如何使用星际文件传输网络（IPFS）搭建区块链服务

IPFS（InterPlanetary File System）是一个点对点的分布式超媒体分发协议，它整合了过去几年最好的分布式系统思路，为所有人提供全球统一的可寻址空间，包括Git、自证明文件系统SFS、BitTorrent和DHT，同时也被认为是最有可能取代HTTP的新一代互联网协议。

IPFS用基于内容的寻址替代传统的基于域名的寻址，用户不需要关心服务器的位置，不用考虑文件存储的名字和路径。我们将一个文件放到IPFS节点中，将会得到基于其内容计算出的唯一加密哈希值。哈希值直接反映文件的内容，哪怕只修改1比特，哈希值也会完全不同。当IPFS被请求一个文件哈希时，它会使用一个分布式哈希表找到文件所在的节点，取回文件并验证文件数据。

IPFS是通用目的的基础架构，基本没有存储上的限制。大文件会被切分成小的分块，下载的时候可以从多个服务器同时获取。IPFS的网络是不固定的、细粒度的、分布式的网络，可以很好的适应内容分发网络的要求。这样的设计可以很好的共享各类数据，包括图像、视频流、分布式数据库、整个操作系统、模块链、8英寸软盘的备份，还有静态网站。

IPFS提供了一个友好的WEB访问接口，用户可以通过本机的 IPFS-HTTP 网关 `http://localhost:5001/ipfs/` 或者 公共的网关 `http://ipfs.io/` 获取IPFS网络中的内容，也可以通过特定的浏览器或者插件通过 `ipfs:/`

or `fs:/`

的方式直接获取内容。也许在不久的将来，IPFS协议将会彻底替代传统的HTTP协议。

一、使用IPFS

1.1 安装

- $ `go get -u -d github.com/ipfs/go-ipfs`
- $ `cd GOPATH/src/github.com/ipfs/go-ipfs`
- $ `make install`

1.2 初始化

- $ `ipfs init`

1.3 加入IPFS网络

- $ `ipfs daemon`

1.4 获取内容

- $ `ipfs cat /ipfs/QmYwAPJzv5CZsnA625s3Xf2nemtYgPpHdWEz79ojWnPbdG/readme`
- `http://localhost:5001/ipfs/QmYwAPJzv5CZsnA625s3Xf2nemtYgPpHdWEz79ojWnPbdG`

1.5 发布内容

- $ `ipfs add hello.jpg`
- IPFS文件还可以抽象成特殊的IPFS目录，从而标注一个可读的文件名（透明的映射到IPFS哈希），在访问的时候会像HTTP一样获取一个目录索引。在IPFS上建立网站的流程和过去一样，而且把网站加入到IPFS节点的指令只需要一条指令：`ipfs add -r yoursitedirectory`。

1.6 缓存内容到本地

- $ `ipfs pin add -r QmcKi2ae3uGb1kBg1yBpsuwoVqfmcByNdMiZ2pukxyLWD8`
- 缓存到本地的内容不仅可以自己使用，还能为其他节点提供资源

1.7 IPNS域名访问

- IPFS哈希只能用来表示不可变数据，因为一旦数据改变，哈希值也会改变。从某种意义上来说，这是保持数据持续性的好的设计。但是我们也需要一种方法来标记最新更新网站的哈希，这个方法我们称作IPNS。
- IPNS的原理是从域名的TXT记录里获取IPFS哈希地址，然后根据这个哈希地址从IPFS网络中获取数据。 比如 http://ipfs.io/ipns/ipfs.git.sexy
- 接下来IPFS还打算支持Namecoin。Namecoin从理论上完全实现了分布式Web的去中心化，整体的运行中不再需要中心化的授权。支持了Namecoin的IPFS不再需要ICANN、中心服务器，不受政治干涉，也无需授权证书。

1.8 更多

- 更多信息请浏览IPFS的API文档https://ipfs.io/docs/api/
- 如果你想基于IPFS做一些开发，这里有各种语言实现的API调用接口https://github.com/ipfs/ipfs#api-client-libraries

二、IPFS是如何工作的

2.1 身份验证

和比特币相似，每一个节点都会由NodeId(公钥的哈希值)来标识，节点存储着公钥和加密过的私钥。
首次连接时，节点间交换公钥，并检查 hash（other.PublicKey）是否等于other.NodeId。如果没有，则终止连接。

```
type NodeId Multihash

type Multihash []byte

// self-describing cryptographic hash digest


type PublicKey []byte

type PrivateKey []byte

// self-describing keys


type Node struct {

    NodeId NodeID

    PubKey PublicKey

    PriKey PrivateKey

}
```

2.2 网络

每个节点与网络中的相连的其他数百个节点进行定期通信。
IPFS的网络传输具有如下特性：

- 传输： IPFS可以使用任何传输协议，如 WebRTC 和 uTP。
- 可靠性：如果底层网络不能保证可靠性，IPFS可以使用 uTP 或 SCTP 来保证

- 连接：IPFS还使用 ICE NAT 穿越技术。
- 完整性：使用哈希校验和检查消息的完整性。
- 真实性：可以使用发送者的公钥和HMAC来检查消息的真实性。

同时IPFS不仅仅是通过IP来连接节点，还支持很多其他协议。IPFS内部使用不同的地址格式来选择不同的网络协议。

```
# an SCTP/IPv4 connection

/ip4/10.20.30.40/sctp/1234/



# an SCTP/IPv4 connection proxied over TCP/IPv4

/ip4/5.6.7.8/tcp/5678/ip4/1.2.3.4/sctp/1234/
```

2.3 路由

IPFS通过通过基于 S/Kademlia 和 Coral 的 DSHT 来寻找匹配的节点和特定节点的地址信息，IPFS的对象和使用模式的大小类似于 Coral 和 Mainline，因此 IPFS DHT 根据其大小对存储的值进行区分。 小值（等于或小于1KB）直接存储在DHT上。对于更大的值，DHT存储拥有这些块的节点NodeId。
DSHT的接口定义如下：

```
type IPFSRouting interface {

    FindPeer(node NodeId)

    // 得到指定节点的地址



    SetValue(key []bytes, value []bytes)

    // 小值可直接存储在DHT上



    GetValue(key []bytes)

    // 从DHT中获取值
```

```
    ProvideValue(key Multihash)
    // 宣布此节点可以提供一个大值

    FindValuePeers(key Multihash, min int)
    // 得到拥有特定大值的所以节点
}
```

2.4 块交换

在IPFS中，通过使用 BitSwap 协议与其他节点进行块(block)交换来实现数据分发。
BitSwap 维持着两个列表，想要获得的块和已保存的块。但与 BitTorrent 不同的是，BitSwap 不限于一个torrent中的块。BitSwap 节点可以从整个IPFS网络获取所需的块，而不管这些块属于哪些文件，这大大提高了下载效率。
同时，网络中存在一些激励节点会主动缓存和传播稀有的文件片段。

2.4.1 信用体系

我们希望所有的节点都乐于分享他们拥有的块，但某些自私节点只从P2P网络中获取块，而从不做种。
IPFS使用了一套简单的信用系统来解决这个问题。

- 从其他节点获取块会产生“债务”，向其他节点发送块可以偿还“债务”。
- 每个节点都记录与相连节点间的“债务”情况。
- 欠债越多的节点其优先级越低，如果一个节点只获取而从不奉献将会很快被其他节点进行忽略超时操作。

2.4.2 策略

BitSwap 采用的不同策略对整体的演变表现有着非常不同的影响。
在 BitTorrent 中，虽然规定了标准策略，但是也已经实现了许多其他方法，从 BitTyrant (尽可能分享)到BitThief（利用漏洞 并且永远不会分享），到 PropShare (按比例分享)。
我们需要的策略的目标应该是：

1. 最大化节点的交易性能和整体交换效率
2. 防止“吃白食”的情况发生

3. 有效抵抗其他未知策略

4. 对受信任的节点限制宽松

一种在实践中可行的策略是一个跟债务率挂钩的算法
节点的负债率 `r = bytes_sent / (bytes_recv + 1)`
发送率 `P (send|r) = 1− 1/(1+exp(6−3r))`
当节点的负债率超过已建立信用额度的两倍时，发送率迅速降低。

2.4.3 账单

BitSwap 节点维持与其他节点的传输计费账单，当节点间建立连接时，双方交换账单，如果账单不匹配，则清除已有账单，重新开始记账。当然，恶意节点可能会故意丢失账单，希望清除债务，其他节点可以将其视作不当行为，并拒绝。
账单的数据结构如下：

```
type Ledger struct {

    owner      NodeId

    partner    NodeId

    bytes_sent int

    bytes_recv int

    timestamp  Timestamp

}
```

2.4.4 接口规范

```
// Additional state kept

type BitSwap struct {

  ledgers map[NodeId]Ledger

  // Ledgers known to this node, inc inactive



  active map[NodeId]Peer
```

```
  // currently open connections to other nodes

  need_list []Multihash
  // checksums of blocks this node needs

  have_list []Multihash
  // checksums of blocks this node has
}

type Peer struct {
  nodeid NodeId
  ledger Ledger
  // Ledger between the node and this peer

  last_seen Timestamp
  // timestamp of last received message

  want_list []Multihash
  // checksums of all blocks wanted by peer
  // includes blocks wanted by peer's peers
}

// Protocol interface:
interface Peer {
```

```
  open (nodeid :NodeId, ledger :Ledger);

  send_want_list (want_list :WantList);

  send_block (block :Block) -> (complete :Bool);

  close (final :Bool);

}
```

2.5 Merkle DAG

DHT 和 BitSwap 技术让 IPFS形成一个用于快速而强大的存储和分发块的 P2P 系统，。
在此之上，IPFS还构建了一种有向无环图 Merkle DAG，使用嵌入数据源中的目标哈希散列构建对象之间的链接。Merkle DAGs 为IPFS提供了许多有用的属性，包括：

1. 内容寻址：所有内容（包括链接）都由其多哈希校验和进行唯一标识。
2. 防篡改：所有内容都使用其校验和进行验证。如果数据被篡改或损坏，则IPFS会检测到该数据。
3. 去冗余：所有内容完全相同的对象，只存储一次。这对索引对象特别有用，比如git tree和commits，或者是公共部分的数据。

IPFS对象的定义如下：

```
type IPFSLink struct {

    Name string
    // name or alias of this link

    Hash Multihash

    // cryptographic hash of target

    Size int

    // total size of target

}



type IPFSObject struct {
```

```
  links []IPFSLink

  // array of links

  data []byte

  // opaque content data

}
```

Merkle DAG 是一种非常灵活的数据存储方式，唯一的要求是 a) 使用内容寻址 b) 使用上述编码格式。

这使得我们可以用路径的方式访问对象，`/ipfs//`，
比如 `/ipfs/XLYkgq61DYaQ8NhkcqyU7rLcnSa7dSHQ16x/test/foo.txt`

2.6 文件系统

在 Merkle DAG 之上，IPFS还定义了一组对象用于对版本化文件系统进行建模。
这个对象模型类似于Git：

1. block：可变大小的数据块。
2. list：块或其他列表的集合。
3. tree：块、列表或其他树的集合。
4. commit：树的版本历史中的快照。

2.6.1 blob

blob 对象包含了可寻址的数据单元，表示一个文件。

```
{

"data": "some data here",

// blobs 是没有 link 的

}
```

一个IPFS文件由 blobs 和 lists 构成

2.6.2 list

list 对象将很多去重的 blobs 连接到一起，包含了一组有序的 blob 或 list 对象。


```
{
    "data": [
        "blob",
        "list",
        "blob"
    ],
    "links": [
        {
            "hash": "XLYkgq61DYaQ8NhkcqyU7rLcnSa7dSHQ16x",
            "size": 189458
        },
        {
            "hash": "XLHBNmRQ5sJJrdMPuu48pzeyTtRo39tNDR5",
            "size": 19441
        },
        {
            "hash": "XLWVQDqxo9Km9zLyquoC9gAP8CL1gWnHZ7z",
            "size": 5286
        }
    ]
}
```


2.6.3 tree

tree 对象代表一个路径，内容包括 blob、list、tree、commit，同时标记了对象的名称。


```
{
    "data": [
        "blob",
        "list",
        "blob"
    ],
    "links": [
        {
            "hash": "XLYkgq61DYaQ8NhkcqyU7rLcnSa7dSHQ16x",
            "name": "less",
            "size": 189458
        },
        {
            "hash": "XLHBNmRQ5sJJrdMPuu48pzeyTtRo39tNDR5",
            "name": "script",
```

```
            "size": 19441
        },
        {
            "hash": "XLWVQDqxo9Km9zLyquoC9gAP8CL1gWnHZ7z",
            "name": "template",
            "size": 5286
        }
    ]
}
```

2.6.4 commit

commit 对象代表对象的历史快照。

```
{
    "data": {
        "type": "tree",
        "date": "2014-09-20 12:44:06Z",
        "message": "This is a commit message."
    },
    "links": [
        {
            "hash": "XLa1qMBKiSEEDhojb9FFZ4tEvLf7FEQdhdU",
            "name": "parent",
            "size": 25309
        },
        {
            "hash": "XLGw74KAy9junbh28x7ccWov9inu1Vo7pnX",
            "name": "object",
            "size": 5198
        },
        {
            "hash": "XLF2ipQ4jD3UdeX5xp1KBgeHRhemUtaA8Vm",
            "name": "author",
            "size": 109
        }
    ]
}
```

2.7 命名和可变状态

到目前为止，IPFS堆栈形成了构建内容寻址对象 DAG 的P2P交换。它可以用于发布和检索不可变的对象，甚至可以跟踪这些对象的版本历史。但是，仍缺少一个关键组件：可变命名。没有它，用户就得在IPFS系统外获取到新的内容地址了。

2.7.1 自验证命名

1. 定义节点的NodeId为该节点公钥的哈希
2. 通过 /ipns/的方式可以访问该节点下的内容
3. 当其他节点从该节点获取文件时，可以验证其公钥和NodeId是否匹配

通过自验证命名，我们可以实现这样的访问效果
`/ipns//docs/test.md` 而不必用 `/ipfs/` 这样的方式

2.7.2 更加友好的命名方式

自验证命名虽然解决了一些问题，但对用户来说还不够友好， IPFS提供了如下解决方案：

1. 节点链接
   通过执行 `ipfs link //friends/bob /`
   便可将 bob 节点链接到 alice 节点的 friends/bob 路径下，这样只需要知道alice的地址就可以访问bob了

2. 域名访问
   IPNS可以从域名的TXT记录里获取IPFS哈希地址，然后根据这个哈希地址从IPFS网络中获取数据
   例如我们设置 ipfs.benet.ai 的TXT记录为"ipfs=XLF2ipQ4jD3U ...",
   访问 /ipns/ipfs.benet.ai 便相当于 ipns/XLF2ipQ4jD3U ...

3. Proquint可发音方案
   IPNS支持将哈希地址译成可发音的单词
   例如 /ipns/dahih-dolij-sozuk-vosah-luvar-fuluh
   将解析为 /ipns/KhAwNprxYVxKqpDZ

4. 短地址服务
   以shorten.er为例，用户可以获得一个指向特定地址的链接
   例如 `/ipns/shorten.er/foobar`
   将解析为 `/ipns/XLF2ipQ4jD3UdeX5xp1KBgeHRhemUtaA8Vm`
   这时shorten.er就类似于我们今天使用的DNS服务器了。

3. 基于IPFS的应用

`ipfs.pics` - 免费的永久图床，上传和分享你的图片。
`Orbit Chat` - 基于IPFS的去中心化聊天室，用户可以自由创建和加入`channel`并参与讨论。

`Neocities` - 免费帮助人们创作和发布网页的组织，将用户上传的网页永久存储在IPFS网络中，即使Neocities关闭了，人们仍然可以在IPFS网络中浏览到这些创作。
`AKASHA` - 基于IPFS和以太坊的下一代社交博客平台。
`git-ipfs-rehost` - 将你的git仓库托管在IPFS网络上。
`Global Upload` - 文件传输服务
`IPFS SEARCH` - 搜索IPFS网络中的内容

4. IPFS与区块链技术的结合

IPFS弥补了现有区块链系统在文件存储方面的短板，将IPFS的永久文件存储和区块链的不可篡改、时间戳证明特性结合，非常适合应用于 保护版权、身份证明、来源证明等方面。
同时用基于区块链的代币来激励IPFS节点存储数据也是最好的选择。
在去中心化的世界里，QTUM智能合约提供各种逻辑服务，IPFS提供文件资源，两者结合，共同构建去中心化的网络世界。

5. 打赏地址

比特币：1FcbBw62FHBJKTiLGNoguSwkBdVnJQ9NUn
以太坊：0xF055775eBD516e7419ae486C1d50C682d4170645

6. 技术交流

- 区块链技术交流QQ群：348924182
- 「区块链部落」官方公众号

长按，识别二维码，加关注