CHISE の IPFS 化における諸課題について

守岡 知彦 （国文学研究資料館）

1 はじめに

CHISE の全機能を IPFS ベースで再実装する際の課題について議論する。

CHISE project の前身の UTF-2000 project は 1998 年に始まったが、現在の CHISE の中核となる XEmacs CHISE の前身である XEmacs UTF-2000 は 1999 年に開発が始まり、2001 年に現在の名称に変更した後、2000 年代前半にはそのエンジン部分の実装はほぼ完了し、その後の開発は基本的に Emacs Lisp 等のアプリケーション層と文字データの拡充にとどまっている。XEmacs CHISE は、Lisp 言語の柔軟性を活かして、文字オントロジーを中核とした文字処理技術のテストベッドとして文字間の関係記述、楷書の歴史的字体用例や古漢字データの収録、文字と形態素や古辞書データ、その他用例データ等との対応づけ等、さまざまな実験を展開することができ、階層的素性名方式や多粒度漢字構造モデルをはじめとする漢字記述のための手法やモデルを生み出す母体ともなったが、開発から 20 年以上が経過しており、現在の計算機環境に合わない部分も少なくない。そのため、数年前から CHISE システムの再実装・近代化に関する検討や実験を行っている。

現在、多くのユーザーは Web サービスを介して CHISE を利用していると考えられるため、Web との親和性は重要であるが、その一方で、現在の CHISE 関連 Web サービスは XEmacs CHISE が持つ CHISE 関連機能のごく一部のみを使っているに過ぎない。特に文字オントロジーにアクセスするためのプログラミングを行う上で現在の CHISE の Web サービスは不十分であり、また、文字オントロジーの複雑さから Lisp 以外の言語でデータセットを簡便に利用するための環境の整備もあまり進んでいない。

サービスの永続化という観点で考えた場合、Web サービスはサーバー環境やドメイン名の維持にコストがかかり、その長期維持が容易でないという問題もある。Web は URL という識別子によって情報にアクセスする仕組みであるが、URL は、通常、サーバー（ホスト）の名前とサーバー内での場所（URL パス）に基づいて構成されるものであり、情報の内容ではなく情報を置いた場所に付けられた識別子である。紙の本との対比でいえば、URL は本（内容）に付けられた ID ではなく本棚の場所に付けられた ID といえる。本棚に置く本を入れ替えることができるのと同様に URL の中身も変更可能である。また、コンテンツに紐付けられた URL が変わってしまうとコンテンツの中身が何も変わらずそのまま存在しているにもかかわらず、今までの URL ではアクセスできなくなってしまい実質的な情報の消失を招く。前述のたとえでいえば、本を置く棚を変えた

り、本棚や建物自体が移動したり、あるいは図書館が再編されたりして本を置く場所が変化すると見つけ出せなくなる訳である。一方、Web には紙の本やパッケージ化されたコンテンツと同様の弱点も存在する。それは当初予期していた以上の大量のアクセスが急に生じるとサーバーがさばききれなくなって情報にアクセスできなくなってしまうことである。紙の本であれば増刷をかければタイムラグはあるもののいずれ読めるようになるだろう。同様に Web もサーバーを増強したり CDN を使うなどして地域毎に分散されたエッジサーバーを用意してやれば問題は解決する。しかし紙の本やパッケージ化されたコンテンツならばこの問題はコンテンツが記録されたメディアを入手するまでの問題であって手元に持っているなら影響は受けないはずである。Web においてもローカルキャッシュは存在するが、通常、ユーザーはそのキャッシュを意識せず明示的に制御しない。よって、ある時、急に、それまで見えていたサイトの情報が急に見えなくなるという危険性は常に存在する。特に、災害時など切実な時に限って。

近年、話題になっている分権化 (decentralized) された Web[*1]はこうした問題の解決を目指して開発されているものである。本稿で扱う IPFS もそのための要素技術を提供するものである。

IPFS は従来型 Web の欠点のいくつかを解決している反面、分権化アーキテクチャーや P2P ベースのファイルシステムであることに起因する問題も抱えており、従来型のサービスを単純に置き換えることはできず、IPFS の性質に合わせた設計を行う必要があると考えられる。本稿では、CHISE 文字オントロジーを IPFS 上に載せ JavaScript を用いて現実的な Web アプリケーションで利用可能なサーバーレス実装を実現する上での要件について議論する。

2 IPFS の概要

IPFS (InterPlanetary File System) [4] [8] は Protocol Labs 社が中心となって開発しているオープンソースの P2P 型分散ファイルシステムである。

通常の Web では URL (IRI) という資源の場所に基づくアドレッシング (location-addressing; 『場所アドレッシング』) になっており、識別子の指す先が変化する可能性があり、また、オブジェクトに変化がなくてもその識別子が変化してしまうと参照できないという性質を持っている。このことは情報資源の長期保存においては良い性質とはいえず、このため DOI のような永続的識別子の利用が注目されている。

これに対し、IPFS ではオブジェクトのハッシュ値に基づいて内容 ID (CID; Content IDentifier) を生成する仕組みがプロトコルによって定義されており、いつどこで誰が行っても対象とするオブジェクトが同一のバイト列を持っていれば同じ CID が生成されるため、DOI や DNS のように機関を認証して権限を委譲する必要はなく、結果的に、非常に小さな粒度のデータから非常に大きなデータセットの集合といったスケーラビリティーで永続的な識別子を生成することができる。

[*1] これはしばしば Web3 などと呼ばれる。

2.1 IPFS のプロトコルスタックと実装

IPFS は表 1 に示す 4 層で構成されており、下位の 3 層及び IPFS が提供する幾つかのアプリケーション[*2]に関して、JavaScript および TypeScript と Go 言語による実装が提供されている。

アプリケーション（IPFS, その他）
IPNS　名前解決
IPLD　Merkle DAG によるデータモデル
libp2p　ネットワークやルーチング、データ転送

表 1　IPFS のプロトコル階層モデル

表 1 のアプリケーション層に位置するファイルシステムとしての IPFS（ここでは『狭義の IPFS』と呼ぶことにする）は P2P 型分散ファイルシステムであり、

1. コマンドラインインターフェイス
2. Web インターフェース
3. ファイルシステムへのマウント

という 3 つの方法でアクセスすることができる。

1 つ目の方法は Go 言語による実装 (Kubo; 旧 go-ipfs) が提供するコマンドラインインターフェイス (CLI) であり、Linux や Mac, Windows 上のターミナル環境や Linux 環境、各種 Unix 系 OS などで POSIX シェルなどを介して Kubo の ipfs コマンドを利用する方法である。また、JavaScript による実装 (js-IPFS) も jsipfs という Kubo の ipfs コマンドと同様の CLI を提供している。

2 つ目の方法は Kubo が提供する Web API, あるいは、js-IPFS やその後継として開発中の TypeScript による実装である Helia が提供する API [*3]やアプリケーション、IPFS に対応した Web ブラウザーや拡張モジュール、あるいは、IPFS パブリックゲートウェイサービスなどを利用して Web 上で利用する方法である。標準の Web UI として **IPFS Web UI** [3] が用意されている（図 1）。

3 つ目の方法は FUSE (Filesystem in Userspace) を利用してファイルシステムにマウントする方法である。[*4]

Kubo の場合、コマンドラインインターフェースと Web API で等価な機能が提供されており、狭義の IPFS に限らず、IPLD や libp2p などの下位層の機能も利用可能である。js-IPFS や Helia に関しても同様である。

IPFS Web UI は IPFS ノードの各種情報（ノードの ID (Peer ID), 実装の名前とバージョン、

[*2] 分散ファイルシステムとしての IPFS もこのアプリケーション層に位置するものと看做すことができる。
[*3] js-IPFS や Helia の API は Node.js 等を用いて Web ブラウザー外でも実行可能である。
[*4] ただ、現時点では書き込みに関しては遅く、まだ実験的なものと考えた方が良いかも知れない。

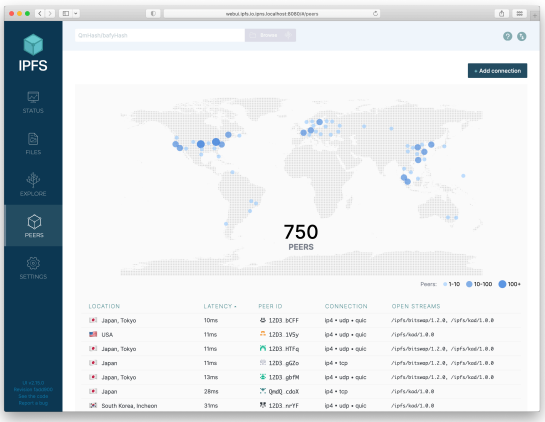


図 1　IPFS Web UI

UI のバージョン、転送速度等）の閲覧や各種設定、ファイルやフォルダーのアップロードやダウンロード、起点となる CID の IPLD の情報の閲覧したり IPLD のリンクをたどる機能などが用意されている。

IPFS Desktop [2] と呼ばれる Kubo と IPFS Web UI を一つのパッケージにまとめたものも用意されており、Mac, Windows, Linux, FreeBSD 用のバイナリーパッケージも用意されている。また、macOS で Homebrew を利用している場合、

```
% brew install --cask ipfs
```

でインストールすることも可能である。

IPFS や IPNS のための URI スキーマとして、`ipfs:`, `ipns:` が登録されており、`ipfs://`*CIDv1b32*[*5] や `ipfs://ipfs/`*CID* や `ipns://`*PeerID*, `ipfs://ipns/`*PeerID* といった URI を記述することができる。現在、Firefox と Chrome, Brave, Opera, Edge では **IPFS Companion** [1] というアドオンソフトウェアが利用可能であり、各々のアドオンストア等から入手可能である。

2.2　IPFS へのファイルの追加

Linux 環境（あるいは、UNIX 系 OS, Mac, Windows 上の Linux）などのターミナル上で bash や zsh 等の POSIX シェルを使って ipfs コマンドを実行する場合、

```
% ipfs add ファイル名
```

とすれば、ファイル名の内容が IPFS 上に追加される。

[*5] CIDv1 で Base32 でテキスト化した CID. `QmVmSzHuGK1VTBaUgYfUFZFWbefQ3jrLDkgndpzgntzjSQ` のような CIDv0 の Base58BTC でテキスト化した CID は使えない。

例えば、

```
% echo 'This is a test.'  > test.txt
```

として test.txt というファイルを作り

```
% ipfs add test.txt
```

を実行すれば

```
added QmXnVqt4nuU1W1XuBbYVsoj4a16NCdp1Yw6n35JZBiaaic test.txt
```

という結果を得る。ここで、QmXnVqt4nuU1W1XuBbYVsoj4a16NCdp1Yw6n35JZBiaaic はこのファイルの CID である。この CID を使って、

```
% ipfs cat QmXnVqt4nuU1W1XuBbYVsoj4a16NCdp1Yw6n35JZBiaaic
```

を実行すると

```
This is a test.
```

という結果を得る。

なお、ipfs コマンドは

```
% echo 'This is a test.'  | ipfs add
```

のように標準入力からデータを追加することも可能である。この場合も、ファイル名を指定した場合と同じ QmXnVqt4nuU1W1XuBbYVsoj4a16NCdp1Yw6n35JZBiaaic が得られる。ファイル名を変更したとしてもデータの内容が同じであれば同じ CID が得られる。

```
% ipfs add -r ディレクトリ名
```

とすれば指定したディレクトリの内容を一括して追加することができる。

例えば、

```
% mkdir testdir
% echo 'This is a test.'  > testdir/test1.txt
% echo ' これはテストです。' > testdir/test2.txt
% ipfs add -r testdir
```

とすると

```
added QmXnVqt4nuU1W1XuBbYVsoj4a16NCdp1Yw6n35JZBiaaic testdir/test1.txt
added QmV5Q8oN5XxKkgkdNF4j2YuLWdr3piUkBANrFXYffNesj4 testdir/test2.txt
added QmdzCodhnmWtNMRRudDHFCtUhNAdWHxE5D2Vm6MNvXdtiu testdir
```

という結果を得る。[*6]ここで、QmdzCodhnmWtNMRRudDHFCtUhNAdWHxE5D2Vm6MNvXdtiu は testdir の CID で、その下に 2 つの CID が含まれている。

QmdzCodhnmWtNMRRudDHFCtUhNAdWHxE5D2Vm6MNvXdtiu はディレクトリの CID であるので、

```
% ipfs cat QmdzCodhnmWtNMRRudDHFCtUhNAdWHxE5D2Vm6MNvXdtiu
```

とすると

```
Error: this dag node is a directory
```

という風なエラーになる。[*7]

ディレクトリに含まれるファイルの情報を得たい場合、

```
% ipfs ls ディレクトリ名
```

とすれば良い。

例えば、

```
% ipfs ls QmdzCodhnmWtNMRRudDHFCtUhNAdWHxE5D2Vm6MNvXdtiu
```

とすれば

```
QmXnVqt4nuU1W1XuBbYVsoj4a16NCdp1Yw6n35JZBiaaic 16 test1.txt
QmV5Q8oN5XxKkgkdNF4j2YuLWdr3piUkBANrFXYffNesj4 28 test2.txt
```

という結果を得る。

ディレクトリ内のファイルの内容を見たい場合、

```
% ipfs cat QmV5Q8oN5XxKkgkdNF4j2YuLWdr3piUkBANrFXYffNesj4
```

のように個々のファイルの CID を指定しても良いが、

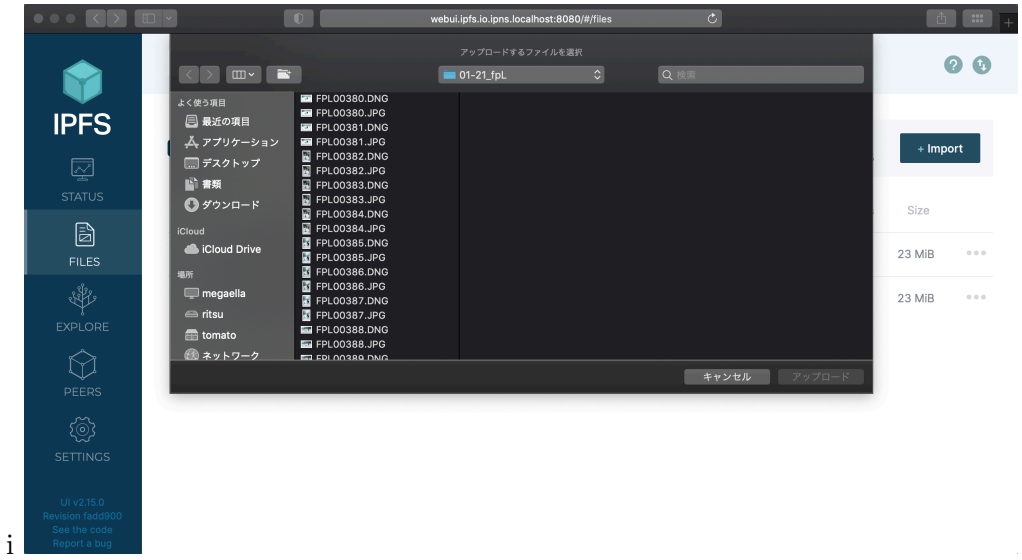
```
% ipfs cat QmdzCodhnmWtNMRRudDHFCtUhNAdWHxE5D2Vm6MNvXdtiu/test2.txt
```

のようにディレクトリの CID を起点にしたパス名で指定しても良い。

ファイルやディレクトリの追加は IPFS Desktop や IPFS Web UI でも実行可能である。IPFS Desktop もしくは Web ブラウザー上で IPFS Web UI を起動し、FILES タブを開き、その画面中の Import ボタンを押し、File メニューを選択し、アップロードするファイルを選択すれば選択したファイルが追加される（図 2)。同様に Folder を指定すれば指定したフォルダー（ディレクトリ）が追加される。

[*6] testdir/test1.txt の中身が上述の test.txt と同じで同じ CID になっている。

[*7] dag は次節で述べる IPLD のオブジェクトという意味である。



i

図 2　IPFS Web UI でアップロードするファイルの選択画面

2.3　IPLD

IPLD (Inter Planetary Linked Data) [9] [6] は IPFS のデータモデルを提供する層で、Merkle DAG と呼ばれる暗号学的ハッシュに基づく有向非巡回グラフ (Directed Acyclic Graph; DAG) と、Merkle paths と呼ばれる Merkle DAG を名前付きリンクをたどってトラバースしたものを UNIX 風の階層的ファイル名に対応させる仕組みと、IPLD の正規化されたデータ形式 (IPLD Canonical Format) 等の仕様により、JSON や XML 等の外部形式を IPLD の世界に格納したり IPLD の世界のオブジェクトを指定した外部形式で出力することができる。

IPLD は IPFS のデータモデルであるので、IPFS 上のさまざまなアプリケーションは全て IPLD の DAG として記述されたデータを持っている。また、CID は複数の版、複数のコーデック (multicodec)、複数のハッシュ関数 (multihash)、複数のバイナリーテキスト化方式 (multibase) をサポートするが、ここでは IPFS の Go 実装 (go-ipfs) の ipfs コマンドの ipfs dag {get|put} や IPFS HTTP API の /api/v0/dag/{get|put} で読み書きできる cidv1-cbor-sha2-256 形式、即ち、版は CIDv1 でコーデックとして CBOR, ハッシュ関数として SHA2-256 を用いるものとする。また、バイナリーのテキスト化方式として Base32 を用いる。

IPLD のオブジェクト（DAG のノードとなるもの）は JSON のオブジェクトと同様な key と value の対の集合である。例えば、

```
{ "name": "Paul Marie Ghislain Otlet" }
```

は

```
bafyreifgdqbtjikozqxadmayhl53sqscn5glca44lk6kb3brqm7wozd2si
```

という CID を持つ IPLD のオブジェクトの JSON 表現である。

Linux 環境（あるいは、UNIX 系 OS, Mac, Windows 上の Linux）などのターミナル上で bash や zsh 等の POSIX シェルを使って ipfs コマンドを実行する場合、

```
% echo '{ "name": "Paul Marie Ghislain Otlet" }' | ipfs dag put
```

とすれば `bafyreifgdqbtjikozqxadmayhl53sqscn5glca44lk6kb3brqm7wozd2si` という結果を得ることができる。

なお、CID は自己記述的な形式 (self-describing format) であり、版やコーデック、ハッシュ関数、テキスト化方式のそれぞれにどれを用いたかの情報が埋め込まれている。各形式は `ipfs cid base32` CID で cidv1-cbor-sha2-256 の Base32 に変換できる。また、例えば古い版で使われていた Base58BTC (Bitcoin の Base58) でテキスト化された CID `zdpuAwbrw9r5jy1gUoB61EoNLGRossy171TGKyB2AbAvmArmo` を使って

```
% ipfs dag get zdpuAwbrw9r5jy1gUoB61EoNLGRossy171TGKyB2AbAvmArmo
```

しても

```
% ipfs dag get bafyreifgdqbtjikozqxadmayhl53sqscn5glca44lk6kb3brqm7wozd2si
```

と同じ結果

```
{"name":"Paul Marie Ghislain Otlet"}
```

を得ることができる。

IPLD に JSON のオブジェクトを格納する場合、IPFS のようにバイト列そのものが格納される訳ではなく、正規化された CBOR (Concise Binary Object Representation) [5] 形式に変換されて格納される。この際、空白・改行の有無・個数や key-value 対の順番は捨象されるため、これらの差異に関らず同じオブジェクトとして扱われる。また、現状、go-ipfs の dag サブコマンドでは JSON 以外の形式がサポートされていないが、原理的には、XML や YAML, RDF といった形式であっても、それが同一の key-value 対の集合を表現しているならば同一のオブジェクトとして扱われることになっている。

IPLD では他のオブジェクトへのリンクは key が `"/"` で value が CID である link-object と呼ばれる特殊なオブジェクトで表現される。

例えば、

```
{ "/": "bafyreifgdqbtjikozqxadmayhl53sqscn5glca44lk6kb3brqm7wozd2si" }
```

は link-object の JSON 表現である。

また、

```
{
```

```
  "name":  "Henri La Fontaine",
  "collaborator":
    { "/": "bafyreifgdqbtjikozqxadmayhl53sqscn5glca44lk6kb3brqm7wozd2si" }
}
```

を JSON 表現として持つ IPLD オブジェクト

```
bafyreibjtnctwmcejbmangiu4ahoni26pggeci7chl3gieibwraxoblpqm
```

のようにある key の value に他のオブジェクトへのリンクやリンクの配列を入れることができ、これにより DAG を構成することができる。

Merkle-paths の規定により、リンクを持つオブジェクトはリンクを値として持つ key を用いて、CID/key のような派生的 CID によってリンク先を参照することができる。

例えば、

```
% ipfs dag get bafyreibjtnctwmcejbmangiu4ahoni26pggeci7chl3gieibwraxoblpqm/collaborator
```

は {"name":"Paul Marie Ghislain Otlet"} という出力を返す。

もし、リンク先にも同様に名前付きのリンクがある場合、CID/a/b/c/... のように階層的にリンクをたどることができる。

2.4 IPNS

IPFS やそのデータモデルである IPLD は CID というデータ内容のハッシュ値から作成される識別子によってデータを参照する方式を採っているため、当然のことながらデータの内容が変化すれば CID が変化してしまうため、そのままではあるデータの最新版を追いかけることができない。IPNS (InterPlanetary Name System) はこの問題を解決するための手段として用意されている IPFS の名前解決の仕組みである。

IPNS は IPFS ネットワークのノードにコンテンツの CID を対応させるための仕組みである。ある変化し得るコンテンツを公開したいユーザーは自分の IPFS ノードにおいて、そのコンテンツに対応する公開鍵（と秘密鍵のペア）を作成することができる。

例えば、QmVmSzHuGK1VTBaUgYfUFZFWbefQ3jrLDkgndpzgntzjSQ という CID を公開したい場合、

```
% ipfs name publish QmVmSzHuGK1VTBaUgYfUFZFWbefQ3jrLDkgndpzgntzjSQ
```

とすれば

```
Published to k51qzi5uqu5dgexul2taew9vs76cvwui0o2kdmbujzztq1d89zsn07z8a8tgv3: \
/ipfs/QmVmSzHuGK1VTBaUgYfUFZFWbefQ3jrLDkgndpzgntzjSQ
```

という結果を得て、公開鍵

k51qzi5uqu5dgexul2taew9vs76cvwui0o2kdmbujzztq1d89zsn07z8a8tgv3

に対応する CID として QmVmSzHuGK1VTBaUgYfUFZFWbefQ3jrLDkgndpzgntzjSQ が登録される。

こうすると、

```
% ipfs cat /ipns/k51qzi5uqu5dgexul2taew9vs76cvwui0o2kdmbujzztq1d89zsn07z8a8tgv3
```

とすれば QmVmSzHuGK1VTBaUgYfUFZFWbefQ3jrLDkgndpzgntzjSQ の内容が表示できる。

この後、ファイルを書き換えて新たな CID を得た場合、同様に

```
% ipfs name publish 新しい CID
```

とすれば、公開鍵 k51qzi5uqu5dgexul2taew9vs76cvwui0o2kdmbujzztq1d89zsn07z8a8tgv3 に対応する CID がその新しい CID となり、

```
% ipfs cat /ipns/k51qzi5uqu5dgexul2taew9vs76cvwui0o2kdmbujzztq1d89zsn07z8a8tgv3
```

すればこの新しい内容が表示される。

もしブラウザーが ipfs: に対応していれば

ipfs://ipns/k51qzi5uqu5dgexul2taew9vs76cvwui0o2kdmbujzztq1d89zsn07z8a8tgv3

で表示可能である。

また。もしブラウザーが ipns: に対応していれば、

ipns://k51qzi5uqu5dgexul2taew9vs76cvwui0o2kdmbujzztq1d89zsn07z8a8tgv3

で表示可能である。

あるいは、gateway.ipfs.io や cloudflare-ipfs.com などのパブリック IPFS ゲートウェイを使って、例えば、

https://cloudflare-ipfs.com/ipns/k51qzi5uqu5dgexul2taew9vs76cvwui0o2kdmbujzztq1d89zsn07z8a8tgv3

にアクセスすれば良い。

IPNS の公開鍵はデフォルトのものの他に好きなキーを作ることができる。

例えば、

```
% ipfs key gen dag-test
```

とすれば新たなキー dag-test が作られ、これを使って

```
% ipfs name publish --key=dag-test bafyreicgcphmhxby6chj7pvpk4umqbxylgqgiokpzoputhlxfah2psmrnq
```

のようにキーとして dag-test を指定して CID を登録すると、

```
Published to k51qzi5uqu5dkt1gpxsu50u9130rrjw0pfi05alkilbjs24t4ngbwftqv5hm0g: \
/ipfs/bafyreicgcphmhxby6chj7pvpk4umqbxylgqgiokpzoputhlxfah2psmrnq
```

のような結果を得て、公開鍵

k51qzi5uqu5dkt1gpxsu50u9130rrjw0pfi05alkilbjs24t4ngbwftqv5hm0g

に対応する CID として

bafyreicgcphmhxby6chj7pvpk4umqbxylgqgiokpzoputhlxfah2psmrnq

が登録される。

この CID は IPLD のオブジェクトなので、

```
% ipfs dag get /ipns/k51qzi5uqu5dkt1gpxsu50u9130rrjw0pfi05alkilbjs24t4ngbwftqv5hm0g
```

とすると JSON bafyreicgcphmhxby6chj7pvpk4umqbxylgqgiokpzoputhlxfah2psmrnq の内容である {"A":1,"B":2} が表示される。

このように IPNS は IPFS ネットワークの各ノードに設けられた key-value ストアのようなものと考えることができる。デフォルトでは self というキーが用いられ、他にも名前を指定してキーを作成することができる。登録されている IPNS のキーのリストは

```
% ipfs key list
```

で見ることができ、

```
% ipfs key list -l
```

とすると公開鍵とキーのペアを表示することができる。

IPNS は特定のノード (Peer) に依存するため、そのノードが単一障害点になりうる。しかしながら、名前解決の結果は自分のローカルノードにも一定期間[*8]保存されるため、もしローカルノードにキャッシュされていれば IPNS の発行元であるノードが落ちていても名前解決できると考えられる。また、実験的な機能ではあるが、IPFS pubsub 機能を用いて IPNS の名前解決を行う仕組みも実装されている。

2.5 DNSLink

DNSLink[7] は DNS の TXT レコードに IPNS のアドレス（自分のノードの公開鍵）を記載することにより、IPNS 上で DNS の名前解決の仕組みを利用できるようにしたものである。DNSLink では使いたいドメイン名を *domain* としたとき、_dnslink.*domain* に対する DNS の TXT レコードに"dnslink=/IPNS のアドレス" を記載することにより、/ipns/*domain*/ で/ipns/自分のノードの公開鍵/ の指す内容にアクセスできる。

例えば、IPNS のアドレスが

/ipns/k51qzi5uqu5djz2h3bzxk3wphm1g45qlpebasm07qyuq944n7lcx9nq2lfx22w

[*8] DNS の TTL (Time to Live) と同様である。ipfs name publish のオプショナル引数 –ttl で指定することができ、その既定値は 1 時間である。

であるとき、

```
_dnslink.chise.org.  3060 IN TXT "dnslink=/ipns/k51q...x22w"
```

のように _dnslink.chise.org の TXT セクションに "dnslink=/ipns/..." の設定を行った場合、/ipns/chise.org は

```
/ipns/k51qzi5uqu5djz2h3bzxk3wphm1g45qlpebasm07qyuq944n7lcx9nq2lfx22w
```

と等価になる。

3 IPFS か IPLD か

（狭義の）IPFS、すなわち、IPFS のプロトコルスタックにおけるファイルシステムを提供するアプリケーション層のプロトコルでは、一般的なファイルシステムと同様な機能を提供しており、ここに JSON や XML 等のファイルを置けば IPFS の P2P ネットワーク上においてそれらで記述された構造データを共有することができる。もちろん、JSON や XML 以外の任意のファイルを置くこともできる。IPFS ではバイト列が変われば CID が変化するため、本質的に同じ内容を表現した JSON ファイルであってもスペースやタブの数や改行の有無や位置、要素の順番等が変わっただけで CID が変化してしまう。なるべく CID は変化しない方が望ましいため、何らかの正規化を行うことが望ましい。同様に、ディレクトリー構成も決めておく必要がある。

一方、IPFS プロトコルスタックにおける下位層（データモデル層）を担うプロトコルとして IPLD [9] というものも存在する。これは RDF と同様な有向非巡回グラフに基づくデータモデルであり、構造データの記述という観点では IPLD だけでも十分である。[14] また、IPLD では JSON 等の入力形式に対して正規化を行なって CID を生成するため、同じデータを示す JSON データであればスペース・タブ・改行等の数が違っても、あるいは、オブジェクト内の属性の順番が異なっても、同じ CID が生成されるというメリットも存在する。[*9] しかしながら、現在の所、IPLD を示す標準的な URI スキーマが存在しないため、RDF との相互運用上の問題が存在したり、Web ブラウザー上で直接扱うことが難しくなるなどの問題が存在する。これに対し、IPFS や IPNS に対してはそれぞれ ipfs:, ipns: が登録されており、Web ブラウザーでのサポートも進んでいる。

こうした現状を鑑み、CHISE 文字オントロジーの実用的な IPFS 化においては、現状では IPNS の利用を前提に、IPLD を直接使うのではなく、IPFS 層を用いるのが現実的だと考えられる。

IPLD を用いる場合でも、大きなデータの自動分割を行ったり、データ閲覧や可視化のためのツールを整備すれば問題ないはずであるが、そうしたツールの整備が不十分な現状では、Web ブラウザー上で通常のファイルと同様に操作可能な IPFS の方が IPLD よりもメンテナンスコストが低いと考えられる。

[*9] 逆にいえば、IPLD ではこれらの情報は捨象された正規化された構造データのみが保持されるということでもある。

4　IPNS をどう使うか

IPNS は、（現状では通常）特定の Peer に依存しているため、分権的な仕組みとはいえず、IPNS を発行した Peer が消失すると名前解決できなくなるという問題がある。また、IPNS の名前は CID と同様に人間可読性が低い。そして、この問題を解決するために DNSLink を用いると、可読性が改善する代わりに DNS 依存性が生じてしまう。

とはいえ、DNSLink を併用したとしても、DNS のドメイン名で指し示される IPFS のノードが存在するサーバーはそのドメイン内になくても良く、それどころかインターネットとは NAT 越しに間接的に繋がっているだけで LAN の外側からは直接接続できないような環境や固定 IP が存在しないような環境であっても機能するという利点は存在し、そのドメイン上で Web サーバーを運用するのに比べると自由度は高くなるといえる。しかしながら、サービスの永続性という観点で見た場合、DNSLink で用いたドメイン名が維持できなくなれば名前解決できなくなるという点で、従来の Web が抱えていた問題の一部が残っていることは否めない。

この問題を緩和するための方法として、DNSLink は新規利用者のための入り口としてのみ用い、プログラム上では用いないという手法が考えられる。

DNS 依存性の問題だけでなく、任意のバージョンの環境の永続化を行いたい場合、プログラムでは IPNS 自体を用いず、ある特定のデータセットに対応する CID を埋め込むようにすれば良いといえる。こうすれば、将来そのデータセットの内容が更新されたとしてもその特定のデータセットを使った場合の環境が保存される。そして、データセットが更新された場合、その新しいデータセットを参照するプログラムを IPNS で指示する（更新する）ようにすれば良いといえる。アプリケーションを HTML や CSS などのファイルと JavaScript で構成すれば、Web ブラウザーさえあれば任意のバージョンの環境を丸ごと保存できると考えられる。そして、更新の際にのみ IPNS を用いれば DNSLink で用いているドメイン名が失効しても手元の環境は保全される。

4.1　転送時間の問題

IPFS は P2P (Peer to Peer) 型分散ファイルシステムであり、ネットワークトポロジー的に遠くに位置するノード間では転送に時間がかかるという問題点がある。IPFS のデータモデルである IPLD では（用いるコーデックにもよるが）最大 256KB という比較的小さなブロックを単位にデータを記述し、狭義の IPFS では大きなファイルは複数のブロックに分けて表現する。このため、遠くにあるノードから大きなファイルを要求した場合、すべてのブロックが揃うのに時間がかかり、一定時間内にデータの転送が終わらずに処理が失敗してしまうという事態に遭遇しがちである。例えば、「一」や「丿」や「口」といった生産性が非常に高い部品の ideographic-products 素性値データは１ブロックに収まらないためこの問題が生じやすい。

この問題を解決する方法としては、大きなデータを１ブロックに収まるように分割して、データが揃ったものから処理するようなアルゴリズムを用いることが考えられる。つまり、狭義の IPFS

がやっているようなことを対象アプリケーション用にチューンした形で自前で行う訳である。ただ、IPFS 化に際して個別に処理ロジックの修正を行うのは手間がかかり、バグの混入も招きかねず、人文情報系データベースの現実的な分権化という観点では現実的ではないかも知れない。

なお、ファイルシステムとしての IPFS において、一つのディレクトリーに数万個を超えるような多数のファイルやディレクトリーを作成した場合、そのメタデータやリンク情報が１つのブロックに収まりきらないため、多数のブロックの連鎖によって表現されることになり、大きなデータの場合と同様な問題が生じる。よって、ファイルのリストを取るような処理を行う場合、１つのディレクトリー中のファイルとディレクトリーの総数はせいぜい数百個に収めたほうが現実的である。

転送速度の問題に対処するための別のアプローチとして、大きなデータに関して P2P 的な転送を諦めるという方法も考えられる。もし、データを置いているノードとの間に十分な帯域があるなら、例えば Kubo の場合、自分のノードの `~/.ipfs/config` を設定し、データを置いているノードを自分のノードのピア（peer; 隣接ノード）あるいはゲートウェイとして陽に指定してしまうことが考えられる。また、データセット全体を Git リポジトリにして別途配布し、各ノード上でそのクローンを IPFS 上に投入するという方法も考えられる。

5　IPFS 版 CHISE IDS 漢字検索

CHISE 関連 Web サービスの IPFS 化の実証実験の一環として「CHISE IDS 漢字検索」の IPFS 化の試みを行った（図 3）。[*10] [15] これは CHISE 文字オントロジーの ideographic-products 素性のデータを IPFS 上に置き、JavaScript のプログラム (ipfs-ids-find) [11] から参照し集合演算を行うことで「CHISE IDS 漢字検索」[10] と同様な機能をサーバーレスで実現したものである。ipfs-ids-find は npm install 可能な JavaScript のモジュールの他、コマンドラインインターフェース版 [12] と Web サービス版 [13] が存在する。

この Web サービス版（「IPFS 版 CHISE IDS 漢字検索」）は HTML と CSS と JavaScript のファイルだけで構成されており、プログラムにデータセットの CID を埋め込むことで IPNS や DNS に対する依存性を除去している。このため、ある版の CID、例えば、Ver.0.5.2 の CID である Qme94UnoCa5fXGUNFnVByyf9sWdbSpbUQDhVsjHLvds4DA を用いて `ipfs://ipfs/Qme94UnoCa5fXGUNFnVByyf9sWdbSpbUQDhVsjHLvds4DA/index.ja.html` を参照すれば、このアプリケーションとそこで参照しているデータセットを１つ以上のノードで pin し続けるかぎり、[*11] この HTML, CSS, JavaScript を解釈可能な Web ブラウザーがあればいつでもこの版を実行することが可能である。

IPFS 版 CHISE IDS 漢字検索は IPFS 非対応な Web ブラウザーのために `https://www.chise.org/ipfs/` を IPFS gateway として用いたが、ローカルで Kubo の ipfs daemon を動かし、IPFS 対応ブラウザー（例えば、Firefox で拡張機能「IPFS コンパニオン」[*12] を有効にした状

[*10] `https://www.chise.org/ipfs/Qme94UnoCa5fXGUNFnVByyf9sWdbSpbUQDhVsjHLvds4DA/index.ja.html`
[*11] これをずっと動かし続けたいのであれば、自分のノードに pin すれば良い。
[*12] `https://github.com/ipfs/ipfs-companion`

図 3　IPFS 版 CHISE IDS 漢字検索 Ver.0.5.2

態）で動かすとローカルの IPFS ノードを介して実行される。

また、IPFS の転送速度の問題を考慮し、IPFS gateway となる www.chise.org に Git もしくは rsync を用いてあらかじめデータセットを転送しその IPFS ノードで ipfs add -r することで P2P 転送なしにデータが揃った状態を実現し高速化を図っている。

こうした工夫により、概ね従来版よりも高速な処理を可能としており、実用的に利用可能な代替サービスになったと考えられる。

6　CHISE 機能全体の IPFS 化

CHISE 機能全体を IPFS 化する場合、XEmacs CHISE の提供する基本となる CHISE 関連機能、すなわち、

get-char-attribute　ある文字オブジェクトの素性値を得る
put-char-attribute　ある文字オブジェクトの素性値を設定する
define-char　文字素性の集合によって文字オブジェクトを定義する
map-char-attribute　ある文字素性に対するイタレーター（ある文字素性を持つ各文字に対し文字

と素性値という 2 引数を取る指定した関数を適用する）

encode-char　ある文字オブジェクトの指定した CCS における符号位置を得る

encode-char　指定した CCS の符号位置に対応する文字オブジェクトを返す

といったものである。より複雑な処理はこれらの組み合わせで実現することが可能である。[*13]

put-char-attribute や define-char といったデータの変更を伴う処理を無視すれば、これらは文字オブジェクトの素性値を表現するファイルと CCS と符号位置から文字オブジェクトのパス名を得るためのインデックスファイルによって構成することができる。

但し、文字オブジェクトの総数はおおよそ 50 万前後、素性対はおおよそ 180 万〜200 万程度存在し、その数は日々増えつつあるため、ナイーブに実装して 1 つのディレクトリーに多数のファイルやディレクトリーを作ってしまうとうまく動かなくなってしまう。特に、イタレーターを現実的な速度で動くようにするためには工夫が必要であろう。

7　おわりに

IPFS を用いて CHISE を再構成する際の問題点とその解決法について議論した。CHISE の基盤である CHISE 文字オントロジーは有向非巡回グラフ (DAG) で記述されたデータセットであり、ここの文字オブジェクトの記述は比較的コンパクトである代わりにその総数は多いという特徴がある。この性質から IPLD での記述に向いていると考えられるが、現状、Web ブラウザーでの利用を考えれば、IPLD を直接使うよりも（IPFS プロトコルスタックのアプリケーション層に位置する）ファイルシステムとしての IPFS（狭義の IPFS）を用いた方が無難であると考えられる。また、IPFS における P2P 転送の遅さの問題を鑑みれば IPFS 以外の手段、例えば、Git リポジトリを介し git clone することや、rsync 等での転送、を併用した方が現状では現実的であるかもしれない。将来的には IPFS pubsub を利用した購読型ルーティングや Accelerated DHT 等の手法も試したい。

こうした議論を踏まえ、CHISE 関連 Web サービスの IPFS 化の実証実験の一環として、「IPFS 版 CHISE IDS 漢字検索」を試作した。これは従来の「CHISE IDS 漢字検索」の持ついくつかの機能は省略されているものの、部品の集合を指定して漢字を探すという基本機能に関しては十分機能し、前述した P2P 転送の問題を解決するための工夫を行うことにより、速度的にも従来版に負けない検索を実現した。

但し、これは ideographic-products 素性という単一の文字素性だけを対象としたものであり、1800 程度存在する全素性を扱うことや CHISE を構成するために必要なすべての基本関数を実現するためにはもう一工夫が必要である。

また、書き込みを伴う処理を実現するためには、CAP 定理、すなわち、

一貫性 (Consistency)　すべてのデータ読み込みにおいて、最新の書き込みデータもしくはエラー

[*13] encode-char は get-char-attribute で実現することができる。

のどちらかを受け取る。

可用性 (Availability)　ノード障害により生存ノードの機能性は損なわれない。つまり、ダウンしていないノードが常に応答を返す。単一障害点が存在しないことが必要。

分断耐性 (Partition-tolerance)　システムは任意の通信障害などによるメッセージ損失に対し、継続して動作を行う。通信可能なサーバーが複数のグループに分断されるケース（ネットワーク分断）を指し、1 つのハブに全てのサーバーがつながっている場合は、これは発生しない。ただし、そのようなネットワーク設計は単一障害点をもつことになり、可用性が成立しない。RDB ではそもそもデータベースを分割しないので、このような障害とは無縁である。

のうちの C, A, P の 3 つ全てを満たすことはできないことを考慮しなければならない。このため、Conflict-free replicated data type (CRDT) というデータ構造が提案されている。これは C を諦める代わりに A と P の 2 つを満たすための方法であり、IPFS pubsub ではこのデータ構造の実現が進められている。この仕組みを用いる場合、一貫性 (Consistency) は諦める必要があるが、文字の記述において一定の制約をかけることにより、各ノードで別々に書き込まれたデータを集約する際に整合性が回復されるような仕組みを用いることが考えられる。また、こうした分散システムに起因する問題とは別に、各ユーザーがローカルノードに保存した自分の文字オントロジーをメインストリームにフィードバックするための仕組みやワークフロー、ガイドラインなどを考える必要もあるだろう。

このようにさまざまな課題はあるものの、現状の CHISE 関連 Web サービスの IPFS 化は可能であると考えられ、今後は CHISE の基本関数の実装を進め、より具体的な課題について検討を進めたいと考えている。

参考文献

[1] IPFS Companion. `https://github.com/ipfs/ipfs-companion`.

[2] IPFS Desktop. `https://github.com/ipfs/ipfs-desktop`.

[3] IPFS Web UI. `https://github.com/ipfs/ipfs-webui`.

[4] Juan Benet. IPFS - content addressed, versioned, P2P File System (draft 3). *arXiv preprint arXiv:1407.3561*, 2014 年.

[5] Carsten Bormann and Paul Hoffman. *Concise Binary Object Representation (CBOR)*. Internet Engineering Task Force (IETF), 2013 年 10 月. RFC 7049.

[6] David Dias. IPLD—the “thin-waist” merkle dag format. `https://github.com/ipld/specs/blob/master/IPLD.md`.

[7] Protocol Labs. DNSLink. `https://docs.ipfs.tech/concepts/dnslink/`.

[8] Protocol Labs. IPFS is the distributed web. `https://ipfs.io/`.

[9] Protocol Labs. IPLD. `https://ipld.io/`.

[10] 守岡知彦. CHISE IDS 漢字検索. `https://www.chise.org/ids-find`.

[11] 守岡知彦. ipfs-ids-find. https://www.npmjs.com/package/ipfs-ids-find.

[12] 守岡知彦. ipfs-ids-find-cli. https://www.npmjs.com/package/ipfs-ids-find-cli.

[13] 守岡知彦. Ipfs 版 chise ids 漢字検索 ver.0.5.2. https://www.chise.org/ipfs/Qme94UnoCa5fXGUNFnVByyf9sWdbSpbUQDhVsjHLvds4DA/index.ja.html.

[14] 守岡知彦. 内容アドレッシングを用いた多粒度漢字構造情報表現の試み. 情報処理学会論文誌, Vol. 61, No. 2, pp. 171–178, 2020 年 2 月.

[15] 守岡知彦. 漢字構造検索機能の ipfs 化の試み. じんもんこん 2023 論文集, pp. 161–168. 情報処理学会, 情報処理学会, 2023 年 12 月.