**背景**

考虑到线上的业务需求，要存储大量的业务现场日志，包括原始报文、HTTP请求内容或应答内容、原始邮件、格式文档、音视频数据等，这些非结构化数据体量巨大，每秒并发量较高，且大多数呈现为0~100KB左右的小文件，其比例达到了90%以上。如此规模的小文件，采用单个文件独立存储在磁盘的的方案已不可用。

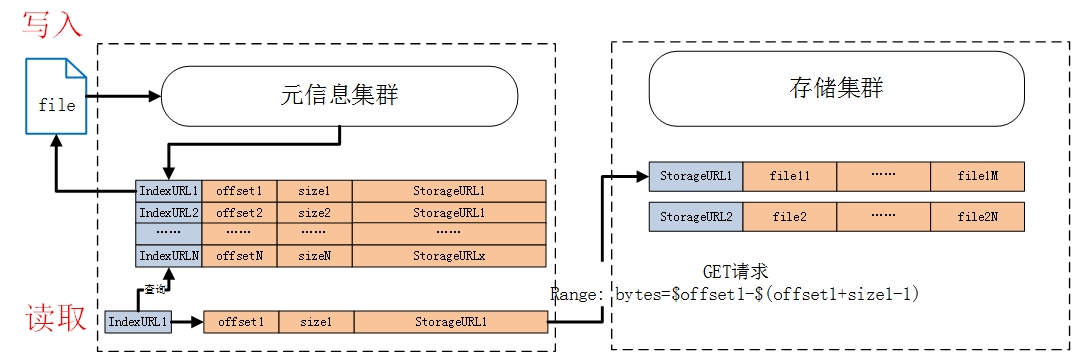
海量小文件存储是存储界的难题，目前的文件系统，包括本地文件系统、分布式文件系统和对象存储系统，都是主要针对大文件设计的，比如XFS/EXT4、Lustre、GlusterFS、GPFS、ISLION、GFS、HDFS，在元数据管理、数据布局、条带设计、缓存管理等实现策略上都侧重大文件，而海量小文件应用在性能和存储效率方面要大幅降低，甚至无法工作。以EXTx文件系统写数据为例，向磁盘写入数据进行大量的元数据操作，包括更新inode目录、目录、inode和数据块位图等。当操作连续大文件时，对元数据的操作开销可被庞大的数据操作开销分摊，但小文件的有效读写率小于大文件的，当小文件数量急剧增加时，对大量元数据的操作会严重影响系统的性能。

（参考<https://blog.csdn.net/zhengshifeng123/article/details/53198745>）

**海量小文件存储原理**

针对海量小文件存储，优化的方案是采用小文件合并的策略，即将若干个小文件合并为一个大文件，按照位置偏移和大小进行索引。这样若干个小文件的inode就减少为了一个，大大减轻了磁盘管理文件的负担。

小文件的元信息采用独立的外部存储实现，本方案采用的是redis。元信息的内容包括文件的类型、用户元数据、小文件实际存储位置（大文件的URL）、在大文件中的偏移offset、小文件的实际大小等内容。



文件块大小的决定：对小文件进行合并时，要限制文件块的大小，不宜过大，也不宜太小。经过实际测试，发现文件块大小为4MB~10MB都可以将磁盘、网络带宽打满，对于1~100KB的小文件来说足够。

考虑到后端对象存储不支持追加的操作，用户写入不宜延迟太大的问题，对文件块的大小作了另一层限制，即根据时间进行。当用户写入若干小文件，超过一定时间，文件累计大小依然没有达到文件块的大小时，也要进行存储，避免等待过长时间从界面上无法获取文件内容。此时，文件块的大小可能并未达到理想的阈值。

**MesaFS设计简介**

MesaFS实现海量小文件存储特点是实时性、高吞吐，作为非结构化消息，需要提供文件内容的获取路径（URL）。同时，该URL通常与结构化日志进行关联，通过结构化日志的字段可以通过HTTP的形式访问实际存储的内容。

FastDFS返回文件存储路径的方式是通过HTTP的响应内容进行的。MesaFS提供了用于存储小文件的客户端，用户在调用客户端API后会立即得到一个存储路径，以便用户能够实时发送结构化消息日志。

（采用提供客户端动态库的方式是由于前端程序需要一个直接调用的库，无需再开发新的客户端。弊端是在众多前端程序之间无法做小文件的合并，这需要提供一个后端服务来收集所有前端的小文件。但是使用后端服务的方式时，前端动态库无法直接提供最终的小文件检索路径，需要演变为fastDFS那种方式，这个后续再讨论。）

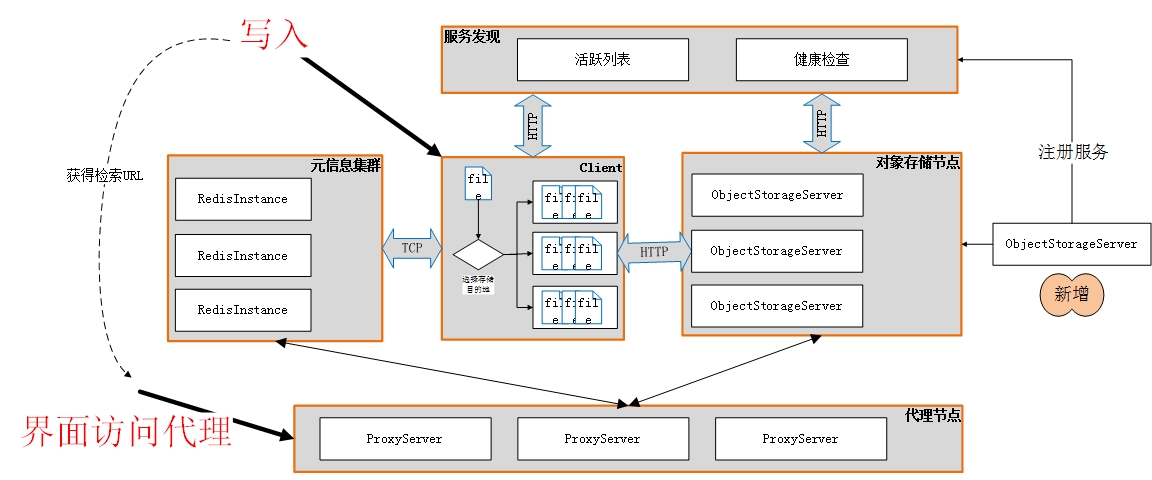
对于小文件合并存储，涉及了两种URL：第一种是小文件的检索URL，用户发起存储请求时，存储系统应返回该URL，作为下次获取内容时使用；第二种是对象存储URL，若干个小文件合并存储后会形成一个大文件，该URL用于检索该大文件的内容。

基于此，返回小文件存储路径有两种方法：

第一种：以大文件的存储URL、offset、size三者的组合作为小文件的URL。例如，大文件的URL为<http://192.168.1.10:9000/bucketname/large_file_url>，则小文件的URL为<http://192.168.1.10:9000/bucketname/large_file_url?offset=x&size=y>，其中offset表示小文件在大文件中的偏移，size表示小文件的大小。该种方案不需要用户输入小文件的文件名。用户在使用小文件索引URL获取文件内容时，需要经过代理进行转换，将URL的参数offset、size转换为请求头Range。

第二种：需要用户输入小文件的文件名，MesaFS实现小文件文件名与最终存储的大文件之间的映射。此时，代理无法从URL中获取小文件的偏移和大小，因此需要访问redis元信息库进行查询。查询到元信息之后，根据元信息中的大文件URL、小文件的偏移和大小，再向存储系统发起内容获取请求。

MesaFS选择了第二种方案，原因是第一种方案无法实现用户小文件的文件名和返回的索引URL之间的映射，需要用户自己实现映射。从界面访问时是以索引URL进行的，当需要存储原始文件内容时，无法得知小文件的类型。此外，为了支持获取文件的用户元信息，也需要在代理层访问redis元信息库，此时，代理访问redis元信息库是一种附加操作，并未引入其他网络操作，只不过需要在代理层增加用户元信息的解析。



MesaFS的存储架构如图所示，主要包含了服务发现、redis集群元信息存储库、对象存储节点、对象存储代理节点以及客户端动态库。

**写入操作流程**：用户直接使用提供的客户端动态库，调用相关的API，将文件名、文件内容传递给客户端动态库，客户端会根据文件累积大小、超时时间两种策略决定何时进行上传。客户端会将文件内容写入对象存储集群，同时返回给用户一个索引URL，将文件的元信息写入redis动态库。

**读取操作流程**：有两种读取方法：第一种是用户调用客户端动态库，传入要获取的文件名，客户端会以回调函数的方式返回文件内容；第二种是基于外部访问，比如通过界面访问，用户在写入的时候，客户端返回了一个该文件的索引URL，可以直接基于该URL获取文件内容。实际上，该URL指向的并非是对象存储集群，而是对象存储代理节点，代理节点会根据URI访问redis元信息库，查询文件的实际存储路径、偏移位置和大小，再根据元信息访问对象存储集群，获取文件内容。

**服务发现**

MesaFS支持动态扩容。第一层是redis元信息库扩容，使用的是集群模式，参见redis集群扩容方法。第二层是对象存储节点扩容，当需要进行对象存储节点扩容时，无需对已有的部署架构做任何调整，即不需要修改配置文件，也不需要重启任何程序，平滑实现节点扩增。这依赖于服务发现功能。

为了均衡对象存储节点的压力，客户端动态库会根据文件的存储路径的哈希值选择目标对象存储节点。为了实现故障转移和节点扩容，需要一种机制动态管理对象存储活跃列表，新增活跃节点时，客户端能够获取到这一事件。但节点故障时，能够检测到故障的节点，并将其从活跃列表中移除。该机制参考《Wired\_Load\_Balancer设计方案.docx》。

所有的对象存储节点都需要注册一个服务，通过服务实现节点的健康状态检查和活跃节点列表维护。客户端从服务中心拉取活跃的对象存储节点。

**对象存储节点**

目前的对象存储服务是基于Minio的，采用的是Amazon AWS S3 API。由于Minio的备份机制是在所有节点之间进行文件切割并备份，用户向任何一个节点发起GET请求，并获取到原始文件内容。但是这种方式对集群规模有限制（最大32台节点），且过多的节点间文件切分会导致性能低下。

MesaS设计之初就考虑到支持了任意后端对象存储集群的部署方式，采用集群方式、或是采用单机模式的对象存储节点，MesaFS并不关心。对于无可靠性要求的情形，为了高性能，可以采用单机的模式；对于有可靠性要求的场景，可以部署若干小集群，多个小集群构成一个大集群，小集群内部实现故障迁移和数据备份。

对象存储集群可以采用Ceph，它支持用户配置副本数，从而避免在整个大集群内都进行文件切片。Ceph目前并不推荐在生产环境使用，其安装部署运维比较复杂，可以作为备用方案使用。

存储集群的扩展比较灵活，具体的扩展方式是根据采用的对象存储服务进行的，或是Minio或是Ceph。当扩展完毕对象存储节点后，需要在Consul中注册新增的节点服务，以便客户端能够发现这一事件。MesaFS提供了注册服务的脚本。

采用对象存储方案的原因在于：前端程序处理流量是流式进行的，要求非结构化日志文件能够流式进行存储，能够支持超大文件（上GB的文件）存储。对于超大文件，在前端程序内并不缓存完整的文件内容，而是以分段上传的方式进行存储。对象存储支持这一特性，且基于HTTP restful接口，为我们编写客户端程序提供了很好的自主性。对于缓存应用，客户端是基于网络异步模式驱动的，要求其存储数据与业务处理在同一个线程内进行。对于众多其他开源工具而言，必须使用定制的客户端动态库才能进行数据存储，这对我们的使用场景限制较多。

**redis元信息存储集群**

小文件合并存储的一大特性是要求具备文件元信息的存储，改变了文件元信息以inode节点存储的方式。

Facebook的haystack系统自己实现了一套元信息存储机制。FastDFS有专门的tracker server来进行协调。MesaFS根据自身的特点，选用了可线性扩展的redis集群作为元信息存储方案。首先，Redis异步访问速率非常块，不会成为系统的瓶颈；其次，redis支持线性扩展，支持持久化、支持备份；再次，代理模块能够经过稍微改变，使用redis进行元信息检索；还有，redis支持超时，能够设置存储时间，并自动进行淘汰，这非常适合做存储时间限制，无需运维人员手动进行移除过期文件；最后，redis支持发布订阅机制，开发者能够基于此开发其他附加的业务，比如对Minio对象存储进行缓存管理、对非结构化数据进行多维度统计等。

redis存储小文件元信息是JSON格式，支持了Content-Type、Content-Encoding、Content-Disposition、Content-Length标准HTTP头部元信息，支持用户自定义二进制私有元数据。针对HEAD请求，直接从redis中获取元信息，无需访问对象存储集群，具有非常高的速率。

**客户端动态库**

对象存储集群、Redis集群只需进行安装部署，即可调用客户端动态库进行非结构化日志文件存取。

对于需要扩容对象存储节点、开启对象存储故障转移时，客户端动态库基于WiredLB动态库实现了活跃节点列表维持。

为了提供上传下载的速率，客户端针对每一个TCP链接，采用了HTTP Pipeline技术，无需等待前一次的应答即可发起下一次的请求。为避免服务器压力过大，客户端对pipeline的并发数进行了限制，当并发http请求达到阈值时，客户端将拒绝服务。

客户端支持三种存储模式：

1. Minio存储模式。只使用Minio作为存储服务，每个文件是一个object，元信息和文件内容都存储在Minio，无需redis。一般在测试环境使用。

2. LOSF模式，即Lots Of Small Files，海量小文件存储模式。该模式会对小文件进行合并存储，元信息存储在redis中，文件内容合并存储在对象存储服务。

3. Cache模式，针对HTTP代理缓存模式使用。该模式将小文件、元信息都存储在Redis中，大文件存储在对象存储服务。

客户端支持多种API，主要有两种：

1. 异步线程内API。该种模式是针对异步编程使用，当调用者所开发的程序是异步模式的，采用该种API。能够保证日志存储与业务处理在同一个线程内，客户端不会开启额外的线程。

支持三种模式的API接口：

（1）完整文件上传接口，用户可以将一个完整的文件，通过一个API调用即可实现上传存储。

（2）流式上传接口，对于网络流场景下，用户可以通过多次调用API，每次传入一个文件片段的方式进行流式存储。客户端会根据最终的文件大小决定采用分段上传还是合并上传。

（3）流式GET接口，用户通过调用一个API，设置回调函数，客户端会把文件元信息、文件内容通过多次回调函数的调用，返回给用户。

2. 独立线程API。该种模式下，客户端会创建独立的线程进行文件上传、获取操作，此时业务处理所在的线程不是异步驱动，或无法采用异步驱动。该种模式下的API与第一种相同，也提供了另外三种API。

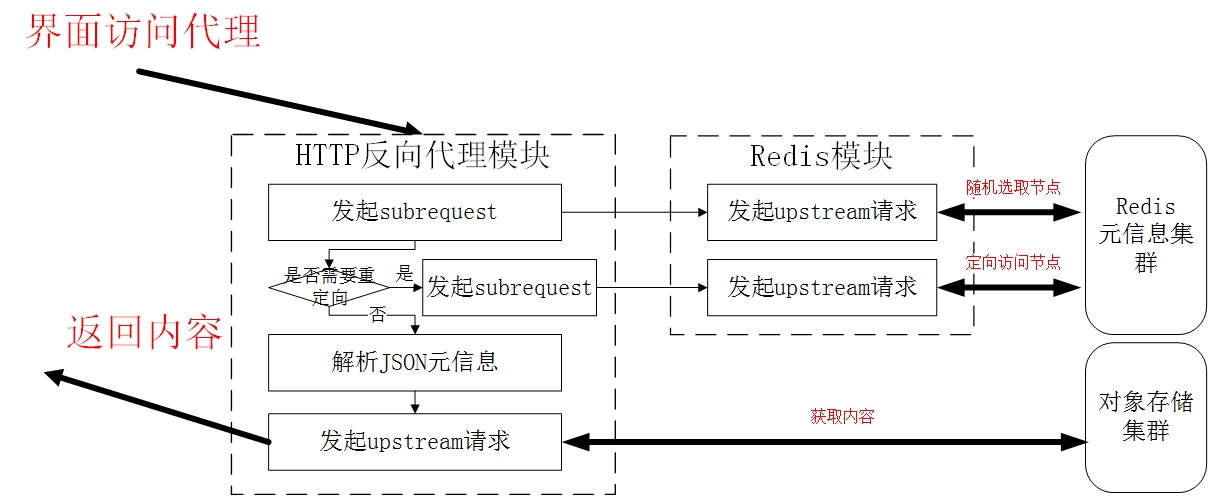
**对象存储代理节点**

前文述及，LOSF采用的小文件索引URL并不具备小文件的存储位置信息，该URL未标识小文件的最终存储路径，而是将小文件的URL与最终的对象存储路径进行了映射，元信息存储在了redis中。因此，直接访问小文件索引URL是无法获取实际的文件内容的。需要在对象存储、Redis元信息之前加入一层代理。

通过加入该层代理，用户能够从界面上结构化日志的相关字段，点击小文件的索引URL，即可完成小文件内容的下载。

MesaFS采用的代理是openresty（基于Nginx的改进版），openresty支持了redis单节点的访问，并未支持redis集群。

由于openrersty并不支持redis集群，且redis内部元信息是自定义的JSON格式，openresty自身无法对其解析。MesaFS在openresty的Redis模块、HTTP代理模块的基础上，对原生的功能进行了修改，支持了redis元信息的解析。其工作模式如下图。



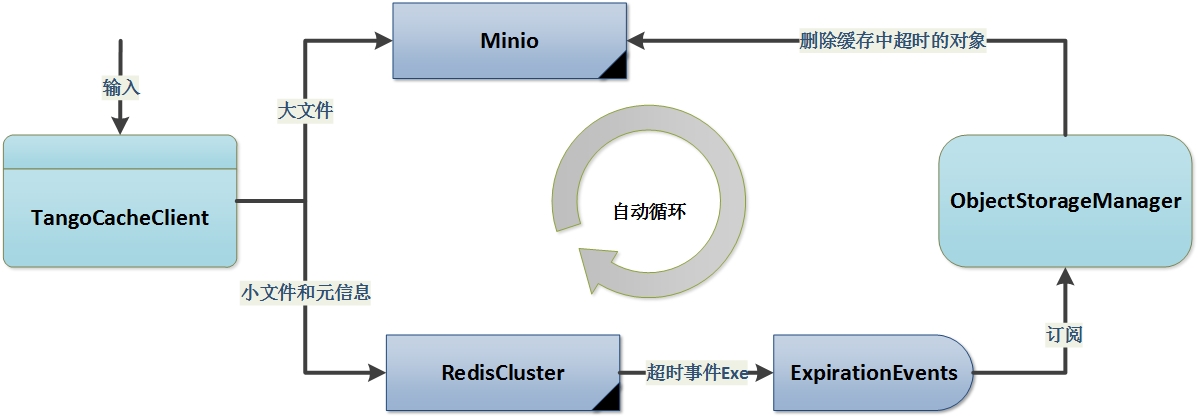
MesaFS在Nginx的ngx\_http\_proxy\_module以及redis2-nginx-module两个模块的基础上，对其修改，采用subrequest和upstream多级组合的方式，新增了Nginx的directive，定制实现了MesaFS海量小文件获取的方式。

当用户点击小文件URL时，该http请求被传送到了Nginx的HTTP反向代理模块，该模块会通过发起子请求，通过最多两级子请求就能够定位元信息存储的最终redis节点。当查询到redis文件元信息后，解析元信息得到小文件在对象存储中的实际位置、偏移和大小，再通过发起upstream请求，将文件内容透传到用户。

**缓存管理模块**

使用对象存储服务的一大问题是：存储的非结构化日志文件具有其生命周期，即具有存储时长方面的要求。这是由于磁盘、内存空间有限，不能持续无限制地存储，而Minio对象存储服务不支持动态超时淘汰。当所存储的小文件生命周期到达时，期望能够自行删除，而不是让运维人员定期手动删除，这会持续增加运维人员的工作量，且集中删除会影响对象存储正常的服务。

MesaFS基于该种需求，实现了基于redis发布订阅机制的缓存管理服务。对于写入的海量小文件，每一个合并后的大文件都在redis中设定一个定时器，当定时结束时，通过redis的订阅机制，获取到这一超时事件，然后再通过Amazon AWS S3接口向对象存储服务发起删除请求。这一流程如下图所示。



通过部署缓存管理模块，用户不必关系磁盘、内存占用以及超时文件删除的问题，一切都是自动进行的。

此外，针对对象管理还需进行一些数据统计，展示存储系统的状态。比如热门IP、host，这一切可基于redis的发布订阅实现。待后续补充。