部長挨拶

櫻井 謙司 ⟨sakura@mma.club.uec.ac.jp⟩

2001/11/22

本日は、MMA にお越しいただきありがとうございます。

MMA は ”Microcomputer Making Association” の略で、1970 年代に出来たと伝えられています。

以前は MMA の名前通りにコンピューターを製作していたようですが、最近はパソコンの高性能化、低価格化に伴い、活動の中心はハード製作から、ネットワーク関連の研究、ソフトウェア開発にシフトして来ました。

そのような流れの中、今年はプログラミング能力を向上させるべく、ACM の主催する国際大学プログラミングコンテスト (ICPC) に参加しました。初めての参加でしたが、思いもかけず国内予選を突破し、函館未来大学でのアジア地区予選に進出することができました。

今年は 1 チームのみの参加でしたが、来年は 3 チームほど参加して、アジア地区予選に出られればいいと思います。

今年の MMA は、IPsec の展示、1、2 年生による展示、恒例のジャンク屋等をやりますので、どうぞ楽しみください。

ADSL

kawai ⟨kawai@mma.club.uec.ac.jp⟩

2001 年 11 月

概要

最近よくテレビなどで「ADSL」と言う言葉を耳にします。なんかおっさんが勇気をだして木村拓也の胸にある青い FLETSって書いてあるボタン押すような CM があったりとか。どっかの会社が ADSL を変えるとか言ってるけど結局はプロバイダの CM だったりとか。そんなわけでちょっくら「ADSL」とやらを調べてみよ〜じゃないか、ってわけです。

1 原理

普通の電話回線（アナログ回線）って言うのはメタルケーブルで周波数は 300Hz〜3.4kHz を使っているそうだ。しかしこれはそのメタルケーブルに乗せることができる周波数の極一部しか使っていない。

そこでそれよりも高い周波数の波を使うことで今までの電話も使用できデータの通信もできるようにする。これが xDSL と呼ばれる技術だそうだ。ADSL は xDSL のひとつでアップロードとダウンロードのスピードが違う非対称なのが特徴だそうで。んで ADSL で標準化されているのが 1.1Mhz までの帯域を使う G992.1 勧告の G.dmt 方式（フルスペック）と、その半分の 552kHz まで使う G992.2 勧告の G.lite 方式（ハーフレート）の 2 種類がある。しかし G.dmt 方式を使うと音声通信を干渉するためスプリッタという装置が必要になる。よって G.lite 方式を使うほうが一般的だそうだ。

接続には RFC2516 によって標準化された PPPoE というプロトコルを使って ISP とやらに接続してインターネットを楽しめるようにするらしい。

2 わからん単語がいっぱい

G992.1 or G992.2 勧告、RFC2516 によって標準化された PPPoE、ISP なんじゃ？　これらは。何かを知ろうとするとわけのわからん言葉が多くて困るね。（俺が無知なだけだってば

3 G992.1 or G992.2 勧告

これは国際電気通信連合 (ITU) という国連の下部組織によって出されたものだそうだ。具体的に G992.1 orG992.2 がどんなものが見ようとしたのだが、どうやらパスワードがないと文章を見せてくれないようだ。残念ながら中身はわからない。

4 RFC2516 によって標準化された PPPoE

PPPoE と言うのは Point to Point Protocol over Ethernet の略で元々 PPP を LAN(Ethernet) で使えるようにようにしたものだそうだ。んじゃRFC2516ってなんでしょう？これは IETF という団体がだした文章 (RFC) の 2516 番目のものってことらしい。して中身は英語だったので俺が適当に訳してみました。[1] 全文載っけようかと思いましたが止めます。長過ぎです、適当にまとめますと 1 対 1 の接続だった PPP を 1 対複数で接続できるようにしましょう。それは Discovery 段階と PPP セッション段階の 2 つの段階があります。それぞれの段階でクライアントとサーバの間でパケット投げ合います。そんな感じです。もっと詳しく知りたいなら RFC1661,1662,2364 あたりもみないとダメなようです。
そんな気力ありません !! (

[1] `http://delegate.uec.ac.jp:8081/adsl/rfc2516_j.txt` においておきました。(編)

5 ISP

これは Internet Service Provider の略でようはプロバイダことだそうだ。

6 最後に

まあこんなもんしか調べられませんでした。ADSLってもんがどんなもんだか多少はわかったつもりになってもらえればこれ幸いです。以上終わり。

参考文献

[1] DSL について http://www.planettt.com/adsl/adsl.htm

[2] スキーデジタル用語辞典 http://yougo.ascii24.com/

[3] FC2516 http://www.ietf.org/rfc/rfc2516.txt

arima ⟨arima@mma.club.uec.ac.jp⟩
2001 年 11 月

概 要

私はMMAに入りましたが初心者で、知識はとても先輩方にはかないませんので今回は７年間も触っているが全然詳しくない、素人から見たパソコンについてを自分なりに語ろうかと思います。

突然ですがここ近年で大きく変わったのはネットワーク関係でしょう。私がインターネットに手を出したのはつい２年程前で、正直その方面の知識はあまりありません。そんな素人でも簡単にネットワークに繋げられる時代になったというのは喜ばしいことです。身近な例として私は、学校の課題レポートが出たりして手頃な参考書が無い時力を発揮しました。検索エンジンでキーワードを調べれば大抵自分が求めていた情報が手に入ります。ネットワークを皆が使うになった恩恵に預かることが出来ました。しかしネットワーク自体はもっと昔から存在していて、当然昔のパソコンでも繋ごうと思えば繋ぐことも可能なはずです。

そこで昔のパソコンでもインターネットに繋げられるか実験してみました。機種はｄｉｇｉｔａｌ製のノート型、ＯＳはＷｉｎｄｏｗｓ９５でおそらく６年ほど前に出たと思われるものです。（といってもＷｉｎ９５は自分で無理やり入れたのですが）ＣＰＵはＰｅｎｔｉｕｍ以前のものだしメモリも２４メガしかなく、今売られているものに比べたらひとたまりもありません（？）。それでもルーターを通してＡＤＳＬ接続を行ってみました。ちなみに７５０ｋｂｐｓぐらいでます。

ＩｎｔｅｒｎｅｔＥｘｐｌｏｒｅｒ３を起動しました。驚いたことに早いではありませんか。どのｗｅｂページもあっという間に表示してくれます。今のパソコンと大差ありません。６年前のパソコンでも十分に動作することがわかりました。調子に乗ってＩｎｔｅｒｎｅｔＥｘｐｌｏｒｅｒをＵＰＤＡＴＥでバージョンアップしようとしました。ところが遅いではありませんか。どの作業も時間がかかります。てっきりハングアップしたかと思ってしまうほどです。やはりデータを扱うには無理があるようです。ただネットサーフィンするだけならこれでもいいようですが。

次にいちかばちかＷｉｎｄｏｗｓ３．１でインターネットをやってみようと思いました。機種はｐｃ９８２１LP でノート型、能力は上のとあまり変わりません。しかしここでひとつ重大な問題が発生しました。Ｗｉｎ３．１で使用できるＬＡＮカードが見つからなかったのです。ＡＴ互換機でないことも見つけられなかった原因のひとつでした。さすがにここまでは一般的には対応させていなかったみたいです。残念。

以上があまり大したことはやっていませんが実験の成果です。しかしMMAに入っていなければとてもこんなパソコンをいじってみようとは思わなかったでしょう。少なくとも一台のパソコンが現役に復帰できそうです。ＷｉｎｄｏｗｓＸＰが発売されたというのにいまだに昔のＯＳをいじっているのも何ですが、使えるものはやはり使わないともったいないので、皆さんも５，６年前のパソコンを持っていたらネットに繋いでみましょう。何かに使えるかもしれません（サーバーとか？）。新しくパソコンを買おうと思う人にはお薦め出来ませんが、値段は安くできます。ただしこれはノート型だけの話で、デスクトップ型はこの限りではありません。家が狭いのでノート型しかないのが理由です。すみません。

最後に、ＭＭＡはＯＳがｕｎｉｘのサークルなのに話が完全にＷｉｎｄｏｗｓになってしまいました。申し訳ありません。

WebDAV解説～詳細編～

pokka ⟨pokka@mma.club.uec.ac.jp⟩

2001 年 11 月

概 要

WebDAV[1]は、その名称からもわかるとおり、Web を通じて分散オーサリングを行うための新しいプロトコルだ。地理的に分散した Web ページの制作者が協力して Web リソースを作成し、編集できるように HTTP を拡張し、それに基づく分散型ファイルシステムを実現する。

Windows プラットフォームや MacOS 上では、広い範囲で利用可能になっている。例えば、Windows2000 の「Internet Information Server(IIS)」の場合、それに「FrontPage Extension」を組み込むだけで、WebDAV サーバとして機能するようになる。しかも、Windows 用の Web フォルダは、恐らく世界で最も普及している WebDAV クライアントであると言える。また、「Microsoft Office2000」も、一般ユーザー向けのアプリケーションとしては、世界で初めて WebDAV クライアント機能を標準でサポートした。このほか、MacOS X の Finder にも WebDAV クライアントが組み込まれており、その環境は同 OS のローカル・フォルダと同様に違和感なく利用できるようだ。加えて、開発途上ではあるが、MacOS 8.1 以降に対応した WebDAV クライアント「Go1iath」も GPL ライセンスの下でリリースされている。

さらに、アドビシステムズの「GoLive5」および「Acrobat5」、およびマクロメディアの「Dreamweaver4」といった商用のアプリケーションにも、WebDAV クライアントが組み込まれており、これらの製品にはそれぞれ、Windows 版と MacOS 版が用意されている。このように、WebDAV を取り巻くソフトウェア環境は急ピッチで整備されつつあり、その潜在的なユーザー数もかなりの数に達している。今後、MacOS や Windows プラットフォームのみならず、Linux を含むさまざまな 0S 上での WebDAV サポートが進展すれば、WebDAV の裾野は一層の広がりを見せることになろう。

1 URL ベースのリポジトリ操作

Web ページの制作者 (以下、Web オーサー) らは、もっぱら手もとのマシンで Web リポジトリを構築/編集し、FTP を使って Web サーバ (側のリソース) との同期を取ってきたのである。むろん、この方法でも、1 人の Weh オーサーが Web リポジトリを管理するのであれば問題はない。ただし、複数の Web オーサーが共同してリポジトリを管理するのには向いていない。例えば、複数の Web オーサーによ 1 る分散オーサリングを行うには、少なくとも編集中のリソースを排他的にロックできる機構が不可欠となるが、こうした仕組みは FTP では実現できない。また、HTTP1.1 の PUT 機能にしても、このような仕組みを作るうえでは役不足だ。

そこで浮上してくるのが、WebDAV である。WebDAV は、分散オーサリングの必須機能、すなわち (編集中のリソースの) 排他ロック機能をプロ. トコル・レベルで備えている。しかも、WebDAV を用いれば、サーバのディレクトリ階層を抽象化し、それを URL ベースのものに置き換えることが可能になる。そもそも、Web 技術の大きな利点は、すべてのネットワーク・リソースを URL によって表現しうる点にある。これに対して、FTP は、URL を直接的に扱うことができない。そのため、FTP の場合、Web リポジトリを格納するサーバのディレクトリ構造をクライアント側にじかに見せてしまうことになる。

これは、サーバの管理者にとっても、リポジトリの利用者である Web オーサーにとっても厄介な問題だ。管理者も、(セキュリティ上の理由から) サーバのディレクトリ構造をすべて Web オーサーらに開示するのは避けたい。つまり、(Web オーサーらが)Web リポジトリにアクセスするのに最低限必要な情報のみを、彼らに公開できればそれに謹したことはないのだ。こうした仕組みを、WebDAV は実現する。なぜならば、WebDAV では、サーバ内のリソースの指定から、名称変更、移動やコピーに至るまでの操作を、すべて URL で行えるように定義されているからである。

2 ファイル属性の抽象化

WebDAV におけるファイル操作の抽象化は、ファイル属性の操作にまで及んでいる。ファイルは通常、「名前」以外にも多くの属性 (プロパティ) を有している。例えば、作成日時や更新日時、アクセス権、所有者、MIME-type などが、それだ。これらの属性は、その種類も表現方法もプラットフォームによってさまざまに異なる。しかし、WebDAV は、そうしたファイル属性を XML で表現することにより、それらをプラットフォームから独立したかたちで抽象化すること

[1]Web-based Distributed Authoring and Versioning

ができる。このこと (つまり、属性表現に XML を採用したこと) はまた、ファイルの属性に将来的な拡張性を与えることも意味する。

実際、WebDAV の拡張計画では、(XML ベースの) ファイル属性をファイルに対するアクセス制御や、所有者の割り当てに使用することが予定されている。また、ファイル属性を、HTML のメタ・データに相当するサーチ情報を格納するための手段として用いることも想定されている。例えば、HTML ページ以外のさまざまな Web コンテンツ (画像やオーデイオ・リソース、ワープロ文書、など) も、メタ・データを属性に格納することで、サーチ・エンジンで統一的に扱うことが可能になるのだ。

このほか、ディレクトリ操作に欠かせない機能として、その一覧表示がある。WebDAV の場合、ディレクトリに相当するリソース表現の手法としてコレクションという概念が導入されており、メンバ名と属性のリストをコレクションの属性として XML 形式で表現することができる。言うまでもなく、XML を用いれば、入れ子のデータ構造を自然なかたちで記述することができる。したがって、階層構造のメンバ・リストも無理なく表現することが可能なのである。

以上のように、WebDAV は特定のプラットフォームに依存しない、抽象化されたプロトコルとして設計されている。これは、NFS や SMB,AUFS といった従来型のネットワーク・ファイルシステムのプロトコルが、特定のプラットフォームやそのファイルシステムに大きく依存してきたのとは対照的だ。さらに、WebDAV では、FTP と同様にファイル全体を一括して転送することができる。しかも、WebDAV の場合、クライアントがファイル中のどの部分を変更しようとしているかを、サーバ側が知る必要もない。そのため、プロトコルも単純化され、ファイアウォールを越えてサーバとクライアントを (緩やかに) 結ぶことが可能だ。これはまさに分散型ファイルシステムを構成するのに最適な仕組みと言えるだろう。

3　リソース編集とロックのメカニズム

通常、クライアント側での Web ページの編集作業では、リソース全体を GET して、編集を施し、それをまた PUT でサーバ側に流し込む、といった一連の操作が行われる。一般的なファイルシステムとは異なり、HTTP には、オープンやクローズに相当するプロトコルはない。ただし、HTTP の GET と PUT は、バイナリ転送によって文書を送受するので、どのような形式のファイルであっても取り扱うことができる (アクセスを制御する「.htaccess」ファイルも PUT の対象となる)。ただし、PUT 処理を行ったリソースの MIME-type の割り当ては、サーバの実装と設定に依存することになる。

ならば、複数の Web オーサーが、リソースの編集を行うとすればどうだろうか。HTTP1.1 の場合、CVS のように各自の変更内容をオリジナルのリソースにマージするような機能は提供されていない。つまり、各自の変更結果は、リソース全体を PUT することでリポジトリに反映されるようになっているわけだ。そのため、最後に PUT した修正内容が、それ以前の内容を上書きしてしまうことになる。ただし、HTTP1.1 では、リソースのま更が行われるたびに「E-Tag」と呼ばれるユニーク ID がリソースに対して発行される。そのため、ある Web オーサーがリソースに変更を加えたのち、それを PUT する際には、もとのリソースに他者による変更が加えられていないかどうかを (ユニーク ID を通じて) チェックすることができる。これに加えて、WebDAV では、リソースにロックをかけるためのメソッドが提供されている。それが、LOCK と UNLOCK だ。LOCK メソッドによるロックが行われると、「鍵」に相当するロック・トークンが発行される。つまり、ロックがかけられているリソースを変更するには、このロック・トークンが必要になるのだ。また、ロックを解除する際には、UNLOCK が用いられる。

4　ロックの時間制限

言うまでもなく、ロックを用いる際の典型的な操作の流れは、編集を行う前にリソースにロックをかけ、編集を終えて、それを PUT したあとにロックを解除する (アンロックする)、といった格好になる。より具体的に言えば、(1)LOCK、(2)GET、(3) ローカルでの編集、(4)PUT、(5)UNLOCK、といった順番で処理が流れることになるのだ。

ここで、クライアントに異常が発生し、ロックが UNLOCK されなかった場合に備えて、ロック発行時にはロックのタイムアウト時間を付与することが可能になっている。これにより、UNLOCK が実行できない事態になっても、サーバ側

で自動的にロックを解除することができる。タイムアウト値には無限大 (Infinite) を指定することも可能で、これはロックの自動解除を指定しないことと同義になる。Infinite を指定した場合は、UNLOCK を別途発行してロックを解除することになる。

一方、特定リソースがロックされているかどうかは、その属性を参照することで確認することができる。端的に言えば、ロックされたリソースの「lockdiscover」属性に「activelock」値が格納されるので、その有無を確認すれば、当該リソースがロックされているか否かがわかるわけだ。また、lockdiscover の中には、該当するロック・トークンも混在しており、そのトークンは誰でも得ることができる。というのも、WebDAV プロトコルでは、ロック・トークンの行使を制限するための認証機構を提供しないからである。それゆえに、論理的には発行者以外であっても、ロック・トークンを行使することができるのだ。

もっとも、非発行者によるロック・トークンの行使と、それを用いたロックの解除が可能かどうかは、WebDAV サーバの実装形態にゆだねられている。mod_dav の場合、ロック・トークンの所有者認証を行わない。そのため、ロック・トークンさえ取得してしまえば、他者がかけたロックを無効にすることができる。

5 排他と共用

WebDAV のロックの振る舞いには、排他ロックと共用ロックの 2 種類が定義されている。排他ロックは、ロック・トークンが一度だけ発行されるというもので、1 人の Web オーサーが、他者による (編集中の) リソースの変更を一切排除したい場合に用いられる。

対する共用ロックでは、ロック・トークンが何度でも発行できる。そのため、複数人でリソースの変更が行えるようになる。むろん、共用ロックの場合、リソースを共用しているユーザー (Web オーサー) 間で何らかの連絡を取り合い、リソース変更の整合性を確保しなければならない。そのため、WebDAV では、共用ロックの所有者一覧を属性を通じて得るための仕組みを提供しているが、Web オーサー同士の連絡手段や (リソース変更点の) 調整手段などは提供していない。実際、その仕様にも、「(共用ロック時にリソースの整合性を確保したい場合には) 電子メールなり、電話なり、チャットなり、ポストイットをディスプレイにはり付けるなりの連絡手段を取るように」と記されているのだ。そうした事情からか、ほとんどの WebDAV クライアントは、排他ロックは扱っているが、共用ロックはサポートしていない。現時点で、両方をサポートしているのは、前出の GoLive5 のみである。なお、WebDAV クライアントは、ロック・トークンの初回の取得時に、XML の「lockinfo」エレメントを送出し、ロックの型や振る舞いをサーバに知らせる。このとき、lockinfo エレメントには、ロックの所有者を値として含めることができるが、この値は必須の情報ではない。ただし、ロックの所有者名を lockinfo エレメントに含めた場合には、その情報は lockdiscover 属性を通じて参照することが可能となる。

6 ロック実行の柔軟性

以上、WebDAV のロック機構について、少々長めに説明してきたが、実のところ、この仕組みの実装は義務づけられているわけではない。実際、ある WebDAV サーバはロックの機構を有していない。または、共用ロックだけを備えている、あるいは排他ロックしか使えないというサーバも、WebDAV のサーバとして十分に存在しうる。さらに言えば、リソースごとに、ロック機能の有効/無効を決定できるような実装もありうる。なぜならば、特定のリソースに対して、ロック機能が有効かどうかは supportedlock プロパティで参照できるからである。もっとも、こうしたロック機能の無効/有効を制御するような仕組みは、mod_dav にはない。また、現時点での WebDAV クライアントは、すべてのリソースがロック可能であることを前提に設計されている。このほか、ロック機能とは直接的なかかわりはないが、WebDAV におけるサーバ上のリソースの取り扱いについて、もう 1 つ着目すべきことがある。それは、WebDAV では、HTTP1.1 の定義に従って GET と PUT が動作するように規定されていることだ。実は、この規定には、2 つの問題がある。

1 つは、CGI や PHP、および SSI(Server-Side-Include) のソース・コードと、サーバ側の処理結果として出力されるリソースとをいかに区別するかという点だ。つまり、WebDAV の仕様では、ソース・コードを直接取得するための特別な GET 方式が定められていない。そのため、この問題はサーバ側の設定や運用によって解決する必要があるのだ。

2 つ目は、GET がディレクトリ名に相当する URL をデフォルト・リソース (通常は「index.html」) を含めた URL として扱うことに関連する問題である。要するに、デフォルト・リソースが省略された URL に対して、クライアントから PUT で書き込み要求を送出した際に、それを index.html への書き込み要求とするかどうかは WebDAV サーバの実装に依存するということだ。ちなみに、mod_dav の場合は、こうした形での PUT を index.html に対する書き込み要求とはしない。

7　コレクションの操作

前述したとおり、WebDAV によるディレクトリの操作には、コレクションの概念が導入されている。ここで言うコレクションとは、ディレクトリの一群を抽象化し、URL ベースでのハンドリングを可能にしたもののことである。WebDAV の場合、GET と PUT 以外の HTTP メソッドは、コレクションに対する操作が行えるよう再定義されており、ディレクトリ一覧に相当するメンバの一覧は、コレクションの属性として取得されるようになっている。このとき、どの深さの階層までを取得するかは、HTTP ヘッダの「Depth:」で指定することができる。

例えば、この値を無限 (Infinity) にすれば、すべての階層のコレクションを取得することが可能だ。また、新しいコレクションを追加するには、MKCOL メソッドを使う。WebDAV では、同メソッドによって 1 階層単位でコレクションを追加していくようになっており、PUT によるリソースの追加は、階層の末端に位置するコレクションに対してのみ行えるようになっている。例えば、コレクションが「/」のみの場合、「/foo/bar/baz」というリソースをいきなり PUT することはできない。それを行うには、まず「MKCOL /foo」を実行してから、「MKCOL /foo/bar」を実行し、そのあとで「PUT /foo/bar/baz」を実行する、といった手順を踏む必要がある。

さらに、コレクションに対して DELETE メソッドを実行すると、コレクションが内包するすべてのリソースとコレクションが消去される。このほかコレクションをロックすると、それを構成するメンバの追加や消去にはロック・トークンが必要になる。

8　リソースの移動とコピー

一方、WebDAV では、MOVE メソッドと COPY メソッドによって、Web リポジトリ内のリソースとコレクションに対する名称変更や移動、複製 (コピー) といった操作を行うことができる。これらのメソッドを用いた場合、リソースとその属性がともに操作対象となるが、その移動先もしくはコピー先は同一サーバ内に限定される。また、同一コレクション内で移動操作を行った場合、リソースの名称は変更される。これは、Linux の「mv(1)」の場合と同様だ。さらに、リソースのコピー/移動を行う際には、そのコピー先/移動先に対する書き込み許可が必要とされる。むろん、それらがロックされている場合にはロック・トークンも求められる。

なお、コレクションをコピーする際には、コピー対象となるサブコレクションの階層を指定できる。要するに、コレクションに内包されるサブコレクションのうち、どの深さまでのものをコピーするかを指定することができる。その深さを、無限 (Infinity) にすると、コレクションに内包されるサブコレクションの内容がすべてコピーされる。逆に、深さをゼロに設定すると、コレクションだけをコピーし、そのメンバ・リソースのコピーは一切行われない。

さらに、コレクションに対して MOVE を実行する際には、深さは常に無限となる。つまり、指定したコレクションに内包されるすべてのメンバ・リソースが、その木構造ごと移動されるのだ。

9　本当の最後に

以上で WebDAV 解説-詳細編を終わりにしますが、この文を書くにあたって参考になった書籍・Web サイトをいくつか掲載します。

参考文献

[1] DG ジャパン　LinuxWorld 2001 年 12 月号

[2] 世代プロトコル WebDAV の可能性 技術仕様徹底解説 http://www.atmarkit.co.jp/flinux/special/webdav/webdav01a.html

[3] ebDAV ホーム http://www.webdav.org/

間違いだらけの趣味プログラムへの取り組み方

takkun ⟨takkun@mma.club.uec.ac.jp⟩

2001 年 11 月

概 要

趣味でプログラムを書く時に、ちょっとでも、真面目にプログラムを書こう、とか思うと、車輪の再発明をしないように、とか、きちんとしたアルゴリズムを考えてから、とか、いろいろと調べ物をしたり、頭を悩ませるような事を多くする事になります。そういう事というのはとても重要なのですが、ここでは、あえて、そういう面倒くさい事をしないプログラムの書き方について考えてみたいと思います。

1 まず書け

とりあえず、プログラムを書こうと思っているのであれば、何か目標とする最終形態のような物があると思います。たとえば、縦書きエディタが作りたいとか、メジャーでないフォーマットの画像ファイルのビューワを作りたい、とか、そういう目標のような物です。

そういう物は、あくまでも最終目的です。そんなものは最初からできっこないのです。とりあえず、おぼろげながらの考えでいいので、code を書きはじめましょう。書きはじめるというのはそれなりに重要です。書きはじめないと何もはじまらないし、書き出すことによって初めてわかってくる問題というものはそれなりにあります。もちろん、書きはじめる前に、熟考してもいいかもしれません。しかしながら、書きはじめる前に、熟考して気がつく問題というのは、書きはじめてしまってからも気がつくものです。それに、ただ考えるだけで何も書かない人よりも、そのアイディアを形にした人の方が、世間的評価はだいたい高い方向にあります。要するに、口だけのクズは掃いて捨てろ、とにかく書け、と、いうことです。

2 玉砕せよ、しかし走り続けよ

では、書きはじめてからそれが完成するまでそのまま走り続けられるか、と、いうと、普通はそうは問屋が卸してくれません。だいたい、最初に詳細な設計なんて物を考えずに書きはじめるわけですから、話がでかくなるに従ってボロが出はじめます。そう感じたら、今まで書いてきたその code は捨てましょう。そうです。捨てるのです。さっさと捨てて新しく code を書き直しましょう。なお、ここで、熟考してはいけません。ボロが出た部分、すなわちマズいと感じた部分を、マズくない形に一から作りなおすことだけに集中して、さっさと書き直すのです。code を書くのを止めてはいけません。ただただ書きつづけるのです。書けばかくほど、書き直せばかきなおすほど、ボロの真の意味がわかってくると思います。真の意味がわかったその時には、素晴らしいデータ構造とアルゴリズムがあなたのプログラムに実現するに違いありません。それに、新しく書き直す場合には、別に今までのプログラムの形やインタフェースにこだわる必要はなくなります。きっと、あなたのプログラムはよりシンプルな構造をもつようになるでしょう。

3 使える物を使え

さて、あなたの作りたいプログラムの最終形態は、いったいどんな動作をするのでしょう。コマンドラインで便利なフィルタのような物でしょうか、それとも、ウィンドウがあって、ボタンがあって、というような、便利な GUI のプログラムでしょうか。

当然、その目的に応じたライブラリやプログラム言語というものが、多分、存在しているでしょう。でも、どんなものがあるのか調べて使うという事は、あえてせずに、自分の知っている言語、自分の知っているライブラリで code を書きはじめてしまいましょう。たぶん、その便利なライブラリやプログラム言語というものは、知っているだけで使った事が無いものだったりするものが多いと思います。自分のよく知っているライブラリやプログラム言語というものは、なんとなくこれをこういう風に使おう、と、ほややーん、と、思い浮かぶはずです。そういう、慣れている物を使いましょう。

新しい物を触ると、たいていは今までにない事が出来たり、今までいろいろ面倒だった事が簡単になっていたりする、と、言われていると思います。しかし、そんなものは幻想です。そんなことができるようになるには、大抵は時間がかかってしまいます。あなたの作りたいプログラムは、今、作りたいのです。今、作りたいというその意欲が重要なのです。そんなことで足踏みなどせずに、気にしないで自分の知っているものだけでガリガリ書きはじめましょう。足踏みなどをするということは、「明日やるよ」と、言っているようなものです。そんなことではいつまでたっても完成しやしません。

4　下らないことは考えるな

プログラムが思ったとおりに動きはじめると、とても楽しい気分になって、勝手に将来の夢が膨らむことと思います。ああいう機能を付け加えると楽しそうだな、とか、こういう事ができるようにもできちゃったりするな、とか、です。そんな事は考えない事です。考えても気にしない事です。だいたい、まだ最初に考えた機能だって全てが実装できているわけじゃないのに、新しい機能を付け加えるなんて、ありえません。そんなことは考えちゃいけません。さっさと最初の最終目的まで作り上げようと努力すべきです。そうです。作り上げる事が重要なのです。だって出来上がってはじめて、あなたの当初の目的が達成されるのですから。

5　やる気がなくなったら、やめちまえ

がりがり code を書いていると、だんだんそのプログラム自体を作り上げる元気がなくなってくることがあります。そうしたら、そのまま捨ててしまいましょう。いいんです。捨ててしまいなさい。そんな作る元気がなくなったようなプログラムなんて、完成させなくたっていいんです。というか、作る元気がなくなったということは、そのプログラムのニーズ自体がなくなったとか、あなたの望む事が変わったとか、そういう理由が背後にあるのです。だって、本当に欲しいプログラムであるならば、やる気がなくなることはありえないはずです。そうです。やる気がなくなってまで、そのプログラムを書きつづける必要はありません。さっさとあきらめて、次のプログラムを書きましょう。その捨てたプログラムを作る事によってあなたが得られた知識はいつまでも残ります。それでいいんです。十分ではないですか。

6　他人に意見を求めよ

調子に乗ってプログラムが書けていたりする時は、いいのですが、たまに、簡単には回避できなかったりする問題が発生することがあります。そうしたら、お友達にそのことについて聞いてみましょう。そのお友達がその問題について興味が無くたってかまいません。相手に自分の問題を理解させるという行動は、自分の考えを整理するのにとても役に立つからです。もちろん、お友達がその問題について興味津々だった場合はもっと便利です。多分、そのお友達は答えを知っているか、答えに近い考えを持っていると思われるからです。第一、お友達は、あなたの質問に興味が無くたって、無下に扱いはしないでしょう。そうでなくても、お友達と話をするというのは、楽しい事です。少し息抜きが出来たということでも十分に効果的です。

7　とりあえず公開せよ

プログラムが思ったとおりに動きはじめると、とても楽しいです。その楽しい気分の時に、そのまま、WebPage でも作って、公開してしまいましょう。多分、自分のプログラムなんて、誰も嬉しいと思ってくれません。それでも、公開しましょう。それで、いいんです。人は、そうそうあなたに面と向かって「クズソフトを公開しやがってこのクズ！」とは言わないものです。陰口なんて、言わせておけばいいんです。観測できないものは存在しないも同じ。というか、陰口を言われるということは、それだけその相手に気にされたということです。あなたの着眼点は少しだけズレていたにしろ、いいところをついていたということです。素晴らしいではないですか。それよりもなによりも、公開するというのはなんとも大きなプレッシャーになります。このプレッシャーはかなり、重要です。公開する、ということを考えると、あ

なたの code はまた一味違った美しい code となるでしょう。また、伊達でもよいので、たくさんのソフトを公開しているゼェーというのは、ハッタリとしては十分な効果を持ちます。あなたは他人から少し偉く見えるようになるでしょう。以上のように、公開していて損をすることというのは、ほとんどありません。さっさと公開してしまいましょう。

あ、免責事項はきちんと書いておきましょうね。

8　ドキュメントを書け

code を書きはじめたら、がむしゃらに code を書きましょう。けれども、一緒にドキュメントも書きましょう。そのプログラムが何をするのか、コマンドラインオプションは何を受け付けるのか、環境変数を参照するのか、ユーザインタフェースはどうなっているのか。これについては片手間でかまいません。でも書きましょう。なぜなら、きっとあなたはしばらく前に書いた code、たとえば今書いているもののふたつ前に書いていた code の中身なんてものは、きっとおぼろげながらにしか覚えていないでしょう。そういう code というのは、後から読むと、まるで赤の他人、いや、赤の自分の書いた code となっていて、理解できっこないようになっているに違いありません。そういう時に、何をするもので、どういう事ができるのか、とかを残しておくというのは重要です。もちろん、それなりの量のあるドキュメントというものは、後で公開する時にハッタリとして十分に効果的ですしね。

9　本を読め

学校への行き帰り、部屋でぼへーっとしたい時、いろいろと空白の時間というものはできるものです。また、四六時中モニタに向かっていると腰が痛くなります。手も疲れます。肩も凝ります。たまには息抜きもいいでしょう。そういう時には、本を読みましょう。本には先人達の通ってきた道が記されています。特に、先人達の開発したアルゴリズムなどは良い題材です。ほけーっとした時間で、あなたの code をきれいにするインスピレーションを与えてくれるかもしれません。もちろん、大抵の本には、今あなたが作っているプログラムに対する答えなんて物はないでしょう。そんなことは気にしないでいいんです。多分、後で作るプログラムで役に立ちます。暇な時間は本を読みましょう。

10　そして書け、書きつづけよ

code を書くということは、それなりに大変な作業です。データ構造とアルゴリズムを考えて、それを code に書き起こす、それだけの事でも、結構ミスを繰り返す物です。しかしながら、そういうミスを繰り返す事によって、あなたのプログラミングのセンスはよくなることはあっても、悪くなることはないでしょう。実際、code を書きつづけられるという能力は、かなり、重要です。どんな時でも code を書いていたゼェーというのは、精神的に強くなれます。ちょっとやそっとの code なんてのは、書けない事はない、とか、思って書きはじめる原動力になります。良い意味の強気になりやすいということです。いかなる時でも書きつづけるというのは、大変です。時には辛い事もあると思います。でも、そういう時こそ、ニヤリと笑って書きつづけましょう。端からみれば男です。物が完成するまで走り続けるのです。なにかをやり遂げるという事は素晴らしい事です。途中で投げ出す事は誰にでも出来ます。やり遂げる、これが、大事なのです。code を書きつづけなさい。完成に向かって。

11　最後に……

趣味プログラムにかける時間があるって事は、とても素晴らしい事ですね。

デバッグ

山本 ⟨yamamoto@mma.club.uec.ac.jp⟩

2001 年 11 月

1　はじめに

「デバッグ」と言えばプログラムの「バグ」を取り除くこと当たり前のように使っているが、まだ当時１９４７年にはそんな言葉はなかった。リレー式計算機エイケン MARK　に紛れもない「虫」そのものが突っ込むまでは。その「蛾」はリレー式スイッチに挟まり MARK　を故障させた。そして、その亡き骸は故障の原因として業務日誌に貼り付けにされて、スミソニアン協会の国立アメリカ歴史博物館にそれが展示されてる。以後、エイケンに仕事の進捗状況を尋ねられるたびに、彼女、ホッパーは「虫を取っています」と答えた。また、Mark II はしばしば動作しなくなったが、そのたびに他のオペレーターたちも「デバッグ」を口にした。

これが、デバッグの由来。って蛾の死体をセロハンテープでノートに貼るか？普通。ページを開くたびに蛾の死体を目にするのは想像したらなんとも嫌な気分になる。今じゃ水分が無くなってゾンビ化[1]してると思うが、当時はまだ瑞々しかったと思う。（蛾の死体に瑞々しいって言葉使うのにも気が引けるものがあるが。）これを読んでくれてる人も今まで生きてて目にしたことがあると思われるが、例えば、道端の犬の糞なんてその典型例だ、うんこはうんこでも乾燥して干からびてるのと、まだ水分を保っている状態のものでは不快度のレベルが違う。しかも湯気が立ってたりしたときにはもう、最悪だ。強烈な悪臭とその不快感で即刻その場を立ち去れと自分の脳が命令する。他にも、蛙やミミズが道路で横たわっているのでも同様だ。それを、それを貼り付けて保存するなんて普通じゃないな、こいつらは。最初これについての記事を読んだとき、そんな経緯があったんだとちょっと感慨深く思ったが、スミソニアン博物館のページで、写真でその蛾の死体の貼り付けてるノートを見ると、そんな思いも吹っ飛んでしまった。

これではあまりにも短いので、これに関連した物を紹介すると、

エイケン MARK　は、ハーバード大の H. エイケン が、１９４３年にＩＢＭと開発した Mark Ｉの後継機。Mark Ｉは数値も計算式もカードにより入力し、記憶装置に入力後、２３桁の１０進数の乗算を数秒で行う自動逐次計算方式で、現在の計算機の原形と言えるものだった。ただし、機械内部ではリレー回路による自動処理が行われていて、一回の計算ごとにワイヤーをつなぎ変えてプログラムをセットする（ワイヤードロジック：Wired Logic）電気機械式の計算機で、２進数による初の計算機。リレー（継電器）は、電磁石が動力で、回路に流れる電流を制御し入切断する。接点がくっついたり外れたりするスイッチの開閉状態をそれぞれ 1 と 0 に割り当てることにより計算。ワイヤードロジックは、機械語の命令を、配線論理によって実現する方法。現在は RISC ベースの CPU で利用される制御方式。また、一般に公開された最初のコンピュータで、コンピュータというものが広く認識される契機を作った。

このエイケン Mark　って Computer でいうと古代ってことになるのか？これの前なんか歯車式（石器時代）だし、この後に電子式（これが中世か？）の ENIAC あたりが登場することを考えれば。ノイマン型以前の Computer について知らなかった（以後も大して知らないが）自分にとっては、どれも古代遺産と言っても、その印象はさほど変わらない。

グレース・マレー・ホッパーは、ニューヨークのバッサ・カレッジで数学の教授で、海軍の数学アドバイザーとして 1944 年、ハーバード大学の Mark I プロジェクトに参加、Mark I のプログラムとマニュアルの作成を担当。そして保険業界で初めてコンピュータを導入したプルデンシャル社に Mark I を売り込んだ、コンピュータ業界最初で最大の営業マン。その後ハーバード大学で数学を教えて、プルデンシャル社ではプログラミングを手伝っていたが、コンピュータでの事務処理を実現させるため、1949 年、エッカート・モークリ社に入社。そこで UNIVAC I 用に英語を用いてデータ処理を行える言語 FLOWMATIC（フロウマティック）を開発。

FLOWMATIC は、ソースコードに英文を用いた最初のコンパイラ型言語。1959 年 4 月、彼女は国防総省のプログラミング担当として CODASYL に参加。こうしてデータ処理言語 FLOWMATIC を土台に、COBOL が生まれた。

[1] ミイラ化？ (編)

1986 年 8 月 14 日、海軍少将として 79 歳で退官。米海軍史上最年長の将軍だった。その後も、DEC 社の上級コンサルタントとして活躍。1992 年 1 月 1 日息を引き取った。享年 85 歳。

この人、鬼のようにすごい人だな。実際、アメージング（驚異の）・グレースって呼ばれてたらしいが、彼女は電子工学には長けてなかったそうだが、それが逆に一般の人にわかりやすい言語を生む要因のひとつになったそうだ。１９５０年前後に女性で教授ってのも、あまり想像できないし、学者としてにせよ、軍に入り、プロジェクトにおいて大きな役割を果たしたと言うのは、アメリカだからなのか（アメリカでも２次大戦前後で女性の地位に変化があった）。そう考えると日本は後進国だったんだと思えてくる。

COBOL は、ビジネス言語で, データ処理は科学計算とは異なり、計算自体は単純であるが、扱うデータが複雑となるし、処理を行う人が数学に慣れていないことが多いので、数学でなく英語で記述できるように開発された。

ホッパーによって規格化が提唱、1959 年の会合で共通ビジネス言語の望ましい特徴がリストアップされ、CODASYL（データシステム言語委員会）と呼ばれるコンピュータの専門化のグループによる統一言語の開発が開始。六ヶ月で COBOL が誕生した。

Computer の歴史に興味があったり、もっと詳しく知りたい人は、下記のページなどを見たらいいと思います。多分新たな発見があると思います。汚い話もあってすいませんでした。

参考にしたもの

HOTWIRED APAN http://www.hotwired.co.jp/news/ (デバッグについてのページは再発掘できず)
デジタル進化論 http://www.nhk.or.jp/denno/rekishi/
コンピュータの歴史 http://isweb10.infoseek.co.jp/novel/l-branch/gaku-comhistry.html
ソフトウェアの２０世紀　http://www.shoeisha.com/book/pc/20c/
広島経済大学　情報処理概論　コンピュータの歴史 http://members.tripod.co.jp/miku2001/ip02.html