ipfs介绍与使用

# 介绍

首先IPFS是分布式文件系统，它允许多地多节点形成拓扑网络，然后再通过网络间进行点对点的相互传递，而不需要经过中心节点，这个是基于ipfs的libp2p模块完成的，但我们不能保证每个节点间总是能相互访问（对称NAT限制），这时就需要一台两边都能访问的中间节点进行转发消息，IPFS也给出了简单的配置就可以完成这一操作，



对于两台需要通信的节点需要将EnableAutoRelay设为true，而中间节点需要将EnableRelayHop设为true。

对于应用层而言它是自带CDN的，在我们添加或者获取文件的时候，也就是“哪里的服务器离我们本机近就从哪里的务器上下载”，这个实际是广播所有的节点，哪台服务器先给反馈了就从哪台进行传输，所以在本地测试时高性能服务器总是能先一步通信，尽管我在应用层代码中填的网关ip可能并不是这台服务器。

这次的发现是我在加密系统上测试的时候偶然注意到的，那时候我把加密写在了jes5000服务器ipfs的源代码里，并且jes5000与eostest05高性能服务器连成分布式存储网络，在我的前端浏览器页面中，我“绑定”了上传的节点为jes5000，



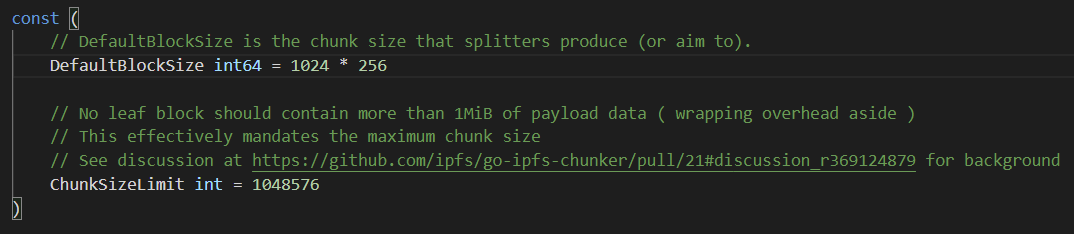
Swarm用来连接和上传，api网关用来下载文件

数据通过上传到了服务器上，如果是经过jes5000那么它应该是加密。可是当我直接在服务器上cat（打印）文件信息的时候却发现是明文的。于是我把eostest05断开，这下数据变成密文的了，下载我并没有试过，但是应该也是一样的道理。

基于这个特性可以发现，IPFS的高并发几乎是纯天然的。要是某台机器的负载增高那么其相应广播的效率自然也就不如别的机子了。

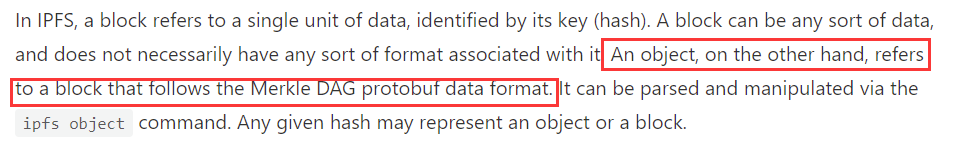
上传的文件有两种形式存在，他们分别是Object和Block，都基于UnixFS。

Block会将大文件进行分块，而Object则是一个完整的信息，默认的文件分块是256KB，在源代码中可以看到，

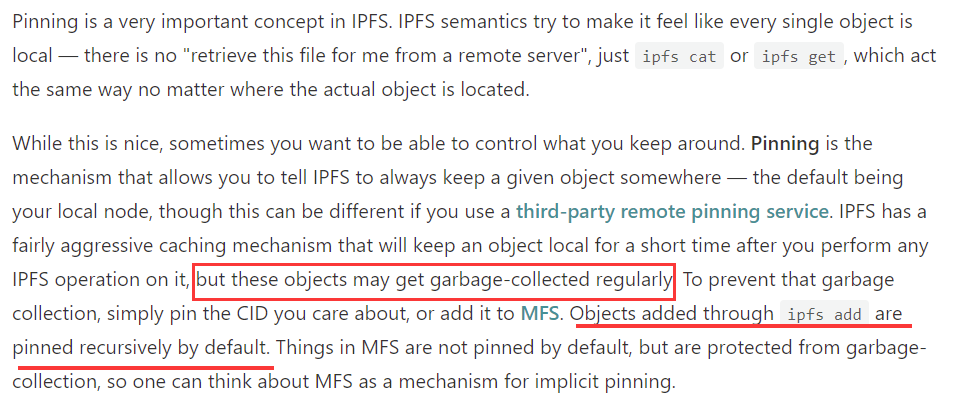


也可以看到可以指定所上传文件的分块大小，但是上限为1MB。可以通过ipfs ls <CID>来查看这个文件的分块数量，在这个基础上直接执行ipfs add <文件>，所有低于这个块大小的文件都会被视作为Object。

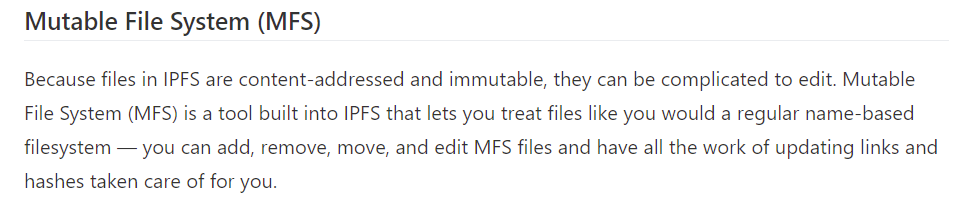
Block基于二进制数据块存储，而Object则基于MerkleDAG数据结构，



对于Object数据（不确定此刻的object是不是指的object数据）而言，他们会被视作为缓存，Ipfs自带的GC可能会将其清理。Pin指令可以解决这个问题，将其固定避开GC的清理，ipfs add已经自带pin了。



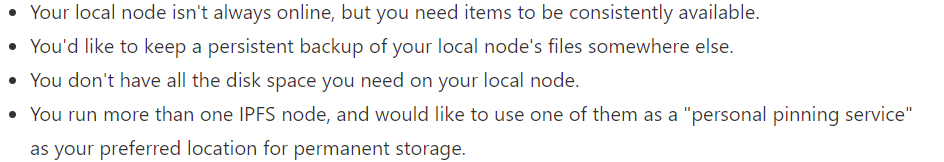
一些不是那么重要的数据可以使用ipfs pin rm使他们变得可以被GC，可以看到除了pin以外将文件添加到MFS也可以确保在GC之外。MFS除了Unixfs之外的另一种存储形式，它和我们认知的大多数文件系统相同，它允许文件的增删改查（Unixfs上是不允许删除的），以及被允许拥有更直白的文件名称。通过ipfs files指令可以完成对MFS的文件处理。



上述内容都是相对于本地而言的，对于ipfs添加的数据，其实都存储在本地.ipfs中，如果上传的数据未经过其他节点的访问，那么断开本地节点的连接时，其他节点将无法访问到文件，这个我们理解的大多数文件系统是一致的，即只存储自己的数据。

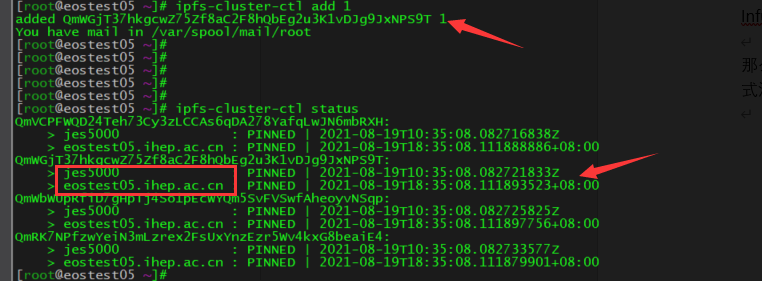
显然，如果每次都需要add都需要去别的服务器上get一下，那未免也太不方便了，为了解决这个文件同步的问题，ipfs推出集群管理项目ipfs-cluster。

除了ipfs-cluster之外，ipfs还提供了另一种方式，这个场景常被作用于商业化，ipfs pin remote。关于remote pin主要解决了以下的几个问题：



它释放了本地存储的压力，这需要你提供一个更具有存储价值的节点负责永久固定（pin）这些文件数据，而该节点需要提供一个pinning service，有诸多如 Pinata, Temporal, Crust, Infura都为用户提供了远程固定服务，当然也可以自己实现这个服务。

那么ipfs-cluster是怎么做的呢，它将整个ipfs集群视作个体，我们不再使用ipfs add的方式添加文件，而是使用ipfs-cluster提供的操作ipfs-cluster-ctl add，这时候ipfs-cluster就会帮我们自动在多节点上固定



Eostest05上传该文件

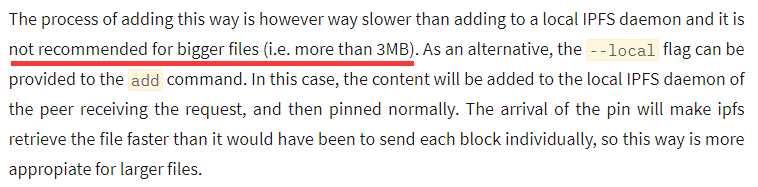


Jes5000日志

观察上图中的日志，可以发现每一次同步文件都会在本地先执行一边GC，除此之外ipfs-cluster每间隔15分钟也会执行一次，这个在具体的配置项中可以修改

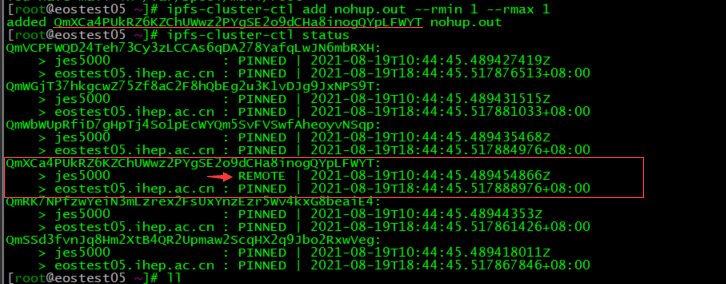


显然，在节点足够多的时候这样做是十分耗时的，对大文件的处理并不友好，官方也仅建议处理小于3M大小的文件才这样做



对于真正的大文件需要加上—local指令，但是这样做就有可能导致它虽然确保了不会被GC清理，可一旦local所在的宿主机下线这个文件也会从ipfs网络中消失，这是不可避免的。

我们可以指定固定的节点数量，通过—rmin和—rmax



可以看到在显示状态的时候jes5000不在是PINNED

# 使用

ipfs在网络通讯中的传输是加密且安全的，但是对于整个开放式的ipfs拓扑而言，ipfs的文件内容始终是明文的，ipfs并不负责对文件内容的加密与解密。

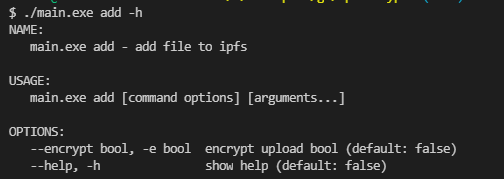
为了确保内容的保密性，我们将ipfs组成私网，再基于客户端本地的加解密，提供一个管理密钥与元数据的服务器来对文件内容进一步处理。

系统提供了尽可能简单的操作模式给予使用者，对ipfs的add与get进行一次重新封装，封装的主要意义是保证在上传或下载时对文件的加解密。

以下是Go语言实现的命令行客户端加解密的过程演示：



通过add对文件进行上传，如果加上-e可以加密文件后上传，



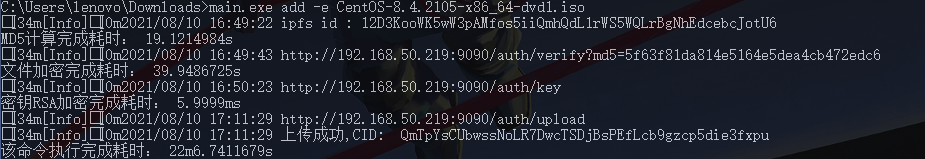
下方我们加密上传了一个10G的文件，通过打印的内容可以看到整个过程的耗时情况：

MD5校验 19s

AES 加密文件39s

RSA 加密密钥5ms

上传总用时 22m6s

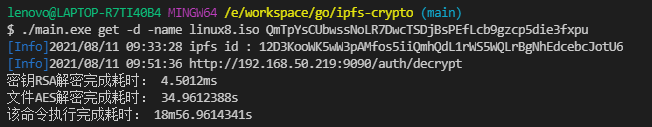


下载并解密使用get -d，同样，把刚刚那个文件下载下来。耗时：

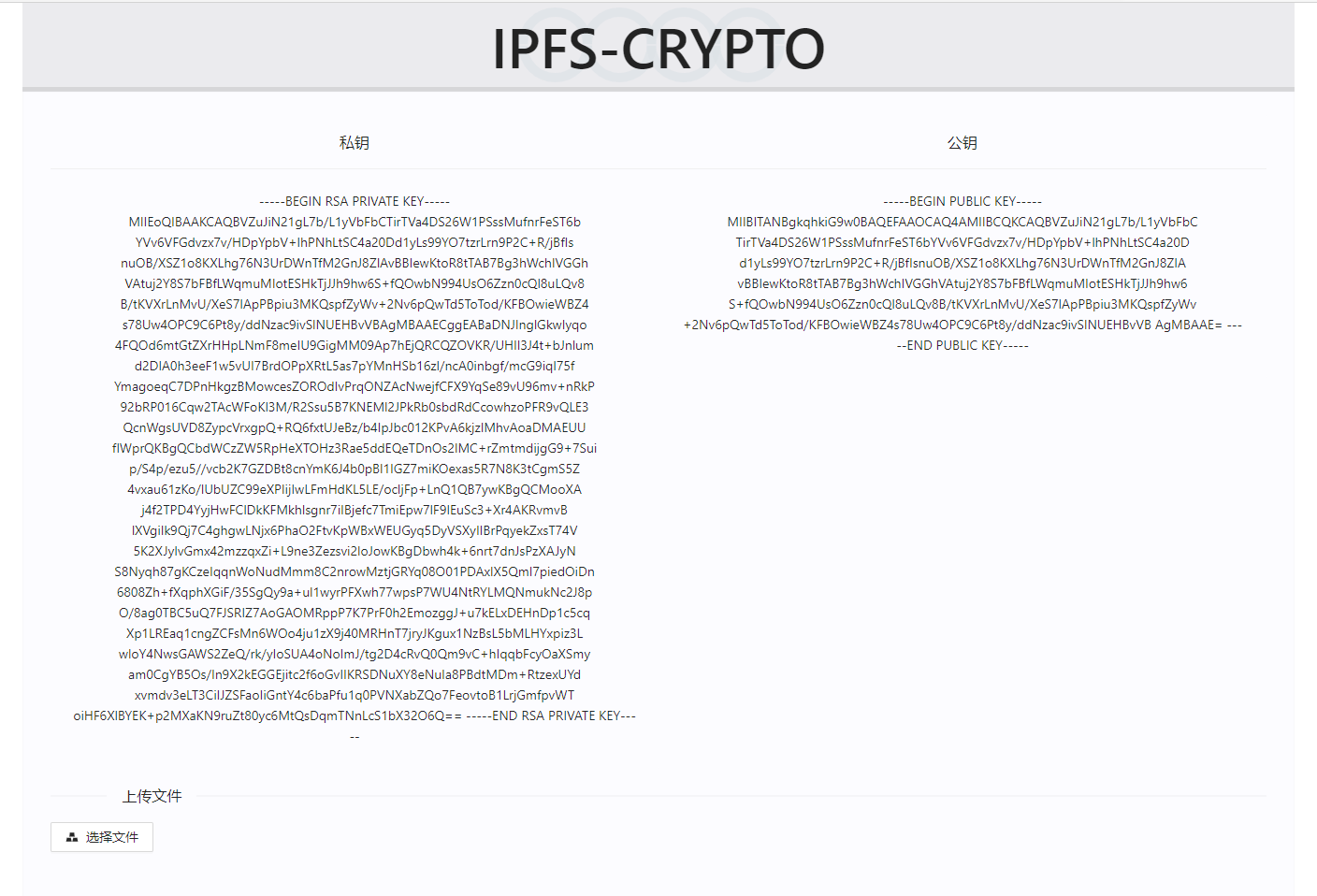
RSA密钥解密 4.5ms

AES文件解密 35s

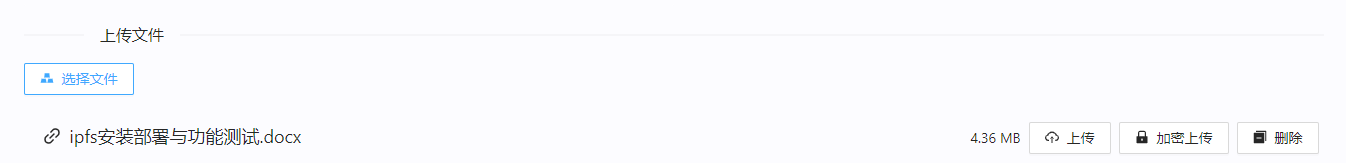
下载总用时 18.5m



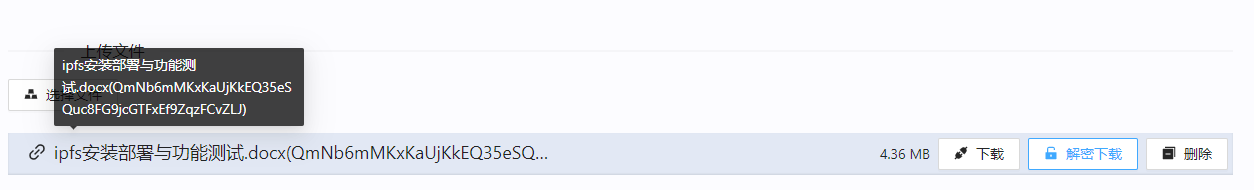
下面是JS提供的浏览器客户端演示，由于JS脚本语言本身性能欠佳，此处不做时间的计算



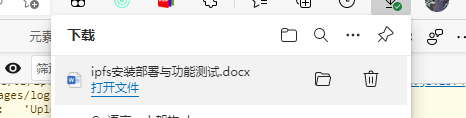
首先生成两个本地的公私钥，点击下方的上传文件



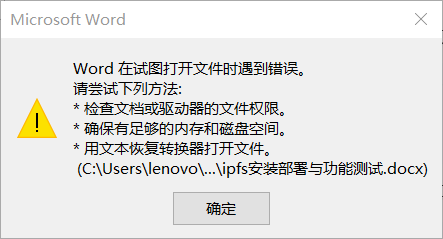
可以选择加密或者不加密上传，这里我们点击加密上传



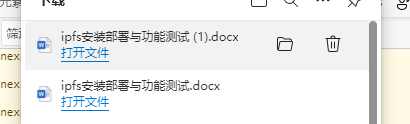
上传后文件可以选择下载，或者解密下载，为了演示需求，我们直接下载加密的文件



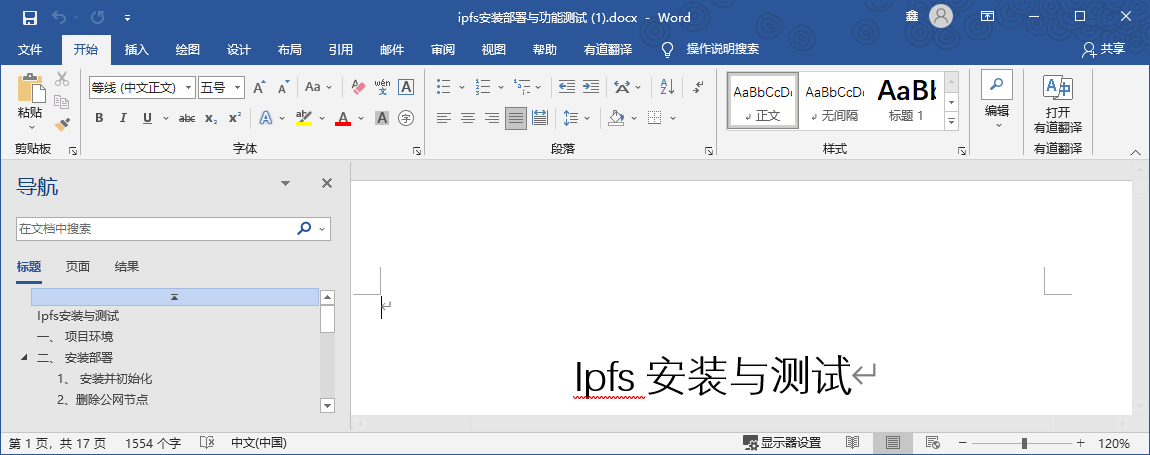
打开文件



错误了，因为未解密的文件二进制是混乱的，无法正常访问，我们再点击解密下载



打开文件



解密下载的文件可以正常访问。

当然，上述环境都是基于内网实现的，以及根据个人计算机的不同，届时使用加解密的耗时可能会有一些偏差。

查看具体系统的设计介绍请转移到《文件加密存储系统设计文档》

查看不同大小的文件测试请转移到《文件加密测试》

# 总结

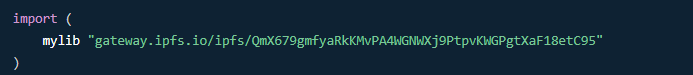
通过上述内容对ipfs体系已经有了较清晰的认识，但是我们再对其做一次总结，根据它的各个特性提供不同的服务

私有分布式文件存储：ipfs自带CDN加速，文件存储永不丢失，节点新增删除方便

文件隐私：提供客户端与服务端来对文件内容加解密，确保了文件内容的隐私性

对外文件存储管理：提供pinning service服务，就像 Pinata, Temporal, Crust, Infura,

Git站点：对于整个项目代码存在IPFS中就可以进行有效的版本管理，同时也确保了一些包不会因为依赖包的版本迭代而导致失效，像这样



内容共识：ipfs-cluster集群统一所有节点信息，使某一内容具有不可抵赖性