月刊 USP MAGAZINE
Shell Script Is The Supreme Glue Technology
FOR THE SOPHISTICATED SHELL SCRIPTERS

世界で唯一の

シェルスクリプトマガジン

2014
12 December
Vol. 20
¥500

未来に活きる!現場で使える!
データモデリング

IPv6 新時代を体感しよう!

組織文化を変える汗と涙の物語
アジャイル改善塾

中小企業手作り□化奮戦記

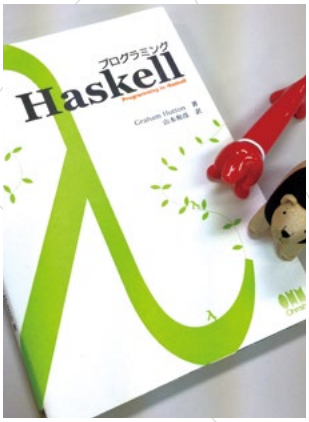
ユニケージエンジニアの作法

特集

エンジニアになりたい!

新人のためのUNIX基礎講座

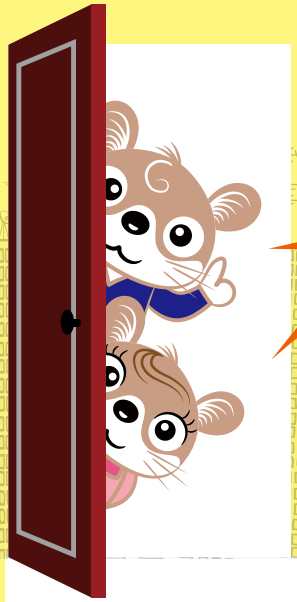
CONTENTS



月刊 USP MAGAZINE
世界で唯一の
シェルスクリプトマガジン
2014 December vol.20

USP 友の会
マスコットキャラクター
ちんじゅうちゃん


親子月
今月号の特集は、初心に帰って「新人の為のUNIX基礎講座」
そもそもパソコンってなんなのかな？



特集

エンジニアになりたい！

新人の為のUNIX基礎講座

～抽象化について詳しい人に教えてもらったよ！～

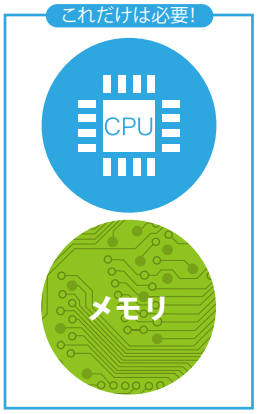
Writer：中村和敬　協力：筮田浩明、星春菜

1, コンピュータとは何か

どうしたら先輩たちみたいにシステム開発を出来るようになるんだろう？　そういえば今まであんまり意識せずにパソコンを使ってきたけど、パソコンの事何にも知らないな。そもそもパソコンってなんなのかな？

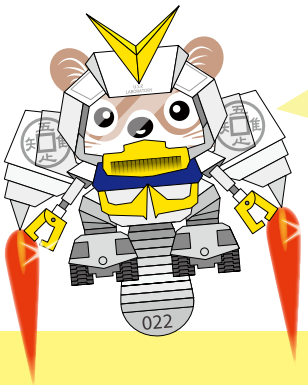
パソコンは「パーソナルコンピュータ」、つまり個人用コンピュータの略称です。コンピュータは「計算機」、計算をする機械の事です。現代のコンピュータはCPU、メモリ、HDD、キーボード、ディスプレイ等、様々な部品で出来上がっています（Fig. 1-1）。この中では、CPUが計算を行う部品で、CPUとメモリの組み合わせがコンピュータを作るための最低限の部品です。

Fig. 1-1 コンピュータの色々な部品



キーボード

CPUはメモリ上に記録されたプログラムに従って動作し、計算を行います。例えるなら、CPUは非常に計算の速い人で、メモリ上に記述した計算問題を解いてくれるというふうに言えるでしょう。
ただし、CPUもメモリも電子機器なので、計算をお願いするには電気信号を使わなければなりません。



0101010101010101010101
0101010101010101010101
0101010101010101010101
0101010101010101010101
0101010101010101010101

!?

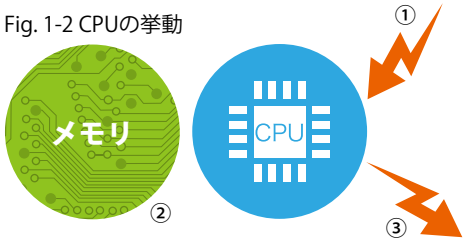
大学でコンピュータの
抽象化を研究していた、ちんじゅ子さん

入社1年目のちんじゅ太くん

IT企業に流れ着いたちんじゅ太君。入社してから雑用をこなしていますが、空いた時間の傍ら、ユニケージを勉強しています。でも、コマンドコマンド、ってみんな言うけど、一体なんのことかさっぱりわかりません。そんな時、USPマガジンで新人向けに優しい記事を作ってくれるという企画があるということで、先輩社員のちんじゅ子さんに思い切って質問してみました。

一方で、コンピュータの他の部品、HDD、キーボード、ディスプレイ等も電気信号によって動作します。
CPUはメモリの力を借りながら、電気信号を受け取って、計算し、電気信号を送り出します（Fig. 1-2）。

Fig. 1-2 CPUの挙動



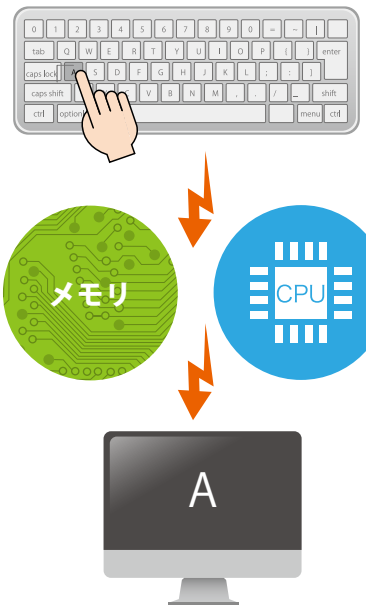
①電気信号を受け取る

②計算する

③電気信号を送り出す

この性質を使用すると、コンピュータの他の部品との間で意味のある信号のやり取りをする事ができます。

Fig. 1-3 CPUによる部品の制御

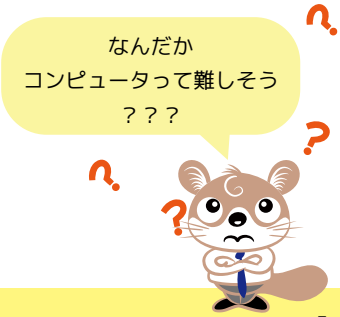


①押されたキーボードのキーに対応する電気信号を送り出す

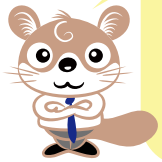
②信号に対応する文字を算出

③モニターに文字を表示する信号を送信

例えば、CPUはキーボードから送られてきた信号から、キーボードのどのキーが押されたのかを知る事ができます。
そして、ディスプレイに対して、押されたキーに対応する文字を ディスプレイに表示するための信号を作り出して送る事ができます（Fig. 1-3）。
このような仕組みにより、電気信号でしかお話できないコンピュータの部品と人間との間で、情報をやり取りする事が出来ます。



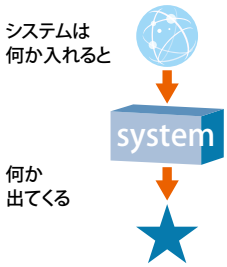
2, システムとは何か



コンピュータってすごく難しそうで、なんだかよくわからないや。
こんなんで僕にシステム開発が出来るのかな。

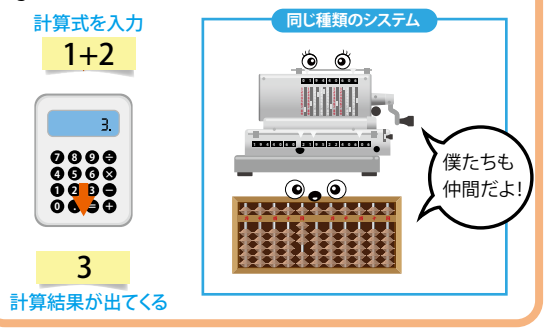
コンピュータは複雑な機械です。しかしその背後にある考え方は非常に単純なものです。それは、色々な物事をシステムとして理解する考え方です。システムは一般的な考え方ですが、IT系企業でシステムと言ったら、大抵の場合コンピュータを組み立てて作った、コンピュータシステムを指します。システムが何については、色々な説明の仕方があります。コンピュータについて理解する場合は、入力と出力のあるもの、という説明の仕方がよいでしょう（Fig. 2-1）。このとき、入力と出力がどのように行われるかについては、気にする必要はありません。

Fig. 2-1
システムには入力と出力がある



身近にある計算をする機械として、最もコンピュータらしいコンピュータは電卓です。電卓は、キーを押す事で数式を入力され、液晶画面を通じて計算結果を出力する機械です（Fig. 2-2）。

Fig. 2-2 電卓システムとその仲間

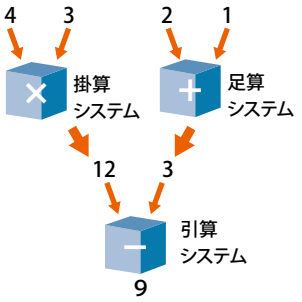


電卓をシステムとして説明する場合、「電卓は数式を入力されると、計算結果を出力するシステムである」という説明になります。そのように説明することで、実はそろばんや手回し計算機は、電卓と同じ種類のシステムだと理解する事が出来るようになります。

さて、入力と出力のあるシステムの重要な特徴として、あるシステムの出力を別のシステムの入力に繋げて、新しいシステムを作る事ができるという点があります。

実際の電子機器を繋ぐ場合には、それぞれの間で適切な変換器を通じて通信をするなどして、電気信号の形式が合うようにしなければなりません。しかしシステムとして見た場合にはそういった事を気にする必要はありません。

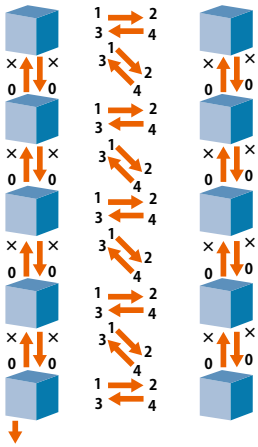
Fig. 2-3
システムを繋げて大きなシステムに



このように単純なシステムを繋ぐ事で、複雑なシステムを組み立てる事ができます（Fig. 2-3）。

システムの繋ぎ方には特に制限はないので、もっと複雑な繋ぎ方をする事もできます（Fig. 2-4）。

Fig. 2-4 もっと複雑なシステム



一方で、あるシステムを繋ぐ事が出来たからといって、出来たシステムが正しく動作するとは限りません。複雑な繋ぎ方で組み立てられたシステムは、正しく動作するかどうかも分かりにくくなります。

コンピュータなどの、複雑なシステムを組み立てる時には、一定の型に従って小さなシステムを組み立てる事で、そういった問題を回避しています。日常生活や企業の業務で発生する情報を処理するためのシステムの場合は、単純な型のみでシステムを組み立てる事ができます。

あれれ?なんか簡単そうに思えて来たぞ?
コンピュータはなんだか難しそうだったけど、システムとして考えれば良いのだね。

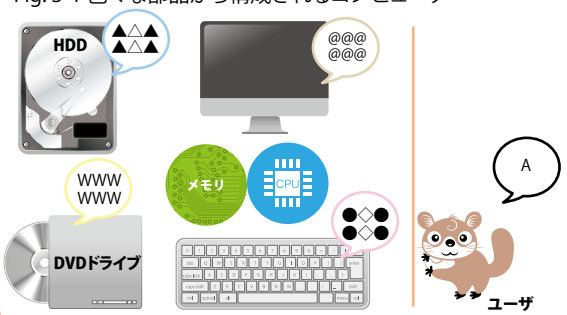
3, OSとは何か

うーん、システムの事はなんとなくわかった。でもやっぱりコンピュータは難しいなぁ。

システムとしてのコンピュータと、実際の電子機器としてのコンピュータとの間には大きな隔たりがあります。この間を埋めるのが、オペレーティングシステム(OS: Operating System)と呼ばれるプログラムです。

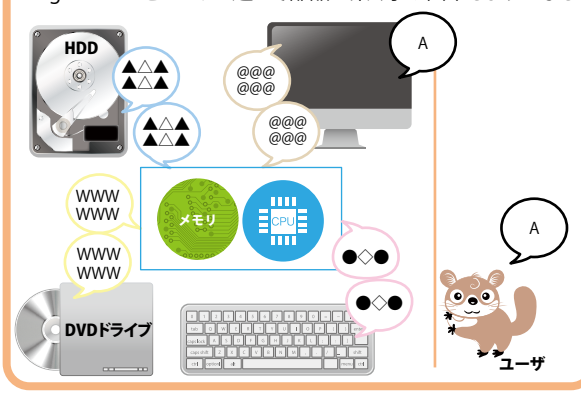
コンピュータは色々な部品によって構成されています。それぞれの部品はシステムです。各々の部品は異なる(電気信号の)言葉を使います(Fig. 3-1)。同じ種類の部品であってもメーカーや機種が異なれば、やはり異なる言葉を使います。
部品同士をそのまま繋いでも、言葉が違うのでシステムとして動作する事ができません。

Fig. 3-1 色々な部品から構成されるコンピュータ



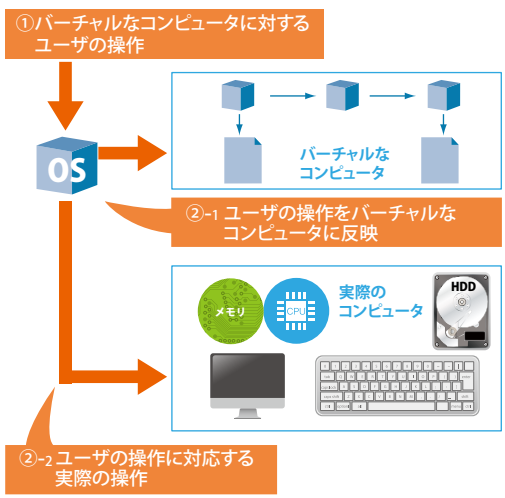
CPUはメモリの力を借りて計算をする事によって、自分以外の部品の様々な言葉を使う事ができます。CPUを通すことで、コンピュータの部品を繋ぐ事ができるようになります(Fig. 3-2)。
さらに、キーボードやディスプレイといった入出力装置を操作することで、コンピュータを使用する人間、ユーザとの対話もできるようになります。

Fig. 3-2 CPUとメモリを通じて部品を繋ぐ事が出来るようになる



また、部品には同じ目的のものがあります。例えばUSBメモリ、DVDドライブ等は、全て異なる種類の部品ですし、それぞれのメーカー毎にも操作方法が異なります。しかし、これらの部品は、いずれもデータを保存する目的で使用されるものです。
OSはこれら同じ目的の部品の重要ではない違いについては隠し、重要な事柄のみを見えるように、見え方を変えてくれます。この事を抽象化といいます。

Fig. 3-3 OSの提供する抽象世界と実際のコンピュータ



OSはコンピュータを抽象化して、ユーザには抽象化された、バーチャルなコンピュータを見せてくれます(Fig. 3-3)。これにより、ユーザはシステムとしてコンピュータを操作する事ができます。ユーザがバーチャルなコンピュータを操作すると、OSは実際のコンピュータを操作して、対応する実際の操作を行います。

何気なくパソコンを使っていたけど、OSが人間に使い易くてわかり易い様に色々動いてくれてたんだね！

4, UNIXによる抽象化

なるほど…

じゃあ、OSはコンピュータをどんなふうに抽象化してくれているの?

OSがどのようにコンピュータを抽象化して
ユーザに提供するかUNIX系OSを取り上げて説明しましょう。
UNIXは1969年、アメリカの電話会社、AT&Tのベル研究所で開発されたオペレーティングシステム(OS: Operating System)です。その後、UNIXの設計をまねたOSが幾つも開発されました。
LinuxやFreeBSDはそうして開発されたOSで、こういったOSはUNIX系OS(UNIXライクシステム)などと呼ばれています。

コンピュータの部品として、まず記憶装置が挙げられます。
HDD、USBメモリ、DVD(とDVDドライブ)等は記憶装置です。
記憶装置はデータを保存する目的で使用される部品です。
UNIX系OSは記憶装置をファイルとディレクトリという形式で抽象化します。ユーザはファイルとディレクトリを操作することで、データを保存したり読み出したり、整理できるようにします(Fig. 4-1)。

Fig. 4-1 記憶装置をファイルとディレクトリとして抽象化

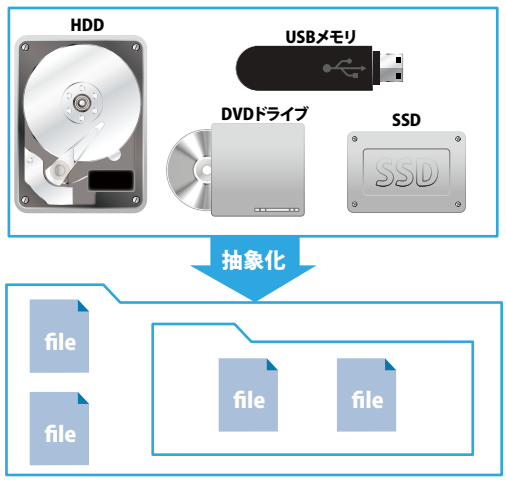
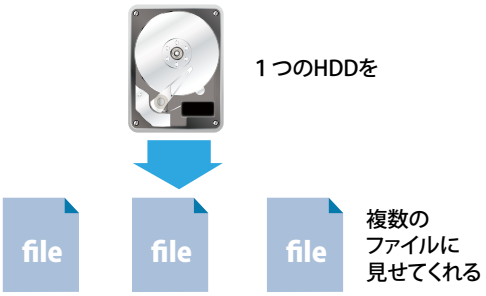
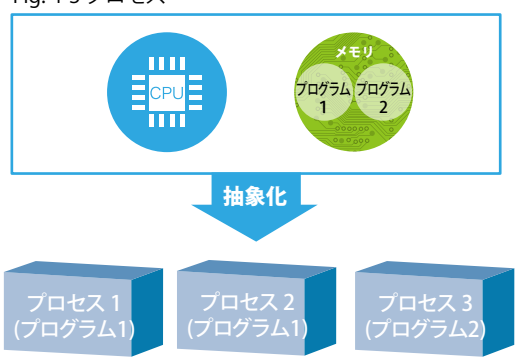


Fig. 4-2
一つの記憶装置を複数のファイルに見せかけてくれる



HDD、USBメモリ、DVDといった記憶装置は、それ自体にはファイルやディレクトリのような仕組みはなく、単純に一つのデータを記録する部品です。
UNIX系OSは記憶装置の、データを記憶する領域を細切れにして管理することで、記憶装置に複数のデータを記録できるようにします。このひとかたまりのデータがファイルになります。
言ってみればUNIX系OSは、一つの記憶装置を、あたかも複数の記憶装置のように見せかけてくれます(Fig. 4-2)。
さらに、ディレクトリという形で階層構造の中でファイルを管理するための仕組みを提供しています。
また、もう一つ重要な部品として、CPUとメモリが挙げられます。これらは計算をするため、つまり、プログラムを実行する目的で使用される部品です。また、動作中のプログラムを、プロセスと言い、UNIX系OSは動作中のプログラムをプロセスという形で抽象化し、操作できるようにしてくれます(Fig. 4-3)。

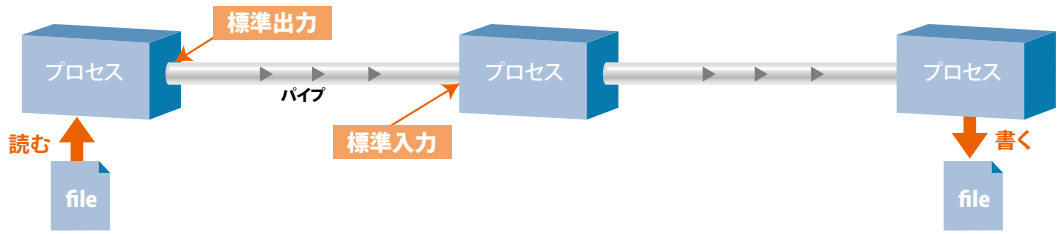
Fig. 4-3 プロセス



一つのCPUは、そのままでは一度に一つのプログラムしか実行できません。
UNIX系OSは、プログラムの走行を中断したり再開したりするCPUの機能を用いて、一度に複数のプログラムが実行されているかのように見える機能を提供しています。
UNIX系OSはこの機能により、一つのCPUをまるで複数のCPUであるかのように見せかけてくれます。

ファイルとプロセスは、UNIX系OS上でシステムを作る際の基本的な要素です。プロセスは、ファイルに対して、データを読み出したり、書き込んだりする事ができます。
UNIX系OSは、ファイルとプロセスに加えて、プロセス同士を繋いでプロセスの間でデータのやり取りが出来るようにするために、パイプという仕組みを提供しています。
このような仕組みをプロセス間通信と呼びます。
実はインターネットもプロセス間通信の仕組みの一つですが、今回はパイプに限定して説明します。

Fig. 4-4 ファイルとプロセスとパイプ



UNIX系OSのプロセスには、入力専用の口と出力専用の口があります。それぞれ、標準入力と標準出力と呼びます。
UNIX系OSを使用すると、プロセスの標準出力と標準入力を繋いでシステムを組み立てる事ができます（Fig. 4-4）。

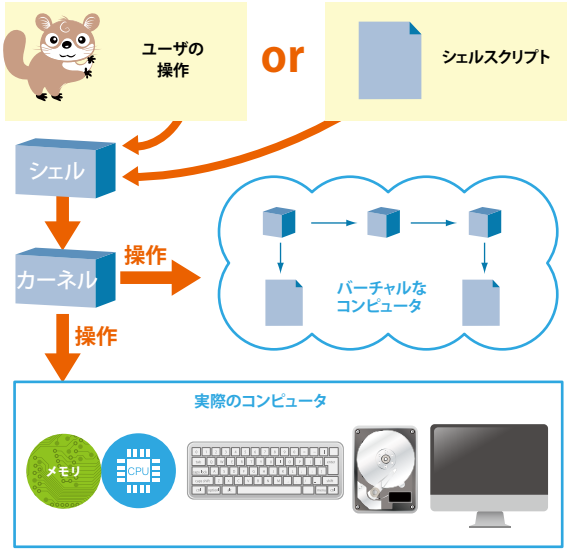
そうか、プロセスでファイルを読み書きして、さらにパイプでプロセスを繋げてシステムを作るんだね。

5, シェルによるシステム構築

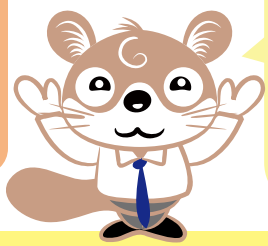
なんだか自分にもシステムを作れそうな気がして来たぞ!!
でも実際にどうしたらいいの?

UNIX系OSはコンピュータを抽象化し、ユーザがプロセスを繋いで、システムを組み立てることができるようにしてくれます。UNIX系OSは、OSの本体であるカーネルと、直接ユーザとやり取りをするシェルの、二つの要素から出来ています。

Fig. 5-1 シェルとカーネルとシェルスクリプト



ユーザはシェルを通じてカーネルを操作して、システムを組み立てます。シェルに対する操作は、通常キーボードから文字列を入力して行います。操作の手順は、文字列のデータとして表現出来るので、ファイルに保存する事もできます。ファイルに格納された操作の手順を、シェルスクリプトと呼びます（Fig. 5-1）。
シェルスクリプトの書き方を学ぶと、UNIX系OSを用いて簡単にシステムを組み立てる事が出来ます。



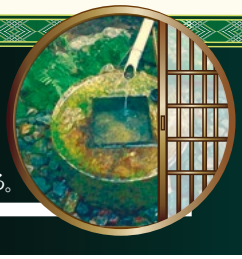
なるほど、システムを組み立てるための操作の手順をファイルに保存したものがシェルスクリプトか。
シェルスクリプトが書ければ色々なシステムを作れるということだね!ようし!!書き方を勉強してみよう!!

第十二回

ユニケージエンジニアの作法

監修:USP研究所エンジニア まとめ:松浦智之

ユニケージとは、ソフトウェアにおける一つの型である。
そこにはどんな作法があり、どんな理があるのか。
これまで口伝によって伝えられてきた作法を今、文章として残す。
前回に引き続き、コードの添削を通じて、データの持ち方を考える。



作法その十三、マスターデータはたくさん分割・複製してもよい

前回の連載では、アマチュアプログラマーが書いたプログラムをユニケージエンジニアに添削してもらい、そこから作法を学ぶという形式で議論を行いました。その進め方が好評だったこと、またユニケージエンジニアが実戦の場で書いたプログラムを見ていく連載「コードレビュー」が始まっていることもあり、本連載ではもう一度添削を受ける形式で、私が執筆を担当させていただきます。

東京メトロの「オープンデータ活用コンテスト」

今年 9/12 から 11/17 までの間、東京メトロが「オープンデータ活用コンテスト」[*1] を催していました。

これは東京地下鉄株式会社（東京メトロ）が、自社の路線や駅の設備、時刻表、運賃などの情報、そしてさらに、運行状況や列車が今どの駅に在線してるかといった現在のデータに至るまで、様々な情報を WebAPI を通じて公開するので、それを用いて世の中の役に立つアプリケーションを開発してくださいという趣旨で開かれたコンテストです。

応募して賞金を狙わずとも、鉄道の、しかも今の情報が取得できるという仕様に心惹かれるのは、鉄道オタク、ソフトウェア開発者である方なら、十分理解していただけるのではないでしょうか。更に「ユニケージの作法に基づき、シェルスクリプトでこれを開発できるのか確かめてみたい」という欲求もありました。

コンテストの応募受付が始まって早速、ユーザー登録を行い（登録しないと API は利用できない）、仕様を確認しながら利用してみました。その結果、OAuth のような複雑な認証も要らず、簡単にデータを取り出せることがわかったため、週末を使って一気に試作アプリケーションを作ってみました。

列車接近情報表示アプリ「メトロパイパー」

こうして試作したのが「メトロパイパー」というアプリケーションです。

駅へ行くと「○○行の電車が 2 つ前の駅を出発している」といった情報が電光掲示板で見られたりしますが、以前からあの情報が、どの駅についても駅にいなくても見られればと思っていました。例えば近くの喫茶店で時間を潰している時、30 分に 1 本の快速電車が 2 つ、3 つ前の駅に来ているとわかればそろそろ駅へ向かおうと思えます。また駅のホームに発車時刻ぎりぎりに到着して電車がいなかった時、それはもう出発してしまったのか、遅れていてまだ来ていないのか、任意の駅の接近情報が見られればすぐわかります。

Web から操作できるものも後から作りましたが、基本はコマンドで、知りたい駅と行きたい方面の駅ナンバー（例えば、丸ノ内線大手町駅なら M18）を引数で与えると**画面 1** のようなテキストを返します。

その後、Web ブラウザーからも使えるようにしました。[*2]

画面 2 がその画面です。まず画面上部左側の select

*1 http://tokyometro10th.jp/future/opendata/

ボックスで知りたい駅を選ぶと路線が確定しますので、Ajax を利用して右側の select ボックスに該当路線の駅ナンバーの選択肢を書き入れます。後はそれを選択すれば裏側で先程のコマンドが実行されて在線状況が確認できるというわけです。

各プログラムは三つの役割に分類される

プログラムは大きく三つに分かれます。一つ目は、最初に駅名や路線名のコードと名称の情報を WebAPI から取得し、マスターファイル群を作るためのプログラム（MK_METRO_MST.SH）。二つ目は、現在の在線状況を WebAPI に問い合わせ、先程作ったマスターファイル群と結合して最終的なテキストを生成するプログラム（VIEW_METROLOC.SH など）。三つ目は、二つ目のプログラムを Web インターフェースに対応させるため、Web ブラウザーからのリクエストに応えるプログラム（GET_LOCINFO.AJAX.CGI など）です。これらのプログラム一式は、GitHub に置いてあります[*3]のでご覧になってみてください。

マスターファイルは二種類

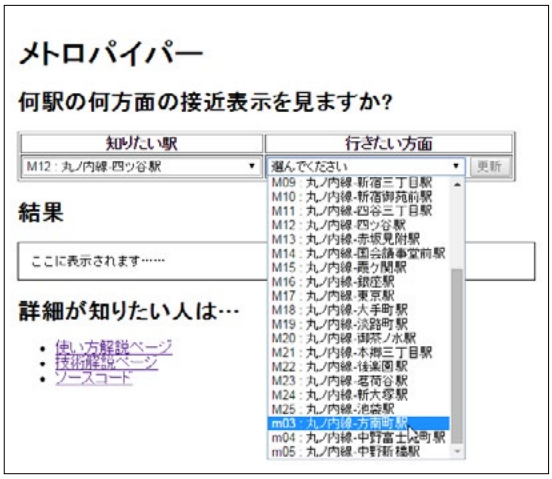
プログラム本体同様に重要なのがマスターファイルです。どの列車（行先・種別・車両製造会社）が今どの駅にいるかといった情報は全て内部コードで返ってきますので、最終的にこれを人間にわかりやすい名称に変換しなければなりません。各コードと名称の対応も WebAPI や指定 Web ページをアクセスすることでわかるようになっていますが、名称はめったに変わるものではなく、インストール時に一度わかれば十分です。そこでマスターファイルを用意するのです。

マスターファイルは二種類用意しました。一つは東京メトロの駅名のマスターファイル（SNUM2RWSN_MST.TXT）、もう一つはそれ以外（路線名や列車種別名、他社線を含む行先駅名、車両製造会社名）のマスターファイル（METRO_VOC_MST.TXT）です。東京メトロの駅名マスターだけなぜ独立して持たせているかというと、その主な理由は、内部コードと駅名以外に駅ナンバーがあるためです。先程お見せしたように、コマンド引数では駅ナンバーを受け付けるようにしているため、単純に内部コードと名称の対応だけ持っていればよいその他のマスターデータとは構造が違うためです。

```
$ ./VIEW_METROLOC.SH Z08 Z01
2014/10/05 23:26:26 現在

.. Z14 押上          各停 中央林間行（東急電鉄車両）
.. |               ↓急行 中央林間行（東武鉄道車両）
.. Z13 錦糸町
.. |
.. Z12 住吉
.. |
.. Z11 清澄白河
.. |
.. Z10 水天宮前
.. |
.. Z09 三越前
.. |
>> Z08 大手町        各停 中央林間行（東急電鉄車両）
.. |
.. Z07 神保町
.. |
.. Z06 九段下
.. |
.. Z05 半蔵門
.. |
.. Z04 永田町
.. |
.. Z03 青山一丁目    急行 中央林間行（東急電鉄車両）
.. |
.. Z02 表参道
.. |
.. Z01 渋谷          各停 中央林間行（東急電鉄車両）
$
```

画面 1.　接近情報取得コマンドの実行例



画面 2.　Web ブラウザーから使えるように対応したもの

指摘その一、データはまず絞り込みから

さて、ユニケージエンジニアにこのアプリケーションを添削してもらいました。

前回の PalPay 決済アプリケーションに比べると、ユニケージらしくなっているという評価でしたが、それでも指摘が無いわけではありません。

*2 http://lab-sakura.richlab.org/METROPIPER0/
*3 https://github.com/ShellShoccar-jpn/metropiper0

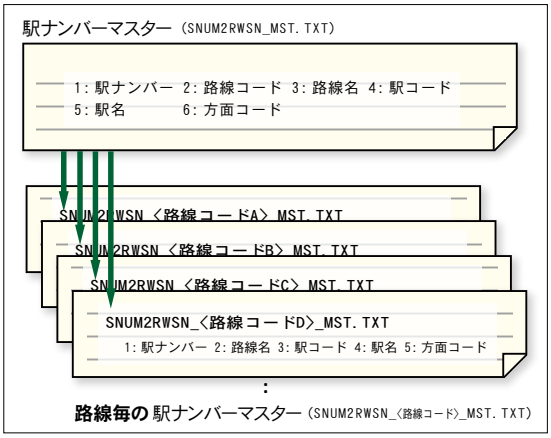


図 1.　駅名マスターは、路線毎に 1 ファイル作る

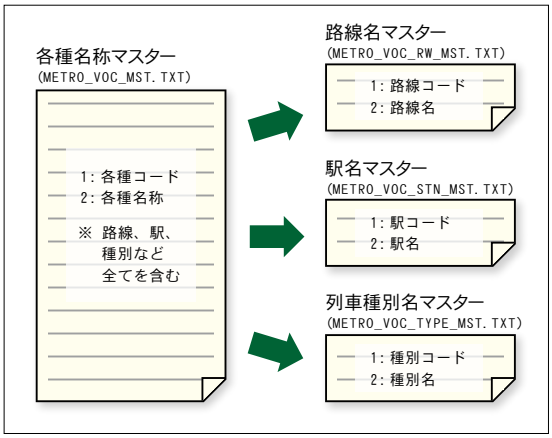


図 2.　各種名称マスターは、その種類毎に 1 ファイル作る

リスト 1.　GET_SNUM_HTMLPART.AJAX.CGI（抜粋）

```
 :
111 # --- HTML 本体を出力 -------------------------
112 cat "$Homedir/DATA/SNUM2RWSN_MST.TXT"     |
113 # 1: 駅ナンバー（sorted） 2: 路線コード 3: 路線名
114 # 4: 路線駅コード 5: 駅名 6: 方面コード     #
115 awk '{print substr($1,1,1),$0}'            |
116 sort -k1f,1 -k2,2                          |
117 awk '{print $2,$4,$6}'                     |
118 uniq                                       |
119 # 1: 駅ナンバー（sorted） 2: 路線名 3: 駅名  #
120 grep -i "^$rwletter"                       |
121 mojihame -lFROM_SNUM_LIST $Tmp-htmltmpl -
 :
```

データの絞り込みは、早い段階で行うべき。

grep と sort の順序が逆

まず指摘されたのは Web 対応のために作ったプログラムである GET_SNUM_HTMLPART.AJAX.CGI でした。これは画面 2 の説明の際に出てきたもので、画面上部左側の select ボックスで知りたい路線の駅名を選択した際、右側の select ボックスにその路線の駅だけに絞り込んだ選択肢を生成するものです。

リスト 1 にその指摘箇所を抜粋します。駅マスターファイルから指定された路線（Web ブラウザーが指定した路線の頭文字が $rwletter に格納されている）の駅ナンバーと路線名、駅名を駅ナンバー昇順で返そうとしていますが、120 行目で実行している指定路線への絞り込み grep は、むしろ 112 行目の cat の直後にやるべきという指摘です。特に sort コマンドの計算量は、行数が n 倍になれば n 倍では済みませんので他のコマンドよりも影響が大きいのです。

指摘そのニ、マスターデータは、適宜分割・複製

更に重要な指摘を受けました。今度は、現在の在線状況を調べてマスターデータと結合し、最終出力を作るためのプログラム（VIEW_METROLOC_*.SH）でした。

毎回加工せず加工済データを作り置くべき

リスト 2 を見てください。まずは 236、241 行目の join コマンドです。ここでは駅を表す内部コードを駅ナンバーに変換するために、そのための変換表ファイル（$Tmp-sc2snum）と結合するという処理を行っています。このファイルは 131~142 行目で生成しています。この部分はコマンドを起動する都度実行されますが、駅マスターファイルから部分的に切り出しているに過ぎません。駅マスターは都度変わるデータではないので毎回作るのは無駄な作業です。毎回作るくらいなら作り置いておけばいいという指摘でした。

そうすれば毎回の処理も軽くなるうえ、プログラムも短くなります。在線状況を調べるプログラムは調べる作業に徹し、都合の良いマスターデータ作成はマスターデータを作るプログラムに任せるべきなのです。

兼ね合わせたテーブルにしない

また、同じくリスト 2 の 261、266、271 行目についても指摘されました。

ここでやっていることはそれぞれ、列車種別名の結合、行先駅名の結合、車両製造会社名の結合です。先に説明したとおり、METRO_VOC_MST.TXT というのは自社駅名以外の各種名称を引くためのマスターファイルなのですが、「なぜ一つのファイルのままなのか。種別マスター、行先駅マスター、車両製造会社マスター

リスト 2.　添削対象の、VIEW_METROLOC_0.SH（抜粋）

```
 :
130 # --- 当該路線駅ナンバーマスターファイルの作成 ----------------------
131 cat $Homedir/DATA/SNUM2RWSN_MST.TXT |
132 # 1: 駅ナンバー 2: 路線 code 3: 路線名 4: 路線駅 code 5: 駅名 6: 方面駅 code
133 grep -i "^${from_snum%[0-9][0-9]}"  | # 指定路線のみに絞り込む
134 tee $Tmp-this-rw-stn-mst            | # 指定路線の路線駅マスター
135 awk '$6!="-"'                      > $Tmp-dirstns # 方面駅 code を持つ駅のみに絞り込む
136
137 # --- 駅 code から駅ナンバーを引くマスターファイルを作る -------------
138 cat $Tmp-this-rw-stn-mst |
139 # 1: 駅ナンバー 2: 路線 code 3: 路線名 4: 路線駅 code 5: 駅名 6: 方面駅 code
140 awk '{print $4,$1}'      |
141 # 1: 路線駅 code 2: 駅ナンバー
142 sort -k1,1               > $Tmp-sc2snum
 :
234 # 1: 方面 code 2: 発車駅 code 3: 到着駅 code 4: 種別 code 5: 目的駅 code 6: 車両所有業者 code
235 sort -k2,2                                                          |
236 join -1 1 -2 2 -o 2.1,2.2,1.2,2.3,2.4,2.5,2.6 $Tmp-sc2snum -       |
237 # 1: 方面code 2: 発車駅 code 3: 発車駅ナンバー 4: 到着駅 code 5: 種別 code 6:目的駅 code 7: 車両所有業者 code
238 awk '{print $1,$3,$4,$5,$6,$7}'                                     |
239 # 1: 方面 code 2: 発車駅ナンバー 3: 到着駅 code 4: 種別 code 5: 目的駅 code 6: 車両所有業者 code
240 sort -k3,3                                                          |
241 join -1 1 -2 3 -a 2 -o 2.1,2.2,2.3,1.2,2.4,2.5,2.6 $Tmp-sc2snum -  |
242 sed 's/  / - /'                                                     |
 :
260 sort -k2,2                                                              |
261 join -1 1 -2 2 -a 2 -o 2.1,2.2,1.2,2.3,2.4 $Homedir/DATA/METRO_VOC_MST.TXT - |
262 sed 's/\([^. ]\{1,\}\)  /\1 \1 /'                                       |
263 awk '{print $1,$3,$4,$5}'                                               |
264 # 1: 現在居る 3 桁駅ナンバー 2: 種別名 3: 目的駅 code 4: 車両所有業者 code       #
265 sort -k3,3                                                              |
266 join -1 1 -2 3 -a 2 -o 2.1,2.2,2.3,1.2,2.4 $Homedir/DATA/METRO_VOC_MST.TXT - |
267 sed 's/\([^. ]\{1,\}\)  /\1 \1 /'                                       |
268 awk '{print $1,$2,$4,$5}'                                               |
269 # 1: 現在居る 3 桁駅ナンバー 2: 種別名 3: 目的駅名 4: 車両所有業者 code         #
270 sort -k4,4                                                              |
271 join -1 1 -2 4 -a 2 -o 2.1,2.2,2.3,2.4,1.2 $Homedir/DATA/METRO_VOC_MST.TXT - |
272 sed 's/\([^. ]\{1,\}\)  /\1 \1 /'                                       |
273 awk '{print $1,$2,$3,$5}'                                               |
 :
```

この一時マスターファイルを、毎回作成するのは計算の無駄。しかもコードも長くなる。

別のプログラムで作り置きしておけば、計算量も減るし、131-142 行目のコードは不要になる。

それぞれ、
・列車種別名
・行先名
・車両製造会社名
を JOIN しているだけなのに、混ざっているせいで、無駄な計算が発生している。

そのうえ、どの行が何の名称を JOIN しているのか分かりづらい。

と、分けるべき」という指摘でした。（図 2）

分けていれば、まず、一度の join（及びそのための sort）で扱わねばならない行数が減り、処理が軽くなります。そして、ソースコードも分かりやすくなります。今は 3 つとも結合するファイル名が METRO_VOC_MST.TXT なので、それぞれの join が一体何を結合しているのかはっきりしないのです。

実は先程の $Tmp-sc2snum も同様で、133 行目で路線の絞り込みを行わず、keycut（Tukubai コマンド）などを用い、始めから路線毎に分けたマスターファイルにしておくべき、とも指摘されました。（図 1）

マスターらしいマスターに拘らず真に必要な形式のデータを作成せよ

マスターデータの「マスター」とは（親機・子機の）親、あるいは（主従の）主という意味です。そう考えると、支配を受ける親が複数あることは不自然なので、マスターデータを複製したり、分割させたりといったことはすべきでないように思えます。

しかしユニケージでは、マスターデータとて必要ならば気にせず分割・複製を行うそうです。理由は、今見てきたとおり、無駄な処理を発生させないことと、プログラムを分かりやすくするためです。一般的に、マスターデータを分割・複製することを懸念する直接的な理由は、それによってデータの不整合が生じやすくなることであろうと思われますが、元データに変更が生じた時には速やかに複製先にも反映させるなどして、しっかりした管理を怠らなければよいのです。

そもそも、何のためにマスターデータを作るのか。それは最終的に得たいデータを効率よく生成するためであって、マスターらしいマスターデータを作るためではないはずです。

ISBN 978-4-9048-0714-9
C3455 ¥500

9784904807149

USP 研究所
定価（本体 500円+税）

1923455005002