过廉价的服务器来提供使用与大规模，高并发场景下的 Web 访问问题。它 采用可扩展的系统结构，利用多台存储服务器分担存储负荷，利用位置服务器定位存储信息，它不但提高了系统的可靠性、可用性和存取效率，还易于扩展。

1.2 分布式存储概述

1.2.1 分布式存储的兴起

分布式存储的兴起与互联网的发展密不可分，互联网公司由于其数据量大而资本积累少，而通常都使用大规模分布式存储系统。

与传统的高端服务器、高端存储器和高端处理器不同的是，互联网公司的分布式存储系统由数量众多的、低成本和高性价比的普通 PC 服务器通过网络连接而成。

> 其主要原因有以下三点

1. 互联网的业务发展很快，而且注意成本消耗，这就使得存储系统不能依靠传统的纵向扩展的方式，即先买小型机，不够时再买中型机，甚至大型机。互联网后端的分布式系统要求支持横向扩展，即通过增加普通 PC 服务器来提高系统的整体处理能力。
2. 普通 PC 服务器性价比高，故障率也高，需要在软件层面实现自动容错，保证数据的一致性。
3. 另外，随着服务器的不断加入，需要能够在软件层面实现自动负载均衡，使得系统的处理能力得到线性扩展。

1.2.2 分布式存储的重要性

从单机单用户到单机多用户，再到现在的网络时代，应用系统发生了很多的变化。而分布式系统依然是目前很热门的讨论话题，那么，分布式系统给我们带来了什么，或者说是为什么要有分布式系统呢?

升级单机处理能力的性价比越来越低

企业发现通过更换硬件做垂直扩展的方式来提升性能会越来越不划算;

单机处理能力存在瓶颈

某个固定时间点，单颗处理器有自己的性能瓶颈，也就说即使愿意花更多的钱去买计算能力也买不到了;

出于稳定性和可用性的考虑

如果采用单机系统，那么在这台机器正常的时候一切 OK ，一旦出问题，那么系统就完全不能用了。当然，可以考虑做容灾备份等方案，而这些方案就会让系统演变为分布式系统了;

云存储和大数据发展的必然要求

云存储和大数据是构建在分布式存储之上的应用。移动终端的计算能力和存储空间有限，而且有在多个设备之间共享资源的强烈的需求，这就使得网盘、相册等云存储应用很快流行起来。然而，万变不离其宗，云存储的核心还是后端的大规模分布式存储系统。大数据则更近一步，不仅需要存储海量数据，还需要通过合适的计算框架或者工具对这些数据进行分析，抽取其中有价值的部分。如果没有分布式存储，便谈不上对大数据进行分析。仔细分析还会发现，分布式存储技术是互联网后端架构的神器，掌握了这项技能，以后理解其他技术的本质会变得非常容易。

1.2.3 分布式存储的优势

可扩展

分布式存储系统可以扩展到数百甚至数千个这样的集群大小，并且系统的整体性能可以线性增长。

高可用性

在分布式文件系统中，高可用性包含两层，一是整个文件系统的可用性，二是数据的完整和一致性

低成本

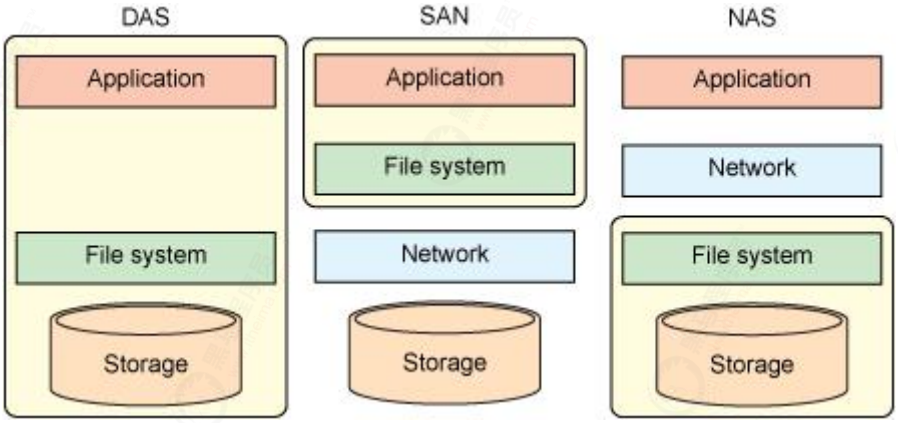
分布式存储系统的自动容错和自动负载平衡允许在成本较低服务器上构建分布式存储系统。此外，线性可扩展性还能够增加和降低服务器的成本。

弹性存储

可以根据业务需要灵活地增加或缩减数据存储以及增删存储池中的资源，而不需要中断系统运行

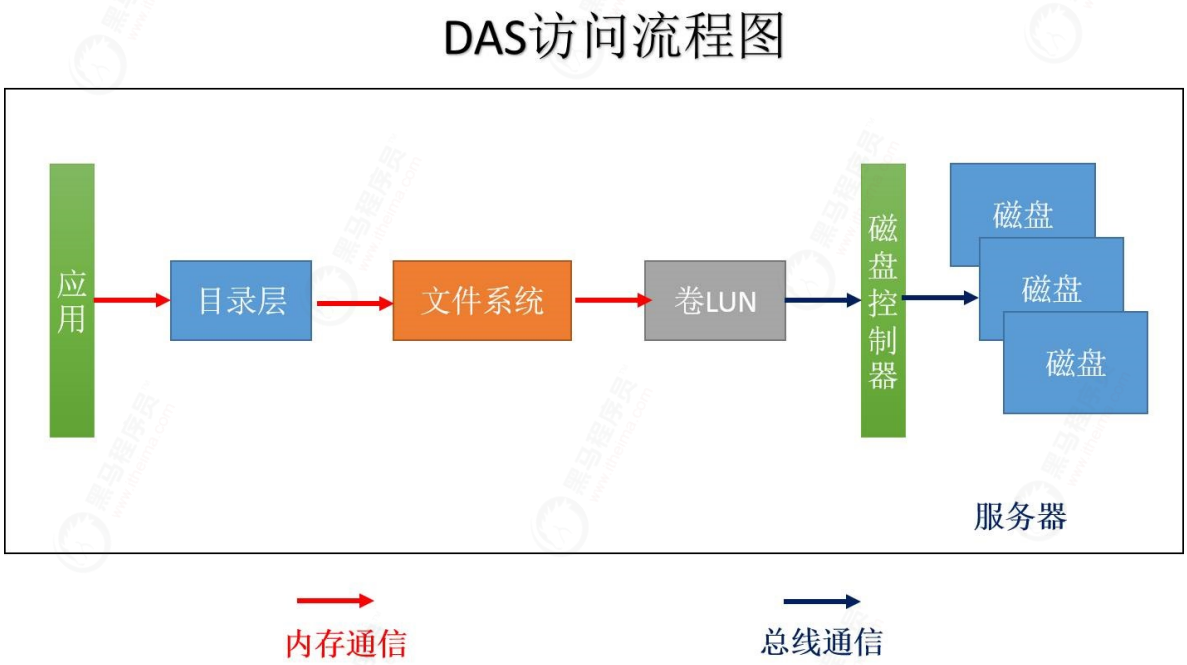
1.3 数据存储类型

> 一般情况下，我们将存储分成了4种类型，基于本机的DAS和网络的NAS存储、SAN存储、对象存储。对象存储是SAN存储和NAS存储结合后的产物，汲取了SAN存储和NAS存储的优点。



1.3.1 DAS

> DAS将计算、存储能力一把抓，封装在一个服务器里。大家日常用的电脑，就是一个DAS系统。

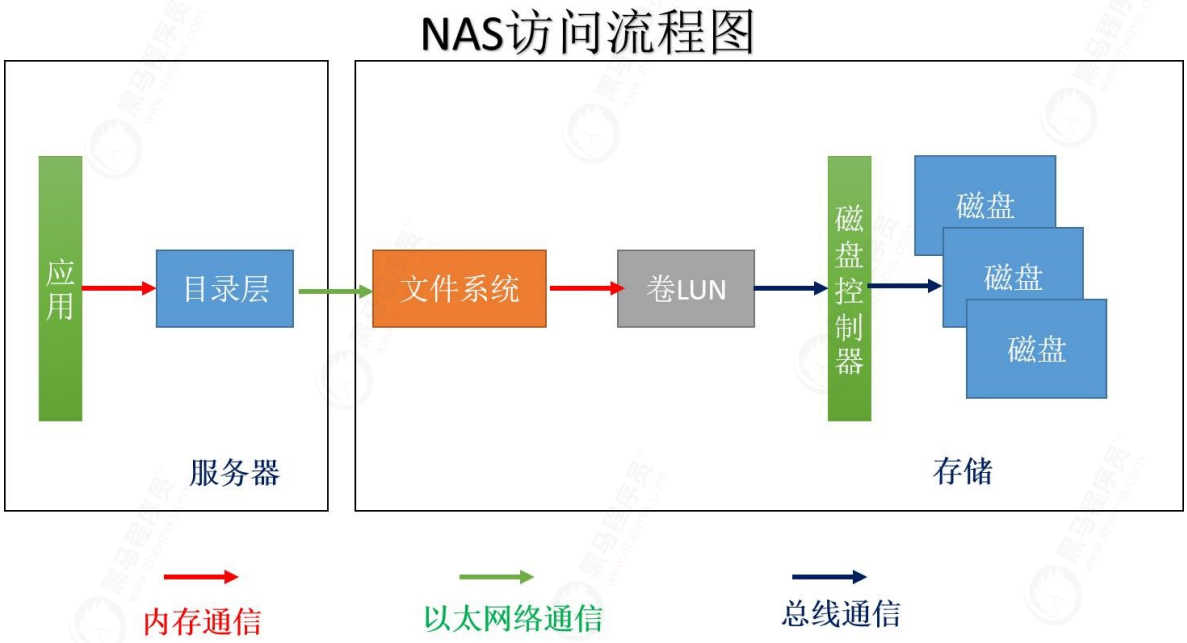


1.3.2 NAS

如果将计算和存储分离了，存储成为一个独立的设备，并且存储有自己的文件系统，可以自己管理数据，就是NAS

计算和存储间一般采用以太网络连接，走的是CIFS或NFS协议。服务器们可以共享一个文件系统，也就是说，不管服务器讲的是上海话还是杭州话，通过网络到达NAS的文件系统，都被翻译成为普通话。

所以NAS存储可以被不同的主机共享。服务器只要提需求，不需要进行大量的计算，将很多工作交给了存储完成，省下的CPU资源可以干更多服务器想干的事情，即计算密集型适合使用NAS。



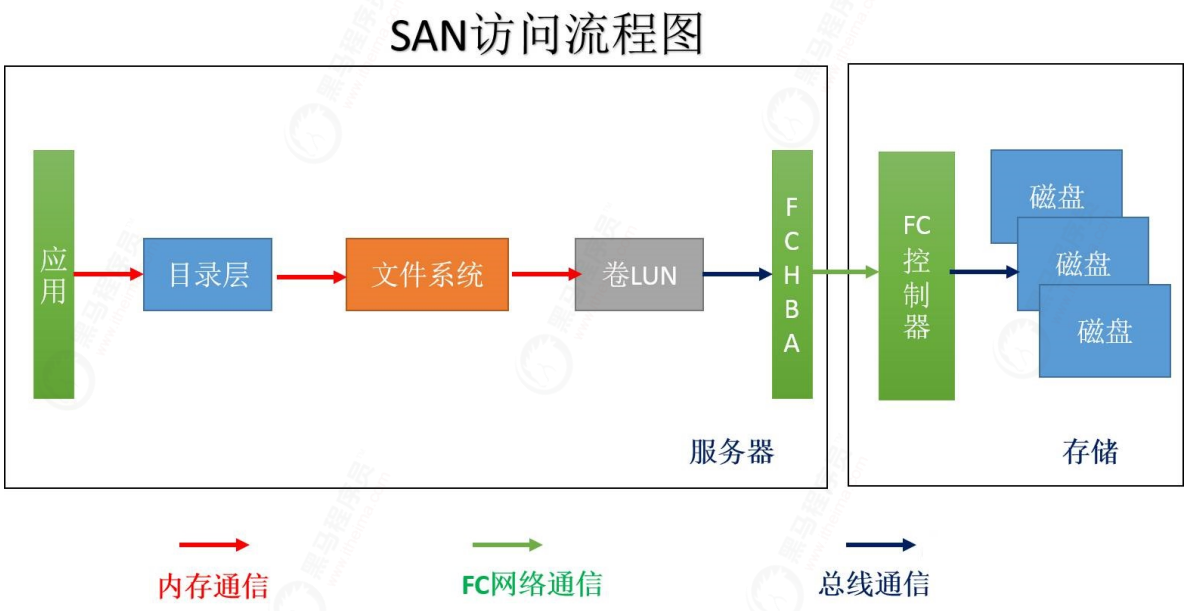
1.3.3 SNA

计算和存储分离了，存储成为一个独立的设备，存储只是接受命令不再做复杂的计算，只干读取或者写入文件2件事情，叫SAN

因为不带文件系统，所以也叫"裸存储"，有些应用就需要裸设备，如数据库。存储只接受简单明了的命令，其他复杂的事情，有服务器端干了。再配合FC网络，这种存储数据读取/写入的速度很高。

但是每个服务器都有自己的文件系统进行管理，对于存储来说是不挑食的只要来数据我就存，不需要知道来的是什么，不管是英语还是法语，都忠实记录下来的。

但是只有懂英语的才能看懂英语的数据，懂法语的看懂法语的数据。所以，一般服务器和SAN存储区域是一夫一妻制的，SAN的共享性不好。当然，有些装了集群文件系统的主机是可以共享同一个存储区域的。



2、主流分布式文件系统（了解）

2.1 分布式文件系统介绍

> 目前主流的分布式文件系统有：GFS、HDFS、Ceph、Lustre、MogileFS、MooseFS、FastDFS、TFS、GridFS等。

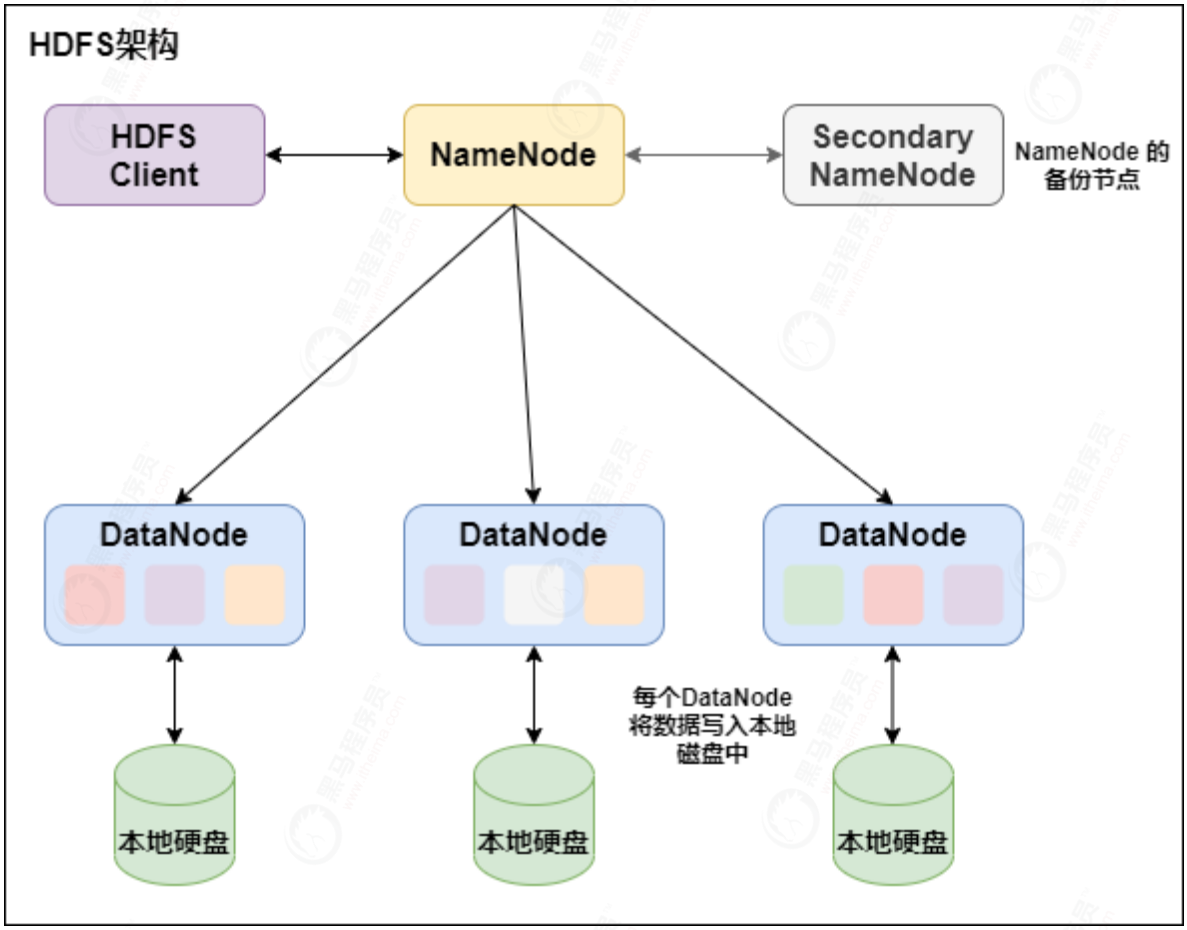
2.1.1 GFS（Google File System）

Google公司为了满足本公司需求而开发的基于Linux的专有分布式文件系统。尽管Google公布了该系统的一些技术细节，但Google并没有将该系统的软件部分作为开源软件发布。

2.1.2 HDFS（Hadoop Distributed File System）

HDFS（Hadoop Distributed File System）是 Hadoop 项目的一个子项目。

是 Hadoop 的核心组件之一， Hadoop 非常适于存储大型数据 (比如 TB 和 PB)，其就是使用 HDFS 作为存储系统. HDFS 使用多台计算机存储文件，并且提供统一的访问接口，像是访问一个普通文件系统一样使用分布式文件系统。



2.1.3 TFS（Taobao FileSystem）

TFS是一个高可扩展、高可用、高性能、面向互联网服务的分布式文件系统，主要针对海量的非结构化数据，它构筑在普通的Linux机器 集群上，可为外部提供高可靠和高并发的存储访问。

TFS为淘宝提供海量小文件存储，通常文件大小不超过1M，满足了淘宝对小文件存储的需求，被广泛地应用在淘宝各项应用中。它采用了HA架构和平滑扩容，保证了整个文件系统的可用性和扩展性。同时扁平化的数据组织结构，可将文件名映射到文件的物理地址，简化 了文件的访问流程，一定程度上为TFS提供了良好的读写性能。

2.1.4 Lustre

Lustre是一个大规模的、安全可靠的，具备高可用性的集群文件系统，它是由SUN公司开发和维护的。该项目主要的目的就是开发下一代的集群文件系统，可以支持超过10000个节点，数以PB的数据量存储系统。目前Lustre已经运用在一些领域，例如HP SFS产品等。

2.1.5 MooseFS

MooseFS是一款相对小众的分布式文件系统，不需要修改上层应用接口即可直接使用，支持FUSE的操作方式，部署简单并提供Web界面的方式进行管理与监控，同其他分布式操作系统一样，支持在线扩容，并进行横向扩展。MooseFS还具有可找回误操作删除的文件，相当于一个回收站，方便业务进行定制；同时MooseFS对于海量小文件的读写要比大文件读写的效率高的多。

但MooseFS的缺点同样明显，MFS的主备架构情况类似于MySQL的主从复制，从可以扩展，主却不容易扩展。短期的对策就是按照业务来做切分，随着MFS体系架构中存储文件的总数上升，Master Server对内存的需求量会不断增大。并且对于其单点问题官方自带的是把数据信息从Master Server同步到Metalogger Server上，Master Server一旦出问题Metalogger Server可以恢复升级为Master Server，但是需要恢复时间。目前，也可以通过第三方的高可用方案（heartbeat+drbd+moosefs）来解决 Master Server 的单点问题。

2.1.6 MogileFS

由memcahed的开发公司danga一款perl开发的产品，目前国内使用mogielFS的有图片托管网站yupoo等。MogileFS是一套高效的文件自动备份组件，由Six Apart开发，广泛应用在包括LiveJournal等web2.0站点上。

2.1.7 FastDFS

是一款类似Google FS的开源分布式文件系统，是纯C语言开发的。

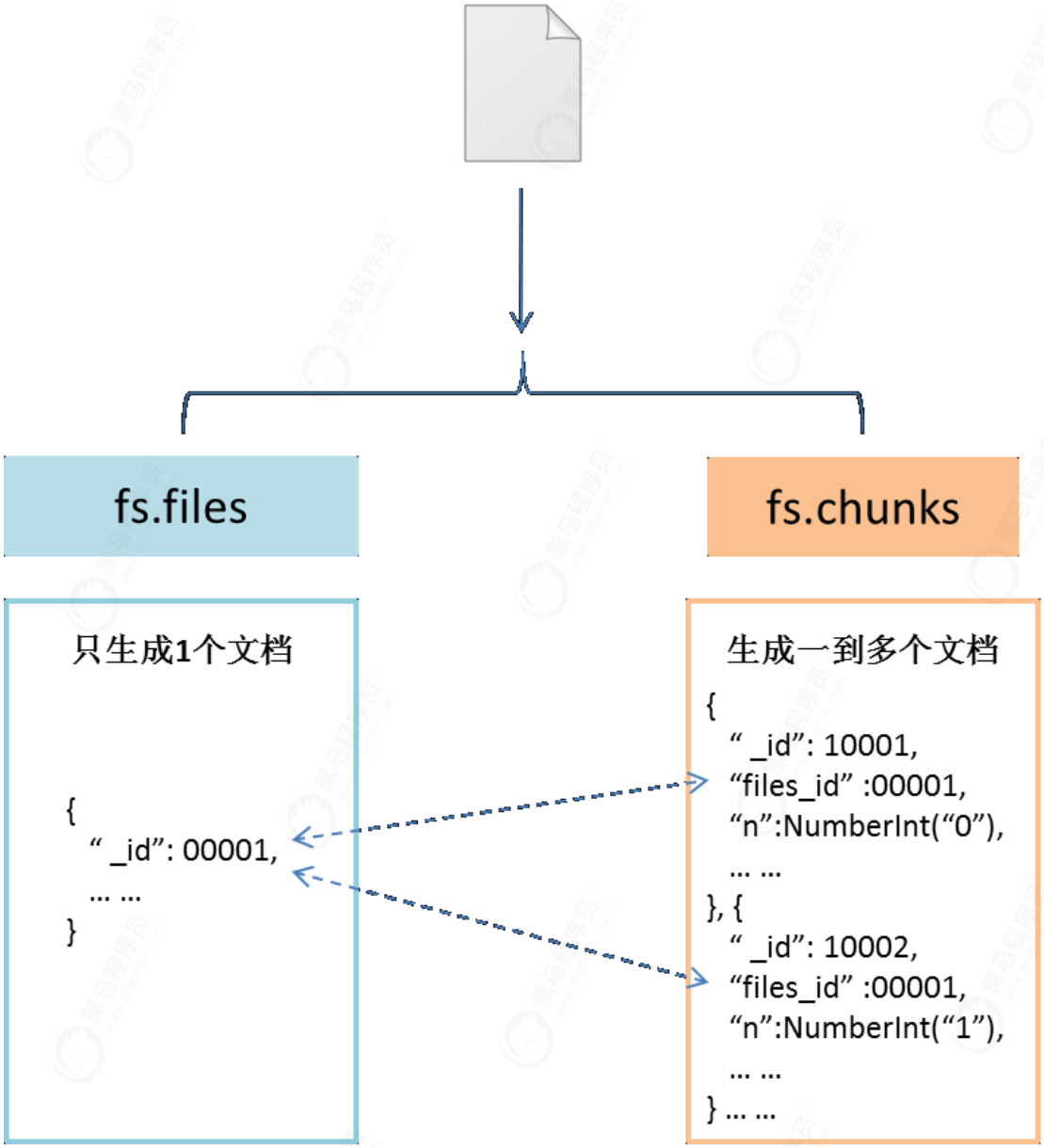
FastDFS是一个开源的轻量级分布式文件系统，它对文件进行管理，功能包括：文件存储、文件同步、文件访问（文件上传、文件下载）等，解决了大容量存储和负载均衡的问题。特别适合以文件为载体的在线服务，如相册网站、视频网站等等。

2.1.8 GlusterFS

开源分布式横向扩展文件系统，可以根据存储需求快速调配存储，内含丰富的自动故障转移功能，且摈弃集中元数据服务器的思想。适用于数据密集型任务的可扩展网络文件系统，具有可扩展性、高性能、高可用性等特点。gluster于2011年10月7日被Red Hat收购。

2.1.9 GridFS

MongoDB是知名的NoSQL数据库，GridFS是MongoDB的一个内置功能，它提供一组文件操作的API以利用MongoDB存储文件，GridFS的基本原理是将文件保存在两个Collection中，一个保存文件索引，一个保存文件内容，文件内容按一定大小分成若干块，每一块存在一个Document中，这种方法不仅提供了文件存储，还提供了对文件相关的一些附加属性（比如MD5值，文件名等等）的存储。文件在GridFS中会按4MB为单位进行分块存储。



2.2 分布式文件系统的对比

2.2.1 系统对比

文件系统	开发者	开发语言	开源协议	易用性	适用场景	特性	缺点
GFS	Google		不开源				
HDFS	Apache	Java	Apache	安装简单，官方文档专业化	存储非常大的文件	大数据批量读写，吞吐量高；一次写入，多次读取，顺序读写	难以满足毫秒级别的低延时数据访问；不支持多用户并发写相同文件；不适用于大量小文件
Ceph	加州大学圣克鲁兹分校Sage Weil	C++	LGPL	安装简单，官方文档专业化	单集群的大中小文件	分布式，没有单点依赖，用C编写，性能较好	基于不成熟的btrfs，自身也不够成熟稳定，不推荐在生产环境使用
TFS	Alibaba	C++	GPL V2	安装复杂，官方文档少	跨集群的小文件	针对小文件量身定做，随机IO性能比较高；实现了软RAID，增强系统的并发处理能力及数据容错恢复能力；支持主备热倒换，提升系统的可用性；支持主从集群部署，从集群主要提供读/备功能	不适合大文件的存储；不支持POSIX，通用性较低；不支持自定义目录结构与文件权限控制；通过API下载，存在单点的性能瓶颈；官方文档少，学习成本高
Lustre	SUN	C	GPL	复杂，而且严重依赖内核，需要重新编译内核	大文件读写	企业级产品，非常庞大，对内核和ext3深度依赖	

文件系统	开发者	开发语言	开源协议	易用性	适用场景	特性	缺点
MooseFS	Core Sp. z o.o.	C	GPL V3	安装简单，官方文档多，且提供Web界面的方式进行管理与监控	大量小文件读写	比较轻量级，用perl编写，国内用的人比较多	对master服务器有单点依赖，性能相对较差
MogileFS	Danga Interactive	Perl	GPL		主要用在web领域处理海量小图片	key-value型元文件系统；效率相比mooseFS高很多	不支持FUSE
FastDFS	国内开发者余庆	C	GPL V3	安装简单，社区相对活跃	单集群的中小文件	系统无需支持POSIX，降低了系统的复杂度，处理效率更高；实现了软RAID，增强系统的并发处理能力及数据容错恢复能力；支持主从文件，支持自定义扩展名；主备Tracker服务，增强系统的可用性	不支持断点续传，不适合大文件存储；不支持POSIX，通用性较低；对跨公网的文件同步，存在较大延迟，需要应用做相应的容错策略；同步机制不支持文件正确性校验；通过API下载，存在单点的性能瓶颈

文件系统	开发者	开发语言	开源协议	易用性	适用场景	特性	缺点
GlusterFS	Z RESEARCH	C	GPL V3	安装简单，官方文档专业化	适合大文件，小文件性能还存在很大优化空间	无元数据服务器，堆栈式架构(基本功能模块可以进行堆栈式组合，实现强大功能)，具有线性横向扩展能力；比mooseFS庞大	由于没有元数据服务器，因此增加了客户端的负载，占用相当的CPU和内存；但遍历文件目录时，则实现较为复杂和低效，需要搜索所有的存储节点，不建议使用较深的路径
GridFS	MongoDB	C++		安装简单	通常用来处理大文件（超过16M）	可以访问部分文件，而不用向内存中加载全部文件，从而保持高性能；文件和元数据自动同步	

2.2.2 特性对比

文件系统	数据存储方式	集群节点通讯协议	专用元数据存储点	在线扩容	冗余备份	单点故障	跨集群同步	FUSE挂载	访问接口
HDFS	文件	私有协议（TCP）	占用MDS	支持		存在	不支持	支持	不支持POSIX
Ceph	对象/文件/块	私有协议（TCP）	占用MDS	支持	支持	存在	不支持	支持	POSIX
Lustre	对象	私有协议（TCP）/RDAM（远程直接访问内存）	双MDS	支持	不支持	存在	未知	支持	POSIX/MPI
MooseFS	块	私有协议（TCP）	占用MFS	支持	支持	存在	不支持	支持	POSIX
MogileFS	文件	HTTP	占用DB	支持	不支持	存在	不支持	不支持	不支持POSIX
FastDFS	文件/块	私有协议（TCP）	无	支持	支持	不存在	部分支持	不支持	不支持POSIX
GlusterFS	文件/块	私有协议（TCP）/RDAM（远程直接访问内存）	无	支持	支持	不存在	支持	支持	POSIX
TFS	文件	私有协议（TCP）	占用NS	支持	支持	存在	支持	未知	不支持POSIX

3、选型参考

3.1 按特性分类

- 适合做通用文件系统的有：Ceph，Lustre，MooseFS，GlusterFS；
- 适合做小文件存储的文件系统有：Ceph，MooseFS，MogileFS，FastDFS，TFS；
- 适合做大文件存储的文件系统有：HDFS，Ceph，Lustre，GlusterFS，GridFS；
- 轻量级文件系统有：MooseFS，FastDFS；
- 简单易用，用户数量活跃的文件系统有：MooseFS，MogileFS，FastDFS，GlusterFS；
- 支持FUSE挂载的文件系统有：HDFS，Ceph，Lustre，MooseFS，GlusterFS。

3.2 初步筛选

> 考虑到GFS不开源，学习成本高，且相关特性资料不全面的情况下，暂时先不考虑使用GFS；

- Ceph部署设计复杂，生产环境运维成本很高，暂时排除；
- Lustre对内核依赖程度过重，且不易安装使用，暂时排除；
- TFS安装复杂，且官方文档少，不利于以后的学习使用，暂时先排除；
- 经初步筛选剩下的文件系统有：HDFS、MooseFS、MogileFS、FastDFS、GlusterFS、GridFS。

3.3 根据业务筛选

3.3.1 业务需求

- 因为我们主要用来存储海量图片信息，需要搭建一部管理身份认证图片以及车辆照片的文件系统，照片的文件类型主要是小图片，写操作量少，读操作量大，且对安全性要求较高。
- 随着系统在使用过程中数据量逐步庞大，图片的量会变得繁多，对图片读取速率要求尽可能高但不追求极致（无需到毫秒级）。
- 文件系统需要有较完善的冗余备份与容错机制，功能尽量精简耐用，安装配置应简单且适合于国产环境部署。

3.3.2 需求分析

- 根据需求，首选需要选择适合海量小图片存储的文件系统，适合的文件系统有：MooseFS，MogileFS，FastDFS。
- 其次需要支持冗余备份，适合的文件系统有：MooseFS、FastDFS、GlusterFS
- 符合条件1，2且功能精简的文件系统有：FastDFS

3.3.3 小结

FastDFS功能精简，支持在线扩容、冗余备份，部分支持跨集群同步，不支持FUSE挂载和POSIX访问接口，不存在单点故障，性能较好。

4、FastDFS概述

> FastDFS是用c语言编写的一款开源的分布式文件系统，它是由淘宝资深架构师余庆编写并开源。

FastDFS专为互联网量身定制，充分考虑了冗余备份、负载均衡、线性扩容等机制，并注重高可用、高性能等指标，使用FastDFS很容易搭建一套高性能的文件服务器集群提供文件上传、下载等服务。

4.1 为什么要使用fastDFS

通用的分布式文件系统的优点的是开发体验好，但是系统复杂性高、性能一般，而专用的分布式文件系统虽然开发体验性差，但是系统复杂性低并且性能高。fastDFS非常适合存储图片等那些小文件，fastDFS不对文件进行分块，所以它就没有分块合并的开销，fastDFS网络通信采用socket，通信速度很快。

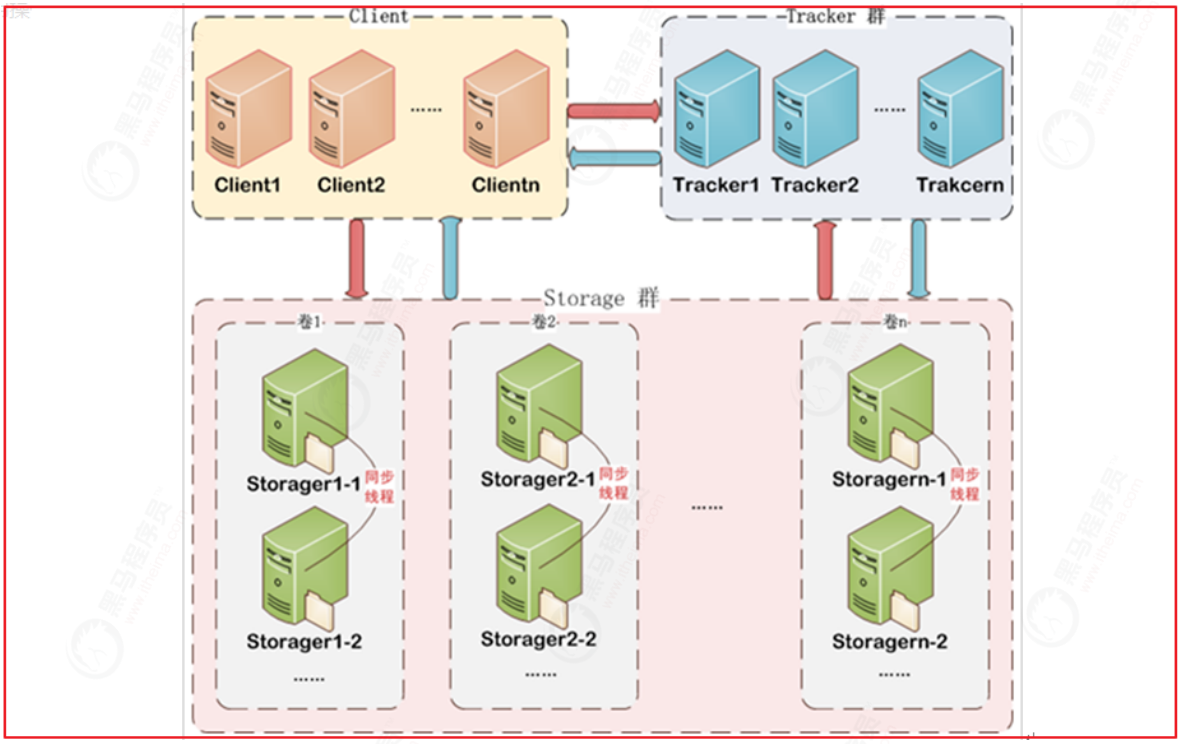
4.2 FastDFS体系结构

FastDFS是一个开源的轻量级分布式文件系统，它对文件进行管理，功能包括：文件存储、文件同步、文件访问（文件上传、文件下载）等，解决了大容量存储和负载均衡的问题。特别适合以文件为载体的在线服务，如相册网站、视频网站等等。

FastDFS为互联网量身定制，充分考虑了冗余备份、负载均衡、线性扩容等机制，并注重高可用、高性能等指标，使用FastDFS很容易搭建一套高性能的文件服务器集群提供文件上传、下载等服务。

FastDFS 架构包括 Tracker server 和 Storage server。客户端请求 Tracker server 请求文件信息，通过 Tracker server 调度最终由 Storage server 完成文件上传和下载。

Tracker server 作用是负载均衡和调度，通过 Tracker server 在文件上传时可以根据一些策略找到 Storage server 提供文件上传服务。可以将 tracker 称为追踪服务器或调度服务器。Storage server 作用是文件存储，客户端上传的文件最终存储在 Storage 服务器上，Storageserver 没有实现自己的文件系统而是利用操作系统的文件系统来管理文件。可以将storage称为存储服务器。



4.2.1 Tracker

Tracker Server作用是负载均衡和调度，通过Tracker server在文件上传时可以根据一些策略找到Storage server提供文件上传服务。可以将tracker称为追踪服务器或调度服务器。

FastDFS集群中的Tracker server可以有多台，Tracker server之间是相互平等关系同时提供服务，Tracker server不存在单点故障。客户端请求Tracker server采用轮询方式，如果请求的tracker无法提供服务则换另一个tracker。

4.2.2 Storage

Storage Server作用是文件存储，客户端上传的文件最终存储在Storage服务器上，Storage server没有实现自己的文件系统而是使用操作系统的文件系统来管理文件。可以将storage称为存储服务器。

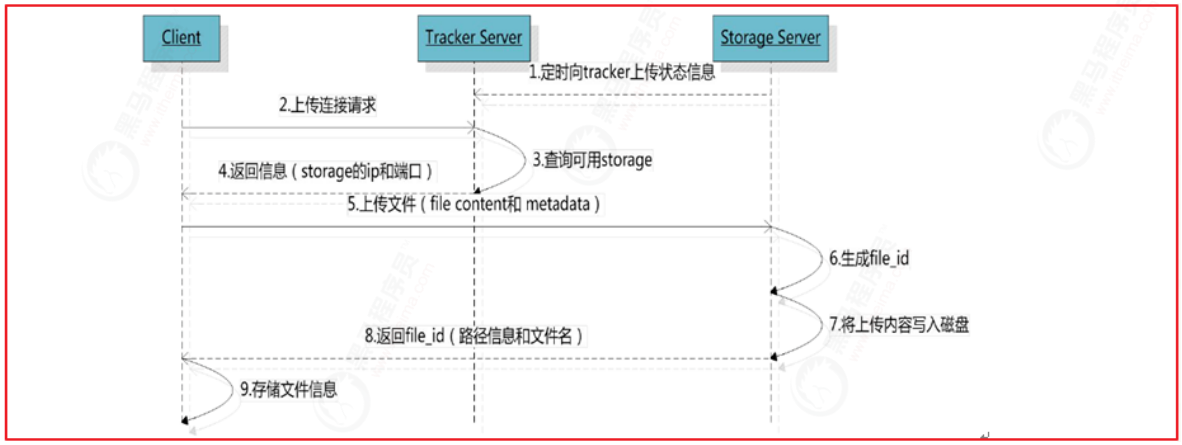
Storage集群采用了分组存储方式。storage集群由一个或多个组构成，集群存储总容量为集群中所有组的存储容量之和。一个组由一台或多台存储服务器组成，组内的Storage server之间是平等关系，不同组的Storage server之间不会相互通信，同组内的Storage server之间会相互连接进行文件同步，从而保证同组内每个storage上的文件完全一致的。一个组的存储容量为该组内的存储服务器容量最小的那个，由此可见组内存储服务器的软硬件配置最好是一致的。

采用分组存储方式的好处是灵活、可控性较强。比如上传文件时，可以由客户端直接指定上传到的组也可以由tracker进行调度选择。一个分组的存储服务器访问压力较大时，可以在该组增加存储服务器来扩充服务能力（纵向扩容）。当系统容量不足时，可以增加组来扩充存储容量（横向扩容）。

4.2.3 Storage状态收集

Storage server会连接集群中所有的Tracker server，定时向他们报告自己的状态，包括磁盘剩余空间、文件同步状况、文件上传下载次数等统计信息。

4.3 文件上传流程



客户端上传文件后存储服务器将文件ID返回给客户端，此文件ID用于以后访问该文件的索引信息。文件索引信息包括：组名，虚拟磁盘路径，数据两级目录，文件名。

group1 /M00 /02/44/ wKgDrE34E8wAAAAAAAAGkEIYJK42378.sh

4.3.1 组名

文件上传后所在的 storage 组名称，在文件上传成功后有storage 服务器返回，需要客户端自行保存。

4.3.2 虚拟磁盘路径

storage 配置的虚拟路径，与磁盘选项store_path对应。如果配置了,store_path0 则是 M00，如果配置了 store_path1 则是 M01，以此类推。

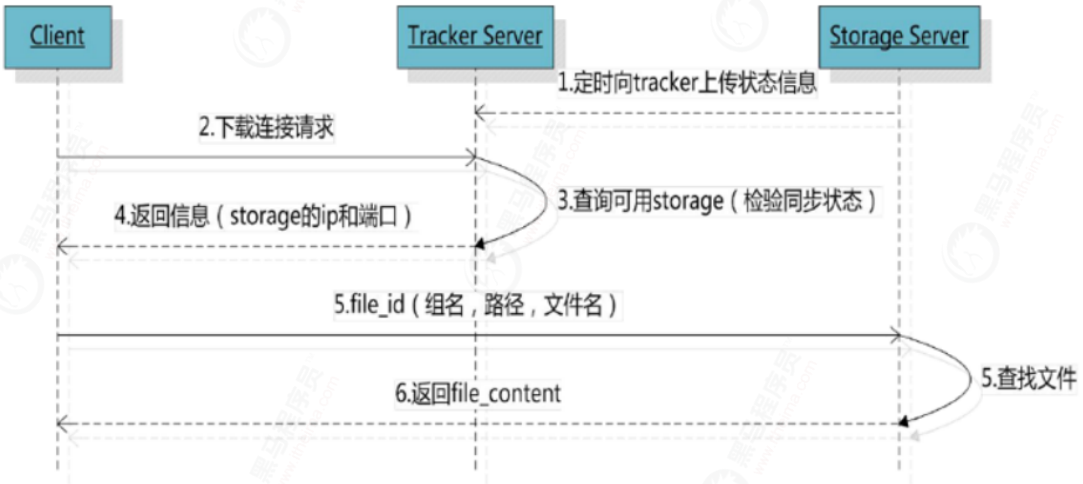
4.3.3 数据两级目录

storage 服务器在每个虚拟磁盘路径下创建的两级目录，用于存储数据文件。

4.3.4 文件名

与文件上传时不同。是由存储服务器根据特定信息生成，文件名包含：源存储服务器 IP 地址、文件创建时间戳、文件大小、随机数和文件拓展名等信息。

4.4 文件下载流程



> tracker根据请求的文件路径即文件ID 来快速定义文件。

group1 /M00 /02/44/ wKgDrE34E8wAAAAAAAAGkEIYJK42378.sh

1. 通过组名tracker能够很快的定位到客户端需要访问的存储服务器组是group1，并选择合适的存储服务器提供客户端访问。

2. 存储服务器根据“文件存储虚拟磁盘路径”和“数据文件两级目录”可以很快定位到文件所在目录，并根据文件名找到客户端需要访问的文件。

5、FastDFS搭建

5.1 原始安装

需要c++等编译环境

详细参考，一步步操作即可。难度不大但是需要自己一步步编译：

https://github.com/happyfish100/fastdfs/wiki

5.2 docker方式

直接使用编译打好包的docker镜像是最快捷的方式

https://hub.docker.com

参考：资料/fastdfs下的 docker-compose.yml , nginx.conf

6、java调用

6.1 pom包配置

> 引入fastDFS的坐标

```
<dependency>
    <groupId>com.github.tobato</groupId>
    <artifactId>fastdfs-client</artifactId>
    <version>1.26.4</version>
</dependency>
```

6.2 FastDfs配置文件

```
fdfs:
  so-timeout: 3000
  connect-timeout: 1000
  thumb-image:
    width: 60
    height: 60
  tracker-list:
    - 116.62.213.90:22122
```

6.3 顺风车源码

6.3.1 DFS工具类

```
@Component
```

```
public class AttachmentHandler {

    private static final Logger logger =
LoggerFactory.getLogger(AttachmentHandler.class);

    @Autowired
    private AttachmentMapper attachmentMapper;

    @Autowired
    private FastFileStorageClient storageClient;

    public ResponseVO<AttachmentPO> uploadFile(MultipartFile file) throws
Exception {
        if (file.isEmpty()) {
            throw new BusinessRuntimeException(BusinessErrors.DATA_NOT_EXIST,
"文件不存在");
        }
        AttachmentPO attachmentPO = getAttachmentPO(file);
        AttachmentPO tmp = attachmentMapper.selectByMd5(attachmentPO.getMd5());
        if (null != tmp) {
            return ResponseVO.success(tmp);
        }
        String url = dfsUploadFile(file);
        attachmentPO.setUrl(url);
        attachmentMapper.insert(attachmentPO);
        return ResponseVO.success(attachmentPO);
    }


    private AttachmentPO getAttachmentPO(MultipartFile file) throws IOException
{
        AttachmentPO attachmentPO = new AttachmentPO();
        attachmentPO.setName(file.getOriginalFilename());
        attachmentPO.setLenght(file.getSize());

 attachmentPO.setExt(StringUtils.getFilenameExtension(file.getOriginalFilename()
));
        attachmentPO.setMd5(CommonsUtils.fileSignature(file.getBytes()));
        return attachmentPO;
    }

    /**
     * 上传文件
     */
    private String dfsUploadFile(MultipartFile multipartFile) throws Exception {
        String originalFilename = multipartFile.getOriginalFilename().
                substring(multipartFile.getOriginalFilename().
                        lastIndexOf(".") + 1);
        StorePath storePath = this.storageClient.uploadImageAndCrtThumbImage(
                multipartFile.getInputStream(),
                multipartFile.getSize(), originalFilename, null);
        return storePath.getFullPath();
    }

    /**
     * 删除文件
     */
    private void dfsDeleteFile(String fileUrl) {
```

```
        if (StringUtils.isEmpty(fileUrl)) {
            logger.info("fileUrl == >>文件路径为空...");
            return;
        }
        try {
            StorePath storePath = StorePath.praseFromUrl(fileUrl);
            storageClient.deleteFile(storePath.getGroup(), storePath.getPath());
        } catch (Exception e) {
            logger.info(e.getMessage());
        }
    }

}
```

6.3.2 文件上传Controller

```
@RestController
@RequestMapping("/api/")
@Api(value = "文件操作Controller", tags = {"文件管理"})
@ApiResponses(@ApiResponse(code = 200, message = "处理成功"))
public class APIController {


    @Autowired
    private AttachmentHandler attachmentHandler;


    @ApiOperation(value = "文件上传接口", tags = {"文件管理"})
    @PostMapping("/upload")
    public ResponseVO<AttachmentPO> upload(@RequestParam("file") MultipartFile
file) throws Exception {
        return attachmentHandler.uploadFile(file);
    }
}
```