サーバー構築/運用術

第1回 キャラクタ・ユーザー・インタフェース

引き続き連載を執筆することになりました。よろしくお願いします。今回は新連載の初回ということで，概論的な話から始めましょう。システムを理解する第一歩ということで，キャラクタ・ユーザー・インタフェースを取り上げます。

（ケイ・ラボトリー　仙石 浩明）

いよいよJavaを搭載した携帯電話が発売されました*1。既に入手していろいろ使っている方も多いことでしょう。さまざまなiアプリ*2が発表されているので，それらを一通りダウンロードするだけでも，かなり楽しめそうですが，本誌の読者の方にはぜひiアプリの開発に挑戦していただきたいと思います。

幸い，無料で入手可能なソフトウエアのみを基に，iアプリ開発環境を構築するためのキットが，ギガフロップス（本社：東京都渋谷区）から発表されています*3。このツールを使えば，あとはJavaでソース・プログラムを書くだけですね。

iアプリの場合，コード・サイズが10Kバイトまでという制限があるので，あまり大層なプログラムは作れません。実際にiアプリを実行してみた方はご存じだと思いますが，シンプルなものが大半です。

つまり開発費をぜいたくにかけて作るソフトウエア*4というよりは，アイデア勝負のソフトウエアです。趣味で作るソフトウエアが，公式コンテンツプロバイダから提供されるソフトウエアを打ち負かす可能性も十分あるわけで，挑戦してみる価値は十分にあるでしょう。

セキュリティの基本

セキュリティ対策というと，次の4点が挙げられることが多いようです。

（1）使わないサービスを停止する
（2）IPフィルタを設定する
（3）ログを定期的にチェックする
（4）最新のセキュリティ情報を入手する

なるほど，確かにこの4点はサーバーをセキュアな状態に保つために重要と言えるでしょう。しかしながら一番重要な点を忘れているような気がしてなりません。それは，

（0）システムを理解する

です。何をやっているか十分理解すること無く，(1)-(4)などの対策を行っても，十分な対策を行えないばかりか，セキュリティ対策済みであると安心してしまい，かえって危険である可能性すらあります。

例えば(4)ですが，あるソフトウエアの特定のバージョンがぜい弱であると書かれたからといって，無批判に最新版を入手してインストールするのは，あまり感心しません。新しいバージョンの方にもっと深刻なセキュリティ・ホール

＊1　著者は，次世代携帯電話のコンテンツ関連技術を研究・開発するベンチャ企業「株式会社ケイ・ラボラトリー」（http://www.klab.org/）で取締役兼CTO（最高技術責任者）を務めています。
＊2　NTTドコモの携帯電話で実行可能なJavaアプリケーションです。
＊3　GADek（GiGA-Appli Developer's Kit=http://g-appli.net/developer/gadek/）を参照してください。
＊4　開発費をぜいたくにかけて作ったソフトウエアの典型が，最近のゲームですね。ユーザーが遊んで楽しい，という次元をはるかに越えたところで泥沼の開発競争が繰り広げられています。

がある可能性だってあるからです。

ある程度枯れてきたソフトウエアの場合，無条件に危険なセキュリティ・ホールというのはあまりありません。特定の設定を行ったとき，特定の条件下でのみ危ない，というぜい弱性も数多く報告されています。もし，その特定の設定を行っていないのであれば，慌ててバージョン・アップするのではなく，バージョン・アップによって何が変わるのかじっくり検討してからでも遅くはないでしょう。

インターネット常時接続がだれにでも手軽に実現できるようになった今，セキュリティがますます注目を集めています。ところが，どのようにセキュリティ対策を行ったら良いか解説するハウツー（Howto）記事/書籍ばかりが増え*5，なぜそのような対策をすべきか，基礎から解説する記事/書籍があまりないのが現状です。

確かに即効性という点ではハウツー物の方が勝るのですが，生兵法は大けがの基とも言います。多少回り道のように思えても，急がば回れで基礎から理解した方が，より確実なセキュリティ対策が可能となります。

システムを理解する最初の一歩，ということで今回はキャラクタ・ユーザー・インタフェース（CUI）を取り上げます。見た目の取っつきにくさから敬遠されがちなCUIですが，現状はグラフィカル・ユーザー・インタフェース（GUI）だけでシステム管理を行うのは限界があります。日ごろからCUIに慣れ親しむことがシステムを理解する近道ではないでしょうか。

キャラクタ・ユーザー・インタフェース

Linuxに限らず，最近はシステム管理ツールのGUI化が進んでいます。決まりきった設定ならばGUI管理ツールを使えば手軽に実施でき，しかもツールにはある程度のエラー・チェック機能があるので，内容をあまり理解していなくても，そこそこ使える設定が可能になっています。しかし，これはセキュリティ的には一番危険なことです。

一般ユーザーが使うアプリケーション，例えばワープロ・ソフトや表計算ソフトであれば，GUI化は大いに結構なことでしょう。ユーザーはPCを道具として使っているだけですから，思った通り*6の文書作成や表計算が結果としてできさえすれば，内部で何が行われているか知る必要はありません。

また，こうした一般ユーザー向けアプリケーションは，用途が決まっているものがほとんどです。「何でもできるソフトウエアは何もできない」と昔からよく言われるように，一般ユーザーにとって使いやすくするには，ユーザーができることをある程度制限するのが一番簡単です。その結果，ソフトウエアが想定している体裁の文書（ワープロの場合）を作るのは簡単だけれども，想定外の体裁のものを作ろうとすると，とんでもなく大変，ということになります。つまり現在の一般ユーザー向けアプリケーションには柔軟性が欠けていると言えるでしょう。

では，管理ツールの場合はどうでしょうか?一般ユーザー向けアプリケーションの場合と異なり，管理者はPCを道具として使っているのではなくて，PCあるいはネットワークを管理しているのですから，PCの中で何が行われているか知ることが不可欠です。

そして，想定外の事態に対応できないツールは使いものになりません。想定された範囲の攻撃しか受けないのであれば，セキュリティ対策に苦労する必要はありませんし，PCが常に想定された範囲で動作するのであれば，トラブルは起り得ないでしょう。そもそも想定外の事態が発生しないなら管理者は不要です。

＊5　Linuxが普及し始めたころは，インストールの方法を解説するハウツー記事/書籍があふれていました。
＊6　細かい調整などをしようとすると，なかなか思った通りにはいかないもののようですが。

つまり管理ツールの場合は，想定外の事態に対応するための柔軟性が重要ということになります。そして残念ながら現状のGUIは柔軟性という点に関してはCUI，あるいはCLI（コマンド・ライン・インタフェース）よりはるかに劣ります。これは「グラフィカル」だからだめ，ということではなくて，GUIが初心者にとっての使いやすさ，すなわち「とっつきやすさ」の向上を目指して進化してきたのだから当然と言えます。なぜなら柔軟であればあるほど，「何でもできる」ツールになり，初心者にとっては何をしていいのか分からない「とっつきにくい」ツールになってしまうからです。柔軟性があるツールを求めるならCUIを使わざるを得ない，これが現状です。

また，管理は遠隔地から行わなければならないことが多々あります。インターネット経由，あるいは電話線や携帯電話経由でリモート・ログインして作業することになりますが，十分な帯域が確保できないことの方が普通だと思います。特に携帯電話だと9600bps以下*7という一昔前の水準の帯域であり，GUI管理ツールを利用するのはかなり無理があります。このような現状では，CUIの方が適していると言えるでしょう。

柔軟性

柔軟性の高いツールを設計する最も効率的な方法は，組み合わせて使えるツールをたくさん作ることでしょう。非常に多くの機能を持つツールであっても，他のツールと組み合わせて使えなければ，そのツールが想定している範囲のことにしか適用できません。

一方，自由に組み合わせて使えるツールならば，それぞれのツールが少ない機能しか持っていなくても，組み合わせることによって機能の数は爆発的に増えます。例えば，n_1, n_2, … n_m 通りの機能を持つm個のツールがあったとすると，$n_1 \times n_2 \times \ldots \times n_m$通りの機能が実現できます*8。

例えば，Webサーバーのログを管理する場合を考えましょう。Webクライアント（ブラウザなど）の種別ごとのアクセス数を数えたい，というニーズはよくあるので，大抵の管理ツールにはその機能が備わっているはずです。しかし，専用の管理ツールでなくても，**図1**に示すように3つの小さいツール，sed，sort，uniqを組み合わせることによって，同様のことが可能です。この例だと，携帯電話「P503i」からのアクセスの方が「F503i」からの約3倍であることが分かります。

十分な種類のツールがあれば，事実上あらゆる用途に使うことが可能です。そして実際に十分な種類のツールと，それらを組み合わせる仕掛けを備えたシステムが，Linuxをはじめとする UNIX互換OSです。事実，ツールを組み合わせる仕掛けであるshスクリプトは，プログラミング言語と呼んでしまって差し支えのないほどの柔軟性を兼ね備え，ブート・スクリプト等，UNIXシステムの基幹部分の多くはshスクリプトで記述されています*9。

従って，UNIXのツールを自在に組合わせて，目的とする機能を実現できるようになれば，どんな想定外の事態が発生しても，的確に対応することが可能となるでしょう。

```
% sed 's/.* //' access_log | sort | uniq -c ↵
  22922 "DoCoMo/1.0/F503i"
     31 "DoCoMo/1.0/F503i/c10"
  66798 "DoCoMo/1.0/P503i"
   3392 "DoCoMo/1.0/P503i/c10"
```

図1　ツールを組合わせて新しい機能を実現

*7　cdmaOneの場合は，14.4kbpsです。
*8　組み合わせの順番によって機能が変わる場合は，さらに数が増えます。
*9　最近は実行効率の観点から，perlスクリプトで記述されるケースも増えてきました。

表1　GNU Emacsの基本的なカーソル移動コマンド

キー	機能
C-b	一文字前に戻る
C-f	一文字次に進む
M-b	一単語前に戻る
M-f	一単語次に進む
C-M-b	カッコなどでくくられた部分を一単位として前に戻る
C-M-f	カッコなどでくくられた部分を一単位として次に進む
C-p	前の行に移動
C-n	次の行に移動
C-a	行頭に移動
C-e	行末に移動
M-a	前の文に移動
M-e	次の文に移動
M-{	前の段落に移動
M-}	次の段落に移動
M-v	前の画面に戻る
C-v	次の画面に進む
M-<	ファイルの先頭に移動
M->	ファイルの末尾に移動

コマンド・ライン・インタフェース

UNIXでは，ユーザーが入力したコマンド・ラインを解釈実行するコマンド・ライン・インタプリタと，shスクリプトを解釈実行するスクリプト・インタプリタが同等である＊10ため，shスクリプトの柔軟性がそのままCLIで利用できます。

従って，どんな想定外の事態が発生しても，CLIならば対応できるはずです。ただし，日ごろからCLIを使いこなしていて意のままに操れるようになっていなければなりません。

日常の管理作業はGUI管理ツールで用が足りることも多いのですが，非常時に普段使っていないCLIをいきなり使おうとしても使いこなせるわけはなく，普段からCLIに慣れ親しんでおく必要があると言えるでしょう。

そして，UNIXのCLIは，見かけはとっつきにくいかも知れませんが，使い慣れてくればGUI管理ツールよりよほど使いやすく効率的です。ちょうど，マウス操作が中心で初心者にはとっつきやすいワープロと，慣れれば慣れるほど使いやすくなってくるキーボード操作中心のエディタとの関係に似ています。

例えば，エディタには，カーソル移動だけでも数多くのコマンドがあります。GNU Emacsエディタを例に，基本的なものを**表1**に列挙します。表中「C-b」は，コントロール・キーを押しながら「b」を押すことを表わします。また「M-b」は，メタ・キー＊11を押しながら「b」を押すことを表わします（以下，同様）。

普通のPCには「メタ・キー」は付いていませんが，LinuxなどではAltキーがメタ・キーの代わりに使えるようです。メタ・キーが使えない環境では，「M-b」の代わりに，Escキーを押してから「b」を押すこともできます。しかしメタ・キーは押しっぱなしできるのに対し，Escキーはその都度離さなければならないので，効率は落ちます。なるべくメタ・キーを使えるように工夫するべきでしょう。

例えば，Windows上の代表的な端末エミュレータであるTeraTerm Pro＊12では，「Setup」メニューの「Keyborad...」で「Meta Key」を有効にすると，擬似的にメタ・キーが使えるようになります。すなわちAltキーを押しながら「b」を押すと，Escと「b」が送信されます。さらに，**図2**に示す行をkeyboard.cnf設定ファイルに付加する＊13ことによって，Altキーとコントロール・キーの両方を押す操作方法（C-M-bなど）にも対応できます。

さらに，各コマンドには引数を与えられるので，「C-u 2 0 C-b」などと順にキーを押すと，20文字前にカーソルを移動することができます。

エディタには非常に多くのコマンドがある（GNU Emacsは約3000個）ので，使いこなせれば極めて効率的に編集できます。同様に，CLIにも多彩な機能があります。CLIの機能を使いこなせれば，GUI管理ツールがいかに非効率なものか理解できるのではないでしょうか。

以下，CLIの機能について基本的なものから順に紹介しましょう。

＊10　Linuxディストリビューションの多くは，コマンド・ライン・インタプリタのデフォルトがbashになっています。スクリプト・インタプリタである/bin/shは，/bin/bashの別名（alias）に過ぎないことが多いようです。コマンド・ライン・インタプリタにはbash以外に，csh, tcsh, ksh, zshなども使われますが，いずれも/bin/shと同等の柔軟性を兼ね備えています。
＊11　昔の端末にはメタ・キーがついていて，このキーを押すと最上位ビットを立てたコードを送信できるようになっていました。例えば「a」を押すと0x61がホストへ送られますが，メタ・キーを押しながら「a」を押すと0xE1が送られました。また，コントロール・キーとメタ・キーの両方を押しながら「a」を押すことも当然できて，この場合コントロール・キーと「a」を押したときに送信される0x01の最上位ビットを立てた0x81が送信されました。
＊12　http://hp.vector.co.jp/authors/VA002416/を参照してください。
＊13　[User keys]の設定は，User1から欠番無しで定義しなければなりません。既にいくつかkeyを定義している場合は，番号の付け替えが必要です。詳しくはTeraTerm Proのマニュアルを参照してください。

```
[User keys]
User1=3088,0,$1B$11
User2=3089,0,$1B$17
User3=3090,0,$1B$05
User4=3091,0,$1B$12
User5=3092,0,$1B$14
User6=3093,0,$1B$19
User7=3094,0,$1B$15
User8=3095,0,$1B$09
User9=3096,0,$1B$0F
User10=3097,0,$1B$10
User11=3102,0,$1B$01
User12=3103,0,$1B$13
User13=3104,0,$1B$04
User14=3105,0,$1B$06
User15=3106,0,$1B$07
User16=3107,0,$1B$08
User17=3108,0,$1B$0A
User18=3109,0,$1B$0B
User19=3110,0,$1B$0C
User20=3116,0,$1B$1A
User21=3117,0,$1B$18
User22=3118,0,$1B$03
User23=3119,0,$1B$16
User24=3120,0,$1B$02
User25=3121,0,$1B$0E
User26=3122,0,$1B$0D
```

図2 TeraTerm Proでメタ・キーを擬似的に使うための設定

```
set editing-mode vi
```

図3 編集モードをviに設定

```
"\C-s": backward-char
"\C-d": forward-char
"\C-e": previous-history
"\C-x": next-history
```

図4 ダイアモンド・カーソル操作に設定

```
ipchains -I input -j ACCEPT -p tcp -s 211.120.2.130 -d 210.145.125.162 5901
```

図5 一時的にファイアウオールに穴を開けるコマンド・ライン

```
ipchains -D input -j ACCEPT -p tcp -s 211.120.2.130 -d 210.145.125.162 5901
```

図6 図5で一時的に開けた穴を塞ぐコマンド・ライン

```
asao.gcd.org:/ # ipchains (M-p)
```

図7 tcshにおけるコマンド履歴検索

行編集

長いコマンド・ラインを入力すれば，ミスタイプする確率も増えます。行編集の機能はCLIにとって必要不可欠と言えるでしょう。編集コマンドは普段使っているエディタと同じキーアサインであるべきです。UNIXの場合，Emacsエディタが標準的なので，Emacsのキーアサインを覚えておけば不自由しないのですが，好みに応じてCLIのキーアサインを変更することもできます。

GNU Readlineライブラリを使用しているコマンド・ライン・インタープリタ（bashなど）であれば，~/.inputrcにおいてキーアサインや編集モードを設定できます。例えばviエディタのキーアサインに慣れているのであれば，**図3**のように設定すると良いでしょう。

あるいは編集コマンドごとにキーストロークを設定することもできます。例えば，WordStarライクなカーソル移動キー（ダイアモンド・キー）を使いたい場合は，**図4**のように設定します*14。詳細は，Readlineライブラリかbashなどのマニュアルを参照してください。

コマンド履歴

過去に入力したコマンド・ラインを検索して表示し，再び実行できるようにする機能です。長いコマンド・ラインでも，過去に入力した類似のコマンド・ラインを検索した上で，行編集機能を使って修正すれば，少ないキー入力で実行することが可能となります。

例えば一時的にファイアウオールに穴を開けたい場合を考えます。穴を開けるためのコマンド・ラインは**図5**のようになるでしょう。しばらく作業をした後，図5で開けた穴をふさぐには，**図6**のようなコマンド・ラインを実行することになります。図5と図6では第1パラメータの1文字しか異なりません。

＊14 ただし，「C-s」は端末への送信を一時停止する設定になっていることが多いので，「stty stop undef」などと実行して，この設定を解除しておく必要があります。

```
asao.gcd.org:/ $ (C-r)
(reverse-i-search)`ipchains ': ipchains -I input -j ACCEPT -p tcp -s 211.…
```

図8　bashにおけるコマンド履歴検索

```
"\ep": history-search-backward
"\en": history-search-forward
```

図9　bashにおいてtcshライクなコマンド履歴検索を行うための~/.inputrcの設定

```
asao:/home/sengoku % less /var/log/httpd/access_log
```

図10　ファイル名補完機能を使って打鍵数を削減

```
asao:/home/sengoku % less /var/l  (C-d)
lib/        local/      lock/       log/        lost+found/
```

図11　ファイル名候補の一覧

そこで，tcshの場合ならば，まず「ipchains」の部分だけ入力しておいて「M-p」を押します（**図7**）。すると「ipchains」で始まるコマンド・ラインが検索され，図5のコマンド・ラインが表示されますから，「-I」の部分を「-D」に修正して，リターンキーを押せばOKです。

bashの場合は，まず「C-r」を押します。するとプロンプトの代わりに「(reverse-i-search)‘ ’」と表示されますから「ipchains」と入力すると，**図8**のように「ipchains」で始まるコマンド・ラインが表示されるので，tcshの場合と同様に編集してリターンキーを押します。

なお，bashにおいてもtcshと同様の検索方法を使いたい場合は，~/.inputrc において**図9**のように設定します。この辺は好みの問題ですが，私は最初に「C-r」を押すbash方式より，まず「ipchains」などとコマンド・ラインを途中まで入力しておいて「M-p」を押すtcsh方式の方が好きです。

補完

コマンド・ラインが長くなる原因の一つは長いファイル名ですが，長いファイル名を入力する手間を省く機能が，ファイル名補完*15です。例えばWebサーバーのログを見ようとして**図10**のようなコマンド・ラインを入力する際，下線を引いた部分の入力だけで済みます。すなわち，「less /v」と入力した段階で Tabキーを押すと，補完が行われて「less /var/」まで表示されます。

次は「log」とディレクトリ名をすべて入力していますが，これは「/var/」ディレクトリの下に「lib」ディレクトリが存在するので，「/var/l」だけでは「/var/log」と「/var/lib」のどちらであるか一意に決まらないため補完が行われないからです。さらに「/var/」には「lock」ディレクトリも存在するため，「/var/lo」まで入力しても一意に決まりません。結局「log」に関しては3文字とも入力する必要があります。

一方，「/var/log/」には，「h」で始まるファイル/ディレクトリが「httpd」しか無いため，「/var/log/h」の段階でTabキーを押すだけで「/var/log/httpd/」まで表示されます。このように同じディレクトリには先頭の1～2文字で互いに区別できるようなファイル/ディレクトリ名を付けておくと，補完が有効に働きます。

なお，ファイル名を途中まで入力した段階でTabキーの代りに「C-d」を押す（tcshの場合。bashの場合は「M-?」を押す）と，**図11**に示すようにファイル/ディレクトリ名候補の一覧が表示されます。

Emacs

Emacsにはshellモードがあり，エディ

*15 「補間」ではありません。

タの中でCLIを使うことができます。エディタの中なので,「行」編集だけでなく,コマンド・ラインおよびコマンドの出力すべてを含めた,あらゆる編集が可能となるので大変便利です(**写真1**)。

EmacsでshellモードをEmacsを使うには,「M-x shell」(「M-x」を押した後「shell」を入力してリターンキーを押す。以下同様)を入力します。すると,shell用のバッファ(編集中のテキストを格納するための領域)が作られ,コマンド・ライン・インタプリタが実行されます。

shellモードでは,リターンキーが特別な機能を持っていて,

(1) カーソルがバッファの最下行にあるときは,最下行に入力されているコマンド・ラインが,コマンド・ライン・インタプリタへ入力され,その出力がバッファに追加されていきます。

(2) カーソルがバッファの最下行にないときは,カーソルのある行のコマンド・ラインが最下行へコピーされた上で,(1)が実行されます。

このほかにもshellモード独自の機能を持っているキーがいくつかあり,主要なものを**表2**にまとめておきます。

パスワードを入力する必要があるコマンドの実行時は,**写真2**のように自動的にパスワード入力モードに切り替わります。これは単に「Password:」というプロンプトを見て判断しているだけなので,コマンドによってはパスワード入力モードに切り替わらず,入力中のパスワードがそのまま画面に表示されてしまうことがあります。その場合は,パスワードを入力する前に「M-x send-invisible」と入力すれば,OKです。

なお,comint-password-prompt-regexp変数に,パスワード入力を求めるプロンプトにマッチする正規表現が設定されているので,パスワード入力モードに切り替わらないプロンプトがある場合は,この変数を適宜修正すると良いでしょう。

表2 shellモード独自の機能を持つキー

キー	機能
リターン	コマンドの実行
Tab	ファイル名補完
M-?	ファイル名候補一覧
C-c C-c	実行中のコマンドを中断する(SIGINTシグナルを送る)
C-c C-z	実行中のコマンドをサスペンドする(SIGTSTPシグナルを送る)
C-c C-u	入力中のコマンド・ラインを消去する

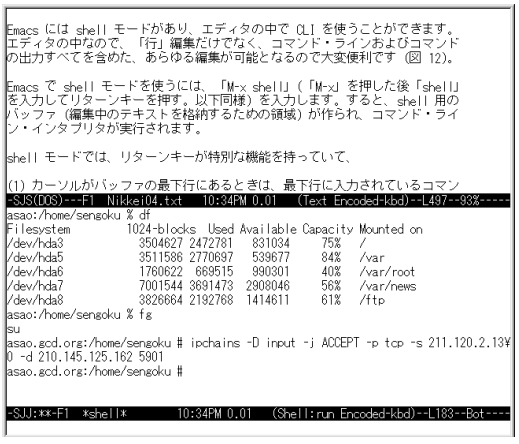

写真1 Emacsエディタの中でCLIを使う

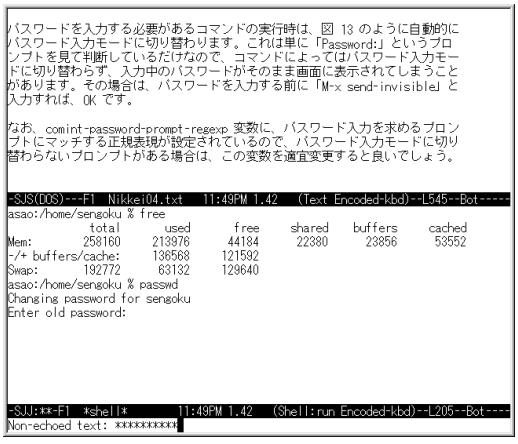

写真2 パスワード(Non-echoed text)入力モード