Special Feature 01
シェルの本質を理解する

Special Feature 02
エンジニアは日々勉強だ!

Software Design

February 02 2013

[ソフトウェア デザイン]
OSとネットワーク、
IT環境を支える
エンジニアの総合誌

2013年2月18日発行
毎月1回18日発行
通巻334号
(発刊268号)
1985年7月1日
第三種郵便物認可
ISSN 0916-6297
定価
1,280円

シェルスクリプティング道場

Special Feature 01
UNIXコマンド、fork、pipeを復習し、高度なスクリプティングへ

Special Feature 02
忙しいITエンジニアのための
超効率的勉強法

Extra Feature
Samba 4.0.0ファーストインプレッション

この個所は、雑誌発行時には広告が掲載されていました。編集の都合上、
総集編では収録致しません。

第50回

IT エンジニア必須の 最新用語解説

TEXT：㈲オングス 杉山 貴章 SUGIYAMA Takaaki
takaaki@ongs.co.jp

テーマ募集
本連載では、最近気になる用語など、今後取り上げてほしいテーマを募集しています。sd@gihyo.co.jp 宛にお送りください。

Firefox OS

「Firefox OS」のコンセプト

iOS や Android、Windows Phone など、各社・各団体からさまざまな OS が出そろってきたスマートフォン市場ですが、そんな中で Web ブラウザ「Firefox」の提供元である Mozilla Foundation によってオープンソースで開発されているのが「Firefox OS」（初期のコードネームは「Boot to Gecko（B2G）」）です。「Firefox」の名前が使われていることからも想像できるように、Web 技術を最大限に活用できることが大きな強みとして挙げられています。

Firefox OS は、新しい Web 標準を活用したオープンなモバイルプラットフォームを実現することを主な目的として開発されています。基本的なコンセプトは、端末上でユーザが使用できるソフトウェアはすべて Web アプリであり、HTML 5 や JavaScript によって開発することができるというもの。Web アプリと言っても、基盤 OS そのものが Web アプリの実行を前提に設計されており、中間レイヤの存在を最小限に抑えることで、極めて効率の良い動作を実現します。

また、通話機能や各種センサー、カメラなどといった端末固有の機能に対して、プラットフォーム固有の言語や API を使うことなくアクセスすることが可能なことなど、Web 標準技術だけでネイティブアプリ開発が行えるという点が極めて大きな強みと言えます。

主要なアーキテクチャ

Firefox OS は、大きく分けると「Gonk」、「Gecko」、「Gaia」と呼ばれる 3 つのソフトウェアレイヤから構成されます。

Gonk は、Linux ベースのカーネルと、Gecko と通信するハードウェア抽象化レイヤから構成される低レベルのオペレーティングシステムレイヤです。カーネルといくつかのユーザスペースライブラリは、オープンソースの Linux や libusb、bluez などであり、HAL のその他のパーツは Android プロジェクトと共有しているとのこと。Gonk は非常にシンプルな Linux ディストリビューションの一種と言えるため、Gecko からは他の既存の OS と同様に扱うことができます。

Gecko は、Firefox や Thunderbird で使われているのと同じ Web 標準仕様の実装をベースとしたアプリケーションランタイムです。HTML や CSS、XUL によるレンダリングや JavaScript の実行をサポートします。

Gaia は、Firefox OS 端末用 UI アプリケーションで、HTML/CSS/JavaScript で記述された多数の UI ツールを提供します。Firefox OS では、ロック画面やホーム画面、電話など、起動後に用意されているアプリケーションはすべて Gaia によって描画されるとのことです。OS とハードウェアに対するインターフェース以外は HTML/CSS/JavaScript で実装されているため、Firefox OS 以外の OS や Web ブラウザ上でも実行できる点が大きな特徴になっています。

最初のリリースは2013年!?

本稿執筆時点で Firefox OS を利用するには、開発者向けサイトからソースコードを入手して自前でビルドする必要があります。テスト開発用プラットフォームに加えて、ARM および x86 デスクトップ向けエミュレータ、Nexus S などの一部の実機での動作がサポートされています。その他に、この OS のデスクトップ版となる「Firefox OS デスクトップクライアント」や、Firefox ブラウザ上で Firefox OS アプリを動作させることが可能な「Firefox OS シミュレータ」などによって、デスクトップ PC 上で Firefox OS を体験することもできるようになっています。

製品化された最初の端末は 2013 年第一四半期のリリースを目指しているとのことで、当面は発展途上国向けのローエンドスマートフォンをターゲットとして展開していくそうです。現実的には、シェア争いという観点で言えば先行きは不透明な部分が大きいプロジェクトですが、技術的なチャレンジとしては極めて興味深い存在と言えるでしょう。SD

Firefox OS
http://www.mozilla.jp/firefoxos/
開発者向け情報サイト
https://developer.mozilla.org/ja/docs/Mozilla/Firefox_OS

この個所は、雑誌発行時には広告が掲載されていました。編集の都合上、
総集編では収録致しません。

Software Design Contents

UNIXコマンド、fork、pipeを復習し、高度なスクリプティングへ

シェルスクリプティング道場

この個所は、雑誌発行時には広告が掲載されていました。編集の都合上、
総集編では収録致しません。

Software Design Contents

第2特集 Special Feature 02

忙しいITエンジニアのための 超効率的勉強法

森川 滋之 | 067

広告索引 AD INDEX

Part 02 Contents

この個所は、雑誌発行時には広告が掲載されていました。編集の都合上、
総集編では収録致しません。

［目次］

Software Design February 2013［CONTENTS］#03

Software Design Contents

Column

Development

OS/Network

Inside View

Part 00 Contents

Logo Design ロゴデザイン ＞ デザイン集合ゼブラ+坂井 哲也
Cover Design 表紙デザイン ＞ Re:D
Cover Photo 表紙写真 ＞ Imagewerks/Getty Images
Page Design 本文デザイン ＞ 岩井 栄子、近藤 しのぶ、SeaGrape、安達 恵美子
［トップスタジオデザイン室］轟木 亜紀子、阿保 裕美、佐藤 みどり
［BUCH+］伊勢 歩、横山 慎昌
森井 一三、Re:D、［マップス］石田 昌治

この個所は、雑誌発行時には記事が掲載されていました。編集の都合上、
総集編では収録致しません。

この個所は、雑誌発行時には記事が掲載されていました。編集の都合上、
総集編では収録致しません。

この個所は、雑誌発行時には記事が掲載されていました。編集の都合上、
総集編では収録致しません。

この個所は、雑誌発行時には記事が掲載されていました。編集の都合上、
総集編では収録致しません。

Software Design plus

Software Design plusシリーズは、OSとネットワーク、IT環境を支えるエンジニアの総合誌『Software Design』編集部が自信を持ってお届けする書籍シリーズです。

サーバ／インフラエンジニア養成読本　仮想化活用編
Software Design編集部 編
定価 1,980円+税　ISBN 978-4-7741-5038-3

サーバ／インフラエンジニア養成読本　管理／監視編
Software Design編集部 編
定価 1,980円+税　ISBN 978-4-7741-5037-6

もっと自在にサーバを使い倒す業務に役立つPerl
木本 裕紀 著
定価 2,780円+税　ISBN 978-4-7741-5025-3

サーバ構築の実際がわかるApache[実践]運用／管理
鶴長 鎮一 著
定価 2,980円+税　ISBN 978-4-7741-5036-9

Webエンジニアのためのデータベース技術[実践]入門
松信 嘉範 著
定価 2,580円+税　ISBN 978-4-7741-5020-8

2週間でできる! スクリプト言語の作り方
千葉 滋 著
定価 2,580円+税　ISBN 978-4-7741-4974-5

PCのウイルスを根こそぎ削除する方法
本城 信輔 著
定価 1,980円+税　ISBN 978-4-7741-4867-0

Androidエンジニア養成読本
Software Design編集部 編
定価 1,880円+税　ISBN 978-4-7741-4859-5

Vyatta入門 実践ルーティングから仮想化まで
近藤 邦昭、松本 直人、浅間 正和、大久保 修一（日本Vyattaユーザー会）著
定価 3,200円+税　ISBN 978-4-7741-4711-6

プロのためのLinuxシステム・ネットワーク管理技術
中井 悦司 著
定価 2,880円+税　ISBN 978-4-7741-4675-1

サーバ／インフラエンジニア養成読本
Software Design編集部 編
定価 1,880円+税　ISBN 978-4-7741-4600-3

Linuxエンジニア養成読本
Software Design編集部 編
定価 1,880円+税　ISBN 978-4-7741-4601-0

Nagios統合監視[実践]リファレンス
㈱エクストランス　佐藤 省吾、Team-Nagios 著
定価 3,200円+税　ISBN 978-4-7741-4582-2

最新刊！

高宮 安仁、鈴木 一哉 著
A5判・336ページ
定価 3,200円（本体）+税
ISBN 978-4-7741-5465-7

最新刊！

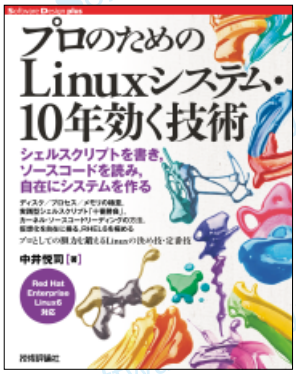
青柳 隆宏 著
A5判・448ページ
定価 3,200円（本体）+税
ISBN 978-4-7741-5464-0

和田 裕介 著
A5判・208ページ
定価 2,180円（本体）+税
ISBN 978-4-7741-5407-7

河村 嘉之、川尻 剛 著
B5変形判・480ページ
定価 2,980円（本体）+税
ISBN 978-4-7741-5438-1

㈱マピオン、山岸 靖典、谷内 栄樹、本城 博昭、長谷川 行雄、中村 和也、松浦 慎平、佐藤 亜矢子 著
B5変形判・256ページ
定価 2,580円（本体）+税
ISBN 978-4-7741-5325-4

三苫 健太 著
B5判・400ページ
定価 3,200円（本体）+税
ISBN 978-4-7741-5189-2



中井 悦司 著
B5変形判・352ページ
定価 3,400円（本体）+税
ISBN 978-4-7741-5143-4

木村 明治 著
B5変形判・416ページ
定価 3,180円（本体）+税
ISBN 978-4-7741-5026-0

技術評論社　162-0846 東京都新宿区市谷左内町21-13
販売促進部　TEL 03-3513-6150 FAX 03-3513-6151

スマートフォンアプリ開発者とデザイナのための総合情報誌

Smartphone Design

Software Design 編集部　編
B5 判　168 ページ　定価 1,659 円（本体 1,580 円＋税）
ISBN 978-4-7741-5335-3

ユーザから支持されるスマートフォンアプリ開発のために「デザインする力」がより一層求められている開発現場。悩めるエンジニアとデザイナがともに力を発揮するためのヒントが満載です！

第1特集　どうしてデザインと開発は両立できないのか？
第2特集　スマホ開発者がUnityを理解しておくべき理由
その他　Web＋ネイティブでスピード・コスト・メンテに効くアプリ開発／Windows Phoneアプリ開発入門／PlayStation Mobileアプリ開発／仮想化技術でスマホ＆タブレットを業務に／既存のPCサイトをスマホ用に変換！／Webブラウザでクロス開発できるappMobi　ほか

現場で使える [逆引き+実践] Androidプログラミングテクニック

石原正樹、松尾源、磯村禎孝、森靖晃、奥谷修治　著
A5 判　464 ページ　定価 2,919 円（本体 2,780 円＋税）
ISBN 978-4-7741-5187-8

普及が進むスマートフォンで注目されるAndroid OSですが、組込みシステムの宿命とも言うべき「リソースの制限、バッテリー駆動」といった、プログラミングに関わる制限事項が多数存在します。また、「新しい情報をリアルタイムで追随しにくい」といった問題、さらには「従来型のC/C++の組込み開発をしてきた人や会社はJavaに不慣れ」「Javaに慣れた人や会社は低レベルの理解が足りず参入に苦労」といったノウハウ不足の問題に対し、本書は徹底的にチューニングの方法を解説することでも寄与します。

iPhoneアプリ開発塾

iOS 5.1 & Xcode 4.3 対応

カワサキタカシ　著
B5 変形判　320 ページ　定価 2,919 円（本体 2,780 円＋税）
ISBN 978-4-7741-5105-2

iPhoneアプリの作り方について書かれた本は数多くありますが、「基本はわかっても応用がきかない」「やっぱりiOSプログラミングは難しい」といった声が多いようです。本書は、そんな悩めるiPhoneアプリ開発エンジニア達に人気のポータルサイト『サルでき.jp』（旧ブログ：サルにもできるiPhoneアプリの作り方）の管理人が、Xcodeの読み方、プログラミングの基本はもちろん、サポートページの作り方までを、“どこよりも敷居の低い”書き方で丁寧に解説しています。

技術評論社　当社書籍・雑誌のご購入は全国の書店、またはオンライン書店でお願い致します。
162-0846　東京都新宿区市谷左内町 21-13　販売促進部　TEL.03-3513-6150　FAX.03-3513-6151

Pocket Reference

小さくても、中身充実！

「あれ何だったっけ？」
「こんなことできないかな？」
というときに、すぐに調べられる
機能引きリファレンス。
軽くてハンディなボディに
密度の濃い内容がギューッと凝縮！
関連項目への参照ページもあって、
検索性もバツグン！

岡本隆史、武田健太郎、相良幸範　著
四六判／272ページ
定価2,604円（2,480円＋税）
ISBN978-4-7741-5184-7

片渕彼富　著、山田祥寛　監修
四六判／448ページ
定価2,709円（2,580円+税）
ISBN 978-4-7741-5067-3

山森丈範　著
四六判／304ページ
定価2,499円（2,380円+税）
ISBN 978-4-7741-4396-5

石田つばさ　著
四六判／448ページ
定価2,289円（2,180円+税）
ISBN 978-4-7741-4836-6

沓名亮典，平山智恵　著
四六判／576ページ
定価2,394円（2,280円+税）
ISBN 978-4-7741-3816-9

細島一司　著
四六判／464ページ
定価2,919円（2,780円+税）
ISBN 978-4-7741-5135-9

土井毅、高江賢、飯島聡、高尾哲朗 著　山田祥寛 監修
四六判／488ページ
定価2,709円（2,580円+税）
ISBN 978-4-7741-4948-6

朝井淳　著
四六判／640ページ
定価2,079円（1,980円+税）
ISBN 978-4-7741-3835-0

山田祥寛　著
四六判／528ページ
定価2,919円（2,780円+税）
ISBN 978-4-7741-4980-6

森直彦　著
四六判／392ページ
定価2,709円（2,580円+税）
ISBN 978-4-7741-4982-0

技術評論社
当社書籍・雑誌のご購入は全国の書店、またはオンライン書店でお願い致します。
162-0846　東京都新宿区市谷左内町21-13　販売促進部　TEL.03-3513-6150　FAX.03-3513-6151

この時代を**生き抜く自信**はありますか？

生きる技術！叢書

危機の時代に
「生きる技術」
を伝える
新しい教養書

ART OF LIVING

最新刊！

内田樹　著
定価：本体 1,580 円＋税
ISBN 978-4-7741-4709-3

パオロ・マッツァリーノ　著
定価：本体 1,480 円＋税
ISBN 978-4-7741-5385-8

藤井聡　著
定価：本体 1,580 円＋税
ISBN 978-4-7741-5023-9

小田嶋隆　著
定価：本体 1,480 円＋税
ISBN 978-4-7741-5316-2

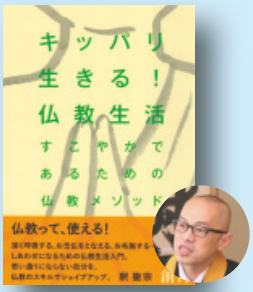
小田嶋隆　著
定価：本体 1,480 円＋税
ISBN 978-4-7741-4870-0

香山リカ　著
定価：本体 1,380 円＋税
ISBN 978-4-7741-5066-6

慎泰俊　著
定価：本体 1,480 円＋税
ISBN 978-4-7741-5152-6

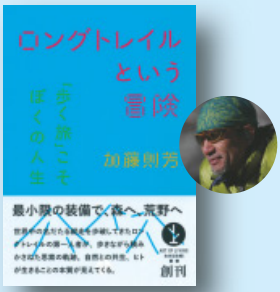
小林弘人　著
定価：本体 1,480 円＋税
ISBN 978-4-7741-4935-6

岩田健太郎　著
定価：本体 1,580 円＋税
ISBN 978-4-7741-4837-3



釈徹宗　著
定価：本体 1,580 円＋税
ISBN 978-4-7741-4708-6

林志行　著
定価：本体 1,580 円＋税
ISBN 978-4-7741-4764-2



加藤則芳　著
定価：本体 1,580 円＋税
ISBN 978-4-7741-4768-0

小川仁志　著
定価：本体 1,580 円＋税
ISBN 978-4-7741-4707-9

鷲田清一、内田樹、
上杉隆、岩田健太郎、
藏本一也　著
定価：本体 1,280 円＋税
ISBN 978-4-7741-4871-7

いま、生きるための教養を！

技術評論社
当社書籍・雑誌のご購入は全国の書店、またはオンライン書店でお願い致します。
162-0846　東京都新宿区市谷左内町 21-13　販売促進部　TEL.03-3513-6150　FAX.03-3513-6151

紙面版
A4判・16頁
オールカラー

電脳会議

DENNOUKAIGI

一切無料

新規購読会員受付中!

『電脳会議』は情報の宝庫、世の中の動きに遅れるな!

『電脳会議』は、年6回の不定期刊行情報誌です。A4判・16頁オールカラーで、弊社発行の新刊・近刊書籍・雑誌を紹介しています。この『電脳会議』の特徴は、単なる本の紹介だけでなく、著者と編集者が協力し、その本の重点や狙いをわかりやすく説明していることです。平成17年に「通巻100号」を超え、現在200号に迫っている、出版界で評判の情報誌です。

今が旬の情報満載!

新規送付のお申し込みは…

Web検索か当社ホームページをご利用ください。
Google、Yahoo!での検索は、

電脳会議事務局　検索

当社ホームページからの場合は、
https://gihyo.jp/site/inquiry/dennou
と入力してください。

一切無料!

◉「電脳会議」紙面版の送付は送料含め費用は一切無料です。そのため、購読者と電脳会議事務局との間には、権利&義務関係は一切生じませんので、予めご了承下さい。

毎号、厳選ブックガイドも付いてくる!!

『電脳会議』とは別に、1テーマごとにセレクトした優良図書を紹介するブックカタログ（A4判・4頁オールカラー）が2点同封されます。扱われるテーマも、自然科学／ビジネス／起業／モバイル／素材集などなど、弊社書籍を購入する際に役立ちます。

株式会社 技術評論社　電脳会議事務局　〒162-0846 東京都新宿区市谷左内町21-13

この個所は、雑誌発行時には広告が掲載されていました。編集の都合上、
総集編では収録致しません。

この個所は、雑誌発行時には記事が掲載されていました。編集の都合上、
総集編では収録致しません。

この個所は、雑誌発行時には記事が掲載されていました。編集の都合上、
総集編では収録致しません。

この個所は、雑誌発行時には記事が掲載されていました。編集の都合上、
総集編では収録致しません。

DIGITAL GADGET

Volume 170

家電のデジタル化≠多機能化

多機能化するデジタル機器へのアンチテーゼ

昨今、1つのデジタル機器にたくさんの機能が詰め込まれるのはごくごく一般的になりました。製品ごとにスペックを比較し、多機能であればあるほど価値があるように思えてしまうのではないでしょうか?

その一方で、ある特定の用途で必要な機能を、それだけ分離させることで使いやすくしたデバイスも出現してきました。また、デバイスに関係なく、各種デバイスをまたがった形でコンテンツやデータを扱えるサービスも増えてきました。これらの登場は、機能そのものではなく、その機能がどう使われるのか、その機能でどういった体験が得られるのかが重要になってきたことを物語っています。

家電製品やデジタルデバイスのコントロールに気軽に使える「リモコン」も、機能を分離したものの1つと考えられるでしょう。特定の機能または、さまざまな操作を、本体ではなく、分離した機器で行えるようにしたものです。

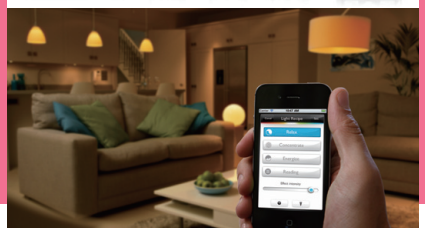
今回は、デバイスの機能として本来一緒だったものが、分かれて機能したり、スマートフォンと連携して動作するグッズをいくつか紹介していきます。

gadget 1

hue

http://www.meethue.com/en-US

電球色をコントロール



Philipsのhueは、スマートフォンやタブレットのアプリから、カラフルな色や明るさを自在に操作できるLED電球です。専用のWi-Fiアクセスポイントと電球が3個ついたスターターキットがあり、あとから電球を追加できます。1台のiOSデバイスで最大50個のhue電球を管理可能です。そのときの雰囲気や生活スタイル、時間にあわせて色と明るさを細かく調整できます。リモコンを画面付きデバイスにすることで、画面のイメージを現実世界に投影する楽しさと操作しやすさを両立させた好例です。同様の機器として、LIFX(http://tech.lifx.co)やBluetooth Bulb(http://www.bluetoothbulb.com)、insteon Light Bulb(http://www.insteon.net/bulb.html)があります。電球や電灯のスイッチ部分を、スマートフォンに分離した機器です。電球をコントロールするバルブ部分を開発し、どんな電球でもコントロールしようとするSparkという製品開発のプロジェクトも進行中です(http://www.kickstarter.com/projects/sparkdevices/spark-upgrade-your-lights-with-wi-fi-and-apps)。

安藤 幸央 —— Yukio Ando ——
EXA CORPORATION
[Twitter] >> @yukio_andoh
[Web Site] >> http://www.andoh.org/

gadget 2

Nest

http://www.nest.com/

室温コントロール

Nestはインテリジェントな機能をもった温度調節装置で、ユーザの生活パターンを学習して温度設定を行います。スマートフォンやインターネット経由での設定も可能です。省エネのための工夫もなされています。米国家庭に設置されているような冷暖房のサーモスタットを置き換える形の工事／配線が必要なため、法令などの関係もあり、残念ながら現状は日本の冷暖房器具には使えないようです。冷暖房機器のコントロール部分と温度センサー部分が分離し、進化したデバイスといえるでしょう。

gadget 4

Zik

http://www.parrot.com/zik/jp/

音楽コントロールヘッドフォン

Zikは、iPhoneでコントロールできるラジコンヘリコプターで知られるParrot社の製品です。高性能なノイズキャンセリングBluetoothヘッドフォンであるとともに、耳当ての外殻がタッチパネルになっており、ジェスチャでiPhoneの音楽アプリのコントロールが可能です。さらに専用アプリを利用すると、ヘッドフォンなのに音像が前方から聞こえてくる音響効果やイコライザによる調節などができます。iPhoneから音楽コントロール部分のみ分離し、機能特化したヘッドフォンといえるでしょう。

gadget 6

EnergyHub

http://www.energyhub.com/

家庭用エネルギー管理システム

EnergyHubは家庭用のエネルギー統合監視システムです。家電製品をEnergyHub専用のコンセントにつないで利用すれば、電気消費量を統合的に管理することができる機器です。計測用のコンセントはZigBee無線タイプのものもあり、このシステムのために配線を増やさなくてすみます。Webサイトやモバイルアプリとの連携も進んでおり、家の電力コントロールを、コンセントの呪縛から離れてネットから監視・コントロールできるシステムです。

gadget 3

popSLATE

http://www.popslate.com

E Inkケース

popSLATEはE Ink搭載のiPhone 5専用ケースで、iPhoneに表示した画面をその背面側にあるE Ink画面（モノクロ）に表示させておくことができます。現在はコンセプトモデルの段階で、クラウドファンディングによって開発資金を募集中です。一度表示させてしまえば、電力消費なしで絵を映し出しておけるE Inkの特性をうまく活用したケースです。バッテリー消費を気にすることなくもう1画面手に入れたようなもので、地図や買い物メモなどで便利に活用できそうです。

gadget 5

iSiri

http://www.ciccaresedesign.com/2012/11/08/isiri/

Siri専用ボタン

iSiriは残念ながらコンセプトのみで、どうやら実現しそうにないデバイスですが、iPhoneの音声認識コントロール部分のみを分離したものといえるでしょう。ボタン形状の本体にはマイク、スピーカー、イヤフォンジャックを備え、iPhone本体とはBluetoothでつながり、腕時計のように装着できるウェアラブルな音声コントローラとして利用することが想定されています。スマートフォンの利用には緻密な画面とタッチパネルの存在が当然と考えがちですが、Siriのみであれば、実は画面は必要ありません。今あるスマートフォンで利用している機能の、ある一部分のみ分離して使えたら便利になる良い例です。

(C)Federico Ciccarese - ciccaresedesign.com

gadget 7

All Sports GPS

http://www.satsportsgps.com/satski/satsportgps.html

スポーツ専用GPSデバイス

All Sports GPSは、スマートフォンのGPS機能部分を分離させ、スポーツ利用に特化したデジタルデバイスです。GPSとはいいつつ、中身はAndroid端末で、距離計測や記録により適した形でオンライン地図を活用することができます。専用のアプリストアAll Sports App Marketには位置情報系、スポーツ関連のアプリが数多く取りそろえられています。なんでもできるスマートフォンは便利ですが、過酷な環境でも使える丈夫なハードウェアと、特定の状況下での使いやすさを考慮したアプリでなければ実用にならないシーンもあります。

gadget 8

COCOROBO RX-V100

http://www.sharp.co.jp/cocorobo/

家電制御できる掃除ロボット

床を掃除してくれるお掃除ロボットCOCOROBOの上位機種RX-V100に「家電コントローラー」というオプションを搭載すると、赤外線通信を使って、エアコンやテレビ、照明など赤外線リモコン対応の機器をコントロールできるようになるそうです。RX-V100にはカメラも搭載されており、ペットや部屋の様子をスマートフォンで確認したりもできます。家電とロボットとスマートフォンがリンクした例です。

gadget 9

CrossFeel

http://www.nelt.co.jp/products/led_sp.html

Bluetooth内蔵スピーカー付きシーリングライト

CrossFeelはBluetoothを経由してスマートフォンでコントロールできる天井に設置されたシーリングライトです。先に紹介したスマートフォンコントロールLEDランプとの大きな違いは、スピーカーも設置されていることです。音楽を聴く用途にはもちろん、鳥のさえずりや、小川のせせらぎといった、環境音を流すといった用途にも使えます。消灯後にもすぐに真っ暗にならず、2〜3分間、淡いブルーグリーンの光で照らしてくれる生活に即したライティングも考えられています。住環境の光と音をスマートフォンと連携させた例です。

gadget 10

Siri Eyes Freeモード

https://www.onstar.com/web/Bluetooth

iPhoneを車で活用

iOS 6から搭載されたSiri Eyes Freeモードは、運転中のドライバーが音声でiPhoneをコントロールできるしくみです。利用には車載機器プラットフォーム仕様であるBluetooth MyLinkで接続し、ハンドル横についている音声入力ボタンを押してSiriを起動するそうです。車用にiPhoneのSiriボタンを分離させて利用できるようにした事例です。

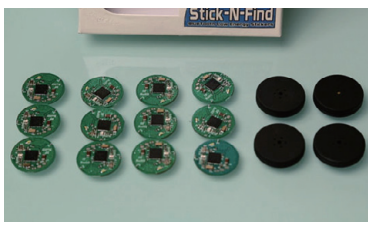
gadget 11

StickNFind

http://www.indiegogo.com/sticknfind

スマートフォン捜索シール

StickNFindは部屋の中でよく行方不明になるもの、たとえばリモコンや財布、薄くて本の間にまぎれてしまうようなものなどに貼っておくと、スマートフォンでありかを発見することができるコインくらいの大きさと厚さのシールです。Bluetooth技術を利用し、電波強度からスマートフォン上でレーダーのように探し出すことができます。いざというときに見あたらない各種リモコンも、このStickNFindのようなデバイスがあれば安心してどこにでも置いておくことができますね。



今、皆さんの身近にある、携帯電話やスマートフォンの機能をあらためて認識してみてください。さまざまなアプリがインストールされ、人それぞれ、多様な機能を持っていると思います。タッチパネルの画面をスワイプし、カテゴリアイコンを開き、アプリを起動するといった手間を何度も経験していると、たびたび、ある機能だけを抜き出して便利に使いたくなるシチュエーションはないでしょうか？　何かを高機能にするだけでなく、何かの機能を分離してより便利にする手はないでしょうか？

目的ややり方が決まっている場合は、ある特定のことしかできない道具のほうが確実に扱えるでしょう。スイスアーミーナイフのように、さまざまな機能を持っていて便利に使えても、それぞれ単機能としては専用器具にはかないません。経験を積んだ宮大工や家具職人は、それぞれ細かな用途向けの、ものすごい数の鉋（カンナ）を持っているそうです。

これからのデジタルデバイスは、なんでもできるスイスアーミーナイフとしての進化の一方で、職人がその技を最大限に発揮できる、特定の専用機器も重宝されてくるのかもしれません。

現在一体化していて当然と思えるものも、分離して使えることを想像すると、新しい可能性が広がります。すでに身のまわりには分離して使うことが一般的なものも数多くあることに気づくはずです。ワイヤレス受話器、ノートパソコンに外付けキーボード、エアコンのリモコン、車のワイヤレスキー……。

今一度、まわりを見渡してみてはいかがでしょうか？　なにか新しい発見があるかもしれません。SD

Can we still stay hungry?

小飼弾の #32 コードなエッセイ

TEXT= 小飼 弾 KOGAI Dan dankogai@dan.co.jp

ハングリーであり続けろ

本連載も、来月で終わる。本連載を引き継ぐ次の連載が今から楽しみなのだが、それはさておき最終回は二度にわけてお届けすることにしよう。キーワードとなるのは、この言葉。

> Stay hungry, stay foolish.

Jobsが引用したことで[注0]今や英語の教科書にも載るほど有名になった言葉だが、引用元の「The Whole Earth Catalog」[注1]がこの言葉を掲載した1972年と比べればもちろんのこと、Jobsの2005年と比較してさえ、ハングリーでいるのは格段に難しくなっている。

そのハングリーさにおいて、コンピュータ産業ほどハングリーだった業界はないだろう。パソコンという言葉が一般化するきっかけとなったApple][に搭載されていたRAMは、ベースモデルで4KB、最高で48KB。2012年のパソコンは2GB～64GBくらいなので、それから100万倍になったことになる。2を底とする対数でみると、20ほど。つまり倍々ゲームをおよそ20回ほど繰り返してきたことになる。

なぜそうだったか。我々ユーザがつねにメモリに対して飢餓状態にあったからだ。「これだけあれば足りる」と思って買っても、翌年には足りなくなっていた。進化したハードウェアに合わせたソフトウェアが出続けることによって。文字だけではなく画像も扱いたい。単に画像が扱えるだけでなくそれを用いたユーザインターフェースがほしい。白黒ではなくカラーがほしい。画面もずっと広くしてほしい。そして静止画像だけではなく、動画も扱えるようになってほしい……。

これらの願いは、ほぼすべてかなえられた。いや、かなえられてしまったというべきか。今やKindleやKoboなどのE-Book Readerを除けば、「カラー」というのはパソコンどころか携帯電話でも24ビットフルカラーが当然で、解像度も最低でXGA（1024×768ピクセル）あるのが当然となっている。動画も1080pフルハイビジョンを難なく扱える。

ポストハングリー時代の到来

こうなってくると、いよいよ人間のほうが「おなかいっぱい」になってくる。これ以上色を増やしても、解像度を高くしても、フレームレートを上げても、もはや人の目には違いがわからないところまできてしまったのだ。私が今これを書いているiMacは1世代前のものであるが、搭載している資源は余りまくっている。メモリ1つとっても、32GBというのは、次ページのスクリーンショットのとおり、その中でOSを5つくらい動かしても余裕で余って

注0）「Steve Jobs' 2005 Stanford Commencement Address」
URL https://www.youtube.com/watch?v=UF8uR6Z6KLc
注1）米国で1968年～1972年に刊行されていたカウンターカルチャーの雑誌。

しまう量なのだ。Windows 8が登場したにもかかわらず、全世界的にPCが売れなくなってきたというのも無理はない。今あるもので十二分に間に合ってしまうのだから。

そう考えると、Jobsのスピーチは見事な引っかけのように思える。2005年にはまだPCにも「飢餓」が残っていたけれど、それが数年のうちに尽きることを彼は知っていたのだから。彼のすごかったところは、それを単に予言するにとどまらず、次にユーザが何に飢えるかをも予測し、それにどんぴしゃりの答えを用意したこと。PCの重厚長大さに満腹したユーザたちは、iPhoneとiPadをむさぼるように使い出した。

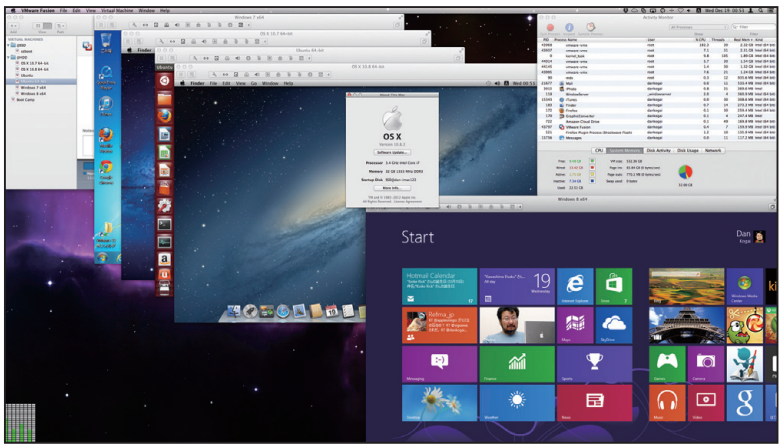
そのポストPCデバイスも、しばらくはPCと同じような倍々ゲームが続いている。シングルコアがデュアルコア、512MBが1GB、3GがLTEに……。しかしその「空腹」の余地は、PCよりずっと早くなくなろうとしている。私のGalaxy S3には2GBのRAMが入っているが、これは本連載が始まったころのネットブックより大きいのだ。4GBの後はいよいよPCと同様に64ビットCPUが必要になるし、ポストPC時代のIntelアーキテクチャの代表であるARMも64ビットコアを発表してはいるけど、スマートフォンやタブレットにそこまで必要か、大いに疑問でもある。実際iOSはAndroidの半分程度のコアとメモリで市場で十二分に競争していることを鑑みれば、人々が飢えているのはそこではないことは賢明な読者のみなさんはすでにお気づきのはずだ。

それでは、ポストハングリー時代、我々は何を売るべきだろうか。2つの路線があり得る。

1つは「まだハングリーな分野」を求めること。「こちら側」のクライアントはとにかく、「あちら側」のサーバの世界は飢餓状態がいまだに続いている。本誌の記事も「こちら側」に属するものより「あちら側」に属するものが多いこともそれを反映していると言える。

そしてもう1つは、ハングリーに代わるのは何かを模索すること。それは何か？ ペインフルだと私は考えている。腹が満ちても、痛みは収まらない。たとえば扱うデータが格段に大きくなったことで、バックアップは格段に難しくなっている。かつては全システムがフロッピーディスクに収まっていて、それを丸ごとコピーしておしまいな時代もあったのに。Microsoftが一番鈍いのはここだろう。Windows 8になった今でさえMacの移行アシスタントやTime Machineのような「鎮痛剤」を用意していないのだから。せっかく新OSが出ても、旧PCからの乗り換えが面倒で買い控えている人々がどれほどいるのか、Redmondの人々にはまだわからないのだろうか？

いずれにせよ、エンドユーザが空腹であることを、我々はもう期待できないということは確かだろう。しかし彼らの苦痛はまだまるで終わっていない。それがある限り、我々の商売の余地が終わることはないはずだ。SD



Google、Apple、Twitter…

深掘り 裏読み 最新Webトレンド

第34回

協力：TechCrunch Japan
http://jp.techcrunch.com/

▶Facebook、モバイルウォーズを仕掛ける
▶Facebookと民主主義

滑川 海彦 NAMEKAWA Umihiko◉techtrans1999@gmail.com（TechCrunch Japan 翻訳者）
高橋 信夫 TAKAHASHI Nobuo◉nobuo.takahashi@nifty.com（TechCrunch Japan 翻訳者）

本連載では、Webメディア「TechCrunch Japan」（http://jp.techcrunch.com/）の記事を翻訳している2人が毎回、同サイトで取り上げている最新Webサービスや企業・ビジネスに関する膨大なエントリを訳出する中で集積している、米国を中心とした動向、さらにその背景を解き明かしていきます。

Facebook、モバイルウォーズを仕掛ける

滑川 海彦

早いもので2013年になりました。筆者が『ソーシャル・ウェブ入門　Google、mixi、ブログ…新しいWeb世界の歩き方』[注1]という本を書いたのが2007年で、この4月で満6年になります。あとから振り返ると2007年というのはいろいろな意味でインターネットとITにとって大きな転機となる年でした。まず1月にスティーブ・ジョブズがiPhoneを発表し、これに対抗するように7月にはGoogleの主導によりAndroid規格が発表されました。

しかし当時この2つのプラットフォームがテクノロジ界の地図を塗り替えるだけでなく、世界中の人々のライフスタイルを一変させてしまうようになるとは誰も考えていませんでした。もしかするとスティーブ・ジョブズ本人でさえそのスピードは予測していなかったかもしれません。iPhoneとAndroidの爆発的な普及のせいで、当初スマートフォンといえばRIMのBlackBerryを代表とするビジネスマン向けのニッチな高級機を指したことも、今ではすっかり忘れられているのではないでしょうか。

iPhone/Android時代になって一気に重要性を増したのが写真と位置情報です。位置情報の要となる地図の重要性については高橋さんが書いていますが、ここではモバイル写真共有とローカル情報をめぐる最近のFacebookの攻勢を見ていきましょう。

iPhoneのカメラが進化し、それに対抗してAndroid機のカメラも長足の進化を遂げると、我々の生活における写真の位置付けが大きく変わりました。何千万もの人々が24時間肌身離さずカメラを持ち歩き、目についたものを何でも記録するようになったのです。2011年6月に老舗写真共有サイトのFlickrで撮影に利用されたカメラでiPhoneが首位に立ったことがこの動きをよく表しています。

FacebookによるInstagramの買収とその後の動向

写真を撮ったら友だちや知り合いに見せたいと思うのが自然です。FacebookやTwitterはそうして撮った写真を共有する格好の場になりました。

モバイルサービスにおける写真の重要性を劇的に示したのがInstagramでした。元Googleのケビン・シストロムらが2010年10月にiPhone向けInstagramアプリをApp Storeにリリースします。Instagramは1年未満で1,000万人のユーザを集め、2012年にはアップロードされた写真が10億枚を超えるという驚異的な成長を遂げます。ここでTwitterとFacebookがInstagramを買収しようと激しく競争しますが、結局マーク・ザッカーバーグが独断で10億ドルとい

注1）技術評論社、ISBN978-4-7741-3081-1

う途方もない額を提示したことでFacebookが勝利し、2012年5月にInstagramはFacebookの傘下に入りました。

当初Facebookは「Instagramは従来どおり独立したサービスとして運営される」としていましたが、12月に入ってInstagramの写真が突然Twitterに表示されなくなりました。これはInstagramのトラフィックがライバルに流れるのをFacebookが嫌ったためでした。同時に、Facebook本体がInstagramのユーザデータを利用し、またInstagramにアップロードされた写真やユーザのプロフィールを広告に利用できるように、利用規約が改正されました。

実はInstagramは買収以前にはまったく売り上げがなく、ビジネスモデルさえありませんでした。1ドルも収入のない会社に10億ドルを払ったのですから収益化を図るのは当然と言えます。

規約改正では、「Facebookの規約改正にはユーザの投票を必要する」という規約も廃止されました。この規約改正に対する投票が行われ、投票した約67万のユーザの88%が反対しましたが、ユーザ投票が強制力を持つには3億人の投票が必要だったため、改正が阻止されることはありませんでした。

こうしたFacebookのやや強引な動きに批判の声も上がっていますが、広告の表示頻度と質がユーザ体験を大きく損なわないようコントロールされている限り、さほど大きな問題にならないでしょう（なお、ケビン・システロムは「改正規約がユーザの写真を広告の一部として第三者に売るように誤解されたようだが、そのような計画はいっさい持っていない」としてその部分を削除すると発表しています）[注2]。

一方、Facebook/Instagramの攻勢に対して、Twitterも早速独自のフィルタを導入するなど写真共有機能の強化で防戦に務めています。Googleは iOS向けの人気写真編集アプリを提供するSnapseedを買収し、2012年12月にAndroid版を公開しました。筆者も利用してみましたが、スワイプやタップだけで輝度やホワイトバランスの調整など高度な編集が簡単にでき、大いに気に入りました。またGoogle+はもちろん、FacebookやTwitterにも同時に一括投稿できるのも便利です。TechCrunchでも絶賛されています[注3]。

モバイル対応機能を強化するFacebook

Facebookはローカル位置情報サービスにも本格的に参入してきました。ザッカーバーグは2012年9月のTechCrunchカンファレンスで「Facebookは『友だちが過去6ヵ月間に訪れたことがあり、いいね！したニューヨークの寿司屋はどれか？』という質問の答えを誰よりもよく知っている」と述べましたが、それがいよいよNearby（**図1**）というモバイルでの新機能で具体化しました。

Nearbyは、付近のレストラン、バー、店舗などのローカルビジネス情報がカテゴリ別に表示され、友だちと評価やコメントを共有できる機能です。2012年12月中旬時点では日本版には導入されていませんが、「付近の情報」タブをアップデートする形で導入されるものと思われます。Facebookの圧倒的なユーザ数と長い利用時間を考えると、YelpやFoursquareなどローカルビジネス情報や位置情報の共有サービスには大きな脅威となりそうです。

Facebookは長年モバイルが弱点だと評されてきましたが、もはやそうは言えなくなってきているのではないでしょうか？

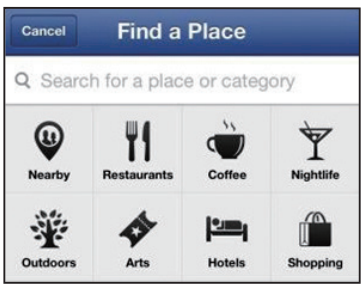


図1　Facebookの新しいモバイル機能Nearby

注2）「Instagramのケビン・システロム、『疑問に答え、間違いを正す』ことを約束」URL http://jp.techcrunch.com/archives/20121218instagram-co-founder-kevin-systrom-says-it-is-committed-to-answering-questions-and-fixing-mistakes/

注3）「Snapseed徹底解説—Googleが買収したiOSの人気写真編集アプリにいよいよAndroid版が登場、iOS版は無料に」URL http://jp.techcrunch.com/archives/20121206google-launches-snapseeds-powerful-photo-editing-app-for-android-makes-its-ios-version-free/

Facebookと民主主義

高橋 信夫

米国では大統領選挙が終わったばかりですが、そんな中2012年12月10日、7ページでも触れているようにFacebookから投票制度がなくなりました。「えっ、そんなものあったの？」と思う方のほうが多いかもしれませんが、Facebookでは2009年4月に初めて全メンバーによる直接投票が行われました。どういう場合に投票が行われるのでしょうか。Facebookが利用規約に大きな変更を加える際には、まずその変更案をFacebookの記事として掲載します。この記事に対して7,000件以上「コメント」が付くと「投票」を行う、と決められています。実は、結果的に最後のFacebook投票の対象となった規約変更の主題は「投票制度の廃止」でした。そして、総投票数の88%が規約変更に反対でした。なぜ、投票制度は廃止されてしまったのでしょうか。

Facebook投票のルール

Facebookメンバーであれば誰でも1票を投じることができます。ただし1つ重要な前提があります。有効投票数が「全メンバー数の30%」に満たないと結果が効力を発揮しないのです。一般の選挙であれば、いくら国民の意識が低くても投票率30%は維持できそうですが、Facebookではそうはいきません。現在Facebookのメンバー数は約10億人ですから、投票を有効にするには3億人が参加しなければなりません。直感的に「無理っぽい」と誰もが思うでしょうが、最終的に投票したのは66万8,500人、全ユーザのわずか0.0668%でした。ちなみに過去2回の投票は、2009年4月が66万5,000人、2012年6月が34万2,000人なので、これでも今回が最高です。

しかし、そもそもどれほどの人が投票のことを知っていたのでしょう。Facebookに入っていても「そんな話は聞いていない」という方が多いことでしょう。実は筆者(高橋)も以前の投票については知りませんでした。ただし、今回Facebookは、全メンバーに投票を呼びかけるメールを送り、専用ページで投票を終えると、自分が投票したことを友だちに知らせるための「シェア」ボタンが表示され、シェアした記事をニュースフィードで目立つように表示する、などの配慮をしました。

Facebookを「民主主義」と呼ぶのは民主主義に対する侮辱だ

(Gregory Ferenstein、2012/12/6)

投票制度の廃止に対して、多くのブロガーたちから「Facebookは民主主義を捨てた」という趣旨のコメントがありました。しかし、この記事は、「そもそもFacebookは民主主義ではない」という主張です。「誰もザッカーバーグを選んでいないし、スパマーを裁く陪審員もいない」というわけです。全体的には「投票≠民主主義」で、ギリシャ時代や米国の女性参政権などにも踏み込んだやや硬い内容ですが、投票そのものよりも対話が重要なので、今後Facebookが別の形でユーザの声を聞くようになればむしろ民主的になるかもしれないと言っています。

今回の「投票を廃止するための投票」でもわかるように、Facebookで30%の投票率を達成することは極めて難しいでしょう。投票の認知度、関心度の問題もありますが、今回の投票でいえば選択肢は規約改訂に「反対」か「賛成」の2択です。反対の人はともかく、「とくに反対でもなくどちらでもよい」と思う人は、反対するのも積極的に「賛成」するのも気が引けて「棄権」してしまうかもしれません。投票所に出かけて目の前に投票箱が置かれているわけではありませんから。結局、「もともと破綻している投票制度がなくなったからといって、大騒ぎすることはないだろう」という話なのですが、米国では民主主義の象徴とも言える「投票」を廃止する、という行為そのものに抵抗を感じる人が多いのかもしれません。

Instagramとのデータ統合

ところで、投票制度の廃止と併せて、もう1つの大きなテーマは「Instagramとのデータ統合」でした。

これは「データ利用規約」を改訂して、系列会社とデータを自由にやりとりできるようにする、というものですが、具体的にはFacebookが2012年に買収したInstagramを指しているのは明らかです。

実は、ユーザに直接影響を与える可能性のある変更はむしろこちらです。現在Instagramには広告がありませんが、近いうちに導入するという噂があります。ところがInstagramのユーザ情報は割に淡白で、それをもとにターゲット広告を打つのは難しい。そこで、Facebookの個人データを利用すればInstagramでもピンポイントの広告を発信できるだろうというわけです。ちなみにこの個人データには、年齢、性別、住所などユーザが明確に指定したものだけでなく、どんな記事を書いたか、誰に「いいね！」を付けたかなども含まれます。写真の画像認識が進めば、そこに写っている商品などから、その人の好みを分析できるようになるかもしれません。最近Facebookのモバイルアプリには「写真自動アップロード」機能が付きました。アップロードされた写真はそのままでは「非公開」なので、全部がシェアされるわけではありませんが、画像認識の対象にはなり得ます。すでに「世界最大の写真共有サイト」であるFacebookは、Instagramとの連携によってますます写真を活用していくことになりそうです。

ついにiOS用Google Mapsアプリ登場

過去二度にわたってお伝えした「Appleマップ問題」に大きな展開がありました。2012年12月13日、待望のiOS用Google Mapsアプリが公開されたのです（**図2**）。「クリスマス前にはリリース」という噂は本当でした。これまでにもiOS 6でGoogle Mapsを使う方法はいくつかありましたが、これは本家作であるだけに安定しているうえに、新しい機能も加わりすばらしい出来映えだと評判です。そしてたちまちApp Storeの無料アプリ部門のランキングでトップに躍り出ました。

Appleのティム・クックCEOが「地図の不具合を詫びた」後、App Storeには「iPhone地図App」というセクションが設けられ、日本ではMapion、Yahoo!ロコなどが紹介されていますが、Google Mapsの姿はありません。もちろん、Google Mapsの登場でこのセクションがなくなったわけでもありません。

Googleマップ公開後のiOS 6移行率は0.2%増（Chitika調べ）。Appleマップによる移行への影響は誇張？

（Darrell Etherington、2012/12/15）

iOSのマップがGoogleからAppleに切り替わったのはiOS 6からです。iPhone 5は最初からiOS 6が入っているので選択の余地はありませんが、それ以前のiPhoneユーザの中にはマップを理由に「iOS 6にはアップデートしない」という人たちがいました。その多くは「Google Mapsが出るまでアップデートは控える」とも言っていました。そういうわけで、ブログ界には「これでiOS 6への移行が一気に進むだろう」という空気が漂いました。

ところが、前述の記事によると、Google Mapsの公開前と公開後1日半経過後のiPhoneトラフィック全体に占めるiOS 6の割合は、72.77%と72.94%で事実上変化はありませんでした。記事ではその理由について、「ほとんどの人がとっくにアップデートを済ませたのだろう」と述べています。「Google Mapsの存在がまだ知られていない可能性」も否定はできませんが、一夜にしてランキングのトップになったのに、マップのためにアップデートを控えるような人たちがこのビッグニュースに気づかないというのも考えにくいですね。むしろ、マップに関係なく「アップデートの必要性を感じない」あるいは「アップデートって何？」という人たちがまだまだいるのかもしれません。肝心のAppleはまだ何も動いていませんが、ユーザにとってはとりあえずホッとしたGoogle Mapsの登場でした。SD

図2　iOS用Google MapsアプリがApp Storeに登場

はんだづけカフェなう

Raspberry Piで I/Oしてみよう（前編）

text：坪井 義浩　TSUBOI Yoshihiro　ytsuboi@gmail.com　@ytsuboi
協力：㈱スイッチサイエンス　http://www.switch-science.com/

Raspberry Piとは

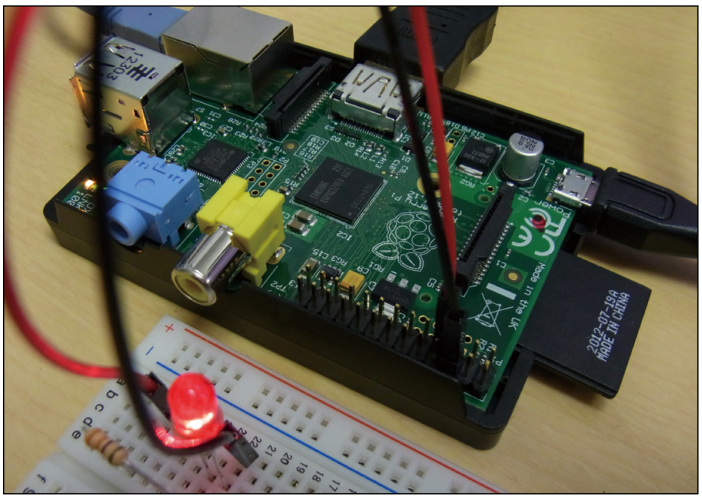
前回の記事で少しだけ紹介したRaspberry Pi（ラズベリーパイ）ですが、本誌読者には馴染みの良いボードだと思いますし、最近では入手もしやすくなりました。Raspberry Piには“Raspbian[注1]”というDebianベースの最適化されたLinuxディストリビューションが存在します。Raspbianの配布イメージをSDカードにddして書き込み、電源を接続するだけでRaspberry PiはLinuxマシンとして動作します（**写真1**）。

もちろんRaspbianだけでなく、Arch Linux ARM、Fedora ARMといったディストリビューションや、FreeBSD[注2]、RISC OSといったOSも動作させることができます。筆者個人的にはFreeBSDを推したいところですが、Raspbianほど移植が進捗しているわけではないので、本稿ではRaspbianを前提に進めさせていただきます。

Raspberry PiではLinuxが走りますので、GPIO（General Purpose I/O；汎用入出力）をシェルスクリプトやC、Java、Python、Rubyといった読者各位の使い慣れた言語でコントロールできます。また、USBがボードに搭載されていますから、Linuxで動くUSBデバイスであれば手軽に接続できます。Ethernetについても、現在、主に流通しているType Bには搭載されていますので、手軽にTCP/IPネットワークに接続することができます。当然ですが、IPv6を使いたいといったニーズにも対応することができます。

注1）http://www.raspbian.org/
注2）http://kernelnomicon.org/?p=185

▼写真1　今回実験するRaspberry PiとLED



入手方法

前回入手方法として「RSコンポーネンツで購入するのが手っ取り早い」と紹介しましたが、日本で購入するにはもう1つ、より早く届く手段があります。ModMyPi[注3]というイギリスにあるRaspberry Piのケースや周辺アクセサリを売っている店があるのですが、こちらでケースとRaspberry Piのセットが販売されています（本稿執筆時、セットで$57.71でした）。RSコンポーネンツでもケースが売られており、本体の金額と足すとこちらのほうが割安なのですが、ModMyPiのほうが早く届くという評判です。

またModMyPiでは、アクセサリの類が豊富で“GPIO Accessories”というカテゴリでは本稿で扱う予定の入出力コネクタをブレッドボードに手軽に接続できる“Adafruit Pi Cobbler Breakout Kit”の扱いがあります。GPIOを手軽に使うには、このキットも同時に入手しておくと良いでしょう。

I/O

GPIOはピンの状態がHIGHなのかLOWなのかを読み込んだり（入力）、ピンの状態をHIGH（1）やLOW（0）にして出力したりできるようになっています。

Raspberry PiのGPIOは3.3Vで、Arduinoなどで主に使われている5Vとは異なることに注意が必要です。また、mbedのGPIOも3.3Vなのですが、mbedは5Vトレラントと呼ばれる5V入力が可能な仕様になっている一方、Raspberry Piは5Vトレラントではありません。つまり、**Raspberry Piに直接5Vを入力してはいけない**ので注意が必要です。

Raspberry PiにはGPIOだけでなく、UART（Universal Asynchronous Receiver Transmitter）、I2C（Inter-Integ rated Circuit）、SPI（Serial Peripheral Interface）といった入出力も備えられています。

Raspbianを書き込む

Raspberry Piを入手したら、まずはRaspbianをSDカードに書き込みましょう。Raspbianを使用するには2GB以上、推奨では4GB以上のSDカードが必要です。筆者は机の引き出しに転がっていた2GBのmicroSDとアダプタを使いました。

Raspbianは**http://www.raspberrypi.org/downloads**からダウンロードできます。筆者は「2012-10-28-wheezy-raspbian.zip」というバージョンを使用して、この記事を書いています。ダウンロードを終えたらhashを確認して、unzipとddを使ってSDカードに書き込みます（**図1**）。なお、Windowsでddする Win32 Disk Imagerというソフトウェアへのリンクが Raspberry PiのDownloadページから張られていましたので、Windowsユーザはこれを使ってみるのも手でしょう。

初回起動するとRaspi-configという各種設定をするメニューが表示されます（**図2**）。

まず［expand_rootfs］で、SDカード全体をrootfsとして使えるようにしましょう。それから、［configure_keyboard］で先にキーマップも変更しておいたほうがよさそうです。［Generic 105-key（Intl）PC］-［Japanese］-［Japanese（OADG 109A）］の順で一般的な日本語キーボードのマップを選択できます。

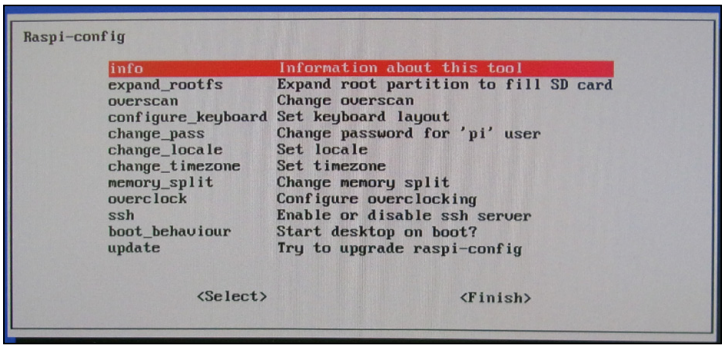
筆者はこのほかに、［change_locale］と［change_timezone］で、ロケール（ja_JP.UTF-8）とタイムゾーン（Asia/Tokyo）の設定、［ssh］でsshdを自動起動し、［boot_behaviour］でXが

▼図1　Raspbianの書き込み

```
$ openssl sha1 2012-10-28-wheezy-raspbian.zip
$ unzip 2012-10-28-wheezy-raspbian.zip
$ mount
$ sudo diskutil unmount /dev/disk1s1
$ sudo dd if=2012-10-28-wheezy-raspbian.img of=/dev/rdisk1 bs=1m
$ sudo diskutil eject /dev/rdisk1
```

注3）https://www.modmypi.com/

▼図2　Raspi-configのメニュー



起動しないように設定しました。最後に<Finish>を選択して終了です。コマンドプロンプトが表示されますが、一応rebootしておきましょう。

標準では、RaspbianはDHCPでIPアドレスを取得してきてくれましたので、Xがどうにも好きになれない筆者はsloginして作業をすることにしています[注4]。

Raspberry PiにLEDをつなぐ

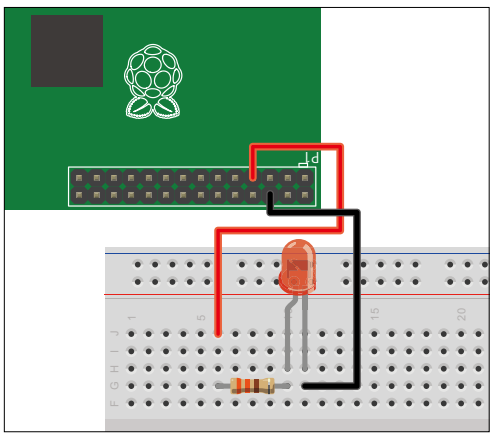
さて、Raspberry Piが使えるようになったら、さっそくI/Oしてみましょう。まず、LEDと電流制限抵抗を図3のようにRaspberry Piにつなぎます。「電流制限抵抗」というと小難しく聞こえますが、要はLEDに電気が流れ込み過ぎないようにするために追加する抵抗です。多くの赤色LEDは1.8Vや2Vで光ります。一方で先述のとおりRaspberry PiのGPIOは3.3Vを出力します。直接つなぐと電気が流れ過ぎてしまい、Raspberry PiのCPUやLEDを痛めてしまう可能性があります。このため、330Ω程度の抵抗をLEDとGPIOの間に挟んでやって、流れる電流を少なくしてやります。Raspberry PiのGPIOで流して良い電流はせいぜい16mAですので、必ず電流制限抵抗を付けましょう。

今回使用したLEDは、筆者の手元にあったOSDR5113Aという赤色LEDで、順方向電圧が2.0Vとありますので、(3.3 − 2.0) ÷ 330 ≒ 0.004ということで、0.004Aつまり4mAの電流が流れることとなります。パーツを入手するのであれば、LEDや抵抗は秋月電子、ジャンパワイヤとブレッドボードはスイッチサイエンスでそろえると良いでしょう。参考までに、部品の入手先を記載しておきます(表1)。今回使う部品はとくにこの型番でなければいけないということはありません。いずれもどこのメーカーのものでもかまいません。抵抗の値(330Ω)も、目安として270～470Ωくらいの範囲であれば異なっても大丈夫です。とはいえ、抵抗の値が大きくなるとLEDの光りが暗くなりますので注意が必要です。

シェルスクリプトでLEDを点滅

では、さっそく接続したLEDをRaspberry Piからコントロールして点滅させてみましょう。UNIXらしく、もっともお手軽なシェルスクリプトでやってみます。viなどでリスト1のようなシェルスクリプトを記述し、chmod +xし

▼図3　配線図



注4)　リソースの限られたRaspberry PiでX Windowを動かしたくないという理由もあります。

▼表1　部品と入手先の例

LED	5mm赤　色LED SLP-9131C-81H（10個入）	100円	秋月電子通商	http://akizukidenshi.com/catalog/g/gI-03557/
抵抗	カーボン抵抗（炭素皮膜抵抗）1/4W 330Ω	100円	秋月電子通商	http://akizukidenshi.com/catalog/g/gR-25331/
ブレッドボード	普通のブレッドボード	250円	スイッチサイエンス	http://www.switch-science.com/products/detail.php?product_id=313
ジャンパワイヤ	ジャンパワイヤ（オス〜メス）	395円	スイッチサイエンス	http://www.switch-science.com/products/detail.php?product_id=209

ておきます。

記述を終えたら、「$ sudo ./gpio4.sh」として実行します。LEDが1秒ごとに3回点滅を繰り返すはずです。sudoしているのは、I/Oポートを触る場合root権限が必要だからです。

このシェルスクリプトでは、最初に/sys/class/gpio/exportに4を書き込むことでGPIO4をkernelからユーザスペースにexportしています。次に、GPIO4の入出力の方向を、今回はLEDへの出力ですのでoutに設定しています。GPIO4の値に1を書き込むとGPIO4のピンがHIGH（3.3V）になり、電流制限抵抗を介してLED、そして0VのGNDに流れ、LEDが光ります。逆に0を書き込むと、GPIO4のピンはLOW（0V）となり、GND（0V）と等位ですので電気が流れず、LEDは消灯します。

PythonでLEDを点滅

RaspbianにはPython 2.7.3rc2が最初から入っていました。余談ですが、筆者はエディタはvi、スクリプト言語はPythonを愛用しています。apt-getしてRubyをインストールすればRubyでも同様のことができますが、プリインストールであることと、筆者がPythonのほうが好きであることから、ここではPythonでLEDを点滅させてみます。

PythonでGPIOをコントロールする際には、RPi.GPIOというモジュール[注5]を使うと良いでしょう（リスト2）。これもプリインストールされていました。

Pythonでも同様に、「$ sudo python gpio4.py」といった具合にroot権限で実行する必要があります。

Raspberry Piで何をしよう

誌面の都合で、今回はI/Oのうち Outputだけになってしまいました。筆者はRaspberry Piを使って、オフィスの玄関の電気錠をHTTPから解錠、施錠できるようにしてみたいと思っています。USBにWebカメラを接続したり、Wi-Fiドングルを接続したりとマイコンボードでは面倒なことが手軽にできるのがRaspberry Piの魅力ではないでしょうか。SD

注5）http://code.google.com/p/raspberry-gpio-python/

▼リスト1　gpio4.sh

```
#!/bin/sh
echo "4" > /sys/class/gpio/export
echo "out" > /sys/class/gpio/gpio4/direction
for i in 1 2 3
do
  echo "1" > /sys/class/gpio/gpio4/value
  sleep 1
  echo "0" > /sys/class/gpio/gpio4/value
  sleep 1
done
```

▼リスト2　gpio4.py

```
import RPi.GPIO as GPIO
import time
GPIO.setmode(GPIO.BCM)
GPIO.setup(4, GPIO.OUT)
for i in range(3):
    GPIO.output(4, GPIO.HIGH)
    time.sleep(1)
    GPIO.output(4, GPIO.LOW)
    time.sleep(1)
GPIO.cleanup()
```

TEXT＝pha　pha.japan@gmail.com

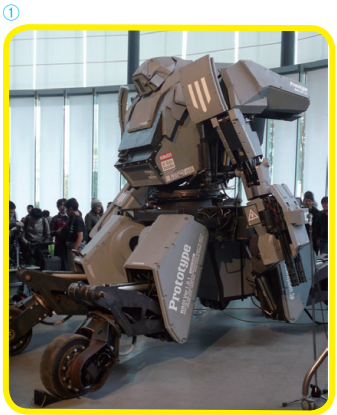
Maker Faire Tokyo 2012に行ってみた

（『Make 日本語版 Vol.1』ISBN978-4-8731-1298-5）

個人によるものづくりの祭典

2012年12月1日（土）、2日（日）にお台場の日本未来館でオライリー・ジャパンの主催で開催された「Maker Faire Tokyo 2012」に遊びに行ってきた。オライリーが発行している『Make』は、電子工作や模型やロボットなどハードウェアを工作していろんなものを作る「個人のものづくり」をテーマにした雑誌だけど、Maker Faireはそんなふうに工作をしている人たちが一同に集まって作品を展示するイベントだ。2011年まで東京では「Make: Tokyo Meeting」（略称MTM）という名称で開催されていたんだけど、2012年からは他の国でのイベントと名称を統一するためにMaker Faireという名前になったそうだ。2012年は約240組の個人や団体が展示をしていてすごい熱気だった。

①



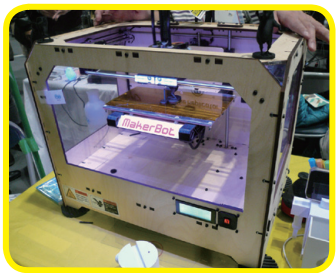
KURATASと3Dプリンタ

今回のイベントの一番の目玉はKURATAS（クラタス）だった（①）。KURATASというのは高さ約4メートル、重さ約4トンの人型巨大ロボットで、実際に人が乗り込んで操縦することもできるし、腕でものをつかんだり花火やBB弾を発射することもできる。KURATASは「巨大ロボットに乗って動かしてみたい」という情熱から始まったプロジェクトで、「水道橋重工[注1]」という名前のたった2人のチームで作られている。今回の展示はプロトタイプだが、量産化して販売も予定されている（価格は135万3,500ドル＝約1億1,100万円）。巨大ロボットが量産化して販売されるという、アニメや映画であったような世界観が実現することを目指しているそうだ。いろんなタイプの巨大ロボットが普通に開発･販売され、街のあちこちで見かけたりするような時代が本当にもうすぐ来るのだろうか。楽しみだ。

あと今回展示が多かったのは3Dプリンタだ（②）。3Dプリンタというのはその名のとおり立体物をプリントできるプリンタで、3Dの立体のデータを入力してやればそれを樹脂などで立体化したものを出力することができる。以前は3Dプリンタは高価だったけれどここ数年で低価格化が進み、今では10万～20万円も出せば安いものが買えるようになっている。会場では3Dプリンタでの作成例としてフィギュアやiPhoneケースなどの展示が多かった。バーチャルなデータ上にあるものが現実世界に立体化されるのってこの世ならざるものがこの世に受肉されるみたいな感じでテンションが上がりますね。こういう技術がメジャーになってもっとバーチャルの世界とリアルの世界がシームレスになっていくと楽しい。

注1）URL http://suidobashijuko.jp/

②



③

多様な出品物

他にもいろいろ面白かったものを紹介。

スライムを触ったりこねたりすることでいろんな音が出るという楽器(③)。テルミンなんかもそうだけど、こういう斬新なインターフェースを実験したい場合って楽器にするのが手っ取り早いのかもしれないと思った。きっちりと実用的な操作をする必要がないので。

ぬいぐるみで作った電気抵抗やコンデンサ(④)。これはかわいいだけじゃなく、下に置かれているブレッドボードに差し込んで本当に使えるそう。無駄にすごい。

レゴで作成された、カフェラテに絵を描くプリンタ(⑤)。ほぼ全部レゴの公式の部品で作られているらしい。レゴと『Make』は相性が良くて、他にもレゴでできた作品がたくさんあった。

ニキシー管を使ったオーディオメーター(⑥)。

SF作家の野尻抱介さんが展示していた初音ミク型スリングショット(⑦)。スリングショットによる狩猟は基本的に免許なしでできるらしい(地域によっては禁止の場所もある)。

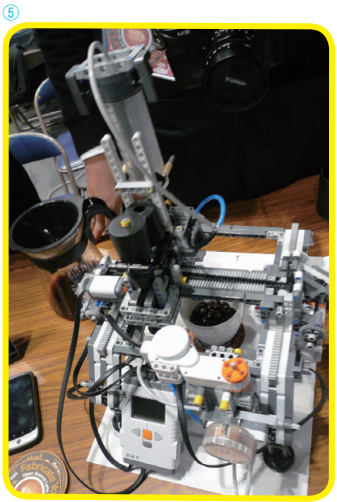
LANケーブルのバリが100グラム420円で売っていた(⑧)。何に使うんだろう……。

新しい産業革命?

最近の『Make』や個人のものづくり界隈の盛り上がりの原因の1つは、『ロングテール』『フリー』などのベストセラーで知られるクリス・アンダーソンの『MAKERS』という本だ(⑨)。かつては何かものを作ろうとすると大きな設備や多額の資本が必要だった。しかし最近では3Dプリンタなどの普及で安価にものづくりの設備が整えられるようになったし、インターネットを使って資金をたくさんの人から集めるクラウドファンディングの登場によって、個人が手軽にものづくりを始めることができるようになった。これは新たな産業革命かもしれないということを『MAKERS』は語る。かつては高価だったパソコンが安価になって今では誰でも手軽にプログラミングを始めることができるようになったけれど、そのソフトウェアで起こった変化が今度はハードウェアの分野で起こりつつあるのだろう。ギークハウスでも3Dプリンタを1台買ってみんなで使おうかな……。SD

④

⑤

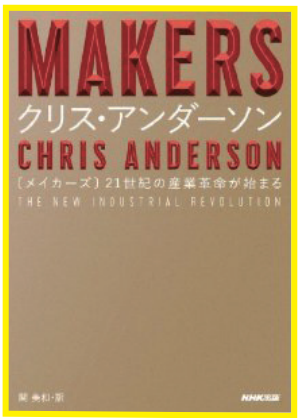


⑥

⑦

⑧

⑨



クリス・アンダーソン著、関美和訳『MAKERS—21世紀の産業革命が始まる』(NHK出版、2012年、ISBN978-4-1408-1576-2)

PRESENT 読者プレゼントのお知らせ

『Software Design』をご愛読いただきありがとうございます。本誌サイト http://sd.gihyo.jp/ の「読者アンケートと資料請求」からアクセスし、アンケートにご協力ください。ご希望のプレゼント番号をご入力いただいた方には抽選でプレゼントを差し上げます。締め切りは2013年2月17日です。プレゼントの発送まで日数がかかる場合がありますが、ご了承ください。

ご記入いただいた個人情報は、プレゼントの抽選および発送以外の目的で使用することはありません。アンケートのご回答については誌面作りのために集計いたしますが、それ以外の目的ではいっさい使用いたしません。ご入力いただいた個人情報は、作業終了後に当社が責任を持って破棄いたします。なお掲載したプレゼントはモニター製品として提供になる場合があり、当選者には簡単なレポートをお願いしております（詳細は当選者に直接ご連絡いたします）。

01 Wi-Fi SD カードリーダー REX-WIFISD1

1名

スマホやタブレットと本製品を Wi-Fi 接続することで SD カードや USB フラッシュメモリのデータを読み書きできます。USB 経由でスマホ充電用のモバイルバッテリーとしても使用可能。iPhone なら 1 台分充電できます。

提供元 ラトックシステム
URL http://www.ratocsystems.com

02 パスワードマネージャー ミルパス

1名

パスワードや ID などを記録するための小型情報管理端末。起動時に必要なマスターパスワードだけを覚えていれば、本端末で最大 200 件のアカウントを管理可能。文字は専用タッチペンで入力できます。

提供元 キングジム
URL http://www.kingjim.co.jp

03 DATAHOTEL オリジナル ウォールマグ

10名

鯖（server）、雲（cloud）と印字された DATAHOTEL オリジナルウォールマグ。完全密閉で持ち運び可能。電子レンジも利用でき、再加熱可能です。鯖、雲のどちらが当たるかは当選してからのお楽しみ。

提供元 データホテル
URL http://datahotel.co.jp

04 V ブロック（ペントレー） ショートタイプ

1名

PC のキーボードの奥に並べて置いて使うペントレーです。ちょっとしたメモを取るときに便利です。ラバーウッドの無垢材を V ブロックの形状に加工したシンプルなデザイン。長さは 30cm。

提供元 margherita **URL** http://www.margherita.jp

05 VMware vSphere 構築・運用レシピ

2名

工藤 真臣、田中 隆行、中本 滋之、樋口 美奈子 著／
B5 変型判、336 ページ／
ISBN = 978-4-8443-3317-3

VMware vSphere を知り尽くした著者陣がさまざまなノウハウを解説します。管理用クライアントソフト vSphere Client、仮想化基盤の ESX/ESXi、管理用サーバの vCenter Server など幅広く紹介。

提供元 インプレスジャパン **URL** http://www.impressjapan.jp

06 OpenFlow 実践入門

2名

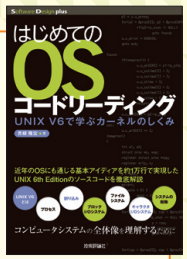
高宮 安仁、鈴木 一哉 著／
A5 判、336 ページ／
ISBN = 978-4-7741-5465-7

SDN の概要から OpenFlow プログラミングフレームワーク「Trema」を利用したネットワークプログラミング、さらに本格的な OpenFlow アプリケーションをケーススタディで解説します。

提供元 技術評論社 **URL** http://gihyo.jp

07 はじめての OS コードリーディング

2名



青柳 隆宏 著／
A5 判、448 ページ／
ISBN = 978-4-7741-5464-0

本書は、Sixth Edition UNIX（UNIX V6）を題材に、OS の全体像をくまなく解説します。カーネルのソースコードを読み解くことで、コンピュータシステムの全体像が理解できるようになるでしょう。

提供元 技術評論社 **URL** http://gihyo.jp

第1特集

UNIXコマンド、fork、pipeを復習し、高度なスクリプティングへ

シェルスクリプティング道場

多機能なプログラミング言語でも、いくつかの手順を踏まないとできない作業が、数個のコマンドや数行のシェルスクリプトだけで簡単に実現できた、ということはありませんか？　それは、シェルスクリプトには「OSの機能を直接使える」「簡単に書ける」など、実用性重視の特徴がたくさんあるからです。高度なシェルスクリプトを書くには、シェルの特徴を理解し使いこなすことが必須です。そのために、本特集ではシェルの動作原理を探求していきます。

CONTENTS

第1章 UNIXの思想に立ち返れば見えてくる なぜ今、シェルスクリプトの習得が必要なのか?

(有)ユニバーサル・シェル・プログラミング研究所　http://www.usp-lab.com
上田 隆一 UEDA Ryuichi　Twitter ID：@uecinfo

ほかのプログラミング言語を十分に使いこなせるエンジニアであっても、基本に立ち返ってシェルスクリプトを習得する意味はあります。それは、開発だけでなく日常の仕事にまでコンピュータをフル活用できるようになるからです。本格的な学習に入っていく前に、まずはシェルスクリプト習得の意義について考えてみましょう。

UNIXの神話

この原稿を執筆中にちょうど出雲大社に参詣して思い出した話ですが、日本の神話によると、日本を作ったのはイザナギノミコト、イザナミノミコトであるとされています。日本書紀や古事記には、この二柱の神が島を作り、神を作った話「国産み、神産み」が記されているそうです。筆者は概略を知っているだけですが、個人的には根が理屈っぽいので、島や神が産まれる因果関係の理屈付けにおもしろさを感じます。

UNIXの黎明期にも「国産み、神産み」に匹敵するかもしれない興味深いエピソードがあります。『Netizens: On the History and Impact of Usenet and the Internet』[注1]という本の第9章 "On the Early History and Impact of Unix Tools to Build the Tools for a New Millenium" にそのときのことがまとめられています。この一部を参考にして、UNIXで起きた「神話」をまとめたいと思います。こちらもソフトウェアが産まれる際の因果関係が話の肝になるようです[注2]。

UNIXがUNIXになった日

1969年、AT&Tベル研でDennis Ritchie、Ken Thompson、Rudd CanadayがUNIXファイルシステムを考えていたとき、Doug McIlroyが「パイプ」をUNIXに実装するように主張していました。

McIlroyの言う「パイプ」というものは、あるコマンドの出力をほかのコマンドに即座に渡す現在のUNIX系OSのパイプと同じものです。RitchieたちはMcIlroyの説明に理解は示したものの、後回しにしていました。

パイプが最初に実装されたのは3年後の1972年です[注3]。3年も後回しにしたパイプでしたが、実装されたその日のうちに、その場にいた誰もがその重要性にすぐ気づいたそうです。実装されたパイプはUNIXのほかの部分や使い方に影響を与え始め、パイプを使うと便利なように古いコマンドが書き直され、シェル上でワンライナーが書かれるようになりました。

grep、sedの誕生

そしてgrepが誕生します。edというエディタに文字列検索がついており、McIlroyがそれを使ってある作業をしていました。しかし、edのメモリの上限に引っかかって作業に難渋し、

注1）Michael Hauben、Ronda Hauben 著、Wiley-IEEE Computer Society Press、1997年

注2）本と同様の文章が、http://www.columbia.edu/~hauben/book/で閲覧できます。

注3）リリースは1973年のVersion 3 UNIXです。

Ritchieに「edから検索機能を取り出してコマンド化してほしい」という依頼を出します。コマンドは一晩でできあがり、edでの検索コマンド**g/<re>/p**（gはglobal、<re>は正規表現、pはprint）にちなんでgrepと名付けられました。grepは毎行処理ですのでメモリをほとんど使いませんし、いちいちedを開く手間もありません。ほかの機能もedから取り出され、まとめてsedというコマンドになりました。

「ソフトウェアツール」という言葉の誕生

この教訓から、「コマンドは単機能で作るべき」、「複雑なことをするにはコマンドをパイプで組み合わせるべき」、「標準入出力から字を読み書きするべき」という考えがベル研内で生まれます。この考え方は「ソフトウェアツール」と名付けられました。コマンドにもソフトウェアツールやUNIXツールという別名がつきます。

このようにUNIXの使い方に明確な方針が立つことで、ソフトウェアツールという考え方を実現するため、多くの単機能なコマンドが作られました。また、ツールを作るためのツール、lexとyaccが誕生し、そこからawkが誕生します。

シェル／シェルスクリプトとはいったい何か

シェルやシェルスクリプトは、この「神話」以降、「ソフトウェアツールを使うための正統な道具」であり続けています。「シェル」は別名「コマンドラインインタプリタ」と呼ばれます。シェルはユーザから文字でコンピュータに対する命令を受け取り、OSが理解できる命令に変換してOSに伝えます。CLI（Command Line Interface）を利用している限り、我々は文字だけでコンピュータと対峙します。

たとえば、我々が

```
$ cat file
```

とシェルに打ち込むと、cat(1)[注4]コマンドが/bin/から探され、fileがcatに渡されてfileの中身が表示されます。

各コマンドは1つの仕事しかしません。しかし、不思議なことに組み合わせるとさまざまな仕事をします。たとえば、某連載の原稿から「hoge」という文字列を検索すると、

```
$ grep hoge *.rst

201202.rst: [ueda@cent LOG]$ cat hoge
201204.rst:* ``<hoge>`` から ↵
``</hoge>`` までの塊
201204.rst:* あるいは ``<hoge ... />``
（略）
```

という検索結果が得られますが、この出力から「hoge」を含むファイルのリストを作りたければ、

```
$ grep hoge *.rst | sed 's/:..*$//' | ↵
sort | uniq
201202.rst
201204.rst
201205.rst
（略）
```

とパイプで4個コマンドをつなげると実現できます。

さらに多くのコマンドを組み合わせるようになると、端末に入力するよりもエディタで書いてファイルに保存するほうが何かと便利になります。これがシェルスクリプトです。「シェルスクリプトはコマンドをたくさん組み合わせて仕事をさせるためのもの」ということになります。

良いユーザになろう

さて、ソフトウェアツールが当時画期的で、UNIXの性格を決定づけたことは知っておくとして、40年後を生きる我々は、それを習得すべきでしょうか？　CLIを使うのは時代遅れで

注4） cat(1)の後ろの数字は、man(1)コマンドでマニュアルを見るときの章番号です。コマンドはmanの第1章に書いてあるので、cat(1)とするとコマンドのcatという意味になります。printfなど、コマンドにもC言語の関数にもあるものなどを識別するときに便利です。

はないでしょうか？

筆者はもちろん「習得すべき」という立場です。CLIは時代遅れというより、長く我々から遠ざかっていたと言うべきで、今は普通の人がMacBookにbash環境を持ち歩いています。CLIが「普通の人」にとって本当に古いものなのか、考える良い機会です。

「普通の人」とあえて書いたのは、UNIXの世界では表裏一体の次の両者を、本稿では分離分割したいからです。

・プログラマ：コンピュータにプログラムを組み込む人
・ユーザ：コンピュータの機能だけ使いたい人

たとえば筆者でしたら、システム開発しているときがプログラマ、名簿の管理や原稿書きをしているときがユーザです。後者のときもプログラミングをしますが、それはTeX原稿をいじるなどのためで、何かをOSに組み込んでいるわけではありません。

でも、道具としては後者の使い方が普通ではないでしょうか。筆者は、ソフトウェアツールを理解し、道具としてシェルやシェルスクリプトを使い倒す「一般の優良UNIXユーザ」が増えれば、世の中がスッキリすると考えています。コンピュータが普及したのはここ十数年のことで、まだまだ道具のほうが人間を振り回しているようですので。

本章では、本誌読者をきっかけとして普通の人まで優良UNIXユーザにするという壮大過ぎる意気込みで、シェルやシェルスクリプトを使い倒す所作について述べたいと思います。

プログラマに良い仕事をしてもらう

プログラマの視点からは、ソフトウェアツールの考え方に従うと見通しがよくて性能も良いプログラムが作れる、ということが言えます。

edからgrepとsedが分離された逸話が象徴的です。ユーザのMcIlroyから見たら、分離する理由は次のようになります。

・edの機能の一部だけを使いたかった
・edのメモリ上限が作業の邪魔をした

edから解放されることでgrepは、

・わざわざedを開かずに利用可能
・出力結果をバッファや中間ファイルに置く必要がなく、ただ標準出力へ放出すれば十分

と、小型で使い勝手の良いものになりました。McIlroyは結局、シェルでコマンドを操作することと引き換えに、よく働くソフトウェアを作ってもらうことに成功しました。

ソフトウェアではなくデータを気にする

一方で、よりedの機能を強化するという考え方もあったはずです。この考え方はプログラミングの統合開発環境やオフィススイートの考え方です。なぜそちらに行かなかったかは、当時のコンピュータには無理だったというのが第一の理由でしょうが、一方で、そちらに行かなかったおかげでUNIXの成功があるとも言えます。次の言葉が核心を突いています。

“データが全てに優先する。もし適切なデータ構造を選んで物事を整理すれば、アルゴリズムはほとんどの場合に自明となる。アルゴリズムではなく、データ構造がプログラミングの中心である。”——R.Pike

我々が使っているプログラムは、データ構造に引きずられています。便利さとデータ形式のバランスを気にすることができれば、見せかけの便利さを追求して、コンピュータに振り回されることはなくなるでしょう。

CLIと対極にいる便利なオフィススイートでは、たびたびデータ構造に無理が発生することがあります。これは宿命です。とくにエクスポート機能で表面化します。エクスポートする必要があるのは、各ソフトウェアでフォーマットが違うからですが、n種類のフォーマットを相互変換するには最悪n^2-n個の変換ソフトウェアが必要です。

変換の必要をなくすには「統一フォーマット」となるわけですが、我々はもう20年近く、その代表的な失敗例と常に戦ってきました。

「なんでこのドキュメント、5MBもあるんだ?メールで送れない!」

悪ノリしてついでに書いておくと

「テキストファイルはみんなが読めないので○○の形式で送ってください」

というIT企業の重鎮というか珍獣を量産する遠因にもなっています。

シェルスクリプトで扱うのは、つまらないテキストファイルです。しかし、多少修正すれば、どのコマンドやソフトウェアも自分が作ったテキストファイルを受け入れます。PDFにも紙にもしてくれるし、コピー&ペーストすれば「統一フォーマット」にも変換してくれます。自分の使うべきツールを自分で見つける必要がありますが、「あの機能がこのソフトウェアにない」という理由で袋小路に追い込まれることはありません。

シェルは実用一辺倒と知る

シェルの話に戻りましょう。シェルの文法は、筆者の主観の枠を越えて汚いのですが、なぜでしょうか? 最近、人気のあるスクリプト言語は、実用性と次の3点のバランスが良く、支持を集めていると言えます。

❶安全性:プログラマにメモリを触らせない
❷ある理論の具現化:オブジェクト指向、関数型、文法上の美しさの追求
❸マルチプラットフォーム:OSとは切り離されている

シェルでもメモリに触れないのは一緒ですが、あとの2つが正反対です。OSべったりですし、文法も結構へんてこです。

一方、C言語などと比較すると、今度はメモリに触れないぞということになります。だいたい、シェル変数は全部文字列です。ポインタどころか、文字列型しかありません。

```
A="This is a pen."  ←文字列
B="1.2345"          ←文字列
C=/usr/local/bin    ←文字列
```

こう考えるとシェルやシェルスクリプトは、とても分が悪いように見えます。が、これは比較の方法がよろしくありません。

まず、どのシェルも「よく使うものほど短く書ける」ようになっています。これが汚い原因です。見かけのきれいさと合理主義は違います。一番よくわかるのは不等号記号の使い方です。次のように、シェルはほかの言語とはまったく違います。

・普通の言語:>は数値の比較
・シェル:>はファイルへの出力

たとえばbashで数字の大小比較をすると、

```
if [ 1 -lt 2 ] ; then ...
```

のようになります。-ltとless thanが一瞬で結びつかない日本人を意味不明な世界に追い込みます。おまけに[の前後に空白がないと叱られますし、bashだと小数も比較できません。とどめを刺すと、これはbashで定義された記号ですらありません。困ったものです。

一方で、ファイルの中身をほかのファイルへの書き出すには、bashだと、

```
$ cat file > file2
```

で済みますが、ほかの言語だとどうでしょうか?

```
data = open("file","r").read()
open("file2","w").write(data)
```

とか、

```
copy("file","file2")
```

と書かなければなりません。

普通の言語では、>は不等号です。また、文法をきれいに保とうとすると、シェルの>のよ

うなショートカットはかえって異物になってしまいます。

シェルの場合は、まず人間が手でコマンドを入力するところから話が始まっています。そのため、頻繁に行う操作に短い記号を割り当てるのは自然なことです。変数も文字列以外のものがあっては困ります。記号が増えます。それはシェルとしては失格です。

こだわらずに書き捨てるように書く

ソフトウェアツールの考え方に照らせば、システムの裏で働くshのスクリプトよりも、ユーザが自分の便利のためにちょっと書きなぐったスクリプトのほうが、シェルスクリプトでなければならない必然性が高いということになります。システム側ではソフトウェアツールを使う必要は必ずあるわけではありませんが、ソフトウェアツールのユーザはたとえ拙くてもシェルスクリプトを使う権利があります。

ユーザにとっては、スクリプトを書く時間の投資よりスクリプトに効果があれば、それで成功です。書き方にこだわったり、コマンドのきわどい機能を知っているのに使わなかったりというのは、ユーザとしては、目的に体がまっすぐ向いていません。

再利用については、ユーザとして書くスクリプトについてはあまり気にすることはないでしょう。自分の書いたスクリプトをノーチェックで使いまわすことは、多くの人の場合、実際にはなかなかないことです。

弊社の例で恐縮ですが、USP研究所ではシステムごとOS間を引っ越すときに自分たちの首を締めないよう、できるだけPOSIXに従う一方で、自分たちでコマンド(usp Tukubaiコマンド)を作ってどんどん新しいことをしています。たとえば、次のようにシェルスクリプトの用途を広げていっています。

- 集計用コマンドを作成し、シェルスクリプトで帳票を出力
- データベース用のコマンドを作り、フラットテキストDBを構築
- HTMLのテンプレートに文字を埋め込むコマンド、CGIのポストを安全にデコードするコマンドなどを作り、Webシステムを構築

これは、弊社が独自仕様を打ち立てたいからではありません。ソフトウェアツールの方法論が現在にも生きていることを証明することが、弊社の使命になっているからです。

用途を広げるためには、新しいコマンドが必須です。そして、Cで書いたコマンドはだいたいのUNIX環境に移植できます。コマンドは移植するもの、シェルスクリプトは書き捨てるものです。

不自由したらまずは探そう なかったら自作しよう

McIlroyはgrepを人に作ってもらってますが、これは、ユーザとして正しい行動です。McIlroyは自分の研究をしていてgrepが必要になりましたが、edのコードを自分で読み、grepを自分で作るのは時間がかかると判断したのだと想像しています。

実はUSP研究所でも、これはよくある光景で、McIlroyの件を弊社の日常風景も勘案してさらに想像すると、おそらくUNIXのコードを管理しているRitchieがコマンドを作るほうが、後々管理の面で良いだろうという判断が働いたかもしれません。

自分の仕事をさしおいて、人もさしおいて自分でコマンドを作るという行為は、最後の選択肢になります。ユーザは面倒な仕事から楽しいプログラミングに逃げずに、自分の仕事に集中すべきです。

コマンドがないときの所作の例をひとつ。昨年本誌12月号の連載「開眼シェルスクリプト」で、`date -f`(標準入力から時刻を受け、加工して出力)というdate(1)のオプションを紹介しましたが、筆者はこの記事を書いていて、FreeBSDに同様の機能がないことに気づきま

▼図1　dateの-fオプションの使用例

```
$ head -n 3 datefile
@1339304183
@1339305265
@1339306807
$ head -n 3 datefile | date -f -
2012年  6月 10日 日曜日 13:56:23 JST
2012年  6月 10日 日曜日 14:14:25 JST
2012年  6月 10日 日曜日 14:40:07 JST
```

▼図3　portsにdateがないか探す(FreeBSDにて)

```
とりあえず探してみる
# find ./ | grep gnu | grep date | grep
-v gnuls
./java/classpath/files/p(略)
ない!
それっぽい名前で探してみる
# find ./ | grep gdate
```

した[注5]。記事はなんとかごまかす(!)ことにして、じゃあFreeBSDでこの処理を本当に使う必要が生じたらどうしようという話になります。これがないと図1のような処理が、図2のように間にwhileが入ってしまいます。入力の行数だけdateが起動して、10万レコードもあると、結構待っていないと処理が終わりません。

もしこれが仕事であるなら、まず真っ先に、FreeBSDのportsコレクションにいつも使っているdate(1)がないか探します(図3)。

……ありません(あるかもしれませんが)。次に、別の人(おそらく社長)に依頼を出します。これでおそらく解決します。

趣味ならば、Pythonか何かで10行くらいのコードを書いて済ませると思います。筆者にとってはそれが一番早い解決方法だからです。ちょっとの間だけ「プログラマ」に戻り、オプションのないコマンドをさっと作り、次のようにパイプに投入します(図4)。そして、しれっと「ユーザ」に帽子をかぶり直します。そんなにスピードは出ませんが、date(1)を10万回呼ぶよりは速いはずです。

コマンドを真面目に作るとオプションの解析

注5) もちろん、逆のパターンもあることは強く明記しておきます。

▼図2　-fオプションを使用しない例

```
$ head -n 3 datafile | tr -d @ | while
read u ; do date -r $u ; done
2012年 6月10日 日曜日 13時56分23秒 JST
2012年 6月10日 日曜日 14時14分25秒 JST
2012年 6月10日 日曜日 14時40分07秒 JST
```

▼図4　自作コマンドの使用例

```
$ head -n 3 datefile | ./epoc2date ←自作コマンド
2012年  6月 10日 日曜日 13:56:23 JST
2012年  6月 10日 日曜日 14:14:25 JST
2012年  6月 10日 日曜日 14:40:07 JST
```

に一番時間がかかるのですが、こういうときはもう入力がわかっているので、オプションは不要です。パイプを使うと、プログラミングの難易度が恐ろしいほど低下します。

もっと余力のある人なら、date(1)のコードを改良してパッチを送ることでしょう。なんでないんだと騒ぎ立ててもいいかもしれません。

manを読もう

シェルスクリプトを書いていて、コマンドの機能が不足していると思ったら、man(1)でコマンドのオプションを調べましょう。その手の不満は、以前にも抱いた人がいる場合が多く、manページの片隅にマニアックなオプションが書いてあることがあります。先ほどの`date -f`もその1つです。

この手の発見は、日頃シェルスクリプトを使っていても頻繁にあります。つい先日指摘されたことですが、ディレクトリにたくさんファイルがあるとlsが遅いので、筆者は`echo * | tr ' ' '¥n'`で代用していましたが、実は`ls -f`を使ったほうが速いとのことです。あと、`grep -R`や`uniq -f`なども最近教えてもらいました。

すごいパフォーマンスをおすそ分けされる

よく使われるコマンドはとにかく速い。GNU grepはXeon W5580(3.2GHz)で図5のようなスピードが出ます。Xeonだからというこ

▼図5　grepの処理速度を計測

```
$ ls -lh TESTDATA
-rw-rw-r-- 1 usp usp 4.0G  6月 15 10:57 TESTDATA
$ wc -l TESTDATA
100000000 TESTDATA          ←1億行!!!
$ head -n 3 TESTDATA
2377 高知県 -9,987,759 2001年1月5日
2910 鹿児島県 5,689,492 1992年5月6日
8458 大分県 1,099,824 2010年2月22日
$ time grep 富山県 TESTDATA > hoge

real    0m3.704s
user    0m2.665s
sys     0m1.038s
$ head -n 3 hoge
7163 富山県 1,371,974 1994年5月26日
2528 富山県 6,407,486 1992年10月1日
1320 富山県 5,784,634 2009年3月7日
```

▼図6　trの処理速度を計測

```
$ tr --ver
tr (GNU coreutils) 5.97
Copyright (C) 2006 Free Software Foundation, Inc.
(中略)
$ time tr -d ',' < TESTDATA > /dev/shm/hoge

real    0m8.723s
user    0m6.133s
sys     0m2.590s
```

▼図7　コマンドをパイプでつないだ場合の処理速度を計測

```
tr単発とほぼ同じ時間で終わる
$ time tr -d ',' < TESTDATA | grep 富山県 > hoge

real    0m9.626s
user    0m10.046s
sys     0m5.815s
grep単発とほぼ同じ時間で終わる
$ time grep 富山県 TESTDATA | tr -d ',' > hoge

real    0m3.719s
user    0m2.961s
sys     0m1.294s
```

とでもありません。CPUの周波数が高く、メモリがそこそこつんであれば同様に出ます。

trも速いコマンドの1つです。HDDが追いつかないのでメモリの上に出力してみます(**図6**)。

「これがシェルスクリプトの速さだ！」と言ったらそれは嘘ですが、シェルスクリプトからこれらのコマンドを使いたい放題です。

速いのはアルゴリズムが良いからですが、良い必然性は地球上でただ1つのGNU grepを託されたプログラマと、ウン千万人いるであろうユーザという社会的構造を考えれば十分でしょう。

特集の後半でも扱われますが、ここにもパイプが力を発揮します。CPUがたくさんあれば、字を右から左に流すようなコマンドは全部マルチプロセスで動きますので、コマンドが増えたからといって計算時間が延びたりしません(**図7**)。

パイプについては、先述の「書きやすい」と、ここでお見せした「並列効果」の2大利点があります。後者の「並列効果」を我々が享受できるようになったのは、普通のマシンのCPUコアが増えた最近のことです。しかし、これは必然的でもあります。流れ作業やバケツリレーというのは、もともと効率が良いうえ、仕事の分担が行いやすいのです。

マシン全体を筆記用具のように使う

ソフトウェアツールの実践のためには、マシンを文房具やデータベースそのものとして扱うという発想が大事です。もちろん、そのうえでGUIツールも使いこなして仕事をしましょう。何かしらのUNIX系OSを用意して、(ちょっと我慢して)メインの環境として使ってみてください。Macユーザには、最初から門戸が開けています。まずはご自身の前にあるMacの端末(ターミナル)を開きましょう。

メモはテキストファイルに取り、あとから見返すときにless(1)やgrep(1)を使うように癖をつけましょう。メモをとるのは(これはもう個人の趣味ですが)Vimが好ましいですが、geditというGUIのエディタもあるので無理しないでこっちを使ってください。

公式な文章を書くときも、なるべくテキストエディタでまとめて、あとからワープロソフト

などにペーストして体裁を整えるように癖をつけてみてください。これは残念ながら適性のある人だけになってしまいますが、適性があれば、すぐにこっちのほうが要領が良いと思うでしょう。

ちょっと慣れてきたら、sed(1)やawk(1)でメモをいじってみてください。長くなればシェルスクリプトにしてください。住所録もテキストを原本にして、アプリケーションや携帯電話にはそれを加工してエクスポートしてください。sedが手についてくると、思ったほど不自由がないと感じるはずです。

シェルの裏(=OSの表)を知る

最後に、ソフトウェアツールを使い込み、より大きなデータをソフトウェアツールで捌くようになったら、シェルの裏で起こっていることをよく知る必要があります。カーネルハッカーになる必要はありませんが、動きを知ると使い方に幅が出ます。良いF1ドライバーは自分の乗るマシンのことをよく知っており、コックピットからマシンの状態をよく把握する能力に長けると言います。

本章ではあまり強調しませんでしたが、シェルというものはOSの直上にあります。「シェルの裏」にはもはやOSしかなく、シェルから大半のOSの機能を呼び出すことができます。次章からは、よりディープなOSとシェルの世界が待っています。

終わりに

ここまでソフトウェアツールとシェルスクリプトについて、とくにユーザの視点で使う理由を書いてきました。UNIX自体をいじるのは楽しいのですが、やはり、プログラム以外の仕事のために使うことが、道具としては最上の使い方です。仕事で使おうという人が1人でも出たなら、うれしい限りです。SD

COLUMN • 本で知るソフトウェアツールとUNIXの思想

本章の内容についてより深く知識を得たければ、次の本がおすすめです。シェル自体の解説本についてはここでは触れませんが、文法の解説に終始せず、コマンドを組み合わせるという観点が濃いものをお勧めします。

・Brian W.Kernighan、P.J.Plauger 著、木村泉 訳
『ソフトウェア作法(Software Tools)』
共立出版、1981年

これは、タイトルどおり「ソフトウェアツールそのもの」の本です。これを読めば、コマンドの成り立ちがわかるはずです。それ以外にも、簡潔なプログラムを書くための知見がぎっしり詰まっています。

・Mike Gancarz 著、芳尾桂 訳
『UNIXという考え方—その設計思想と哲学』
オーム社、2001年

「GancarzのUNIX哲学」の原典です。分量はそんなに多くなく、あまり技術に詳しくなくても読み物として読めます。

・Eric S.Raymond 著、長尾高弘 訳
『The Art of UNIX Programming』
アスキー、2007年

UNIX教の大著です。UNIXの考え方について、Gancarzのものより詳細に、時に攻撃的に記述してあります。やけどしないようにおっかなびっくり読むことをお勧めいたします。「awkは時代遅れだ」と書いてあるのはちょっと引っかかりますが。

・當仲寛哲他 著
『ユニケージ原論』
USP研究所、2010年

UNIX哲学を実践して企業のシステム作る取り組みについて述べられています。プログラミングそのものというよりも、システム開発はいかようにあるべきかが論じられています。

プロセスを使いこなし、OS性能を引き出すために

第2章 シェルの動作原理を“深く”理解する

BSDコンサルティング(株)取締役／最高技術責任者
後藤 大地　GOTO Daichi　daichi@bsdconsulting.co.jp　Twitter ID：@daichigoto、@BSDc_tweet

シェルスクリプト上級者になるためには、OSの性能をフルに引き出せるようにならなければいけません。そのために本章では、普段、当たり前に書いているシェルスクリプトのコードで、カーネルがどう動作するのかを事細かに見ていきます。まずは、シェル実行時のシステムコール、プロセスの動きを理解しましょう。

表裏一体、シェルとカーネル

シェルは無駄なレイヤを持つことなく、ダイレクトにカーネルの機能を利用します。シェルスクリプトはほかのプログラミング言語と比べ、OSの性能をフルに発揮させるような処理を記述しやすく、ハードウェアリソースを使い切るようなプログラミングが簡単です。

しかしながら、シェルスクリプトをうまく使えるプログラマやユーザは、そう多くはありません。これはカーネルがどのように動作するのかを知らないためです。カーネルの動きがわかれば、シェルスクリプトも性能を発揮できる記述ができるようになります。

手続き型のプログラミング言語やオブジェクト指向型のプログラミング言語を使ってきたプログラマが、シェルスクリプトで適切ではない記述をしてしまいがちなのにも理由があります。シェルスクリプトでも手続き型言語やオブジェクト指向型言語のプログラミングを実施できないこともないのですが、あまり意味がありません。シェルスクリプトにはシェルスクリプトに適したやり方があります。

以降、シェルスクリプトにおけるある機能が、カーネルの内部でどのように動いているのかの説明を併用しながら、シェルスクリプトとその機能について紹介します。カーネルとしてはFreeBSD 9-STABLEソースコードを、シェルとしてはFreeBSD 9-STABLEに付随している/bin/shを使います。FreeBSDの/bin/shはashと呼ばれるシェルです。POSIX.1(POSIX:2008)で定められているシェルの機能を実装するとともに、若干の拡張機能が追加されています。

ashはbashと比較して処理が高速であるため、最近ではLinuxディストリビューションでも/bin/shの実体としてbashではなくdashと呼ばれる、ash由来の軽量シェルを採用するケースがあります。

#!/bin/shの意味

シェルスクリプトではファイルの先頭に必ず#!という文字を記述します。これは「シバン」または「シェバン」と呼ばれます。FreeBSDカーネルでは「シェルマジック(SHELLMAGIC)」と呼ばれています。

カーネルはプログラムの実行依頼を受けると、ファイルの種類を判別して、それに適したプログラムの起動を行います。ファイルの先頭が#!になっていた場合、そのプログラムはシェルスクリプトであると判定され、#!に続くパスのプログラムを起動して処理を行います。**#!/bin/sh**と記述されたファイルが実行されると、/bin/shが実行され、シェルスクリプトの内容が実行されます。

▼リスト1　SHEBANG.SH

```
#!/bin/sh
./SHEBANG2.SH
```

▼リスト2　SHEBANG2.SH

```
#!/bin/sh
```

▼リスト3　SHEBANG.SH用のDTraceスクリプト

```
syscall:freebsd:*exec*:entry    /execname == "sh"/ { printf("| %s %d %d syscall %s(¥"%s¥")", execname, pid, timestamp, probefunc, copyinstr(arg0)); }
syscall:freebsd:*exec*:return   /execname == "sh"/ { printf("| %s %d %d syscall %s() -> %d", execname, pid, timestamp, probefunc, arg1); }
fbt:kernel:exec_*_imgact:entry  /execname == "sh"/ { printf("| %s %d %d fbt %s()", execname, pid, timestamp, probefunc); }
fbt:kernel:exec_*_imgact:return /execname == "sh"/ { printf("| %s %d %d fbt %s() -> %d", execname, pid, timestamp, probefunc, arg1); }
profile:::tick-1sec             { exit(0); }
```

▼図1　SHEBANG.SH実行時のトレース

```
% sudo dtrace -s ./DTRACE.SHEBANG -c ./SHEBANG.SH              2>&1 | ./REP.SH
dtrace: script './DTRACE.SHEBANG' matched 15 probes
dtrace: pid 24224 has exited
CPU     ID                    FUNCTION:NAME
  1  56932                     execve:entry | sh 24225 100842280387335 syscall execve("./SHEBANG2.SH")
  1  17406          exec_elf64_imgact:entry | sh 24225 100842280409551 fbt exec_elf64_imgact()
  1  17407         exec_elf64_imgact:return | sh 24225 100842280412912 fbt exec_elf64_imgact() -> 4294967295  ←①
  1  30006          exec_shell_imgact:entry | sh 24225 100842280415938 fbt exec_shell_imgact()
  1  30007         exec_shell_imgact:return | sh 24225 100842280418623 fbt exec_shell_imgact() -> 0  ←②
  1  17406          exec_elf64_imgact:entry | sh 24225 100842280425732 fbt exec_elf64_imgact()
  1  17407         exec_elf64_imgact:return | sh 24225 100842280495053 fbt exec_elf64_imgact() -> 0  ←③
  1  56933                    execve:return | sh 24225 100842280512904 syscall execve() -> 0
%
                                  プログラム名  プロセスID  タイムスタンプ  プローブ種類  関数、システムコール、引数など
```

リスト1、2のシェルスクリプトを実行して実際にカーネルがどのように処理を行うのか調べます。

シェルおよびカーネル内部でどのように処理が進んでいるかはDTraceを使って調べるのが便利です。DTraceとはカーネルおよびユーザランドで動作するソフトウェアの内部の動作をトレースするためのしくみです。DTraceを使うと、カーネルやソフトウェアにデバッグコードを埋め込まずとも、ソフトウェア内部の動きを追うことができます。

利用するにはカーネルおよびソフトウェアをDTrace対応でビルドしておく必要があります。トレース内容はDと呼ばれるスクリプトで記述します。プログラミング言語としてのD言語と、DTraceで使用するスクリプトとしてのDは別ものですので、注意してください。FreeBSD 9.0からユーザランドDTrace機能も有効になっているため、シェルとカーネル双方の動きを追跡できます。#!の動作を調べるために、リスト3のDTraceスクリプトを用意します。

今回はシェルスクリプトの特集であってDTraceの特集ではないので、DTraceスクリプトや出力結果の説明はしません。ここでは、記述されているシステムコールや関数の動きをトレースできるものだ、程度に理解しておいてもらえればと思います。DTraceはとても便利な機能ですので、FreeBSD DTraceの特集もいずれあるんじゃないかと思います :-)

DTraceで動作をトレースしながらSHEBANG.SHを実行すると、図1の結果が得られます。CPU、ID、FUNCTION:NAME は DTrace が出力するデフォルトの情報です。プローブと呼ばれるもので、DTraceが追跡するものの最小単位のようなものだと考えてください。

それよりも右の値は、実行順序や実行内容を知るためにDスクリプトで出力させた内容です。プログラム名、プロセスID、実行時のタイムスタンプ、プローブの種類、実行した関数またはシステムコール、引数があるものは引数、戻

り値、などを表示させています。

SHEBANG.SHから生成されたサブシェルでexecve(2)が呼ばれ、SHEBANG2.SHを実行しようとします。execve(2)はプログラムを実行するためのシステムコールです。どのように動作するかはのちほど説明します。

execve(2)からexec_elf64_imgact()[注1]が実行され、SHEBANG2.SHが通常のバイナリファイルであるかどうか判定されます。判定結果はELF64バイナリではないという結果がでています(図1-①)。戻り値が429496295になっていますが、判定として真値が返ってきていないという点に注目してください。順次判定が実施されますが、ここで重要なのはexec_shell_imgact()[注2]でシェルスクリプトであるかどうか判定し、シェルスクリプトであると判定されていることです(図1-②)。exec_shell_imgact()の実体は/usr/src/sys/kern/imgact_shell.cに記述されています。ソースコードには先頭が#!であるか判定する記述があります(**リスト4**)。

図1のexec_shell_imgact()でシェルスクリプトであると判定されたため、次に/bin/shが実行可能なファイルであるかの判定がexec_elf64_imgact()で実施され、成功しています(図1-③)。/bin/shは通常のELF64の実行ファイルです。このように#!によるスクリプトの実行というのは、OSが提供する基本的な機能で、シェルスクリプトはこの機能を使って起動されています。

次からシェルの提供する基本的な機能について紹介します。

環境変数とシェル変数の違い、型のない世界

シェルスクリプトでは変数としてシェル変数と環境変数が利用できます。変数は=で代入する形式で設定します。シェル変数や環境変数には型という概念はありません。変数はただのデータの入れ物であり、値としては通常は文字列が指定されます。基本的に**リスト5、6**のように使用します。実行すると**図2**のようになります。

ここでは変数としてvおよびVを定義しています。それぞれaとAという値を割り当てています。リスト5-①でvがシェル変数、リスト5-②でVがシェル変数として定義されます。これら変数は$vおよび$Vとしてアクセスできます。

そしてリスト5-③でシェル変数Vを環境変数へ変更しています。環境変数はプロセスを超えて子プロセスへもコピーされるようになります。VAR.SHから呼び出されたVAR2.SHで変

注1) exec_elf64_imgact()はカーネル内部で実行される関数です。指定されたファイルがELF64形式であるかどうか判定し、ELF64形式である場合には実行処理へ移ります。ELF64とは64ビット版のELF実行形式です。バイナリ形式の実行ファイルはこういう形式のものなんだ、という程度に思っておいてたいだければと思います。ELF以外のバイナリ形式があるのか、ということになるわけですが、ELF以外の形式もあります。

注2) exec_shell_imgact()もカーネル内部で実行される関数です。対象がシェルスクリプトであるかどうか判定し、シェルスクリプトであればシェルスクリプトとして実行するための手順へ移ります。

▼リスト4　imgact_shell.cのソースコード

```
(略)
 40 #if BYTE_ORDER == LITTLE_ENDIAN
 41 #define SHELLMAGIC      0x2123 /* #! */
 42 #else
 43 #define SHELLMAGIC      0x2321
 44 #endif
(略)
110         /* a shell script? */
111         if (((const short *)image_
header)[0] != SHELLMAGIC)
112                 return (-1);
(略)
```

▼リスト5　VAR.SH

```
#!/bin/sh
v=a   ←①
V=A   ←②

export V    ←③

printf "shell $v $V¥n"
{ printf "{ $v $V }¥n"; }
( printf "( $v $V )¥n" )
./VAR2.SH
```

▼リスト6　VAR2.SH

```
#!/bin/sh
printf "shell $v $V¥n"
```

数vおよびVへアクセスを試みていますが、環境変数である変数Vは内容が引き継がれていることがわかります(図2-①)。

サブシェルに関してはのちほど説明しますが、このサブシェルにおける動作には注意が必要です。一見すると同じレベルにあるように見えますが、実際には別のプロセスで実行されていますi注3。たとえばリスト7のシェルスクリプトを実行し、リスト8のDTraceを適用して処理をトレースします。すると図3の結果が得られます。

図3-①のfork(2)とはプロセスをコピーする

注3) ただし、このあたりの処理はシェルごとに実装が異なるため、一概にそうとも言えません。詳しくは使っているシェルのソースコードを読む必要があります。

▼図2　VAR.SHの実行結果

```
% ./VAR.SH
shell a A
{ a A }
( a A )
shell  A   ←①
%
```

▼リスト7　VAR3.SH

```
#!/bin/sh
v=a
V=A
( printf "( $v $V )¥n" )
( printf "( $v $V )¥n" )
( printf "( $v $V )¥n" )
```

▼リスト8　VAR3.SH用のDTraceスクリプト

```
syscall:freebsd:fork:entry      /execname == "sh"/ { printf("| %s %d %d syscall %s()", ➥
execname, pid, timestamp, probefunc); }
syscall:freebsd:fork:return     /execname == "sh"/ { printf("| %s %d %d syscall %s() -> ➥
%d", execname, pid, timestamp, probefunc, arg1); }
syscall:freebsd:exec*:entry     /execname == "sh"/ { printf("| %s %d %d syscall ➥
%s(¥"%s¥")", execname, pid, timestamp, probefunc, copyinstr(arg0)); }
syscall:freebsd:exec*:return    /execname == "sh"/ { printf("| %s %d %d syscall %s() -> ➥
%d", execname, pid, timestamp, probefunc, arg1); }
syscall:freebsd:wait*:entry     /execname == "sh"/ { printf("| %s %d %d syscall %s()", ➥
execname, pid, timestamp, probefunc); }
syscall:freebsd:wait*:return    /execname == "sh"/ { printf("| %s %d %d syscall %s() -> ➥
%d", execname, pid, timestamp, probefunc, arg1); }
pid$target:sh:forkshell:entry   { printf("| %s %d %d function %s()", execname, pid, ➥
timestamp, probefunc); }
pid$target:sh:forkshell:return  { printf("| %s %d %d function %s() -> %d", execname, ➥
pid, timestamp, probefunc, arg1); }
profile:::tick-1sec             { exit(0); }
```

▼図3　VAR3.SH実行時のトレース

```
% sudo dtrace -s ./DTRACE.VAR -c ./VAR3.SH      2>&1 | ./REP.SH
dtrace: script './DTRACE.VAR' matched 8 probes
dtrace: pid 29661 has exited
CPU     ID                    FUNCTION:NAME
  2  70362                 expandarg:entry | sh 29661 153435546948403 function expandarg()
  2  70362                 expandarg:entry | sh 29661 153435546955817 function expandarg()
  2  70360                 forkshell:entry | sh 29661 153435546964117 function forkshell()
  2  56818                      fork:entry | sh 29661 153435546969464 syscall fork()    ←①
  2  56819                     fork:return | sh 29661 153435548163699 syscall fork() -> 29662    ←②
  2  70361                forkshell:return | sh 29661 153435548180339 function forkshell() -> 29662
  2  70360                 forkshell:entry | sh 29661 153435548331013 function forkshell()
  2  56818                      fork:entry | sh 29661 153435548333835 syscall fork()
  2  56819                     fork:return | sh 29661 153435548980575 syscall fork() -> 29663
  2  70361                forkshell:return | sh 29661 153435548994446 function forkshell() -> 29663
  2  70360                 forkshell:entry | sh 29661 153435549129318 function forkshell()
  2  56818                      fork:entry | sh 29661 153435549132011 syscall fork()
  2  56819                     fork:return | sh 29661 153435549973612 syscall fork() -> 29664
  2  70361                forkshell:return | sh 29661 153435549989890 function forkshell() -> 29664
%
```

システムコールです。UNIX系OSではfork(2)で自分自身のプロセスをコピーすることで新しいプロセスを生成するしくみを採用しています。最初は意味がわからないかもしれませんが、そういうものだと思っておいてください。詳細は後で説明します。

図3-①②で、サブシェルをfork(2)しています。図3-②から、生成されたプロセスのID番号は29662です。もとのシェルのプロセスIDは29661ですから、別のプロセスであることがわかります。

こうして、いったんサブシェルをfork(2)してからfork(2)された先のプロセス側で変数が展開されます。fork(2)によるコピーであるため、環境変数のみならずシェル変数も引き継がれます。

シェルによる関数

シェルスクリプトは基本的にコマンドを羅列するものです。シェルスクリプトではこのコマンドと同等の動きをする単位として関数という機能を提供しています。**関数名() {内容;}**で定義される処理単位で、同一プロセスIDのまで処理が走るという違いを除いて、コマンドと同様に動作します。

次のスクリプトを実行します。

```
#!/bin/sh
date() { printf("$1 $2 $3 $4 $5¥n"); }
date a b c
```

すると次のように動作します。

```
% ./FUNC.SH 1 2 3 4 5
a b c
%
```

このシェルスクリプトからは、次の2つのことがわかります。

- 関数はコマンドよりも優先される。システムにはdate(1)コマンドが用意されているが、関数として作成したdateのほうが優先して実行されている
- 関数に与えた引数は$1、$2、$3などでアクセスできる。シェルスクリプトに与えられた引数を参照する$1、$2、$3などの変数とは別ものとして扱われる

シェルスクリプトで関数を使うかどうかに関しては賛否両論あります。関数を作成するくらいであればコマンドとして別のシェルスクリプトにしたほうが良いとする考え方もあれば、関数としては単一ファイルにまとめたほうが良いとする考え方、よく利用する機能を関数にまとめたライブラリとしてのファイルを作ったほうが良いとする考え方などがあります。

▼リスト9　SCOPE.SH

```
#!/bin/sh
f1() { v=b; V=B; }
f2() { (v=c; V=C;) }
v=a
V=A

export V

printf "shell $v $V¥n"
f1
printf "shell $v $V¥n"
f2
printf "shell $v $V¥n"
```

▼図4　SCOPE.SHの実行結果

```
% ./SCOPE.SH
shell a A
shell b B  ←①
shell b B  ←②
%
```

変数のスコープ

シェルスクリプトにおける変数のスコープはもっとも理解が難しいところです。プログラミング言語としてのスコープというよりも、どのタイミングで変数が展開されるのかというパース＆展開のタイミングと、どのタイミングでfork(2)およびexecve(2)が実行されるのかという点に依存しています。

たとえば**リスト9**のシェルスクリプトを実行

すると、**図4**の結果が得られます。f1関数における変数への代入は機能していますが(図4-①)、f2関数における変数への代入は影響を与えていません(図4-②)。これはf2関数内部の処理がサブシェル、つまり別のプロセスで実行されているためです。別プロセス側で代入しても、もとのシェルスクリプトには影響を与えません。

fork(2)とexecve(2)

UNIX系OSでは、ユーザから見た場合に、リソースを使用する実体の単位はプロセスです。シェルの視点から見れば、シェルそのものが1つのプロセスであり、シェルを通じて実行されるコマンドそれぞれが1つ1つのプロセスです。

UNIX系OSでは、このプロセスがツリー構造をとるという特徴があります。プロセスにはすべて親子の関係があります。このため、プロセスの相互関係は1つの大きなツリー構造になります。この構造を実現するためのシステムコールがfork(2)とexecve(2)です。

システムコールとはカーネルに処理を依頼する特別な関数だと思ってください。ソフトウェアを安全に実行できるようにするため、CPUが提供する特定の処理はユーザランドのソフトウェアからは直接実行できないしくみになっています。

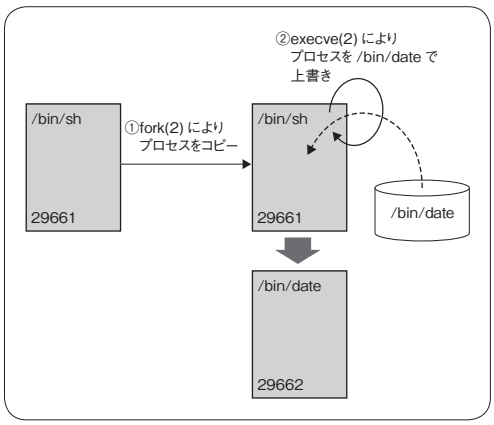
かわりにカーネルにその処理をやってほしいと依頼を出します。この依頼を出すための関数がシステムコールです。

fork(2)はプロセスをコピーして新しいプロセスを生成するためのシステムコール、execve(2)は指定したプログラムでプロセスそのものを上書きするためのシステムコールです。

fork(2)はとくにUNIX系OSを特徴付ける機能です。新しくプロセスを生成する(シェル的に言えばコマンドを実行する)場合、まずfork(2)システムコールを通じて自分自身をコピーします(**図5**)。

コピーした先のプロセスで、次にexecve(2)システムコールを実行して、自分自身を指定されたプログラムで上書きして、上書きしたプログラムを実行します。たとえば、シェルでdateコマンドが入力された場合、シェルはまず自分自身をコピーし、次にexecve(2)を呼び出して自分自身を/bin/dateで上書きして、上書きしたプログラムを実行します。

▼図5　コマンド実行時のプロセスの動き



いったんfork(2)システムコールでコピーを生成するため、fork(2)を実行した側が親プロセス、fork(2)でコピーされた側が子プロセスという親子関係が生まれます。環境変数はfork(2)およびexecve(2)の段階でも引き継がれるため、親プロセスで設定された環境変数は子プロセスにも持ち込まれます。

ps(1)コマンドを使うと**図6**のようにツリー構造を確認できます。そのまま表示すると読みにくいので、カーネルスレッドをps(1)の出力から排除しているほか、cut(1)でツリー構造の確認以外には関係ない部分は切り捨てて表示させています。一番最初に起動されるプロセスはinit(8)と呼ばれるプログラムで、ここではプロセス番号1が割り当てられています。すべてのプロセスはinit(8)をもっとも上の親としてツリー構造を構築します。

シェルスクリプトの動作を理解するというのは、fork(2)とexecve(2)がどのタイミングで実行されているのかを知ることだとも言えます。

▼図6　プロセスのツリー構造

```
% LANG=C ps -auxd -U root | fgrep -v '[' | grep -v 'X' | cut -c 1-10 -c 65-
USER   PID COMMAND
root     1 - /sbin/init --
root   670 |-- /usr/sbin/moused -p /dev/ums0 -t auto -I /var/run/moused.ums0.pid
root   163 |-- adjkerntz -i
root   687 |-- /sbin/devd
root   927 |-- /usr/sbin/syslogd -s
root   943 |-- /usr/sbin/rpcbind
root   970 |-- nfsuserd: master (nfsuserd)
root   971 | |-- nfsuserd: slave (nfsuserd)
root   972 | |-- nfsuserd: slave (nfsuserd)
root   973 | |-- nfsuserd: slave (nfsuserd)
root   974 | `-- nfsuserd: slave (nfsuserd)
root   996 |-- /usr/sbin/lpd
root  1020 |-- /usr/sbin/powerd
root  1065 |-- /usr/sbin/sshd
root  1068 |-- sendmail: accepting connections (sendmail)
root  1075 |-- /usr/sbin/cron -s
root  1176 |-- nfscbd: master (nfscbd)
root  1177 | `-- nfscbd: server (nfscbd)
root  1214 |-- /usr/libexec/getty Pc ttyv1
root  1215 |-- /usr/libexec/getty Pc ttyv2
root  1216 |-- /usr/libexec/getty Pc ttyv3
root  1217 |-- /usr/libexec/getty Pc ttyv4
root  1218 |-- /usr/libexec/getty Pc ttyv5
root  1219 |-- /usr/libexec/getty Pc ttyv6
root  1220 `-- /usr/libexec/getty Pc ttyv7
%
```

1つコマンドを実行するだけでも最低でも1回ずつはfork(2)とexecve(2)システムコールが実行されます。

プロセスの一生

シェルやシェルスクリプトでは「プロセス」または「フォアグラウンドプロセス」、「バックグラウンドプロセス」という言葉が使われます。フォアグラウンドプロセスとは実行が完了するまでシェルが待っているタイプのプロセス、バックグラウンドプロセスとは起動したらシェルと並列に処理が実行されるタイプのプロセスです。コマンドを実行する場合、行の最後に&を指定するとバックグラウンドプロセスとなり、それ以外はフォアグラウンドプロセスとなります。

カーネルにはフォアグラウンドプロセス、バックグラウンドプロセスという区別はありません。カーネルにとってみればどちらもただのプロセスです。フォアグラウンドプロセス、バックグラウンドプロセスの区別はシェルから見た場合に処理を変えるための違いでしかありません。

わかりやすく考えるとすれば、プロセスというのは基本的にすべてバックグラウンドプロセスのようなもので、フォアグラウンドプロセスのほうが特殊なパターンだと考えるとわかりやすいといえます。

リスト10、11のシェルスクリプトを使って動作の違いを追います。トレースには**リスト12**のDTraceスクリプトを使います。それぞれ実行すると**図7、8**の結果が得られます。

ash(/bin/sh)ではforkshell()という関数でシェルをfork(2)する処理が行われます。この関数を抜けたあとの動作に注目してください。フォアグラウンドプロセスとしてコマンドを実行した場合(つまりシェルスクリプトでよく使われる、普通にコマンドを実行した場合です)、そのあとwait4(2)システムコールを実行してfork(2)によっ

て生成されたプロセスの処理が終了するまで待ち処理が入っています(図7-①②)。

一方、バックグラウンドプロセスとして起動したほうでは、frok(2)したあとにwait4(2)システムコールは発行されていません(図8-①)。

つまりシェルやシェルスクリプトにおけるコ

▼リスト10　DATE.SH

```
#!/bin/sh
date
```

▼リスト11　DATE2.SH

```
#!/bin/sh
date &
```

▼リスト12　DATE.SHとDATE2.SH用のDTraceスクリプト

```
syscall:freebsd:fork:entry      /execname == "sh"/ { printf("| %s %d %d syscall %s()", ➥
execname, pid, timestamp, probefunc); }
syscall:freebsd:fork:return     /execname == "sh"/ { printf("| %s %d %d syscall %s() -> ➥
%d", execname, pid, timestamp, probefunc, arg1); }
syscall:freebsd:exec*:entry     /execname == "sh"/ { printf("| %s %d %d syscall ➥
%s(¥"%s¥")", execname, pid, timestamp, probefunc, copyinstr(arg0)); }
syscall:freebsd:exec*:return    /execname == "sh"/ { printf("| %s %d %d syscall %s() -> ➥
%d", execname, pid, timestamp, probefunc, arg1); }
syscall:freebsd:wait*:entry     /execname == "sh"/ { printf("| %s %d %d syscall %s()", ➥
execname, pid, timestamp, probefunc); }
syscall:freebsd:wait*:return    /execname == "sh"/ { printf("| %s %d %d syscall %s() -> ➥
%d", execname, pid, timestamp, probefunc, arg1); }
pid$target:sh:forkshell:entry   { printf("| %s %d %d function %s()", execname, pid, ➥
timestamp, probefunc); }
pid$target:sh:forkshell:return  { printf("| %s %d %d function %s() -> %d", execname, ➥
pid, timestamp, probefunc, arg1); }
profile:::tick-1sec             { exit(0); }
```

▼図7　DATE.SH実行時のトレース

```
% sudo dtrace -s ./DTRACE.PROCESS -c ./DATE.SH          2>&1 | ./REP.SH
dtrace: script './DTRACE.PROCESS' matched 9 probes
dtrace: pid 24254 has exited
CPU     ID                    FUNCTION:NAME
  0  70324                 forkshell:entry | sh 24254 100896816679129 function forkshell()
  0  56818                     fork:entry | sh 24254 100896816684234 syscall fork()
  0  56819                    fork:return | sh 24254 100896818064146 syscall fork() -> 24255
  0  70325                forkshell:return | sh 24254 100896818082667 function forkshell() -> 24255
  0  56828                     wait4:entry | sh 24254 100896818108290 syscall wait4()   ←①
  2  56932                    execve:entry | sh 24255 100896818112587 syscall execve("/bin/date")
  0  56829                    wait4:return | sh 24254 100896818960196 syscall wait4() -> 24255   ←②
%
```

▼図8　DATE2.SH実行時のトレース

```
% sudo dtrace -s ./DTRACE.PROCESS -c ./DATE2.SH         2>&1 | ./REP.SH
dtrace: script './DTRACE.PROCESS' matched 9 probes
dtrace: pid 24269 has exited
CPU     ID                    FUNCTION:NAME
  1  70326                 forkshell:entry | sh 24269 100896936656098 function forkshell()
  1  56828                     wait4:entry | sh 24269 100896936660869 syscall wait4()
  1  56829                    wait4:return | sh 24269 100896936662976 syscall wait4() -> -1
  1  56818                     fork:entry | sh 24269 100896936666722 syscall fork()
  1  56819                    fork:return | sh 24269 100896937850854 syscall fork() -> 24270
  1  70327                forkshell:return | sh 24269 100896937868664 function forkshell() -> 24270
  2  56932                    execve:entry | sh 24270 100896937985109 syscall execve("/bin/date")   ←①
%
```

マンドの実行とは、fork(2)→execve(2)している処理が終了するまでwait4(2)システムコールで待つという処理、ということになります。&を指定するとこのwait4(2)システムコールを発行して終了するまで待つという処理を飛ばします。

サブシェル

シェルスクリプトの動作を理解するうえでもっとも重要になるのがサブシェルです。どのタイミングでサブシェルを生成するのかという実装はシェルごとに多少異なりますので(ashではサブシェルを生成するようなケースでも、サブシェルを生成しないでコマンドを実行するようなシェルもあります)、ここではash(FreeBSDの/bin/sh)を対象として説明します。

サブシェルとはfork(2)で生成されるシェルのコピーを指します。シェルはコマンドを実行したりパイプを処理するときは、いったん自分自身のコピーをfork(2)システムコールを使って生成してから処理を実行します。シェルやシェルスクリプトを実行するというのは、fork(2)してシェルをコピーする処理をするといい変えてもよいほど、頻繁に自分をコピーします。

どのようなタイミングでfork(2)システムコールが実行されるのか調べるために、**リスト13**のDTraceスクリプトを使います。

まずはもっとも簡単な、シバンでシェルを指定したスクリプトのみに対してトレースを実施します。

・SUBSHELL.SH

```
#!/bin/sh
```

図9のようにfork(2)は検出されません。この段階では何もコピーされません。

グルーピングのみを記述します。

・SUBSHELL2.SH

```
#!/bin/sh
{ }
```

この場合も**図10**のように何も検出されません。パース処理のみが実行され、fork(2)する状況になっていないからです。

1つだけコマンドを実行させます。

▼リスト13　システムコール実行タイミング確認用のDTraceスクリプト

```
syscall:freebsd:fork:entry      /execname == "sh"/ { printf("| %s %d %d syscall %s()", ➥
execname, pid, timestamp, probefunc); }
syscall:freebsd:fork:return     /execname == "sh"/ { printf("| %s %d %d syscall %s() -> ➥
%d", execname, pid, timestamp, probefunc, arg1); }
syscall:freebsd:exec*:entry     /execname == "sh"/ { printf("| %s %d %d syscall ➥
%s(¥"%s¥")", execname, pid, timestamp, probefunc, copyinstr(arg0)); }
syscall:freebsd:exec*:return    /execname == "sh"/ { printf("| %s %d %d syscall %s() -> ➥
%d", execname, pid, timestamp, probefunc, arg1); }
pid$target:sh:forkshell:entry   { printf("| %s %d %d function %s()", execname, pid, ➥
timestamp, probefunc); }
pid$target:sh:forkshell:return  { printf("| %s %d %d function %s() -> %d", execname, ➥
pid, timestamp, probefunc, arg1); }
profile:::tick-1sec             { exit(0); }
```

▼図9　SUBSHELL.SH実行時のトレース

```
% sudo dtrace -s ./DTRACE.SUBSHELL -c ./SUBSHELL.SH     2>&1 | ./REP.SH
dtrace: script './DTRACE.SUBSHELL' matched 7 probes
dtrace: pid 24288 has exited
%
```

・SUBSHELL3.SH

```
#!/bin/sh
date
```

図11のようにfork(2)およびexecve(2)システムコールが実行されることを確認できます。シェルスクリプトのプロセスIDは24316です。fork(2)システムコールを呼ぶと、子プロセスとしてプロセスIDが24317のプロセスが生成されます(図11-①)。子プロセスでexecve("/bin/date")が実行され、date(1)コマンドが処理されていることがわかります(図11-②)。これがシェルがコマンドを実行するという処理の一連の流れです。

次にサブシェルを明示的に生成するケースを考えます。( )の指定は明示的にサブシェルを生成して処理を実行するというものです。( )の中に記述したコマンドは、生成された小プロセスにおいて評価され同様に処理が走ります。大本のシェルの変数などに影響を与えたくない場合などに使われます。

・SUBSHELL4.SH

```
#!/bin/sh
( )
```

実行すると図12の結果が得られます。fork(2)システムコールが呼ばれサブシェルが生成されていることがわかります(図12-①)。

そして重要なのが次のシェルスクリプトです。パイプはサブシェルを生成して実行されます(図13)。

・SUBSHELL5.SH

```
#!/bin/sh
date | cat
```

▼図10　SUBSHELL2.SH実行時のトレース

```
% sudo dtrace -s ./DTRACE.SUBSHELL -c ./SUBSHELL2.SH    2>&1 | ./REP.SH
dtrace: script './DTRACE.SUBSHELL' matched 7 probes
dtrace: pid 24302 has exited
%
```

▼図11　SUBSHELL3.SH実行時のトレース

```
% sudo dtrace -s ./DTRACE.SUBSHELL -c ./SUBSHELL3.SH    2>&1 | ./REP.SH
dtrace: script './DTRACE.SUBSHELL' matched 7 probes
dtrace: pid 24316 has exited
CPU     ID                    FUNCTION:NAME
  2  70332                 forkshell:entry | sh 24316 100935894212675 function forkshell()
  2  56818                      fork:entry | sh 24316 100935894217654 syscall fork()
  2  56819                     fork:return | sh 24316 100935895400357 syscall fork() -> 24317  ←①
  2  70333                forkshell:return | sh 24316 100935895415925 function forkshell() -> 24317
  0  56932                    execve:entry | sh 24317 100935895442188 syscall execve("/bin/date")  ←②
%
```

▼図12　SUBSHELL4.SH実行時のトレース

```
% sudo dtrace -s ./DTRACE.SUBSHELL -c ./SUBSHELL4.SH    2>&1 | ./REP.SH
dtrace: script './DTRACE.SUBSHELL' matched 7 probes
dtrace: pid 24331 has exited
CPU     ID                    FUNCTION:NAME
  0  70334                 forkshell:entry | sh 24331 100936008157510 function forkshell()
  0  56818                      fork:entry | sh 24331 100936008162499 syscall fork()
  0  56819                     fork:return | sh 24331 100936009339843 syscall fork() -> 24332  ←①
  0  70335                forkshell:return | sh 24331 100936009355860 function forkshell() -> 24332
%
```

▼図13　SUBSHELL5.SH実行時のトレース

```
% sudo dtrace -s ./DTRACE.SUBSHELL -c ./SUBSHELL5.SH    2>&1 | ./REP.SH
dtrace: script './DTRACE.SUBSHELL' matched 7 probes
CPU     ID                    FUNCTION:NAME
  0  70336                 forkshell:entry | sh 24346 100936122130467 function forkshell()
  0  56818                      fork:entry | sh 24346 100936122136628 syscall fork()
  1  56819                     fork:return | sh 24346 100936123349756 syscall fork() -> 24347
  1  70337                 forkshell:return | sh 24346 100936123367890 function forkshell() -> 24347
  0  56932                    execve:entry | sh 24347 100936123409274 syscall execve("/bin/date")
%
```

図13の出力からだとわかりにくいのですが、シェルはパイプラインが記載された処理をそれぞれサブシェルを生成して実行します。たとえば図13の例ですと、サブシェルがfork()で2個生成され、それぞれにおいてexecve(2)が実行されます。生成されたプロセスは標準入力と標準出力が接続されているので、データがコマンドからコマンドへ流れます。

このあたりはとても間違いが発生しやすいところですので注意してください。それまでリダイレクトで処理していたものを、パイプを挿(はさ)んでcat(1)コマンドで流し込むようにする、といったようなコーディングは頻繁に実施されますが、パイプラインを挿(はさ)み込んだ瞬間に、それはサブシェルで実行されることになります。これはwhileなどシェルが構文として用意している機能を使っている場合に、とくに問題となる部分です。問題の詳細については次章以降で解説します。SD

COLUMN ● 追実験用の環境構築方法

FreeBSD 9.0以降はユーザランドDTraceが搭載されていますので、今回の内容を追実験するには手軽な環境といえます。まず、**リスト14**のようなカーネルオプションを指定します。これは9.1以降を使う場合の指定です。

ユーザランドDTraceを使いたいので、/etc/make.confに**リスト15**の設定を追加しておきます。この状態でカーネルの再構築と再インストールを実施します。

dtrace(1M)の実行にはroot権限が必要です。rootで作業するか、sudoを使うなどして処理を行います。Makefileに処理内容をまとめておけば良いでしょう（**リスト16**）。

ここで1つのポイントとして、DTraceが出力する報告は時系列ではターミナルにはあがってこないということです。このため、DTraceでトレースする段階でタイムスタンプを出力させ、このタイムスタンプに合わせて出力内容を時系列にソートしてあげるシェルスクリプトを介して表示するようにします（**リスト17**）。こうすることで、DTraceの出力がより直感的に理解しやすいものになります。

DTraceは便利な機能です。FreeBSDでシステムを開発する場合や、アプリケーションを運用する場合のパフォーマンス引き上げの必要がある場合など、強力なツールとして活用できますので、活用を検討してみてください。

▼リスト14　カーネルオプションの指定

```
include         GENERIC
ident           DTRACE
options         KDTRACE_HOOKS
options         DDB_CTF
options         KDTRACE_FRAME
makeoptions     DEBUG="-g"
```

▼リスト15　/etc/make.confの設定

```
STRIP=
CFLAGS+=-fno-omit-frame-pointer
WITH_CTF=1
```

▼リスト16　Makefileに指定する処理内容

```
shebang:
        sudo dtrace -s ./DTRACE.SHEBANG -c ./SHEBANG.SH         2>&1 | ./REP.SH

process:
        sudo dtrace -s ./DTRACE.PROCESS -c ./DATE.SH            2>&1 | ./REP.SH
        sudo dtrace -s ./DTRACE.PROCESS -c ./DATE2.SH           2>&1 | ./REP.SH

subshell:
        sudo dtrace -s ./DTRACE.SUBSHELL -c ./SUBSHELL.SH       2>&1 | ./REP.SH
        sudo dtrace -s ./DTRACE.SUBSHELL -c ./SUBSHELL2.SH      2>&1 | ./REP.SH
        sudo dtrace -s ./DTRACE.SUBSHELL -c ./SUBSHELL3.SH      2>&1 | ./REP.SH
        sudo dtrace -s ./DTRACE.SUBSHELL -c ./SUBSHELL4.SH      2>&1 | ./REP.SH
        sudo dtrace -s ./DTRACE.SUBSHELL -c ./SUBSHELL5.SH      2>&1 | ./REP.SH

redirect:
        sudo dtrace -s ./DTRACE.REDIRECT -c ./REDIRECT.SH       2>&1 | ./REP.SH
        sudo dtrace -s ./DTRACE.REDIRECT -c ./REDIRECT2.SH      2>&1 | ./REP.SH
        sudo dtrace -s ./DTRACE.REDIRECT -c ./REDIRECT3.SH      2>&1 | ./REP.SH
        sudo dtrace -s ./DTRACE.REDIRECT -c ./REDIRECT4.SH      2>&1 | ./REP.SH
        sudo dtrace -s ./DTRACE.REDIRECT -c ./REDIRECT5.SH      2>&1 | ./REP.SH
        sudo dtrace -s ./DTRACE.REDIRECT -c ./REDIRECT6.SH      2>&1 | ./REP.SH
        sudo dtrace -s ./DTRACE.REDIRECT -c ./REDIRECT7.SH      2>&1 | ./REP.SH

var:
        sudo dtrace -s ./DTRACE.VAR -c ./VAR3.SH        2>&1 | ./REP.SH
```

▼リスト17　REP.SH

```
#!/bin/sh

tmp=/tmp/$$

cat                                     > $tmp-src
grep -E '(dtrace)|(ID)' $tmp-src        > $tmp-head
grep '|' $tmp-src                       > $tmp-data

cat $tmp-data                           |
grep -v '^$'                            |
sort -k7,7                              |
cat                                     > $tmp-data2

cat $tmp-head $tmp-data2
printf "¥n"

rm $tmp-*
```

第3章 入力元／出力先を巧みに切り替える シェルスクリプトがファイル入出力に強いわけ

BSDコンサルティング(株)取締役／最高技術責任者
後藤 大地　GOTO Daichi　daichi@bsdconsulting.co.jp　Twitter ID：@daichigoto、@BSDc_tweet

シェルは標準入出力という概念があることで、非常にシンプルな記述でファイルやデバイスとの入出力を実現したり、コマンド同士を連携したりできるのが特徴です。本章ではその原理を見ていきます。また、ファイルI/Oにおけるキャッシュの効果についても、実際に測定しながら検証します。

ファイル記述子と入出力切り替え

シェルが得意とする処理はプロセスの生成とプロセスをパイプで接続して処理すること、そしてファイルの入出力です。どちらも余計な処理を入れず、ダイレクトにシステムコールが呼ばれます。簡単なシェルスクリプトを通じて、実際にどのようにシステムコールが実行されているのか調べます。

標準出力のリダイレクト

ファイルの入出力の一番最初にはopen(2)システムコールが使われます。入出力の切り替えにはdup2(2)システムコールが使われます。**リスト1**のDTraceスクリプトを用意してシェルの動作を追います。

次のシェルスクリプトを実行します。open(2)システムコールを呼ぶだけのシェルスクリプトです。

・REDIRECT.SH

```
#!/bin/sh
: > /dev/null
```

実行すると**図1**の結果が得られます。/dev/nullを1537というフラグ指定でopen(2)し(図1-①)、3というファイル記述子を得ています(図1-②)。フラグはシステムコールや関数の引数として　渡される値で、どのように動作するのかといった内容を指定するものです。open(2)システムコールを実行すると、ファイル記述子(ファイルディスクリプタ)と呼ばれる数字が返ってきます。ファイル記述子とは、OSがファイルを識別するために使う番号のことで、以降は、この数字を指定してread(2)/write(2)を実行することで、このファイルへの読み書きとなります。

実際にこのシェルスクリプトを実行することで処理されるashのソースコードは**リスト2**の行が該当します。open(2)のフラグとしてO_WRONLY、O_CREAT、O_TRUNCを加算したものが指定されています。書き込みモードで開き、ファイルが存在しなければ新規作成、ファイルが存在している場合には中身を全部消して先頭から利用、という指定です。

フラグに定義されている数値は**表1**のとおりです。フラグを指定するときは、これらの値を足した値で指定します。今回の場合は、1 + 512 + 1024 = 1537ということで、DTraceで補足した数値に一致します。

open(2)したあとにdup2(2)システムコールで、標準出力を/dev/nullへ差し替えています(図1-③)。

ファイル記述子として0、1、2はすでに予約済みです。0が標準入力、1が標準出力、2が標準エラー出力です(**表2**)。

▼リスト1　入出力切り替え確認用のDTraceスクリプト

```
syscall:freebsd:open:entry
/execname != "sh" && copyinstr(arg0) != "/dev/null" &&
                     copyinstr(arg0) != "/dev/stdout"/
{
        of=0;
}

syscall:freebsd:open:entry
/execname == "sh" &&
 (copyinstr(arg0) == "/dev/null" ||
  copyinstr(arg0) == "/dev/stdout")/
{
        printf("| %s %d %d syscall %s(¥"%s, %d¥")",
                execname, pid, timestamp, probefunc, copyinstr(arg0), arg1);
        of=1;
}

syscall:freebsd:open:return
/of == 1/
{
        printf("| %s %d %d syscall %s() -> %d",
                execname, pid, timestamp, probefunc, arg1);
}

syscall:freebsd:dup2:entry
/execname == "sh" && arg0 < 5/
{
        printf("| %s %d %d syscall %s(%d, %d)",
                execname, pid, timestamp, probefunc, arg0, arg1);
}

profile:::tick-1sec             { exit(0); }
```

▼リスト2　REDIRECT.SH実行時に処理されるashのコード

```
open(fname, O_WRONLY|O_CREAT|O_TRUNC, 0666)
```

▼図1　REDIRECT.SH実行時のトレース

```
% sudo dtrace -s ./DTRACE.REDIRECT -c ./REDIRECT.SH     2>&1 | ./REP.SH
dtrace: script './DTRACE.REDIRECT' matched 5 probes
dtrace: pid 24378 has exited
CPU     ID                    FUNCTION:NAME
  2  56824                       open:entry | sh 24378 100986542342790 syscall open("/dev/null, 1537")  ←①
  2  56825                      open:return | sh 24378 100986542358382 syscall open() -> 3  ←②
  2  56994                       dup2:entry | sh 24378 100986542363164 syscall dup2(3, 1)  ←③
```

▼表1　open(2)のフラグ一覧

open(2)フラグ	値(16進数)	値(10進数)	説明
O_RDONLY	0x0000	0	読み込みのみ許可して開く
O_WRONLY	0x0001	1	書き込みのみ許可して開く
O_CREAT	0x0200	512	ファイルが存在しない場合には新規作成
O_TRUNC	0x0400	1024	ファイルが存在する場合、内容を全部削除して先頭から書き込む
O_APPEND	0x0008	8	ファイルが存在する場合、ファイルの最後へ追記していく

▼表2　予約済みのファイル記述子

ファイル記述子番号	意味
0	標準入力
1	標準出力
2	標準エラー出力

このようにファイルに出力するという処理は、open(2)システムコールでファイルを開き、dup2(2)システムコールで標準入出力を開いたファイルへ向けるという処理です。

標準入力のリダイレクト

次に、ファイルへの出力ではなく、ファイルから入力する場合をトレースします。次のシェルスクリプトで/dev/nullから入力を受けるという指定になります。

・REDIRECT2.SH

```
#!/bin/sh
: < /dev/null
```

実行すると**図2**の結果が得られます。open(2)のフラグに0が指定されています。0はO_RDONLYです(図2-①)。ashのソースコードとしては**リスト3**の行が該当します。O_RDONLYを指定してオープン処理が行われています。

/dev/nullを開いてファイルディスクリプタ番号3が返ってきています(図2-②)ので、dup2(3, 0)という指定で、標準入力が/dev/nullに置き換わっていることがわかります(図2-③)。open(2)システムコールのフラグやdup2(2)システムコールで指定する番号が変わっているだけで、ファイルへの書き込みもファイルからの読み込みの指定も、処理の流れは同じです。

▼リスト3　REDIRECT2.SH実行時に処理されるashのコード

```
open(fname, O_RDONLY)
```

標準エラー出力のリダイレクト

次に、出力を標準エラー出力へ向ける指定をトレースします。

・REDIRECT3.SH

```
#!/bin/sh
: 2> /dev/null
```

実行すると**図3**の結果が得られます。ファイルをオープンする場合と処理の流れは同じで、dup2(2)する対象が標準出力である1から標準エラー出力である2に変わっているという違いがわかります(図3-①)。

2>という指定は1>と組み合わせて次のように使われたりします。エラー出力は/dev/nullへ捨てて、エラーではない出力だけを得たいといった場合や、またはエラー出力だけをファイルに落としたいといった場合に使われます。

▼図2　REDIRECT2.SH実行時のトレース

```
% sudo dtrace -s ./DTRACE.REDIRECT -c ./REDIRECT2.SH    2>&1 | ./REP.SH
dtrace: script './DTRACE.REDIRECT' matched 5 probes
dtrace: pid 24392 has exited
CPU     ID                    FUNCTION:NAME
  0  56824                       open:entry | sh 24392 100986646956320 syscall open("/dev/null, 0")  ←①
  0  56825                      open:return | sh 24392 100986646968334 syscall open() -> 3  ←②
  0  56994                       dup2:entry | sh 24392 100986646971440 syscall dup2(3, 0)  ←③
%
```

▼図3　REDIRECT3.SH実行時のトレース

```
% sudo dtrace -s ./DTRACE.REDIRECT -c ./REDIRECT3.SH    2>&1 | ./REP.SH
dtrace: script './DTRACE.REDIRECT' matched 5 probes
dtrace: pid 24406 has exited
CPU     ID                    FUNCTION:NAME
  1  56824                       open:entry | sh 24406 100986750624735 syscall open("/dev/null, 1537")
  1  56825                      open:return | sh 24406 100986750639260 syscall open() -> 3
  1  56994                       dup2:entry | sh 24406 100986750642860 syscall dup2(3, 2)  ←①
%
```

・REDIRECT4.SH

```
#!/bin/sh
: 2> /dev/null 1> /dev/stdout
```

実行すると**図4**のように2>と1>の処理が随時実行されていることがわかります(図4-①②)。>という表記は1>という表記の1を省略したものです。

この手の書き方でよく使われるのが2>&1です。>&はdup2(2)システムコールを呼び出す指定です。2>&1という指定で、dup2(1, 2)を実行するという意味になります。次のシェルスクリプトを実行して動きをトレースします。

・REDIRECT5.SH

```
#!/bin/sh
: 2>&1
```

図5のようにdup2(2)システムコールが呼び出され、標準エラー出力が標準出力へ差し代わることを確認できます(図5-①)。標準エラー出力を捨てることなく拾いたい場合に使われます。

>&で指定する数字の順序が逆になるパターンもあります。1>&2で、dup2(2, 1)が実行されますので、標準出力が標準エラー出力へ差し代わります。ワーニングやエラー出力などを実施したい場合に使われます。

・REDIRECT6.SH

```
#!/bin/sh
: 1>&2
```

図6のような実行結果が得られます。2>&1や1>&2は魔法の指定方法のように説明されることがありますが、これはdup2(2)を呼び出すための指定であり、内部のしくみがわかっていれば適切に活用できる機能です。

▼図4　REDIRECT4.SH実行時のトレース

```
% sudo dtrace -s ./DTRACE.REDIRECT -c ./REDIRECT4.SH    2>&1 | ./REP.SH
dtrace: script './DTRACE.REDIRECT' matched 5 probes
dtrace: pid 24420 has exited
CPU     ID                    FUNCTION:NAME
  0  56824                       open:entry | sh 24420 100986854326166 syscall open("/dev/null, 1537")
  0  56825                      open:return | sh 24420 100986854339528 syscall open() -> 3
  0  56994                       dup2:entry | sh 24420 100986854342888 syscall dup2(3, 2)  ←①
  0  56824                       open:entry | sh 24420 100986854346728 syscall open("/dev/stdout, 1537")
  0  56825                      open:return | sh 24420 100986854357840 syscall open() -> 3
  0  56994                       dup2:entry | sh 24420 100986854359596 syscall dup2(3, 1)  ←②
%
```

▼図5　REDIRECT5.SH実行時のトレース

```
% sudo dtrace -s ./DTRACE.REDIRECT -c ./REDIRECT5.SH    2>&1 | ./REP.SH
dtrace: script './DTRACE.REDIRECT' matched 5 probes
dtrace: pid 24434 has exited
CPU     ID                    FUNCTION:NAME
  2  56994                       dup2:entry | sh 24434 100986957953029 syscall dup2(1, 2)  ←①
%
```

▼図6　REDIRECT6.SH実行時のトレース

```
% sudo dtrace -s ./DTRACE.REDIRECT -c ./REDIRECT6.SH    2>&1 | ./REP.SH
dtrace: script './DTRACE.REDIRECT' matched 5 probes
dtrace: pid 24448 has exited
CPU     ID                    FUNCTION:NAME
  0  56994                       dup2:entry | sh 24448 100987062414716 syscall dup2(2, 1)
%
```

追記モードでの標準出力のリダイレクト

これまでの指定は基本的にファイルに書き込む段階でO_TRUNCが指定されていますので、ファイルの内容は削除されます。削除せずに追記したい場合には>>を使います。

・REDIRECT7.SH

```
#!/bin/sh
: >> /dev/null
```

実行すると**図7**の結果が得られます。open(2)のフラグとして521が指定されていることがわかります(図7-①)。

このオープン処理に対応するシェルのソースコードは**リスト4**のように記述されています。O_WRONLY が 1、O_CREAT が 512、O_APPENDが8ですので、1 + 512 + 8 = 521で、トレース結果の521と一致します。O_APPENDを指定することで、ファイルの内容は削除されず、一番後ろに追記することになります。

このように、シェルにおけるファイル入出力の扱いはカーネルの提供するシステムコールに直結しています。ほかのレイヤが間に挟まることはなく、ほぼダイレクトにカーネルの機能を使用しています。シェルスクリプトがこのあたりの操作を得意としているのは、こうしたダイレクトな実装になっているという理由があります。

キャッシュファイルとI/O

シェルスクリプトで処理をする場合、キャッシュについて知っておいたほうが便利です。キャッシュを活用したファイルを使えるようになると、処理速度が見違えて高速になります。

最近のOS／カーネルであれば、使われていない主記憶メモリがあれば、なるべくディスクキャッシュとして活用しようとします。少なくともFreeBSDやLinuxのカーネルはそのように実装されています。キャッシュに載ったデータは、ディスクにアクセスすることなくキャッシュからアクセスされるようになるため、動作速度がとても高速になります。

▼図7　REDIRECT7.SH実行時のトレース

```
% sudo dtrace -s ./DTRACE.REDIRECT -c ./REDIRECT7.SH    2>&1 | ./REP.SH
dtrace: script './DTRACE.REDIRECT' matched 5 probes
dtrace: pid 24462 has exited
CPU     ID                    FUNCTION:NAME
  2  56824                       open:entry | sh 24462 100987167993502 syscall open("/dev/null, 521")  ←①
  2  56825                      open:return | sh 24462 100987168009618 syscall open() -> 3
  2  56994                       dup2:entry | sh 24462 100987168016069 syscall dup2(3, 1)
%
```

▼リスト4　REDIRECT7.SH実行時に処理されるashのコード

```
open(fname, O_WRONLY|O_CREAT|O_APPEND, 0666)
```

▼リスト5　CACHE.SH

```
#!/bin/sh

tmp=/tmp/$$
dd if=/dev/zero of=$tmp-out bs=1024x1024x10 count=100     ←①
dd if=/dev/zero of=$tmp-out2 bs=1024x1024x10 count=2000   ←②
/usr/bin/time -lph cat $tmp-out > /dev/null     ←③
/usr/bin/time -lph cat $tmp-out > /dev/null     ←④
rm $tmp-*
```

▼表3　CACHE.SH実行時の計測結果

	1回目	2回目
実行時間	1.94秒	0.23秒
システム時間	0.59秒	0.23秒
ユーザ時間	0秒	0秒
ブロック入力回数	8014回	0回
自発的コンテキストスイッチ回数	2698回	3回
強制コンテキストスイッチ回数	270回	30回

これは**リスト5**のようなシェルスクリプトで確認できます。実行に使用したマシンは16GBのメモリを搭載しています。最初のdd(1)で1GBのファイルを作成し(リスト5-①)、2つ目のdd(1)で20GBのファイルを作成しています(リスト5-②)。1つ目のdd(1)で作成したファイルは作成時にはキャッシュに載っていますが、2つ目のdd(1)が実行されている間に、キャッシュから排除されます。

このあとで、cat(1)を使って最初に作成したファイルにアクセスするという処理を2回続けます(リスト5-③④)。1回目はキャッシュが効かない状態、2回目はキャッシュが効いた状態でcat(1)が実行されることになります。

実行すると**図8**の結果が得られます。これを整理したのが**表3**です。最初は1.94秒かかっているcat(1)コマンドの処理(図8-①)が、2つ目では0.23秒(図8-③)と、8倍から9倍の高速化が観測されます。処理の内訳を見てみると、1回目のcat(1)ではディスクからデータを読み込むためのブロック入力回数が8,014回発生しているのに対し(図8-②)、2回目は0回になっていることがわかります(図8-④)。つまりデータがキャッシュに載ったため、2度目のアクセスではディスクまでデータの読み込みにいっていないということになります。1回目はディスクアクセスが発生しているため自発的なコンテキストスイッチの回数が多いこともわかります。

最近のマシンは主記憶メモリのサイズが大きいので、このようにキャッシュを意識してプログラミングを組めるようになると、何億件といったテキストデータも高速に処理できるようになります。データがキャッシュに載っているかどうかは目で確認しにくいところがあります。この場合、tmpfs(5)やmfs(5)といったメモリファイルシステムを使って明示的にメモリ上に配置するというやり方があります。SD

▼図8　CACHE.SHの実行結果

```
% ./CACHE.SH
100+0 records in
100+0 records out
1048576000 bytes transferred in 2.632610
secs (398302812 bytes/sec)
2000+0 records in
2000+0 records out
20971520000 bytes transferred in 71.960708
secs (291430151 bytes/sec)
real 1.94     ←①
user 0.00
sys 0.59
       1736  maximum resident set size
         12  average shared memory size
       2203  average unshared data size
        138  average unshared stack size
        133  page reclaims
          0  page faults
          0  swaps
       8014  block input operations     ←②
          0  block output operations
          0  messages sent
          0  messages received
          0  signals received
       2698  voluntary context switches
        270  involuntary context switches
real 0.23     ←③
user 0.00
sys 0.23
       1736  maximum resident set size
         12  average shared memory size
       2040  average unshared data size
        128  average unshared stack size
        133  page reclaims
          0  page faults
          0  swaps
          0  block input operations  ←④
          0  block output operations
          0  messages sent
          0  messages received
          0  signals received
          3  voluntary context switches
         30  involuntary context switches
%
```

第4章 品質だって気を付けたい シェルスクリプトのエラーハンドリングとデバッグ

(有)ユニバーサル・シェル・プログラミング研究所　http://www.usp-lab.com
當仲 寛哲　TOUNAKA Nobuaki　tounaka@usp-lab.com

“ソフトウェアの品質”という言葉には2つの側面があります。1つは、プログラムの間違い(バグ)がないこと、もう1つは、動作するプログラムにトラブルが起こったときでも、安全に対処できるということです。本章ではシェルスクリプトのエラー処理の方法とデバッグの方法について説明します。

シェルスクリプトの品質

シェルスクリプトは他の言語によるプログラムに比べて、間違い(バグ)を起こしにくいと言われています。それは、1つ1つのコマンドが、長年あるいは世界中の人に使われているうちに、標準的な使い方において、バグが皆無になっていることが大きな原因です。シェルスクリプトは、これら、“完成された間違いのないコマンド”を順番に起動しているだけのプログラミングです。シェルスクリプトにおけるバグの原因は、コマンドの使い方の間違いとか、起動の順番の間違いなど、初歩的な要因に限られることが多いです。

このようにバグが比較的少ないシェルスクリプトですが、万一バグによって動作が異常になる場合、どのような状態になるのでしょうか。

シェルスクリプトの異常は、常に“エラー”によって検出されます。そしてシェルスクリプトのエラーの起こり方は2つしかありません。それはシェル(本章ではbashを扱います)自身が報告するエラーと、各コマンドが報告するエラーです。そしていずれの場合も、エラーと呼ばれるものの実体はシェルやコマンドが終了時に返す終了値であり、この値は都度シェル変数“$?”にセットされるというしくみになっています。

つまり基本的にはシェル変数“$?”をウォッチしていれば、トラブルが起こったことが直ちにわかり、その値に応じて対処すれば良いという非常にシンプルなしくみによって、シェルスクリプトの品質は保たれているのです。

エラーハンドリング

終了ステータス／パイプステータス

エラーハンドリングの基本は、シェルや各コマンドが終了時にシェル変数“$?”にセットする値を調べることから始まります。ではどのように調べたら良いかご存じでしょうか？　それはコマンドを実行した直後にシェル変数“$?”の値をechoコマンドを使ってシェル変数の値を表示してみれば良いのです(**図1**)。

0が出力されましたね。これは正常終了したls(1)が、エラー情報(この場合は正常終了情報

▼図1　正常終了時の$?の値

```
$ ls -l exist-file      ← 何かコマンドを実行してみる(exist-fileが存在するとき)
-rw-rw-r-- 1 usp usp 229 12月  8 13:35 exist-file
$ echo $?               ← 直後にシェル変数$?を表示する
0                       ← 0が出力されます
```

である値0)をシェルに返し、シェルがシェル変数$?にこの値をセットしたのです。

シェル変数$?は、コマンドを実行するたびにその値が、直前に実行されたコマンドの終了値で上書きされてしまいます。ですからそのコマンドのエラーを調べたいときは、必ずコマンド実行の直後に調べることが大切です。

それでは、コマンドがエラー終了した場合はどうなるでしょうか(**図2**)。

今度はシェル変数$?は2になりました。このように、コマンドがエラーを起こす場合は、終了値が0以外になります。エラーの値は1～255になりますが、どの値になるかは、各コマンドの仕様に任されています。

このことは思わぬ誤解を生む場合があります。たとえば、文字列を検索するgrep(1)というコマンドの動作です(**図3**)。

文字列"Linux"の検索の場合と文字列"Windows"の検索の場合とで違う結果がでました。これはgrep(1)コマンドが指定文字列が見つからなかったとき、終了値を1とする(つまりエラー)ように作られているからです。このように、人が一般にエラーと思うこと(=予期せぬこと)が起こることと、各コマンドがエラーだと思うことにずれがある場合があります。ですからシェルプログラミングにおいては、各コマンドがどのような場合にエラー(終了値が0以外)を返すのか、マニュアルなどで知っておく必要があります。

ここまでの例は、コマンドが返すエラーでしたが、シェル自身が返すエラーはどうでしょうか(**図4**)?

エラーメッセージをよく見ると、内部コマンドのエラーの場合も、シェル(bash)がエラーメッセージを出力していますね。

それではコマンドが組み合わさった場合はど

▼図2　エラー時の$?の値

```
$ ls -l non-exist-file  ← コマンドが失敗するようにしてみる
ls: non-exist-file: No such file or directory
$ echo $?               ← 直後にシェル変数$?を表示する
2                       ← 0でない値(2)が出力されます
```

▼図3　grep(1)コマンドでの$?値の変

```
$ cat OS-file       ← OS の名前が記述してあるファイル
Linux
FreeBSD
Solaris
AIX
$ grep Linux OS-file > result     ← Linux という文字列を検索してみる
$ echo $?
0
$ grep Windows OS-file > result   ← Windows という文字列を検索してみる
$ echo $?
1
```

▼図4　シェルのエラーによる$?値の変化

```
$ bad_command         ← 存在しないコマンドを起動してみる
-bash: bad_command: command not found ← シェル自身がエラーメッセージを出力
$ echo $?
127
$ cd bad_directory    ← シェルの内部コマンドがエラー終了する場合
-bash: cd: bad_directory: No such file or directory
$ echo $?
1
```

うなるでしょうか。必ず正常終了する(終了値＝0となる)true(1)と、必ずエラー終了する(終了値＝1となる)false(1)を使って実験してみましょう(**図5**)。

このように順次起動の場合は、最後に実行したコマンドの終了値がシェル変数$?にセットされます。正確には各コマンドの実行直後にシェル変数$?がセットされ、コマンドが順次実行されるたびに値が上書きされていくので、結果として最終コマンドの終了値がシェル変数$?にセットされます。

パイプラインの場合はどうでしょうか(**図6**)。

これは順次起動と同じ結果で、最後に実行したコマンドの終了値となりますが、途中のコマンドの終了値は一度もシェル変数$?にセットされることはありません。

bashの場合は、配列変数PIPESTATUSに各コマンドの終了値がセットされます(**図7**)。これを使えば、パイプで連結されたいずれかのコマンドがエラーを起こした場合を知ることができます。

図8のように与えられた引数をすべて足す関数plusを定義します。このような関数を作っておけば、**図9**のように、パイプで連結されたコマンド群の終了値の合計を1つの終了値とできます。

最後に、シェルスクリプトの最後に記述するexitコマンドですが、その引数はシェルスクリプトが返す終了値になります(**図10**)。

-eオプション

エラーのなんたるかはここまでの説明でおわかりいただけたと思います。それでは、シェル

▼図5　true(1)コマンドとfalse(1)コマンド実行後の$?値

```
$ true; false; true   ← ; を使ってコマンドを順次起動してみる
$ echo $?
0
$ true; true; false
echo $?
1
```

▼図6　パイプライン実行時の$?値

```
$ true | false | true
$ echo $?
0
$ true | true | false
$ echo $?
1
```

▼図7　配列変数PIPESTATUSの値

```
$ true | false | true
$ echo ${PIPESTATUS[@]}   ← 配列変数の場合、すべての要素を
0 1 0                       出力するには配列変数名[@]とします
```

▼図8　コマンドラインによるplus関数の定義

```
$ function plus () {
> n=0
> for var in "$@"; do
>   n=$((n+var))
> done
> return $n
> }
$ plus 1 2 3      ← 1+2+3を終了値にセットする
$ echo $?
6
```

▼図9　パイプライン結果をplus関数によって加算する

```
$ false | true | false
$ plus ${PIPESTATUS[@]}   ← 各コマンドの終了値を
$ echo $?                   足して1つの終了値にする
2
```

▼図10　シェルスクリプトの中にexitが入っている場合

```
$ cat sample1
#!/bin/bash

date
exit 0              ← "0"が終了値になる
$ ./sample1
Sat Dec  8 14:49:01 JST 2012
$ echo $?
0                   ← exitで指定した終了値がシェル変数$?にセットされる
```

スクリプトがエラーを起こしたとき、シェルスクリプトの実行をその場で止めるにはどうしたら良いでしょうか。

bashには“-e”というオプションがあります。シェルスクリプトの冒頭に、

```
#!/bin/bash -e
```

と記述することによって、シェル変数$?の値が0以外(つまりエラー)になったとき、その時点でシェルを終了します。実験として、まず“-e”オプションを付けないで、途中でエラーが発生するシェルスクリプトを走らせてみましょう(**図11**)。

次に“-e”オプションをつけてみます(**図12**)。見事にエラーを起こしたコマンドの直後でシェルスクリプトは停止します。そしてシェルスクリプトの終了値は、エラーを起こしたコマンドの終了値と等しくなります。

パイプで連結されている場合は、シェル変数$?の値はパイプライン最後のコマンドの終了値になるので、ただ-eオプションを付けるだけでは、パイプラインの途中で起こったエラーを検知できません(**図13**)。

これを乗り切るために、先出のplus関数を定義して、各コマンドラインの直後に“plus ${PIPESTATUS[@]}”の記述を挿入します(**図14**)。

パイプを使わない単独のコマンドであっても、PIPESTATUSの値がセットされることに注意してください。この場合は、配列の要素数が1となり、${PIPESTATUS[0]}の値は$?の値と同じになります。

つまり、単独コマンドであろうが、パイプで連結したコマンド群であろうが、直後に、“plus ${PIPESTATUS[@]}”という記述を挿入しておけば、-eオプションによって、コマンドのエラー時にただちに終了できます。

シグナル、トラップ

エラーが起こったときに、ただちに終了するのではなくて、ある処理を行ってから終了させたい場合があります。たとえば標準的なエラー

▼図11　エラーが発生するシェルスクリプトに-eをつけない場合

```
$ cat sample2
#!/bin/bash
true;  echo pass1
false; echo pass2
true;  echo pass3
exit 0

$ ./sample2
pass1
pass2
pass3
```

▼図12　エラーが発生するシェルスクリプトに-eをつけた場合

```
$ cat sample3
#!/bin/bash -e     ← “-e”を付ける
true;  echo pass1
false; echo pass2
true;  echo pass3
exit 0

$ ./sample3
pass1              ← falseコマンドを実行した直後に終了する
$ echo $?
1                  ← falseコマンドの終了値がシェルスクリプトの
                     終了値になり、それがシェル変数$?にセットされる
```

▼図13　パイプラインの途中エラーは検知できない

```
$ cat sample4
#/bin/bash -e

true | false | true; echo pass1  ← falseはパイプ途中のコマンドなので
true; echo pass2                   終了値がシェル変数$?にセットされない

$ ./sample4
pass1
pass2    ← 停止せず通り過ぎてしまう
```

メッセージを出したり、終了値を統一したいときなどです。

このようなときには、“trap <関数名> ERR”の記述が便利です。本来“trap”は外部からの割り込みに対して反応する関数を宣言するものですが、擬似シグナル“ERR”を使って、シェルスクリプト内で発生するエラーに対して反応する関数を宣言できます(**図15**)。

-eを外すことにより、エラーが発生するたびにエラーハンドラを実行させることができます(**図16**)。

この方法の良いところは記述が簡単なことですが、エラーハンドラに引数を渡すことができないのが難点です。このときは、**図17**のようにあらかじめグローバル変数にエラー情報をセットしておくというやり方があります。

このようなやり方が可能なのは、シェルスクリプトの変数は常にグローバル変数だからです。22行目でセットした、シェル変数“lineno message code”の値をエラーハンドラ“handler”の中で使用できます。

trapを使わず、通常の関数で定義する場合は**図18**のようにします。

このサンプルでは、関数“handler”で配列変数“PIPESTATUS”の各要素の値を合計し、それが0でなければ、引数を出力しています。9行目のn=0の記述をコメントアウトしているのは、もしこれを入れるとn=0という代入処理が実行されることにより、以前のPIPESTATUSの値が変化してしまうからです。2行目で、nの初期値を0にしています。この関数は1回呼ばれるとシェルスクリプト自身が終了するので、関数が複数回呼ばれることにより、nの値が加算されることが実質ないことを利用しているという意味で、技巧的過ぎるかもしれません。

▼図14　plus ${PIPESTATUS[@]}を追加する

```
$ cat sample5
#!/bin/bash -e

function plus ()
{
    n=0
    for var in "$@"; do
        n=$((n+var))
    done
    return $n
}

true | false | true; plus ➥
${PIPESTATUS[@]}; echo pass1
true; plus ${PIPESTATUS[@]}; echo ➥
pass2

$ ./sample5
$
$ echo $?
1
```

▼図15　エラー後処理の指定

```
$ cat sample6
#!/bin/bash -e          ← エラーが起こったら終了させる

trap handler ERR        ← 終了させる前に実行する関数を宣言する

function handler ()
{
    echo error occurred ← エラーメッセージの表示
    return 3            ← どんなエラーでも終了値を3にする
}

true;   echo pass1
false;  echo pass2      ← false でエラーが発生する
false;  echo pass3

$ ./sample6
pass1
error occurred
$ echo $?
3
```

▼図16　-eを外して実行する

```
$ ./sample7   ( ソース省略 )
pass1
error occurred
pass2
error occurred
pass3
```

▼図17　グローバル変数にエラー情報をセット

```
$ cat -n sample8
     1  #!/bin/bash -e
     2
     3  trap handler ERR
     4
     5  function handler ()
     6  {
     7      echo LINE:$lineno MESSAGE:$message
     8      return $code
     9  }
    10
    11  function plus ()
    12  {
    13      n=0
    14      for var in "$@"; do
    15          n=$((n+var))
    16      done
    17      return $n
    18  }
    19
    20  lineno=$LINENO; message="error1" code=1
    21  true; plus ${PIPESTATUS[@]}; echo pass1
    22  lineno=$LINENO; message="error2" code=2
    23  false; plus ${PIPESTATUS[@]}; echo pass2
    24  lineno=$LINENO; message="error3" code=3
    25  false; plus ${PIPESTATUS[@]}; echo pass3

$ ./sample8
pass1
LINE:22 MESSAGE:error2
$ echo $?
2
```

▼図18　通常関数での定義

```
$ cat -n sample9
     1  #!/bin/bash                          ← -e をいれない
     2  n=0                                  ← ここで n=0
     3  # エラーハンドラ関数
     4  #   第1引数：行番号
     5  #   第2引数：エラーメッセージ
     6  #   第3引数：終了コード
     7  function handler ()
     8  {
     9          # n=0                        ← あえて n=0 としない
    10          for var in "${PIPESTATUS[@]}"; do
    11                  n=$((n+var))
    12          done
    13          [ $n -eq 0 ] && return 0
    14          echo LINE:$1 MESSAGE:$2
    15          exit $3
    16  }
    17
    18  true;  handler $LINENO "error1" 1; echo pass1
    19  false; handler $LINENO "error2" 2; echo pass2
    20  true;  handler $LINENO "error3" 3; echo pass3

$ ./sample9
pass1
LINE:19 MESSAGE:error2
$ echo $?
2
```

デバッグ

シェルスクリプトの品質を向上させるために、開発時におけるデバッグは必須です。しかし、シェルスクリプトには一般的な統合開発環境のようなものはほとんど存在しません。それでは、シェルスクリプト開発においてデバッグはどのように行うのでしょうか。

基本はcat

シェルプログラミングで特徴的なのは、処理するデータを常にファイルの中に納めて、ファイルの操作によって、やりたいことを実現することにあります。シェルプログラミングにおいても、データを変数に納めて、通常のコンピュータ言語のように記述することもできますが、制約やあいまいな点が多く、処理スピードもかなり遅いので、思い切って、

「シェルプログラミング＝ファイル操作のコマンドを並べる」

と割り切ることが肝心です。たとえば、奇数行だけ演算する簡単なプログラムを、変数型の普通プログラミング風とシェルスクリプトらしいプログラムの2種類で記述してみます(**図19**)。

それでは10万行の4列のテキストデータをサンプルで作ってみます(**図20**)。

それぞれを実行させてみます(**図21**)。

実行時間が約27倍違います。プログラムの行数もsample10は6～17行目の実質12行、sample11は6～8行目の実質3行になります。プログラムの読みやすさもsample10が奇数行だけを取り出す仕掛けと各項目の演算の記述が入り組んでいるのに対して、sample11は入力ファイルから奇数行だけとっていったん作業ファイルに出力して作業ファイルに演算だけを施し、結果ファイルに出力というたいへんわかりやすい構造になっています。

このように、シェルプログラミングは、データをすべてファイルに納めて、やりたいことをそのまま順番に記述し、それぞれの処理を入力ファイルから出力ファイルへ施すだけで、速い、

▼図19　シェルスクリプトらしさの比較

```
$ cat sample10
     1 #!/bin/bash
     2 #
     3 # 奇数行だけ演算するプログラム(普通プログラミング風)
     4 #
     5
     6 n=0                                # 行カウンタを初期化
     7 cat data              |
     8 while read a b c d; do             # 各行の項目を変数にセット
     9   n=$((n+1))                       # 行カウンタのインクリメント
    10   [ $((n%2)) -eq 0 ] && continue   # 偶数行をスキップする
    11   a2=$((a+1))                      # 各項目の演算と変数への代入
    12   b2=$((b*2))                      # 各項目の演算と変数への代入
    13   c2=$((a+b+c))                    # 各項目の演算と変数への代入
    14   d2=$((d-1))                      # 各項目の演算と変数への代入
    15   echo $a2 $b2 $c2 $d2             # 変数の出力
    16 done                 > data2
    17 exit 0

$ cat sample11
     1 #!/bin/bash
     2 #
     3 # 奇数行だけ演算するプログラム(シェルらしいプログラム)
     4 #
     5
     6 sed -n '1~2p' data                          > work   # 奇数行だけ取り出す
     7 awk '{print $1+1,$2*2,$1+$2+$3,$4-1}' work > data2  # 演算する
     8 exit 0
```

短い、わかりやすいの3つのメリットを引き出すことができるのです。

シェルプログラミングのデバッグがcat(1)だというのは、シェルプログラミングの作法にのっとってプログラムすれば、すべてデバッグは入力ファイルや出力ファイルの中身をcat(1)で確認することに尽きるという意味なのです。

たとえばsample11において、作業ファイルworkをcat(1)してみると、**図22**のようになり、確かに奇数行だけ取り出されていることがわかります。

tee

シェルプログラミングにおいて、各コマンドをパイプでつなぐことにより、

❶一連の処理を作業ファイルを発生させることなく簡潔に記述できる
❷パイプラインで接続された各コマンドはOSによって自動的に並列実行され、処理効率・スピードが向上する

というメリットがあります。

sample11とまったく同じ処理をするsample12(**図23**)がさらに見やすく、スピードもアップしました。このようにパイプライン接続は非常にメリットがあるのですが、デバッグという観点からすると、作業ファイルがないので各コマンドの入力ファイル、出力ファイルの中身を確認できません。このような場合、どう対処すれば良いでしょうか。その答えは、tee(1)です。

パイプラインの中途に、“tee <作業ファイル名>”という記述を挟み込むことにより、パイプラインを遮ることなく、パイプを流れるデータを作業ファイルにコピーして取りおきすることができます(**図24**)。デバッグはこのようにして取り出した作業ファイルの中身をcat(1)で確認することによって行います。

“-xv” “+xv”

シェルスクリプトにはデバッグにおいて、強力な機能が備わっています。それは走行ログの

▼図20　10万行の4列のテキストデータを作る

```
$ seq 100000 | tee a b c d > /dev/null
$ paste -d ' ' a b c d > data
$ cat data
1 1 1 1
2 2 2 2
3 3 3 3
...
100000 100000 100000 100000
```

▼図21　図19のシェルスクリプトの実行結果

```
$ time ./sample10
real    0m4.251s
user    0m3.561s
sys     0m0.674s

$ time ./sample11
real    0m0.153s
user    0m0.143s
sys     0m0.009s
```

▼図22　結果の確認

```
$ cat work | head
1 1 1 1
3 3 3 3
5 5 5 5
7 7 7 7
9 9 9 9
11 11 11 11
13 13 13 13
15 15 15 15
17 17 17 17
19 19 19 19
```

▼図23　sample11を改良する

```
$ cat sample12
     1 #!/bin/bash
     2 #
     3 # 奇数行だけ演算するプログラム(パイプを使って高速化)
     4 #
     5
     6 sed -n '1~2p' data                    |     # 奇数行だけ取り出す
     7 awk '{print $1+1,$2*2,$1+$2+$3,$4-1}' > data2  # 演算する
     8 exit 0

$ time ./sample12
real    0m0.140s
user    0m0.142s
sys     0m0.009s
```

出力機能です。走行ログというのは、シェルスクリプトが実行されているとき、リアルタイムで今どのコマンドが実行中なのかを表示する機能です。走行ログを表示するには、

-x オプション：現在実行しているコマンドを表示する
-v オプション：現在読み込んでいるシェルスクリプトの部分を表示する

の2つのオプションを使います。

sample12の冒頭の部分を“#!/bin/bash -x”に書き換えて実行してみます(**図25**)。

その瞬間に実行されているコマンドが“+”記号とともにリアルタイムで順次表示されます。冒頭を“#!/bin/bash -v”に書き換えると、**図26**のようにシェルスクリプトそのものが、実行の進行具合に合わせてリアルタイムで順次表示されます。

“-x”と“-v”オプションを同時に指定することにより、それぞれの走行ログが混ざって出力されますが、実行されているコマンドと実行されているシェルスクリプト上の位置が同時にわかるので、デバッグするときの参考になります。

数多い繰り返し構文などで、一時的に-xオプションをやめたいときは、スクリプトの中で

▼図24　tee(1)によって中間結果を取り出す

```
$ cat sample13
     1 #!/bin/bash
     2 #
     3 # 奇数行だけ演算するプログラム（パイプを使って高速化+デバッグ）
     4 #
     5
     6 sed -n '1~2p' data                      |        # 奇数行だけ取り出す
     7  tee work                                |        # デバッグ用ファイル
     8 awk '{print $1+1,$2*2,$1+$2+$3,$4-1}' > data2  # 演算する
     9 exit 0
```

▼図25　sample12の実行表示（#!/bin/bash -xの場合）

```
$ ./sample12
+ sed -n 1~2p data
+ awk '{print $1+1,$2*2,$1+$2+$3,$4-1}'
+ exit 0
```

▼図26　sample12の実行表示（#!/bin/bash -vの場合）

```
$ ./sample12
#!/bin/bash -v
#
# 奇数行だけ演算するプログラム（パイプを使って高速化+デバッグ）
#

sed -n '1~2p' data                      |        # 奇数行だけ取り出す
awk '{print $1+1,$2*2,$1+$2+$3,$4-1}' > data2  # 演算する
exit 0
```

▼図27　リアルタイム表示をスイッチする

```
#!/bin/bash -xv

set +x      ← -x オプションの抑制
for((i=0; i<10000; i++)); do
  echo $i   ← echo コマンドが1000回実行されるので、-x を抑制しないと
done            10000 行の走行ログが出力されてしまう。
set -x      ← -x オプションの復活
```

“set +x”とします。再び-xオプションを有効にしたいときは、“set -x”とします。

exec 2>

走行ログを目視ではなく、ファイルに保存したい場合はどうすれば良いでしょうか。それには“exec コマンド”を使います。もともと、execコマンドは“exec <コマンド名>”として使用し、現在実行しているシェルのプロセスのテキスト空間を指定したコマンドで置き換えるというコマンドですが、めったにこの機能を使用することはないでしょう。

```
$ exec 2> log
```

とすることで、現在実行しているシェルの標準エラー出力を指定ファイルにリダイレクトします。走行ログはシェルの標準エラー出力に出力されているので、“exec 2> log”という記述をシェルスクリプトの冒頭にすることで走行ログをファイルに保管できます(**図28**)。

このスクリプトを実行すると、“exec 2> log”記述以降の走行ログがファイルlogに保管されます(**図29**)。

-xvオプションを付けているので、実行コマンドのログ(+コマンド名)と読み込んだシェルスクリプトのログが混在してログファイルに記述されます(**図30**)。

このような手法により、自動起動されたシェルスクリプトやバックグラウンド起動されたシェルスクリプトも、走行ログを簡単にログファイルに保管することができ、途中でエラー終了したシェルスクリプトのエラー発生位置を特定できます。SD

▼図28　走行ログをファイルに保管

```
$ cat sample14
#!/bin/bash -xv    ← 走行ログを標準エラー出力に出す
#
# 奇数行だけ演算するプログラム
#

exec 2> log                                    # 走行ログをファイルに出す
sed -n '1~2p' data                     |       # 奇数行だけ取り出す
awk '{print $1+1,$2*2,$1+$2+$3,$4-1}' > data2  # 演算する
exit 0
```

▼図29　“exec 2> log”記述以降の走行ログが保管される

```
$ ./sample14            ← 初めの走行ログは画面(標準エラー出力)に出る
#!/bin/bash -xv
#
# 奇数行だけ演算するプログラム
#

exec 2> log                                    # 走行ログをファイルに出す
+ exec                  ← exec実行以降は ファイル“log”へ 標準エラー出力がリダイレクトされる
```

▼図30　実行コマンドのログとシェルスクリプトのログを混在

```
$ cat log
sed -n '1~2p' data                     |       # 奇数行だけ取り出す
awk '{print $1+1,$2*2,$1+$2+$3,$4-1}' > data2  # 演算する
+ sed -n 1~2p data
+ awk '{print $1+1,$2*2,$1+$2+$3,$4-1}'
exit 0
+ exit 0
```

第5章 基本だけど奥が深い パイプのしくみを読み解く

BSDコンサルティング(株)取締役／最高技術責任者
後藤 大地　GOTO Daichi daichi@bsdconsulting.co.jp　Twitter ID：@daichigoto、@BSDc_tweet

シェルをもっともシェル足らしめている機能のひとつがパイプライン(以下パイプと略)です。本章ではシェルから呼ばれたパイプがどのように処理されているかを解説します。

表裏一体：パイプとカーネル

シェルでは、コマンドとコマンドの間に「|」を挟むことで、「|」の左側に記述したコマンドの標準出力を、右側に記述したコマンドの標準入力に接続できます。パイプは連続して使用することができ、20個、30個といったコマンドをパイプで接続して処理することができます。

シェルのパイプは、pipe(2)システムコールが生成するペアのディスクリプタを、それぞれのコマンドの標準出力および標準入力に対してdup2(2)システムコールで入れ替えるという処理です。パイプは、ダイレクトにpipe(2)システムコールおよびdup2(2)システムコールを使った処理に置き換わり、ほぼオペレーティングシステムが提供するネイティブな機能を使ったものといえます。

シェルやシェルスクリプトはマルチコア／メニーコアを活用するためのもっとも簡単で、もっとも効率の良い方法です。オペレーティングシステムはプロセスをそれぞれコアに割り当てて動作させます。バックグラウンドプロセスとして並列処理させることもできますし、パイプで接続して相互通信させた状態で並列処理させることもできます。

スレッドとの違いは、メモリ空間を共有するかどうかにあります。メモリ空間を共有した方がよい場合にはスレッドが便利ですが、共有メモリを使えばプロセスでも同様の処理を実現できます。プロセスはマルチコア／メニーコアを活用するためのもっとも基本的で、かつ、開発効率のよい方法の1つです。

パイプの正体：パイプでコマンドはどう起動されるか

pipe(2)システムコールは2つのファイル記述子を生成するためのシステムコールです。pipe(2)システムコールが生成するファイル記述子はペアになっており、片方のファイル記述子に書き込んだデータは、もう片方のファイル記述子を経由して取り出すことができます。この操作はプロセスを超えて機能します。

FIFOファイル(または名前付きパイプ)を使うと、2つのプロセスの間でデータのやり取りが可能になりますが、pipe(2)はこれをファイルシステムの名前空間を使用せずに実現するものです。シェルはpipe(2)システムコール、fork(2)システムコール、execve(2)システムコール、dup2(2)システムコールを組み合わせて、コマンドの出力を接続して動作させます。

実際にどのような手順で機能しているのかは、シェルにトレースコードを仕込んで調べることができます。FreeBSD ash(/bin/sh)のソースコードは/usr/src/bin/sh/以下にまとまっています。pipe(2)、fork(2)、execve(2)、dup2(2) は eval.c、

exec.c、jobs.cファイルで使われていますので、関係するところにトレースコードを追加します。

図1〜3(オリジナルの拡張子に.orgに変更してあります)のようなコードを適用します。

▼図1　diff -u eval.c.org eval.cの結果(先頭＋が追加行)

```
--- /usr/src/bin/sh/eval.c.org  2012-12-11 18:24:47.000000000 +0900
+++ /usr/src/bin/sh/eval.c  2012-12-12 12:33:31.000000000 +0900
@@ -44,6 +44,7 @@
 #include <unistd.h>
 #include <sys/resource.h>
 #include <sys/wait.h> /* For WIFSIGNALED(status) */
+#include <sys/time.h>
 #include <errno.h>

 /*
@@ -541,11 +542,23 @@
                close(prevfd);
                error("Pipe call failed: %s", strerror(errno));
            }
+pid_t mypid = getpid();
+struct timeval t;
+gettimeofday(&t, NULL);
+int64_t timestamp = t.tv_sec * 1000 * 1000 + t.tv_usec;
+out1fmt("sh %d %ld pipe():evalpipe -> %d %d¥n", mypid, timestamp, pip[0], pip[1]);
+flushall();
        }
        if (forkshell(jp, lp->n, n->npipe.backgnd) == 0) {
            INTON;
            if (prevfd > 0) {
                dup2(prevfd, 0);
+pid_t mypid = getpid();
+struct timeval t;
+gettimeofday(&t, NULL);
+int64_t timestamp = t.tv_sec * 1000 * 1000 + t.tv_usec;
+out1fmt("sh %d %ld dup2(%d, %d):evalpipe¥n", mypid, timestamp, prevfd, 0);
+flushall();
                close(prevfd);
            }
            if (pip[1] >= 0) {
@@ -553,6 +566,12 @@
                    close(pip[0]);
                if (pip[1] != 1) {
                    dup2(pip[1], 1);
+pid_t mypid = getpid();
+struct timeval t;
+gettimeofday(&t, NULL);
+int64_t timestamp = t.tv_sec * 1000 * 1000 + t.tv_usec;
+out1fmt("sh %d %ld dup2(%d, %d):evalpipe¥n", mypid, timestamp, pip[1], 1);
+flushall();
                    close(pip[1]);
                }
            }
```

※eval.cはコマンドラインに入力されたコマンドや引数などを展開した結果を評価するためのプログラムです。この部分でpipe(2)やfork(2)などが実行されます。

▼図2　diff -u exec.c.org exec.cの結果（先頭＋が追加行）

```
--- /usr/src/bin/sh/exec.c.org  2012-12-11 18:23:23.000000000 +0900
+++ /usr/src/bin/sh/exec.c  2012-12-12 11:11:01.000000000 +0900
@@ -153,6 +153,20 @@
    ssize_t n;
    char buf[256];

+// PID
+pid_t mypid;
+mypid = getpid();
+
+// TIMESTAMP
+struct timeval t;
+int64_t timestamp;
+gettimeofday(&t, NULL);
+timestamp = t.tv_sec * 1000 * 1000 + t.tv_usec;
+
+out1fmt("sh %d %ld execve(%s, **argv, **envp)¥n", mypid, timestamp, cmd);
+flushall();
+
+printf("sh ");
    execve(cmd, argv, envp);
    e = errno;
    if (e == ENOEXEC) {
```

※exec.cでexecve(2)システムコールが実行され、コマンドラインで指定されたコマンドが実行されます。

▼図3　diff -u jobs.c.org jobs.cの結果（先頭＋が追加行）

```
--- /usr/src/bin/sh/jobs.c.org  2012-12-11 18:22:47.000000000 +0900
+++ /usr/src/bin/sh/jobs.c  2012-12-12 11:09:52.000000000 +0900
@@ -785,6 +785,18 @@
        checkzombies();
    flushall();
    pid = fork();
+// PID
+pid_t mypid;
+mypid = getpid();
+
+// TIMESTAMP
+struct timeval t;
+int64_t timestamp;
+gettimeofday(&t, NULL);
+timestamp = t.tv_sec * 1000 * 1000 + t.tv_usec;
+
+out1fmt("sh %d %ld forkshell() -> %d¥n", mypid, timestamp, pid);
+flushall();
    if (pid == -1) {
        TRACE(("Fork failed, errno=%d¥n", errno));
        INTON;
```

※jobs.cはジョブ制御に関するプログラムです。ashのソースコードはパーサ部分は複雑で追うのがたいへんですが、それ以外のコードは比較的理解しやすい内容になっています。

これをmakeコマンドでビルドします。

```
# /usr/src/bin/sh
# make
```

「cat /dev/null | head | sort」コマンドを実行して、このときのpipe(2)、fork(2)、execve(2)、dup2(2)の実行手順を確認すると**図4**のようになります。ここでも実行の順序を把握するために、タイムスタンプに対してsort(1)で整列をかけます。

この出力結果から、次のことがわかります。

❶シェルは入力されたコマンドにパイプがあることを確認すると、そのパイプの個数だけpipe(2)システムコールを実行してファイル記述子のペアを生成する
❷コマンドはそれぞれfork(2)システムコールを使ってサブシェルが生成され、処理はそちらに移る
❸生成されたそれぞれのサブシェルの内部でdup2(2)システムコールを実行し、標準入力または標準出力、またはその双方をpipe(2)で生成されたファイル記述子へ置き換える
❹execve(2)システムコールを実行して、指定されたコマンドを実行する

簡単にまとめると次のようになります。

❶pipe(2)でパイプ(ファイル記述子のペア)を生成
❷fork(2)でサブシェルを生成
1. dup2(2)で標準入出力をパイプへ連結
2. execve(2)でコマンドを実行

この結果、それぞれのコマンドの入出力が連結され、一連のデータの流れとして処理されることになります。このあたりの処理は実装にもよるので、ash以外では別の手順で処理しているかもしれませんが、大枠としてこのような仕組みになっていると思っておいて良いと思います。

パイプとファイル記述子

「cat /dev/null | head | sort」というコマンドを実行した場合、pipe(2)システムコールは2回呼ばれます。最初のpipe(2)で3、4のファイル記述子のペアが生成されます。2つ目のpipe(2)システムコールでは4、5のファイル記述子のペアが生成されます(**図5**)。

pipe(2)システムコールが生成するペアのファイル記述子には方向性があります。pip[0]は読み込み専用のディスクリプタ、pip[1]は書き込

▼図4 「cat /dev/null | head | sort」の実行手順の表示

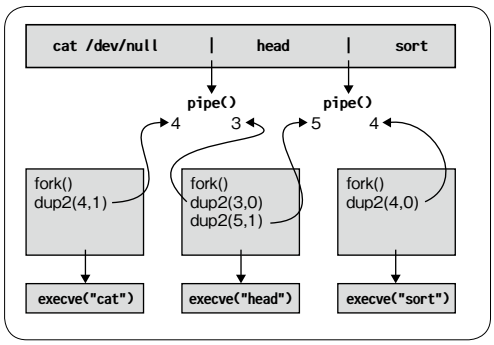
```
% ./sh -c 'cat /dev/null | head | sort' | sort -k3
sh 7691 1355284038009881 pipe():evalpipe -> 3 4
sh 7691 1355284038012322 forkshell() -> 7693
sh 7693 1355284038012524 forkshell() -> 0
sh 7691 1355284038014204 pipe():evalpipe -> 4 5
sh 7693 1355284038015686 dup2(4, 1):evalpipe
sh 7691 1355284038015686 forkshell() -> 7694
sh 7694 1355284038015873 forkshell() -> 0
sh 7693 1355284038016568 execve(/bin/cat, **argv, **envp)
sh 7691 1355284038019607 forkshell() -> 7695
sh 7694 1355284038019757 dup2(3, 0):evalpipe
sh 7695 1355284038019800 forkshell() -> 0
sh 7694 1355284038019958 dup2(5, 1):evalpipe
sh 7694 1355284038020558 execve(/usr/bin/head, **argv, **envp)
sh 7695 1355284038024122 dup2(4, 0):evalpipe
sh 7695 1355284038024802 execve(/usr/bin/sort, **argv, **envp)
%
```

み専用のディスクリプタです。今回のケースでは、1つ目のdup2(2)で生成される3と4では、3が読み込み専用で4が書き込み専用となります。2つ目のdup2(2)で生成される4と5に関しては、4が読み込み専用で、5が書き込み専用です。

シェルの「|」の機能は、このファイル記述子をdup2(2)で結びつけるという機能ということになります。標準入力は0で、標準出力は1ですので、1つ目のpipe(2)システムコール実行の後に、dup2(3,0)とdup2(4,1)と実行することで、コマンドの出力結果が次のコマンドの入力へ流れることになります。

pipe(2)システムコールを実行して、ペアのファイル記述子を作成する処理はシェルが実行しますが、dup2(2)システムコールを使って入出力をパイプに割り当てる処理は、それぞれのサブシェルが実行しています。こうすることでデータが流れる状況を整えたあとで、execve(2)システムコールを実行してコマンドを起動しています。コマンドはすでにデータが流れる状況ができた後で実行されます。これがパイプ機能です。

▼図5 「cat /dev/null | head | sort」実行時の処理の流れ



パイプでマルチプロセス

パイプで接続されたコマンド(プロセス)はそれぞれが個別にコアに割り当てられますので、プロセッサの性能を使い切る用途に向いています。たとえば、**図6**のマシン(NEC Express 5800 R120d-1M)でマルチプロセスを実行します。32論理コア(8コア/12スレッドのIntel Xeon CPU E5-2690を2基搭載)のマシンです。

処理単位として**リスト1**のスクリプトを用意します。/dev/nullに対して書き込み要求を発生させるスクリプトです。実際にIOは発生せず、fcntl(2)システムコール、open(2)システムコール、write(2)システムコール、close(2)システムコールが繰り返しコールされる処理になります。

図7のようにCMD.SHを32個並列で処理させます。

top(1)コマンドでプロセスの処理状況を見ると、次のようにそれぞれのコマンドが別々のコアに割り当てられて実行されていることがわか

▼図6 今回実行するマシン

```
% kenv smbios.system.product
Express5800/R120d-1M [N8100-1791Y]
% sysctl hw.model hw.ncpu
hw.model: Intel(R) Xeon(R) CPU E5-2690 0
@ 2.90GHz
hw.ncpu: 32
%
```

▼図7 CMD.SHをパイプで32個つなぐ

```
% ./CMD.SH | ./CMD.SH | ./CMD.SH | ./CMD.SH |
  ./CMD.SH | ./CMD.SH | ./CMD.SH | ./CMD.SH |
  ./CMD.SH | ./CMD.SH | ./CMD.SH | ./CMD.SH |
  ./CMD.SH | ./CMD.SH | ./CMD.SH | ./CMD.SH |
  ./CMD.SH | ./CMD.SH | ./CMD.SH | ./CMD.SH |
  ./CMD.SH | ./CMD.SH | ./CMD.SH | ./CMD.SH |
  ./CMD.SH | ./CMD.SH | ./CMD.SH | ./CMD.SH |
  ./CMD.SH | ./CMD.SH | ./CMD.SH | ./CMD.SH &
%
```

▼リスト1 CMD.SH

```
#!/bin/sh
while :
do
        printf "¥n" > /dev/null
done
```

ります(**図8**)。

CMD.SHはパイプを経由してデータが流れることがないので、パイプではなくバックグラウンドプロセスとして起動しても同じです。バックグラウンドプロセスとして起動した場合、それぞれのプロセスの独立性が強くなります。パイプで接続して起動した場合、特定のデータに対する処理を複数のコマンドに割り振って処理させるといったことが容易に実現できます。

たとえば、**図9**のような32列の数値データで構成された1億行のデータを処理するケースを考えます。

それぞれの列の値に1を加算した結果を得るシェルスクリプトを用意します(**リスト2**)。

このシェルスクリプトは単一のawk(1)コマンドで処理を済ませています。1プロセスでの処理です。**リスト3**のシェルスクリプトは処理

▼図9　DATAファイルの形式

```
% wc -l DATA
 100000000 DATA
% head -2 DATA
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 ↵
0 1 2 3 4 5 6 7 8 9 0 1
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 ↵
0 1 2 3 4 5 6 7 8 9 0 1
%
```

▼図8　プロセスの処理状況

```
last pid:  2283;  load averages: 18.05,  5.54,  2.10    up 0+00:29:57  17:56:52
59 processes:  31 running, 28 sleeping
CPU:  0.6% user,  0.0% nice, 68.0% system,  0.0% interrupt, 31.4% idle
Mem: 23M Active, 12M Inact, 1000M Wired, 13M Buf, 123G Free
Swap: 4096M Total, 4096M Free

  PID USERNAME  THR PRI NICE   SIZE    RES STATE   C   TIME   WCPU COMMAND
 2260 daichi      1  52    0 14504K  2132K RUN    22   0:57 80.37% sh
 2273 daichi      1  52    0 14504K  2132K CPU12  13   0:58 79.49% sh
 2257 daichi      1  52    0 14504K  2132K CPU21  21   0:55 79.39% sh
 2274 daichi      1  52    0 14504K  2132K CPU28  17   0:58 78.96% sh
 2253 daichi      1  52    0 14504K  2132K CPU2   14   0:56 78.17% sh
 2263 daichi      1  52    0 14504K  2132K CPU10  15   0:56 78.17% sh
 2262 daichi      1  52    0 14504K  2132K CPU2    2   0:57 76.95% sh
 2281 daichi      1  52    0 14504K  2132K CPU7   12   0:56 76.17% sh
 2268 daichi      1  52    0 14504K  2132K CPU13  13   0:55 75.98% sh
 2270 daichi      1  52    0 14504K  2132K CPU8    8   0:57 75.88% sh
 2280 daichi      1  52    0 14504K  2132K CPU25  28   0:58 75.78% sh
 2267 daichi      1  52    0 14504K  2132K CPU12   9   0:56 75.00% sh
 2276 daichi      1  52    0 14504K  2132K CPU11  10   0:56 74.66% sh
 2251 daichi      1  52    0 14504K  2132K CPU27  26   0:56 74.46% sh
 2279 daichi      1  52    0 14504K  2132K CPU1    1   0:56 74.46% sh
 2258 daichi      1  52    0 14504K  2132K CPU25  20   0:56 74.27% sh
 2277 daichi      1  93    0 14504K  2132K CPU29  29   0:57 73.88% sh
 2278 daichi      1  52    0 14504K  2132K CPU24  24   0:57 73.78% sh
 2272 daichi      1  52    0 14504K  2132K CPU16  22   0:55 73.78% sh
 2256 daichi      1  52    0 14504K  2132K CPU5    5   0:56 73.68% sh
 2271 daichi      1  52    0 14504K  2132K CPU26  31   0:56 72.46% sh
 2255 daichi      1  52    0 14504K  2132K CPU4   15   0:58 72.27% sh
 2261 daichi      1  52    0 14504K  2132K CPU7    6   0:55 72.17% sh
 2266 daichi      1  52    0 14504K  2132K CPU0    0   0:55 72.17% sh
 2252 daichi      1  52    0 14504K  2132K CPU0    4   0:56 72.07% sh
 2275 daichi      1  52    0 14504K  2132K CPU17  27   0:56 71.29% sh
 2254 daichi      1  52    0 14504K  2132K RUN    20   0:56 70.90% sh
 2264 daichi      1  52    0 14504K  2132K devfs  19   0:54 70.90% sh
 2269 daichi      1  52    0 14504K  2132K CPU29  18   0:55 70.07% sh
 2250 daichi      1  52    0 14504K  2132K devfs  22   0:55 68.90% sh
 2265 daichi      1  52    0 14504K  2132K CPU6    3   0:55 67.97% sh
 2259 daichi      1  52    0 14504K  2132K RUN    30   0:53 67.97% sh
```

を32個のawk(1)コマンドに割り振って処理をしています。32プロセスでの処理です。

CMD2.SHもCMD3.SHも**図10**のように得られる結果は同じです。

それぞれ実行時間を計測すると、**図11**、**図12**、**表1**の結果が得られます。

単一プロセスで処理した場合と、32プロセスで処理した場合とで2倍以上、処理時間の差が出ています。今回の例はメモリ上の入出力負荷やコンテキストスイッチにかかる負荷が大き

▼リスト2　CMD2.SH（単一プロセスで処理）

```
#!/bin/sh

awk '{  $1=$1+1;   $2=$2+1;   $3=$3+1;   $4=$4+1;
        $5=$5+1;   $6=$6+1;   $7=$7+1;   $8=$8+1;
        $9=$9+1;   $10=$10+1; $11=$11+1; $12=$12+1;
        $13=$13+1; $14=$14+1; $15=$15+1; $16=$16+1;
        $17=$17+1; $18=$18+1; $19=$19+1; $20=$20+1;
        $21=$21+1; $22=$22+1; $23=$23+1; $24=$24+1;
        $25=$25+1; $26=$26+1; $27=$27+1; $28=$28+1;
        $29=$29+1; $30=$30+1; $31=$31+1; $32=$32+1;
        print;
}'
```

▼リスト3　CMD3.SH（32プロセスで処理）

```
#!/bin/sh

awk '{ $1=$1+1; print }'        |
awk '{ $2=$2+1; print }'        |
awk '{ $3=$3+1; print }'        |
awk '{ $4=$4+1; print }'        |
awk '{ $5=$5+1; print }'        |
awk '{ $6=$6+1; print }'        |
awk '{ $7=$7+1; print }'        |
awk '{ $8=$8+1; print }'        |
awk '{ $9=$9+1; print }'        |
awk '{ $10=$10+1; print }'      |
awk '{ $11=$11+1; print }'      |
awk '{ $12=$12+1; print }'      |
awk '{ $13=$13+1; print }'      |
awk '{ $14=$14+1; print }'      |
awk '{ $15=$15+1; print }'      |
awk '{ $16=$16+1; print }'      |
awk '{ $17=$17+1; print }'      |
awk '{ $18=$18+1; print }'      |
awk '{ $19=$19+1; print }'      |
awk '{ $20=$20+1; print }'      |
awk '{ $21=$21+1; print }'      |
awk '{ $22=$22+1; print }'      |
awk '{ $23=$23+1; print }'      |
awk '{ $24=$24+1; print }'      |
awk '{ $25=$25+1; print }'      |
awk '{ $26=$26+1; print }'      |
awk '{ $27=$27+1; print }'      |
awk '{ $28=$28+1; print }'      |
awk '{ $29=$29+1; print }'      |
awk '{ $30=$30+1; print }'      |
awk '{ $31=$31+1; print }'      |
awk '{ $32=$32+1; print }'
```

▼図10　実行結果

```
% head -2 DATA | ./CMD2.SH
1 2 3 4 5 6 7 8 9 10 1 2 3 4 5 6 7 8 9 ↵
10 1 2 3 4 5 6 7 8 9 10 1 2
1 2 3 4 5 6 7 8 9 10 1 2 3 4 5 6 7 8 9 ↵
10 1 2 3 4 5 6 7 8 9 10 1 2
% head -2 DATA | ./CMD3.SH
1 2 3 4 5 6 7 8 9 10 1 2 3 4 5 6 7 8 9 ↵
10 1 2 3 4 5 6 7 8 9 10 1 2
1 2 3 4 5 6 7 8 9 10 1 2 3 4 5 6 7 8 9 ↵
10 1 2 3 4 5 6 7 8 9 10 1 2
%
```

▼図11　CMD2.SHの実行時間

```
% /usr/bin/time -p ./CMD2.SH < DATA > ↵
/dev/null
real 1423.06
user 1421.17
sys 1.88
%
```

▼図12　CMD3.SHの実行時間

```
% /usr/bin/time -p ./CMD3.SH < DATA > ↵
/dev/null
real 660.70
user 20730.18
sys 130.73
%
```

▼表1　実行時間の比較

	1プロセス版	32プロセス版
実時間	1423.06秒	660.70秒
ユーザ時間	1421.17秒	20730.18秒
システム時間	1.88秒	130.73秒

く、あまり処理はスケールしませんが、それでも複数のプロセスに処理を振り分けることで、処理時間の短縮を実現できます。

CPUはどうデータを流しているか？

パイプの実装はOSごとに異なります。FreeBSDのケースでは、/usr/src/sys/kern/sys_pipe.cにパイプ処理の基本的な関数と実装があります。pipe(2)システムコールが呼ばれると、/usr/src/sys/kern/sys_pipe.cに記載されているsys_pipe()→kern_pipe()がコールされ、パイプが準備されます。

パイプ間を流れるデータは基本的にはページ単位(4KB)でデータがプロセスからプロセスへ渡っていきます。最近ではその引渡しで利用するメモリサイズを引き上げて、データの転送速度の高速化が実現されています。この値はサイズを引き上げれば高速化するというものではなく、カーネル内部の動きやプロセッサ、マシンアーキテクチャなどを加味して設定する必要があります。大き過ぎても性能はでません。

パイプを経由するデータは、それぞれのプロセスがwrite(2)システムコールおよびread(2)システムコールを実行することで一方通行で流れていきます。read(2)システムコール／write(2)システムコールが排他制御も担当することになります。

FreeBSDの場合、パイプに使用できるカーネル用メモリの空き容量が50%未満である場合、新規パイプに対しては16KBのメモリを確保し、パイプのやり取りに使用します。このメモリのサイズは使用状況に応じて64KBまでダイナミックに拡張されます。パイプに使用できるカーネル用メモリの空き容量が50%から25%である場合には、新規パイプに割り当てられるメモリは4KBになります。さらにパイプに使用できるカーネル用メモリの空き容量が25%を下回った場合、既存のパイプのメモリも4KBまで縮小されます。

FreeBSDのパイプの実装はよく調整されており、最小限のメモリサイズで最大限の効果をあげるようになっています。こうしたFreeBSDカーネルの挙動はカーネルオプションを指定したり、ソースコードに若干の変更を加え、カーネルを再構築することでパイプの利用するメモリサイズや、諸条件を変更することができます。

しかし、メモリサイズを引き上げてもあまり効果は得られません。特定の用途に対して、数パーセントといった性能の向上を実現することはできますが、劇的な向上は見込めません。デフォルトのアルゴリズムとメモリサイズはよいバランスになるように調整されています。パイプにおいてコマンドを実行する順序や、そもそもの処理内容を工夫する方が効果が見込めます。

このように、パイプで接続されたコマンドが実際にどのように起動され、どのように処理されているのかを知ることで、より効率の良いシェルプログラミングが可能になります。

動作のしくみはわかったわけですから、これにディスクI/Oとメモリ I/Oの速度の違いを加味し、CPUの論理コア(スレッド)の数とI/Oの詰まり具合を考慮すれば、ほぼハードウェアの性能を使い切るようなシェルスクリプトを書くことができます。

シェルスクリプトはもっとも簡単にハードウェア性能を使い切るための効率的なツールです。SD

第6章 より上を目指す シェルスクリプトの覚えておくと便利な技

BSDコンサルティング(株)取締役／最高技術責任者
後藤 大地　GOTO Daichi　daichi@bsdconsulting.co.jp　Twitter ID：@daichigoto、@BSDc_tweet

本章では、ここまでの章で扱わなかったシェルスクリプトを利用する上で便利と思われる事柄についてまとめました。

パイプの目詰まり問題、ダブルバッファ

パイプで接続されたコマンドの入出力は、ほかのコマンドの入出力の影響を受けます。たとえば32個のコマンドをパイプで接続した場合、どこかに入出力が遅いコマンドがあったり、入出力サイズが不規則であるようなコマンドがある場合、ほかのすべてのコマンドがこのコマンドの挙動に引きずられます。

ディスクやメモリの入出力速度が高速であるようなケースでは問題にならないことが多いのですが、入出力が遅いようなケースではこの問題が顕著に現れます。こうした問題に対しては通常、ダブルバッファと呼ばれる手法を使うことで処理の高速化ができます。

図1のように、コマンドとコマンドの間にdd(1)を2つ挟みます。1つめのdd(1)が、最初のコマンドからの出力をバッファリングする役割を負います。2つめのdd(1)が、2つめのコマンドに対する出力バッファリングを担当します。このようにdd(1)を挟み込むことでコマンドからの出力が、ある程度まとまったサイズで連続して次のコマンドに渡ることになり、処理速度が高速になります。

ダブルバッファリングで気を付ける必要があるのは、最初のdd(1)と2つめのdd(1)で指定するブロックサイズが、前後のコマンドによって変わるということです。ここでは1MBを指定していますが、この値は前後のコマンドに合わせて変更する必要があります。実際に動作速度を計測して適切な値を探します。

▼図1　ダブルバッファ

```
% command1 file1 | dd obs=1024x1024 | dd
obs=1024x1024 | command2 > file2
```

排他処理（ln -s）

シェルスクリプトから排他制御を行う場合、シンボリックリンクを使用します。排他制御したいシェルスクリプトや処理同士で、同じファイルに対して同じシンボリックリンクを作成しようと試みます。シンボリックリンク作成時に排他処理が実施され、シンボリックリンクの作成に成功したプロセスとそれ以外のプロセスという区別ができます。シンボリックリンクの作成に成功した1つのプロセス以外のプロセスを排他します（**リスト1**）。

▼リスト1　LOCK.SH

```
#!/bin/sh

if ln -s $0 lock
then
        排他中の処理
        rm -f lock
fi
```

シンボリックリンクではなく、ファイルを作成したり、ディレクトリやハードリングなどを排他処理のロックとして利用することもありますが、これらは推奨されません。ファイルやディレクトリを作成して排他制御用のロックとして使用する場合、タイミングによっては排他制御にならないことがあります。確実に処理を排他的に実施できるのはシンボリックリンクの作成です。

▼リスト2　numcount.c

```
#include <unistd.h>
#include <fcntl.h>
#include <stdlib.h>
#define _WITH_DPRINTF
#include <stdio.h>
#include <stdarg.h>

int
main(void)
{
    char b;
    FILE *f = fdopen(0, "r");
    b = getc(f);
    while (EOF != b) {
        switch (b) {
            case '3': dprintf(3, "%c", b); break;
            case '4': dprintf(4, "%c", b); break;
            case '5': dprintf(5, "%c", b); break;
            case '6': dprintf(6, "%c", b); break;
            case '7': dprintf(7, "%c", b); break;
            case '8': dprintf(8, "%c", b); break;
            case '9': dprintf(9, "%c", b); break;
        }
        b = getc(f);
    }
}
```

▼図2　リスト2の実行結果

```
% printf '3344456678999' | ./numcount | xargs
% printf '3344456678999' | ./numcount 3>&1 | xargs
33
% printf '3344456678999' | ./numcount 4>&1 | xargs
444
% printf '3344456678999' | ./numcount 5>&1 | xargs
5
% printf '3344456678999' | ./numcount 6>&1 | xargs
66
% printf '3344456678999' | ./numcount 7>&1 | xargs
7
% printf '3344456678999' | ./numcount 8>&1 | xargs
8
% printf '3344456678999' | ./numcount 9>&1 | xargs
999
%
```

順番処理（名前付きパイプ）

コマンドの出力を1つ(標準出力)ではなく、複数に分けて扱いたいことがあります。典型的には標準出力と標準エラー出力に対して別々のコマンドへ処理を回したいケースです。いったんすべてファイルに書き出してから別々のコマンドで処理させれば良いのですが、扱うデータが大規模サイズになってくると、ディスクI/Oに多くの時間を費やすことになります。

このようなケースでは名前付きパイプ(FIFOファイル)を使うことで、ディスクに書き出すことなく一気に処理を進めることができます。

▼リスト3　データファイル

```
74588769163982769182618259009701236418364189282992990810827701712897052981583409571399283740982740972409524099698987547643666565745887691639827691826182590097012364183641892829929908108277017128970529815834095713992837409827409724095240996989875476436665657458876916398276918261825900970123641836418928299299081082770171289705298158340957139928374098274097240952409969898754764366656574588769163982769182618259009701236418364189282992990810827701712897052981583409571399283740982740972409524099698987547643666565
```

▼リスト4　NUMCOUNT.SH

```
#!/bin/sh

rm -f FIF03 FIF04 FIF05 FIF06 FIF07 FIF08 ➥
FIF09
mkfifo FIF03 FIF04 FIF05 FIF06 FIF07 FIF08 ➥
FIF09

wc -m FIF03 > CNT3 &
wc -m FIF04 > CNT4 &
wc -m FIF05 > CNT5 &
wc -m FIF06 > CNT6 &
wc -m FIF07 > CNT7 &
wc -m FIF08 > CNT8 &
wc -m FIF09 > CNT9 &

./numcount < NUMBERS 3> FIF03 4> FIF04 5> ➥
FIF05 6> FIF06 ¥
                    7> FIF07 8> FIF08 9> ➥
FIF09
cat CNT*

rm -f FIF03 FIF04 FIF05 FIF06 FIF07 FIF08 ➥
FIF09 CNT*
```

たとえば**リスト2**のような極端なコマンドを作ったとします。これは標準入力から送られてくるデータのうち、3から9までの数字を抜き出してその都度別々のファイル記述子へ出力するものです。たとえば3があればファイル記述子3へ3を出力します。

図2のようにファイル記述子ごとにデータが出力されていることがわかります。

リスト3のようなデータファイルがあったとします。

リスト4のように名前付きパイプ(FIFOファイル)を活用することで、コマンドの出力結果をwc(1)コマンドに流し込んで結果を得ることができます(**図3**)。

手続き型のプログラミング言語ではif構文やswitch構文などの分岐構文を使って処理を実施しますが、シェルスクリプトではこうした手法でデータを処理させることができます。こちらのアプローチのほうが処理が高速です。分岐構文を活用するスタイルはパースの処理の関係で、あまり性能がでません。

ヒアドキュメント

シェルスクリプトの中で標準出力に渡すデータを記述する方法をヒアドキュメントと呼びます。**リスト5**のように「<<」で指定します。「<<」の右側に指定した文字列が終了のマークになります。ヒアドキュメント内部では変数展開とコマンド置換が有効で、チルダ展開は無効です(**図4**)。

終了のマークをダブルクォーテーションまたはシングルクォーテーションでくくると、**リスト6**のように変数展開とコマンド置換が実施されなくなります(**図5**)。

シェルスクリプト内部でタブでインデントしている場合、**リスト7**のようにインデントのタブもそのままデータとして出力されます。終了の文字列は行頭に書く必要があります。シェルスクリプトとしてはインデントが崩れ、あまりきれいとは言えない状態になります(**図6**)。

タブインデントしつつ、先頭のタブは出力さ

▼図3　リスト4の実行結果

```
% ./NUMCOUNT.SH
      21 FIF03
      30 FIF04
      27 FIF05
      36 FIF06
      42 FIF07
      54 FIF08
      66 FIF09
%
```

▼リスト5　HEREDOC.SH

```
#!/bin/sh

cat<<EOF
$(LANG=C date) ~ $LANG
EOF
```

▼図4　リスト5の実行結果

```
% ./HEREDOC.SH
Fri Dec 14 10:03:04 JST 2012 ~ ja_JP.UTF-8
%
```

▼リスト6　HEREDOC2.SH

```
#!/bin/sh

cat<<"EOF"
$(LANG=C date) ~ $LANG
EOF
```

▼図5　リスト6の実行結果

```
% ./HEREDOC2.SH
$(LANG=C date) ~ $LANG
%
```

▼リスト7　HEREDOC3.SH

```
#!/bin/sh

if :
then
        cat<<EOF
        $(LANG=C date) ~ $LANG
        EOF
EOF
fi
```

せないようにすることもできます。「<<-」のようにヒアドキュメントを指定します(**リスト8**、**図7**)。

「<<-」の指定方法でも、シングルクォーテーションやダブルクォーテーションと組み合わせれば変数展開とコマンド置換を抑制させることができます(**リスト9**、**図8**)。

このあたりのテクニックはあまり知られていないところがあります。シェルスクリプトを読みやすく保つ上で有益なテクニックですので、覚えておくと良いでしょう。

/dev/null

/dev/nullは特別なファイルです。/dev/nullはopen(2)してread(2)またはwrite(2)を実行しても、実際にI/Oは発生しません。ベンチマークなどでディスクI/Oを伴わない処理の性能を比較した場合や、データを捨てたい場合などで使われます。実際にI/Oが発生しないため、流れてくるデータを破棄する処理などを高速に実施できます。

▼図6　リスト7の実行結果

```
% ./HEREDOC3.SH
        Fri Dec 14 10:03:33 JST 2012 ~ ja_JP.UTF-8
        EOF
%
```

▼リスト8　HEREDOC4.SH

```
#!/bin/sh

if :
then
        cat<<-EOF
                $(LANG=C date) ~ $LANG
        EOF
fi
```

▼図7　リスト8の実行結果

```
% ./HEREDOC4.SH
Fri Dec 14 10:03:45 JST 2012 ~ ja_JP.UTF-8
%
```

exec 2>って何?

シェルスクリプトのシバンの次の行に「exec 2> log」のような記述がされることがあります。「exec > out」のような記述が使われることもあります。これは、そのシェルスクリプトの標準エラー出力または標準出力を指定したファイルへリダイレクトするという指定です。

リスト10のシェルスクリプトを実行すると**図9**のような結果が得られます。

おもにエラーログを取る目的や、出力をまとめて特定のファイルに記録する場合などに使われます。詳細は第4章をご覧ください。

while問題

繰り返し処理や、データをシェルのread組み込み関数で取得しながら処理する場合などでwhile構文が活用されます。while構文そのもの

▼リスト9　HEREDOC5.SH

```
#!/bin/sh

if :
then
        cat<<-"EOF"
                $(LANG=C date) ~ $LANG
        EOF
fi
```

▼図8　リスト9の実行結果

```
% ./HEREDOC5.SH
$(LANG=C date) ~ $LANG
%
```

▼リスト10　exectest.sh

```
#!/bin/sh

exec 2> log
exec 1> out

printf 'error log¥n' 1>&2
printf 'success¥n'
printf 'exception¥n' > file1
```

▼図9　リスト10の実行結果

```
% ./exectest.sh
% cat log
error log
% cat out
success
% cat file1
exception
%
```

はシェルが提供する構文であるため、シェルと同じレベルで動作します。しかし、これがパイプの中に挟まるとfork(2)したサブシェルで動作するようになります。この場合、while構文の内部で設定された変数などは、もとのシェルに反映されなくなります。

この動きの違いに気がつかずにデータが処理できなくなったと混乱する方がけっこういます。

リスト11のシェルスクリプトを実行すると**図10**のような結果が得られます。

2つめのwhileはサブシェルで実行されるため、もとのシェルに結果が反映されません。もともとリダイレクトでデータを流し込んでいたところをcatに変更するなどするとこの問題が発生します。

おわりに

本特集では、普段説明されることが少ないシェル内部の動きと、実際にカーネルのどういった機能が活用されているのかを紹介しました。シェルはカーネルの機能をダイレクトに使うソフトウェアです。シェルやシェルスクリプトをうまく活用することで、カーネルの性能をフルに活用した処理を行うことができます。

マルチコア／メニーコアの普及が進む昨今、シェルスクリプトはコアの性能をフルに発揮させる道具として強力です。内部の動きを理解し、優れた性能を発揮するシステムを構築する用途に向いています。

今回取り上げた以外のテクニックも含め、シェルやシェルスクリプトに関するテクニックはユ

▼リスト11　WHILE.SH

```
#!/bin/sh

v=a

while :
do
        v=b
        break
done

echo $v

while :
do
        v=c
        break
done    | cat

echo $v
```

▼図10　リスト11の実行結果

```
% ./WHILE.SH
b
b
%
```

ニバーサル・シェル・プログラミング研究所が運営しているUECというサイト[注1]によくまとまっています。シェルスクリプトに興味がある方は、こちらのサイトもチェックしてみてください。

シェルスクリプトという枠を越えて、開発主要や設計というレベルの話になってきますが、処理するデータをテキストデータで保持することが、さまざまな側面から有益であることを最後に補説しております。

処理したいデータをストリームで処理しやすいテキストデータとして設計することで、本特集で取り上げたようなテクニックを適用できます。テキストデータは可視化が容易で、開発中に処理の流れを簡単に把握できます。問題発生時のデバッグも容易です。

テキストデータの設計が話題に取り上げられることは少ないのですが、そこが重要であることを認識として持っていただければと思います。SD

注1）https://uec.usp-lab.com/

第2特集

日常に追われて自分の性能アップを忘れていませんか？

忙しいITエンジニアのための超効率的勉強法

IT業界にいると、たくさんのメールの処理やドキュメント作成などなど、日々の仕事に追われてしまいます。でも、忙しいからといって、技術的に成長しているとは限りません。日常に追われて自分の成長を怠ると、仕事をこなす力がいつまでたっても向上しません。あなたに仕事のライバルはいませんか？　同じ仕事をしているのに、なぜかいつも一枚上手な存在です。何が違うのでしょうか。本特集では、ITブレークスルーの代表でありブログや執筆・セミナー講演活動を行っている「勉強術の達人」森川 滋之さんによる、ストーリー仕立ての特別講座です。2013年が始まり、気持ちも新たに自分の仕事を見直してみませんか！

ITブレークスルー　森川 滋之 *MORIKAWA Shigeyuki*

イラスト
マツダアヤコ

日常に追われて自分の性能アップを忘れていませんか？

ITブレークスルー　森川 滋之
MORIKAWA Shigeyuki

忙しいITエンジニアのための超効率的勉強法

Prologue

プロローグ

はじめに

本誌の読者には"できる技術者"が多いと思います。僕も現役バリバリのUNIX技術者だったころ（もう15年以上も前ですが）は、本誌にお世話になりました。

とはいえ、中には伸び悩みを感じておられる方もいるかもしれませんし、できない部下をどう育てたらいいかわからないという方もいらっしゃるでしょう。あるいは多忙の中で、新しい技術を身につける時間を捻出したいという方もおられると思います。

そういう方のために、僕が実際にやってきて効果があった勉強法をお伝えしたいと思います。一つでも自己啓発や部下育成のヒントになることがあれば、書いた甲斐があったと思います。

僕は1987年にSEとして就職しました。大学時代にパソコンを持っていたので、ある程度の初歩的な知識がありました。おかげで集合研修の間はなんとかついていけたのですが、現場に配属になったとたん大きくつまずきました。

配属されたのは、通信系のミドルウェアを開発している部門でした。最初の打ち合わせで僕は思い切り焦りました。話されている言葉がまったくわからなかったからです。たった一人外国に置き去りにされたという感じでした。

しかし逃げ出すわけにもいきません。忙しい中、どうしたら効率的に勉強できるのか一所懸命考えました。その結果、編み出したのがこの特集でお伝えする勉強法です。

どんな勉強法でも、「読む・書く・話す」という3つの要素が基本になります。この3つが効率的であれば、勉強法としても効率的だと言えます。と言っても、"魔法"を期待する人にはかなりオーソドックスに見えるでしょう。一番避けたい勉強法かもしれません。しかし、時間はあまり要りま

せん。「継続は力なり」と言いますが、継続できるということに重点を置いています（僕自身は、かなりの怠け者なので）。

勉強法をまじめに書くと、重苦しくなるので、ストーリー仕立てにしてみました。とはいえ本質的な部分は外していないつもりです。そこを読み取っていただければ幸いです。

ストーリーには二人の主人公がいます。一人は新人の青二才蔵君。文学部出身で、昔の僕と同じような悩みを抱えています。もう一人は、ベテランの出来内博氏。今でこそ技術部隊の次長ですが、入社当時の1987年には青二君と同じ悩みを抱えていました。青二君が悩んでいると上司の出来内氏が飲みに連れていく。そこで自分の昔話をする。その話をヒントに青二君が少し成長する。そういう構造のストーリーが3話あります（テーマはそれぞれ「読む・書く・話す」になっています）。時間が行き来するので、わかりづらいという方もいるかもしれないと思い、最初にお断りしておきます。なお、二人の主人公は文系出身ですが、特に文系ITエンジニアを意識して書いているわけではありません。

それでは、まいりましょう！

Profile 登場人物

出来内 博
（できない ひろし）

ナニワ情報システムズの次長。アプリケーションインフラグループのリーダー。今でこそ技術部隊のリーダーだが、入社当初は劣等生だった。

青二 才蔵
（あおに さいぞう）

ナニワ情報システムズの新入社員。博のグループのネットワーク構築チームに配属される。文学部出身で先輩たちについていけず悩んでいる。

高飛車 徹
（たかびしゃ とおる）

才蔵の育成担当。名前の通り"高飛車"だと新入社員たちに言われている。

八破原 猛
（やわはら たけし）

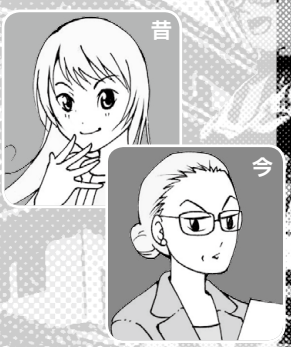
入社当時の博の上司。課長職。今でいうパワハラ上司だが、人望がある。今は、博の事業部の事業部長で常務。

司五斗 優
（しごと すぐる）

入社当時の博の育成担当。まだ４年目だが、部内でも最優秀と噂されている。博に冷たい。

伊達 直人
（だて なおと）

新人時代の博が悩んでいると登場する謎のメンター。濃紺のスーツに虎のマスクを被っている。はたしてその正体は？



中野 あかね
（なかの あかね）

入社当時の博の先輩。根はやさしいのだが、性格がどS。今は人事部長（結婚して高木姓に変わる）。

第1章 明日の朝までにこれを読んでこい！

大阪の西中島南方に本社のあるナニワ情報システムズ(仮名。以下、社名や人名はすべて同様)。地下鉄の駅でいえば新大阪の一つ南の、出張するにはきわめて便利のいい土地である。ところどころにディープな飲み屋が立ち並ぶ横丁があるが、少し歩くとビルばかりだ。ビルといっても高層ビルが立ち並んでいるわけではなく、築うん十年の低層ビルが多い。

駅から徒歩5分の11階建のビルに、ナニワ情報システムズのプロパー400人と協力会社の200人が詰め込まれている。昔は近所の6階建のビルに本社があったが、そこは今コンピュータセンターになっている。

大阪は梅雨入りしたばかりでうっとうしい。もともと湿度が高いので、梅雨時にもなると体にカビが生えるのではないかと思うぐらいだ。そのうっとうしさも、入社3ヵ月目の青二才蔵22歳が感じているものに比べれば大したことはない。そろそろ帰宅しようと思っていたら、育成担当の高飛車徹につかまってしまったのだった。

「青二君。昨日のチームミーティングの話やけど、キミ、ついていけたんかいな？」

「いや。あの、正直あまりよくわからなかったんですけど、まあ、これから勉強します」

才蔵は6月の中旬まで続いた新人研修が終わって、現場に配属されたばかりだった。文学部出身なのでついていけるか心配だった。最後のほうはさすがに難しくなってきたが、なんとかついていけた。それで安心していたのだが、配属直後のチーム会議はショックだった。みんなが何を言っているのかさっぱりわからなかったのだ。ちなみに才蔵が配属されたのは、アプリケーション・インフラ構築グループ。その中のネットワーク構築チームだ。なんとなく技術力が必要そうなチームで、文系の自分が配属された理由がまったくわからない。

「やっぱ、そうやったんやな。まあ、ボクはあんなん大学で勉強してたからなんてことはなかったけど、文学部のキミには難しかったかもしれんな。いや、気づかんで悪かった」

高飛車という名字は伊達じゃない。新人研修の最後は部門研修で、高飛車も講師を務めたのだが、部門に配属された4人の新人が全員「あの人、名前どおりの人やな」ということで一致したのだった。「あの人が育成担当やったらたまらんやろな」と才蔵は思っていたのだが、まさか本当にそうなるとは思っていなかった。

高飛車なだけでなく、話し方がネチネチしている。才蔵は半分スルーしながら聞いていた。

「ん、青二君、聞いてるんか？」

「あ。はい。もちろん」

「ほな、最後に言うたこと復唱してみて」

「ええと。あの。すみません。もう一度お願いします」

「やっぱり聞いてなかったんやな」

高飛車は眼鏡の奥の細い眼でギロっと才蔵をにらんでから、

「まあ、ええわ。あのな、ボク今日・明日と出張やねん。その間かまってあげられへんから、本を一冊読んどいてほしいんや。これ読んだら、昨日の会議の話の半分ぐらいは理解で

きるはずや。明日の夕方戻ってきて、口頭試問をするから、ちゃんと読んどくんやで」

高飛車はこう言って、『TCP/IP実践プログラミング入門』(仮名)という500ページもある本を、もったいぶったポーズで才蔵に手渡し、「じゃあ新幹線に間に合わへんから」と言い残して立ち去って行った。

それにしてもコンピュータ関係の本て、なんでどれもこれもこんなに分厚いんやろ、と才蔵は思った。これで入門やったら、応用って何ページになるんや。コンピュータ関係の本の9割以上(筆者の実感に基づく)は「入門」だということを才蔵はまだ気づいていなかった。本当の「応用」は現場にしかない。

才蔵は、仕方なく1ページ目から読み始めた。

あっという間に夕方。つっかえつっかえ読み進めていたが、気がついたらまだ100ページしか読んでいない。あと400ページどうしたらええんや……。才蔵は目の前が真っ暗になった。「しゃあない。持って帰って徹夜で読もう」

才蔵が帰り支度を始めると、グループリーダーで次長職である出来内博が近寄ってきた。

「青二君。今朝から元気がないみたいやけど、大丈夫か？」

「いや。大丈夫です。高飛車さんから明日の夕方までにこの本読んどけと言われたんですけど、ちょっと難しくてまだ2割ぐらいしか読めてないんで、今日は帰って徹夜で読みます」

「徹夜はやめとけ。それより今から飲みに行こ」

「ええっ？　いや、これ読まんとあかんので……」

「そんなん、明日1日あれば十分お釣りがくるわ。読み方教えたげるから行こうや。俺もすぐに帰り支度するから」

才蔵は断りきれない性格だ。それもあるが、「読み方」に興味があったのでついていくことにしたのだった。

15分後。会社から見て地下鉄御堂筋線の反対側にある「おかめ」という店に二人はいた。ビール大瓶400円。焼き鳥1本90円。ポテトサラダ280円。値段だけでどんな店か想像はつくだろう。ビールで乾杯した後、博は切り出した。

「実はなあ、今朝の高飛車と君の会話を聞いてたんや。それで、たぶん読み切れへんやろなあと思いながら、一日様子を見てたんや。やっぱりあかんかったな」

「え。そうやったんですか。それは人が悪い」

「いや。実は俺も昔おんなじ経験してな。あ。ちなみに高飛車もや。あれってうちのグループの伝統やねん」

「そうなんですか？」

才蔵はあらためてとんでもないところに配属されたと思うのだった。

「で、どうやって読むんですか？」

「結論を急いだらあかん。まあ、俺らは報告を受けるときに結論を先に言え、なんていうけどな、学ぶほうは結論を急いだらあかんのや」

「はあ」

「まあ、しばらく俺の昔話に付き合え」

博は遠い眼をしてこう言った。

◆　◆　◆

1987年にさかのぼる。出来内博はアプリケーション・インフラ構築グループの5代ぐらい前身

の通信技術部に配属された。当時は文系出身の男性SEなどはほとんどいなかったが、SE不足説などというのがまことしやかに喧伝されていた時代だったので、そろそろ文系も採用してみようかというような雰囲気があった。文学部だけどパソコンに興味のあった博は、なんとなくその時流に乗ってナニワ情報システムズに就職できたのだった。

配属されたその日に、もしかしたら失敗したかなと思った。打ち合わせで交わされている言葉がまったく理解できなかったのだ。

そのうえ誰も懇切丁寧に教えてくれない。育成担当の司五斗優は、まだ4年目だったが部内でも一番できると言われた男だった。それなのに、質問しにいっても「そんなん、自分で調べろや」とつれない。「そんな恥ずかしい質問、俺が新人の頃はとてもできへんかったわ」などと言う。

3年目の中野あかねは、仕事もできて、根はやさしいようなのだが、性格がどSで、司五斗に説教された直後にからかうようなことを言う。「"できない"って名前どおりやなあ。でも、がんばりや。そのうちなんとかなる」などときついのかやさしいのかさっぱりわからないことを言う。言った後にギャハハと笑うのでバカにされている気持ちになる。

課長の八破原猛は、今でいうパワハラ上司で、博が困っていても助けてくれるどころか叱りつけたりする(なお八破原は、現在ナニワ情報システムズの常務で、いまだにその癖が抜けず、もちろんパワハラ常務とあだ名されている。しかし、変な人望があり、パワハラで会社を追われる気配はない)。

そして、とうとう辞めたほうがいいんじゃないかと思う"事件"が起きた。配属されて1週間。相変わらずさっぱり話が理解できなくて激しく落ち込んでいた博に、司五斗がこう切り出したのだ。

「もう1週間たったで。そろそろ戦力になってくれへんと困るでぇ」

「はあ」

「何がわからんねん？ 俺には逆にその辺が理解できないんや」

「基礎的なことがわからないというか……。言葉がさっぱり理解できないんです」

「集合研修のとき何してたんや？」

「いや、研修のときには……」

(そんな難しい言葉はどこにも)と博が言いかけたとき、司五斗はこう言った。

「わかった。ほな、これを明日の朝までに読んで来い。そしたら、かなりわかるようになるはずや」

渡されたのは『VTAM[注1]の基礎知識』（仮名）という300ページほどの本だった。司五斗は本を渡すと「ほな今日は帰るわ、昨日徹夜やったし」と言い残して、立ち去って行った。

博はパラパラとのぞいてみた。確かに自分には訳のわからなかった言葉が書き連ねてあった。これをマスターすれば、司五斗の言うとおりわかるようになるかもしれない。ただし、300ページもある。もう夕方だ。明日の朝まで徹夜で読んでも間に合うだろうか？

そんな博の心をまるで理解しないのか、隣で聞いていた中野あかねが「いやあ、たいへんやなあ。きばりやあ。ギャハハ」と笑う。博は、この業界ってこんなに一生懸命勉強しないと一人前になれないのかと暗い気持ちになった。いや、司五斗も中野もそんなに勉強しているようには見えない。忙しくて勉強している暇もなさそうだ。やっぱり才能なんだ。あるいは大学での基礎なんだ。

とりあえず、明日の朝までチャレンジしてみよう。それでだめだったら辞表を書こう。博は追いつめられていた。

「お先に失礼します」と元気なく言いながら、博は退社した。その様子を八破原がするどい目つきでじっと見ていた。

博は会社から徒歩15分のワンルームマンションに住んでいた。今日は自炊する気にもなれず、帰り道にある居酒屋「おかめ」で夕食を取ることにした。混み合っていたが、4人がけのテーブルが1台空いていた。居酒屋だがもともとは大衆食堂なので定食もやっている。ビフカツ定食というのが安くてボリュームもあるので、いつもそれを頼むことにしている。

待っている間に、『VTAMの基礎知識』を読み始めた。これを明日の朝までに読めなんて、不可能だと思えた。司五斗のことだ、きっと口頭試問もするだろう。それに不合格だったら、やっぱり向いてないんだろう。

などと思いにふけっていたら、真ん前で「ここ、相席ええかな？」というこもったような声がした。「あ。はい」と顔を上げると、そこには虎がいた。正しくいうと、虎のマスクをかぶったスーツ姿の男がいたのだった。博は一瞬ギョッとしたが、すぐにここは大阪だということを思い出した。たぶん阪神ファンなんだろう。さすがに大阪でも虎のマスクは違和感があったが、東京の10倍ぐらいは怪しいものへの許容度が高い大阪である。博もすぐに順応した。

なんか見たことがある人だと思った。すぐに八破原に似ていると気がついた。体格もスーツも良く似ている。ただ、ネクタイが全然違うので別人だろうと思った。スーツも濃紺のダークスーツで特徴はない。それに八破原がここにいるはずがない。

「兄ちゃん。なんか悩んでるんとちゃうか？」

虎男はなれなれしく話しかけてくる。まあ、大阪ではよくあることだ。博は食事の間だけ付き合

注1）VTAM：Virtual Telecommunications Access Method。大型汎用機で使用されるIBMの通信ソフトウェア。IBMが提唱したSNA（Systems Network Architecture）を実現するために作られた。現在ではCommunications Serverと呼ばれ、TCP/IPもサポートしている。SNAは今のTCP/IPに該当する、当時の通信系エンジニアの基本中の基本であった。読み方は「ヴィタム」。

うことにした。

「悩んでますけど、ちょっと専門的な悩みなんで」

「そうか。わしは伊達直人というもんや。兄ちゃんは？」

「僕は、出来内博と言います」

「けっさくな名前やな。いかにも仕事ができないちゅう感じや」

伊達直人はそう言って、遠慮なく笑う。博は自分の名字を気にしていたが、そこまであけすけに言われるとかえって好感が持てた。

「当てたろか。悩みはいま読んでた、その分厚い本やろ？」

「げっ。当りです。なんでわかったんですか？」

「がははは。いまその本読みながらため息ついてたやないか」

「ああ。なるほど」

「おおかたあれやろ？ 明日の朝までにその本を読んで来いとでも言われたんやろ？」

「わかりますか？」

「わからいでか。わしは悩める青少年を救うのが仕事なんじゃ」

ああ。それで伊達直人か。絶対偽名だと思っていたが、人助け気取りなんだ。

「ちょっと貸してみい」と、伊達直人は言い終わらないうちに『VTAMの基礎知識』を手に取った。パラパラめくりながら「ふん。コンピュータ関係の仕事かい」とつぶやく。

「わしなら15分やな。それで読めるわ」

「え？ それはありえないでしょう？　第一言葉がわかるんですか？」

「わからんでも読めるわい。なあ、兄ちゃん。本には三種類あるって知ってるか？」

「さあ」

「兄ちゃんに、明日までに読んで来いと命令したやつはそこのところがよくわかってるようやな。そうでないとそんな命令はできへん」

ビフカツ定食が二つ運ばれてきた。一つは伊達直人のものらしい。それにビールの大瓶も。伊達直人はグラスを手に取り、目で注げと指示した。素直に注ぐと、お前も飲めという。明日までに本を読まないといけないというと、そんなん明日早起きしたらできると言われる。博は断れない性格なのだった。しかも一気に飲み干せという。

「よし、ええ飲みっぷりや。ほな、説明しよか。三種類の本の話や。ちゃんとメモするんやで」

博はカバンから手帳とボールペンを取りだした。

「一つは、頭から終わりまで通しで内容を追っかける本。小説なんかがそうや。二つ目は、必要なときだけ参照する本。辞書なんかがそうやな。そしてもう一つ。体系をつかむために一度ざっと読んで、その後は何度も必要な個所を読む本。専門書やマニュアルなんかがそうや」(**図1**)

最初の2つは知っていたが、最後のは少し意外だった。そういう読み方もあるのか。

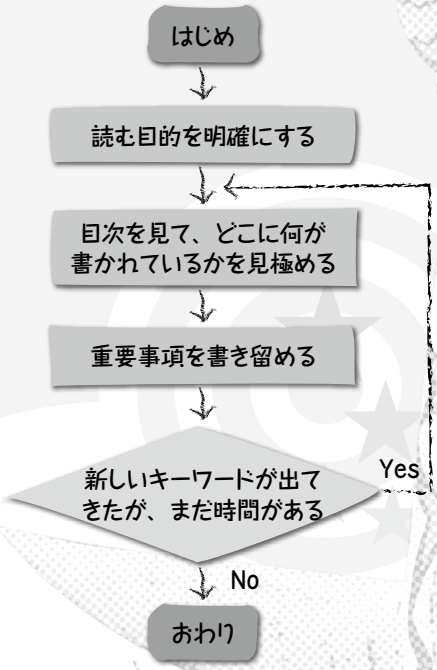
「その本は典型的な最後の種類の本や。だから、言葉なんかわからんでもええ。先に体系を捉まえる[注2]んや。それができれば、言葉同士が勝手につながっていくようになる。それに重要な言葉は何度も出てくる。そういうことも見えてく

▼図1　三種類の本の話のメモ

本には三種類ある

❶ 通しで読む ～小説・マンガなど
❷ 都度調べる ～辞書など
❸ ざっと読んで、部分的に何度も読む
～専門書・マニュアルなど

▼図2　エンジニアのための超速読法



るようになる」

そこまで言うと伊達直人は、博の手帳とボールペンを奪い取り、そこに「エンジニアのための超速読法」とタイトルを書き、フローチャートをかいた（**図2**）。

「肝心なんは、ここや。『目次を見て、どこに何が書かれているかを調べる』。兄ちゃん、ちゃんと目次は読んだか？」

「はあ。一通りは」

「それで何がわかったんや」

「いや、ただ目を通しただけで」

「そやろな。それがあかんのや。薄っぺらい本やと別かもしれんが、それだけ分厚い本で目次がない本はまずないやろ。でも、頭からお尻まで読まなあかんのやったら、そもそも目次なんて要るか？」

そんなこと考えたこともなかった。

「伊達や酔狂でついてるわけやないんやで、目次は。一番肝心な本の構造が書いてあるんや。だから目次を読みこめば、その本の体系がわかる。それだけやない。場合によっては著者の思想までわかる」

「え？　何でですか？」

「著者が何を重要と考えているかに思想が現れる。重要性は、書く順番である程度わかるやないか。最初と最後に重要なことが書いてあるのが普通や。だから、『はじめに』と『おわりに』なんていうのも実は重要やったりする。それともちろん第1章と最終章もな。あと、繰り返し出てくるのも重要な話や。そういう傾向も目次である程度わかる」

なるほど。虎のマスクなんか被っているのでバカにしていたが、だんだんタダ者じゃない気がしてきた。

「目次を読み込んだら、重要だと思う個所を拾い出して、あとはそこを重点的に読み込むんや。読んだらポイントだけメモする。わからない言葉は抜き出しとけ。あとで調べたらいい。索引も重要やな。たくさんのページに出てくる言葉は重要やちゅうこっちゃ。で、最終的にはA4の用紙1枚ぐらいで本の要旨をまとめるんや」（**77ページのコラム図3**）

博は速読法の本を読んだことがある。目玉を上下左右に速く動かす訓練をしろという本もあれば、見開きを一瞬で脳に焼き付けて、次々とページをめくれという本もあった。共通するのは「考えずに見ろ」ということだった。実際試してみたが、頭には何も残らなかった。しかし、この方法なら1時間もあれば多くの情報が残る気がした。伊達直

注2）誤字ではない。筆者ははじめ大阪で就職した。そのとき違和感があったのがこの言葉。大阪にはなぜか「捉える」を「捉まえる」という人が多い。中には「捕らまえる」と書く人もいる。ここまでくると標準語と大阪弁のバイリンガルではあるが大阪弁ネイティブではない筆者には、「捉える」の意味で使っているのかちょっと自信がなくなる。

人が言うように15分でも可能かもしれない。

「どや。気が楽になったやろ」

「はい。やれそうな気がしてきました」

「初めてやから1時間ぐらいかかるやろけど、明日早起きすれば問題ないはずや」

「はあ。そうかもしれません」

「ほな、もうちょっとビールつきあえ。なあに、早起きせなあかんから、ほんの数本や」

伊達直人は適当につまみを頼み始めた。繰り返すが、博は断れない性格だった。1時間ほど説教のようなことを聞かされたが、博は気持ちよく聞いていた。伊達直人はおごってくれたうえ、帰り際にポケベルの番号を書いた紙をくれた。

「また、悩みがあったら相談せい。ポケベルが鳴ったら、一時間後にここにおるわ。おらんかったら、その日は用事があったちゅうことで勘弁してくれ。ほな、またな」

◆ ◆ ◆

「で、どうなったんですか？」

舞台は再び現在に戻る。青二才蔵が興味津々という様子で出来内博に聞く。

「うん。翌日は朝4時に起きて、伊達直人の言うとおりにやってみたんや。そしたら、5時過ぎには読み終わってた。出社したら予想通り司五斗さんの口頭試問があった。これも伊達直人の言うとおりやった。司五斗さんは細かいことでなく体系的な話、たとえばVTAMって何のために必要なんや、みたいな質問ばかりしてきたんや。こちらはそういうとこばっかり読んでたし、メモもしてたんで簡単に答えられた。『よしっ、合格や。その調子で頑張れ！』って言われたときには涙が出そうになった。そしたら、中野さんがコーヒーを淹れてきてくれて、『ご苦労さん。やったやん！』と言ってくれたんで、本当に泣いてしもた。恥ずかしかったなあ」

そう言いながら、博の眼は少しうるんでいるようだった。才蔵もうるうるしかけたが、一番重要なことを聞くことにした。

「で、その読み方は役に立ったんですか？」

「それや。まだ完璧には程遠かったけど、打ち合わせでの話がなんとなくわかるようになってきたんや。英語ってある日突然聞こえるようになるというけど、それに近い感覚かな。相変わらず英語は苦手なんでようわからんけど。とにかくまったくの外国語だったのが、半分ぐらいはわかるようになった。そうなったら面白くなってきて、いろんな本を伊達直人式超速読法で読むようになった。1ヵ月ぐらいやったかなあ。突然霧が晴れたようにみんなの言うことがわかるようになったんや」

「僕にもできるでしょうか？」

「大丈夫や。これってうちのグループの伝統やって言うたやろ？　高飛車もこれをやってから、多少はわかるようになったんや」

何が「ボクはあんなん大学で勉強してたからなんてことはなかった」や。うそつきやん。そう思うと、才蔵ははじめて高飛車が身近に感じられるようになったのだった。

翌日。才蔵は、高飛車の口頭試問を軽くクリアすることができた。高飛車も細かいことではなく、体系的なことや全体的なことしか聞いてこなかったからだ。

「やればできるやないか。この調子やったら、来月からは青二君に、議事録を任せられそうやな。ボクもいい加減あきあきしてたんや」

えっ！ 議事録？ あれをやるの？？？

アプリケーション・インフラ構築グループでは、どのチームもホワイトボードに書いたメモを議事録として回覧することになっている。そのために板書をPCにカラーで取り込めるようになっているホワイトボードを設置している。終了してから書くのは時間のむだというグループリーダー出来内次長の方針なのだ。つまり一発勝負で議事録を書かなければいけない。

これは荷が重い。わかっていないと書けないからだ。

一難去ってまた一難。才蔵の明日はどっちだ？

Column エンジニアのための超速読法

出来内博にふりかかった困難は、ほぼほぼ僕の身に起こったことです。タイガーマスクこそ出てきませんでしたが。

僕の場合は育成担当がとても忙しい人で、週に3日ぐらいは出張していました。なので最初のうちは適当にこれ読んどけとか、このテープ教材を聞いとけとか、そんな指示しかしてくれませんでした。そんな感じだったので、僕もよほどのんびりしていたのでしょう、1週間ぐらいダラダラと勉強していました。

そうしたら、チームミーティングの内容がさっぱり理解できない。実は同期にもう一人、成績優秀な女性がいたのですが彼女はなんとなくわかっている感じ。まあ、こちらは育成担当がつきっきりで教えてくれていたということもあったのですが。

それで僕の育成担当は危機感を抱いたのでしょう。朝呼ばれて、「これから出張に行ってくる。今日は日帰りやから夕方までにこの本を読んどけ。試験するから、そのつもりで読めよ」と言われたのでした。

午前中は頭から順番に読んでいたのですが、昼までに数十ページしか読み終わらない。開き直って、「伊達直人式超速読法」に似たようなことをやりました。とても細かいところまでは理解できないから、体系と重要な概念だけを捉えようとしたのです。

夕方、育成担当の口頭試問に見事合格。「1日で本を読む方法がわかったやろ。こんな感じでまずはたくさん読め」と言われて、その後は本当に何百冊もこの方法で読んだら、いつの間にか「技術のあるやつ」になっていました。

その後、日本の経営コンサルタントの草分けともいえる方のセミナーに参加する機会がありました。テーマは速読法。この速読法が、僕が偶然見つけたものと考え方が同じだったので、我ながらびっくりしました。ただ、この先生の場合は1冊を15分で読めというもの。僕はだいたい半日ぐらい掛けていたのでそんなことできるのかと思いましたが、実際に15分でできてしまったのです。

ただし、このときはビジネス書でした。ビジネス書は、言いたいことが繰り返し出てくるので、割とまとめやすいのです。エンジニア向けの本で15分でできるのだろうか？

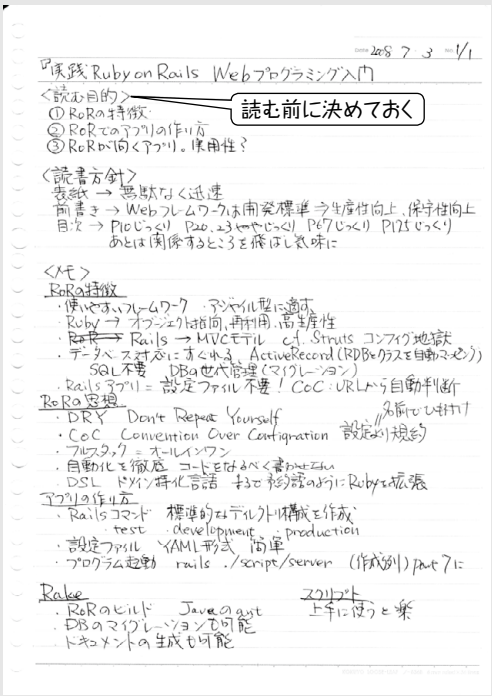
もう4年前になりますが、実際に時間を計ってみたことがあります。タイマーを15分にセットして、その中で読み切れるか試してみました。『実践Ruby on Rails Webプログラミング入門』(ソーテック社)。B5変型判(SD誌を縦に2cmぐらい縮めたもの)で260ページある本です。ちなみにRoRに関する本を読むのはこれが初めてでした。

ギリギリでしたが、ほぼ15分で読み終わりました。そのときのメモを掲載します(**図3**)。

15分にこだわることはありません。1時間かかってもいい(逆にそれ以上掛けないようにしてください。忙しいのですから)。1冊につき1枚、こういうメモを残す。これを週に1回か2回やる。1年も積み重ねれば、周囲のエンジニアとはかなりの差がついているはずです。

なお、ITだけでなく顧客の業務知識や業界知識の習得にも役立つのは言うまでもありません。

▼図3　筆者の作成した読書メモの例



第2章 お前はノートもロクに取れへんやろ？

大阪の夏は暑い。沖縄より気温の高い日も多い。湿度が高いだけでなく、緑が少なくて舗装路が多い。ヒートアイランドというやつだ。夜になっても気温が下がらない。

ということで夏バテ気味の青二才蔵だが、目覚めが悪いのはそのせいだけではなかった。今日のミーティングから書記役をやれと、育成担当の高飛車徹に言われたからだった。

「業務命令やからね」とプレッシャーをかける高飛車の顔が夢に出てきた——気がする。

グループリーダーである出来内博次長の方針で、才蔵の所属するグループでは、打ち合わせの議事録はホワイトボードの板書のコピーということになっている。あとでまとめなおすということは許されていない。時間ももったいないし、書記もイマイチ真剣にならないというのが博の主張だ。つまり一発勝負。内容がわかっていないと書けない。

出来内次長から伝授された速読法のおかげで、打ち合わせの内容がわかるようになってきた才蔵だったが、メモを取ったり、まとめたりするのはちょっと苦手だ。

とはいえ、才蔵には秘密兵器があった。マインドマップである。今まで高飛車が書記役を務めてきたのだが、彼の板書は少し書き過ぎじゃないかというのが才蔵の感想であった。議事録を読むほうも忙しい。簡潔にまとまっているほうが喜ばれるだろう。うまくいけば、自分の株も上がるかもしれない。

不安と同じぐらいの期待をもって、才蔵は打ち合わせに臨んだ。

その夜。場所は、西中島南方の居酒屋「おかめ」。暑いから生ビールでも飲みに行こうと博が誘ったのだった。

「何がそんなに悪かったんでしょう？」

今朝の打ち合わせの件だ。半分不安で半分自信のあったマインドアップでの板書を、高飛車に思い切りけなされた。才蔵はそのことについて質問をしているのだ。

「まあ、アイデアは良かったんやけど、使いどころを間違えたんやなあ」

「あとで見てわかりやすいと思ったんですけど」

「そういう気持ちやろなとは思ったんやけど、議事録にマインドマップはあかん」

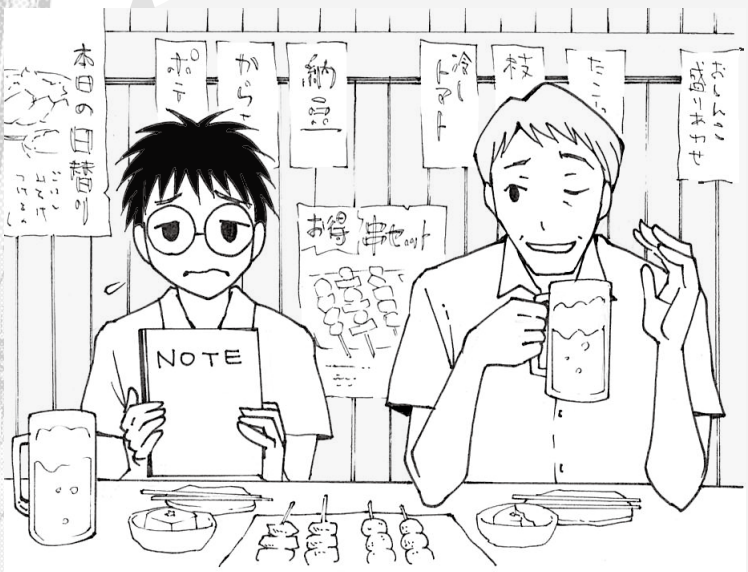
「何でですか？」

「議事録は、話の順番が大事なんや。マインドマップで時系列が表現できないかというと工夫の余地はあるとは思うんやが、基本的には向いてない。あとで見る人は結論はわかるかもしれんが、話の経緯がわからへん。しかし、打ち合わせでは、その経緯が大事だったりするんや」

「なるほどです。でも、高飛車さんのあの言い方はないよなあと思ったんですけど」

高飛車は、「おまえはノートもロクに取れへんやろ？　そやからできるようにならへんのや」と言ったのだった。

「確かに言い方は悪いけど、俺も実はそう思っ



た。青二君はノートの取り方を知らへんのやろなと」

「えぇ～？　僕は大学時代、ノートがきれいと言われてたくさんの友達からコピーさせてと言われてたんですよ」

「うん。たぶん試験に合格するという意味ではいいノートやったんやろ。でも、大学の勉強は身についてるかな？」

そう言われると、返答しようがない。文学部出身なのにIT業界に入った理由も、実は大学での勉強が身についていないと思ったからだった。現状打破が半分、逃げが半分。

「大学とは違って、仕事のための勉強は身に付かないとあかん。そのためのノートの取り方ゆうもんがある。いや、たぶん大学でもな、残って研究を続けているような人は、同じようなノートの取り方をしているはずや」

「教えてください」

「学ぼうとする者は、結論を急いだらあかん」

「はあ」

才蔵は軽いデジャビュを感じた。

「まあ、しばらく俺の昔話に付き合え」

博は遠い眼をしてこう言った。

◆　◆　◆

1987年夏。大阪の夏は暑かった。

出来内博は4年間の大学生活を京都で過ごした。京都の夏は日本一蒸し暑いと聞いていた。それでも当時はエアコンなど贅沢品だったし、文学部の校舎にもそんなものはなかった。扇風機でなんとかしのいできたのだった。だから、暑さには自信があった。でも、大阪に引っ越して初めての夏を迎えて、その自信は消し飛んでしまった。何しろ夜になっても気温が下がらないのだ。

相変わらずエアコンなしで暮らしていたので、博はこのところ寝不足気味だった。それがいけなかったのかもしれない。中野あかねの逆鱗に触れてしまった。

博のチームは、アセンブラでプログラム開発をしていた。新しいプロジェクトで、標準化のためにマクロライブラリを作成することになった。「そんなんは新人の仕事や」という八破原課長の意味不明な方針で、博がライブラリの作成と管理を担当することになったのだった。

アセンブラのマクロライブラリを作れと言われても、博にはチンプンカンプンだった。例の伊達直人式速読法で勉強しようと思ったのだが、すごく薄っぺらい「マクロアセンブラ仕様書」というものしかない。見てもさっぱりわからない。別のプロジェクトのライブラリのソースコードも参考にしたが、自分のプロジェクトに適用できるのかもよくわからない。

育成担当の司五斗優に聞きに言ったら、今回は自分で調べろとは言わなかった。「忙しいので中野に教えてもらえ」と言う。

そこで中野に頭を下げると、「ええけど、高いでぇ」と言う。「いくらでしょうか？」と聞き返すと、「アホか？　ギャハハ」と笑われてしまった。同期の男どもは、美人の中野にあこがれて「おまえ、中野さんの隣の席でええなあ」などというやつが後を絶えない。それに対して博は「いつでも替ってやる」と半ば本気で答えていた。

会議室で中野のレクチャーが始まった。途中わからなくなってきたので質問すると、「ええから、ノート取れや」と乱暴に返される。それでホワイトボードの板書を完璧に写したのだった。45分ぐらいたってから、中野が「ちゃんとノート取ってるか？　見せてみ？」とノートをのぞき込んできた。そして、叫んだのだった。

「なんや、このノートは！　キミ、ちゃんと聞いてたんかっ！」

「いや、一所懸命板書を写してたんですけど」

「アホかぁ！　こんなんで後で見直してわかるかぁ！　もう、この忙しいのにぃ。私の45分を返せぇ！」

そう言うと、中野は会議室を出て行ってしまった。

博は、何を怒られているのかわからず、茫然としてしまった。3分ほど考えたがよくわからない。そこでとりあえず謝りに行ったのだが、「何を怒られてるか、わからんもんを許せるか」と取り合ってくれない。司五斗に取りなしてもらおうとしたが、「中野があないなったら、誰もどうしようも

ない」と相手にしてくれない。

時刻は16時50分。こんな気分では仕事にならない。今日は定時に帰ろうと博は思った。そして、伊達直人のポケベルに連絡を入れたのだった。そのとき偶然だろう、八破原のポケベルが鳴ったようだった。

50分後、博は居酒屋「おかめ」で、やけに苦いビールを飲んでいた。今日も暑かったので美味いはずなのだが、そうは感じられなかった。伊達直人は来てくれるだろうか？　まあ、来なくてもいい。飲んだくれてやる。

立てつけの悪い横開きのドアを乱暴に開ける音がする。そこには、濃紺のダークスーツに虎のマスクのあの男がいた。伊達直人はまっすぐに博のテーブルに向かってくる。

「待ったか？」

「いえ、約束の1時間には、5分も早いです」

「うむ。こう見えても時間厳守でな。で、何があったんや？」

博は、中野あかねとの一件について語った。

「で、その姉ちゃんが何を怒っているのか、さっぱりわからんちゅうんやな」

「はい」

「なんぼわしでも女のヒステリーについてはお手上げやな」

今ならセクハラと言われても仕方ない発言だが、当時は普通の会話だった。

「そこを何とか」

「がははは。冗談や。彼女が怒った理由はようわかる。けっして理不尽ではない。兄ちゃんが悪い」

「どういうことでしょう？」

「うむ。兄ちゃんは高校時代、成績優秀やったろう？」

「まあ、そこそこには」

「大学でも、試験はなんとかこなしてきた。だから卒業できて今こうして就職している。でも、学問という意味ではあかんかったんやないか？」

その言葉は博の胸に突き刺さった。元々は学者になりたくて文学部に入ったのだった。親からは、文学部なんてつぶしのきかない学部はやめとけと言われたが、研究職へのあこがれは強かった。しかし、大学に入ってすぐに勘違いしていたことに気付いた。学者を目指す連中は全然違っていた。自分には才能がないと思ってあきらめた。それで結局畑違いのところにきて苦労している。

「学問の才能がなかったんやと思います」

「そうやないな。兄ちゃんはたぶん要領のええほうなんや。でも、学者のほとんどは逆やな。要領が悪い。才能の違いではないんや。愚直にやれるかどうかが境目なんや」

「要領がいいのは悪いことなんでしょうか？」

「そうとは言えん。仕事でも要領よくやれたほうがええことはたくさんある。ただなあ、何かを身につけようと思ったら、要領のよさより愚直さやな」

「今日の僕は愚直さが足りなかったと」

「結論を急ぐな。愚直ってどういうことやろか？」

「疑問を持たずに言われたとおりに一生懸命やる、ということでしょうか」

「まあ、そういうことやが、本来はあまりええ意味ではないな。バカ正直で臨機応変な行動がとれないということや。でも、学び始めはそれが大切なんや」

「そうでしょうか。言われたことを自分なりに解釈してこそ、身に付くのでは」

「そこやな。たぶん"解釈"の意味がわかってないんやろと思う。解釈とは実践を通じてすることや。知識を吸収する段階では、わからんでもとにかく聞き、そして写し取るということが大事なんや。『まなぶ』の語源は『まねる』というやろ？」

博には、わかるようでわからなかった。

「わしは、江戸時代の教育の仕方は素晴らしかったと思ってる。あれがあったからこそ、日本は植民地にならんですんだんやないかな」

幕末のころの日本の識字率は、世界的な水準でみたら驚異的だった。支配階級である武士はもちろん、庶民の多くも文字が読み書きできた。寺子屋が普及していたからだ。寺子屋の教育法も武士の教育法も初期段階では一緒で、漢文の素読が入口だった。漢文には当時の教養が詰まっていて、それを学ぶのは重要なことであった。その教育法である素読とは実にユニークなものだった。まだ幼い子どもが意味もわからず、ひたすら先生の言うとおりに真似をする。何度も真似しているうちに、子どもは記憶する。その記憶が将来の教養になる。教養は行動を通じて得るものだと当時の人は理解していた。体験をすることで、意味がわからなかった言葉が自分の教養になる。その成果が明治維新だったと伊達直人は説く。

「この素読という勉強法が愚直の見本やな。今の人はこんな教育法では個性が伸びへんと否定するかもしれん。でも、幕末から明治の頃の人のほうが今の人よりよっぽど個性的やった思うんは、わしだけやろか？」

そんな気はする。今の政治家で歴史に残るような個性的な人物がどれだけいるだろうか？　幕末から明治にかけては、あんなに人物がいたのに。まあ、素読と関係があるかはわからないが。

「ということは、僕も愚直に中野さんの言うことをメモすれば良かったんでしょうか？」

「そうや。そういうことや。板書だけ写してもあかんのや。その姉ちゃんの板書は、かなりラ

フやったんやないか？」
「そう言われれば、そうやったかもしれません」
「だから、兄ちゃんはほとんどメモを取ってなかった。それに気づいて叱ったんやろなあ。その姉ちゃんはノートの取り方をよう知ってるみたいや」
そう言えば、チームのメンバーは打ち合わせ中にひっきりなしにメモを取っていることに気がつく。
「たぶん、学者になるような人のノートは、教授の講義をそのまま聞きとって書いてるんやろな。たとえば、ソシュール[注3]という学者がおった。有名な言語学者で20世紀の思想に大きな影響を与えた人や。この人は本を一冊も書かなかった」
「じゃあ、どうやって影響を与えたんですか？お弟子さんが広めたとか？」
「まあ、弟子が広めたのは間違いないんやが、直接の弟子でそんなに影響力のある人はおらんかった。学生の講義録が残っていて、それを弟子たちがまとめて出版したんや。昔の学生がいかに熱心にノートを取っていたかわかるやろ。録音機もビデオもなかった時代やから、ほんまに真剣に書きとったんやろな」
「素読の時代は聞いてるだけでしたが」
「ああ。それは紙が貴重品やったからや。だから一生懸命自分自身を録音機にしたっちゅうわけや。その人たちから見たら、紙に写すなんていうのは真剣味が足りんと言うかもしれんな。ただ、その代わりおぼえるまで何度も何度も聞くことになるけどな」
「書き写しでなく、録音機[注4]じゃダメなんでしょうか？」

「悪いとは言わんよ。ただ、その場での真剣味は紙に写すより劣るやろし、後でテープから書き起こす時間がむだやろ。ノートするんが、わしは一番効率的やと思うなあ。録音機は保険として考えるならええんやないかな」
「ノートの取り方にコツはあるんですか？」
「まあ、愚直にできるだけ一字一句に近い形で書くというのが基本や。考えへんことが大事やな。ひたすら写す。そやけど、多少の工夫はある。わしの方法やけどな」
そう言って、伊達直人はノートとボールペンを要求した。博がカバンから取り出して渡すと、そこに「四色ボールペン・メモ法」というタイトルで何やら書き始めた(**図4**)。
「基本的には黒を使う。それから話にはテーマがあるやろ？　文章でいえば小見出しみたいなもんや。それを青で書く。上手な話し手やったら、今からナニナニの話をするって断るやろ？
それが青や。それから重要やと思ったことは、赤で囲む。下線でもええわ。それから、相手の話やのうて、自分の思いつきをメモしときたい

注3) フェルディナン・ド・ソシュール(1857年-1913年)。スイスの言語学者。記号論を基礎付け、構造主義思想に影響を与えた。「近代言語学の父」といわれる。
注4) ラジカセを想像する人もいるかもしれないが、1980年代の前半には録音できるポータブルカセット(ウォークマンなど)があった。

▼図4　四色ボールペン・メモ法

四色ボールペン・メモ法

1. 黒：相手の言ったこと
2. 青：見出し／テーマ
3. 赤：重要事項（その場で赤で書くのは難しいのであとで囲む）
4. 緑：自分の思ったこと、アイデア

※色にこだわらなくていい／区別が本質

ときもあるやろ。それを緑で書く。まあ、これはわしの色の使い方やから、好みで変えてもええけどな」

「僕は違いますけど、色の区別のつかない人は？」

「下線、波線、囲み、吹き出し、とかを使えばええんやないかな。あとで見分けがつくことが本質や。特に人の話と自分の考えは区別がつくようにしといたほうがええ。お客さんと打ち合わせして、あとで報告書を書かなあかんなんてときもこのメモ法は有効やが、そこでお客さんの言うたことと自分の考えがごっちゃになったらトラブルの元や」

「なるほど。ほかにはありますか？」

「うん。話には順番がある。変にまとめずに順番通りに書いていくのは、その順番というか時系列に意味があるからなんや。ただなあ、人の話というのは、あっちいったりこっちいったりするんが普通や。さっき言うたことと関連することを、突然思い出して言い始めたりするやろ。そういう関連性は矢印で結んでおく。その場では関連があるとわかったことでも、あとで読みなおすと、何でこの人ここでこんな話してるんやろとなりがちや。ノートのページが飛ぶようやったら①とか②とかを使こうてつながりを表したらええ」

伊達直人はここまで言うと、「ああ、のどかわいた」と言って、生ビールの残りを飲み干した。そして、おかわりと焼き鳥を頼んだ。

「ところで、兄ちゃん。わしの話、メモしてへんかったけど大丈夫か？」

「はい。今日のは新しい知識というよりも考え方やったんで、一生懸命聞くほうがええと思いました」

「うん。それでええ。そこは上手に使い分けぇよ。さ。どんどん飲んで、食え」

そしてまたもや、小一時間説教めいた話をされたが、博は心地よかった。なんとなく愛情めいたものを感じたからだ。

伊達直人は今回も勘定を払うと、「ほな、またな」と去って行った。

◆　◆　◆

舞台は現在の「おかめ」に戻る。

「なるほど。僕も愚直に皆さんの言うことをそのまま書いていったら良かったんですね」

才蔵は、ずっと博の話に耳を傾けていたが、一段落ついたのでこう言った。

「そうや。ノートの取り方も板書も基本は一緒や。あとは、色とか矢印も使いこなせれば一番ええ。そやから、うちの会社のホワイトボードは紙のコピーやのうて、PCにカラーのまま取りこめるようになってるんや」と博は答える。

「でも高かったんでしょう？」

「まあ、紙代のランニングコストを考えたらお得かも知れんがな。事業部長の八破原常務に議事録はこうしたいと言ったら、二つ返事でOKしてくれたよ」

「ところで、そのあと中野さんにはどう謝ったんですか？」

「それやけどな。思い出すと泣けてくるな」

博は、ハンカチを取り出し、目頭を押さえた。

「次の日おそるおそる近寄ったんやけど口きいてくれへんねん。それで、ほとぼりが冷めるまで耐え忍ぼうと覚悟を決めたら、机の中に封筒が入ってたんや。俺がそれを見つけたとたんに中野さんはどっかに行ってしもたんや。変やなと思ったんやけど、封筒の中身を見て理由がわかった。照れくさかったんやな」

「何が入ってたんです？」

「A4用紙で10枚ぐらいのノートのコピーと手紙やった」

手紙にはこう書いてあった。「昨日は突然怒り

出してごめん。これは私が新人時代に八破原課長からアセンブラマクロのレクチャーを受けたときのノートです。字は汚いけど、わかりやすいと思います。これを読んでもわからないことがあったら聞いてください」

「ほんまわかりやすいノートやった。こうやってノートって取るんやなという見本のようなノートやったな。俺もすぐに手紙を書いたよ。『本当に助かりました。ありがとうございました。お礼にいっぱいおごりますので、今日の帰りに飲みにいきましょう』って」

「それで、どうなりました？」

「うん。新人時代でお金もなかったんや。で、ここに連れてきた。そしたら、また怒られてん。『あこがれの美人先輩をこんな貧乏くさい店に連れてくるってどういうこっちゃ？　しかも"おかめ"[注5]やと？　バカにしてるんか！？』。そう言いながら、むちゃくちゃ飲んで、最後はからんできたけどな」

「ところで、前回も気になったんですが、中野先輩という方はもう退職されたんですか？」

「ん？　寿退職したとか？　そんなタマか。そうか名字が変わったから、わからへんねんな。人事に高木部長っておるやろ。あの人や」

「あの～」

「何？」

「その高木部長なら、さきほどから次長の後ろの席で飲んでおられますが……」

振り返ると、高木人事部長が博をにらみつけていた。

「出来内クン、キミ偉くなったねぇ」

「あ。いや、その」

「誰が"タマ"やねん？」

「いえ。あの……」

「今度のボーナス覚悟しときや」。そう言って高木はギャハハと笑った。

「どうやら許してもらえたみたい」と博は小声で才蔵に言う。

「それからなあ、恥ずかしい昔話は盆と正月ぐらいにしときやぁ」と高木部長がとどめを刺した。ずっと聞き耳を立てていたらしい。

「次長にも頭の上がらない人がいるんですね」と才蔵。

「そんな人ばっかりや。ところでな、君も最近はかなりわかってきたようやけど、そろそろ人に教えるということも意識せなあかんよ」

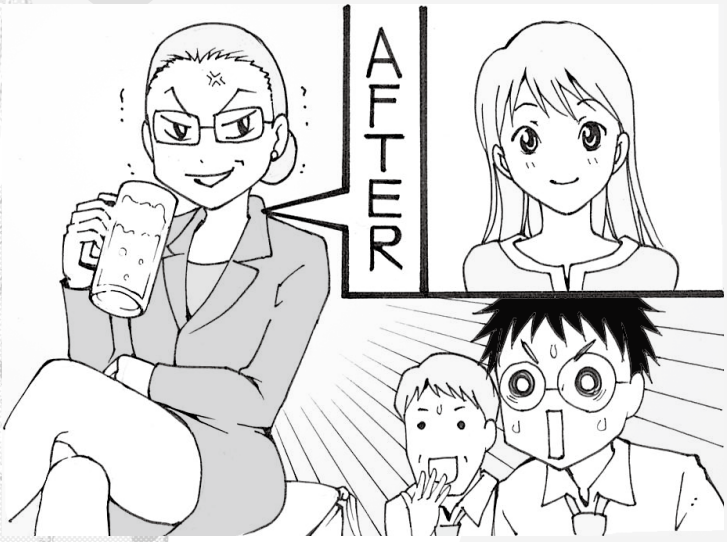
「はあ」

「協力会社の人に教えなあかんという実務的な意味もあるが、うちのグループでは2年目になったら新人研修の講師をするのが伝統なんや」

「えっ。ほんまですか？」

「ああ。今年はたまたま2年目の社員がおらんかったから高飛車にお願いしたけど、来年は君にやってもらわんとあかん。今から準備しとかんとなあ」

一難去ってまた一難。次長はグループの伝統というけれど、これって先輩への恨みを後輩で晴らすというシステムではないかと疑問を持った才蔵であった。



注5）大阪弁では、変な顔の女性のこと。

Column 四色ボールペン・メモ法

マインドマップや「東大合格生のノート」などがもてはやされています。僕はそういうものを否定するつもりはありません。

マインドマップはアイデアを出したり、思考をクリアにしたりするのにはすぐれたツールだと思いますし、「東大合格生のノート」も、すでに知っていることのまとめなおしという意味では非常にすぐれたものだと思います。

ただ、議事録をマインドマップで書くというのは違うと思うのです。打ち合わせには時間軸があり、経緯が重要なことが多いからです。最後のまとめをマインドマップ化するのであれば、それも一つの方法とは思いますが、1枚のマインドマップで議事録だとするのは無理があります。

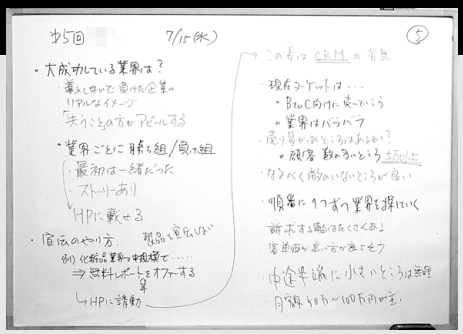
またノートについても、講義の最中に話を取捨選択して、きれいにまとめながら書くというのはかなり難しいと思うのです。講義の内容があらかじめわかっているとか、よほど頭がいいとか、そういうことならわからなくもない。しかし、そうであれば講義を受ける必要があるのでしょうか？

もちろん考えを整理し、体系化するために受ける講義や研修もあります。その際には「東大合格生のノート」は模範的でしょう。でも、日々新しい知識を得るために受ける講義やレクチャーに関しては、「東大合格生のノート」は無理があるというのが僕の考えです。

ところで本来議事録とノートは違うものですから別々に議論すべきなのかもしれません。ただ、僕の場合はたまたま似たような方法論だったので、同じ話としてまとめました。もしそこで混乱してしまった人がいたら申し訳ありません。

ただ、似たような方法論になる理由はあります。それは、仕事の現場において人から話を聞くことはすべて勉強であるという考えが、僕にはあるからなのです。

たとえば、今回の話のように先輩が一通りの技術的なレクチャーをしてくれることがあります。これは間違いなく勉強です。また、客先でヒアリングをすることがあります。これも必ず新しい知識が得られるわけで、やはり勉強なのです。そして、会議もそうなのです。会議というものも新しい知識・情報が得られる場です。というより、そういう会議でないと意味がないと思っています。そういう場でのノートや板書も他の勉強と同じように愚直に書きとっていくべきだと思うのです。



▲写真1　筆者の板書

僕は、仕事でファシリテーターをすることがあります。僕の流儀は板書をしながらファシリテーションをするというものです。終わったあと板書がうまいねと褒めていただくことが多いのですが、何のことはない。伊達直人式メモの取り方を実践しているに過ぎません(**写真1**)。

少なくとも学んでいる間は、愚直に写すというのが一番効率的です。よくセミナーをICレコーダーで録音している人がいます(もちろん許可を取る必要がありますが)。何度も聞きなおす時間のある人はそれでも構わないと思います。ただ、ほとんどの人はそんなに時間を取れないでしょう。僕も取材に行くときは、保険として録音しますが、聞きなおすことはほとんどありません。基本的にメモから記事を起こします。録音に頼ると、(僕の場合は)どうしても真剣味が失われるからです。

愚直に書くという意味では、本文では触れませんでしたが、インストール記録なども一挙手一投足をメモするのが良いでしょう。これは、僕のようなインフラ技術者には財産になりました。

メモするときはとにかく考えないことです。何を残そうか・どうまとめようかなどと判断している間にも、相手はどんどん話を続けます。その分漏れます。考えないことが一番能率的なのです。人は考えていなくても、突然アイデアが出ることがあります。それも書き留めておけばいい。

愚直が一番効率的なのです。

第3章 人に教えてはじめてわかる

入社2年目の春。青二才蔵は、少し慢心していた。

同期30人ぐらいと万博公園で花見をしたときに、仕事や職場の話になった。そのときに同期の話が少し幼稚に感じられたのだった。ああ、こいつらは伊達直人式速読法も、四色ボールペン・メモ法も知らへんのやなと気がついて、ずいぶん差がついたんやないかと思った。何しろ自分はあれからすでに100冊の本を読み、その分の読書ノートを残した。仕事の記録を愚直に書きつづった大学ノートももう4冊目が終わろうとしている。蓄積が違うんや。

自信を持つのはいいことだ。しかし、慢心はいけない。仕事にムラができる。一時期は育成担当だった高飛車に叱られることも少なくなったのだが、花見のあと急に叱られることが増えた。

以前の才蔵だったら、すなおに反省していた。自分はまだまだだという自覚があったからだ。しかし、今は慢心しているので反省どころか反発することが多かった。誤字脱字を叱られても、そんなんある意味しかたないやないか、そのためにチェックする人がおるんやろ。こんな具合だった。

グループリーダーの出来内博と飲みにいって鬱憤を晴らしたいと思うこともあったが、最近は忙しいらしく誘ってくれない。

才蔵はグレてやると思ったこともあったが、グレる度胸はなかった。なので不満を抱えながら日々過ごしていた。

ゴールデンウィークが明けた。1週間後には新人が部門に配属されてくる。最初に1週間の事業部研修があり、その後部門研修になる。その部門研修の講師の1コマを、2年目ながら才蔵は務めることになっていた。彼の担当は、TCP/IPの基本をルーターなどの通信機器と絡めて説明するということだった。

今の才蔵には簡単なことだった。なので、ギリギリになってから準備すればいいと思っていた。ところが、高飛車が明後日にリハーサルをすると突然通告してきた。

まあ1日あれば準備できるやろ、幸い急ぎの仕事もないし。それに昨年の夏に、出来内次長から人に教える準備をしておけと言われた経緯もある。それなりに用意はしていたんや、と高をくくっていたのがいけなかった。

「何を教えたいんか、さっぱりわからん。ちゃんと準備せい。テキストも一から作り直しじゃあ！」と高飛車に言われてしまったのだった。

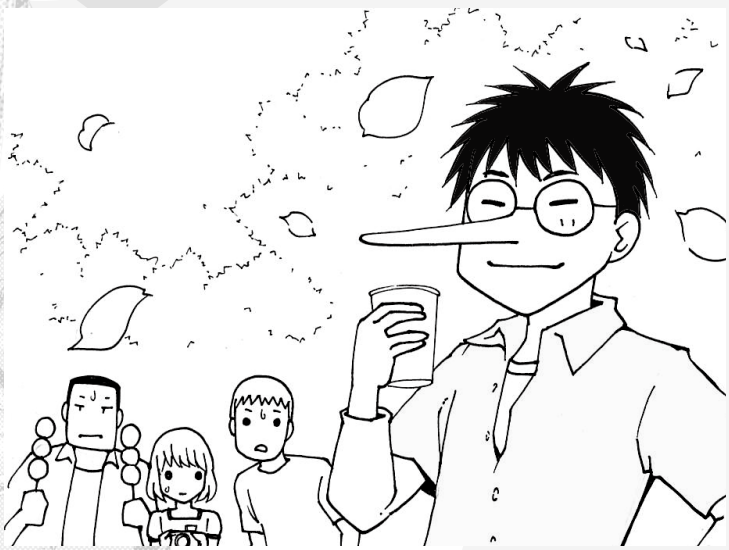
「どう直せばいいんですか？」と質問しても取り合ってくれない。「自分で考えろや」と返される。

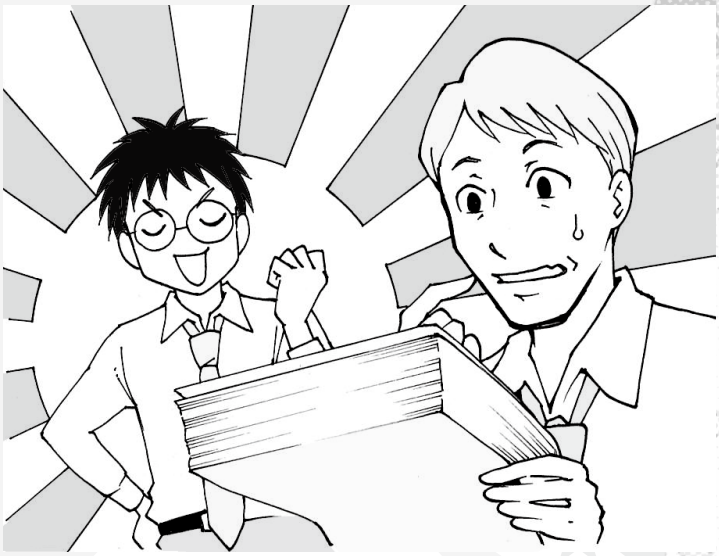
才蔵は、約1年前の配属当時に戻った気分だった。

その夜。西中島南方の居酒屋「おかめ」で、青二才蔵と出来内博がビールで乾杯していた。久しぶりに博が誘ったのだった。

「高飛車さんは、僕に何か恨みでもあるんでしょうか？」

「はあ。何のことや？」





才蔵は、今日のリハーサルの一件を語った。博はテキストを見せてごらんと言う。それは分厚い、内容豊富なテキストだった。才蔵の自信作である。

「うーん。青二君は新人に何が教えたいんや？」

「TCP/IPに関して僕が知っているすべてです」

「君のコマは何時間だったっけ？」と博は知っていながら聞く。

「午後の3時間です」

「3時間でこんだけ詰め込もうということかい？」

「そうです」

「じゃや、このテキストを読んどけというのとほとんど変わらんよなあ」

才蔵は黙ってしまった。確かにそうだ。講師は要らない気がしてきた。

「3時間って長いようやけど、せいぜい1つか2つしか教えられへんもんやで」

「はあ」

「まあ、しばらく俺の昔話に付き合え」

博は、きょろきょろと周りを見回し、自社の社員がいないことを確かめてから話を始めた。

◆ ◆ ◆

1988年の春。出来内博は慢心していた。伊達直人式速読法で100冊の本を読み、仕事でした作業の一挙手一投足を書き留めた自称出来内ノートがもう5冊目になっていた。蓄積が違う。同期ともはっきり差がついたと感じていた。

そんなときだった。元育成担当の司五斗優からこう言われたのだった。

「今年から、部門研修の講師は2年目、3年目を中心にやってもらうことになった。八破原課長が今日の役職会議で提案して部長がOKしたんやそうや。キミはVTAMを新人に教えてくれ」

博のグループの仕事が最近増えてきた。新人の配属も増えるらしい。だったら部門研修でVTAMを教えといたほうがいいだろうということになったのだそうだ。

横で聞いていた中野あかねが、「出来内クン、すごいやん。先生やて。出世したなあ。ギャハハ」とバカにするように笑った。

「ふふふふ。まあ、任せてください」と博は返した。「おおっ！なんや、この自信は。さては慢心というやつやな」と中野も返す。

「ついては明後日リハーサルをやる。明日はほかの仕事はええから、テキストを作ってくれ」と司五斗が宣言した。

急なことだったが、博には本当に自信があった。VTAMを新人に教えるぐらいなんてことはない。俺の知っていることをすべて教えつくしてやる。

そして、二日後。博は久しぶりに伊達直人のポケベルに連絡を入れたのだった。

夕方。居酒屋「おかめ」で、博はビールを飲みながら伊達直人を待っていた。ポケベルを入れてから1時間後に現れる約束だった。

きっちり1時間後。立てつけの悪い横開きのドアを乱暴に開けながら、濃紺のダークスーツに虎のマスクの伊達直人がやってきた。走ってきたのか息が荒い。なんて律儀な人なんだろうと博は感心した。

「すまん。待たせたか？」

「いや。ぴったりです。ありがとうございます」

「うむ。とりあえずビールや。で、何があった？」

博は、今日の午後の新人研修のリハーサルの件について語った。

自信満々でリハーサルの場に臨んだのであったが、相手を務めてくれた司五斗も中野も全否定。中野は「新人にそんなん言うてわかるかぁ！」だし、司五斗に至っては「最近、ようわかってきたんかなと思ったけど、失望したわ」である。そして、例のごとく「何が悪かったんでしょうか？」と聞いても、口をそろえて「自分で考えろ！」だった。

「『新人にわかるかぁ』と言われても、僕だってわからんけど、辛い思いをして勉強したことなんです。それもあって、僕が知ってることは全部伝えたいと思ったんですよ」

「まあ、気持ちはわからんでもないけどな。とりあえずテキストを見してみいや」

それは100ページぐらいあるテキストだった。普及し始めたばかりの当時のワープロで作ったので、ほとんどが箇条書きの文字だった。ときどき手書きの図表が入っている。

「うーん。これはすごいなあ。あれからよう勉強したんやな。この1年の蓄積がようわかるで。で、これを何時間で教えるんや？」

「3時間です」

「ああ。だったら講師は要らんな。このテキストだけ渡せばええ」

「何でです？」

「3時間でこれだけ教えるのは無理やからや。逆になーんにも残らん。それやったら、これをじっくり読んどけちゅうほうがよっぽどましや」

そんなもんだろうか。博はまだ疑問に思っていた。

「あのなあ。3時間って、1つか2つ教えるのがせいぜいの時間やねん。あ。知識って意味やないで。ポイントという意味や。このテキストからは、ポイントがまったく伝わってこないんや」

（ポイントかあ。そう言えば何がポイントなんやろう？　VTAMの初心者がまず知らないといけないことやろうなあ。それって何やったかな？）博が自問自答していると、伊達直人が続けて言う。

「ポイントを1つか2つに絞れるってことは、そのことについてよくわかってるってことと同じや。つまりポイントが絞れんうちは、まだまだわかってないということになるな」

「なるほど。僕は知識が蓄積されれば一人前だと思ってましたけど、そうではなかったんですね」

博は自分の慢心に気がついた。知っているだけで、わかっていなかったのだ。

「うむ。そういうことや。だから、名人や達人

と言われる人たちは、自分の奥義を一言で語ろうと腐心するんや」

たとえば、柳生宗矩の『兵法家伝書』という書物がある。新陰流の極意を記したものであり、最後に「一心多事に渉り、多事一心に収まる。畢竟茲(ここ)に在り」と書かれている。つまり兵法(剣法)とは心の問題というのである。もちろん、これだけでは兵法についてはまったくわからないので、ノウハウの部分も同書には書かれているのだが、しかしこの結論部分を忘れると本当の意味では役に立たない。こういう最重要ポイントがはっきりと言えるということがわかっているということだ、と伊達直人は説く。

「ちょっと難しい話になったけど、なんとなくそんなもんやとわかってくれたらいい。それにここまでの達人になろうというのは、しょせん無理なことやしな。でも、心がけていたら近づくことはできるんやないかと、わしは思ってる。柳生宗矩も50歳を過ぎてからようやくわかってきたと書いてたし」

「はあ。道は遠いですね」

「うむ。道は遠い」

「ポイントについては考えてみます。それをどう教えたらいいでしょうか？」

「一つは、話の軸をはっきりさせることやな。そのVTAMっちゅうのはようわからんが、一言でいえば何をしてるもんなんや？」

「VTAMというのはホストコンピュータ上で動くソフトウェアなんです。端末から受け取ったデータを適切なアプリケーションソフトに渡し、アプリケーションソフトから受け取ったデータを適切な端末に送ります」

「ふむ。中継役ということかな？　でも、VTAM以前にも端末やアプリケーションソフトはあったんやろ。そうやったら中継役もおったんとちゃうかな？」

「おや。なんだか詳しいですね」

「いや。素人がヤマ感で言うただけや」

「実際そうなんです。ただ、以前は端末とアプリケーションは密接に結びついていて融通性がありませんでした。VTAMのVはVirtualつまり仮想的という意味で、アプリケーションは元々ソフトウェアですが、端末も仮想的なもの、たとえばプログラムなどでもいいということになんです。つまり、アプリケーション同士の通信なども可能となり、以前よりずっと融通性が高くなって、可能性が広がったんです」

「ふむ。いま語ってくれたことがVTAMのポイントとちゃうか？　もしそうやったら、兄ちゃんはVTAMのことが本当はわかってたんや。だけど、言語化できてなかった。それが、わしに教えたことで言語化できたってことやないやろか？」

「ほんまにそうです。僕はいまはじめてVTAMのことがわかった気がします。なんか感動してます」

「それが人にものを教える効用なんや。人に教えようと苦心して、はじめて自分でもわかる。ただ知識を書き連ねても、俺はこんだけのことを知ってるわいという自慢に過ぎへん。そこを叱られたんとちゃうやろか？」

本当にそうだと博は思った。先輩らは理不尽に怒ってたわけやなかったんや。

「人間、わかってることしか伝えられへん。さっきまでのままやったら、新人たちからもブーイングやったやろうなあ。そして、兄ちゃんはその理由がわからんまま、新人たちをアホやと決めつけてたかもしれんなあ」

そうやとしたらゾッとする。いや、たぶんそうなってたやろう。何しろ慢心してたんやから。

「さて、いま教えてくれたポイントを軸にすればええ。それをどうやって伝えるかや」

「そうですねえ。いまのやり取りからすると、以前とどう変わったかという話が必要だと思います」

「そうやな。出てきた背景とそれで変わったこと。つまり効用とかメリットとかかな」

「もちろん。VTAM自体が何か、つまりしくみの話も要りますね」

「うむ。あとは何やろ？」

「具体的なイメージでしょうね。たとえばVTAMを利用するときのプログラムの例とか」

▼表1　教えるときの5W1H

Why	必須	技術が出てきた背景（Before/After） 昔はどうだった？ この技術でどう変わった？
What		技術そのものの話〜理論・特徴・しくみなど 受講者が知っていることになぞらえて話すとよい
How		技術の使われ方、方法論 具体的なコーディング例、使い方のコツなど
Where		どこで使われているか？ 事例など
Who	派生	Why、Whereなどで派生的に使う 例）VTAMはIBMが考案した
When		Why、Whereなどで派生的に使う 例）1987年のアプリケーション間接続の事例

「そうやな。具体的な話もないと伝わらへん。そういう意味では、しくみの話をするときにも、新人たちにわかる日常的な話になぞらえて説明してやることも必要やろな」

「なるほど。あとは事例とかでしょうか。うちのグループで実際にやった仕事を例として説明する。こんなことができるんやよ、って」

「うむ。たぶん、そのぐらいでええやろ。5W1Hというが、ポイントを教えようと思ったら、それにまつわるWHY、WHAT、HOW、WHEREを説明することや。ちなみにこの場合のWHEREはどこで使われてるか、つまり事例やね。残りのWHO、WHENは付随的な項目と思ってええ。たとえばWHYを説明するときには、WHOやWHENの話もするやろ。でもメインの話ではないちゅうこっちゃ」（**表1**）

伊達直人は、ビールを飲み干し、お代わりを頼んでから続けた。

「あとは、最後にまとめを入れるぐらいかな。これは結構重要なことなんや」

「どうしてです？」

「人間の記憶は時間とともに失われる。振り返りがないと、翌日にはほとんど忘れてしまう。しかし、最後にここが一番重要やったんやでえともう一度言えば、そこだけは結構長い間定着するもんなんや。だから、ポイントを最後にもう1回言うのが大切なんや」

「そういう意味では、演習問題なんかもあったほうがいいですよね？」

「おお。それを言うのを忘れてた。それも記憶の定着という意味で重要なんや。ぜひ入れてや」

「はい」

「そう考えたら、3時間なんてあっちゅう間やろ」

「はい。あんなに教えられると思っていたのが恥ずかしくなってきました」

「そうや。ええこと思いついた。まとめに入る前に2、3分時間を取って、今日教わったことを一言でまとめるという演習をやってもいいかもしれんな。それで2、3人を指名して答えさせる」

「それをやる理由は僕に言わせてください」

「おう。そうきたか」

「人に教えられるということがわかったということだからですよね」

「よっしゃあ！合格や。ええ研修になるやろ。わしも陰ながら祈ってるで！」

「ありがとうございます。忘れんうちにいまの話をメモしときます」（**図5**）

▼図5　講義の組み立て方

講義の組み立て方

1. 軸を決める（3時間で1つか2つ）
   軸とは、最重要ポイントの言語化
   （ひとことでいえば）
2. 軸ごとに5W1Hを考える
3. 記憶を定着させるための演習問題を考える
4. まとめの言葉を考える
   これも記憶定着のため
   受講者にまとめをさせる演習も効果的

◆　◆　◆

舞台は再び現在の居酒屋「おかめ」。

「そうか。僕も慢心してたんですね。それで高飛車さんが最近厳しかったんや」と青二才蔵。

「うん。俺もそれは感じてたんや。でも、こればっかりは自分で気づかんとなあ」と出来内博が返す。

「それで、今日誘ってくれたんですね」

「それもあるけど、それよりも新人研修のほうが気になってた。このままほっといたら青二君がつぶれそうな気がしてな」

「本当にありがとうございます。このままやったらと思うとゾッとします」

「今の話でわかったかな？」

「はい。大丈夫やと思います」

「よかったで。ほな、がんばりや。期待してるで」

「ありがとうございます。ところで、研修はうまく行ったんですか？」

「ん？　まあ。アンケートを見る限りでは良かったんやないかな。まあ、その話はええ。今日は青二君の講義の成功の前祝いに飲もうやないか」

Column　講義の組み立て方

僕自身が長い間"知識満載系"の研修をやってきたので、この話はかなり自虐系でした。たくさんのものを持ち帰ってもらうことがサービスだと思っていたのです。しかし、これはまったく逆でした。知識を詰め込めば詰め込むほど評判が悪くなっていったのです。

なぜ、そうなるのかを聞いてみました。すると多過ぎて何も残らないとのこと。

それであるとき、ポイントを絞って、ほかはばっさり切り捨ててみたのです。そうしたら、かなりの高評価をいただきました。人はたくさんの知識よりも、何が重要かとか、今すぐ使えることは何かとかを知りたいものだとはっきり認識しました。

とはいえ、たとえばSQL入門といったような、研修自体よりもテキストのほうが価値が高いものもあります。そういう研修については、テキストは知識満載でいいと思います。その中で何を持ち帰ってもらうかを考えて、実際に持ち帰ってもらえるかどうかが講師の腕前なのだと思います。

さて、本文にも書いたように、ポイントを絞る研修を実施するためには、まずは講師がポイントをわかっていないといけません。何がポイントなのかが見えることは、そのことについてわかっていることと同じです。その意味で、「わかろうと思ったら、人に教える苦心をすること」、これに尽きるのではないでしょうか。

研修講師でなくてもかまいません。部下や協力会社の社員に個別に教えるということでもいい。そういう機会がなければ、ブログ等に技術解説を書くという手もあります。顧客に説明するのも広い意味で教えるということになるでしょう。特に顧客に説明する場合は、ポイントが明確に言語化されていなければまったく通じません。最高の勉強の機会だと思います。

ところで、本文に挙げた「教えるときの5W1H」を検証しようと思って、本誌のバックナンバーを読んでみました。新技術の紹介記事などでわかりやすいなあと思ったのは、ほぼこの原則通りに書かれているようです。教える立場の人は意識的か無意識なのかはわかりませんが、こういった原則を踏まえているものだとあらためて思いました。

Epilogue

エピローグ

「この1年よう頑張ったなあ」

高飛車徹が握手を求めてきたので、青二才蔵はちょっと気味悪く思いながらも握手を返した。

居酒屋「おかめ」を借り切っての部門研修スタッフの打ち上げの席のことだった。

「新人のアンケートを見て、ボクは涙が出そうになったで」と高飛車。

才蔵の研修の評判は上々だった。感想欄にはこんなことが書かれていた。

「TCP/IPの本質がわかった気がしました。これを基礎にいろいろと自分で勉強していけそうに思います。」

「2年目でこんな優秀な先輩がいるんだなあ、私も頑張らないといけないと思いました。」

「全社の集合研修でもTCP/IPの基礎を習いましたが、そのときはピンときませんでした。それがこんなにクリアになるなんて思ってもいませんでした。ありがとうございました。」

「とにかく愚直にノートを取れと言われて、何でそんなことをしないといけないのか最初は疑問でしたが、終わってみたら今後の財産になるノートが取れたと思います。」

否定的なコメントはなく、感謝や称賛ばかりだった。

「ほんま、この1年間辛かったで」

「えっ？　何がですか？」

「うちのグループの伝統でな、先輩社員は1年間は新入社員に辛くあたらんといかんことになってるんや」

「な、何でですか？」

「知らんけど、そのほうが新人が育つという出来内次長の方針なんよ」

出来内博は、才蔵と高飛車の会話を横で聞きながら、(そう。25年以上続いているわがグループの伝統なんや)と心の中でつぶやいていた。そして25年前のことを思い出していた。

◆　◆　◆

1988年6月中旬。出来内博の所属する通信技術部の新人研修スタッフは、居酒屋「おかめ」を借り切って、研修の打ち上げ宴会をしていた。

司五斗優が博に握手を求めてきた。

「ほんま、この1年よう耐えたな」

「え？　何のことです」

中野あかねが引き取って説明する。

「去年な、キミらが配属されてくる前に、八破原課長が私たちを集めて相談されたんよ」

「『今年から文系の新卒男子が配属されてくる。正直、どうやって育成したらいいか、迷ってるんや』ってな」と司五斗。

手取り足とり教えるのがいいのか、突き放して自力で這い上がるようにするのがいいのか、八破原はどちらにすべきか悩んでいたのだという。

「三人で2時間ぐらい議論してん。結論は、司五斗さんと私が鬼になって鍛える、その代わり課長が陰でフォローする、そういう作戦やったんよ」

「出来内は、僕のこと鬼やと思ったかもしれんけど、ほんまは心優しい人間なんで、辛かったわ」と司五斗。

「私もやで。親切で気立てのいいあかねさんで有名なのに……」

「いや、中野は楽しんどった」
「ギャハハ。多少はそうかも」
そういうことだったのか。博は涙が出た。涙もろいのだ。
(ん？待てよ。そしたら、あの伊達直人というのは？？？)
そこに八破原がやってきた。
「出来内君。研修のアンケート結果見たで。ようやったなあ。俺も誇らしいわ」
「あの〜。伊達直人って、もしかして」
「えっ？　何のことや？　俺は人前で虎のマスクなんかかぶったりせえへんで」
「それって、自分で正体バラしてますやん……」と中野が突っ込む。
「うわ、しもた」
4人は大声で笑った。博だけは泣き笑いだったが。

◆ ◆ ◆

舞台は再び現在の居酒屋「おかめ」に戻る。
博は25年前のことを思い出しながら、(あれ以来、新人には1年間だけつらく当たるという伝統ができたんや。中にはつらくて辞めた子もおったけど、ほとんどが順調に育ってる。間違いではないはずや)と振り返っていた。
「そやけど、一人だけええかっこしいがおるな」
まるで博の心境を見透かしたかのような発言が聞こえてくる。高木(旧姓中野)あかね人事部長の声だった。
「げげっ。何をしにきはったんです」とうろたえる博。
「出来内クンが、"伝統"をないがしろにしてたから、一言注意しようと思ってな」
そこに、事業部長である八破原常務もやってきた。
「なんや、高木部長？　出来内がなんか粗相をしたんか？」
「それが常務。出来内次長ときたら肝心なことをしてなかったんですよ」
高木は、虎のマスクを被らずに新人にアドバイスしていたところを目撃したことを告げた。

「どうも高飛車クンだけを悪者にして、自分はええ上司を気取りたかったみたいです」
「何やと〜！　あれが一番肝心なんやんけ。来年は部長に推薦しようと思ってたけど、やめや。きちっと虎のマスクを被ってからやぁ！」

"伝統"について明かされたことで温かい気持ちに包まれていた才蔵だったが、このやり取りを見ていると、自分も将来虎のマスクを被らされる気がしてきた。そして、そろそろこんな"伝統"は見直したほうがいいのではと真剣に思うのだった。
SD

筆者プロフィール◆森川 滋之(もりかわ しげゆき)

ITブレークスルー代表。1987年、バブル直前に文学部卒で大手独立系ITベンダーに採用された"文系SE"の草分け。配属直後に"劣等生"の烙印を押されるが、「おまえは技術ではトップになれない。プロジェクトマネジメントに活路を見いだせ。ただし、マネージャとしての迫力があるだけの技術は身につけろ」と言われて一念発起。「技術に強いマネージャ」として、アプリケーション・インフラ構築チームのリーダーを歴任する。現在では、最新のITも業界事情も"なんとなく"わかるという特技を活かして、IT企業のPR記事のライティングやIT系媒体への執筆に従事する。またIT企業向けに、研修講師やチームビルディングのファシリテーターなども務める。著書に『SEのための価値ある「仕事の設計」学』(技術評論社)などがある。

緊急レポート

Samba 4.0.0がやってきた!

SambaによるActive Directoryドメインの構築

たかはしもとのぶ TAKAHASHI Motonobu
monyo@monyo.com／TwitterID：@damemonyo／facebook.com/takahashi.motonobu

2012年12月11日、次世代版となるSamba 4系列(以下Samba 4)の最初のバージョンであるSamba 4.0.0がついにリリースされました。本稿では、新機能のインストール方法を中心に、解説を行っていきます。

Samba 4とは

Samba 4の特徴を一言で表すとすると、**Active Directoryの Domain Service機能(以下ADDS)の実装**に尽きます。これまでのSamba 3系列でもActive Directory(以下AD)への参加機能やAD以前のディレクトリであるNTドメインの機能は実装されていましたが、ADDSは提供できておらず、長らく懸案となっていました。

Samba 4の提供形態を理解する意味でも、Samba 4リリースに至るまでの経緯について、簡単に紹介します。

Samba 4開発の着手と混迷

Samba 4の開発が着手されたのは、2003年のことです。当時はADへの参加機能を目玉としたSamba 3.0.0が無事リリースされた直後であり、ADDSの実装を目玉とするSamba 4も比較的早期にリリースされるものと期待されていました。

しかし、開発は長期化します。最低限のADDSを実装したTP(Technical Preview)版がリリースされたのが2005年、そこからリリースを重ねますが、ADDSの複雑さの前に開発は行き詰まり感を見せていきます。一方Windows Serverは2003→2003R2→2008とリリースを続けていきます。また、Samba 3系列もリリースを重ね、本来Samba 4で実装する予定だった機能を徐々に取り込んでいきます。最終的にADDS以外、Samba 4で実装する予定だった機能の大半がSamba 3系列に取り込まれました。

そんな中、Samba 4の開発に大きな影響を与えるイベントが発生します。2007年12月、EUからMicrosoft社に対して、独占禁止法違反の是正命令が発せられ、技術情報の開示が義務付けられます。開示された技術情報により、Samba 4の開発がようやく加速しました。

そしてデュアルバイナリ化へ

2009年7月にリリースされたSamba 3.4.0では、Samba 4のソースコードが同梱される形で、ソースコードの共有化が始まり、Samba 4のリリースに向けた期待感が高まります。

ところで、Samba 4では従来のsmbd/nmbd/winbinddといったレガシーバイナリ[注1]の機能を**samba**という単一のバイナリに統合することが

注1) 筆者の造語です。

謳われていたものの、開発はADDSとそれに付随する各種機能に注力して行われてきました。さらに2011年6月にリリースされたSamba 4.0.0alpha16からは、「暫定」としてレガシーバイナリもリリースされるという「デュアルバイナリ」形態となります。その後、実環境での運用例もぼちぼち見られるようになりますが、基本的にsambaバイナリはADDSに特化させ、ファイルサーバ機能などは並行してリリースされているSamba 3系列を用いる形がSamba Teamとしての推奨となりました。

2011年後半からSamba 4のリリース形態や時期についての議論が断続的に行われるようになります。sambaバイナリでレガシーバイナリの機能をすべてサポートできるまでリリースすべきでないという意見もありましたが、ADDSを提供するSambaの早期リリースに対する要望が高いこともあり、ADDSに特化したsambaバイナリとその他の機能をサポートするレガシーバイナリを同梱させ、sambaバイナリのファイルサーバ機能としては裏で起動したsmbdに処理を委任する[注2]という、デュアルバイナリ形態でリリースされることになりました(**図1**)。

このように、Samba 4は、Samba 3系列の流れを汲むsmbd/nmbd/winbinddといったレガシーバイナリと、Samba 4で新たに加わった、ADDSを提供するためのsambaバイナリが同梱された形で提供されています。レガシーバイナリの機能については、以下の点を除き、Samba 3.6までとほとんど変わるところがありません。

- security = share およびsecurity = serverが廃止された
- SMB2.1(大半の機能)やSMB3.0(基本的な機能)が新たにサポートされた
- CTDBによるクラスタが正式にサポートされた

ここからはsambaバイナリ(以下samba)に焦点を絞って紹介していきます。

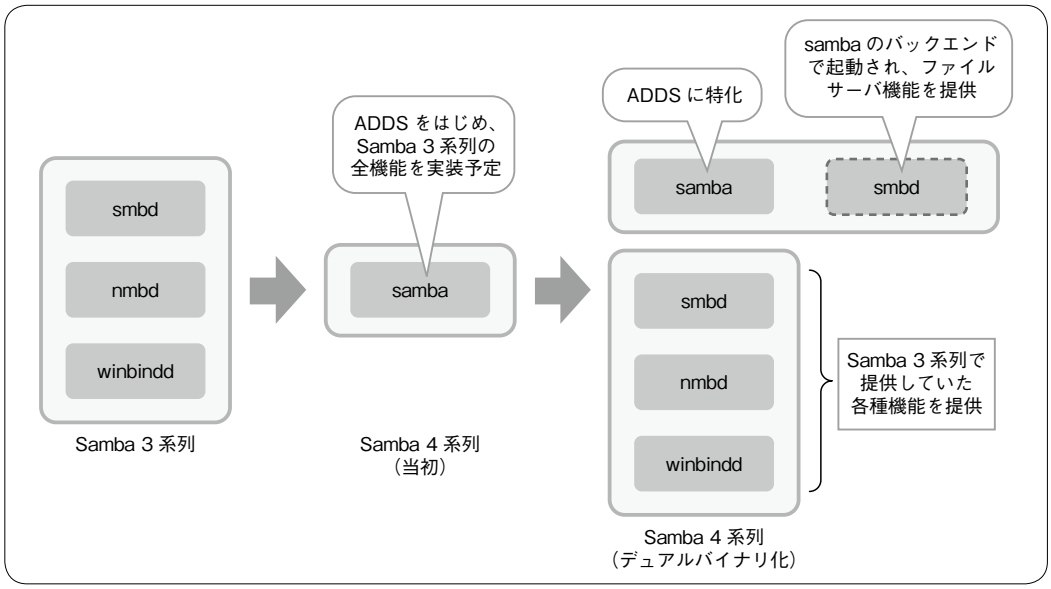
Samba 4の機能

sambaの主要な機能を**表1**に示します。

一見してわかるとおり、提供する機能はADDSに絞られており、ファイルサーバとしての使用は非推奨、プリンタサーバ機能は未実装となっています。ADDSの機能に限っても、お

注2) s3fsと呼ばれています。

▼図1　デュアルバイナリ形態への変遷



もに大規模環境向けの機能は未サポートのものが多いですが、おいおいサポートされていく予定のようです。個別の機能の詳細については、ADDSに関するMicrosoft社のWebサイトや参考文献をご参照ください :-)

Samba 4のインストール

それでは、Samba 4をビルド、インストールしてみましょう。レガシーバイナリを意識する必要がなければ、ビルドは非常に簡単です。

ビルドに必要なパッケージのインストール

sambaでは、それまで別のライブラリを用いていたKerberosやLDAP(Lightweight Directory Access Protocol)が内蔵になり、パッケージのインストールが不要となりましたが、一方で新たにPythonが必須となっています。

CentOS 6とSqueeze(Debian 6)で最低限必要なパッケージを**表2**に示します。

なお、レガシーバイナリでさまざまな機能をサポートさせる場合は、これ以外にもKerberosやLDAP関連のパッケージなどが必要となる場合があります。

拡張属性、ACLの有効化

Sambaが使用するファイルシステムでは、拡張属性、ACL(Access Control List)を有効化しておく必要があります[注4]。ディストリビューションの方式に基づいて、有効化しておいてください。たとえばCentOS 6では/etc/fstabファイルに対し、**図2**のようにaclとuser_xattrというオプションを追記したうえで再起動する必要があります[注5]。Squeezeでも同様です。

セキュリティ設定

システムでファイアウォールが有効になっている場合は、Sambaが使用するポートをブロックしないように、適切に設定を行う必要があります。同じくSELinuxが有効になっている場合は、Sambaが使用する各リソースに適切な設定

注4) ファイルサーバ機能としてNTVFSを使用する場合、これらは必ずしも必須ではありません。
注5) 再起動せずにファイルシステムのオプションを変更しても、もちろんかまいません。

▼表1　Samba 4の機能

	Samba 4.0.0 sambaバイナリ	Samba 3.6系列、Samba 4.0.0 レガシーバイナリ	(参考)Windows Server 2008 R2
Active Directory認証サーバ	◎	×	◎
ドメインの機能レベル	◎	−	◎
FSMOの操作	○	−	◎
マルチフォレスト、マルチドメイン	×	−	◎
Samba DCの参加	○	−	−
Windows DCの参加	△(未サポート)	−	−
RODC	○	−	◎
ディレクトリ複製	○	−	◎
SYSVOLなどのファイル複製	×(未実装)	−	◎
ADドメインへの参加	×	◎	◎
ファイルサーバ	△(ADDSに必要な範囲)	◎	◎
アクセス許可	○	○	◎
ドメインベースDFS	△(ADDSに必要な範囲)	×	◎
プリンタ共有	×	○	◎
ブラウジング機能	×	◎	○

◎ Windowsと同等のサポート
○ とくに記載がなければ、一部対応していない機能があるが、実用上ほぼ問題ないレベル
△ 一部機能が実装されている、もしくは動作するが、サポートされていないなど
× 動作しない

を行う必要があります。

具体的な設定方法は各システムのドキュメントを参照してください。Sambaの動作確認を行うという観点では、これらの機構はいったん無効化しておき、まずはSambaの正常動作を確認したうえで、徐々に設定を行っていくことをお勧めします。

Sambaのビルド、インストール

必要なパッケージをインストールしたら、いよいよSamba本体のインストールです。Sambaのソースアーカイブはsamba-4.0.0.tar.gzという名称ですので、適宜ダウンロードしてください。sambaを動作させるだけであれば、**図3**のように単にconfigure、makeを行えば十分です[注6]。

注6) Samba 4のレガシーバイナリを使用する場合は、従来のSamba 3系列と同様にいくつかのconfigureオプションを意識して設定する必要があります。

▼表2　ディストリビューションごとの必要パッケージ[注3]

ディストリビューション名	必要パッケージ
CentOS	libattr-devel libacl-devel python-devel krb5-workstation dns-utils
Squeeze	libattr1-dev libacl1-dev python-dev attr krb5-user dnsutils libpopt-dev

注3) このほかにgccやmakeなどビルド自体を行うのに不可欠なパッケージ群も必要です。

▼図2　ファイルシステムのオプション変更(CentOS 6)

```
#
# /etc/fstab
(中略)
#
/dev/mapper/VolGroup-lv_root             /       ext4    defaults,acl,user_xattr       1 1
                                                                 ↑これらのオプションを追加

UUID=23bbc041-41f3-4b05-b065-46784021ce06 /boot  ext4    defaults        1 2
/dev/mapper/VolGroup-lv_swap swap                swap    defaults        0 0
```

▼図4　wafによるビルド表示

```
[3749/3757] Linking default/source3/lib/netapi/examples/share/share_setinfo
[3750/3757] Linking default/source3/lib/netapi/examples/file/file_close
[3751/3757] Linking default/source3/lib/netapi/examples/file/file_getinfo
[3752/3757] Linking default/source3/lib/netapi/examples/file/file_enum
[3753/3757] Linking default/source3/lib/netapi/examples/shutdown/shutdown_init
[3754/3757] Linking default/source3/lib/netapi/examples/shutdown/shutdown_abort
[3755/3757] Linking default/source3/lib/netapi/examples/netlogon/netlogon_contro
l
[3756/3757] Linking default/source3/lib/netapi/examples/netlogon/netlogon_contro
l2
[3757/3757] Linking default/source3/lib/netapi/examples/netlogon/nltest
Waf: Leaving directory `/home/monyo/Work/Samba/Compile/i386/samba-4.0.0-centos6/
bin'
'build' finished successfully (22m5.602s)
monyo@centos63-2:~/Work/Samba/Compile/i386/samba-4.0.0-centos6$
```

※ビルド対象の総数(3757)と、現在何番目のモジュールをビルドしているかが各行の先頭に表示される

Samba 4ではwafというPythonベースのビルドシステムが用いられており、ビルド中は**図4**のような表示が続きます。インストールが完了すると、従来のSambaと同様、デフォルトでは/usr/local/samba以下にSambaを構成するファイルがインストールされます。

Samba 4の設定

現在sambaではADのドメインコントローラのみがサポートされています。構築方法としては、新規に構築するほか、今回は誌面の関係上

▼図3　sambaをインストールする

```
$ tar xzvf samba-4.0.0.tar.gz
$ cd samba-4.0.0
$ ./configure
$ make
$ su
# make install
```

扱いませんが、Samba 3のドメインからのアップグレードもサポートされています。

Samba 4のドメインコントローラとWindowsサーバのドメインコントローラが混在する環境は、基本的な参加機能は動作するものの、現状では未サポートとなっています。

ADにはDNSが必須ですので、ドメインコントローラを構築する際にはDNSとの連携方式を選択する必要があります。次の3通りの方式があります(**図5**)。

①Samba 4の内蔵DNSを使用する

Samba 4にはDNS機能が内蔵されています。これにより、Windowsと同様、AD内に格納されるゾーン情報をもとにDNSサービスを提供することが可能となるほか、BINDなど別のプロダクトが不要となります。もちろんセキュアな動的更新など、Microsoft固有の機能もサポートされています。

②BIND9のプラグインを使用する

Samba 4にはBIND9のDLZ(Dynamic Loadable Zone)機能に対応したモジュールが用意されています。これをBIND9に組み込むことで、①と同じくAD内に格納されるゾーン情報をもとにDNSサービスを提供することが可能となります。

③連携しない

ADに格納されるゾーン情報を用いずに、独自にDNSサーバを構築することもできます。

特段の要件がなければ、①の内蔵DNSを使用すれば良いでしょう。

ドメインコントローラの初期設定

それでは、一番簡単な内蔵DNSを用いる方式を例にとって、SambaでADのドメインコントローラを構築する方法を紹介します。

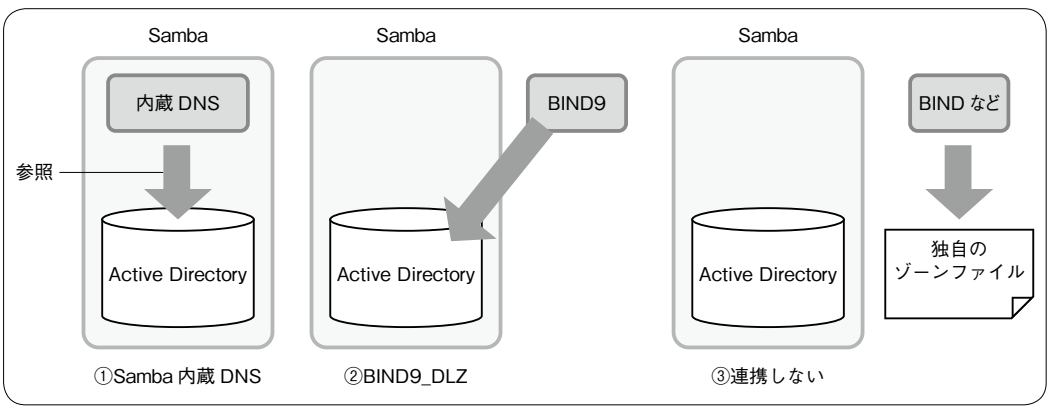
設定はsamba-tool domain provisionスクリプトで行います。--helpオプションを付けて実行すると、さまざまなオプションが表示されますが、大半は通常意識する必要はありません。主なオプションを**表3**に示します。

最低限ドメイン名のFQDNとAdministratorの初期パスワードは設定が必要です。実行例を**図6**に示します[注7]。ここではsamba400.samba.localというFQDNを持つドメインを構築しています。またAdministratorの初期パスワードはP@ssw0rdにしています。

このスクリプトにより、/usr/local/samba/etc/smb.confが自動的に生成されます。なおすでにsmb.confが存在しているとスクリプトが動作しませんので注意してください。またAdministratorの

注7) オプションを付けずにコマンドを実行して、対話的に設定を行うこともできます。

▼図5　DNSとの連携方式



パスワードとして「damedame」のような簡単なものを設定するとエラーになるので、必ず複雑なパスワード(大文字小文字数字記号のうち3種類以上使用)を設定してください。

smb.confと/etc/resolv.confの修正

スクリプトが生成したsmb.confの例を**図7**に示します。

基本的にこのsmb.confに変更を加える必要はありません。ただし、環境によってはdns forwarder行が存在しない、もしくは不適切になっている場合があるので、その場合は適切な値に変更してください。

続いて/etc/resolv.confを、たとえば**図8**のよ

▼表3　samba-tool domain provisionスクリプトの主なオプション

オプション名	意味
--server-role=[dc|member|standalone]	サーバの役割。DCとしてインストールする場合はdcを指定する
--adminpass=*PASSWORD*	Administratorのパスワード
--dns-backend=[SAMBA_INTERNAL|BIND9_DLZ|BIND9_FLATFILE|NONE]	使用するDNSの方式。デフォルトはSAMBA_INTERNAL(Samba内蔵のDNSサーバ)
--domain=*DOMAIN*	ドメイン名(通常はレルム名の先頭部分)
--function-level=[2000|2003|2008|2008_R2]	ドメインの機能レベル(デフォルトは2003)
--realm=*REALM*	レルム名(ドメイン名のFQDN)
--nobody=*USERNAME*	ゲストユーザとして使用するユーザ名(デフォルトはnobody)
--users=*GROUPNAME*	Domain Usersグループにマッピングされるグループ名(デフォルトはusers)
--use-ntvfs	ファイルサーバ機能としてs3fsではなく、NTVFS機能を使用する
--use-rfc2307	UNIX属性を有功にする
--use-xattrs=[yes|no]	NTVFS機能を使用する際に、アクセス許可情報の格納先を指定する

▼図6　ドメインコントローラの初期設定

```
[root@centos63-2 ~]# /usr/local/samba/bin/samba-tool domain provision --domain=samba400
--realm=SAMBA400.SAMBA.LOCAL --adminpass=P@ssw0rd --server-role=dc
Looking up IPv4 addresses
(中略)
Server Role:           active directory domain controller
Hostname:              centos63-2
NetBIOS Domain:        SAMBA400
DNS Domain:            samba400.samba.local
DOMAIN SID:            S-1-5-21-181991817-675121097-4058787377
```

▼図7　smb.confの生成例

```
# Global parameters
[global]
        workgroup = SAMBA400
        realm = SAMBA400.SAMBA.LOCAL
        netbios name = CENTOS63-2
        server role = active directory domain controller
        dns forwarder = 192.168.135.2

[netlogon]
        path = /usr/local/samba/var/locks/sysvol/samba400.samba.local/scripts
        read only = No

[sysvol]
        path = /usr/local/samba/var/locks/sysvol
        read only = No
```

うに設定して、DNSサーバとして自分を設定したうえで、検索対象のドメイン名を先ほど設定したドメイン名にしてください。

Sambaの起動と動作確認

それではSambaを起動してみましょう。従来と異なり、起動するのはsambaバイナリのみです。次のように単に起動してください。

```
# /usr/local/samba/sbin/samba
```

psコマンドで確認すると、**図9**のようにsambaプロセスとsmbd プロセスが起動していることを確認できるはずです。

引き続き、DNSの正常性を確認します。**図10**のようにSRVレコードをクエリして、適切な応答が返却されることを確認します。

さらにsmbclientを使ってSYSVOL共有にAdministratorとしてアクセスできることを確認します。**図11**に例を示します。

時刻同期とWindowsクライアントからのドメイン参加

Active Directoryが適切に機能する上では、クライアントとサーバの時刻が同期されている必要があります。これは手作業で行ってもかまいませんが、Active Directoryの標準ではNTPを用いた時刻同期が行われますので、可能であればSambaサーバをNTPサーバとしても動作させることがお勧めです。

CentOS 6、Squeezeともに、インターネットに接続できる環境であれば、ntpパッケージをイ

▼図8　/etc/resolv.confの設定例

```
domain samba400.samba.local
nameserver 192.168.135.141
```

▼図9　Sambaの起動確認

```
[root@centos63-2 ~]# ps ax | grep samba
 2275 ?        Ss     0:00 /usr/local/samba/sbin/samba
(中略)
 2290 ?        Ss     0:00 /usr/local/samba/sbin/smbd --option=server role
check:inhibit=yes --foreground
 2293 ?        S      0:00 /usr/local/samba/sbin/smbd --option=server role
check:inhibit=yes --foreground
```

▼図10　DNSの正常性確認

```
[root@centos63-2 ~]# nslookup -type=SRV _ldap._tcp.samba400.samba.local.
                                                  ↑ドメインのFQDN

Server:         192.168.135.141
Address:        192.168.135.141#53

_ldap._tcp.samba400.samba.local service = 0 100 389 centos63-2.samba400.samba.local.
                                                     ↑SambaサーバのFQDN
```

▼図11　smbclientによるSYSVOL共有のアクセス確認

```
[root@centos63-2 ~]# /usr/local/samba/bin/smbclient //centos63-2/sysvol -Uadministrator%P@
ssw0rd -c ls
Domain=[SAMBA400] OS=[Unix] Server=[Samba 4.0.0]
  .                                   D        0  Sun Dec 16 16:48:18 2012
  ..                                  D        0  Sun Dec 16 18:15:53 2012
  samba400.samba.local                D        0  Sun Dec 16 16:46:55 2012

                37308 blocks of size 524288. 33580 blocks available
```

ンストールすることで、インターネット上のNTPサーバと時刻同期するNTPサーバを構成できます[注8]。

ntpd 4.2.6以降では、Microsoft社独自のセキュアなNTP認証をサポートしています。Squeezeはntpd 4.2.6以降が同梱されていますので、**図12**のような設定を行うことで、この認証が有効になります。

ntp_signdディレクトリは、グループをntpd実行ユーザの所属グループにしたうえで、パーミッションを750に設定してください。

ここまで設定できたら、Windowsクライアントを実際にドメインに参加させてみましょう。Windowsクライアントからの操作はとくにSambaドメイン固有の事項はありません。通常どおりに参加させます。

注8) CentOS 6では明示的なサービスの起動(もしくはサーバを再起動)が必要です。

Samba 4ドメインの管理

ドメインの管理については、基本的にWindowsクライアント、Sambaサーバ双方で行うことができますが、一部の設定については片側でしか設定できないものが存在します。以下簡単に紹介します。

Windowsクライアントからの管理

RSAT[注9]などの管理ツールをインストールすることで[注10]、Windowsクライアントからドメインを管理することができます。

Active Directoryユーザとコンピュータ、サイトとサービス、ドメインと信頼関係といった各種管理ツールから、ユーザ、グループ、OUの追加や管理、グループポリシーの管理など、

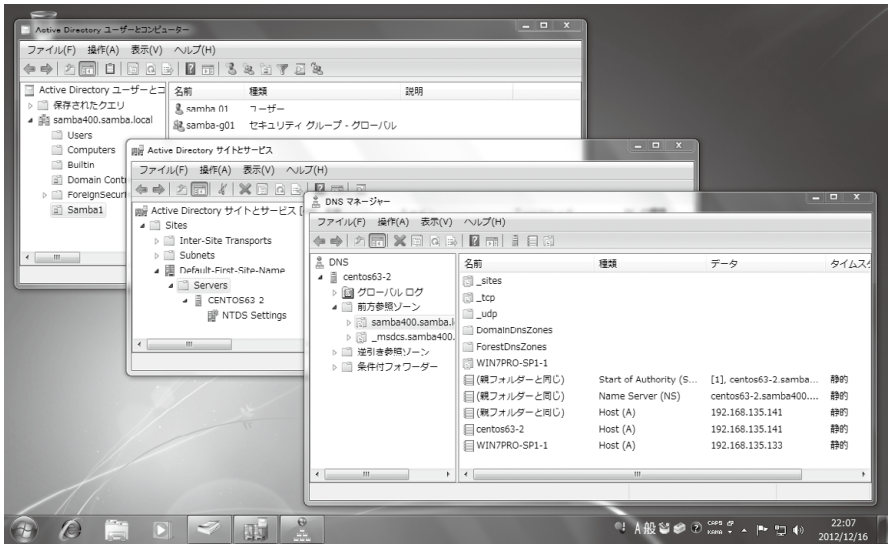
注9) Windows 7 SP1版は以下から入手できます : http://www.microsoft.com/ja-jp/download/details.aspx?id=7887

注10) サーバ版のWindowsをクライアントとして参加させた場合は、サーバに付属の管理ツールで管理ができます。管理ツールの名称はWindowsのバージョンによって多少異なります。なお管理ツール以外にスクリプトなどを用いて管理することもできます。

▼図12　NTP認証

```
ntpsigndsocket /usr/local/samba/var/lib/ntp_signd/
restrict default mssntp
```

▼図13　Windows 7クライアント上の管理ツールからSambaで構築したActive Directoryを参照する



単一ドメインに閉じた機能はほとんどが動作します(**図13**)。

Sambaサーバ上からの管理

Sambaサーバ上から管理を行う場合はSamba 4で新たに追加されたsamba-toolコマンドで行います。samba-toolコマンドには従来のnetコマンドと同様に、多数のサブコマンドがあり、サブコマンドごとに多数のオプションがあります。主要なサブコマンドを**表4**に示します。サブコマンドごとのオプションについては、各サブコマンドに対して--helpオプションを指定することで確認できます。

検証目的のドメイン構築だととりあえず無効にしておきたいパスワード複雑性を含むアカウントポリシーですが、(なぜか)Default Domain Policyを変更しても反映されないので注意してください。**図14**のようにsamba-toolコマンドで設定を変更する必要があります。

DNSサーバの管理

ゾーンの作成やゾーン内のリソースレコードの管理などは、Windows側の管理ツール、samba-tool dnsコマンドいずれでも可能です。

DNSサーバ自体の各種プロパティの設定については、今のところ設定を変更する方法はなさそうです。

Sambaユーザ、グループの管理

従来のSambaでは、Samba上のユーザやグループに対して必ず対応するUNIXユーザやグループを作成する必要がありましたが、Samba 4のsambaバイナリでは、単にドメインコントローラとして使用する限り、この処理が基本的に不要となりました。

何らかの用途で対応するUNIXユーザを作成したい場合は、Winbind機構の設定を行い(**図15**)、/etc/nsswitch.confにwinbindというキーワードを追加することで(**図16**)、内蔵されているWinbind機構により作成されたUNIXユーザ

▼図14　アカウントポリシーの設定を変更する

```
# samba-tool domain passwordsettings --complexity off
```

▼図15　Winbind機構の設定

```
# ln -s /usr/local/samba/lib/libnss_winbind.so /lib/winbind.so
# ln -s /usr/local/samba/lib/libnss_winbind.so.2 /lib/winbind.so.2
```

▼表4　samba-toolコマンドの主なサブコマンド

サブコマンド	説明	Windows管理ツールからの設定可否
delegation	委任の管理	可能
dns	DNSサーバのゾーン管理	可能
domain demote	ドメインコントローラからの降格	不可
domain level	ドメイン、フォレスト機能レベルの設定	可能
domain passwordsettings	アカウントポリシーの設定	不可
drs	ディレクトリ複製の管理	可能
dsacl	ディレクトリのアクセス許可の管理	可能
fsmo	FSMOの管理、設定	
gpo	グループポリシーの管理	可能※
group	グループの管理	可能※
rodc	RODCの管理	可能
sites	サイトの管理	可能※
user	ユーザの管理	可能

が表示されるようになります(図17)。

ユーザのシェルやホームディレクトリは、各々winbind template shellおよびwinbind template homedirパラメータで一律設定することはできますが、ユーザごとに個別に設定することはできません。またUIDやGIDについても標準では変更できないようです。

Samba 4の情報源

Samba 4のsambaでは、これまでのパラメータ主体の静的な設定ではなく、samba-toolコマンドによる動的な設定の比重が高くなっています。さらに、Windowsの管理ツールから多くの設定が行えるようになったため、設定内容の一覧化が困難になってきています。

Samba 4にはsamba-toolコマンドのマニュアルページも付属していますが、コマンドラインのヘルプと同等のざっくりした説明しか記載されていないことなどで十分なドキュメントとはなり得ていないのが現実です。

現在もっともまとまったドキュメントは次にあるSamba WikiのSamba4 AC DC HOWTOおよびそこからリンクされている各種Webページになりますが、基本的に手順書ベースの簡単なもので、網羅的な情報とは言えない状態です。

・Samba4 AC DC HOWTO
https://wiki.samba.org/index.php/Samba4/HOWTO

Sambaのメーリングリストを見ていると、これ以外にもさまざまな設定が散見されます。ドキュメントの充実が望まれるところです。

おわりに

今回はSamba 4のsambaバイナリにより実現したActive Directoryのドメインコントローラ構築について駆け足で説明を行いました。Sambaを用いることで、低負荷な環境であれば256MB程度のメモリのマシンでもドメインコントローラ機能を提供できます。筆者はSamba自身のほか、Exchange ServerやSharePoint Serverの検証のために多くのActive Directoryを仮想環境上に構築していますが、ドメインコントローラをSambaで実装することで使用するリソースの大幅な削減が図れました。

ドキュメントの不足を初め、まだまだ荒削りなところもあるSamba 4ですが、2011年末頃から徐々に実環境での使用が始まりつつあることを考えると、一定の品質は確保できているとも言えそうです。

インストール自体は単純化されていますので、まずはSamba 4を用いてActive Directoryをお手軽に体験してみるのはいかがでしょうか。SD

▼図16　/etc/nsswitch.confへの修正

```
passwd:     files winbind
shadow:     files
group:      files winbind
```

▼図17　Sambaユーザに対応付けられたUNIXユーザの表示

```
[root@centos63-2 lib]# getent passwd
root:x:0:0:root:/root:/bin/bash
(中略)
ntp:x:38:38::/etc/ntp:/sbin/nologin
SAMBA400¥Administrator:*:0:100::/home/SAMBA400/Administrator:/bin/false
SAMBA400¥Guest:*:3000011:3000012::/home/SAMBA400/Guest:/bin/false
SAMBA400¥krbtgt:*:3000020:100::/home/SAMBA400/krbtgt:/bin/false
SAMBA400¥samba01:*:3000021:100:samba 01:/home/SAMBA400/samba01:/bin/false
```

Express5800シリーズ
FreeBSD正式認定の狙い

NECのPCサーバ「Express5800シリーズ」は多様な顧客ニーズに応えるためWindows Server OSのみならず、オープンソースOSへの対応を拡充しています。これまで正式にサポートの対象となっていたのはRed Hat Enterprise Linux（RHEL）とSUSE Linux Enterprise Server（SLES）の2つで、そのほかのOSについては動作確認の情報のみが公開されるといった状況でした。このたびBSDコンサルティングとの協業により、これらのOSに加えてFreeBSDが正式に認定されることをうけ、NECでExpress5800シリーズを担当されている久保淳（くぼ あつし）氏とBSDコンサルティングの後藤大地（ごとう だいち）氏に、これまでの経緯と両社のこれからの展開についてお話をうかがいました。

●聞き手：編集部

NEC Express5800シリーズ、BSDコンサルティングがサポート

――NECがBSDコンサルティングのFreeBSD認定を取得したとのことですが、これからどのような協業を行っていくのか教えてください。

❖

BSDコンサルティング株式会社 取締役 後藤大地（以降、後藤）●はい。Express5800シリーズの対象モデルでFreeBSDが動作するかどうか検証し、問題があれば協議して対応する、そうした動作検証サービスに取り組んでいきます。現在のところ、Express5800/R120d-1M、Express5800/R120d-2M、Express5800/R110e-1Eというモデルを対象に、FreeBSDの動作検証に取り組んでいます。要望に応じて順次検証対象のモデルを広げる予定です。

FreeBSDが対象のハードウェアをどのように認識しているか、ちゃんと動作しているか、負荷に耐えられるかどうかなどを検証します。問題があればNEC様と協力して対応していきます。サポート情報や不具合に対する対策方法などをBSDコンサルティングのサイト[注1]に随時掲載していく予定です。

❖

――サポートするFreeBSDのバージョンはいくつになりますか。最新版だけですか？

❖

後藤●FreeBSDプロジェクトがサポートしている安定版バージョンのうち、直近の2ブランチのリリースバージョンについて動作検証サービスを提供する予定です。たとえば現在ですと、9.1-RELEASE、9.0-RELEASE、8.3-RELEASE、8.2-RELEASE、8.1-RELEASEが対象になります。

❖

――FreeBSDだと、まだ7系を採用しているところも多いようですが、7系はサポートしないのでしょうか。6系以前を採用しているところも少なくないようです。

❖

注1）URL https://www.bsdconsulting.co.jp/

後藤●今のところ、要望があれば別案件として対応することになるだろうと考えています。バージョンがあまり離れすぎると、労力の割にコストが見合わなくなってくるので、ケース・バイ・ケースで判断するしかないという現実があります。対応できることとできないことがあるので、そのあたりの判断も必要になります。

サポート対象外のバージョンの扱いに関しては、クライアント様と直接交渉するか、NEC様とのスキームの中で行うのかは、やはりケース・バイ・ケースでの判断になると思います。古いバージョンを使い続けていると、新しいハードウェアで利用できない、または性能が発揮できないといった問題も出てきますので、できればサポートサービスではなく、アップグレードコンサルティングのほうで最新バージョンへ移行することをご検討いただきたいと考えています。

FreeBSD認定に至った経緯

――NECはこれまでもオープンソースのオペレーティングシステム（OS）に関しては、動作確認情報を報告していたと思いますが、今回BSDコンサルティングから、本格的にFreeBSD認定を取得することに至ったのはなぜでしょうか。

❖

NEC ITハードウェア事業本部・システム製品技術グループ 技術マネージャ 久保淳（以降、久保）●クライアント様に幅広いOSでご利用いただくために、過去に問い合わせいただくなどのニーズがあったオープンソースのOSを選択して動作確認情報を掲載しています。FreeBSDの動作確認情報の公開は2007年5月から開始しています。当時はFreeBSD 4.1-RELEASEでした。2012年12月現在で、過去のモデルを含め累計20モデルで動作確認情報を公開しています。

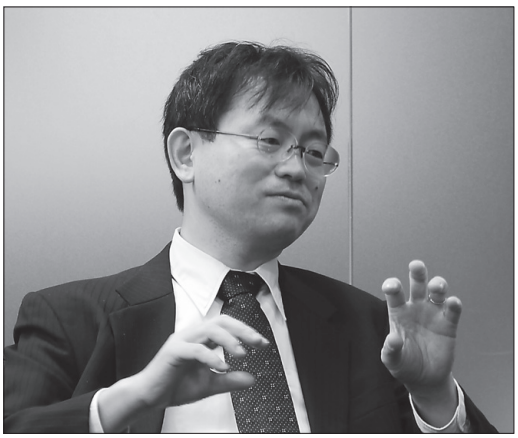
実際にExpress5800シリーズで、どのOSをお使いになるかはクライアント様が選択できます。そのため、OSをNECから購入いただかない場合や、そのOSの動作に関するお問い合わせがない場合には、どのクライアント様でどのOSが動作しているかをNECではすべては把握できません。

しかしながら、今回はExpress5800シリーズの新しいモデルの出荷をはじめたところ、FreeBSDが動作検証情報に関する問い合わせを多くのクライアント様からいただき、それで「実はFreeBSDを使われているお客様が多い」ということが改めてわかったという次第です。

ハードウェアコントローラベンダーがドライバ開発をしていないコミュニティベースのオープンソースOSについては、NECとして動作確認はできますが、問題が発生した場合――たとえばネットワークが動作しないとか――にカーネルやデバイスドライバの開発を含めてNECをはじめハードウェアベンダーで対応するのは困難な場合が多い状況です。そういったケースではより深い技術を持ったサポート企業が必要でした。

そして今般、FreeBSDに関して、今のままでは対応が難しくなっていくと判断し、BSDコンサルティング様の認定を取得させていただきました。今回の対象モデルに関してはBSDコンサルティング様から対処方法やデバイスドライバをご提供いただきますが、今後は製品の出荷段階ではFreeBSDで動作するようになっている、というスキームを構築していきたいと考えています。

■写真　後藤 大地氏
BSDコンサルティング株式会社 取締役



■写真　久保 淳氏
日本電気株式会社 ITハードウェア事業本部・システム製品技術グループ 技術マネージャ

❖

――なるほど。そういった背景があったのですね。具体的に、どのような問題が発生したのかお聞かせ願えますか。

❖

久保●Express5800シリーズの最新モデルでは、アレイボードはLSI社の最新のMegaRAIDを採用しています。このアレイボードのFreeBSD向けドライバはLSI社から提供されました。しかしながら、導入にあたっては多少の操作が必要で、FreeBSDの古いバージョンではそのままでは起動ができませんでした。そのため、起動時にカーネル内部のデバイスドライバを無効にするなどの操作が必要です。

また、オンボードのネットワークは従来のIntelチップからBroadcomのBCM571X系のチップへ変更しました。FreeBSDのソースにはチップ名の記述がありますが、実はコントローラを認識する程度で、充分に動作しないということが、今回の活動の中で判明しました。こうした状況に対応するためにもBSDコンサルティング様との協力体制が必要だと実感しました。現在では、いくつかのモデルで検証作業を開始してもらっていますし、今後さらに対象モデルを拡充し、NECのサイトに情報を掲載していく予定です。

❖

――検証作業はどの程度進んでいるのでしょうか。

❖

後藤●サポートするバージョンすべてについて一気に対応方法を提示するのは時間的に難しいので、最新版のリリースバージョンから順次結果を報告していく予定です。要望があるクライアント様には個別にも対応しています。一般情報は先ほどのBSDコンサルティングのサイトに掲載していきますので、よろしくお願いします。

意外（？）と使われているFreeBSD

――FreeBSDを使われている企業というのは、結構多いものなのでしょうか。弊社でも内部システムはFreeBSDベースで構築していますが。

❖

久保●そうですね、いくつかのインターネット事業者様でFreeBSDを採用されていることは、業界的にもよく知られていると思いますが、今回の件で、多くの企業様が採用していることが改めてわかりました。

❖

後藤●最近だと、パナソニックさんがスマートビエラのOSとしてFreeBSDを採用したニュースが目新しいところでしょうか。高性能アプライアンス、組み込み、家電、ゲーム機、エッジサーバ、ISP/ホスティング、社内システムなどいろんなシーンでFreeBSDが使われているのですが、表には出てきませんね。

❖

――弊社の読者の方々にもFreeBSDは需要が高いですし、実際には多くのシーンで使われているんでしょうね。

❖

後藤●FreeBSDを大量に導入する企業様では、あらかじめFreeBSDが動作するハードウェアに的を絞ってハードを調達する傾向があります。どのハードで動作するか検証する手間を省きたいからです。FreeBSDを積極的にサポートしてくれるベンダー

もありますので、そうしたベンダーのハードウェアの組み合わせに絞ってマシンを調達するといったスタイルです。

これからはそうしたスタイルに加えて、NEC Express5800シリーズも検討対象に追加できるようになると思います。Express5800シリーズの動作検証対応モデルに関してはあらかじめFreeBSDの動作検証が実施されるようになりますから、購入時の検討対象として利用しやすくなるのではないかと思います。要望があればより多くのモデルに対して動作検証サービスを提供できますので、ハードウェアベンダーとしてもクライアントとしても利益のある話ではないかと思います。

対応してほしいモデルがあれば、NEC様のほうにプッシュしていただきたいです。BSDコンサルティングとしては、FreeBSD対応を謳ったハードウェアが市場に増えることを喜ばしいことだと考えていますので、今後もより多くの製品の動作検証に取り組みたいと考えています。そのためには、まずクライアント様からのお声がけが必要ですね。SD

Column PCサーバExpress5800シリーズ

NECはデータセンターやIDC市場、企業内データセンター、オフィスでの利用、現場に設置しての稼働など、さまざまなビジネスシーン向けに多種多様なPCサーバ「Express5800シリーズ」を提供しています。スケーラブルHAサーバ、SIGMABLADE、ftサーバ、アプライアンスサーバInterSec、スタンダードラックサーバ、ECO CENTER、スリムサーバ／水冷式スリムサーバ、スタンダードタワー、Gモデルなど、国内向け出荷に強い実績があります。

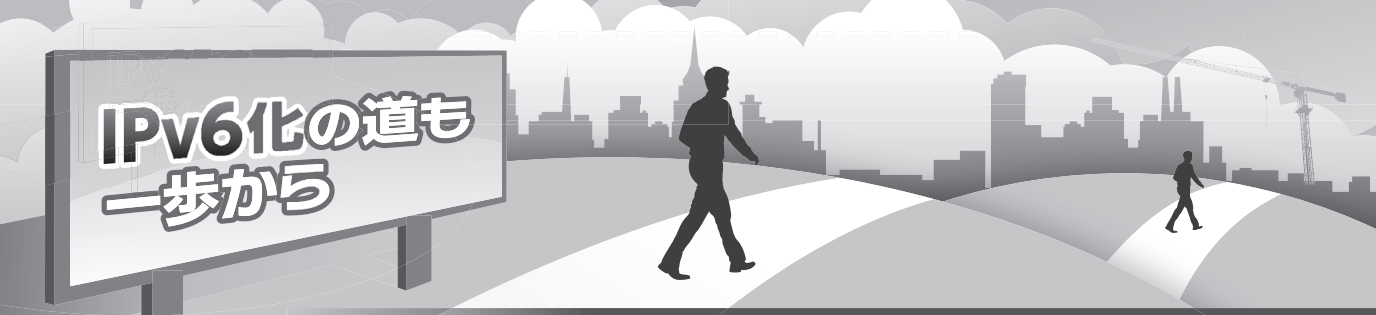
BSDコンサルティングがサポートを開始したのはこの中のスタンダードラックとECO CENTERです。これらラックサーバは、ほかの競合プロダクトと比較して省電力性が高いこと、40度での稼働をサポートしていること、FreeBSDをはじめOSS動作確認を提供しているといった特徴があります。

省電力は国内市場の要望として優先度の高いものになっています。大震災以降、国内のベンダーは省電力のラックサーバに対して強い興味を示しています。国内企業向けの出荷業績が強いNECでは、こうした要望を汲み取って高い省電力性を実現したラックマウントサーバの提供に注力しています。

フロント側からノードの挿抜が可能になっているなど保守性を考慮したハードウェア設計、優れた運用管理を実現するEXPRESSSCOPEエンジン3の搭載などもポイントです。BMCのソフトウェアリセットのみならずハードウェアリセットに対応しているところなども、細かいところですが気の利いた設計になっています。

Express5800シリーズポータルサイト
URL http://www.nec.co.jp/exp/

■写真　FreeBSD動作検証の対象となっているExpress5800/R120d-1M（左）、Express5800/R120d-2M（中央）、Express5800/R110e-1E（右）



第3回 押さえておきたいIPv6とIPv4の10個の違い

IPv6普及・高度化推進協議会　IPv4/IPv6共存WG　アプリケーションのIPv6対応検討SWG
廣海 緑里 HIROMI Ruri　**渡辺 露文** WATANABE Tsuyufumi　**新 善文** ATARASHI Yoshifumi　**藤崎 智宏** FUJISAKI Tomohiro

ちょっと前までのIPv6プロトコルの説明

今回は、IPv6プロトコルとアドレス管理について取り上げます。

多くのIPv6の教科書では、**表1**のようなIPv6の特徴が挙げられています。しかし、これは1990年代のIPv6標準化開始当初のもので、その後IPv4への機能追加により、IPv4でも普通に利用できるようになっているものも少なくありません。結果として、表1にあるような特徴でのIPv4との違いは、「アドレス空間がとても広い」という1点に大きな違いがあるということになってしまいます。

そうは言っても、実際には両者にはプロトコル互換性がなく、IPv6とIPv4を相互接続させるには特別な処理が必要になるなど、運用するには押さえておかなければいけない違いがいくつかあります。

今押さえておきたいIPv6プロトコルの中身

ここでは「IPv6の基本」として、そのプロトコルについてIPv4との比較から考えてみます。違いを10個に絞って説明します(**表2**)。

①プロトコルの互換性がない

IPv4パケットのヘッダの宛先や送信元としてIPv6アドレスを指定することはできません。逆に、IPv6パケットのヘッダの宛先や送信元にIPv4アドレスを指定することもできません。IPv4とIPv6で相互接続性を確保するためには、各種移行／共存技術(トランスレータ、アドレスマッピング技術、トンネル接続など)を利用します。

②アドレスの長さが変わった

図1はIPv4アドレスとIPv6アドレスのアドレス設計の違いを示した図ですが、全長が32ビット(IPv4)から128ビット(IPv6)に長くなっています。IPv4ではホストに使えるアドレスを

▼表1　IPv6の主な特徴

特徴	説明
アドレス空間の拡張	32ビット(約34億個)から128ビット(約340澗個)へ拡張
固定長ヘッダ、階層化アドレス構造	ヘッダを固定長とすることで，ルータなどへの処理負荷を軽減
プラグ＆プレイ	アドレス自動設定機構の標準装備
IPsecの標準搭載	暗号化技術を標準搭載
マルチキャストの標準搭載	マルチキャスト技術を標準搭載
移動体通信への対応	Mobile IP による固定網と移動網のシームレス化

確保するために、1つのセグメントの収容端末数からマスク長を決めてセグメント分割するという、アドレス数をものすごく節約する運用が行われてきました。

(例)32ビットのうち、/29でサブネット分割して、端末6台のアドレスを確保する

一方、IPv6では真ん中でネットワークと端末を区切るのが一般的で、ルータとルータの接続用などでたとえ2台しか機材がない場合でも/64のネットワークが利用できます。

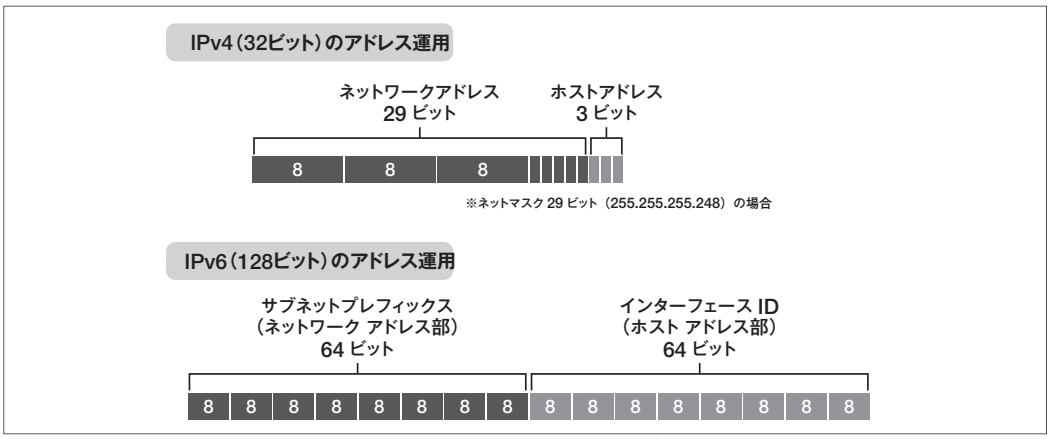
(例)128ビットのうち/64で分割し、64ビット分全部端末に払いだす

このように、すべてのサブネットを/64で分割した運用が可能になります。たとえば、ISPから/48を割り振られた場合、16ビット分のサブネット数(65,536)があり、たいていの企業で

▼表2　IPv4のみの運用から変わる点

	IPv4との違い	どう変わったか
①	プロトコルの互換性がない	IPv4パケットのヘッダの宛先や送信元としてIPv6アドレスを指定することはできない
②	アドレスの長さが変わった	32ビットから128ビットになった。ネットワーク部とホスト部の長さも変わった
③	アドレスの表現形式が変わった	10進数とピリオド(.)という表記から、16進数とコロン(:)になった
④	HTTPのリクエストなどで使うURLの書き方が変わった	FQDNを用いる場合は同じ。IPアドレスを直接指定する場合は[]で囲む。(HTTP/HTTPSなどで使うURIの規定)ソフトウェアプログラム内での参照では、プログラミング言語によって作法が違うので必ず確認する(" "で囲む場合など)。ローカルアドレスではインターフェースIDも付く
⑤	端末に複数のアドレスが付く	IPv4とIPv6、IPv6のローカルアドレスとマルチキャストアドレスとグローバルユニキャストアドレス……のようにたくさんのアドレスを利用する
⑥	端末へのアドレス設定方法が変わった	LAN内でDHCPによる自動設定が一般的だったが、prefix情報とイーサネットアドレス由来の端末アドレスを組み合わせて使う機構を利用する
⑦	IPアドレスを直接指定する場合、IPv4とIPv6が共存している環境では処理順序決めが必要	通信相手にパケットを送る場合、送信元アドレスをIPv6にするのか、IPv4にするのか、IPv6はどのアドレスを使うかなど、処理順序の取り決めが必要
⑧	IPv6処理のために開発言語の拡張がされている	アドレス構造が変わったためIPv4依存の処理関数などは利用できない。IPv4とIPv6共存のための処理関数が増えている
⑨	DNSにIPv6専用のレコードがある	DNSでホスト名からIPv6アドレスを取得するには、AAAAレコードを参照する
⑩	通信のしくみ、通信処理が変わった	Path MTU Discoveryというしくみで、End to Endで転送するパケットサイズの調整が行われる。IPv4のブロードキャストのarpから、マルチキャストを使うndpが導入された

▼図1　IPv4とIPv6のアドレス運用



は十分なスペースを確保できます。サブネットマスクの算出に苦労していた管理者にとって、IPv6は管理が楽になり、管理に使っていた時間を別のことに振り分けられるようになります。

③アドレスの表現形式が変わった

IPv4アドレスでは、32ビットを8ビットずつ区切って10進数で表記し、区切り文字には「.」(ドット)が使われていました。

(例)192.0.2.1

IPv6アドレスでは、128ビットを16ビットずつ区切って16進数で表記し、区切り文字には「:」(コロン)を使います。また、IPv6ならではの表現形式として、区切った16進数の先頭のゼロは省略可能だったり、ゼロとコロンが連続する場合は省略を表す「::」を使う短縮型が認められています。

(例)2001:db8::1

図2に示したアドレスは、いずれも同じものを指します。柔軟性があるのは良い面がある一方で、運用上では混乱をきたす原因ともなります。そのため、表記方法を統一するための推奨を記載したRFC文書[注1]があります。ログ解析などを前提とする場合には、省略なしで記載するようにしたほうがいいという意見もあるようです。

④HTTPのリクエストなどで使うURLの書き方が変わった

IPv6、IPv4ともにFQDN(Fully Qualified Domain Name)[注2]を利用する場合、IPアドレスはDNSが処理してくれるため変更ありません。

IPv6アドレスを直接入力する場合には、次のように**[]**で囲みます。

```
http://[2001:db8::1]/index.html
http://[2001:db8::1]:8080/index.html
```

また、HTTP/HTTPS以外のプロトコルでは、**[]**ではない囲み方(**"2001:db8::1"**など)をする場合もあるので事前に調べたほうが良いでしょう。

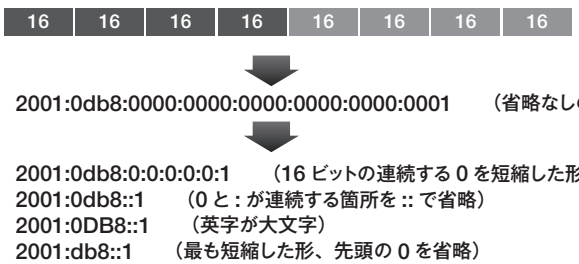
実はこういった情報はあまりインターネット上に出ていないようですが、それは、基本的にはアプリケーションの通信先の指定にはFQDNを使い、IPアドレスを直書きするのは「百害あって一利なし」である、という考え方が浸透していることの表れと言えそうです。実際に直書きしてしまうと、利用に制限が出たり、途中で運用形態を変えられなくなるなど良いことがありません。

ただし、これはDNSの利用を前提としている場合のことであって、組込み系プログラムの一部など、使えるリソースに制限がある場合は、この限りではないことがあります。

注1) RFC5952 http://tools.ietf.org/rfc/rfc5952

注2) ホスト名などを省略せずにすべて指定したドメイン名。

▼図2 IPv6のアドレス表記



⑤端末に複数のアドレスがつく

PCなどの端末に付与されるアドレスの見え方も変わります。従来のIPv4アドレスに、IPv6アドレスが追加になるだけではなく、IPv6プロトコル群で利用される特別な目的を持ったアドレスも追加設定され、端末には数多くのIPv6アドレスが付くようになります。

IPv6のプロトコル群で利用されるアドレスの代表的なところでは、リンクローカルアドレスと呼ばれる同一のリンク上のローカルな通信で利用されるもの、マルチキャストアドレスと呼ばれるアドレス自動設定や非到達性確認(Neighbor Unreachability Discovery)などに使う特別な役割を持ったアドレス、プライバシー保護のためにホスト部にランダムな値を利用するアドレスなどです(**表3**)。**図3**はWindows 7でアドレスがたくさん付いている例です。

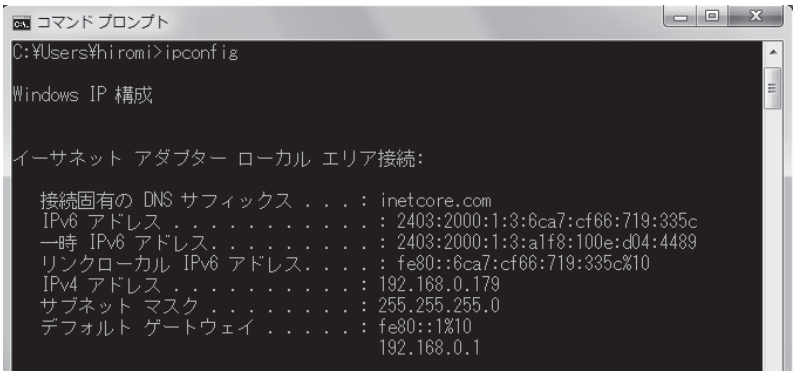
⑥端末へのアドレス設定方法が変わった

IPv4のLANに端末を接続する場合には、DHCPによるアドレス自動設定が一般的でした。IPv6のアドレス設定では、ホスト部とネットワーク部それぞれの設定プロトコルがあり、さらにDNSサーバやデフォルトルータなどその他の必要な情報の伝達に使うプロトコルもあります。それらを組み合わせて端末の接続情報を準備します。なお、IPv4でのホスト部とネットワーク部に相当する用語も変更されているので確認しておきましょう(**図4**)。

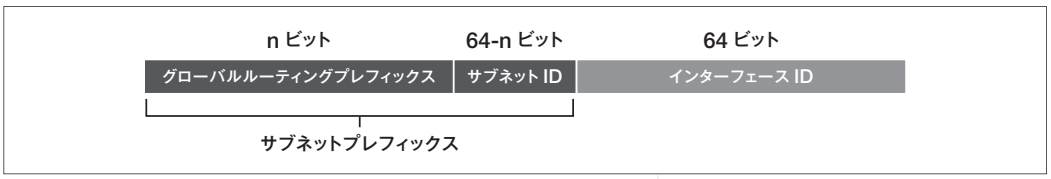
▼表3　ノード／ルータに付与されるIPv6アドレスの例

ノードが使うIPv6アドレス	ルータが使うIPv6アドレス
・ループバックアドレス(::1/128) ・全ノードマルチキャストアドレス(ff0x::1) ・要請ノードマルチキャストアドレス(ff02::1:ff00:0/104) ・インターフェース毎に1つのリンクローカルアドレス(fe80::/10) ・インターフェース毎に1つまたは複数のユニキャストアドレス ・自分が所属するグループのマルチキャストアドレス	・全ルータマルチキャストアドレス(ff0x::2) ・サブネットルータエニーキャストアドレス(サブネットプレフィックス以外All 0)

▼図3　たくさんのアドレスがついている例(Windows 7)



▼図4　IPv6アドレスの構成要素の名称



IPv6のアドレス自動設定には、SLAAC(ステートレスアドレス自動設定)と呼ばれるNDP(近隣探索プロトコル)のRA(ルータ広告)と自ホストで生成するインターフェースIDを組み合わせて設定するものや、IPv6版のDHCP(DHCPv6)を使うものなどいくつかの方法があります(**表4**)。

また、実際に通信を行うためには、デフォルトルータやDNSサーバなどの情報も必要になりますが、そうした情報の伝達方法も運用方針や機材の実装状況に従って技術を選択して利用する形になります。とくにDHCPv6クライアントは、端末ごとに実装状況が違っており、うまく運用できないことがあるため事前の確認が必要です。

⑦IPv4とIPv6が共存している環境では処理順序決めが必要

インターネットの標準団体IETF(The Internet Engineering Task Force)が発行している文書では、IPv4とIPv6が共存している環境での処理順序はIPv6を優先することが推奨されています。しかし、IPv6接続が普及していない段階では、IPv6→IPv4という処理順序だと遅延が起きることが懸念されています。そのため、実際にはIPv4が優先される端末実装やネットワーク環境、アプリケーションもあります。開発言語によって、処理の優先順を設定するもの(Javaのシステムプリファレンスなど)もあります。

端末が通信相手にパケットを送る場合、送信先と送信元アドレスをIPv6、IPv4どちらにするのか、IPv6はどのアドレスを使うのかなど、処理順序の取り決めがRFC6724[注3]という文書になっていますが、実装されていない端末OSや優先を変えたい場合のユーザインターフェースがない実装などもあり統一されていない状況です。

⑧IPv6処理のために開発言語の拡張がされている

IPv6では、名前解決の参照に利用されるレコードやアドレス処理方法が変わったため、これまで使われてきたIPv4依存の処理関数などは利用できません。IPv6対応済みの開発言語では、IPv4とIPv6共存のための処理関数が追加されており、対応版を使っていくことが推奨されます。

ただし、開発言語ごとにIPv6の対応方針は異なっており、デュアルスタック用の関数が用意されるケース(C言語のgetaddrinfoなど)やクラス関数が拡張されるケース(Perl言語のNet::DNSなど)があります。それまで実行していたコードを大幅に変更することなくデュアルスタック環境で利用できるように、IPv4とIPv6の取り次ぎをするようなクラス関数が定義されているものもあります(Perl言語のNet::INET6Glueなど)。

注3) RFC6724　http://tools.ietf.org/rfc/rfc6724

▼表4　IPv4/IPv6アドレスの自動設定方法の違い

	IPv6			IPv4
	RA	DHCPv6 (stateful)	DHCPv6-lite (stateless)	DHCPv4
IPアドレス	○ プレフィックス情報を通知	○	—	○ /32を通知
デフォルトルータの伝達	○	—※1	—	○
サーバアドレスの配布(ネームサーバやSIPサーバなど)	△※2	○	○	○

※1　経路情報の配布について標準化進行中
※2　DNSサーバアドレスの配布は[RFC6106]で標準化済み

⑨DNSにIPv6専用のレコードがある

DNSによる名前解決で利用するIPv6専用のリソースレコードが作られています。DNSでホスト名からIPv6アドレスを取得するには、IPv6専用のAAAAレコードを参照します(**図5**)。

逆引き(PTRレコード)も、IPv4とは記載方法が異なります。アドレスを4ビットずつ16進数で表記し、"."(ドット)で区切ります。また、IPv6では端末アドレスは自動生成されることが多い上に、一時アドレスなどもあるため登録管理が難しいといった理由から運用されていないケースが多く、逆引きでの解決を前提とした処理は通信不通などの問題を起こすかもしれません。IPv4で電子メールの送信やSSLのアクセスで使われる逆引きを認証として使う方法が普及していますが、IPv4での逆引き認証に代わる別の方法を考案し、運用していくことが求められそうです。

(例)2001:db8::1の逆引き表記
1.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.
0.8.b.d.0.1.0.0.2.in6.arpa

⑩通信のしくみ、通信処理が変わった

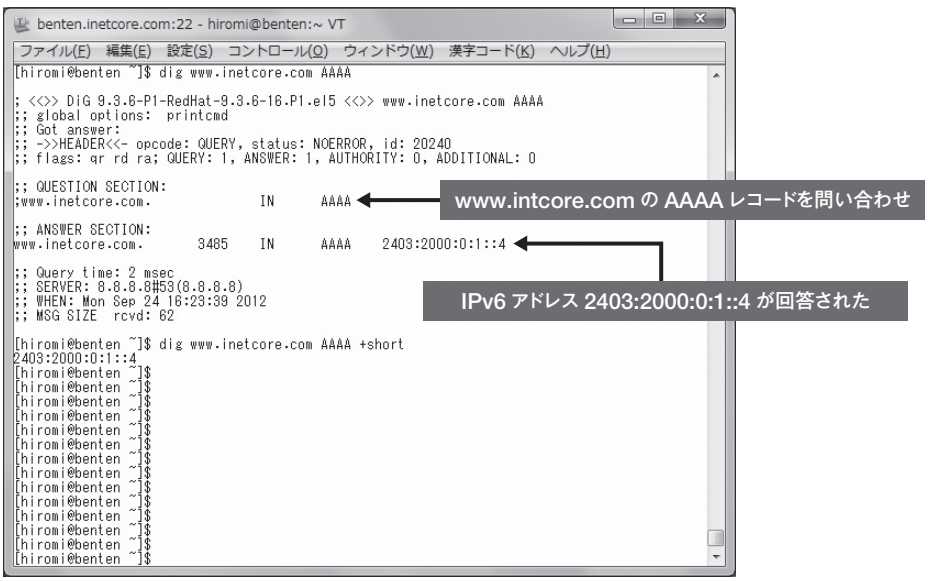
IPv6では、通信時にPath MTU Discoveryというしくみにより、End to Endで転送するパケットサイズの調整が行われます。IPv4のように中継ノードでのパケットフラグメント処理は行われません。Path MTU DiscoveryにはICMPv6が利用されます。そのため、Destination UnreachableやPacket Too Bigといった検査に使われるICMPv6をフィルタしてしまうと、通信障害が発生してしまう可能性があります。そのため、フィルタすることは推奨されません。

◆ ◆ ◆

今回は、IPv4から変わったところを10個取り上げてみました。この他にもリンクローカルアドレスが通信に使えるなどの相違点があります。SD

連載を通じての質問やコメント、取り上げてほしいトピックを募集します。最終回などで取り上げて、追加解説したいと考えています。質問、コメントは編集部(sd@gihyo.co.jp)までお送りください。

▼図5　AAAAによる名前解決の例



ハイパーバイザの作り方

ちゃんと理解する仮想化技術

第5回 I/O仮想化「割り込み編・その2」

浅田 拓也（ASADA Takuya）Twitter @syuu1228

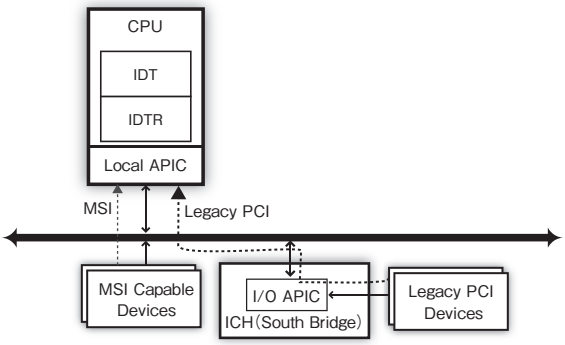
はじめに

前回の記事では、割り込み仮想化の話の前提となるx86アーキテクチャにおける割り込みのしくみを各コンポーネントごとに解説しました（**図1**）。仮想環境でどのように割り込みを実現するのか、前回解説した各機能ごとに見ていきます。

仮想化における内部割り込みと外部割り込み

VT-x環境においては、内部割り込みはCPUがすべて処理を行うため、基本的にハイパーバイザが介入する必要がありません。一方、外部割り込みについては、ハイパーバイザの介入が必要となります。それぞれを見ていきましょう。

▼図1　割り込みにかかわるコンポーネントと仮想化範囲



CPUへの割り込みの挿入

仮想CPUで任意の割り込みを発生させるには、VMCSのVM-Entry Control FieldsにあるVM-entry interruption-information fieldにベクタ番号と割り込みタイプを書き込みます（**表1**）。ただし、このフィールドに値をセットして外部割り込みを発生させるだけでは、Local APICのレジスタ値は適切に更新されず、ハイパーバイザが新しい値を計算し、セットする必要があります。

内部割り込みの仮想化

実機での内部割り込みは、次の手順で処理されます。

①ゲストマシン上のソフトウェアが例外を発生させるか、INT命令の実行によりCPUで内部割り込みが発生
②CPUはIDT上のゲートデスクリプタを読み込み、割り込みハンドラを実行
③割り込みハンドラが内部割り込みを処理
④IRET命令で直前のコンテキストへ復帰

▼表1　VM-entry interruption-information field

ビットポジション	内容
7:0	ベクタ番号
10:8	割り込みタイプ：通常は0（外部割り込み）を使用
11	スタックに例外のerror codeをpush
31	有効化ビット

これらはすべて、ハイパーバイザの介入が不要です②～④については、**IDT/IDTR の仮想化**にて説明します。ただし、ここでもVMCSの設定により、内部割り込みを契機としてVMExitを発生させることができます[注1]。ただし、この利用方法は一般的ではありません。

外部割り込みの仮想化

内部割り込みはソフトウェアを起因としCPU内部で発生するため、VT-xによりハイパーバイザの介入なしに仮想化することが可能でした。一方、外部割り込みは、ハイパーバイザが割り込みを送り込みます。これは、デバイスがソフトウェア的に、ハイパーバイザ内に実装されているためです。

I/O APICを通して割り込む場合

実機でのI/O APICを通して割り込む場合の外部割り込みは、次のような手順で処理されます。

①デバイスが割り込みラインからI/O APICへ割り込みを送信
②I/O APICが割り込みを受け取り、Redirection Table Entryに指定されたDestination IDが示すLocal APICへ割り込みを転送
③Local APICがCPUへ割り込み
④CPUはIDT上のゲートデスクリプタをロードし、割り込みハンドラを実行
⑤割り込みハンドラが外部割り込みを処理
⑥割り込みハンドラがLocal APICへEOIを書き込み、割り込み処理の終了を伝達
⑦IRET命令で直前のコンテキストへ復帰

このうち、④、⑤、⑦については内部割り込みと同様の処理であり、ハイパーバイザの介入は必要ありません。①～③は次のように仮想化されます。まずあらかじめ、ゲストOSが割り込みを初期化するときにI/O APICのRedirection Table Entryへ宛先Local APICが設定されます。実際にデバイスが使われ始め、ハイパーバイザがデバイスのエミュレーションを行うと、割り込みを仮想CPUへ送る必要が出てきます。

デバイスエミュレータからの割り込みを受け、ハイパーバイザはデバイスに対応するRedirection Table Entryの値を参照し、宛先の仮想CPUを選びます。宛先の仮想CPUが決定されたら、ハイパーバイザは宛先CPUのLocal APICのIRRレジスタを更新し、VMCSに割り込みの挿入を設定します。割り込み挿入が設定された仮想CPUがVMEnterされると、以降は内部割り込みと同様に、実機とほぼ同じ手順で割り込みの受付が行われて割り込みハンドラが起動されます。

⑥のEOI書き込みに関しては、Local APICのEOIレジスタへのアクセスを、ハイパーバイザが介入してエミュレーションを行う必要があります。

まとめると、外部割り込みを仮想化するには、ハイパーバイザでIO APIC・Local APICのエミュレーションを行い、仮想CPUへ割り込みを挿入する必要があります。

MSI/MSI-X割り込みを用いて割り込む場合

実機では、次の手順で処理されます。

①デバイスがPCI Configuration Spaceに指定されたDestination IDが示すLocal APICへ割り込みを転送
②Local APICがCPUへ割り込み
③CPUはIDT上のゲートデスクリプタをロードし、割り込みハンドラを実行
④割り込みハンドラが外部割り込みを処理
⑤割り込みハンドラがLocal APICへEOIを書き込み、割り込み処理の終了を伝達
⑥IRET命令で直前のコンテキストへ復帰

MSI/MSI-X割り込みを用いる場合の違いは、割り込み先がI/O APICのRedirection Table Entryに書いてあるのではなく、PCI Configuration Spaceに書いてある、という点だけです。これを仮想化する

注1) VMCSのVM-Execution Control FieldsのException Bitmapの各ビットが各例外のベクタ番号に対応していて、ここに1を設定するとその例外が発生した時にVMExitが発生するようになります。通常の例外は基本的にVMExitする必要がありませんが、連載第2回（Intel VT-xの概要とメモリ仮想化）で解説したシャドーページングを行うには、ページフォルト例外でのVMExitが必須になります。

場合、ハイパーバイザで宛先の仮想CPUを選択するときの参照先が変わりますが、あとはI/O APICを通じた割り込みと同じです。

IDT/IDTRと割り込みハンドラ

IDT/IDTRや割り込みハンドラに関しては、とくにハイパーバイザが介入すべき処理はありません。このためVMExitは発生せず、すべてCPUが仮想化を行います[注2]。VT-xにおいて一部の汎用レジスタはVMX root mode/VMX non-root modeの切り替えのときにコンテキストをハイパーバイザで保存する必要がありました。しかし、IDTRレジスタのコンテキスト保存／復帰については、ハイパーバイザは関与しません。

これは、CPUによって行われるためです。ゲストマシン上のIDTの作成や割り込みハンドラのアドレスの設定は、通常のメモリアクセスと同様に行われます。また、VT-x non-root modeでは、実機での動作と同様に割り込みや例外を受け付け、割り込みハンドラを実行する機能が備わっています。ゲストマシンのIDTRは、IDTの構築後に設定されます。なお、一般的な手法ではありませんが、VMCSの設定[注3]によりIDTRへの読み書きを契機としてVMExitを発生させることもできます[注4]。

このうち、②～④でハイパーバイザの介入が不要であることは、すでに説明しました。①ですが、ゲスト環境から内部を発生させ、割り込みハンドラを起動する処理の中で、とくにハイパーバイザの介入は必要ありません。

ただし、ここでもVMCSの設定により、内部割り込みを契機としてVMExitを発生できます。この場合、内部割り込みをゲストOSに渡さずハイパーバイザで横取りして処理をしたり、デバッグ機能としてゲスト環境上の内部割り込みの回数をカウントしたりといった機能を実装できます。しかしながら、そのような使い方は一般的ではありません。

まとめると、内部割り込みの一連の処理に関しては、特にハイパーバイザが介入すべき処理はありません。このためVMExitは発生せず、すべてCPUが仮想化を行います。

Local APICの仮想化

Local APICはメモリマップドI/Oでアクセスするため、通常のメモリマップドI/Oの仮想化手法が使えます。しかし、高速化のためにVT-xにはLocal APICへのアクセスを特別扱いしてハンドルする機能が実装されています。このことは、連載第3回（I/O仮想化「デバイスI/O編」）で解説しました。つまり、図1で示したように、**部分的**にVT-xによる仮想化支援を受けることができます。

しかしながら、このVT-xによる仮想化支援機能はハイパーバイザが何もしなくても完全にCPU側でレジスタ値の更新などを行ってくれるというものではありません。CPUへ外部割り込みを挿入する場合、ゲストOSから見てつじつまが合わなくならないようLocal APICのレジスタ値を同時に設定する作業はハイパーバイザから行う必要があります。具体的には、次の操作を行います。

①割り込み発生時点でIRRレジスタに割り込むベクタ番号のビットをセット
②仮想CPUがVMExitするのを待つ／またはIPIなどを用いてVMExitさせる
③IRRにセットされた最高優先度のビットをクリア、同じビットをISRにセット
④TPRとISRの値に基いてPPRを更新
⑤VM-entry interruption-information fieldに割り込みをセット
⑥VMEnterして割り込みを発生させる

注2） ただし、割り込みハンドラ内でIOポートアクセスなどの操作を行えば、そこではVMExitが発生します。

注3） VMCSのVM-Execution Control FieldsのSecondary Processor-Based VM-Execution ControlsにあるDescriptor-table exitingにビットを立てることで、LGDT、LIDT、LLDT、LTR、SGDT、SIDT、SLDT、STRの各命令を実行しようとした時にVMExitするようになります。

注4） この場合、ハイパーバイザはIDのシャドーイングを行ってゲストOSが意図する割り込みハンドラと異なる割り込みハンドラを設定できます。また、IDTRへのアクセスをイベントとして受け取り、デバッグ機能を実装することもできます。

また、割り込みハンドラの終了を通知するためにゲストOSがEOIレジスタへ書き込んできたときにも、ハイパーバイザの介入が必要です。具体的には、次の作業を行います。

①EOIへの書き込みによりVMExit
②IRRが0ならVMEnterしてゲストへ復帰。IRRに値があればIRRにセットされた最高優先度のビットをクリア、同じビットをISRにセット
③TPRとISRの値に基いてPPRを更新
④VM-entry interruption-information fieldに割り込みをセット
⑤VMEnterして割り込みを発生させる

こちらも、前回の記事で解説した実機上のLocal APICの挙動と同じです。

I/O APICの仮想化

I/O APICもゲストOSからメモリマップドI/Oでアクセスされます。ただし、Local APICと異なり高速化用の特別なVMExitなどの仮想化支援機能はありません。使われ方としては、前述のとおりゲストOSの初期化時に割り込み先CPUの設定をメモリマップドI/O経由で受け取り、仮想デバイスから割り込みを送るときの仮想CPU選択に設定された値を用います。

MSI/MSI-X割り込みの仮想化

MSI/MSI-X割り込みの場合は、PCI Configuration Spaceへの書き込みにより割り込み先CPUの設定を受け取ります。書き込むデバイスやレジスタのフォーマットは違いますが[注5]、使い方はほぼI/O APICと変わりません。

物理ハードウェアからの割り込みへの対処

VMX non-root modeの実行中に物理ハードウェアから割り込みが来た場合、ハイパーバイザでこれを受け取り割り込みハンドラを起動して処理する必要があります。このために、ハイパーバイザはVMCSの初期化時にVM-Execution Control FieldsのPin-Based VM-Execution ControlsにあるExternal-interrupt exitingにビットをセットします。これにより、ゲストマシンは外部割り込み発生時にVMExitするようになります。また、M-Exit Control FieldsにあるVM-Exit ControlsのAcknowledge interrupt on exitビットを1に設定した場合、外部割り込みはVMExit時に"**acknowledged**"になり、割り込みベクタ番号はVM-Exit information fieldsのVM-exit interruption informationに保存されます。VMMはこのベクタ番号を参照して、割り込みハンドラを起動し割り込みを処理します。

一方、Acknowledge interrupt on exitビットを0に設定した場合は割り込みは"**acknowledge**"されず、RFLAGSレジスタのIFビットでマスクされている状態になります。このままIFフラグをセットすれば、IDTに設定された割り込みハンドラが起動して割り込みを処理できます。通常のOSの上にハイパーバイザを実装する方式では、IDTによる割り込みハンドルが行われているため、後者の方法を取る場合がほとんどです。

まとめ

いかがでしたでしょうか。今回はIntel VT-xにおける割り込みの仮想化方法を中心に解説してきました。次回はソフトウェア側の実装に移り、「VT-xを用いたハイパーバイザの実装方法の基礎」を中心に解説します。SD

注5）参考資料
（http://d.hatena.ne.jp/syuu1228/20120105/1325757315）

Emacs 64bit化計画!

使いやすいエディタ環境を作りませんか!

第6回 COM対応(その1)

太田 博志 *Hiroshi Oota* ● TwitterID @h2oota　イラスト:黒崎 玄

はじめに

さてWin64版のGNU Emacsを紹介する目的で始めた本連載も後2回の予定です。今回と次回でWin64版開発の大きな目標であったCOMサポートについて解説していきます。

概要

Windows上で作業していると他のアプリケーションとの連携ができたらと思う場面に出会うことがよくあります。このような場合は、他のアプリケーションに移り、結果をコピーするかファイルに保存して利用することになります。

しかし、Emacs上にワークフローを構築していたりすると定型的な操作が面倒になります。EmacsはUNIX上で開発されたものなので、UNIX上のパイプを基本としたツール類を取り込むためのパッケージは数多く開発/公開されていますが、Windowsではパイプインターフェースのツールはあまり多くありません。Windows上で動作するLL言語はCOM(OLE)インターフェースを実装することでパイプインターフェースでは得られないアプリケーションとの連携を実現していますが、やはり、他の言語で記述するのは、それらのスクリプトの出力をもう一度パースする必要があり、隔靴掻痒(かっかそうよう)の感があります。

以前からの構想であったCOM(OLE)を実験的に実装してみました、まだ、ほんの基本部分しかできていませんが、今回と次回の2回で紹介していきます。COMオブジェクトはMicrosoftの営業戦略上OLE2、ActiveX、COMと複数の名称で呼ばれますが、本稿ではCOMを使います。

COM or .NET

マイクロソフトのコンポーネント技術開発は.NETフレームワークにすでに移行していますが、COMもまだまだ現役で、何よりCでインターフェースが取れるメリットが大きいです。

Emacsから.NETオブジェクトを利用するためにはC++/CLIが使えるかもしれませんが、もともとC++/CLIは.NETからCやC++のライブラリを楽に使用するための言語で、逆方向で使うのはけっこう苦労しそうです。.NETはひとまず保留として今回はCOMのサポートにしました。

組み込み方法の検討

COMサポートのあるLL言語のほとんどはオブジェクトシステムをもっていて、COMオブジェクトもネイティブオブジェクトと同様の記法で操作可能となっています。Emacsではネイティブなオブジェクトシステムのサポートはなく、EmacsLispで書かれたeieioやlunaと呼ばれるパッケージが存在します。eieioはEmacsの公式配布にも含まれていますが、筆者はeieioを使用した経験がないので、今回は使用せずネイティブな関数でのサポートとします。

ネイティブな関数でサポートしていけばeieioに組み込むことも可能でしょう。

Perlでの実装

リスト1にPerlでのCOMオブジェクトの利用例を記載します。Perlのオブジェクトと同様に->演算子でメソッド呼び出しが可能になっています。ActiveWorkbook、ActiveSheet、Range、SelectはExcelオブジェクトからインポートした名前がPerlオブジェクトのメソッド呼び出しと同じ記法で呼び出しています。プロパティはPerlのハッシュと同じ記法でアクセスすることも可能です。

▼リスト1　Perlの例

```
# perlでのCOM（OLE）使用例
# Excelオブジェクトを作成する
$Excel = Win32::OLE->GetActiveObject('Excel.Application')
# ワークブック、ワークシート、レンジを指定し、選択状態にする。
$Excel->ActiveWorkbook->ActiveSheet->Range("B1")->Select;
```

実装方法

C言語レベルでCOMの処理の流れは、おおざっぱに次のようになります。

①COMを利用するためOleInitializeを呼び出し初期化する
②CLSIDFromProgIDを使い、COMオブジェクトのPROGID("Excel.Application")からCLSIDを取得する
③CoCreateInstanceを使いCOMオブジェクトを生成する。IDispatchインターフェースが取得できる
④取得したIDipspatchインターフェースのinvokeを呼び出しCOMオブジェクトを操作する
⑤OleUninitializeでCOMの利用を終了する

Emacsのインターフェース

リスト1に戻って検討してみましょう。この実行式の意味は、

①**"Excel.Application"**……Excelオブジェクトを生成する。"Excel.Application"はCOMオブジェクトを識別するPROG_IDと呼ばれる
② **$Excel->ActiveWorkbook**……ExcelのActiveWorkbookプロパティを取得する。ActiveWorkbookプロパティはWorkbookオブジェクトを値として持つ
③**->ActiveSheetの部分**……①で取得したWorkbookオブジェクトのActiveSheetプロパティを取得している。戻り値としてWorkSheetオブジェクトが得られる
④同様に②の戻り値のWorkbookオブジェクトのActiveSheetプロパティを取得する。ActiveSheetプロパティの値はWorkSheetオブジェクトである
⑤Worksheetオブジェクトからセル範囲を指定してRageオブジェクトを取得する。Rageオブジェクトは複数のセルを持てるが、この例ではB1のセル1つだけ
⑥最後にRangeオブジェクトのSelectメソッドを実行してセルをセレクト状態にする

これらのサポートのために2つの関数を作成します（**リスト2**）。

・win32com-create prog-id

PROGIDまたはCLSIDを文字列で与えてCOMオブジェクトを作成します。引数をPROGIDとしてCLSIDに変換を試みます、失敗したら文字列をCLSIDを表す文字列とします。

・win32com-invoke obj method &aux args

objで指定されたCOMオブジェクトのmethodを呼び出します。引数が与えられた場合はVARIANT型に変換します。呼び出し結果はEmacsLisp型に変換します。

各々の処理をC言語風に書くとリスト2のような流れです（実際のAPIではありません）。

これらはGNU Emacs（Win64版）のダイナ

▼リスト2　win32comサポート関数

```
/* win32com-create */
/* ProgIDから CLSID を取得 */

if (CLSIDFromProgID(progid, &clsid) != SUCCESS)
    CLSIDFromString(progid, &clsid);
/* CLSID からIDispatchインターフェースを取得(IID_IDispatchを指定) */
CoCreateInstance(&clsid, &IID_IDispatch, (void*)&pDispatch);

/* win32com-invoke */
/* GetIDsOfNamesでmethodのDispIDを得る */
pDispatch->GetIDsOfNames(method, &DispID);
/* Invoke を使って Select メソッドを実行 */
pDispatch->Invoke(obj, DispIP, args);
```

ミックライブラリサポート機能を利用して実装します。**リスト3**のような構造体を定義しpDispatchをラップします。

オブジェクトの表記

ここまででExcelを起動してメソッドを呼び出すための準備ができました。先ほどのリスト1は作成した関数を使い**リスト4**のようになりますが、Perlの例に比べて格段に見難いです。可読性の向上を検討しましょう。リスト1をそのままEmacsLisp風に記述すると**リスト5**のように記述することになります。

少しは見やすくなりましたが表記が逆順になり、Rangeの引き数が離れてしまいイヤな感じです。さらに、COMオブジェクトではこれらの名前はメソッド名なので、オブジェクト固有の処理を行います。同じ名前でも処理対象オブジェクトの種類により別のものなのです。EmacsLispには単一名前空間なので、EmacsLispで記述されるアプリケーションは関数名、変数名としてモジュール名をプリフックスにした名前を付けることで名前の衝突を防いでいます。

名前の衝突を避ける必要があるので、この流儀に従いそれぞれの名前が衝突しないように無理やりプリフィックスをつけると**リスト6**のような感じでしょうか。

見やすくならず逆にイヤさが倍増してしまいました。しかもこの方法ではExcelオブジェクトを生成したときにすべての名前をインポートして関数として定義する必要があります。この

▼リスト3　win32com_object

```
struct win32com_object
{
  struct vectorlike_header header;
  Lisp_Object type;

  void (*finalizer)(struct Lisp_Vector *);
  IDispatch *pDispatch;
};
```

▼リスト4　EmacsLisp風記述(その1)

```
(setq excel (win32com-create "Excel.
Application"))
(win32com-invoke
   (win32com-invoke
     (win32com-invoke
        (win32com-invoke
       excel
             'ActiveWorkbook)
          'ActiveSheet)
     'Range "B10")
    'Selecet)
```

▼リスト5　EmacsLisp風記述(その2)

```
(Select
   (Range
      (ActiveSheet
         (ActiveWorkbook Excel))
      "B1"))
```

▼リスト6　EmacsLisp風記述(その3)

```
(Microsoft_Excel-Workbook-Range-Select
   (Microsoft_Excel-Workbook-Range
     (Microsoft_Excel-Workbook-ActiveSheet
        (Microsoft_Excel-ActiveWorkbook Excel))
     "B1"))
```

▼リスト7　EmacsLisp風記述（その4）

```
(win32com-invoke "Excel.ActiveWorkbook.ActiveSheet.Range" "B1")
```

▼リスト8　EmacsLisp風記述（その5）

```
(win32com-invoke
   (win32com-invoke excel 'ActiveWorkbook.ActiveSheet.Range "B1")
   'Select)
```

方法は採用できません。

それではPerlの記述に戻って

```
$Excel->ActiveWorkbook->ActiveSheet-
>Range("B1")
```

この部分をEmacsLisp風に簡潔に書く方法を考えます。

リスト7のように記述できればだいぶすっきりしますが、Excel.ActiveWorkbook.ActiveSheet.Rangeの部分を文字列で与えるのはかっこよくないのでシンボルとすることにします。シンボルは評価されてしまうのでクオートして

```
(win32com-invoke 'Excel.ActiveWorkbook.
ActiveSheet.Range "B1")
```

となります。マクロを使ってクオートなしの形式をリスト4のように展開してやる方法もありますが、中間値となるCOMオブジェクトの生存期間は短いほうが良く、C言語で記述するCOMインターフェース内で中間オブジェクトは使用後すぐに解放する方法にします。

全体は**リスト8**となります。これで大分すっきりしていい感じですが、excelは変数名なので'excel.ActiveWorkbook.ActiveSheet.Range とクオートしてしまう部分がしっくりきません。excelの部分は外に出します。eieioはCLOS風の総称関数をサポートしているので、このほうがeieioとの相性も良いはずです。

COMオブジェクトは階層が深くなり、途中で改行したくなる場合もありますが、クオートではそのようなことはできません。リストにする方法もありますがクオートが必要になり、あまりかっこよくありません。ベクター型を使うとどうでしょう。

```
(win32com-invoke
  (win32com-invoke excel
  [ActiveWorkbook ActiveSheet Range] "B1")
   'Select)
```

かなりすっきりしました。クオート方式と同じ"."で接続する形式との混在も許して、

```
(win32com-invoke excel [ActiveWorkbook.
                        ActiveSheet.Range] "B1")
'Select)
```

とできれば長くなっても適宜改行できます。Common Lispのようなリーダーマクロが利用できれば、もっとかっこいいシンタックスも可能でしょうが、あまり贅沢も言っていられません。これを採用します。

win32com-invokeでは渡されたメソッドがベクター型ならその要素（シンボル）の表示文字列を取得し、"."で分割し、プロパティを表すシンボルとして扱ってやればうまくいきそうです。

データ型変換

COMオブジェクトへのデータの受け渡しはVARIANT型を使用して行います。VARIANT型とは型情報も持ち、多様な型のデータを格納できます。この型とEmacsLispのデータを適切に相互変換することが必要です。

文字列型はutf-8とutf-16（UNICODE）の相互変換も必要です。多次元のVT_ARRAYはベクターを要素に持つベクターに対応させました。**表1と表2**に示します。

▼表1　EmacsLispからCOMオブジェクト

EmacsLisp	COMオブジェクト
win32com	VT_DISPATCH
string	VT_BSTR
integerp(32ビット以内)	VT_I4
integerp(32ビット超)	VT_R8
float	VT_R8
nil, t	VT_BOOL
sequence	VT_ARRAY
要素が整数3つのリスト[注2]	VT_DATE

▼表2　COMオブジェクトからEmacsLisp

COMオブジェクト	EmacsLisp
VT_DISPATCH	win32com
VT_UNKNOWN	win32com
VT_UI1,VT_I2,VT_I4	integer
VT_BSTR	string
VT_ERROR	integer
VT_BOOL	nil, t
VT_DATE	要素が整数3つのリスト[[fn:date]]

インストール、利用法

筆者のWebページ[注1]からダウンロードしてwin32com.dllをc:/Program Files/GNU/emacs 23/binにコピーしてください。

最新情報がHPおよびreadme.txtに書かれているので目を通してください。

● 関数

(win32com-create progid)

……progidまたはclsidを文字列で与えてCOMオブジェクトを作成する

(win32com-invoke obj method &aux args)

……objのmethodを呼び出し、結果を返す

(win32com-getproperty obj property-name)

……property-nameで指定するobjの属性値を取得する

(win32com-putproperty obj property-name val)

……property-nameで指定するobjの属性値をvalに設定する

(win32com-destroy obj)

……objを解放する

● Excelのコントロール例

```
(setq win32com (load-dynamic-library "win32com.dll"))
```

……win32comをロードする

```
(setq excel (win32com-create "Excel.Application"))
```

……Excelオブジェクトを作成

```
(win32com-putproperty excel 'Visible t)
(win32com-putproperty excel 'Visible nil)
```

……表示/非表示の切り替え

```
(setq bk (win32com-invoke excel #[Workbooks] 'Add))
```

……新規ブックの作成

```
(setq sh (win32com-getproperty bk 'ActiveSheet))
```

……シートの取得

```
(win32com-putproperty
  (win32com-getproperty sh #[Range] "B1")
'Value "Test")
```

……値を書き込む

```
(win32com-invoke excel 'quit)
(win32com-destroy excel)
```

……excelを終了

まとめ

今回はGNU Emacs(Win64版)の大きな開発動機であったCOMサポートの解説を、COMオブジェクトサポートのCでの実現方法とEmacs Lispでの表記方法に関する解説を中心に行いました。次回ではこれを応用した例題を紹介しようと予定しています。SD

注1）http://hp.vector.co.jp/authors/VA052357/

注2）(current-time)が返す1970/1/1からのマイクロ秒でのカウントを3つの整数に分解しリストにしたもの。

Column Emacs24サポートに関して

Emacs24がリリースされてからかなり時間が経過し、首を長くして待っている方も国内に数名はいらっしゃると思います。重い腰を上げてEmacs24をダウンロードしコンパイルしてみましたが、ダイナミックライブラリサポートに必要な構造が大きく変更されていました。

Emacs24ではガベージコレクタが間接参照オブジェクトを管理しないように変更されました。
間接参照オブジェクトは静的に確保されるので、ガベージコレクタの管理させるのは無駄な事なので管理外としたのでしょう。
しかし、これはダイナミックローダーにとってはとても痛い変更です。
ダイナミックモジュール中で宣言されている静的領域はアンロードで消滅してしまいます。これでは困るので、ダイナミックローダでは間接参照オブジェクトを動的に確保しています。
Emacs23までは間接参照オブジェクトもガベージコレクタの管理下にあったので、アンロード時は特別な処理なしにまま参照が無くれば回収してくれていました。
対応方法検討中……

とブログに書きましたが、まだ手がついていません。win32comサポートの次に行いますので、今月号（2月号）が発売するころには完成しているといいなぁ……。

テキストデータならお手のもの

開眼シェルスクリプト

㈲ユニバーサル・シェル・プログラミング研究所　http://www.usp-lab.com
上田 隆一　UEDA Ryuichi　Twitter @uecinfo

第14回 簡易メーラーを作る ――メールファイル操作の応用

はじめに

前回に引き続き、サーバのMaildirに溜まったメールをいじります。今回は、メーラー(ただしリードオンリー)を作ってみます。この中でシェルの機能を使い、届いたメールのフィルタリングやCLI(Command Line Interface)での最低限必要なインタラクティブな操作を実現します。

この企てを考えついたのは、仕事中にいちいちブラウザやGUIメーラーを開くのが面倒だと思ったからです。CUIのメーラーは便利なものがいろいろあるのにリードオンリーのものを作ってどうするんだという話ですが、筆者としては、既存のものではこれがちょっと気になります。

束縛するインターフェースは作るな。
――ガンカーズのUNIX哲学から

メーラーに入り、エディタのような画面が開いてプロンプトの$が消えてしまった瞬間、我々はgrepが使えないことを覚悟させられます。メールなんてせいぜいお客さんの名前で検索をかけて、あとは見ないで捨てますので[注1]、これは困ります。プロンプトが消えるのはエディタもそうですが、エディタと違ってメールリーダに熟達しようという気が筆者に起こりません。

ということで、怠け癖が極限に達すると人はこんなシェルスクリプトを書くという例をお見せしたいと思います。そういやこんな言葉もあったなということで、以下の名言を。

「私は発明が必要の母だと考えません。私のなかでは発明は暇と直接関係していて、多分怠惰とも関連しています。面倒を省くという点で。」
――アガサ・クリスティー

簡易メーラーの仕様

今回はMaildirのnewに届いたメールを取り込んで、

- フィルタのルールに応じて振り分けて受信トレイに置き、
- Vimでメールを読み込み専用で開いて表示し、
- 見たメールを未読トレイから既読トレイに移す

というツールを作ります。Maildirについては、前号と前々号の本連載で説明していますが、要はメールアカウントの~/maildir/new/というディレクトリに1メール1ファイルでメールが届く方式のことです。

今回は、メールの届くサーバにメーラーを作り込みます。サーバはVPS上で動いています。**図1**に環境を示します。

制限事項など

このサーバではSMTPサーバが動いており、Maildir方式で各アカウントにメールを配信しています。今回は既読のメールを~/Maildir/new/から~/Maildir/cur/に移すという操作をしますので、ほかのメーラーとの併用は考えません。

注1) 本当のような、嘘のような……。

今回はこの環境に直接メーラーを作っていきますが、自分のノートPCなどリモートから使うメーラーを作ることも可能です。このときは、スクリプトで使うコマンドをsshでリモートから動かせるようにします。それをやると解説するにはコードが長くなるので今回はやめておきます。

また、最近では単なるHTML形式を越えたグラフィカルなメールがありますが、そういうものを見るのは諦めます。たいていの場合、その手のメールは筆者にとって重要ではありません。

最後にお断りですが、今回はやることが多いので、Tukubaiのコマンドについて説明していません。plus、self、loopj、gyo、delfがTukubaiのコマンドです。https://uec.usp-lab.comで機能をお調べください。

メールの取り込み処理を作る

最初に、バックエンドで受信したメールをメーラーに取り込んで整理する部分を作ります。まず、**図2**のようにディレクトリを準備します。

- DATA：前処理をしたメールを整理して置く場所
- FILTERS：メールを振り分ける条件を書いたスクリプトの置き場所
- TRAY：受信トレイ

これらのディレクトリに対し、「~/Maildir/new/に届いたメールをDATAに取り込んで、FILTERS内のフィルタにマッチしたものをTRAYに置く」という働きをするスクリプトFETCHERを作りましょう。

まず、フィルタリングの前準備としてメールを加工するところまでを**リスト1**のように記述します。**$1**に~/Maildir/new/下のファイル名を指定します。8～12行目がエラーチェック関数、13行目がメールがあるかどうかのチェック、それ以降がメールを扱いやすいように変換する部分です。変換部分では、

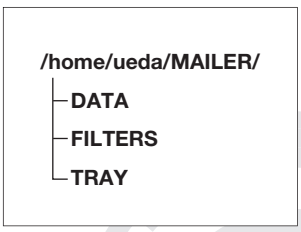
▼リスト1　FETCHERの前半部分

```
01 #!/bin/bash
02 # FETCHER <mailfile>
03 # written by R. Ueda (USP lab.) Nov. 20, 2012
04 dir=~/MAILER
05 mdir=~/Maildir
06 tmp=~/tmp/$$
07
08 ERROR_CHECK(){
09         [ "$(plus ${PIPESTATUS[@]})" -eq 0 ] && return
10         rm -f $tmp-*
11         exit 1
12 }
13 [ -f "$mdir/new/$1" ] ; ERROR_CHECK
14
15 # データのUTF8変換、整形済みヘッダ作成##############
16 nkf -wLux "$mdir/new/$1"                          |
17 tee $tmp-work                                     |
18 # ヘッダを作る
19 sed -n '1,/^$/p'                                  |
20 awk '{if(/^[^ ¥t]/){print ""};printf("%s",$0)}' |
21 #最初の空行の除去と最後に改行を付加
22 tail -n +2 | awk '{print}' > $tmp-header
23 ERROR_CHECK
24 #ヘッダと本文をくっつける。
25 sed -n '/^$/,$p' $tmp-work      |
26 cat $tmp-header - > $tmp-utf
27 ERROR_CHECK
```

▼図1　環境

```
$ cat /etc/redhat-release
CentOS release 6.3 (Final)
$ uname -a
Linux mail.usptomonokai.jp 2.6.32-279.5.2.el6.x86_64 #1 SMP
 Fri Aug 24 01:07:11 UTC 2012 x86_64 x86_64 x86_64 GNU/Linux
$ bash --version
GNU bash, version 4.1.2(1)-release (x86_64-redhat-linux-gnu)
（省略）
```

▼図2　ディレクトリ構成



- ヘッダ部分を加工したもの($tmp-header)
- 検索／表示のためにUTF-8変換したもの($tmp-utf)

を作っています。

ヘッダの加工では、19行目のsedで取り出し、20行目のawkでヘッダに入っている余計な改行をとる処理をしています[注2]。**リスト2**は、To:に複数のアドレスが指定されているヘッダの例ですが、こうやって改行をとっておけばTo:をgrepするだけで全部のアドレスが取得できます。22行目のawkは最終行に改行が抜けたテキストに改行を付ける常套手段です。

ERROR_CHECKはコマンドやパイプラインの終了ステータスを監視し、エラーがあったら処理を止める関数です。13行目の「指定したファイルがMaildirにあるか」のチェックは、DATAディレクトリ内を汚さないために必須です。

フィルタを準備

後半部分を示す前に、このメーラーで作る「フィルタ」をお見せします。まず、「all」という名前で次の極小スクリプトを用意しました。allは必ずこのメーラーに準備しておきます。

全部受理するallフィルタ
```
$ cat ./FILTERS/all
#!/bin/bash
true
```

ほかにも次のようなものを用意しました。これは、とあるFreeBSDのサーバから届くシステム管理用メールに反応するフィルタです。

rootからのメールかどうか調べるフィルタ
```
$ cat ./FILTERS/bsd.usptomo.com
#!/bin/bash
grep -i '^from:' < /dev/stdin   |
grep -q -F 'root@bsd.usptomo.com'
```

このように、標準入力からメールを読み込んで、条件にマッチしたら終了ステータス**0**を返すスクリプトを準備しておきます。もちろん、ほかの言語を使ってもいいですし、もっと長いフィルタを作ってもかまいません。

この方法をとっておくと、たとえば優秀なスパムフィルタがあったときに、それをラッパーするシェルスクリプトを書けばそれを利用できるので、メーラーの方法に束縛されることがなくなります。執筆にあたってスパムフィルタについては何も調査していませんが、何も心配していません。まさにUNIX哲学。

フィルタリング

では、FETCHERの後半部分を**リスト3**に示します。12行目までででメールのヘッダをフィルタごとの新着トレイに置いて、万事うまくいったら13行目以降で残りのファイル処理を確定しています。

トレイにはヘッダのファイルを置いて「そのトレイにメールがある」という目印代わりにします。新着のメールは、たとえばフィルタallに適合したものは./TRAY/all/new/下に置きます。あとのリーダーのところで出てきますが、既読のメールは./TRAY/all/20121125/というように日付のディレクトリを作って整理します。

取り込んだメールやヘッダのファイル名には、整理のためにもとのメールファイル名の頭に年月日と時分秒を付けておきます。その処理のた

注2) このコードは折り返した行の頭に空白があることを期待していますが、厳密にはあるとは限りません。

▼リスト2　メールヘッダのTo:部分の加工例(改行をとる)

before
```
To: ueda@xxx.jp, r-ueda <r-ueda@yyy.com>,
        Ryuichi UEDA <ryuichiueda@zzz.com>
```

after
```
To: ueda@xxx.jp, r-ueda <r-ueda@yyy.com>, Ryuichi UEDA <ryuichiueda@zzz.com>
```

めに、4、5行目でファイル名のUNIX時間から年月日、時分秒を求めています。2012年12月号の本連載で説明したように、メールのファイル名の先頭には10桁で1970年1月1日からの秒数がついており、

```
$ date -d @1234567890
2009年  2月 14日 土曜日 08:31:30 JST
```

のように、dateコマンドで日付と時間に変換できます。

同じく4、5行目の**${1:0:10}**は**$1**の先頭から10文字という意味です。次のように、任意の変数に対して使えます。

```
$ A=12345
 2文字目（先頭を0と数えて1文字目）から3文字抽出
$ echo ${A:1:3}
234
```

7～12行目のfor文で、フィルタに1つずつUTF8変換したメールを入力していきます。8行目のcontinueは、for文のそれ以降の文をスキップするコマンドです。フィルタにマッチしたときだけ、9行目以降の処理が行われ、フィルタの新着トレイにヘッダのファイルが置かれます。

14～16行目はかなり変な書き方をしていますが、これはERROR_CHECKをいちいち書くのを避ける小技です。コマンドを全部**&&**でつないで、どれか1つが失敗したらそこで終わってERROR_CHECKに処理が飛び、**exit 1**します。

FETCHERができたので、~/Maildir/new/下のメールを指定して実行してみます。**図3**のように、各フィルタの新着トレイにメールがあることが確認できます。

リーダーを作る

では、リーダー（スクリプト名：READER）を作っていきましょう。まずは冒頭部分を**リスト4**に示します。READERにはオプションでトレイのパス、メールをリスト表示するときに何件表示するかを指定します。

▼リスト3　FETCHERの後半部分

```
01 # フィルタ ####################################
02 cd "$dir/FILTERS" && [ -e "all" ]  ; ERROR_CHECK
03 # ファイル名のUNIX時間から年月日、時分秒を計算
04 D=$(date +%Y%m%d -d "@"${1:0:10}) ; ERROR_CHECK
05 T=$(date +%H%M%S -d "@"${1:0:10}) ; ERROR_CHECK
06
07 for f in * ; do
08         ./$f < $tmp-utf || continue
09         mkdir -p $dir/TRAY/$f/new
10         cat $tmp-header > $dir/TRAY/$f/new/$D.$T.$1
11         ERROR_CHECK
12 done
13 # ファイルを移して終わり ###############
14 mkdir -p "$dir/DATA/$D"                  &&
15 cat $tmp-utf > "$dir/DATA/$D/$D.$T.$1"  &&
16 mv "$mdir/new/$1" "$mdir/cur/$1"
17 ERROR_CHECK
18
19 rm -f $tmp-*
20 exit 0
```

▼リスト4　READERのヘッダ部分

```
01 #!/bin/bash
02 #
03 # READER <dir> <num>
04 # written by R. Ueda (USP lab.) Nov. 20, 2012
05 tmp=~/tmp/$$
06 dir=~/MAILER
07
08 ERROR_CHECK(){
09         [ "$(plus ${PIPESTATUS[@]})" -eq 0 ] && return
10         rm -f $tmp-*
11         exit 1
12 }
13 #先にメールを取得 ################
14 echo ~/Maildir/new/*    |
15 tr ' ' '¥n'            |
16 awk '!/¥*$/'           |
17 sed 's;^..*/;;'        |
18 xargs -r -n 1 -P 1 $dir/FETCHER
19 ERROR_CHECK
```

▼図3　FETCHERの実行

```
$ ./FETCHER 1352657044.Vfc03I468a21M42631.hoge1
$ ls ./TRAY/*/new/*.1352657044.Vfc03I468a21M42631.hoge1
./TRAY/all/new/20121112.030404.1352657044.Vfc03I468a21M42631.hoge1
./TRAY/bsd.usptomo.com/new/20121112.030404.1352657044.Vfc03I468a21M42631.hoge1
```

14～19行目は、さっき作ったFETCHERを使ってトレイを更新する処理です。14～17行目でディレクトリ名を除去したファイルのリストを作り、18行目のxargsでFETCHERに1つずつ処理させています。

ここでは、新着メールがなくてもエラーが発生しないように、細工がしてあります。まず、新着メールがないと*がそのままパイプに通っていきますが、これを16行目のawkで除去しています。grepを使うと検索結果の有無で終了ステータスが変わり、ERROR_CHECKに引っかかるので、代わりにawkを使っています。また、xargsは通常、入力が空でもコマンドを1回実行してしまいますが、これを**-r**オプションで抑制しています。

Maildir/new/に何百もメールがあると、この部分は当然時間がかかります。しかし、こういう場合は別の端末からFETCHERを起動しておけばよいので、気を利かせることはやめましょう。これはCUI信奉者が自分で使うものですので……。

次に**リスト5**のように、メールのリストを表示してメールを選択してもらう部分を記述します。

ここまでの部分を実行すると、**図4**のような出力が出ます。

番号と着信日時、メールのSubjectが表示され、どの番号のメールを見るか入力を促します。

では、リスト5のスクリプトを見ていきましょう。まず2行目で、**$1**で指定されたトレイに移動しています。**cd " ${1:- $dir/TRAY/all/new}"**とありますが、これは「**$1**が空ならば**$dir/TRAY/all/new**」という意味になります。9行目のtailのオプション指定でもこの方法を使っています。

4～10行目はトレイのファイルのリストを作って、リストが空ならそのまま処理を終えるという処理が書いてあります。その後のコードは、各メールの受信時刻とSubjectを抽出し、画面に出力するための細かい文字列処理です。次の2つの表を作っています。

▼リスト5　READERのインタラクション部分

```
01 #メールのリストを作る ########################
02 cd "${1:-$dir/TRAY/all/new}" ; ERROR_CHECK
03
04 #表示対象ファイルの抽出
05 echo *                    |
06 tr ' ' '¥n'               |
07 grep -v '¥*'              |
08 sort                      |
09 tail -n "${2:-10}" > $tmp-files
10 [ $(gyo $tmp-files) -eq 0 ] && rm -f $tmp-* && exit 0
11
12 #subjectのリストを作成
13 cat $tmp-files                    |
14 xargs grep -H -i '^subject:'      |
15 sed 's/:[Ss]ubject:/ /' > $tmp-subject
16 #1:ファイル 2:subject
17 ERROR_CHECK
18
19 #日付のリストを取得し、subjectのリストと連結
20 cat $tmp-files                    |
21 xargs grep -H -i '^date:'         |
22 sed 's/:[Dd]ate:/ /'              |
23 #1:ファイル 2～:date
24 self 1 2 3 4 6                   |
25 sed 's/:[0-9][0-9]$//'           |
26 loopj num=1 - $tmp-subject       |
27 #1:ファイル名 2~日時、subject
28 tac                              |
29 awk '{print NR,$0}'              |
30 #1:リスト番号 2:ファイル名 3～:日時, subject
31 tee $tmp-list                    |
32 #リストの表示
33 delf 2
34
35 cd - > /dev/null
36 echo -n "どのメールを見ますか？(番号):  "
37 read n
```

▼図4　READERのインタラクション出力

```
$ ./VIEWER
1 Sun, 25 Nov 07:10 処理エラー
2 Sun, 25 Nov 06:00 【先着3名】怪しいアレが5000円！【怪しい.com】
3 Sun, 25 Nov 04:00 Logwatch for mail.usptomonokai.jp (Linux)
4 Sun, 25 Nov 03:04 bsd.usptomo.com security run output
(略)
10 Sun, 25 Nov 01:00 【再送】本当に致命的なエラー
どのメールを見ますか？(番号):
```

- 12～17行目：ファイル名とSubjectの対応表
- 19～25行目：ファイル名と時刻の対応表

26行目のloopjでこれらの対応表をくっつけます。あとは新着順に並び替え、番号を付けて**$tmp-list**に表示します。33行目で画面に出力しますが、このときはファイル名をdelfで削ります。

35行目の**cd -**は、前回のcdをする前のディレクトリに戻るためのコマンドで、手で端末を操作するときにもよく使うものです。

36、37行目では、番号を入力するようにユーザに促し、**read n**で番号を受け付けています。端末からユーザが入力した数字(正確には任意の文字列)が変数**n**に代入されます。

最後、**リスト6**に残りの部分を。まず、2行目で入力してもらった番号からファイル名を抽出しています。ここで**n**に変な文字列が入っていると、4行目でファイルがないので弾かれます。あとはメールから必要なヘッダとメールの文を取り出して、viewで開いています。

viewは単にVimをリードオンリーで開くためだけのコマンドです。Vimでファイルを読むので、筆者の場合は普段のVimの使い方でメールが読めます。また、見ているファイルを別のディレクトリにそのまま保存できるなど、筆者と全国1,000万人(？)のVimユーザには異常に便利なメールリーダになります。

viewを正常に閉じると10行目以降で各フィルタの新着トレイから、読んだメールを日付別の既読トレイに移動します。既読のトレイを開いた場合は、とくに何も起こりません。この処理は各フィルタのトレイ全部に対して行います。

終わりに

今回はシェルスクリプトでメールリーダを作ってみました。今後真面目に作り込むと便利になるかもしれません。

返信機能を付けるとすると、おそらくviewで保存したメールを処理し、返信用のメールの雛形を作るスクリプトを作ることになります。メールはmailコマンドか何かで送ればよいですし、メールアドレスの入力が面倒ならVimの補完ツールの利用や、メールアドレスを提示するコマンドを作ればなんとかなるでしょう。

また、「何件メールがトレイにあるか」などは、それこそlsとwcを使えば事足ります。束縛するインターフェースでないので、なんとかなります。

今回は正直言いまして、かなりエクストリームなプログラミングになってしまいましたので、次回からはもうちょっとマイルドな話題として、バイナリデータをいじって遊んでみたいと思います。SD

▼リスト6　READERの後半部分

```
01 #メールを表示 ####################################
02 f=$(awk -v n="$n" '$1==n{print $2}' $tmp-list)
03 m="$dir/DATA/${f:0:8}/$f"
04 [ -f "$m" ]                                          &&
05 grep -E -i '^(from|to|cc|date|subject):' $m > $tmp-work &&
06 sed -n '/^$/,$p' $m >> $tmp-work                     &&
07 view $tmp-work
08 ERROR_CHECK
09 #既読トレイに移す(newの中だけ) ##################
10 for t in $dir/TRAY/* ; do
11         [ -e "$t/new/$f" ] || continue
12         mkdir -p $t/${f:0:8}
13         mv -f $t/new/$f $t/${f:0:8}/$f
14         ERROR_CHECK
15 done
16
17 rm -f $tmp-*
18 exit 0
```

第34回

iPad mini登場! アプリ開発で押さえるべきポイント

スマートフォンの認知度を一般に広めただけでなく、ソフトウェア開発においても新しい波を作り出してしまったiOS。開発者たちは何を見、どう考えているのか。毎回入れ替わりでiOS向けアプリケーション開発に関わるエンジニアに登場いただき、企画・開発のノウハウやアプリの使いこなし術などを披露してもらいます。

嶋田 智成 SHIMADA Tomonari
http://www.blueair.co.jp/

世界を変えるiPad miniの登場

Appleは2012年10月24日にiPad miniを発表し、11月2日に販売を開始しました(**写真1**)。iPadシリーズに追加されたiPad miniは、これまで販売されてきたiPadの後継機ではなく、新たな製品カテゴリとして追加されたiPadです。最大の特徴は、ディスプレイがこれまでの9.7インチから7.9インチになり、片手でつかんで持てるほどに小さく、薄く、軽くなったことです(**写真2**)。

本稿では、この新たな製品カテゴリに対して、iOSアプリ開発者は何を検討すべきかについて考えてみます。

▲**写真1　iPad mini**

写真2　片手でつかめる大きさ▶

iPad miniについて

iPad miniの仕様を見ると、第2世代のiPad 2とそれほど変わりません。最新のiPadやiPhone 5と比べると、同じ最新デバイスなのに見劣りするのは確かです。しかしこれは、デバイスの仕様によってユーザの利用スタイルが変わるだけでなく、ユーザがどのようなシーンでデバイスを使うかを想定した結論でしょう。

iPadの9.7インチのディスプレイは見やすく鮮やかで、ユーザ自身のみならずその周囲にいる人と共有して利用できます。iPad miniの7.9インチのディスプレイでは周囲の人といっしょに見るのは難しいでしょうが、本体はiPadに比べて薄く、軽いので、長い時間片手で持つことができ、ユーザが個人で使うには最適なサイズと言えます。こういった視点でも、従来のiPadとは違ったアプリの方向性が考えられるのではないでしょうか。

ディスプレイに関してもう1点、iPad miniのディスプレイの幅は、左右のベゼルぎりぎりまで広げられています。ベゼルの

▼図1　iPad、iPad mini、iPhone 5の実寸サイズでの表示比較

幅は最も短いところで6.7mmしかなく、持ったときに指がディスプレイに触れてしまうことがしばしばあります。しかし、タッチ操作と誤認識されないしくみが搭載されているので操作への支障はほとんどありません(100%誤動作しないとは言い切れませんが)。たとえばiBooksなどの電子書籍アプリでは、ディスプレイの左右のエリアをタッチすることでページが切り替わりますが、本体を持つ指がディスプレイに触れていても、誤ってページをめくらないようになっています。

7インチタブレット

iPadに代表されるタブレットデバイスは、これまではB5ノートサイズで10インチくらいのディスプレイを搭載しているものが主流でした。7インチのタブレットデバイスはiPad miniの発売とほぼ同時期にAndroidやKindleなどが販売され、にわかに注目が集まりました。iPad miniの発売前に販売が開始されたAndroid OS搭載のGoogle Nexus 7は、iPad miniのライバルと目されている機種です。

7インチタブレットはそのサイズから、携帯コンテンツビューアとして普及することが予想されています。すでに普及しているスマートフォンも携帯コンテンツビューアと言えますが、ディスプレイのサイズが4インチほどしかなく、電子書籍などのアプリでは1画面に表示できるコンテンツのサイズがどうしても小さくなってしまいます。7インチタブレットであれば、文庫本の1ページを原寸で表示することが可能です。

iPad miniのアプリ

iPad miniでは、iPad向けの27万5,000個のアプリをそのまま利用できます。iPhone向けアプリも互換モードで動作するので、App Storeで公開されている約70万個のアプリを利用可能です。またユーザがこれまで入手したアプリを料金を追加することなく利用できます。

iPad mini用アプリ開発の注意点ーデザイナー

ディスプレイとUI要素のサイズ

iPad miniのディスプレイは7.9インチで、iPadの9.7インチと比べると約80%のサイズです。ディスプレイは小さくなりましたが、画素数で見るとiPad 2と同じ縦1024ピクセル、横768ピクセルです。これまでiPad用に作成したアプリのデザインを変更する必要はありませんが、

ボタンなどはiPadの80%のサイズになるため、ユーザが操作するエリアが小さくなります。

図1を参照してください。Appleは画素数を変えずにiPad miniのディスプレイを小さくするために、画素密度を高くしました。画素密度とは画像を表示する画素の細かさのことで、1インチあたりのピクセル数をppi(pixel per inch)という単位で表します。iPad miniを含めiOSデバイスはそれぞれ、画面のサイズとppiが異なります(**表1**)。Retinaディスプレイは、通常のディスプレイと同じサイズでも画素密度が2倍になります。

iPad miniの画素密度は163ppiで、iPhone 3GSと同じです。画素密度が同じなので、iPad miniとiPhone 3GSでは、画素数が同じボタンは見た目のサイズが同じになります。Retinaディスプレイを搭載したiPhone 4以降および第4世代以降のiPod touchは画素密度がiPhone 3GSのちょうど2倍ですが、同じ画素数のボタンはiPad miniと見た目のサイズが同じです。つまり、iPad miniでは、ユーザインターフェース(UI)要素はiPhoneと同じサイズになります。

図1と表1から次のことがわかります。

- iPad 2とiPad Retinaディスプレイモデルでは、画素密度が異なるが、ディスプレイのサイズが同じであり、ディスプレイに表示されるボタンなどのUI要素は同じサイズになる
- iPad miniとiPhone 5では、ディスプレイのサイズが異なるが、iPhone 5の画素密度がiPad miniの2倍になっており、ディスプレイに表示されるボタンなどのUI要素は同じサイズになる(ホームのアイコンサイズは異なる)

パソコン用のアプリは、マウスなどで画面上のカーソルを動かしたりキーボードを使ったりして操作します。一方、iOSなどタッチディスプレイデバイス用のアプリは、ユーザが直接指を使って操作します。パソコン用のアプリでは、カーソルの大きさが同じなので、異なる形状のボタンを設置しても同じように操作できます。タッチディスプレイデバイス用のアプリでは、ユーザの手の大きさや指の形が異なるため、ボタンのサイズや形状が異なるとうまく操作できない場合があります。

また、iOSデバイスでも種類によって画素密度が異なるため、デザイン上は同じサイズのボタンでもiPhone、iPad mini、iPadで異なるサイズで表示され、操作性が異なる場合があることにも気をつける必要があるでしょう。

デザインをどう作るか

iOSアプリ開発では、必要なデザインやアートワークはどのように作ればよいのでしょうか。Appleは、UIや操作などのデザインに関して「iOSヒューマンインターフェイス ガイドライン」[注1]を提供しています。このガイドラインでは、iOSプラットフォームの世界観を自分のアプリに取り入れるためのポイントを説明しています。

UI画面のデザインには、Photoshopなどのペイントソフトを使用します。ペイントソフトはピクセル単位での編集が可能であるため、UI画面のデザインに適しています。

iOSアプリをデザインする場合、ペイントソ

注1) URL https://developer.apple.com/jp/devcenter/ios/library/documentation/MobileHIG.pdf

▼表1 iOSデバイスのディスプレイの仕様(ディスプレイが横向きのとき)

仕様	iPad mini (Wi-Fi)	iPad 2(Wi-Fi)	iPad Retinaディスプレイモデル(Wi-Fi)	iPhone 5(Retinaディスプレイ搭載)
ディスプレイのサイズ(対角)	7.9インチ	9.7インチ	9.7インチ	4インチ
画素数(ピクセル)	1024×768	1024×768	2048×1536	1136×640
画素密度(ppi)	163	132	264	326
ディスプレイの実サイズ(cm)注	16.0×12.0	19.7×14.8	19.7×14.8	8.9×5.0
アスペクト比	4:3	4:3	4:3	16:9

注) 計算式:2.54cm(=1インチ)÷画素密度(ppi)×画素数(ピクセル)

▼表2　iOSデバイスのディスプレイの解像度

仕様	iPad mini/iPad 2	iPad Retina ディスプレイモデル	iPhone 5 / iPod touch (第5世代)	iPhone 4S/iPod touch (第4世代)
解像度	1024 × 768、72dpi	2048 × 1536、72dpi	1136 × 640、72dpi	960 × 640、72dpi
カラーモード	RGB	RGB	RGB	RGB
ピクセル縦横比	正方形ピクセル	正方形ピクセル	正方形ピクセル	正方形ピクセル

▼表3　ピクセルからcmへの変換[注]

	iPad mini	iPad 2	iPad Retinaディスプレイモデル	iPhone 5(Retinaディスプレイ搭載)
44ピクセル	**0.7cm**	**0.8cm**	0.4cm	0.3cm
88ピクセル	1.4cm	1.7cm	**0.8cm**	**0.7cm**

注)計算式：2.54cm(＝1インチ)÷画素密度(ppi)×画素数(ピクセル)

▼表4　cmからピクセルへの変換[注]

	iPad mini	iPad 2	iPad Retinaディスプレイモデル	iPhone 5(Retinaディスプレイ搭載)
10cm	642ピクセル	520ピクセル	1040ピクセル	1284ピクセル

注)計算式：画素密度(ppi)÷2.54cm(＝1インチ)×サイズ(cm)

フトのファイルをデバイスのディスプレイの解像度に合わせて設定します(**表2**)。複数のデバイスに対応するアプリを開発する場合には、解像度が大きいサイズ(Retinaディスプレイのサイズ)をベースにデザインを作成し、最後にダウンスケールします。ダウンスケールの際にはピクセル不足によるエッジのボケに注意してください。ダウンスケール後に1ピクセルの線になるようにするには、Retinaディスプレイのデザインでは2ピクセルの線にしなければなりません。

ハーフピクセル問題

App Storeから入手したアプリを使用していると、ボタンの画像やアートワークがぼやけて見えるアプリがあります。この問題の原因の1つが画像のピクセル不足によるボケです。プログラムで画像を中央に表示する場合、ディスプレイの幅の1/2から画像の幅の1/2を引いた値を、画像を配置するための起点にします[注2]。画像の幅が奇数のとき、幅の1/2の端数が0.5ピクセルになるため、起点の位置にも0.5ピクセルの端数が出てしまいます。画像の解像度の最小値は1ピクセルなので、起点の位置にハーフピクセルの端数が出てしまうとぼやけた感じで表示されてしまいます。この問題を解決するには、画像の幅を偶数にするか、中央に配置する際に起点で端数が出ないようにプログラムで小数点以下を切り捨てた数値を採用します。

ディスプレイの実サイズ

アプリのデザインを行う際には、デザインを実際のサイズでプリントアウトしてペーパーモックを作成します。実際に表示されるサイズでプリントアウトすることで、アプリの使いやすさ(UX：User eXperience)を検証します。iPad miniなどのデバイスではディスプレイのサイズの単位にインチを採用しているため、実サイズのモックを作成するときにcmに変換する必要があります。

Appleの「iOSヒューマンインターフェイスガイドライン」では、タップ可能なUI要素の快適な最小サイズを44ピクセル(Retinaディスプレイでは88ピクセル)と定義しています。44ピクセルおよび88ピクセルをcmに変換すると、**表3**のようになります。また、10cmを各デバイスでピクセルに変換した値を**表4**に示します。

AppleのWebサイトでは、iOSデバイス用のケースを制作する際に使用する外形寸法図が公開されており、ディスプレイなどの細かいサイズを確認できます[注3]。

毎年のように新しいデバイスが登場し、それ

注2)　iOSは左上を起点とします。

注3)　「Designing cases for iPod, iPhone, and iPad」
URL http://developer.apple.com/jp/resources/cases/

▼リスト1　デバイスの機種と世代を特定する

```
// ヘッダの読み込み
#include <sys/sysctl.h>
// (省略)
size_t size;
sysctlbyname("hw.machine", NULL, &size, NULL, 0);
char *machine = malloc(size);   // メモリを確保する
sysctlbyname("hw.machine", machine, &size, NULL, 0);    // 機種情報を取得する
NSString *platform = [NSString stringWithUTF8String:machine];
                                // char型からNSString型に変換する
free(machine);                  // メモリを解放する
NSLog(@"%@", platform);         // 出力する

// 出力
// iPad 2,5：iPad mini Wi-Fiモデル
// iPhone5,2：iPhone 5 au版
```

によって仕様、ディスプレイのサイズや解像度、画素密度が多様化しています。最近ではiOSデバイスにiPad miniやiPhone 5が追加され、画素密度や解像度が変更されました。新しいデバイスが登場しても、ユーザは何も意識せずに既存のiOSアプリをそのまま新しいデバイスで使用できます。これは、iOSプラットフォームとiOSデバイスを提供するのがAppleのみであり、それによってデバイス間の互換性を高いレベルで保つことができているからです。

iPad mini用アプリ開発の注意点ー開発者

すでにリリースされているアプリをiPad miniに対応させる場合、変更することはとくにありません。iPad miniに搭載されているカメラやGPS、ジャイロスコープなどデバイスに固有の機能は、最新の第4世代のものと仕様はやや劣るものの同じです。アプリでも、iPadの種類に関係なく、デバイスに固有の機能に同じ方法でアクセスできます。iOSプラットフォームは使用するデバイスに依存しないように設計されており、アプリもデバイスに関係なく動作します。iPad miniの仕様はiPad 2とほぼ同等なので、負荷がかかる処理やメモリを多く使用するアプリを作成する場合には注意が必要です。

AppleのDeveloperサイトでは、デバイスの互換性リストを公開しており、デバイスごとに使用できる機能を確認できます[注4]。

プログラムで iPad miniを識別

アプリでデバイスごとに処理や使用するアートワークを制御したい場合があります。ユーザが使用しているデバイスの機種と世代を特定する方法はいくつかあります。たとえば、ディスプレイのサイズを取得してそのサイズから使用しているデバイスの機種を特定できます。この方法では、ディスプレイのサイズ(解像度)からデバイスを識別します。解像度が同じで画素密度が異なるデバイスを識別する際には向いていません。デバイスの機種と世代を特定するコードを**リスト1**に示します。

機種情報から対応するデバイスを特定する場合、次のサイトが参考になります。

- 「The iPhone Wiki Models」
  URL http://theiphonewiki.com/wiki/index.php?title=Models

新たなデバイスの登場でアプリ開発者は得をするのか

iOSデバイスは毎年ほぼすべての機種が更新

注4)「Appendix B: Device Compatibility Matrix」
URL http://developer.apple.com/library/ios/#documentation/LanguagesUtilities/Conceptual/iTunesConnect_Guide/B_DeviceCompatibilityMatrix/DeviceCompatibilityMatrix.html

されます。更新によってiOSデバイスの出荷台数は増え続け、現在ではiOSデバイスの出荷台数がMacの出荷台数を上回っています。デバイスの種類と出荷台数が増えることでアプリの売り上げもまた増えるかというと、必ずしもそうとは限りません。これはiOSデバイスでアプリを入手するときに使用するApple ID(iTunesアカウント)に関係します。1つのApple IDで10台のデバイス(Macを含む)を管理できます。つまり、1つのApple IDで入手したアプリは最大10台のデバイスにインストール可能です。有料アプリは一度購入すると、再度購入することはできません。ユーザの多くは、Apple IDを1つだけ持ち、iPhone、iPad、iPod touchなど複数のiOSデバイスを更新します。そのため、iOSデバイスが増えることがアプリの売り上げに直結するとは限らないのです。

実はデバイス数の増加よりもApple IDのアカウント数の増加がポイントになります。Apple IDのアカウント数は2012年9月時点で4億3,500万と発表されました。2012年10月には、これまでのアプリのダウンロード数は350億回、Appleが開発者に支払った金額は65億ドルに達したと発表されています。2012年3月の発表からわずか半年でダウンロード数は100億回、開発者への支払い額は25億ドル増えたことになります。iOSアプリの市場がまだまだ拡大を続けていることは事実です。

ビジネス、教育機関向けのアプリ開発

2012年9月から、日本でも「Volume Purchase Program」が始まりました。これは、App Storeで販売されているアプリを一括購入(ライセンス購入)できるプログラムです。iOSデバイスを複数台導入した企業や教育機関などで、業務や授業に使用するアプリを購入する際に利用できます。通常のアプリ購入では同じApple IDで同じアプリを複数購入することはできません。この一括購入プログラムを使用すれば、デバイスの台数分同じアプリを購入できます。

開発者にとっても、コンシューマ市場向けのみビジネスユース市場にアプリの販売を拡大できるので、売り上げアップにつながります。開発者がVolume Purchase Programに参加するには、iTunes Connect[注5]でアプリを申請する際にビジネス向けの販売を許可するだけです。教育機関向けの一括購入プログラムではボリュームディスカウントがあり、開発者が許可している場合にのみ同じアプリを20個以上購入すると50%の割引価格になります。

また、Volume Purchase Programでは、企業向けにカスタマイズされたアプリを非公開で開発者から調達できる、カスタムB2Bアプリケーションシステムを利用できます。ビジネス向けアプリの導入システムが整備されたことで、開発者はビジネス市場に参入しやすくなりました。Volume Purchase Programについては次のサイトを参照してください。SD

- 「ビジネス向けのApp Store一括購入」
  URL http://www.apple.com/jp/business/vpp/
- 「Apple Volume Purchase Program」
  URL http://www.apple.com/jp/education/volume-purchase-program/
- 「iTunes Connectデベロッパガイド」
  URL https://developer.apple.com/jp/devcenter/ios/library/documentation/iTunesConnect_Guide.pdf

注5) URL https://itunesconnect.apple.com/

● 嶋田 智成(しまだ ともなり) 合資会社ブルーエアー(Blueair Inc.) 代表取締役

大学在学中の2005年に制作会社の合資会社ブルーエアーを立ち上げ、WebサイトやWebシステム案件を中心にスマートフォン向けサイトやアプリの開発などにも携わる。現在は講師やiOSデバイスビジネス導入支援、技術コンサルタントとしても活動している。

第34回 マルチプラットフォーム開発環境を使ってみよう(3)

モバイルデバイス初のオープンソースプラットフォームとして、エンジニアから高い関心を集めるGoogle Android。いち早くそのノウハウを蓄積したAndroidエンジニアたちが展開するテクニックや情報を参考にして、大きく開かれたAndroidの世界へ踏みだそう！

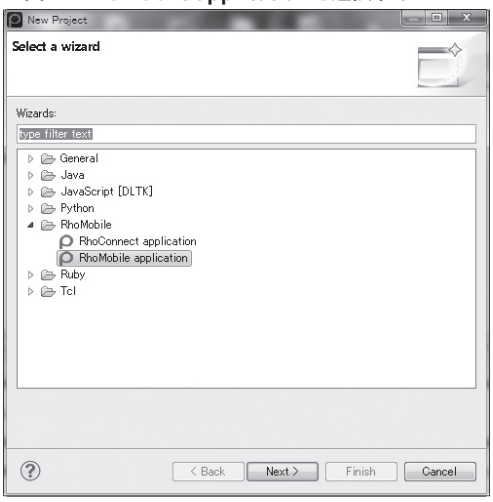
よこいど／日本Androidの会 横浜支部
嶋崎 聡　SHIMAZAKI Satoshi
Twitter @sato_c
motosumi64@gmail.com

はじめに

横浜支部の嶋崎です。Nexus 7がちょうどいい軽さ、画面の大きさ、鮮明さ、値段とこれまでのタブレット端末よりもお得感がありますね。以前は日本から開発用端末の購入はできませんでした。それが、簡単な手続きでGoogle Playからも買えるようになって、便利になったもんだと思います。

さて、今回は前号でインストールしたRhodes環境を使って、実際にアプリを制作する手順を紹介していきます。

▼図1　RhoMobile applicationを選択する



RhoStudioでプロジェクトを作成する

今回作成しているプロジェクトは、GitHubに載せています[注1]。書き換えるたびにコミットしていますので、どこを書き換えたかなどはすべて確認できます。

プロジェクトの作成

まずはRhoStudioでプロジェクトを作成しましょう。Eclipse経験者にはお馴染みと思いますが、[File]メニューの[New]→[Project]を選ぶか、Project Explorerで右クリックして、[New]→[Project]を選んでください。

アプリの種類をRhoConnect applicationまたはRhoMobile applicationから選びます。

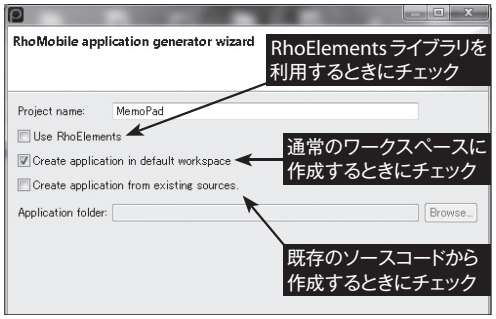
RhoConnectは、モバイルアプリ拡張用サーバといったものです。基幹業務用アプリなど、規模が大きいアプリでモバイル機器同士を連動させるためのバックエンドサーバを作成するためのフレームワークです。端末側にはRhodesを必ず使わなくてはならないという制限はありません。

RhoMobileは文字どおりスマートフォン用のアプリ向けフレームワークで、今回使うのはこちらです(**図1**)。

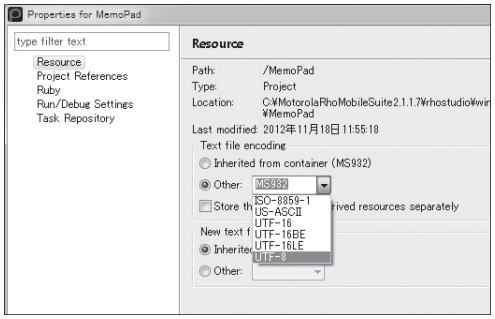
プロジェクトでは、デフォルトではRhoMobile

注1) URL https://github.com/sato-c/MemoPad

▼図2　プロジェクトの内容を設定する



▼図3　漢字コードをUTF-8に設定する



フレームワークを中心にソースコードを作成しますが、[Create application from existing sources]にチェックを入れて既存のソースコードを流用することも可能です。また、[Use RhoElements]にチェックを入れると、RhoElementsライブラリが追加されます。RhoElementsは、ハードウェア依存の機能を利用する際に使うライブラリです。

今回は、MemoPadというプロジェクト名にしました。入力した名前はメニュー画面などの表示にも使われます(**図2**)。

[Finish]ボタンをクリックするとプロジェクトが作成され、Project Explorerに作成したプロジェクトが表示されます。プロジェクトにはフレームワークやベースになるソースコードがすべて入っているので、このまま実行してもちゃんとした画面が表示されます。このあたりは、プロジェクトを作成するとそのままでもViewが表示されるAndroidアプリと似ています。

アプリの内部で漢字を使う場合、デフォルトの漢字コードを設定しておきましょう。プロジェクト の[Properties]で[Text file encoding]がMS932(CP932)になっていたら、UTF-8を選んで[OK]ボタンをクリックします(**図3**)。これで、erb側では日本語を書いておいたのに、画面には謎の文字列が表示されるようなことがなくなります。

View部分にはWebViewが使われており、ファイルはHTMLベースで作られています。ユーザインターフェースはRubyのerbテンプレートエンジンとjQueryMobileを使って構築します。JavaScriptで関数を書けば、インタラクティブなユーザインターフェースも実現できます。

プロジェクトの実行

他のIDEと同じようにRhoStudioでもプロジェクトを作成した後にすぐに実行できます。[Run]メニューの[Run Configurations]を選択して実行用の設定を行ってください。[Run Configurations]画面の左側で[RhoMobile Application]を選択し、左上の新規作成のアイコンをクリックして設定を作成した後、[Project Name]はMemoPad、[Platform]はAndroidを選択します。[Simulator type]は、Android端末がつながっている場合はDeviceを選択すれば、その端末で動きます。それ以外の場合には、エミュレータ(RhoSimulator、Simulator)を選択できるため、環境に応じて変更してください。[Run]ボタンをクリックすると、アプリが実行されます(**図4**)。

Controllerの実装

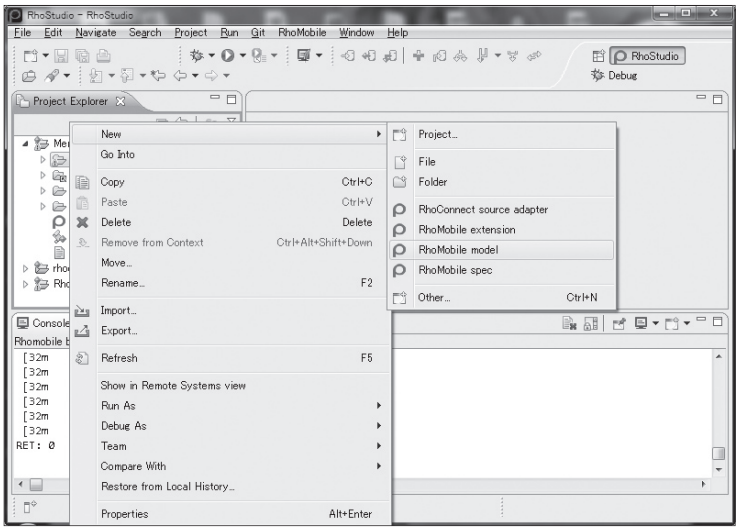
メニューやログイン画面などが動くことを確認できたでしょうか。確認できたら、機能を追加していきましょう。次はControllerの追加です。

Rhodesは、MVC(Model View Controller)というデータ構造、表示、処理を分離した構造で実装されています。Viewは画面表示を行います。それ以外の処理はControllerが行います。データ入出力に関係するModelについては、後ほど説明します。

Project ExplorerでMemoPadを展開し、appを右クリックして[New]→[RhoMobile model]を選

▼図4　そのまま実行するとメニューが表示される

▼図5　Controller/Modelの作成



択します(**図5**)。

Model information画面では、データベース名([Model name])とその構成([Model attributes])を設定します。今回はデータベースの構成を題名(subject)、日付(date)、内容(body)の3つにしました(**図6**)。

[Finish]ボタンをクリックすると、設定したControllerとModelが自動的に生成されます。appフォルダには、指定した名前(今回はMemoPad)でフォルダが生成され、必要なファイルが格納されます。また、testフォルダには、生成されたControllerをテストするためのモジュールが追加されます(**図7**)。

これで、ModelとControllerの設定は終わりました。しかし、このままでは実行する手段がないのでテストができません。そこで、appフォルダの下にあるindex.erbの19行目

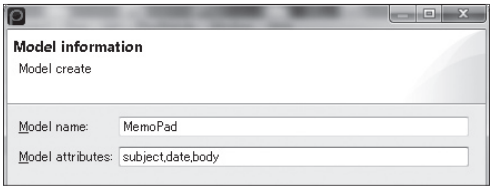
```
<li><a href="#">Add link here...</a></li>
```

を次のように書き換えます。

```
<li><a href="MemoPad">MemoPad</a></li>
```

こうすることで、app配下に作成したMemoPadを呼び出せます。この状態でデバッグ環境から実行してみましょう。デバッグモードで実行する

▼図6　Modelの設定



には、Project Explorerでプロジェクトを右クリックし、[Debug as]→[Debug Configurations]を選びます。先ほど、試しに実行するときに作った設定を選んで[Debug]ボタンをクリックします。自動的にRhoSimulatorが起動してPC用のデバッグ環境が開きます。

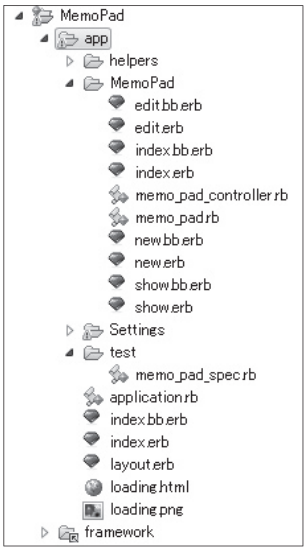
デバッグモードで実行した場合、ブレークポイントを使えます。Eclipseと同じように停止させたい行をクリックしてブレークポイントを設定すると、実行がいったん停止して、変数の内容などを確認できます。

メモ帳アプリの作成

説明が前後してしまいましたが、今回はサンプルとして日付と題名も保存できるメモ帳を作成します。仕様を**表1**に、機能連携図を**図8**に示します。

さて、仕様も決まったので実装していきましょ

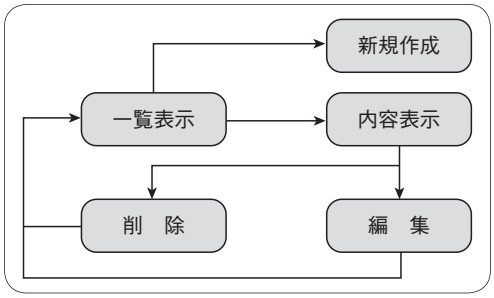
▼図7　設定完了



▼表1　MemoPadアプリの仕様

概要	
日付、題名とともに数行程度のメモを行う。 記入した内容は、アプリ内のデータベースに保存する	
機能	
1. メモ作成	メモを作成する。画面には、日付と題名のテキストボックスと本文記載用のテキストエリアを配置する。保存ボタンは、画面上部と画面下部にわかりやすい形で配置する
2. メモ一覧表示	保存されているメモの題名を一覧表示する。題名をタップした場合は、内容を表示する。メニュー画面も兼ねるので、新規作成ボタンを配置する。新規作成ボタンを押すとメモ作成画面へ遷移する
3. メモ内容表示	一覧から選択されたメモの内容を表示する。編集ボタン、削除ボタンを配置する。ボタンが押された場合にはそれぞれの機能に遷移する
4. メモ編集	選択したメモを再編集する。保存ボタン、破棄ボタンを配置する。保存ボタンは編集内容を上書き保存する。破棄ボタンは編集内容を保存しないで一覧表示に戻る
5. メモ削除	選択したメモを削除する。内容表示の下に削除ボタンとキャンセルボタンを配置する。キャンセルボタンが押された場合は一覧表示に戻る。削除ボタンが押された場合は削除してから一覧表示に戻る

▼図8　機能連携図



う。……と思ったのですが、実際に自動で作ったプロジェクトを動かしてみると機能のほとんどは、すでに実装されている状態でした。

プロジェクトの改造

今のところ、まだ削除機能が実装されていません。フォームで本文を入力する部分も1行入力のままなので、このあたりも直していきましょう。すべての部分で細かな手直しが必要になりますが、裏を返せばこれくらいの変更しかしなくてもよいということです。

■本文入力の領域を広げる

これは、<form>の<input>（タイプ＝text）を<textarea>に変更することで修正します。新規入力部分（new.erb）と編集部分（edit.erb）の2つのerbを編集しましょう。

まず、new.erbの27行目（**リスト1**）を**リスト2**のように書き換えます。

続いて、edit.erbの29行目（**リスト3**）を**リスト4**のように書き換えます。

これでテキストフィールドがテキストエリアに変更され、複数行の入力ができるようになりました。テキストエリアの幅は、デバイスの画面サイズに合わせて変更されます。高さは、入力している行数によってjQueryMobileが自動的に広げています。

▼リスト1　new.erbの27行目ー変更前

```
<input type="text" id="memopad[body]" name="memopad[body]"
    <%= placeholder("Body") %> />
```

▼リスト2　new.erbの27行目ー変更後

```
<textarea id="memopad[body]" name="memopad[body]"
    <%= placeholder("Body") %>></textarea>
```

▼リスト3　edit.erbの29行目ー変更前

```
<input type="text" id="memopad[body]" name="memopad[body]"
    value="<%= @memopad.body %>" <%= placeholder( "Body" ) %> />
```

▼リスト4　edit.erbの29行目ー変更後

```
<textarea id="memopad[body]" name="memopad[body]"
    <%= placeholder("Body") %>> <%= @memopad.body %></textarea>
```

■削除機能を追加する

デフォルトでは編集画面からしか削除ができません。それでは不便なので、削除機能も追加していきます。削除用のボタンに関しては、編集画面のツールバーにあったので、そのコードをコピーして利用しましょう。edit.erbの8～10行目のリンク(**リスト5**)を使います。

内容を確認した後に削除ボタンを押せるようにしたいので、フッタを作ってそこに配置します。show.erbの35行目(最後の の前)に**リスト6**のコードを追加しました。

これで内容を確認してから削除ができるようになりました(**図9**)。

しかし、このままでは削除ボタンを押してすぐにメモが削除されてしまいます。そこで、JavaScriptで確認ダイアログを表示して、本当に削除するのかを確認できるようにしましょう。別のJavaScriptのコードは別途jsファイルに記述してshow.erbのヘッダで読み込むか、show.erbで追加したフッタの後(41行目以降、最後の の前)に**リスト7**のコードを追加します。

また、削除ボタンからJavaScriptの関数を呼び出すように、追加した削除ボタンのコード(リスト6)を**リスト8**のように変更します。このとき、他の場所でも使えるように引数に遷移先のURLと確認用メッセージを取るようにします。これで引数を変更すれば、削除以外でも確認ダイアログを出せるようになります。

▼リスト5　edit.erbのリンク

```
<a href="<%= url_for :action => :delete, :id => @memopad.object %>"
    class="ui-btn-right" data-icon="delete" data-direction="reverse">
    Delete
</a>
```

▼リスト6　show.erbに削除ボタンを追加

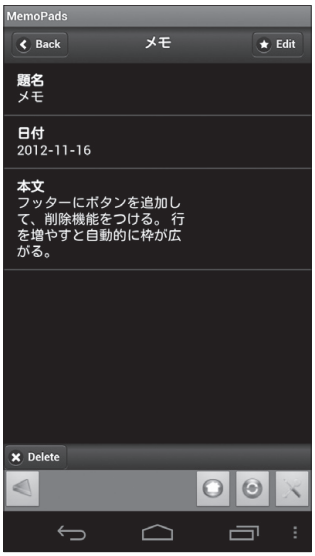
```
<%# 画面下にバーを固定する %>
<div data-role="footer" data-position="fixed">
    <a href="<%= url_for :action => :delete, :id => @memopad.object %>"
        class="ui-btn-left" data-icon="delete" data-direction="reverse">
        Delete
    </a>
</div>
```

▼リスト7　show.erbに追加するJavaScriptのコード

```
<%# 削除の確認をする %>
<script>
function confirmDialog(url,msg) {
    result = confirm(msg);

    // 確認ダイアログでYesを選ぶとtrueが返されるので、ページを遷移させる
    if (result == true) {
        $.mobile.changePage(url, {reverse: true});
        return true;
    }
    return false;
}
</script>
```

▼図9　固定されたフッタに［Delete］ボタンを配置



▼リスト8　show.erbの削除ボタンの変更

```
<%# 画面下にバーを固定する %>
<div data-role="footer" data-position="fixed">
    <a href="javascript:confirmDialog(
             '<%= url_for :action => :delete, :id => @memopad.object %>',
             '削除しますか?');"
        class="ui-btn-left" data-icon="delete" data-direction="reverse">
        Delete
    </a>
</div>
```

■日付と題名の入力を省略する

次に新規作成のときにデフォルト値として今日の日付を入れ、題名が省略されたときにデフォルトの題名を追加するようにしましょう。memo_pad_controller.rb側で変数を設定すると、View側でその値を反映できます。memo_pad_controller.rbのnew関数で@todayという変数に日付を設定します（「@memopad = MemoPad.new」の次の行に追加）。

```
@today    = Date.today().to_s()
```

また、Dateクラスを利用するために、memo_pad_controller.rbの3行目に次のコードを追加してください。

```
require 'date'
```

変数を設定しただけでは表示されないので、new.erbにも反映します（**リスト9**）。HTMLの<form>には、デフォルト値を設定するためのvalueパラメータがあります。ここに@todayの内容を表示するように書き換えます。

次にmemo_pad_controller.rbのcreate関数を**リスト10**のように変更し、内容（body）が空である場合にはデータベースにメモを追加しないようにします。また、題名がない場合はデフォルトで「＜今日の日付＞のメモ」という題名を付けるようにします。

これで、内容がないメモは作成されず、題名がない場合は仮の題名が付けられるようになりました。ベースになる部分に手を加えるだけで、このように使えるアプリができました。ぜひ一度試してみてください。

まとめ

Rhodesでのアプリ開発はいかがでしたか。普段Android SDKを使ってアプリを作っていると、どうしても「めんどうだなー」と思う部分が出てきますが、今回はそういうことが多少減っていたように思います。

RhodesはRubyなのでわからないという方には、JavaScriptを使えるPhoneGapもあります。Javaで作るのもいいですが、スクリプト言語でアプリを作る環境も試してみてください。

2012年7月号の本連載で紹介した、PlayStation Mobile DeveloperProgramが2012年11月20日より正式に開始しました[注2]。年間のライセンス利用料も7,980円と正式に発表されました。Androidデバイスへの対応の拡大も発表されたので、この先もいろいろな展開を期待できそうです。SD

注2）URL https://psm.playstation.net/portal/ja/

▼リスト9　new.erbの変更（21行目）

```
<%# @todayとしてアサインされている変数を表示する %>
<input type="text" id="memopad[date]" name="memopad[date]"
    value="<%= @today %>" <%= placeholder("Date") %> />
```

▼リスト10　memo_pad_controller.rbのcreate関数の修正

```
def create
    if @params['memopad']['body'] && @params['memopad']['body'] != ''
        if @params['memopad']['subject'] == ''
            @params['memopad']['subject'] = "#{Date.today().to_s()}のメモ"
        end
        @memopad = MemoPad.create(@params['memopad'])
    end
    redirect :action => :index
end
```

嶋崎 聡（しまざき さとし）よこいど／日本Androidの会 横浜支部所属

Androidでゲームを作りたいがために長年敬遠してきたJavaを覚えたまではよいのですが、使い始めたときの環境（Android 1.6）では描画とGC（と自分自身のスキル）のおかげで求める速度が出せないまま、いつのまにか当初の目的を忘れて全然違うものばかり作っています。そろそろゲームを……。

地方エンジニアあるある

田中 耕輔
Kosuke Tanaka
レッドハット(株)
グローバルサービス本部プラットフォームソリューション統括部
ソリューションアーキテクト

大阪にもいてますねん

まいど。レッドハットでソリューションアーキテクト(SA)をやっています、田中と申します。え？もうSAの話はお腹いっぱい？……いやいや、ちょっと待ってください。今回は少し趣向を変えて本社・恵比寿から飛び出しまして、大阪から番外編として「なにわ通信」でお送りしたいと思います。どうぞお楽しみください。

レッドハット日本法人の本拠地は、ご存じのとおり東京・恵比寿駅前にあります。何度かの引越しを経験してはいますが、1999年の設立以来ずっと、東京から国内すべての対応を行っていました。そんな中、西日本における活動の拠点として、2009年末に「西日本支社大阪営業所」が開設されたのです！

そんな大阪営業所の守備範囲ですが、基本的に近畿以西となっています。この守備範囲に含まれる、多くのエンドユーザ様、パートナー様を対象に、日々営業活動に励んでいます。肝心の場所ですが、一度の引越しを経て、現在は大阪の堺筋本町駅前にあります。全国から「そんな駅知らねーぞ」という声が聞こえて来そうですね。はい、すいません。梅田とか心斎橋といったメジャーな駅ならばよかったのですが……。「恵比寿」と「堺筋本町」では釣り合いがとれないので、今回は「なにわ」と銘打っております。

大阪まめちしき

大阪の顧客を訪問、あるいは大阪の支店で打合せなどといった理由でたまに大阪へ出張するかも？——という方は多数いらっしゃると思います。そのような方向けに、知っておいて損はない豆知識をご紹介したいと思います。

東西南北を意識せよ！

大阪市中心部の道路は、「碁盤の目」になっています。そのため地図もシンプルで、比較的迷うことなく目的地へ辿りつけます。ただし、東西南北を把握していれば……です。初心者にはこれが意外と難しいのです。「御堂筋線本町駅で降りて東」と伝えたにもかかわらず、ひたすら西へ進んで遭難する出張者は後を絶ちません。市内中心部の地下鉄は、ほとんどが南北か東西に走っています。それを意識して、電車を降りてからも「自分は今どの方角を向いているか」を反芻しつつ移動するのがコツです。

地下街に気をつけろ！

東京には新宿ダンジョンという恐ろしい場所があると伝え聞きます。大阪にも、それに匹敵するスポットがあります。そう、「梅田地下街」です。梅田周辺は、ほぼ全域が地下で繋がっており、信号や天候によらず移動が可能でとても便利です。ただ、多数の中〜小規模な地下街が無秩序に繋がっているため、初心者の目には複雑怪奇に映ります。「B1を歩いていると思ったら、いつの間にかB2にいた。何を(略)」ということがリアルに起きる恐ろしい場所です。私も高3の時、長崎から初めて大阪にやって来たその日に洗礼を受けました。地上では見えていた目的地ですら、地下を通ると辿りつけない……

という恐ろしい体験でした。アドバイスとしては、やはり無理をしないことです。まずは確実に短距離移動で経験値を積み重ねましょう。

キタ派？　ミナミ派？

大阪では「キタ」「ミナミ」という地名を使います。「キタ」といえば梅田周辺、「ミナミ」といえば難波・心斎橋周辺です。多くの百貨店が建ち並び、ちょっと洗練された雰囲気なのがキタ。かたや、コテコテ大阪的な繁華街がミナミです。大阪出張時の宿泊先、あなたはどうしてますか？交通の便を考えて大阪駅前でしょうか。でも大阪を満喫したいという方は、ぜひミナミにトライしてください。「今晩の宿はミナミです」と言うと、「こいつやるな……」的な印象を与えます（※かなり偏見が入ってます）。同様なことが、大阪府北部と南部にもあります。ちょっと上品ぶった北部住民、「河内弁」に代表される粗野な印象を持たれがちな南部住民という構図です。大阪人との会話でネタに困ったら「お住まいは何処ですか？　北？　南？」と振ってみてください。周囲を巻き込んで、意外と盛り上がると思います。その場で南北戦争が勃発しても責任は持ちませんが……。

地方エンジニアあるある

私は、大阪営業所の立ち上げ直後にレッドハットに入社しました。以来ずっと、ただ1人の西日本地域専任のSAとして活動を続けています。レッドハット入社前には、某サーバベンダーの大阪オフィスにおいて、インフラ構築にかかわるエンジニアとして10年ほど勤務していました。そんな私から、本誌の読者の中にも少なからずいらっしゃるであろう「地方エンジニア」の方々と、共感できるようなネタを挙げてみようと思います。

相談したい！

これは誰もが常日頃から感じていることではないかと思います。とくに地方オフィスではエンジニアの数も限られ、気軽に相談できる機会が少ない、場合によっては相談相手自体が存在しない……ということがままあります。もちろん会社ですから、ちゃんと質問すれば誰かしら相談にのってくれるような環境は、どこかにあるとは思います。私の場合は、国内のSA＆コンサル部隊宛のエイリアスに質問メールを投げると、恵比寿にいる超絶優秀エンジニア達が何かしら力になってくれます。でも、そうではないんですよね……。もっと雑談レベルで気軽に、いろいろとテクニカルな情報交換をやりたいと思うのは、エンジニアとしての自然な要求ではないでしょうか。

分担？——何それおいしいの？

これも人が少ない部署の特色だと思います。ある程度の規模の事業所であれば、製品カット、あるいは顧客カット（顧客別、業種別など）で担当を分けるのがよくあるパターンだと思います。いろいろな顧客を担当するのは、個人的にはそれほど苦にはなりませんが、「全部の製品を担当する」というのは、なかなかの覚悟が必要です。全部の製品を片っ端からマスターすることができれば一番ですが、なかなかそんなパワーは持ち合わせていません。必然的に守備位置は「広く浅く」という感じになってしまうのではないでしょうか。私の方針は、ボールが飛んできたら全速力で落下地点に駆けつけ、ボールが落ちてくるまでに急いで最適な捕球姿勢を学ぶ……という感じですね。え、行き当たりばったり？——いえいえ、これぞ地方エンジニアの生きる知恵だと思います。

沖縄の案件って……

大阪営業所の守備範囲は「基本的に近畿以西」と書きましたが、基本から外れる例外があります。そうです、沖縄です。遠隔地への出張は、移動のしんどさもありますが、同時にそれを補う程の楽しみがある場合も多いと思います。そ

の筆頭が「沖縄」「北海道」という響きではないでしょうか(※個人の感想です)。そんな魅力的な沖縄案件というのは、原則東京からの対応になると聞いています……残念。この残念なルールは前職でもありましたので、意外とスタンダードなのかもしれません。というわけで、私はまだ沖縄へ一度も行ったことがありません。旅行したいならプライベートで行け、って話なんですけどね……。

つきまとう不安

ちょっと暗い話になりますが……地方拠点というものは、会社業績の影響をモロに受ける場合があります。その最たるものが「撤退」ではないでしょうか。拠点自体を閉鎖するということもあるでしょうし、特定の部署の人員だけ引き揚げる……なんてこともあるでしょう。人員を本社に集約というパターンもあるでしょうし、ドライな会社であれば速攻クビ宣告なんてこともあり得ます。もちろん失職するのは困りますし、私のように「東京では働きたくないでござる」という人間にとっては転勤だって大問題です。そういう方々は、単なる一エンジニアという感覚を越えて、拠点の営業的な存在意義というものを、常に意識しているのではないでしょうか。

機材がない！

地方オフィスに検証ラボがないというのは、やはり皆の頭を悩ませている点ではないでしょうか。たとえ検証ラボがあったとしても、本社と比較すると激しく見劣りするというのが実状でしょう。私がいる大阪営業所には、もちろん専用の検証ラボルームなんてものはありません。それでも、空いている席を1つ確保(不法占拠？)して、検証スペースとしています。検証用として支給してもらったノートPC3台で、何とかやりくりしようとするのですが、やっぱりそれだけでは足りませんね。あればあるだけ使ってしまう、というのは悪い癖なのかも。

指令「検証環境を確保せよ」

ここまで、地方エンジニアとして気になる事柄をいくつか挙げてみました。ほとんどのものは「頑張れ」または「我慢しろ」という体育会系な解決方法が主となってしまうのですが、機材不足の問題については技術者として何とかしたい所です。ここで少し、いかにして検証環境を充実させるかについて考えてみたいと思います。

困っているという時点で、会社の経費で買ってもらうという選択肢は消えているでしょうから、必然的に自腹で用意することが前提となります。とにかく安く、効率的に入手する必要がありますので、必要最低限の部品を買ってきて組み立てる、あるいは組み換えるというのが第一候補ではないでしょうか。これを「余計な出費」と考えるか「密かな楽しみ」ととらえるかで、モチベーションが大きく違ってきます。私は断然後者だったりするのですが、そうでない方は、ぜひPCのDIYを趣味の1つとして加えてみてはいかがでしょうか。もちろん個人所有機材の持ち込みを禁止している会社もあるでしょうが、そのようなポリシーがやや緩いのも、地方の醍醐味ではないかと思います。

最低限の機材がそろったとしても、検証を必要とする案件や、試したい新製品などは次から次にやってきます。PCが1台増えたからといってあまり状況は改善しません。そんな我々の救世主として現れたのが「仮想化」というテクノロジではないでしょうか。今の時代、ある程度メモリに余裕のあるPCがあれば、Linuxをインストールしてあっというまに仮想化ホストのできあがりです。仮想マシン上に検証環境を構築するようにすれば、検証用の機材不足なんて、あっという間に解決できます！　ただし、OS自体の動作検証や、OS上で動作するアプリケーションについての検証作業であれば……ですが。

仮想化の落とし穴と、抜け道

仮想化によって、検証機材の問題は一気に解決されたかのように思えました。先月号で中井氏が紹介していた環境なんて理想的ですね。しかし、一筋縄では行かない問題が再び浮上します。そう「仮想化環境自体の検証」が必要とされるようになったことです。レッドハットは以前から、Red Hat Enterprise Virtualization（RHEV）という仮想化環境の統合管理製品を販売しています。バージョンも3.1となり、機能的・実績的にも十分こなれた製品となっています。ほかにも、OpenStackなどもあったりして、いろいろな仮想化環境をいじくる必要があります。

これは困りました。RHEVやOpenStackなどの仮想化製品は、複数の物理マシンを仮想化ホストとして統合管理するのが売りのソフトウェアです。検証環境では必然的にたくさんの物理マシンが必要となり、手持ちの検証用仮想化ホストだけでは手も足も出ません。やはり機材を潤沢に用意するしか手はないのでしょうか。

しかし、最近ではこの問題も解決されつつあります！　ご存じの方も多いでしょうが、Nested KVMという、仮想マシンの中でもKVMの機能を使えるようになる機能です。RHEL6では残念ながら使えないのですが、最近のFedoraであれば利用可能になっています。これを使うと、1台のPCの中で、複数のハイパーバイザからなる仮想化の統合環境を構築することが可能になります。

統合仮想化環境を持ち出そう

私も現在は、外出時に愛用しているノートPC（CF-J10）だけで、どこでもRHEVのデモができるような環境を構築しています。ホストOSはFedora 17で、その上の仮想マシンとして仮想化ホスト（RHEV-H）×2、管理サーバ（RHEL 6）×1を動かすような構成です。

折角なので、FedoraのNested KVMを使って、RHEV環境あるいはコミュニティ版のoVirt環境を構築する際の注意点を書いておきます。

まず、Nested KVMの機能ですが、初期状態ではAMD製のCPUでのみ利用可能になっているようです。Intel製のCPUを利用する場合は、

```
echo "options kvm_intel nested=1" > /etc/modprobe.d/nested-kvm.conf
```

などとしたうえで、リブートするか、kvm_intelモジュールをリロードしてください。

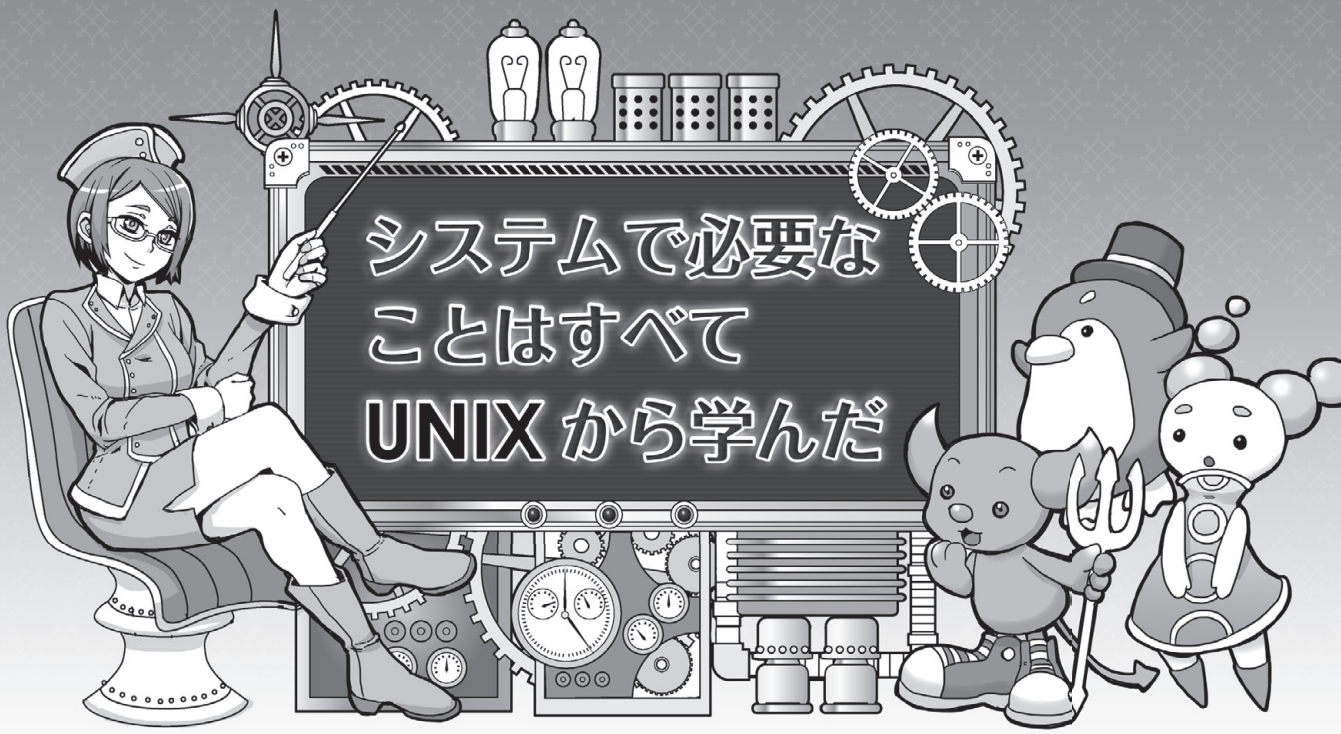
また、仮想マシンを作成する際の注意点ですが、CPUの「モデル」と「トポロジー」を手動で指定しておいてください。仮想マシンマネージャのProcessorの画面にて、「ホストCPUのコピーの設定」ボタンを押すのと、トポロジー設定で「ソケット数」と「コア数」を指定しておけば大丈夫です（前者は仮想マシン内でCPUのvmxやsvmフラグを有効にするため、後者はRHEVマネージャが正しくCPUのコア数を認識できるようにするため、です）。

いかがでしょうか？　あまり無茶な負荷をかけるわけにはいきませんが、いろいろと遊べるだけでなく、多くの場面で活用できる環境ができあがるのではと思います。

さて次回は

今回は趣向を変えて大阪からお送りしましたが、次回からはふたたび筆を恵比寿にお返しします。

これまでの回は私も含めて、OSや仮想化、ストレージといったプラットフォーム製品をおもに扱っているエンジニアが担当してきました。次回からはいよいよ、レッドハットのもう1つの顔、ミドルウェア製品部門のエンジニアが登場です。どうぞご期待ください！ SD



第8回 「コンソール」

サーバやネットワーク機器の管理にはコンソールが不可欠だ。かつてはコンソールといえばRS-232Cポートでのシリアル接続が標準だったが最近は様子が変わってきているようだ。

有限会社エムブイシステムズ　水越 賢治　MIZUKOSHI Kenji　　イラスト：髙野涼香

GUIは特殊なものだった

PCやスマートフォンの普及でGUIを備えたコンピュータ機器は身の回りにたくさんあり、コンピュータの操作はすっかりGUIが当たり前になっている。スマートフォンの普及でタッチUIが当たり前になった。Windows 8の登場以降、ノートPCの画面に触って「あれ、動かない？」とやっている人を家電量販店の店先でも見かける。

しかしコンピュータにGUIが登場するのは研究として1970年代、実用的に使用できるようになったのは1980年代になってからだ。CPUが16bitで、搭載メモリが128KBほど、記憶装置がフロッピーという時代なので、GUIを実現するためのグラフィック表示システムは高価だった。MacintoshはGUIを実現したもののメモリの制限からモノクロ表示だったし、IBM-PCは8色表示だった。

業務用のコンピュータは1台を複数人で使用するTSS（Time Sharing System）が当たり前の時代で、コンピュータにはターミナルという表示だけの機器をつないで使用していた。このためのインターフェースがRS-232Cに代表されるシリアルインターフェースであり、コンピュータの管理にもシリアルインターフェースが使用された。管理用ポートはコンソールポートまたは、マネージメントポートなどと呼ばれて、一般ユーザは使用できないようになっていた。

RS-232C

長い間コンピュータの入出力インターフェースとしてRS-232Cが主力だった。正確にはRS-232Cだけでなく MacintoshはRS-422インターフェースだったしミニコンピュータの周辺機器は20mAカレントループだったが、電気的仕様が異なるだけでコンピュータ的にはほぼ同じようなものだ。なぜRS-232Cが標準的イン

ターフェースとして多用されたのか。それはハードウェアが単純で実装しやすかったことと、コマンドラインでのコンピュータ機器の操作に最も都合が良かったからだ。

RS-232Cに代表されるシリアル通信は非同期通信とも呼ばれる。1bitずつデータをやりとりするためハードウェアが簡単に実現できた。通常8bitつまり1byteずつデータをやりとりするため文字データの通信に便利でCUIを実現するにはうってつけだった。モデムのようなシリアル通信機器も開発され遠距離でデータ通信するためのインターフェースとしても使用された。

GUIにタッチパネルの時代さらにはSiriなど音声認識が携帯電話でも実用的に利用される時代になったが、コンピュータのコマンド入力や文書作成などの文字入力はやはりキーボードでの操作が効率的で、スマートフォンと接続するBluetoothキーボードが多数発売されている。ハードウェアの実現とソフトウェアの実装の両方が容易なため、長い間シリアルインターフェースは主力として使用されてきた。

USB

しかしGUIの発展やインターネットの普及でRS-232Cなどのシリアルインターフェースはその通信速度の遅さが問題となってきた。昔は実用的には32kbps、現在の最高でも230kbpsという速度は文字データを送受信するだけなら良いが、音声や画像をやりとりするには遅すぎる。またハードウェア的にも現代のコンピュータがRS-232Cを扱うのは都合が良くない。古典的シリアルインターフェースは遅すぎて現代のCPUが相手をするには手間がかかりすぎるのだ。CPU側からするとごくたまに1文字ずつデータを送ってきて、すぐに返事を要求するので、頻繁に邪魔されて「うっとうしい」存在といえる。いわゆるフローコントロールの問題だ。

そうした状況を受けIntelとMicrosoftなどがWindowsの時代に合わせたシリアルインターフェースであるUSB(Universal Serial Bus)を策定した。低速なシリアル機器から高速なストレージ機器、キーボードなどの入出力機器など何でも接続でき、フローコントールなどCPU側からコントロールするのが面倒な処理もUSBチップが処理するようにした。USBはパソコンだけでなく家電や携帯電話などにも広く使われることで低価格化して、今ではRS-232Cより実装価格がはるかに安くなっている。

低価格化、PC側がUSBしかないこと、そしてストレージとしても使えることから、ルータやスイッチなどの通信機器の管理コンソールとしてもUSBが普及してきている。USBポートでPCと接続して専用の設定ツールで設定したり、ファームウェアを読み込ませたりできる。ルータなども高度化によってファームウェアのサイズも大きくなってきていて、シリアル通信でアップロードするのは困難だ。tftpなどLAN経由では公開ネット上にある機器のファームウェアアップロードはセキュリティ上の理由からやりたくない。こうした用途にはUSBストレージでのアップデートが効果的だし、設定ファイルやログをUSBストレージに書き出してバックアップすることもできる。ストレージ用にコンパクトフラッシュやSDカードを用意している機種もあったがUSBメモリに一般化されそうだ。

Webインターフェース

ルータやLANスイッチなどのネットワーク機器では従来からLAN経由での設定と管理ができるものが多かった。telnetで接続してコマンドを実行しftpまたはtftpによってファイルのやりとりをするスタイルだ。こうした機器にはもともとLANポートがあるのだからそれを利用すればソフトウェアだけで実現できる。

UNIXサーバではもともと独立した管理用シリアルポートを持っていてこれをコンソールとして利用していたが、10年ほど前からLAN経

由での管理ポートが設けられるようになってきた。サーバ、とくに大型のサーバはサポートベンダによる24時間保守が求められ、障害時の原因追及と早期復旧が求められる。このためにシステム本体とは別にRAS(Remote Access Service)ポート、または管理プロセッサと呼ばれる独立したコンピュータが積まれていて、これを経由して本体の管理運用や障害診断ができるようになっていた。これがLANに対応したのだ。以前は管理プロセッサは16bitマイコンが多かったが最近ではSH32やARMチップなど32bitプロセッサが普通で、OSは組み込みLinuxが増えている。LANもついているので管理プロセッサ自体で仕事ができそうだ。ここにWebサーバ機能も搭載されてきている。Webブラウザでアクセスすれば電源のON/OFFやログの取得、障害時にはどのモジュールが壊れているのかなどがわかる。ブレードサーバや大型サーバでは、アイコンをクリックすれば障害のあるモジュールだけを停止して運用を継続できる。

インターネットの進展でコンピュータ機器はほとんど何らかの形でネットワークインターフェースを持つようになった。ユーザ側もWebに慣れている。そのため機器の管理設定のためにマネージメントプロセッサがWebサーバ機能を持つのは自然の流れだっただろう。

USBシリアルアダプタ

USBインターフェースやWebインターフェースでのシステム管理が主流になってきているとはいえ、まだ従来からのRS-232Cによるコンソールがついている機器もまだある。ルータやスイッチはWebインターフェースとともにまだRS-232Cもほとんどついている。設定管理作業が終わっていない機器をインターネットに接続することは危険なため、初期設定はシリアルコンソールから実行したい。またWebインターフェースではあらかじめ設定ファイルを作っておいてシリアルから「流し込む」こともやりにくい。

こうしたときにはUSBシリアルアダプタをPCに接続してルータのコンソールに接続する必要が出てくる。USBシリアルアダプタは各種販売されていてWindowsには対応しているので、あとはターミナルソフトを用意すれば作業はできる。多くはTeraTermが使われているようだ。ネットワークエンジニアにはまだ必要なツールだ。やはりこういった作業もUNIXで行いたい、あるいは普段使いはMacBookだという人もいるだろう。いくつかのベンダーの製品はLinux対応やMac OS対応をうたっているが、製品は限られる。

実はUSBシリアル通信のICチップを作っているベンダーは2社ほどしかなく回路の設計もバリエーションをつけられる構造にもなっていないので、対応がうたわれていないアダプタでもそのOS用のドライバさえあれば動作する。広く利用されているFTDi社のチップのドライバはFreeBSDやLinuxには標準搭載されているので、**図1**のようにUSBシリアルアダプタを接続すればデバイスが生成される。**図1**はFreeBSD 8.3にFTDiチップ搭載のUSBアダプタを差し込んだときのコンソールメッセージで、USBデバイスとしてuftdi0が、シリアルデバイスとしてttyU0とcuaU0が生成された。あとはtipで対象機器のコンソールに接続すればよい。

FTDiでは各種OS用にドライバを配布しているので、標準で提供されていないOSのドライバも入手して試してみよう[注1]。筆者は2つほどのメーカーのUSBシリアルアダプタを

▼図1　FTDi-USB-Serial

```
# ugen0.2: <RATOC Systems,Inc.> at usbus0
uftdi0: <USB-Serial Converter> on usbus0

# cd /dev
# ls -l *U0*
crw-rw----  1 uucp  dialer    0, 103 Dec  8 00:11 cuaU0
crw-rw----  1 uucp  dialer    0, 104 Dec  8 00:11 cuaU0.init
crw-rw----  1 uucp  dialer    0, 105 Dec  8 00:11 cuaU0.lock
crw-------  1 root  wheel     0, 100 Dec  8 00:11 ttyU0
crw-------  1 root  wheel     0, 101 Dec  8 00:11 ttyU0.init
crw-------  1 root  wheel     0, 102 Dec  8 00:11 ttyU0.lock
#
```

注1） URL FTDi Driver http://www.ftdichip.com/FTDrivers.htm

MacBookに接続して使用しているが、OS標準のcuコマンドで問題なく使用できている。期待どおり動かなくてもUSBシリアルアダプタのメーカーには責任はないので問い合わせしないように。

Webの自動化

最新のブロードバンドルータやストレージ機器、あるいはFireWall製品などは管理インターフェースとしてWebしか備えていない製品も多くなってきた。Webインターフェースしか装備していないのはコストダウンのため以外にも、セキュリティ機器のように設定にはユーザに対して各種情報を提示して、そのうえでの設定が必要な機器もあり、こうした処理にはWeb UIが適しているという事情があるだろう。

人間が操作するという点では、現在ではWebブラウザまたはそれに類するGUIツールを使用するのが一番だろう。文字だけでなく画像やヘルプへのリンクなどを同時に表示できユーザビリティが向上する。すでに多くの人がWeb UIでの操作に慣れているし、設定のためのクライアントツールもPC以外にスマートフォンやタブレット端末など現代的な機器も使用できる。コマンドラインでのコンピュータ機器の操作がすでに一般的でなくなっているので、システム機器の管理にもコマンドラインでの操作だけでなくWeb UIでの操作が求められるのも自然の流れだろう。

人間が操作するにはWeb UIは誠に便利だ。エンジニアだけでなく一般のコンピュータユーザでもWeb UIには慣れているので多くの人が操作できる。その一方で、コンピュータにとってはWeb UIはなかなかに厄介だ。HTMLとFormを解釈し、その規定に沿った形でデータを入出力、その後リターンされた結果を解析する作業が待っている。サーバ側は簡単になるが、クライアント側はその分だけ面倒になる。Webブラウザが使えるPCならまだ良いが、それ以外の機器ではHTMLを求められるレベルで解釈し実行するソフトウェアを用意する必要がある。

Machinize

さまざまなPerlライブラリを開発、公開しているJesse Vincentの作品の1つ「Machinize[注2]」。PerlからWebにアクセスするモジュールでHTMLでのデータ取得だけでなくForm入力にも対応しているので、データの送出にも対応できる。SSLやPROXY経由での通信にも対応。Webでの管理機構しか対応しない機器についてもログを取得したり定期的に変更する設定をコマンドラインから実行できて便利に使える。スクリプト化してcronに登録したりNagiosのコマンド化をすれば監視業務にも使える。

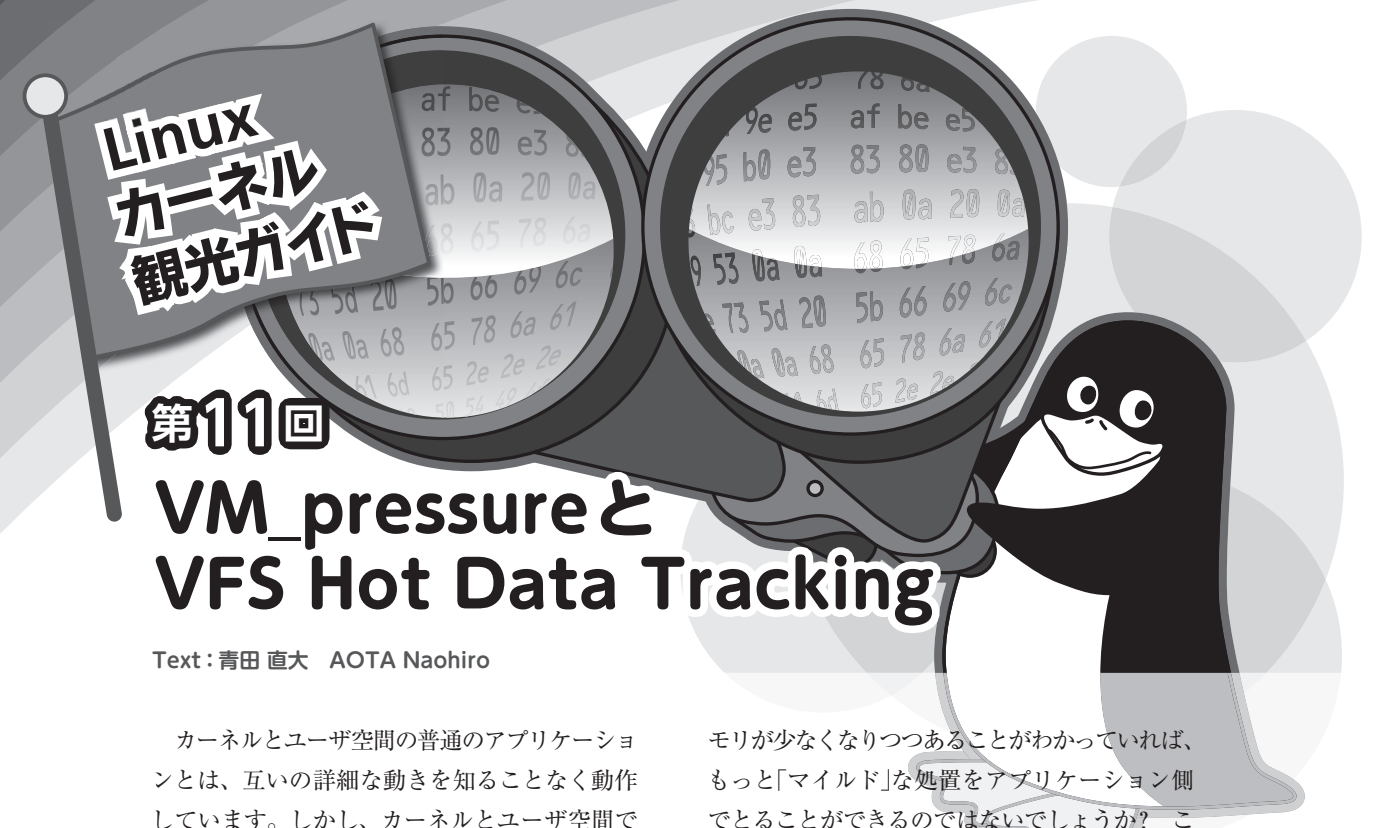
perldocに豊富なサンプルが付いているし、さまざまな使い方のアイデアがWebで紹介されているので参考にしていただきたい。筆者はブロードバンドルータのログを定期的にクリアする用途に使用している。

同様の機能を持ったモジュールがRubyやPHPにも同じ名前であるので、好きな言語で管理機構が作れるだろう。しかしPerl以外のモジュールは作者が異なり、機能も少し異なる。

また、本来の目的はテスト用だが、Webでの入出力をスクリプト化するSelenium[注3]というツールもある。

なお、Machinizeを使用するとWebからの情報取得を自動化して実行できるいわゆる「クローラ」を簡単に実現できるが、クローラはWebサーバ負荷が高く嫌われ者なので、使用には注意すること。自分の管理対象機器やWebサイトで使用する分には楽しく使おう。SD

注2）Machinize（http://search.cpan.org/~jesse/WWW-Mechanize-1.72/）
注3）Selenium（http://seleniumhq.org）



第11回 VM_pressureとVFS Hot Data Tracking

Text：青田 直大　AOTA Naohiro

カーネルとユーザ空間の普通のアプリケーションとは、互いの詳細な動きを知ることなく動作しています。しかし、カーネルとユーザ空間で協調して、より良いパフォーマンス／より良い安定性を目指すためのカーネル機能の開発も提案されています。今回はその中からvmpressure_fd(VM_pressure cgroup)とVFS Hot Data Trackingを紹介します。

VM_pressure

ユーザ空間のアプリケーションは(プロセス間通信を除けば)あたかも自分だけが動いているかのように、システムのメモリを1人で独占して好き勝手に使います。もちろん、これはシステムの物理的なメモリ空間が見えているわけではなく、仮想的なメモリ空間です。

アプリケーションがメモリの新しい領域(ページ)に読み書きする都度、カーネルは実際のメモリとのマッピングを行います。メモリを多く消費するプログラムがいれば、いつかは物理メモリが尽きてしまいます。

メモリが尽きそうになると、実行中のプロセスを強制終了させてメモリを空ける「OOM killer」という動作が実行されます。しかし考えてみると、いきなりどれかのプロセスが終了されるというのはかなり困ったものです。空きメモリが少なくなりつつあることがわかっていれば、もっと「マイルド」な処置をアプリケーション側でとることができるのではないでしょうか？　これが「VM_pressure」が開発された動機です。

それではカーネルのメモリ管理の大雑把な動きを見ながら、VM_pressureがどういうものかを見ていきましょう。freeの出力を見るとわかりますが、システムが「使っている」メモリには大きく分けて3種類のものがあります。

1つはいわゆる「アプリケーションが使用しているメモリ」と「カーネルが動的に確保しているメモリ」です。

残りの2つはカーネルが特別な用途に利用しているメモリである「バッファ」と「キャッシュ」です。「バッファ」はディスクブロックのバッファ用に一時的に確保されるものです。この部分はディスクと同期する(必要であればディスクに書く)ことで破棄できます。「キャッシュ」はディスクから読み込んで変更されていないファイルデータを一時的に保存しているものです。このキャッシュの中に必要なデータがあれば、(メモリより遅い)ディスクを読みに行く必要がなくなるので、パフォーマンスが改善されます。このキャッシュの部分はディスクに書いてあるものと内容が変わっていないので、メモリが不足してくればすぐ破棄しても問題ありません。

メモリの回収

カーネルはメモリが不足してくると、まずは前節で挙げたバッファやキャッシュを解放したり、あるいはアプリケーションが使っているメモリをディスクにswap outすることで必要なメモリを確保しようとします。この処理はmm/vmscan.cのtry_to_free_pages()からはじまります。この関数はscan_control構造体を初期化し、実際の処理を行うdo_try_to_free_pages()を呼び出します。scan_control構造体はメモリ回収のコードのためのパラメータや、そのコードでどれだけのメモリが解放されたかを管理するためのデータ構造です。いろいろな値が設定されますが、ここではnr_to_reclaim(回収するページの数)をSWAP_CLUSTER_MAX(=32)に、priority(回収優先度)をDEF_PRIORITY(=12)に初期化していることを覚えておきましょう。priorityの数値は0〜12の値をとり、小さくなるほど優先度は「上がり」より多くの労力をかけてページの回収を行います。do_try_to_free_pages()は、priorityの値を1つずつ下げながら、nr_to_reclaim個のページの回収に成功するか、priorityが0になるまでページ回収を行うshrink_zones()とshrink_slab()を呼んでいきます。今回はVM_pressureの動きを見るのが目的なので、shrink_slab()の中は追いかけないことにします。

shrink_zones()を追っていくと最終的にLRU(Least Recently Used)リストを操作する部分であるshrink_lruvec()にたどり着きます。よく使われているページをswap outなどで回収しても、またすぐ使うことになってディスクから読み出すことになってしまい、効率はかえって悪くなってしまいます。そこでLRUリストという、最後に使われてからの時間順に並べられたリストを使って回収するべきリストを探しています。Linuxでは、次の5つのLRUリストを使用しています。

- 匿名ページの非アクティブリスト
- 匿名ページのアクティブリスト
- ファイルページの非アクティブリスト
- ファイルページのアクティブリスト
- 回収不可能リスト

ファイルページはディスク上のファイルと関連付けられているページであり、匿名ページはそうでないページでswap outできるものです。アクティブなリストと非アクティブなリストの2つを管理して、よく使用されるページをアクティブリストに、そうでないページを非アクティブに置こうとします。こうしておいて、普段は非アクティブな(あまりアクセスされていない)リストだけをスキャンして、アクティブなリストを無視することで、スキャンの処理を簡素化しています。後述しますが、アクティブなリストはある一定の条件を満たしたときにだけスキャンされています。 shrink_lruvec()では、文字通り回収ができない「回収不能リスト」以外の4つのリストを順にスキャンし、ページを回収しようとします。このときにスキャンするページの数が先ほどのpriorityの値で決定されます。具体的には「(各々のLRUリスト中のページ数)

▼表1　処理のルール

swapできる	参照されている	referencedフラグ	挙動
×	○	○	アクティブ
×	○	×	複数から参照されていればアクティブ
×	×	○	ディスクに書かないなら回収
×	×	×	回収
○	○	−	アクティブ
○	×	−	回収

/(2のpriority乗)」のページをスキャンします。

では、個々のリストがスキャンされる条件とその動きを見てみましょう。

まずは、非アクティブリストです。まず、リストを最後尾からスキャン数のページだけ(あるいはリストが空になるまで)スキャンして、一時的なリストにページを移動します。このリストを基本的に**表1**のルールにしたがって、アクティブへ移動、あるいは回収していきます。このルールを適用する前に、referencedフラグを取得し同時に落とします。このフラグは、今参照されていているもの(厳密には、さらにswapできないもの)には再び立てられます。表1のreferencedフラグはどう解釈したら良いのでしょうか。

referencedフラグは「swapできない」ときにしか使われていないので、そちらに注目して表を見てみます。referencedフラグは、前回のスキャンで参照されていたかどうかを表しています。つまり、前回と今回の両方のスキャンで2回とも参照されていたときにだけ、アクティブに移動する。2回とも参照されていなかったときにだけ、ディスクI/Oを行ってでもページを回収するというようなコードになっています。

たとえば、1時間に1回だけ多くのページにアクセスされるようなコードを書いたとき、その1回だけで多くの(実際にはそんなにアクティブでもない)ページがアクティブとなりページ回収効率が下がってしまいます。これを防止するためにreferencedフラグを使っているというわけです。このほかにも本当にswapできるかどうかを確認したりと細かい調整は多いのですがその部分の説明は割愛します。

▼表2　総メモリ量とratioの関係

総メモリ量	ratio
10MB	1
100MB	1
1GB	3
10GB	10
100GB	31

次は匿名ページのアクティブリストのスキャンです。このリストのスキャンは、

(非アクティブリストのページ数)×ratio＜
(アクティブリストのページ数)

のときにだけ行われます。ratioは非アクティブリストのページ数とアクティブリストのページ数との許容比率となっていて、

$$ratio = \sqrt{10 \times (システムのメモリサイズ(GB単位))}$$

で計算されます。これは**表2**のようになります。

また、ファイルページのアクティブリストは単純にアクティブなファイルページが非アクティブなファイルページよりも多くなったときにリストのスキャンを行います。

アクティブなリストのスキャンでは、参照されている実行可能なファイルページをアクティブなままにしておくという一部例外を除いて、単純に指定された数のページをリストの末尾から取り除き、非アクティブなリストへと移していきます。

vmpressureのフック

vmpressureでは「priorityの値を12から0まで1つずつ下げながらループ」する部分の冒頭と、「priorityで決められる数のページをLRUリストからスキャン」し終わった部分との2ヵ所にコードを追加してメモリが逼迫していることを検出します。

1つ目の「priorityの値を下げながらのループ」の部分では、priorityがvmpressure_level_oom_prio(＝4)以下になっているときに通知を出します。priorityの値がここまで下がっていて、まだメモリの回収が十分にできていない、ということは相当メモリが圧迫されていて、OOM killerがもう間もなく実行されるかもしれない、ということです。もう1つの「LRUリストをスキャン」したあとの部分では、そこでスキャンしたページの数がvmpressure_win(＝512)を超えたとき

に通知を行います。

今のvmpressureは2種類の通知を行うようなコードを提案しています。1つめはもともとから提案されていたもので、スキャンしたページ中の回収されなかったページの割合が、

- 99%以上ならOOM
- 60%以上ならMEDIUM
- それ以外ならLOW

という、3段階に分けられた通知を行います。ユーザランドのアプリケーションは、mempressure cgroupのmempressure.levelを開き、イベント通知に使われるファイルデスクリプタを作成するeventfd()を使い、cgroup.event_controlに“<eventfdのファイルデスクリプタ><mempressure.levelのファイルデスクリプタ><レベル>”を書いてイベント通知を登録します。レベルというのは先に挙げた“oom”、“medium”、“low”のどれかです。アプリケーションはread()またはpoll()などを使ってイベントの到着を待ちます。イベントが来れば、アプリケーションはそれぞれのレベルに合わせてメモリを解放するコードを実行できます。

chunkの概念

もう1つ新しく提案されている通知方法があります。上の通知方法ではイベント通知登録を行ったすべてのアプリケーションに通知を行います。アプリケーション側はどの程度メモリが逼迫しているのかということしかわからないので、場合によっては、1MBのメモリを空けるためにアプリケーションに数十MBのメモリを空けさせているかもしれません。そこで“chunk”という概念を導入します。

chunkはアプリケーションが解放できるメモリ量の1つの単位です。アプリケーションはmempressure.leveの代わりにmempressure.shrinkerのファイルデスクリプタを、“レベル”の代わりに“chunkのサイズ”をcgroup.event_controlに書いてイベント通知を登録します。そして、mempressure.shrinkerに“<eventfdのファイルデスクリプタ><chunkの数>”を書きます。カーネルは、これであるeventfdに通知したときにどの程度のメモリサイズを1単位として、どの程度の個数解放できるかを知ることができました。カーネルはこれを見ながら必要な分だけの通知を行います。

アプリケーションがeventfdをread()すると、通知が来るまでブロックし、解放したいchunkの数がread()の結果として返ってきます。アプリケーションはこの数のchunkを解放します。しかし、ときにはうまく解放できないこともあるでしょう。そのときにはあらためて、“<eventfd><解放できなかったchunk数>”をmempressure.shrinkerに書いておくとカーネルが認識しているchunkの数を増やしてくれます。

具体的な例を見てみましょう。アプリケーションA、B、Cがそれぞれ200MB×1、100MB×3、50MB×10とchunkのサイズと数を登録しているとします。ここで100MBの回収をすると、Bに1つ解放するように通知します。さらに400MBの回収をすると、Aに1つ、Bに2つの解放を通知します。ここでBが1つのchunkの解放に失敗して、1つmempressure.shrinkerに書いていたとします。さらに、200MBの解放を行うとBに1つ、Cに2つの解放を通知します。

このようにvmpressureはシステム全体の空きメモリの減少をアプリケーションに伝えて、より柔軟にそうした状況に対応する／してもらうことができるようになります。しかし、shrinkerのほうは新しく提案されたばかりですし、もともとはvmpressurefdという新しいシステムコール作るといった提案で、cgroupを使ったインターフェースになったのも最近のことです。

まだまだ議論は尽きなさそうですが見ていておもしろい機能ではないでしょうか。

VFS Hot Data Tracking

BtrfsではZFSのように複数のデバイスにまたがるファイルシステムを作ることができます。ZFSはSSDをキャッシュとして利用する機能を備えています。SSDはHDDに比べてI/Oパフォーマンスが優れてはいるものの、容量はまだ大きくはなく値段も高いので、すべてをSSDに移してしまうことは難しいものです。ここで、よくアクセスされるデータをSSDに置くようにすれば、容量の問題を解決しながら、ファイルシステムの全体的なパフォーマンスを改善できるようになります。Linuxでもこれと同じ機能を実装しようと、VFSのレベルでどのデータがよくアクセスされているのかを計測するためのシステムを導入しようとのpatchが提案されています。

VFS Hot data trackingは、readpages()やダイレクトI/Oを行う関数などにフックを導入し、読み書きごとに、それぞれの回数、最後に読み書きされた時刻、「平均読み書き間隔」の6つの値をhot_update_freqs()関数の中で記録しています。これらの値はiノードごと(ファイルごと)にも記録されますが、さらにファイルを1MBずつに区切った領域ごとにも記録しています。これで具体的に1つのファイルのどの部分がよくアクセスされているのかも知ることができます。

ここで本当の平均読み書き間隔を計算することはできないので、現在の「平均読み書き間隔」をavg、今回の読み書きから前回の読み書きの間隔をdelta(単位:ナノ秒)として、

$$\text{newavg} = \frac{1}{16}(15\text{avg} + \text{delta})$$

として、ちょっと変わった加重平均のように計算されています。

温度の計算

さて、これらのデータをもとにしてhot_update_delay(=デフォルトで300秒)ごとに、古いデータの削除と「温度」の測定を行います。古いデータというのは、最後に読み書きのいずれかをしてから、hot_kick_delay(=デフォルトで300秒)経過したデータになります。「温度」というのは、上に挙げた6つの値から計算されるデータのアクセス度合いの基準で大きくなればなるほど、「熱い」ほどよくアクセスされるデータだということになります。これは一見複雑な計算をしているように見えるのですが、実のところ、

$$\text{temp} = \frac{1}{8}(\text{NRR} + \text{NRW} + 2\text{LTR} + 2\text{LTW} + \text{AVR} + \text{AVW})$$

NRR = 読み出し回数 << 20
NRW = 書き込み回数 << 20
LTR = (0xFFFFFFFF − 最後の読み出しからの時間(ナノ秒単位) >> 30)
LTW = (0xFFFFFFFF − 最後の書き込みからの時間(ナノ秒単位) >> 30)
AVR = ((u64)−1 − 平均読み出し時間(ナノ秒単位)) >> 40
AVW = ((u64)−1 − 平均書き込み時間(ナノ秒単位)) >> 40

で計算されています。ただし、LTR、LTWが負になる場合は0にし、AVR、AVWが0xFFFFFFFFより大きくなる場合は0xFFFFFFFFとします。結局NRRなどの値はすべて、読み書き回数が多いほど/最後の読み書きからの時間が短いほど/平均読み書き時間が小さいほどに大きくなる値になっている、というわけです。

この温度計算と同時に、温度によってソートされたリストされたリストも生成し、メンテナンスしています。このリストはメモリが逼迫しているときにメモリを解放するために使われます。計測データ/温度データは読み書きが激しくなってくると、けっこうなメモリを消費するものです。メモリが足りなくなると温度が低いものから順番にデータを削除してメモリを空けるようになっ

ています。

計測／温度データの活用法

さて、ではこれらのデータをどうやってユーザランドから使用するのでしょうか。一般的にはioctl()を使ったインターフェースを使います(**リスト1**)。

このように情報を知りたいファイルを開いて、ioctl(FS_IOC_GET_HEAT_INFO)を使います。liveを1にしていると、ioctl呼び出しで、温度を再計算させて、その値をとります。これが0であれば、上で説明したように定期的に計算している温度をそのままもらいます。

ほかにもdebugfsから情報をとることができます。**図1**のようにhot_spots_inodeからは温度が高いものから低いものへとソートされた情報、rt_stats_inodeからはinodeごとの情報、range_dataからはファイルの範囲ごとの情報がとれるようになっています。

これらの情報をもとにして、よくアクセスされるファイルの配置に最適化をかけることができるようになるわけですね。

まとめ

今回はまだまだ提案段階ながらも、カーネルとユーザランドで協調してのパフォーマンスの改善につながりそうな2つの機能について紹介しました。執筆時点で、すでにリリースされたLinux 3.7には以前に紹介したTCP fast openのサーバサイドの実装が入っています。これでTCPのレイテンシが改善されるわけですが、ユーザランド側でも対応するコードが必要です。今年はそういった対応が進んで新しい機能を楽しめると良いですね。SD

▼リスト1　実際の利用法

```
#include <sys/types.h>
#include <sys/ioctl.h>
#include <sys/stat.h>
#include <fcntl.h>

typedef u_int64_t __u64;
typedef u_int32_t __u32;
typedef u_int8_t __u8;

struct hot_heat_info {
    __u64 avg_delta_reads;
    __u64 avg_delta_writes;
    __u64 last_read_time;
    __u64 last_write_time;
    __u32 num_reads;
    __u32 num_writes;
    __u32 temp;
    __u8 live;
};

#define FS_IOC_GET_HEAT_INFO _IOR('f', 17,
            struct hot_heat_info)

int main()
{
  struct hot_heat_info h;
  int fd = open("target", O_RDONLY);
  h.live = 1;
  ioctl(fd, FS_IOC_GET_HEAT_INFO, &h);
  return 0;
                    ⋮
```

▼図1　debugfs情報

```
$ cat /sys/kernel/debug/hot_track/loop0/hot_spots_inode
inode #5248773, reads 0, writes 244,
avg read time 18446744073709, avg write time 822, temp 111
inode #878523, reads 0, writes 1,
avg read time 18446744073709, avg write time 5278036898, temp 109
inode #878524, reads 0, writes 1,
avg read time 18446744073709, avg write time 5278036898, temp 109
$ cat /sys/kernel/debug/hot_track/sdb/rt_stats_inode
inode #279, reads 0, writes 2, avg read time 18446744073709551615,
avg write time 4923343766042451, temp 109
$ cat /sys/kernel/debug/hot_track/sdb/range_data
inode #279, range start 0 (range len 1048576) reads 0, writes 2,
avg read time 18446744073709551615, avg write time 1058147040842596150, temp 64
```

第34回 Ubuntu Monthly Report

リモートデスクトップの活用

今回はUbuntuで使えるリモートデスクトップサービスを、Androidがクライアントになるものを中心に5つ紹介します。

Ubuntu Japanese Team　あわしろいくや　AWASHIRO Ikuya　ikuya@fruitsbasket.info

リモートデスクトップの重要性

最近リモートデスクトップが2つの理由で注目されているように思います。ここでのリモートデスクトップは、仮想／実機を問わずデスクトップの画面をほかの端末に転送するサーバないしプロトコル（まとめてサービス）とします。

まず第一がデスクトップの仮想化、すなわちVDI（Virtual Desktop Infrastructure）の普及によるものです。今どきはDaaS（Desktop as a Service）でしょうか。ここで、リモートとクライアントでどのような手段で通信するのかはとても重要です。Microsoftが最近のWindows（Server）の新バージョンで着実にリモートデスクトップサービスの機能を強化していることからも、それがうかがえます。

もう1つはAndroidやiOSを搭載したタブレット端末の普及で、これに的を絞ったサービスもいろいろ出てきていておもしろくなってきています。確かに現状すべての仕事をAndroidやiOSで完結するのは難しいので、携帯性と実務上の兼ね合いで重要です。

これら2つは同じサービスを使用することもあれば違うサービスを使用することもあり、ここではその分類で紹介することにします。

Ubuntuのバージョンと動作環境

今回検証したのは、VirtualBoxのゲストOSとしてインストールしたUbuntu 12.04 LTSです。12.10にしなかったのは、3Dアクセラレーションの強制によってパフォーマンスの低下が見られるからです。VirtualBoxでは3Dアクセラレーションを有効にできますが、今回はしていません。もし実験環境で3Dアクセラレーションを有効にしている場合は、無効にしてください。VirtualBoxにもリモートデスクトップサーバ機能がありますが、もちろん今回は使用しません。

VNC（Virtual Network Computing）

VNCサーバはいろいろとありますが、今回はUbuntuにデフォルトで入っているVinoを使用します。Unity Dashに“vino”と入力すると「デスクトップの共有」が出てきますので、これを起動してください。「他のユーザが自分のデスクトップを表示できる」と「他のユーザがデスクトップを操作できる」にチェックを入れると設定完了です。この手軽さがいいです。VNCクライアントはたくさんありますし、今回Androidでは“Jump Desktop Free”[注1]を使用しました。無料版はアカウントが1つしか作れませんが、とりあえず使ってみる分にはこれでいいでしょう。“Automatic Setup”と“Manual Setup”が選択できますが、今回は後者を使用します。“Connection Type”を“VNC”にするのを忘れないでください（**図1**）。

VNCはこのように手軽ですが、Vinoの場合ログイ

注1）https://play.google.com/store/apps/details?id=com.p5sys.android.jump.free&hl=ja

ンしている必要があることと、重いプロトコルなのでせいぜい室内利用が関の山であることなど、欠点も目立ちます。今表示している画面をそのままタブレット端末で見たいという場合にはこれでいいでしょうし、それ以外の場合は別のVNCサーバを探してみるといいかもしれません。あと、相対的にセキュリティが弱いことにも注意です。今回は試していませんが、“Jump”はSSHのトンネリングにも対応しているようなので、これを使ってみるのもいいかもしれません。

RDP

UbuntuでもRDPは使えます。xrdpというパッケージをインストールするのですが、リポジトリにあるものはキーボードの扱いに問題があるので、これを修正したものをPPAで配布していますので、インストールします。方法は次のとおりです。

```
$ sudo add-apt-repository ppa:ikuya-fruitsbasket/xrdp
$ sudo apt-get update
$ sudo apt-get install xrdp
```

インストールが終わったら自動的に起動しますので、今見ている画面をとばすことはできないので[注2]ログアウトしてください。あとはIPアドレスを控えておき、RDPクライアントを起動します。Androidではやはり“Jump Desktop Free”を使用します（**図2**）。

注2）WindowsのRDPでも同じですが。

図1　Jumpでデスクトップを表示（VNC）。右上のディスプレイのアイコンはVinoを使用時に表示される

図2　Jumpでデスクトップを表示（RDP）。転送量を削減するためか、Unity Dashが真っ黒くなっているのが印象的

“Connection Type”を“RDP”にすることと、“Auto Login”の設定を忘れずに行ってください。“Domain”は空欄のままでいいです。あと“Keyboard”も“Japanese”にします。

VNCでも同じですが、日本語入力もできます。残念ながらAndroidのIMEは使用できず、IBusを使用することになりますので、画面右上のキーボードアイコンからAnthyなりMozcなりを起動してください。

リモート専用でローカルでログインしない、という場合にはこれを使用するといいかもしれません。

TeamViewer

TeamViewerはWindows、Mac、Linuxに対応したリモートデスクトップサービスで、もちろんUbuntu用のパッケージもあります。興味深いのはWineで動作していることです。

Linux版をダウンロード[注3]し、インストールします。インストールは、ダブルクリックするとUbuntuソフトウェアセンターが起動するので、これの指示に従って行います。今回はAndroid[注4]用のクライアントを使用するので、それもダウンロード[注5]します。

まずはUbuntuで[TeamViewer7]を起動し、ライセンスの表示などを経て（読み終わったあと×ボタンで消せばいいです）設定画面が表示されるので、まずは“Computers & Contacts”でアカウントを作成し、ログインします。Androidでも「コンピュータ」で作成したアカウントでログインし、「マイコンピュータ」で「オンライン」になっていることを確認し、「リモートコントロール（パスワードを使用）」を選択します。そしてUbuntuに表示されているパスワード（数字4桁のようです）を入力すればログインできます（**図3**）。画質が悪い場合は「セッション設定」（右下の歯車アイコン）をタップし、「画質」を「画質の最適化」にしてください。

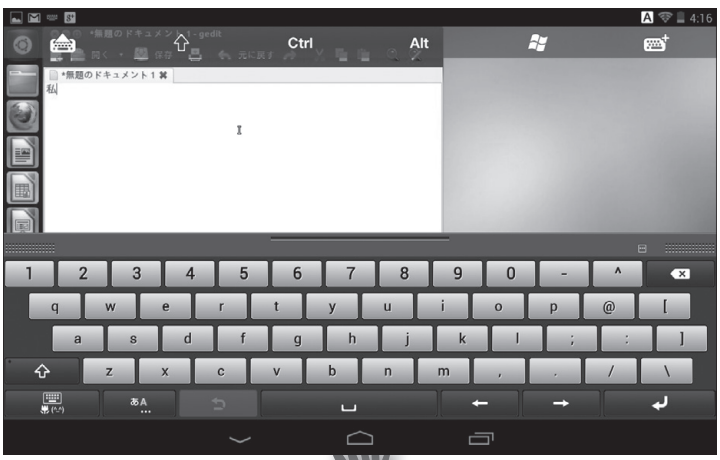
現在表示している画面が転送されるのでとても便利ですが、設定がやや煩雑なのと商用ライセンスが高額（通常価格55,300円）なのがネックです。

Splashtop

Splashtop（企業名）は以前DeviceVMという社名で、Splash OSというすぐに起動するLinuxベースのOSを販売していました。しかし、現在はSplashtopという社名に変更し、メインの事業をこのリモートデスクトップサービスの販売に変更しているようです。

Splashtop（製品名）は無印（1相当）と2があり、Ubuntu用は2しかありませんが、Google Playでは無印と2があります。間違わずに後者をインストール

図3　TeamViewerで日本語入力を行っているところ。日本語は直接入らないので英語キーボードを使用している



注3）http://www.teamviewer.com/ja/download/linux.aspx
注4）ちなみにNexus 7です。
注5）https://play.google.com/store/apps/details?id=com.teamviewer.teamviewer.market.mobile

してください[注6]。ライセンス体系なども大幅に変更されているのでお気を付けください。また、サーバはSplash Streamerという名前です。

さらに注意点として、Splash Streamer バージョン2.0.0.11はVirtualBox上のゲストOSで動作させると画面が著しく乱れてまったく使用できなくなるため、ここでだけVMware Player 5.0.1を使用しています。

まずはSplash Streamerをダウンロード[注7]します。2012年12月上旬現在、12.04用しかないのも12.10にしなかった理由の1つです。TeamViwerとは異なりネイティブのバイナリーなので、ファイルサイズはとても小さいです。ダウンロードの際には、メールアドレスが必要です。これをインストールし、Unity Dashで“splashtop”を検索し、起動しておいてください。まだアカウントがないのでログインできませんし、Ubuntu用のSplashtop 2はリリースされていません[注8]。

アカウントの登録はSplashtop 2（クライアント）で行いますので、Google Playでインストール[注9]し、起動してください。ここでアカウントを作成し、Splashtop Streamerでログインしてください。Splashtop 2に戻るとログインしたUbuntuが出てくると思いますので、これをクリックしてログインしてください（**図4**）。

なお、WANからログインする場合は月額0.99ドルないし年額9.99ドル支払う必要があるうえ、もちろん商用版もありますが、TeamViewerとは違って大規模ユーザ向けなのかもしれません。なぜなら、1ユーザあたりの価格が明示されていないからです。

まだまだ新しいからか、VirtualBoxのゲストOSでは正しく動作しないとかUbuntu用のクライアントがないとか問題もありますが、ネイティブで動作して設定も簡単なので、こなれていくと魅力的になるのではないでしょうか。

X2go

AndroidやiOSで動作するクライアントはありませんが、Ubuntu的にイチオシなのがどうやらX2goのようです。2012年10月29日からコペンハーゲン（デンマーク）で行われたUbuntu Developer Summitで関係者による会合が行わなわれたそうです[注10]。Ubuntu 12.10からはlightdm（ログイン画面）に「リ

注6） 前者は有料ですが2のクライアントとしては使用できません。
注7） http://www.splashtop.com/streamer/linux#download
注8） 開発中ではあるようです。
注9） https://play.google.com/store/apps/details?id=com.splashtop.remote.pad.v2
注10） http://blog.x2go.org/index.php/2012/11/07/co-operation-between-canonical-x2go-edubuntu-and-fleten-net-in-denmark/

図4 Splash 2を使用しているところ。Unity Dashの後ろに透けて見えるのがSplashtop Streamerのログインダイアログ。操作も一番タッチパネルっぽかった

モートログイン」が追加され、Universal Client Configration Service[注11]で登録したデスクトップにログインできる機能がつきましたが、現状プロトコルはRDPのみの対応です。将来的にはここからX2goにログインする機能を実装するなど、興味深いことが話されていたとのことです。

X2goのパッケージはデフォルトのリポジトリにはないので、PPAを追加してからインストールします。まずはサーバのインストールから行います。

```
$ sudo add-apt-repository ppa:x2go/stable
$ sudo apt-get update
$ sudo apt-get install x2goserver
```

クライアントは次のようにインストールします。もちろん別マシン（もちろん仮想マシンでも可）が必要です。今回はUbuntu 12.10にインストールしています。

```
$ sudo apt-add-repository ppa:x2go/stable
$ sudo apt-get update
$ sudo apt-get install x2goclient
```

あとはUnity Dashで“x2goclient”を検索して起動してください。

左上の新規アイコンをクリックして[注12]“Session Preferences”を起動して“Session Name”“Host”“Login”“Sesion type”などを設定します。接続時のクオリティは“Connection”タブで設定します。“Settings”はキーボードを設定するところがあるので、日本語キーボードを使用している場合は“Keyboard layout”を“jp”に、“Keyboard model”を“jp106”にします。“OK”をクリックすると設定が保存されて前の画面に戻り、右上に先ほど作成したSession nameと設定の要約が表示されますので、これの任意の場所をクリックします。するとパスワードが表示されるのでログイン時のパスワードを入力し、ログインしてください（**図5**）。

ただし、問題が2つあります。1つはキーボードがおかしいことで、これは［システム設定］-［キーボードレイアウト］に［日本語］を追加し、これを有効にすると解決します[注13]。もう1つがやっかいで、今のところ有効な解決方法が見つけられていないのですが、IBusが起動しません。x2goserver-run-extensionsによると/usr/lib/x2go/extensions/に/etc/X11/Xsession.d/80im-switchのシンボリックリンクを数字3桁＋アンダースコア（001_80im-switchなど）のルールに従って作成すればログイン時に起動するはずですが、起動しないか、起動してもGUIがいっさい表示されませんでした。ひょっとしたらもっといい方法があるのかもしれませんが、X2goは割に大きくて難しいことをしているので、ベストの方法を見つけることができませんでした。今回はやむを得ずIBusを使用すると仮定して.bashrcに、

```
export XMODIFIERS=@im=ibus
export GTK_IM_MODULE=ibus
```

を追記し、ログイン後に“ibus-daemon -xd”を手動で実行することにしました。

というわけで現状やや扱いにくいですが、ログインにSSHを使うのでセキュアなこと、なんといってもオープンソースであること、そしてUbuntuに統合されて使いやすくなることに期待しましょう。SD

図5　X2goを使用しているところ。キーボードと日本語入力に問題がなければいいのだが……

注11) https://uccs.landscape.canonical.com/

注12) 手元の環境ではなぜかツールチップが表示されませんでした。

注13) ただ、どうしても解決しないこともありましたが、法則性は見つけられませんでした……。初回ログイン時は設定したキーマップを無視するバグでもあるんでしょうか。

バックナンバーのお知らせ　BACKNUMBER

バックナンバー好評発売中！常備取り扱い店もしくはWebより購入いただけます

本誌バックナンバー（紙版）はお近くの書店に注文されるか、下記のバックナンバー常備取り扱い店にてお求めいただけます。また、「Fujisan.co.jp」（http://www.fujisan.co.jp/sd）や、e-hon（http://www.e-hon.ne.jp）にて、Webから注文し、ご自宅やお近くの書店へお届けするサービスもございます。

2013年1月号

第1特集
いざというときに備える
システムバックアップ

第2特集
IT業界のキーパーソンに聞く
2013年に来そうな「技術」・「ビジョン」はこれだ!

特別付録
法輪寺電電宮情報安全護符シール

1,380円

2012年12月号

第1特集
判断をあおぐ／経緯を説明する／手順の理解を得る
文章を書くためのアタマの整理術
なぜエンジニアは文章が下手なのか?

第2特集
高速・高機能HTTPサーバ
Nginx構築・設定マニュアル

一般記事
・エブリデープログラマの発想と実践

1,280円

2012年11月号

第1特集
もし、OpenFlowでやれと言われたら?
SDN、仮想化でネットワークはどうなる

第2特集
サーバの運用支援に
グラフィカルなリソース監視ツールを!
Muninが手放せない理由

一般記事
・SkeedSilverBulletとは?

1,280円

2012年10月号

第1特集
サーバ管理自動化の恩恵とリスクを見直しませんか?
Chef入門

第2特集
lprコマンドが動く裏側のしくみがわかる!
Linuxプリント環境の教科書

一般記事
・SSH力をつけよう!
・JSX入門［前編］

1,280円

2012年9月号

第1特集
理解の壁を乗り越えるFinal Answer!
C言語のポインタは必要ですか?

第2特集
セキュリティ向上をあきらめない管理者になる!
SELinuxを無効にしない理由

一般記事
・機械学習ライブラリ『Mahout』入門［後編］

1,280円

2012年8月号

第1特集
いま読んでおくべき本はどれだ?
エンジニアのパワーアップ読書

第2特集
いま改めてお勧めするOSはこれだ!
FreeBSD、Debian、Ubuntu、CentOS、Gentoo

一般記事
・機械学習ライブラリ『Mahout』入門［前編］

1,280円

Software Design　バックナンバー常備取り扱い店							
北海道	札幌市中央区	MARUZEN&ジュンク堂書店　札幌店	011-223-1911	神奈川県	川崎市高津区	文教堂書店　溝の口本店	044-812-0063
	札幌市中央区	紀伊國屋書店　札幌本店	011-231-2131	静岡県	静岡市葵区	戸田書店　静岡本店	054-205-6111
東京都	豊島区	ジュンク堂書店　池袋本店	03-5956-6111	愛知県	名古屋市中区	三洋堂書店　上前津店	052-251-8334
	新宿区	紀伊國屋書店　新宿本店	03-3354-0131	大阪府	大阪市北区	ジュンク堂書店　大阪本店	06-4799-1090
	渋谷区	紀伊國屋書店　新宿南店	03-5361-3315	兵庫県	神戸市中央区	ジュンク堂書店　三宮店	078-392-1001
	千代田区	書泉ブックタワー	03-5296-0051	広島県	広島市南区	ジュンク堂書店　広島駅前店	082-568-3000
	千代田区	丸善　丸の内本店	03-5288-8881		広島市中区	丸善　広島店	082-504-6210
	中央区	八重洲ブックセンター本店	03-3281-1811	福岡県	福岡市中央区	ジュンク堂書店　福岡店	092-738-3322
	渋谷区	MARUZEN &ジュンク堂書店　渋谷店	03-5456-2111				

※店舗によってバックナンバー取り扱い期間などが異なります。在庫などは各書店にご確認ください。

デジタル版のお知らせ　DIGITAL

デジタル版Software Design 好評発売中！紙版で在庫切れのバックナンバーも購入できます

本誌デジタル版は「Fujisan.co.jp」（http://www.fujisan.co.jp/sd）と、「雑誌オンライン.com」（http://www.zasshi-online.com/）で購入できます。最新号、バックナンバーとも1冊のみの購入はもちろん、定期購読も対応。1年間定期購読なら5%割引になります。デジタル版はPCのほかにiPad／iPhoneにも対応し、購入するとどちらでも追加料金を払うことなく、読むことができます。

February 2013

NO.16

Monthly News from jus

Japan UNIX Society

日本UNIXユーザ会　http://www.jus.or.jp/
法林浩之　HOURIN Hiroyuki　hourin@suplex.gr.jp

大阪秋の陣　—KOF2012—

今回は、毎年秋の恒例行事となっている「関西オープンソース2012＋関西コミュニティ大決戦」(KOF2012)と、その中で行ったjus研究会の模様を報告します。

KOF2012

■関西オープンソース2012
**　＋関西コミュニティ大決戦**
【日時】2012年11月9日(金)13:00～18:00
**　　2012年11月10日(土)10:00～18:00**
【会場】大阪南港ATC ITM棟 9階、10階

KOFは関西からオープンソースやITコミュニティ活動を発信する場として行われているもので、今年で11回目になります。jusは初年度から共催団体の1つとして参加しています。今年も多くの団体の協力を得て盛大な行事となり、2日間合計で約1500人が会場を訪れました。以下、実施されたプログラムの一部を紹介します。詳しくはKOFのWebサイト(**http://2012.k-of.jp/**)を参照してください。

■基調講演とユーザ企画

実行委員会で講師を招待して実施する基調講演は、「スマホ、ビッグデータ」(岡村久道)、「もしも普通のエンジニアが起業したら?!」(藤川真一)など4件を行いました。一方、参加団体が自主的に企画しセミナーなどを実施するのがユーザ企画です。「2つのAndroid端末でリモートコントロール！」(データ変換研究所)、「はじめてのconcrete5でアドオン開発！」(concrete5関西ユーザーグループ)など38件が行われました。jusもjus研究会大阪大会を行いました。

■展示企画／ステージ企画／ジュンク堂

展示企画には、ぷらっとホーム、お名前.comレンタルサーバー、フォーディー・ジャパン、灘校パソコン研究部、ホビーロボット研究会など49組織が出展しました。出展団体は、企業、学校、NPO、コミュニティ、個人などバラエティに富んでおり、それらが組織の規模に関係なく1組織あたり机1個で展示を行いました。また、展示会場の片隅にはステージを設け、「地域活性化にITを」(和田和子)、「KDDIウェブ公式キャラ、雲野コア誕生までとプロモーション効果分析」(阿部正幸)など23件のセッションを実施しました。加えてジュンク堂書店KOF店では参加団体による推薦書籍の紹介や著者サイン会も行いました。

■ウォーキングツアー／懇親会

KOFでは実行委員の引率で場内を見学するウォーキングツアーを実施しています。今年は金曜日2回、土曜日3回の計5回実施し、各回10名前後の参加がありました。そしてKOFの目玉の1つと言えるのが懇親会で、今年も139人が参加しました。会場が食べ放題の店ですので争わなくても安心してたくさん食べられるだけでなく、ビアスポンサー(たけおかラボ、キイレジャパン、うえだうえおうぇあ)によるベルギービールや、大関の協賛に

よる日本酒も好評でした。

■終わりに

同じイベントを10年以上もやっていると、参加団体や客層、さらに運営体制や利用するツールも変わっていきます。今回のKOFでもDrupalでのWebサイト構築やPeaTiXを用いた懇親会受付などを試みました。これからも時代の流れに対応し、このイベントを長く続けるべく、試行錯誤を繰り返していきたいと思います。来年のKOFは2013年11月8日（金）、9日（土）、場所は同じく大阪南港ATCにて開催の予定です。

jus研究会大阪大会

■Appleと過ごした四半世紀
【講師】魚井宏高
（大阪電気通信大学／ワンボタンの声）
【司会】法林浩之（日本UNIXユーザ会）
【日時】2012年11月10日（土）14:00～14:50
【会場】大阪南港ATC ITM棟 9階セミナールーム3

KOF2012の中でjus研究会を行いました。講師の魚井さんは、大学においてはデジタルゲーム学科の先生であり、また25年以上に渡ってMacおよびアップル製品を使い続けている生粋のアップルユーザでもあります。今回はそのアップルとの付き合いを中心に講演していただきました。参加者は27人でした。

■アップル製品を布教する日々

魚井さんがアップルと出会ったのは学生時代。まだMac登場以前で、Apple IIなどを触っていました。そこへ1984年にMacintoshが発売され、近くの研究室に置いてあったものを使っていたのですが、どうしても自分で欲しくなり、1986年にMacPlus漢字Talkを購入しました。以来、周囲にMacの良さを教えるとともに、ユーザグループを作って活動を行ってきました。主な活動履歴としては、MacPressでの執筆、Kinki UserGroup Conferenceの開催、スティーブ・ジョブズさん来阪（1992年）、雑誌連載「魚井センセが教えたる！」、MacFan EXPO、iWeek、Apple User Groups Summer Tourなどがあります。

■デジタルゲーム業界に優秀な人材を

その一方で、2003年に開設されたデジタルゲーム学科の組織作りにも奔走しました。学生の教育用PCとしてMacBookを持たせたり、設立当初（つまり10年前）から学科の建物全体を無線LAN対応にするなど、先進的な取り組みを行ってきました。また、カリキュラムは6つの科目群から2つを選んで学ぶユニット選択方式が採用されています。これはゲームを作るにはハードウェアやプログラミングだけでなく、画面やインターフェースデザインの技術、チームによる製作手法、さらにはゲームのシステムや世界観、キャラクタの創作能力など幅広い知識が必要であり、それに対応した人材を育成することを狙ったものです。

■Macユーザとしての活動が大きな財産に

最後に最近の取り組みとして、アップル関連のニュースを解説するポッドキャスト番組「ワンボタンの声」の紹介がありました。2007年の開始以来、これまでに750回以上もの配信を行っており、魚井さんもレギュラー出演しています。

◇　◇　◇

まとめとして、最初はアップルやMacに惚れ込んで使うようになったのが、やがてMacユーザとして生きていく中でさまざまな人と出会い、それが財産になっていったこと、とくに大阪という地域ではそれが重要だったという回顧がありました。そして、「みなさんもぜひユーザグループに参加してください」というコメントで講演を締めくくりました。最後は時間が足りなくなってしまい、一部を割愛せざるを得なかったのが残念ですが、中身の濃い講演でした。SD

Hack For Japan
エンジニアだからこそできる復興への一歩

Hack For Japan

第14回 福島のITエンジニアと復興を支援する「エフスタ!!」の活動

“東日本大震災に対し、自分たちの開発スキルを役立てたい”というエンジニアの声をもとに発足された「Hack For Japan」。本コミュニティによるアイデアソンやハッカソンといった活動で集められたIT業界の有志たちによる知恵の数々を紹介します。

● Hack For Japanスタッフ
鎌田 篤慎 KAMATA Shigenori
Twitter @4niruddha

我々、Hack For Japanではこれまでの復興支援にかかわる活動から、東北地方で頑張るさまざまなITエンジニアたちと出会い、その人たちが活動するコミュニティともつながっています。今回はその中から、2009年から活動をはじめた、福島県でひときわ元気のあるITエンジニアのためのスキルアップ応援コミュニティ「エフスタ!!注1」をご紹介します。地方都市での勉強会やイベント運営のヒント、原発問題を抱える福島の現状について、福島の未来を願う彼らの活動と共にお伝えしたいと思います。

「エフスタ!!」との出会い

「エフスタ!!」との出会いはまだ震災の傷跡もなまなましかった2011年7月、Hack For Japanメンバーで仙台在住の小泉勝志郎（@koi_zoom1）さんによる塩釜市浦戸諸島の視察注2で仙台に訪れた際、前日に開催されていたデブサミ東北注3に参加したときのことでした。

セッションの1つにあったITコミュニティによるライトニングトーク（LT）大会で、東北を拠点に活動するコミュニティが行うLTの中でも一番元気があり、それでいて原発事故の影響を受けている福島の現状を切実に訴えるその姿勢、LTの随所に伝わってくる「福島が本当に好きだ」という気持ち、そして猪苗代湖ズの「I love you & I need youふくしま」をメンバー全員で合唱するインパクトは今でも強く印象に残っています。

ちょうどその頃のHack For Japanは、福島県下でのITコミュニティとのつながりが会津地方の方たちを中心としたものとなっていました。福島県は広く、大きく分けて「会津」「中通り」「浜通り」と3つの地方があります。それぞれの地方で気候も大きく違いますし、歴史的に見るともともと異なる土地ということも影響しているのか、同県内のITエンジニアたちの交流自体も進んでいないように感じていました。Hack For Japanとしては会津以外の地方で活動的なコミュニティの方たちともつながりを持つことで、福島県内の3つの地方で効果的に連携をとり、ITによる復興支援活動が拡げられればと考えていたのです。

そこで、このLTが終わった直後に、活動拠点を中通り地方の郡山とするエフスタ!!さんにお声がけさせていただき、それがきっかけで今でも交流が続いています。

「エフスタ!!」とは

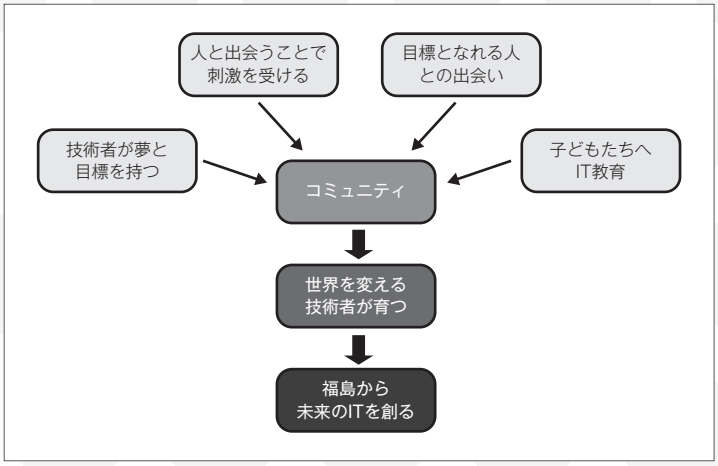
エフスタ!!誕生の経緯や由来について、代表をつとめる大久保仁注4さんに伺いました。もともとITが好きな大久保さんは、“楽しみを持つ人はそうでない人に比べて20倍も幸せを感じる”という説を知り、ITを楽しんで仕事ができるよう自身が勤めている会社を変えたい、夢と希望を持った技術者を育てたい、そのために教育に力を注ぎたいと考えたそうです。しかし、会社のしくみを中から変えるには時間がかかってしまうという思いから、その活動に加

注1）URL http://efsta.com/
注2）URL http://blog.hack4.jp/2011/07/hack-for-japan_13.html
注3）URL http://codezine.jp/devsumi/2011/tohoku

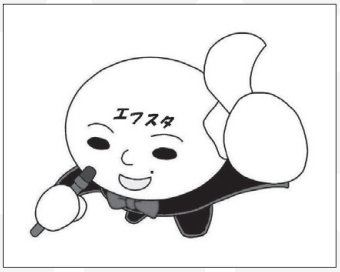
注4）URL http://el.jibun.atmarkit.co.jp/jin/

第14回 福島のITエンジニアと復興を支援する「エフスタ!!」の活動

◆図1 「エフスタ!!」活動理念



◆図2 エフスタ君



えてさらに視野を広げ、「福島のITが変われば、おもしろくなれば、夢と希望を持った技術者が増える、そして世界を変える技術者がやがて誕生する」という発想から、そうした理念を実現するためのきっかけ作りの場としてエフスタ!!を立ち上げるに至ったとのことです(**図1**)。

また、エフスタ!!の語源は「福島のスタイルを変える」から「エフスタイル」に略され、語呂の良さから今の「エフスタ!!」に落ち着いたそうです。これが今ではエフスタ君といったキャラクターまで誕生し愛されています(**図2**)。

地方のITコミュニティが抱える課題

地方のITコミュニティが勉強会やイベントを行う際に、必ずと言っていいほど直面する課題があります。それは「集客」です。これは平日の夜や週末ごとに勉強会やイベントが開催され、参加したいものが重なって両方に参加したいのに参加できない、といった悩みを抱える都内に在住するエンジニアの方たちにはイメージが沸かないかもしれません。

地方都市ではITエンジニアの数も非常に少なく、興味を持つテーマも人を集客するという観点から見ると限られてしまい、勉強会やイベントに人が集まりません。また、勉強会やイベントをやったとしても参加する人が少なければ長くは続かず、さらに勉強会やイベントが開催されなくなるという悪循環に陥りがちです。これはインターネットなどを利用することで空間に縛られずに仕事が可能であるとされるITエンジニアにおいても、いまだにエンジニア人口が首都圏に集中しているからだとも言えます。

そうしたこともあって、地方都市ではイベントを開催する際に著名な講演者を招き、インターネット上で宣伝、告知していたとしても、都内では考えられないほど人が集まらないことがあります。

しかし、エフスタ!!では福島県の郡山という地方であっても、他の地方都市で開催される勉強会では考えられないほどの参加者が集まります(**写真1**)。筆者もHack For Japanスタッフの西脇資哲(@waki)さんと共に、Hack For Japanの活動内容の紹

◆写真1 受付に並ぶ参加者たち

介と西脇さんによる「プレゼンテーション・デモンストレーションスキルアップ〜エバンジェリスト養成講座〜」という講演で、2011年9月に初めてエフスタ!!の勉強会に参加したところ、会場は満員で、参加者も和やかながらも積極的に参加していました。筆者は仕事柄、地方都市での開発者向けイベントにかかわることがよくあります。その経験からして、1コミュニティが主催しているイベントで教室一杯に人が集まり、参加者の意欲、意識も高く、それが自然に運営されているという例は非常に少ないため、そのことに衝撃を受けたのが今でも記憶に残っています。

「エフスタ!!」の集客力

なぜだろうかと不思議に思っていたこの秘密を、大久保さんから伺うことができました。

通常、IT関連の勉強会やイベントを開催しようとしたとき、まず最初に行うのはブログやTwitter、Facebookなどからの告知、IT勉強会カレンダーなどへの登録だと思います。しかし、こうした活動は先ほど挙げた悪循環により、地元の勉強会をチェックするという習慣がそもそもない、あるいはもはやなくなってしまっている可能性が高いという点で、勉強会やイベントの認知度を上げるには向いていないのです。

そこで大久保さんらエフスタ!!のメンバーは地道にIT企業や学校を訪問し、コミュニティの理念と活動を伝え、徐々に勉強会の参加者を増やしていったそうです。それも普通のIT企業や学校は土曜日、日曜日は基本的に休みであるため、メンバー自身の有給休暇を利用して訪問していくといった形で、です。これは誰もができる行動ではありませんが、福島を変えたい、夢と希望を持った技術者を増やしたい、といった思いが通常では考えられないほどの集客数と熱意ある参加者が集まるコミュニティを作ったのだと思います。

こうしたエフスタ!!の活動は地方都市で勉強会やイベントを開催するコミュニティの方にも必ずヒントになるだろうと思っています。

そうして熱意のある福島のエンジニアたちが集い、2009年の立ち上げから軌道に乗って成長してきたエフスタ!!でしたが、2011年3月11日以降、福島を取り巻く環境は他の被災地とはまったく別の大きな問題を抱えたことで、エフスタ!!の活動にも大きな影響を与えたのでした。

「エフスタ!!」とHack For Japan

福島に夢と希望を持った技術者を増やすという目的が、震災後の原発の問題も重なったことで福島の現状を訴えるという使命を帯びたものとなりました。いま、福島県下のいたる所に放射線のモニタリングポストが設置されています。ガイガーカウンタを所持している福島県民の方も少なくありませんし、レンタルビデオ店でガイガーカウンタのレンタルを行っていたり、小学校の運動会も体育館で行っています。

福島では日本の少子化問題に加えて、このような原発の問題により福島の未来を担う子どもたちの人口が急激に減少し、2040年には現在の4割もの子どもたちが福島からいなくなってしまうという予測がたっているそうです。もちろん放射線の影響を強く受けてしまう子どもたちの未来を考えれば仕方のないことなのですが、福島の未来を変えよう、夢と希望を持つエンジニアを増やそうという理念のもとに活動してきたエフスタ!!からすると本当に悲しい現実です。

しかし、常に放射線を意識せざるを得ない現実からも目をそらさず、福島を愛し「震災前の福島を取り戻す」という気持ちで、エフスタ!!は以前からの勉強会に加えて福島の現状を啓蒙する活動を実施しています。そうした中で、冒頭に書いたようにHack For Japanともつながるのですが、エフスタ!!メンバーの影山哲也さんは、震災後も変わらず継続していた運営の中で、福島県外の人が語る「福島の現状」が現実の福島の現状と乖離（かいり）している点に違和感を感じていました。「これは実際の福島がまだまだ県外の人たちに伝わっていないのではないか？メディアでの原発に関する報道が減少しているからなのではないか」と。この違和感を少しでも解決の方向に向けようと、今回の初のエフスタ!!東京開催

へとつながったのです。

エフスタ!!TOKYO開催

2012年12月8日に、都内にて「エフスタ!!勉強会Vol.11 IN TOKYO[注5]」が開催されました（**写真2**）。Hack For Japanからは及川卓也（@takoratta）さんによる「見る前に跳べ～ギークの工夫で社会を変えよう～2012年冬」と題して「Developers Summit 2012」で発表された内容のアップデート版での発表が行われました。前述の西脇さんのときのように、エフスタ!!の勉強会では毎回IT業界のプロフェッショナルを招いて講演をしてもらうことで、エフスタ!!に集まるエンジニアたちのスキル、マインドの底上げを狙っています。

エフスタ!!に参加されたことのない方から見ると、ITプロフェッショナルの講演や福島の話となると非常にかたい勉強会という印象を抱かれるかもしれません。これは実際に参加していただくとわかりますが、エフスタ!!の勉強会は非常にアットホームな雰囲気となっています。毎回、福島名物のおやつが配られるおやつタイムもあり、またパネルディスカッションでゲストと参加者を交えておもしろおかしいトークを織り交ぜることで、参加者のみなさんは真剣に聴講することと、会を楽しんでリラックスして参加することの両方ができているのです。

参加者のスキル、マインドを高め、リラックスした後に福島の現状を語る場も合わせて用意することで、参加者に福島にも興味を持ってもらう。初の東京開催となった今回は、エフスタ!!立ち上げメンバーでもある大久保さんと影山さんのお2人の講演で、エフスタ!!の設立した経緯や福島で出会った熱いエンジニアたちの話、そして福島の現状を伝えるセッションでは都内からの参加者も多かった今回に合わせ、東京開催に参加された方たちに期待するところなどをお話しいただきました。

また、福島の現状を伝える福島からの参加者らによるLT大会では、福島各地での線量計の数値を測定して歩いた様子を伝えるビデオや、厳格な食品線量検査の結果、安心して食べられる福島の物産の紹介、エフスタ!!代表の大久保さんへの感謝の手紙など、多種多様な発表がされましたが、そのどれもが福島を愛する気持ちで一杯のものでした。

そして最後にエフスタ!!メンバー全員が前にそろって、メンバーの1人である本多裕幸さんがiPadで演奏する「I love you & I need youふくしま」に合わせて合唱する傍ら、大久保さんによる「震災前の福島を取り戻したい」という気持ちの込められた熱いLTで、初のエフスタ!!東京開催は締めくくられたのでした。

これから長い時間をかけて復興していく福島を皆さんもどうぞ応援ください。SD

◆写真2　エフスタ!!勉強会の様子

注5）URL http://kokucheese.com/event/index/60320

Text=Bart Eisenberg　E-mail jaysteller@hotmail.com　　Translation=嶋崎正樹 SHIMAZAKI Masaki

コンピュータサイエンス2.0［Part 4］：生涯学び続けることが仕事の一部

Chris Timossi

元ソフトウェアエンジニア、ローレンス・バークレー国立研究所

学ぶ立場からコンピュータサイエンスを考える

Chris Timossiが35年に及ぶキャリアの大半を過ごした職場は、カリフォルニア大学バークレー校を臨む丘に設置された粒子加速器である。ローレンス・バークレー国立研究所（LBL）内にあり、ALS（Advanced Light Source）と名付けられたその施設は、世界最大の高等教育科学プロジェクトのようにも見える。巨大な円形のシンクロトロン[注1]にはアルミホイルでくるまれたパイプがつながれていて、パイプの内部では75億個もの電子が1秒あたり150万回転の速度で駆けめぐり、太陽の10億倍も明るい光を放出している。この放射光は、地球科学、生物学、化学、材料科学など、驚くほど広範な学問分野の研究に利用されている。Timossiが大学院生として研究していた物理学ももちろんその1つだ。

2011年6月に退職するまで、Timossiはシンクロトロンの制御システムの設計と保守に携わっていた。ミニコンピュータの黎明期に就職し、インターネットや携帯、マイクロコントローラの時代に引退した形だ。最新技術に遅れずについていけるように、Timossiは数えきれないほどの講座、ワークショップ、セミナーなどに参加してきた。今なお研究所の「アフィリエイト会員」としてそうした講座を受講する資格を持ち、実際に受講している。コンピュータ教育に関するこのシリーズでは、教える立場の3人にインタビューしてきたが、今回は生涯にわたって学び続けてきた人物の話を聞こう。

私がChrisと出会ったのは1976年だった。ちょうど彼がLBLで最初の業務に就いた年だ。彼はそれ以来の友人であり、バックパッキングのパートナー、技術関係の相談役でもある。Chrisのおかげで私はある日本の技術者と知り合いになり、そのまた知り合いだった雑誌編集者が技術関連の記事を書ける米国人を探していた縁で、二十数年前に本誌の記事を担当することになった。Chrisを紹介する記事を書いていると、ちょうど回転する電子のように、歴史がまさに一巡した感がある。私はこの連載で40人以上のインタビューを行ってきたが、今回のインタビューを締めくくりにしたい。新しい生活――今ほど締め切りに追われない生活を始めようと思っている。ただ次回もう一度だけ登板させていただき、これまでを振り返ってみよう。

→ スタート地点は粒子加速器のオペレータ

―最初はコンピュータ処理をどのようなものだと考えていましたか？

注1）磁場を強めると同時に加速周波数を変えて軌道半径を一定に保つ方式の加速器のこと。

私がカリフォルニア大学サンタバーバラ校で修士課程にいたころ、物理学部はちょうどコンピュータを導入し始めたところでした。DECのPDP-8があったと思います。物理学の教授の1人がプログラミングに夢中になっていました。それで私も関わるようになったのです。私が興味を持ったのは研究用の機器です。デジタル回路のプロジェクトを手がけていたころ、周囲ではハードウェアとソフトウェアのインターフェースに取り組んでいました。カラーテレビを接続し、いわば最初期のカラーディスプレイを作っていたりしていましたね。Intelが初期のマイクロプロセッサを発表したばかりのころです。当時はまだその程度の段階でした。

私はローレンス・バークレー国立研究所に粒子加速器のオペレータとして就職しました。就職できて大喜びでした。なにしろ巨大な科学研究所ですから、いろいろなチャンスがありそうに思えたし、実際にそうでした。研究所ではミニコンが標準装備されていて、その他にもCDC（Control Data Corporation）社の巨大なマシンが何台かありました。私が興味を持っていたのは各種制御を行うアプリケーションで、当時としてもかなり特殊でした。誰にとっても変化の時代でしたね。オペレータの多くは古い物理的なボタンに慣れていました。粒子加速器の調整項目ごとにあったボタンを、キーボードで演奏するみたいに扱っていました。そのような操作をすべてコンピュータのディスプレイで行うというのはかなり斬新なことだったのです。

―どのようなものを使っていましたか？

当時研究所では、おもにFORTRANを使っていました。入力はカードやテープで行っていました。また、OSはまだハードウェアごとに違いました。

なぜ技術に遅れずについてこられたか

―それから35年を経て、すっかり様変わりした時代に研究所を退職したわけですが、今までどのように続けてこられたのですか？

研究所が大学と連携していたことが本当にありがたかったです。職場はメインキャンパスから丘に登ったところにあります。働きながら履修できる制度があり、コンピュータサイエンスの学部生と同じ講座を受講できたのです。そんなわけで、粒子加速器のオペレータとして雇われた私は、まず最初に関連する講座を履修しました。当時授業ではPascalを使っていましたが、OSの講座ではCを使っていました。その2つが最初に学んだ言語です。

その後、私はマイクロプロセッサをオンボードで使う別のグループに配属されました。当時Intelは8085と8086のチップを売り出していました。そのため、それらのチップをサポートするボードと、ISISというOSを用いたプログラミング用の開発マシンを提供してくれたのです。プログラミング言語はPL/Mでした。それらについて学ぶため、Intelが教える講座も受講しました。これは当時に限らず今もよくあることです。最新の技術について最初に教えるのはやはりベンダです。当時プログラミングに使えるメモリ容量はほんのわずかでした。独立した組込みのグラフィックスカードのプログラミングも行いました。やがて、PC、Windows、OS/2が登場しました。それでMicrosoftに直接関わる機会が増えました。

―カンファレンスでBill Gatesに会ったことがあるのですね。

ええ。制御システム用にExcelを多用している時期があり、Microsoftが興味を示したのです。ニューヨークでのOS/2用Excelの発表会に招待されたりしました。Microsoftは当時はまだそれほど大きくありませんでした。Bill Gatesは会場の後ろのほうにいて、私は彼と気軽に技術の話をしました。

昔とは大きく変わったこと

―キャリア全体を通じて、最も大きく変化したものは何ですか？

ハードウェアのプラットフォームそのものですね。私がPL/Mとマイクロプロセッサで仕事を始めたころは、メモリの制約にとても神経を使わなくて

SoftwareDesigner #45 Chris Timossi

元ソフトウェアエンジニア、ローレンス・バークレー国立研究所

はなりませんでした。今最も将来性のあるプラットフォームは携帯電話で、膨大な可能性を秘めた分野です。かつての私たちにとってマイクロプロセッサがそうだったように、斬新でワクワクします。もう1つは、Atmelなどが出している極小のマイクロコントローラです。CPUや入出力装置、メモリなどがすべてチップに収まっています。昔なら数万ドルもしたようなとてつもなく高度なツールも付属しています。今では趣味でそのようなツールを手にできるのです。制御システムを作ろうと思えば、開発キットにすべての機能が含まれています。

―携帯電話で制御システムですか？

粒子加速器の制御まではできないとしても、そこにはさまざまな可能性があります。簡単な例ですが、私はiPhone用にApp Storeからライト制御アプリを購入しました。このアプリを使えば、車から降りずに玄関の電灯を点けることができます。粒子加速器の分野でも世界全体でも、知識がひたすら末端へと応用されてきているという現象が起こっています。何かにプロセッサを組み込みたいと思えば、ほとんどの場合、そのためのツールが存在します。次の段階は、そうしたデバイス同士でどのように通信を行うかを考えることにあるでしょう。大雑把に言えば、私が仕事を始めたころの技術もそのような段階にありました。唯一異なるのは、今は「RAMが4Kバイトしかなかった時代を覚えている？」なんて言うことですね。

そんなわけで、趣味でプログラミングをする人にとってはとても良い時代になりました。また、プロになろうと思う人にとっては、この分野は今までと同様に急速に変化しています。たとえば私の同僚は、プログラミング可能なデバイスとの超高性能デジタルインターフェースの構築に取り組んでいます。これは非常に斬新な取り組みです。

どのように学べばよいか

―そのようなことはどこで学べばよいのでしょうか。

若いプログラマには、学べる機会があればどのような機会も活用してほしいとアドバイスしたいです。今でもベンダから教材を入手できます。その点はあまり変わっていません。ベンダが新しい製品を出すときには、何とか短期間で売ろうとします。そのために、マーケティングの一環として多くの教材を開発していたりします。ときには有料の場合もあります。とくに高度に専門的な内容の教材はそうです。一方、オープンソースの技術については、YouTubeのビデオで解説されていたりする場合もあります。

―書籍はどうでしょうか。

書籍もずいぶん購入しました。実際にそのうちの何冊を腰を据えて読んだかは疑問ですけどね。研究所に勤めていたころには、O'Reillyのライブラリにオンラインでアクセスできるプログラムがありました。なので、1冊をじっくり読むより、そのライブラリを巨大なデータベースとして使い、サンプルを探していました。実際に仕事に携わるエンジニアにとっては、1冊を隅から隅まで読み通すよりもそのほうが理にかなっていると思います。おそらく、だからこそ講座の履修にも意味があるのでしょう。講座を受講すれば、気を散らすことなくじっくり勉強できますから。ベンダがサポートする無料のイベントもたくさんあります。私は今でもそのようなイベントに出かけています。

―ソフトウェアエンジニアにとっては生涯学び続けることが仕事の一部みたいな話ですね。

実際にそうだと思います。一方で、私が関わってきた分野などではさらに専門化が進んでいます。昔はそこまで進んではいませんでした。当時はプロジェクトチームで何が必要かを見定め、全員で短時間でそれに対応していくというやり方でした。たとえば、今では厳密に粒子加速器の制御システムソフトウェアだけを専門とするグループがあります。仕事の幅をかなり絞り込んでいるんですね。昔はいろいろなことを自分たちでやらなければならず、OSに関わる

こともありましたが、今は少なくともいくつかの分野では状況が変わっています。

どのような分野でも学び続けることが大切

―若いプログラマが自分の専門性を見いだし、その開発がキャリアのすべてになる場合もあるのでしょうか。そんなに専門化が進んでいるのですか？

それはおもしろい質問ですね。狭い専門分野で経験を積み、それをキャリアにしてしまう可能性は理論的にはあり得るでしょう。ですが、実際にはそうはならないのではないかと思います。専門化にも流行り廃りがありますからね。逆にコンピュータサイエンスがもたらした変化の前に戻っているようにも思うのです。つまり、誰もが自分は機械工学や電気の技術者、あるいは物理学者だと思っていた時代です。将来について考えている学生なら、自分が選んだ分野の基礎をよく理解したうえで大学を卒業したいと思うはずです。それはつまり、専門分野がかなり多様化し変化も速いために、仕事をしていくうえでどの研究分野もそれだけで十分とは言えなくなっているからです。たとえば、粒子加速器の制御システムに関心があるなら、ハードウェア／ソフトウェアのインテグレータと思って仕事をするよりも、機械工学や電気の技術者と思うほうがよいかもしれません。

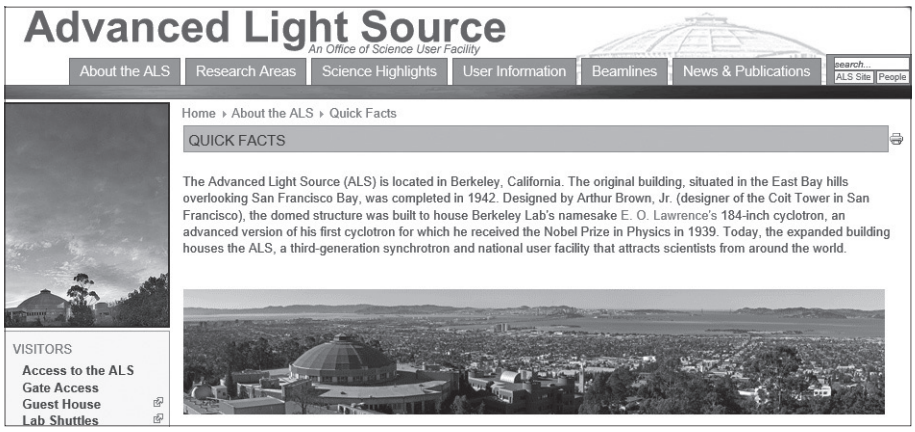
―技術以外の教育はどうですか？あなたは大学のころからシェークスピアに興味を持っていました。オペラを観に行ったりもします。地図やアウトドア活動にも関心があります。全体としてどう調和しているのでしょうか。

どれも技術の場合と同じようなものです。たとえば、私は妻のLoriとオレゴンのシェークスピアフェスティバルに何年も通っていますが、そのときには必ず無料の講演会にも行きます。シェークスピアは、私がつねづねもっと学びたいと思ってきたテーマです。歴史的な背景、役者がどう演じるか、当時の発音、そして最大規模の原典コレクションの1つが日本にある話など。シェークスピア関連の書籍なら隅から隅まで読みます。

同じプロセスは、アウトドアの経験にも活かされています。シエラネバダ[注2]にハイキングに行く場合、あらかじめジョン・ミューアなど昔の探検家の話を読んでおくと、より濃密な時間を過ごせます。風景を見ながら、昔の人が冬の嵐に立ち向かったり、黄金を求めてやってきたりしたことに思いを馳せることができます。あるいはガイドブックを読んでおけば、昔の人が苦労して場所を記し道順を示してきたことを想うことができます。SD

注2）カリフォルニア州の東部を縦貫する山脈。

▼サンフランシスコ湾を望む丘にあるALS



この個所は、雑誌発行時には記事が掲載されていました。編集の都合上、
総集編では収録致しません。

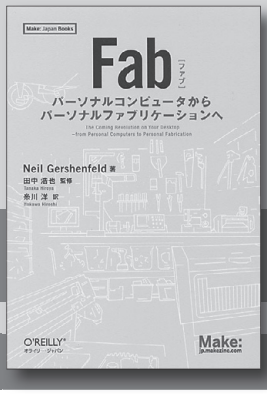
この個所は、雑誌発行時には記事が掲載されていました。編集の都合上、
総集編では収録致しません。

この個所は、雑誌発行時には記事が掲載されていました。編集の都合上、
総集編では収録致しません。

この個所は、雑誌発行時には記事が掲載されていました。編集の都合上、
総集編では収録致しません。

BOOK no.1 Fab
パーソナルコンピュータからパーソナルファブリケーションへ

Neil Gershenfeld【著】／田中 浩也【監修】／糸川 洋【訳】
A5判、264ページ／価格＝2,415円（税込）／発行＝オライリー・ジャパン
ISBN＝978-4-87311-588-7



本書は絶版となっていた『ものづくり革命』※の復刊本。現在、世界的にパーソナルファブリケーション（以下、PF）が注目を集めている。3Dプリンタやレーザーカッターを使い、インターネットで情報を共有し合えば、個人でも高度な創作に取り組めるというもので、MITで行われた「（ほぼ）なんでもつくる方法」という講座をきっかけに始まった活動だ。本書はその講座を担当した著者が、受講者および世界各地のPFの先駆者が作った作品、製作の動機、製作過程などを紹介する。高度な機器やノウハウがオープンになると、企業には無理なニッチな作品を個人が作り出すようになる。そんなものづくり文化の変化を感じることができる。

※ソフトバンククリエイティブ、2006年、978-4797333145

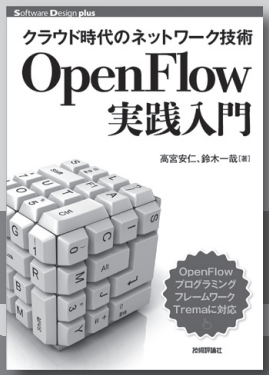
BOOK no.2 VMware vSphere構築・運用レシピ

㈱ネットワールド 工藤 真臣、ネットアップ㈱ 田中 隆行、シスコシステムズ合同会社 中本 滋之、樋口 美奈子【著】
B5変型判、336ページ／価格＝3,570円（税込）／発行＝インプレスジャパン
ISBN＝978-4-8443-3317-3

VMware vSphereの実践的なテクニックを紹介する1冊。vSphere Client、ESX/ESXi、vCenter Server、ネットワーク、ストレージ、仮想マシン、CLI、トラブルシューティングの分野から全96個のレシピが紹介されている。同製品の利用者を対象としているため、冒頭から専門的なレシピ紹介が始まる。いずれも具体的な課題を示し、それに対する解決策と操作方法を説明するという流れ。単なる操作方法だけでなく「vSphereの○○を理解する」といった課題も用意されており、そこでは図を用いながら数ページに渡って本製品の機能やしくみを解説する。操作手順だけでなく、しくみを理解しながら、効果的な使い方を習得できるよう工夫されている。

BOOK no.3 クラウド時代のネットワーク技術 OpenFlow 実践入門

高宮 安仁、鈴木 一哉【著】
A5判、336ページ／価格＝3,360円（税込）／発行＝技術評論社
ISBN＝978-4-7741-5465-7



本書は本誌連載「こんな夜中にOpenFlowでネットワークをプログラミング！［Trema編］」（2011年11月号～2012年5月号）を大幅に加筆修正したもので、3つのパート（「SDN/OpenFlow入門」「OpenFlowプログラミング入門」「ケーススタディ」）で構成されている。OpenFlowのしくみが身近な例でわかりやすく説明されており、自宅や職場で実際に試すことができるプログラムのほか、OpenFlowスイッチの自作法やGoogleをはじめとする大規模データセンターでのOpenFlow活用例なども紹介されている。一歩先を目指すネットワーク技術者はもちろん、ネットワーク制御の自動化に興味を持つプログラマの方にも活用できる1冊。

BOOK no.4 はじめてのOSコードリーディング
UNIX V6で学ぶカーネルのしくみ

青柳 隆宏【著】
A5判、448ページ／価格＝3,360円（税込）／発行＝技術評論社
ISBN＝978-4-7741-5464-0

1975年にリリースされたSixth Edition Unix（UNIX V6）のカーネルのソースコードを解説した書籍。はるか昔のOSのコードを読むなんてよほどの物好き？と思う人もいるかもしれないが、UNIX V6のカーネルにはOSの基本となるアイディアがほとんど実装されていて、最新OSの理解の助けにもなる。かつ、カーネルソースコードのボリュームも約1万行と、1人でも読み切ることが可能な範囲。OSのカーネルのソースコードを読み解くことで、コンピュータシステムの全体像が理解できるようにもなる。UNIX V6のコードをプロセス、割り込み、ファイルシステムなどのパートごとに解説した本書。じっくり読んでみていただきたい。

Hardware

キングジム、
周囲の人とその場でデータを共有できる ワイヤレス共有メモリ「パケッタ」を発売

㈱キングジムは2012年12月19日より、ワイヤレス共有メモリ「パケッタ」を販売開始した。

同製品は無線LAN環境のない会議やミーティングなどの場で、出席者同士がデータを共有することを想定して開発された製品。インターネット環境がなくても、パケッタがその場で新たなネットワークを作ることで、複数の人とすぐにデータのやりとりができる。データを共有する際の手順は次のとおり。

①あらかじめ共有したい端末（無線LAN機能が付いたPCやiPhoneなど）すべてに専用ソフトをインストールする
②パケッタの所有者がSSIDとパスワードを決めて「マイネット」を設定する
③パケッタを持っていない人は自分のPCやiPhoneなどから、そのマイネットを選択する
④マイネットにつながった全員が、自分の端末からパケッタの中身を参照できるようになる。データはドラッグ＆ドロップでやりとりが可能

パケッタの所有者同士なら、ワンクリックで相手と接続できる「かんたん接続」機能も利用できる。

通信規格はIEEE802.11b/g/nに対応し、通信速度は150Mbpsと高速であるため、容量が大きいデータも簡単に転送できる。

PCのほか、iPhoneやiPadなどのiOS端末でも利用が可能。最大32人同時にデータを共有できる。

本体色がホワイトでメモリ容量が8GB（9,870円（税込））のタイプと、本体色ブラックでメモリ容量16GB（14,700円（税込））のタイプの2種類がある。

◀パケッタ WS10-16G

CONTACT ㈱キングジム
URL http://www.kingjim.co.jp

Hardware

アライドテレシス、
10G対応インテリジェント・スタッカブルスイッチ 「CentreCOM x510」シリーズをリリース

アライドテレシス㈱は、ギガビット・インテリジェント・スタッカブルスイッチ「CentreCOM x510シリーズ」を12月26日より出荷開始した。

同シリーズは、10/100/1000BASE-Tポートを24ポート搭載した「AT-x510-28GTX」、10/100/1000BASE-Tポートを48ポート搭載した「AT-x510-52GTX」からなるインテリジェント・エッジ向けのxシリーズの新製品。

■CentreCOM x510シリーズの特徴

- 業界標準のコマンド体系に準拠したOS「AlliedWare Plus」を搭載。SwitchBlade x908やx900シリーズなどのxシリーズと組み合わせることで、コアからエッジまでの品質均等化を実現
- スタティックルーティング機能を標準搭載。ヘッドオフィスやブランチオフィスのディストリビューションスイッチやエッジスイッチとして最適
- 10ギガビットイーサネットモジュール（SFP+）に対応したスロットを4基備えており、別売のSFP+モジュールを搭載することで高速／大容量の10ギガビットイーサネット環境を構築できる
- 固定式冗長電源を標準搭載し、オプションのリダンダント電源装置を使用せずに電源冗長化が可能
- 複数のスイッチに双方向40Gbpsの帯域幅を持つ専用のスタックモジュール「AT-StackXS/OPシリーズ」を装着することで1台の仮想スイッチとして扱うことができるVCSをサポート。また、「ロングディスタンスVCS」を標準搭載し最大9kmまでの長距離VCSが可能

▼価格

製品名	標準価格（税別）
AT-x510-28GTX	298,000円
AT-x510-52GTX	498,000円
AT-StackOP/0.3（ファイバースタッキングモジュール（300m））	198,000円
AT-StackOP/9.0（ファイバースタッキングモジュール（9.0km））	448,000円
AT-StackXS/1.0（カッパースタッキングモジュール（1.0m））	50,000円

CONTACT アライドテレシス㈱
URL http://www.allied-telesis.co.jp

Hardware

バッファロー、最新のIntel Atomプロセッサ搭載の法人向けNASのラックマウントモデルを発売

㈱バッファローは2012年12月12日、法人向けNAS「テラステーション5000シリーズ」のラックマウントモデル「TS5400Rシリーズ」（容量：16TB/12TB/8TB/4TB）を発売した。

同製品は19インチラックに格納できるラックマウント（1U）対応の法人ユーザ向けのNASで、ファイルサーバとしての利用はもちろんPCサーバのバックアップにも利用できる。

高速転送を実現するためIntel Atomデュアルコアプロセッサ（2.16GHz）とDDR3メモリ2GBを搭載しているほか、4台のハードディスクを搭載し、保存データを保護するRAID 1/5/6/10と、4台の合計容量を1ドライブとして利用できるRAID 0に対応している。

Amazon Simple Storage Service（Amazon S3）との連携や遠隔地のテラステーションとの共有フォルダの同期に対応しており、インターネット回線を通じた遠隔地へのバックアップなどに活用できる。

監視カメラで撮影した動画データの保存先としても利用が可能なIPカメラ（RTSPカメラ）対応となっており、カメラ1台分のライセンスを標準添付している。無償提供のソフトウェアで映像データの管理／監視が行えることに加え、オプションの追加ライセンスを購入すれば最大10台のカメラによる録画に対応する。IPカメラによる監視環境の構築にも活用できる。

▼価格

容量・ドライブ構成	型番	価格（税込）
16TB（4TB × 4）	TS5400R1604	514,500 円
12TB（3TB × 4）	TS5400R1204	420,000 円
8TB（2TB × 4）	TS5400R0804	357,000 円
4TB（1TB × 4）	TS5400R0404	231,000 円

▲TS5400Rシリーズ

CONTACT ㈱バッファロー
URL http://buffalo.jp

Software

グレープシティ、Windows 8対応コンポーネント7製品を同時発売

グレープシティ㈱は、Windows 8、Visual Studio 2012といった新しいWindows環境に対応するコンポーネント7製品と「JPAddress 辞書更新サービス」の販売を2012年12月5日に開始した。具体的な発売製品は右表のとおり。

今回の7製品と1サービスの同時発売を記念して、次の2つのキャンペーンが行われている。

■JPAddressの無償提供

下記に該当する場合は「JPAddress for .NET 1.0J」の開発ライセンスを無償で提供する。提供期間は2013年3月末日の出荷分まで。

- SPREAD Desktop Pack 2013Jの購入者
- InputMan Desktop Pack 2013Jの購入者
- InputMan for Windows Forms 6.0Jユーザが7.0Jにバージョンアップする場合

■Desktop Packシリーズの同時購入キャンペーン価格

入力コンポーネントInputManと表計算データグリッドのSPREAD、それぞれのDesktop Pack 2013Jを一緒に購入する人に割引価格（3万円～8万円引き）を提供。キャンペーン価格の適用は3月末日の申し込み分まで。

▼新製品一覧

製品名	ライセンス価格（税込）	概要
InputMan for Windows Forms 7.0J	126,000 円	日本仕様の入力支援コンポーネント
MultiRow for Windows Forms 7.0J	126,000 円	日本仕様の 1 レコード複数行グリッド
SPREAD for Windows Forms 7.0J	168,000 円	Excel ライクなグリッドコンポーネント
SPREAD for WPF 1.0J	168,000 円	最強の表計算グリッド
SPREAD Desktop Pack 2013J	189,000 円	それぞれの WPF と Windows フォーム用コンポーネントが同梱されたセット製品
InputMan Desktop Pack 2013J	147,000 円	
JPAddress for .NET 1.0J	31,500 円	日本仕様の住所検索ライブラリ
JPAddress 辞書更新サービス	年額 47,250 円～ ※	JPAddress の住所ファイル更新サービス

※ 住所ファイルを配布するクライアント数により異なる

CONTACT グレープシティ㈱
URL http://www.grapecity.com/tools

Service

NTTスマートコネクト、仮想専用サーバ「スマートコネクトVPS」を提供開始

西日本電信電話㈱（以下、NTT西日本）とエヌ・ティ・ティ・スマートコネクト㈱（以下、NTTスマートコネクト）は、現在提供中の「Bizひかりクラウド」のサービスラインナップの拡充の一環として、セキュアな閉域網からも利用でき、共用サーバながら仮想的に専用サーバと同等の機能を有するIaaSメニュー「スマートコネクトVPS」を2012年12月11日から提供開始した。

データセンター内に構築したサーバ群を、仮想化技術を用いて、ユーザごとに専用のVPSとして提供する。個々のユーザのニーズに合った性能のVPSを複数組み合わせて利用できる。

■スマートコネクトVPSの特徴

- 「vSphere」によるVPSの提供
  数多くの導入／運用実績を持つVMware社のハイパーバイザ「vSphere」を採用しているため、豊富なアプリケーションに対応している
- 設備二重化などによる安定稼働
  サーバ設備の二重化はもちろんのこと、サーバ設備の故障発生時にHA技術を活用したサーバ切り替えを行い、ユーザシステムの安定稼働を実現する。また、スナップショットによる世代管理とシステム全体のバックアップの2つ機能を標準で提供することにより、サーバ設備の故障発生などの場合に、ユーザのデータやシステムをスナップショット／バックアップ取得時点に復旧可能
- 監視運用保守
  ユーザのシステムを24時間365日体制で監視し、安心して利用できる環境を提供する
- 多様なネットワーク接続
  インターネットに加え、フレッツ・VPNワイドなどの閉域網や、学術情報ネットワーク（SINET4）など、多様なネットワークと接続することが可能

料金は、ユーザごとに要望を聞きながら個別に見積もりを行う。

CONTACT NTTスマートコネクト㈱
URL http://www.nttsmc.com

Event

シスコシステムズ、「Cisco Connect Japan 2013」を開催

シスコシステムズ合同会社が、グローバルで展開しているマーケティングキャンペーン「Internet of Everything—インターネットですべてをつなぐ」の一環として、2013年2月13日～14日に「Cisco Connect Japan 2013」を開催する。

同イベントでは、ネットワークの力によって変革される未来とシスコが提唱するビジネステクノロジを体感できるという。40社の同社のパートナー企業の協力のもと、各界で活躍しているゲストによる講演、技術動向や先進事例を紹介するセッション、最新ソリューションを体験できる展示などのプログラムを実施する。基調講演、ゲスト講演として、次の講演が予定されている。

- **「コラボレーション革命～あなたの組織を引き出す10のステップ」出版記念講演**
  シスコシステムズ シニアバイスプレジデント　カール・ウィージ氏、シスコシステムズ バイスプレジデント　ロン・リッチ氏
- **チーム力を最大限に導くプロフェッショナルの行動力とコラボレーション**
  脳科学者　茂木健一郎氏
- **クラウドで広がる新しい世界 ～人とモノ、モノとモノをつなげる～**
  モデレータ：東京大学大学院 情報学環 学環長　須藤修氏
- **生き残るための攻撃的経営 ～マルチデバイス・クラウド・ソーシャル武装論～**
  慶應義塾大学 政策メディア研究科 特別招聘教授　夏野剛氏

▼開催概要

イベント名	Cisco Connect Japan 2013
日時	2013年2月13日（水）・14日（木）
会場	東京ミッドタウンホール＆カンファレンスおよびザ・リッツ・カールトン東京
主催	シスコシステムズ合同会社
参加費	無料（事前登録制）

CONTACT シスコシステムズ合同会社
URL http://www.cisco.com/web/JP

Service

IDCフロンティア、データセンター・クラウド相互接続サービス「プライベートコネクト」を提供開始

㈱IDCフロンティアは、利用者のWAN環境とIDCフロンティアの閉域網をインターネットを介さず安全に相互接続する「プライベートコネクト」を2012年12月6日より提供開始した。

IDCフロンティアの各データセンターや同社が提供するサービスは閉域網によって接続されているが、そこに利用者のWANを相互接続可能にした。これにより、同社のデータセンターやパブリック／プライベートクラウドサービスと、利用者のデータセンターや他クラウドとの接続が行えるようになる。このサービスでパブリッククラウドの異なるリージョン間を接続すれば、広域分散クラウドとしてシステムを構築することも可能。物理的な障害の影響を避けられ、広域サーバ負荷分散と組み合わせれば、負荷分散やバックアップサイトへの自動切り替えもできるようになる。

将来的には、外部通信キャリアや他事業者クラウドサービスとの相互接続も予定しており、第1弾としてソフトバンクテレコム㈱が提供する各種閉域ネットワークサービスとの接続を行う。

CONTACT ㈱IDCフロンティア
URL http://www.idcf.jp

Service

Amazon Web Services、データ集中型アプリケーションに最適化された「ハイストレージインスタンス」を発表

米国Amazon Web Services社（以下、AWS）は2012年12月26日、新たなAmazon Elastic Compute Cloud（Amazon EC2）のインスタンスファミリーである「ハイストレージインスタンス」を発表した。

このインスタンスは大容量データに高速アクセスが要求されるアプリケーション向けに最適化されており、35 EC2 Compute Units（ECUs）のコンピュート性能、117GiBのRAM、最大秒間2.4GBのI/O性能を出せる24個のハードディスクドライブで構成された48TBのインスタンスストレージを提供する。

インスタンスごとに大容量のダイレクトアタッチトストレージを持っているため、AWSクラウドで稼働するHadoopワークロード、ログ処理やデータウェアハウジングのようなデータ集中型のアプリケーション、大量のデータセットの処理と分析するための並列ファイルシステムに最適だという。

現在は、米国東部（北バージニア）リージョンで利用可能で、今後数ヵ月の間にほかのAWSリージョンでも利用可能になる予定。

CONTACT Amazon Web Services
URL http://aws.amazon.com/jp

Software

SafeNet、次世代ソフトウェア収益化プラットフォーム Sentinel LDKを発表

日本セーフネット㈱は2012年12月11日、米国SafeNetのソフトウェア保護、ライセンス付与、ライセンス管理プラットフォーム「Sentinel License Development Kit（以下、Sentinel LDK）」の提供を開始した。

Sentinel LDKとは、ハードウェアキーベースのプロテクション（HL）とソフトウェアアクティベーションによるプロテクション（SL）を融合した業界初のソフトウェア著作権マネージメント。128bitAES暗号アルゴリズムに基づいた強力なコピープロテクションに加え、ハードウェアとソフトウェアから自由にプロテクションキーを選択することで、多彩なライセンシング機能を使用できる。コピープロテクション機能、アンチ・リバースエンジニアリング機能、セキュア・ライセンシング機能などを備えており、ソフトウェアベンダはさまざまな方法で違法コピーの防止が可能になる。

同製品のプロテクションをソフトウェアへ組み込むためのツールキット「Sentinel LDKデモキット」は無料で提供されている。

CONTACT 日本セーフネット㈱
URL http://jp.safenet-inc.com

Software

エンバカデロ、
64bit Windowsに対応したC++Builder XE3のアップデートをリリース

エンバカデロ・テクノロジーズは2012年12月10日、C++用ビジュアル開発環境「C++Builder XE3」のアップデートを提供開始した。同製品およびRAD STUDIO XE3のProfessional版以上を購入した登録ユーザは無償で入手できる。

同製品はマルチプラットフォーム開発を特徴としており、従来よりWindows XP/Vista/7/8（Windows 8 UIサポート）、Mac OS X（Retinaディスプレイサポート）、そしてWeb向けのアプリケーションが単一コードで作成できる。今回のアップデートでWindows向け64bitコンパイラを搭載したことにより、同開発環境で作成したアプリケーションの稼働環境は従来のWin32とMac OS XにWin64が加わり、次のようなアプリケーションの開発が行えるようになる。

- 広大なメモリ空間の利用
- 64bit CPUの能力をフル活用
- C++11の機能やライブラリなどを利用可能
- 高度に最適化されたネイティブコードの生成

CONTACT エンバカデロ・テクノロジーズ合同会社
URL http://www.embarcadero.com/jp

Hardware

NEC、
OpenFlowに対応した「UNIVERGE PFシリーズ」のコントローラ機能を強化、スイッチの新製品を発売

NECはOpenFlowによる新たなネットワーク制御技術ProgrammableFlowに対応した製品「UNIVERGE PFシリーズ」において、プログラマブルフロー・コントローラ「UNIVERGE PF6800」（以下、PFC）の機能を強化し、併せて、プログラマブルフロー・スイッチ（以下、PFS）の新製品「UNIVERGE PF5248」「UNIVERGE PF5220」を2012年12月27日から販売開始した。

PFCで強化したのは、PFSへの設定を容易にするコンフィギュレーション簡易化機能の追加と、世界初となる仮想ルータでのIPv6ルーティングへの対応。

PFSは10Gポートにサーバを直接収容可能な「UNIVERGE PF5248」と、エッジスイッチとして最適な24ポートGbEスイッチ「UNIVERGE PF5220」をラインアップに追加した。

▼価格

製品名	価格（税別）	備考
UNIVERGE PF5248	215万円～	2ポート（10/100/1000BASE-T） 8ポート（10GBASE-R、1000BASE-X）
UNIVERGE PF5220	139万円～	24ポート（10/100/1000BASE-T） 2ポート（10GBASE-R、1000BASE-X）

CONTACT 日本電気㈱
URL http://jpn.nec.com

Topic

IPv6普及・高度化推進協議会、
アプリケーションのIPv6対応ガイドライン基礎編（第1.0版）を公開

2012年12月5日、IPv6普及・高度化推進協議会は「アプリケーションのIPv6対応ガイドライン基礎編」（第1.0版）を公開した。

同協議会のIPv4/IPv6共存WG アプリケーションのIPv6対応検討SWGは、アプリケーションのIPv6対応や、Webアプリケーションおよびソケットを直接扱うアプリケーションの両面についての情報を整理している。

2012年5月にソケット編のパブリックコメントを募集し、集まったコメントを盛り込み、「アプリケーションのIPv6対応ガイドライン基礎編」としてまとめた。この度、正式版として次の3種類の文書を公開した。

- アプリケーションのIPv6対応ガイドライン基礎編（第1.0版）
- アプリケーションのIPv6対応ガイドライン基礎編添付資料 アプリケーションのIPv6化例示プログラム集
- Asterisk の IPv6対応について

同SWGでは、今後も引き続きWebアプリケーションの対応についてまとめ、公開に向けた準備を行っていく。

CONTACT IPv6普及・高度化推進協議会
URL http://www.v6pc.jp

Letters from Readers

インターネット利用時間が1.5倍に増加

総務省の調査によると、インターネットの利用時間が5年間で1.5倍に増えたそうです。前回の調査では1日平均25分だったのに対し、今回は39分とのこと。しかし、39分程度ならちょっとスマホを使っているだけでいってしまいます。GmailやSkypeなどを使っていると、もっといきます。こういうWebサービスが仕事で当たり前に使われるようになると、次の調査ではもっと爆発的に増えそうです。

2012年12月号について、たくさんのお便りありがとうございました！

第1特集 なぜエンジニアは文章が下手なのか

ITエンジニアが書く仕様書、マニュアルなどの文章はわかりにくい、とよく言われます。本特集ではそれを改善するために、わかりにくくなる原因を探り、情報を整理し、表現する手法を紹介しました。

自分も文章が下手なのですごく頭に入ってきました。下手シリーズは、ぜひ、またやってください。今度は「プレゼンが下手」をよろしくお願いします。

和歌山県／たろサさん

まさに今直面している課題です。

愛知県／岩井さん

秀逸でした。今までシステム構築をしてきましたが、「マニュアル」「システム説明書」など、ユーザに好評だったことは、ほとんどありません。とくに「知識・目的」のGAPの説明が良かったです。今後システム設計・構築に関わるエンジニアの人は、今号の考察・知見を参考に、さらに「伝える技術」を磨き、顧客に喜ばれるシステム作りをしてもらいたいと思います。お世辞ではなく、今号は永久保存版にします。

富山県／むさしまるお

文章の書き方というのはどの分野においても一番重要なアウトプットの要素であり、以前からとても興味があり、特集があっただけでも満足でした。

大阪府／河合さん

文章の書き方の本は多数出版されているし、購入もしています。しかし、今月の特集はエンジニアに特化した内容でとても参考なりました。今後活用できると良いのですが。

愛知県／なおなおさん

「コンピュータ技術と関係ないのでは?」という声もあった一方で、普段の業務で苦労しているため、役に立ったという意見もたくさんいただきました。

第2特集 Nginx構築・設定マニュアル

ここ数年急速にシェアを伸ばしているOSSのWebサーバ「Nginx」。高速で軽量というのが特徴です。そのNginxのインストールや設定の方法、具体的な利用のしかたを取り上げました。

基本的には商用サーバを使うので今は利用することはないが勉強になった。

兵庫県／yoneさん

WebサーバといえばApacheが当たり前という考えが自分の中にあったのですが、Nginxというものは初めて知りました。シェアもそこそこあるようですし、自分でも機会があれば構築してみたいと思いました。

神奈川県／あろていさん

まだまだApache一辺倒なきらいがあるのでApache、lighttpd、Nginxでのパフォーマンス比較をやってほしいです。

東京都／hiddenさん

Webサーバは Apacheを使っている読者の方がやはり多いようですね。大規模なWebサービスを手がける企業においては、大量並列処理が得意なNginxはすでにWebサーバの選択肢の1つとして考えられているようです。今後、注目に値するソフトウェアだと思います。

一般記事 エブリデープログラマの発想と実践

プログラミングの楽しさにすっかりはまり、サンデープログラマならぬエブリデープログラマになってしまった著者による、初心者に向けたプログラミング入門記事をお送りしました。

プログラマにかぎらず多くの人がコンピュータを使う時代になってきましたので、誰もがある程度はプログラミングできたほうがハッピーになれる確率が上が

ると思います。

大阪府／SoGeeさん

昔、ひたすら自己のためだけに趣味でプログラミングをしていたころを思い出しました。いつまでも初心を忘れないようにしていきたいと思いました。

石川県／Keiさん

自分に必要なものを自ら作ることができる。作っている過程も楽しい。それがプログラミングの良いところです。仕事になると忘れがちですが、そういう楽しさを思い出していただけたのではないでしょうか。

一般記事 「Mahout」勉強会レポート

2012年9月26日、機械学習ライブラリ「Mahout」の勉強会が開催されました。同ライブラリ開発の第一人者であるGrant Ingersoll氏を迎えての開催です。その勉強会の様子をレポートしました。

自分の関心のある分野の記事でした。

神奈川県／眞　泰志さん

機械学習にたいへん興味あります。

福島県／黒田さん

触れる接点がないためとっつきにくかった。

兵庫県／長谷川さん

機械学習に興味があるという方、身近では使うことはないという方、両極端でした。しかし、私たちがよく使うWebサービスでは、ビックデータを解析する手段として機械学習はなくてはならない技術になっています。すでに誰もが恩恵を受けている技術と言えるでしょう。

フリートーク

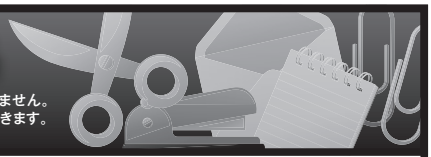
Windows 8を発売日にインストールしました。画面をタッチすると、肩がこります。目の前に液晶ディスプレイ2画面、手元に横スクロール用の小さな画面で合計3画面です。これで快適になりました。

東京都／WIN8さん

担当は考えが古いせいか、タッチパネル搭載のPCでも画面に直接指を触れるのに抵抗を感じてしまいます。Windows 8搭載PCを購入しても、使い慣れるまでに時間がかかりそうです。

エンジニアの能率を高める一品

仕事の能率を高める道具は、ソフトウェアやスマートフォンのようなデジタルなものだけではありません。
このコーナーではエンジニアの能率アップ心をくすぐるアナログな文具、雑貨、小物を紹介していきます。



Vブロック（ペントレー）／ショートタイプ

1,890円（税込）／margherita　http://www.margherita.jp

机で仕事をするとき、よく使う筆記具は常に手元に置いておきたいところですが、机上が散らかるのが困りもの。そんなとき、ペントレーを使うと雑多なものを置く場所が決まって、机上がスッキリします。ペン立てだと、数本のペンの中から必要な1本を選ぶ手間すら面倒なことがありますが、トレーの上には一番よく使う1本だけを置いておけば、そんな面倒も感じません。付箋紙やクリップ、スマホを置くのにも良いです。本製品はキーボードの奥に置いて使うことを想定して開発されたようですが、ケーブルが真ん中についているキーボードではトレーを置く位置をずらす必要があります。ワイヤレスキーボードだとそんな問題もなく、ケーブルがないことでさらに机上がきれいになります。ショートタイプは意外に置ける量が少ないので、こまごましたものが多い人は60cmのロングタイプが便利かもしれません。

（読者プレゼントあります。16ページ参照）

▲ワイヤレスキーボードのほうが使いやすく、机上もスッキリする

祝 12月号のプレゼント当選者は、次の皆さまです

①My Passport for Mac USB 3.0 ..... 沖縄県　鳥居恭時様
②充電式ラジオライト グラビカ ..... 和歌山県　平林忠様

★その他のプレゼントの当選者の発表は、発送をもって代えさせていただきます。ご了承ください。

次号予告

Software Design
March 2013

2013年3月号
定価1,280円　176ページ

2月18日発売

［第1特集］OpenStack、CloudStack、Cloudfoundry、Scalr、Eucalyptus

オープン環境でスキルアップ! もっとクラウドを活用してみませんか?

［第2特集］光、ギガビット、高速ネットワークを体験!

実践! ワイヤリングの教科書

休載のお知らせ

「温故知新　ITむかしばなし」（第20回）、「インターネットサービスの未来（これから）を創る人たち」（第20回）は都合によりお休みいたします。

お詫びと訂正

以下の記事に誤りがございました。読者のみなさま、および関係者の方々にご迷惑をおかけしたことをお詫び申し上げます。

■2013年1月号

●第1特集「いざというときに備えるシステムバックアップ」

P.19　特集タイトル部分
［誤］:「システム復旧術」
［正］:「システムバックアップ」

P.55　コラム　左段下から3行目
［誤］:HDDは長い期間512KB/Sectorでしたが
［正］:HDDは長い期間512B/Sectorでしたが

P.55　コラム　右段上から1行目
［誤］:/Secotrですので
［正］:/Sectorですので

SD Staff Room

●忙しくて夜空の月も観てない日々で、年末が暮れようとしています。そんな時期に編集後記を書いていますが、まだ特集製作が終わっていません（汗）。通り一遍ならばこなすことはきっと簡単なのに、春夏秋冬いつも原稿を依頼しては書を作り、気がつくと1年過ぎてしまいます。漢一匹編集稼業也。（本）

●先月のノロウイルス感染に引き続き、今度は年末進行中に伝染性単核球症（キス病）というのに罹患する。調べてみると爆笑問題の田中が入院した病気らしい。肝機能障害と血小板減少を併発し、忘年会は全て酒無し。免疫力が落ちてるのを実感。年末感ゼロだ。今年は病気にならないことを目標に。健康第一!（ま）

●もうすぐ息子がドラクエⅡ（!）をクリアしそうだ。かつてゲーセンでこづかいをつぎ込み、アトラスを通過できずに終わってしまった私。こうして父を追い越していくのだなぁあ……と、奥付のネタのために無理矢理こじつけてみたりする年末進行の恐ろしさ。本年もよろしくお願いいたします。（キ）

●「私はキリスト教ではないので、クリスマスには特に何もしません」という人をたまに見かけます。妻の祖母（90歳）はクリスチャンですが、昨年のクリスマスに特別なことはしなかったようです。これはキリスト教云々というより、彼女がたんに日本の「おばあちゃん」だからだと思います。（よし）

●Kindle Paperwhite 3Gが届きました。個人的には、iPadやAndroid端末で読むより紙の本に近い、心落ち着く感じがします。新聞もこれで読めると、新聞紙っぽくて馴染む気がします。端末の普及には時間がかかると思いますが、持ち歩きたいデバイスだなと感じる次第です。（ほ）

●昨今、バックカントリースキーにハマってる（ゲレンデ外の裏山で滑る）。不整地・深雪なので太い板が有利。僕の板は片方で普通の板の2本分。今のところチューン代は普通の板と同じだが、RV車の洗車みたくデカイ=割高になり、いつぞやか今シーズンは片方だけで……なんてなりませぬよう。（yeti）

●今シーズン初生ガキ~と喜んだのも束の間、ノロウイルスに侵されました。奇跡的に症状は軽かったのですが結局寝込むこと数日。体力には自信があったのに……。1週間以上経った今もまだ胃腸の調子は戻らず（涙）。美味しいものも食べたいし、スノボにも行きたいから早く本調子に戻ってほしいなぁ。（まつ）

ご案内

編集部へのニュースリリース、各種案内などがございましたら、下記宛までお送りくださいますようお願いいたします。

Software Design 編集部
ニュース担当係

［FAX］
03-3513-6173
※ FAX番号は変更される可能性もありますので、ご確認のうえご利用ください。

［E-mail］
sd@gihyo.co.jp

Software Design
2013年2月号

発行日
2013年2月18日

●発行人
片岡　巌

●編集人
池本公平

●編集
金田富士男
菊池　猛
吉岡高弘
＊
細谷謙吾（書籍編集長）
取口敏憲

●編集アシスタント
松本涼子

●編集協力
坂井直美
金子卓也(トップスタジオ)

●広告
中島亮太
北川香織

●発行所
㈱技術評論社
編集部
TEL：03-3513-6170
販売促進部
TEL：03-3513-6150
広告企画部
TEL：03-3513-6165

●印刷
図書印刷㈱

この個所は、雑誌発行時には広告が掲載されていました。編集の都合上、
総集編では収録致しません。

この個所は、雑誌発行時には広告が掲載されていました。編集の都合上、
総集編では収録致しません。