Special Feature 01
SDN/OpenFlowの今
Special Feature 02
伝わる図解術
2013 December 12
[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
Software Design
2013年12月18日発行
毎月1回18日発行
通巻344号
（発刊278号）
1985年7月1日
第三種郵便物認可
ISSN 0916-6297
定価
1,280円

SDN/OpenFlowの流れを総まとめ!

▶▶▶ Special Feature 01

SDN/OpenFlowで幸せになれますか

仮想ネットワークをどう使うか

▶▶▶ Special Feature 02

開米瑞浩直伝

下手でも好印象で効果絶大

エンジニアの伝わる図解術

▶▶▶ 特別企画

基礎の基礎だけど案外知らない?

LinuxとFreeBSDのファイルシステムの良い・悪いところをご存じですか?

▶▶▶ 一般記事

未来を見透すヘッドマウントディスプレイ「Mirama」a.k.a.VIKING

iOS 7アプリの魅力はどうやって引き出すのか

▶▶▶ 新連載

ベンチマークを極める実践テクニック

サーバマシンの測り方

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。

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。

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60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。

Project Avatar

Java と HTML5 をつなぐ「Project Avatar」

「Project Avatar」は、クラウドベースのJava EEアプリケーションとHTML5ベースのクライアントのシームレスな連携をゴールとして、開発が進められているアプリケーションフレームワークです。Oracle社によって2011年にコンセプトが発表され、2年に渡る開発を経て、2013年9月にオープンソース化が発表されました。

Webを利用したアプリケーションを開発するうえで、HTML5はすでになくてはならない存在になっています。しかしHTML5を利用する場合、そのアプリケーションロジックはJavaScriptで実装することが前提となるため、エンタープライズ規模のアプリケーション開発という視点では必ずしも優れているとは言えない側面があります。

Project Avatarが目指すのは、サーバサイドを担うJava技術とクライアントサイドを担うHTML5との間にあるギャップを埋め、両者を組み合わせたハイブリッドなアプリケーション開発技術を確立することです。具体的には、サーバサイドではRESTやJSON、Web Sockets、Server-Sent Events（SSE）といった通信技術をサポートするJavaScript向けのデータサービスを提供し、クライアントサイドでは任意の言語で書かれたサービスとHTML5コンポーネントとのバインディングや、JavaScriptをより簡単に利用するためのフレームワークを提供するという構成になります。

Avatar のアーキテクチャ

Avatarにおいてサーバサイドの基盤になるのは、2013年6月にリリースされたJava EE 7と、2014年にリリース予定のJava SE 8（現在はEarly Access版が公開中）です。Java EE 7は、従来のバージョンに比べてバッチ処理技術の標準化や開発生産性の向上といったエンタープライズ要件を意識した改良が行われている一方で、Web SocketやREST、JSONといったWebアプリケーション技術のサポートにも注力されている点が大きな特徴です。

AvatarではこのJava EE 7をベースに、「Project Nashorn」によって拡張された独自のJava EEコンテナが利用されます。Nashornは Java仮想マシン上で稼働するJava Scriptエンジンです。開発者はこのNashornの上で“Avatarサービス”と呼ばれるサービスを実装することにより、クライアントサイドのアプリケーションとの連携を実現します。Avatarサービスは Node.jsのプログラミングモデルによって実装できるほか、既存のJavaライブラリを呼び出して利用することも可能です。また、JavaScriptを使わずにPure Java実装によってサーバサイドのサービスを構築することもできます。

クライアントサイドはHTML5やJavaScriptを中心とするWeb標準技術が基盤となります。ここでは、Avatarはサーバから受け取ったモデルデータをHTML5コンポーネントに関連付ける役割を果たします。この関連付けはJava EEに含まれるExpression Language（EL）式を使って定義します。EL式が利用できることで、動的なページを作るためのコントローラのロジックをJavaScriptで記述する必要がないというメリットがあります。

クライアントサイドのしくみで特徴的なのは、JavaScriptによる実装が最小限で済ませられるように意識されている点です。これは、JavaScriptによるアプリケーションロジックの実装を可能な限り削減するというAvatarの基本コンセプトに基づくものです。とくにエンタープライズ系の開発者の場合、JavaScriptに関するノウハウを持たないケースも少なくないことから、Avatarの存在は大きな武器になるでしょう。

オープンソース化されたAvatarは、公式サイトにて最初のEarly Access版が公開されています（本稿執筆時点）。そのほか、チュートリアルとしてさまざまなサンプルアプリケーションの実装例もまとめられており、手元のPCで実際の動作を確認することができます。SD

Avatar
https://avatar.java.net/

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。

Contents

Software Design Dec. 2013

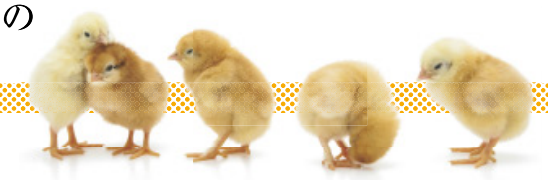
1

第1特集 Special Feature 01

SDN/OpenFlowの流れを総まとめ!

SDN/OpenFlowで幸せになれますか?

仮想ネットワークをどう使うか 017



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。

ソフトウェア・デザイン［目次］

Contents

Software Design Dec. 2013

2

第2特集 Special Feature 02

エンジニアの伝わる図解術

下手でも好印象で効果絶大

特別企画 Article

一般記事 Article

巻頭Editorial PR Editorial PR

アラカルト A La Carte

広告索引 AD INDEX

技術評論社の本が電子版で読める！

電子版の最新リストは

Gihyo Digital Publishingの

サイトにて確認できます。

http://gihyo.jp/dp

GIHYO Digital Publishing 未来の"普通"を、今。

Google play

amazon.co.jp

※販売書店は今後も増える予定です。

法人などまとめてのご購入については
別途お問い合わせください。

■お問い合わせ
〒162-0846
新宿区市谷左内町21-13
株式会社技術評論社　クロスメディア事業部
TEL：03-3513-6180
メール：gdp@gihyo.co.jp

ソフトウェア・デザイン［目次］

Contents

Software Design Dec. 2013

3

Column

Development

OS/Network

Inside View

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 ＞ Sunnybeach/Getty Images
Illustration イラスト ＞ フクモトミホ、高野 涼香、中川 悠京
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。

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！

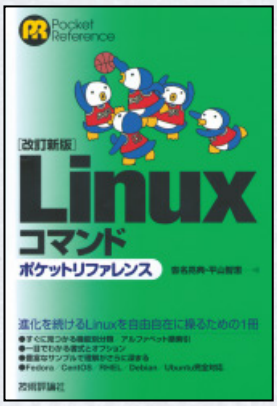
田口貴久ほか　著、日本仮想化技術株式会社　監修
四六判／352 ページ
定価 3,129 円（2,980 円＋税）
ISBN 978-4-7741-5600-2

牟田口大介　著
四六判／592 ページ
定価 2,919 円（2,780 円＋税）
ISBN 978-4-7741-5542-5

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

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

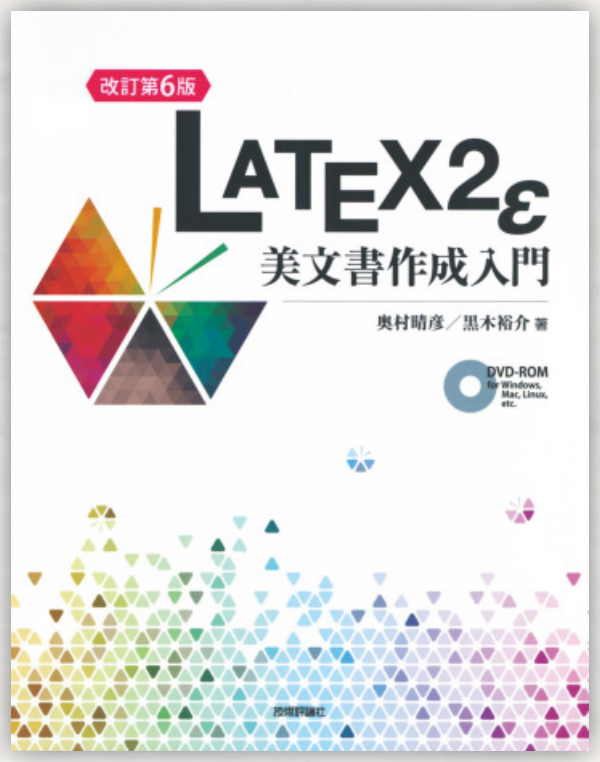
土井毅、髙江賢、飯島聡、髙尾哲朗 著　山田祥寛 監修
四六判／488 ページ
定価 2,709 円（2,580 円＋税）
ISBN 978-4-7741-4948-6

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

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

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

LaTeX入門の定番書、待望の改訂第6版登場！



［改訂第6版］LaTeX2ε 美文書作成入門

奥村晴彦、黒木裕介　著
B5 変形判／432 ページ
定価 3,360 円（3,200 円＋税）
ISBN 978-4-7741-6045-0

DVD

概要

前身となる「LaTeX 美文書作成入門」発行から22年間、おかげさまでみなさまにご愛顧いただいております LaTeX 入門の定番書に、待望の最新改訂版が登場です！
LaTeX のインストールはもちろん、「自分で体裁を変更したい」といったある程度高度な知識が必要なところまで幅広く網羅。TeXを使うすべての人にお勧めの一冊です。
今版では、長年の加筆・修正でわかりにくくなったり古くなっていた項目を整理し、最新の環境に合わせて新たに構成しなおしました。また、国際的なTeX Live 2013に対応し、これまで以上にかんたんに各種パッケージをご利用いただけます。
付録DVD-ROMにはTeX Live 2013と関連ツールを簡単にインストールできるインストーラーを収録（Windows／Mac）。LinuxやFreeBSDなどでもご利用いただけます。

こんな方におすすめ

・TeXを学習しようとしている人
・論文をTeXで執筆する学生・教師

本書電子版もGihyo Digital Publishingにて販売中。
こちらもご利用ください。
https://gihyo.jp/dp

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。

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。

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。

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

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。

OpenFlow実践入門
高宮 安仁、鈴木 一哉 著
定価 3,200円+税　ISBN 978-4-7741-5465-7

はじめてのOSコードリーディング
青柳 隆宏 著
定価 3,200円+税　ISBN 978-4-7741-5464-0

Webサービスのつくり方
和田 裕介 著
定価 2,180円+税　ISBN 978-4-7741-5407-7

日本一の地図システムの作り方
㈱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
定価 2,580円+税　ISBN 978-4-7741-5325-4

Androidアプリケーション開発教科書
三苫 健太 著
定価 3,200円+税　ISBN 978-4-7741-5189-2

プロのためのLinuxシステム・10年効く技術
中井 悦司 著
定価 3,400円+税　ISBN 978-4-7741-5143-4

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

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

業務に役立つPerl
木本 裕紀 著
定価 2,780円+税　ISBN 978-4-7741-5025-3

Apache[実践]運用/管理
鶴長 鎮一 著
定価 2,980円+税　ISBN 978-4-7741-5036-9

プロになるためのデータベース技術入門
木村 明治 著
定価 3,180円+税　ISBN 978-4-7741-5026-0

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
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

最新刊！

沼田 哲史 著
B5変形判・360ページ
定価 3,200円(本体)+税
ISBN 978-4-7741-6076-4

ニコラ・モドリック、安部 重成 著
A5判・336ページ
定価 2,780円(本体)+税
ISBN 978-4-7741-5991-1

水野 操、平本 知樹、神田 沙織、野村 毅 著
B5判・128ページ
定価 2,480円(本体)+税
ISBN 978-4-7741-5973-7

沓名 亮典 著
A5判・416ページ
定価 2,880円(本体)+税
ISBN 978-4-7741-5813-6

小飼 弾 著
A5判・200ページ
定価 2,080円(本体)+税
ISBN 978-4-7741-5664-4

青木 直史 著
A5判・288ページ
定価 2,980円(本体)+税
ISBN 978-4-7741-5522-7

中井 悦司 著
B5変形判・384ページ
定価 2,980円(本体)+税
ISBN 978-4-7741-5937-9

Japanese Raspberry Pi Users Group 著
B5変形判・256ページ
定価 2,380円(本体)+税
ISBN 978-4-7741-5855-6

菅野 裕、今田 忠博、近藤 正裕、杉本 琢磨 著
B5変形判・336ページ
定価 3,200円(本体)+税
ISBN 978-4-7741-5567-8

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

データサイエンティスト養成読本編集部 編
B5判・152ページ
定価 1,980円(本体)+税
ISBN 978-4-7741-5896-9

Software Design編集部 編
B5判・176ページ
定価 1,880円(本体)+税
ISBN 978-4-7741-5888-4

データベースエンジニア養成読本編集部 編
B5判・136ページ
定価 1,980円(本体)+税
ISBN 978-4-7741-5806-8

WINGSプロジェクト 著
B5判・352ページ
定価 2,580円(本体)+税
ISBN 978-4-7741-5611-8

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

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。

DIGITAL GADGET

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

Volume 180 >>>>>

未来に投資～デジタルガジェットの宝庫～クラウドファンディング

ハードウェアの作り方の進化

ハードウェアの設計のためのプラットフォームを提供するUpverter社によると、近い将来、下記のような推移が予想されているそうです（Upverter社のブログより）。

2014年頃…ハードウェアのプロトタイプが一晩で作れるようになる
2016年頃…ハードウェアのオンデマンド製造が可能になる
2018年頃…農業関連の革新的デバイスが登場する
2019年頃…ハードウェア機器の予約販売が主流に
2020年頃…デバイスの50%以上がそれまでになかった新しいハードウェア企業によるものになる。医療に関するデバイス開発の流行がはじまる
2025年頃…革新的ハードウェアメーカーが世界で100社以上に

まだ世の中に生まれていない新規ハードウェアデバイスを予約購入する風潮は、多くの人々から資金を集めるクラウドファンディング「Kickstarter」で一般化しました。海外発送可能な商品も多く、日本で話題になるプロジェクトもあります。最近カナダでもクラウド資金調達サービスを開始しました。2013年1月に発表された2012年の成果では、2億7439万1721ドル（約240億円）と、2011年の9934万4382ドルを大きく上回る結果となりました。

The Best of Kickstarter 2012 — Kickstarter
http://www.kickstarter.com/year/2012

デジタル機器が目立つKickstarterですが、実際のプロジェクトはビデオゲーム開発、ゲーム周辺機器、ミュージシャン、アーティスト、映像制作、服飾、アートプロジェクトなど多岐にわたっています。とくに最近のKickstarterはビデオゲーム関連での資金調達と、成功事例の積み重ねが顕著です。全体としてゲーム企画の1/3が成功しているとのことですが、逆に考えると2/3が失敗しているとも言えます。ゲームの場合、資金調達してから実際に開発して完成するまでの期間が長くかかるため、すぐに購入してゲームを楽しみたいユーザからは敬遠されているようです。

一例を挙げると、ゲームタイトル「Torment: Tides of Numenera」は、90万ドルの目標額に対して418万ドル（約4億円）の資金を7万人以上の人々から集めました。Androidベースの小型ゲーム機「GameStick」は開始1日で目標額の10万ドルを突破し、約6,000人の支援者から約65万ドル（6500万円相当）を集めました。

ここまで資金調達できると喜ばしいだけでなく、それだけ期待が大きく多くの人に支えられていることが実感できるのもクラウドファンディングならではと言えるでしょう。

また、多くのユーザの資金援助により完成したゲームも質の高いものばか

OUYA Android-powered Gaming Console
家庭用ゲーム機
［獲得資金8,596,474ドル］

FORM 1 3D Printer
安価で高精細な3Dプリンタ
［獲得資金2,945,885ドル］

Oculus Rift
安価なVRヘッドマウントディスプレイ
［獲得資金2,437,429ドル］

Pebble E-Paper Watch
本連載でもたびたび紹介しているアプリがインストールできる電子ペーパー腕時計
［獲得資金10,226,845ドル］

※本記事で紹介しているものは国内未発表・未発売のものを含んでおります。

りで、ゲーム関係の賞を受賞しているものもあります。ゲーム企業による開発プロジェクトでは前例のなかったような新しいタイプのゲームや、途中で企業がつぶれて開発プロジェクトが頓挫してしまったゲームの継続開発、海外ゲームのローカライズ、ゲーム関連の周辺機器、ゲーム機器の写真アーカイブサイトなど、ゲーム関連だけでもさまざまです。さらにクラウドファンディングでデビューする若いクリエータ世代も目立つようになってきました。

支援者の心をひきつける紹介ビデオの作り方

Kickstarterにはプロジェクトの概要を紹介する、短めでありながら魅力的な紹介動画が必ずといっていいほど用意されています。3D CGや実写合成を使って、あまりにも見栄えの良いプロモーションビデオを作ってしまうのも考えものです。ベイパーウェア（Vaporware：蒸気のような、実体のないもの）と印象づけてしまうのは損です。成功しているプロジェクト、多額の資金を集めているプロジェクトはどれも印象的な紹介ビデオが用意されています。

成功する秘訣に紹介ビデオが重要であるとして、どんな紹介ビデオが成功に結びつくのでしょうか？　Kickstarterからのアドバイスにより明らかにされているいくつかのポイントを紹介しましょう。

- 撮影前に十分準備しましょう。何回もデモの練習をしましょう
- 60～90秒くらいの短いビデオにしましょう。肝心な事柄は一番最初に
- 原稿を読み上げるのではなく、あなたの言葉でしゃべりましょう
- その商品は何ができますか？　何が違いますか？　視聴者の疑問に答えましょう
- 画面があなたを映していないときも何か話し続けるようにしましょう
- 可能であれば、ビデオ映像は随時更新していきましょう
- ナレーションをじゃましない、著作権に問題のない音楽を使いましょう
- 高価な機材を使わなくてもいいので「はっきり」としゃべりましょう
- あなた自身、あなたの背景を紹介しましょう（短時間で）
- あなたとプロジェクトのストーリーを語りましょう（少年の頃から考えていたアイデアであるなど）
- なぜその資金が必要なのかを説明しましょう（高価な機材や素材が必要であるなど）
- 成功しているプロジェクトの紹介ビデオを参考にしましょう
- 必ず最後に「ありがとう！」と感謝しましょう

この映像作りのコツはクラウドファンディングばかりではなく、スマートフォンアプリやWebサービスのプロモーションなど各所で役立つことでしょう。

著名な映画監督や有名俳優がKickstarterで映像や映画製作のプロジェクトを始めるなど、いわゆる本気のプロもクラウドファンディングに注目し、その資金力だけでなく、自由さや支援者との距離の近さに期待しています。著名なアーティストやデザイナが、クライアント仕事ではなく自分自身のプロジェクトのためにKickstarterを活用しているのです。

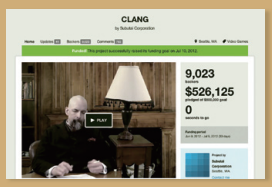
SF小説家Neal Stephensonのプロジェクトや、著名なグラフィックデザイナStefan Sagmeisterのプロジェクト、映画監督David Fincherのプロジェクトなどが注目されています。

クラウドファンディングのこれから

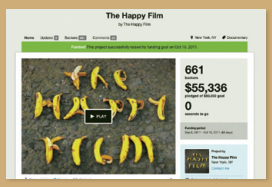
日本でも、スポーツ専門、アスリートの応援のためのクラウドファンディングや、音楽やアート系に強いコンテンツクリエータへのクラウドファンディング、社会貢献を専門としたものなどさまざまなタイプのクラウドファンディングが広がってきています。

筆者もライブラリつきのコワーキングスペースの設立を支援したり、教え子の学生たちのプロジェクトを支援したり、小額ながらも積極的に活用しています。常に情報がアップデートされ、プロジェクトの進行状況がわかり、ワクワクや焦燥感が共有でき、応援した商品が手元に届いたり支援したイベントが成功したときは、数多くの支援者のたった1人だとしても、その喜びはひとしおです。

クラウドファンディングでは、言わずもがなですが、多額の資金が集まったから成功となるのではなく、目標金額が集まり、多くの人に支えられながらプロダクトやコンテンツの開発が上手くいき、広く知られるようになってプロジェクトそ



Neal Stephensonのゲームプロジェクト「CLANG」（現在残念ながら保留中）



Stefan Sagmeisterの「Happy Film」プロジェクト

David Fincherのホラーアクション系アニメーション「The Goon」

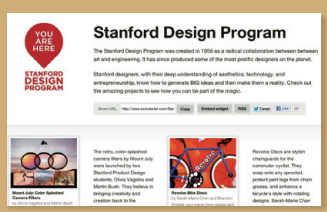
クラウドファンディングニュース「Vourno」

のものが成功することが成功だと言えます。

一方で、Kickstarterは失敗プロジェクトを検索エンジンから見つけられにくくしているという批判もあります。失敗から学べることも多く、同じ間違いを繰り返さないよう失敗事例の共有も大切です。しかし「うまくいく!」というイメージも大切でしょう。Kickstarterは今までになかった初めての試みも歓迎される傾向があります。もちろん過去に類似プロジェクトを成功させていたり、その道の専門家が参加していると心強いでしょう。

ルーヴル美術館もクラウドファンディングを活用しはじめました。美術館や博物館は古くからクラウドファンディング的に寄付を集めて運営している例が多く、館内の展示室や椅子1つにまで、寄付した人の名前が入っています。日本でも「谷町」という、大相撲でひいきの力士の後援をする風潮が古くからあります。クラウドファンディングで映像ニュースを提供する「Vourno」(https://www.vourno.com/)も登場し、ジャーナリズムやメディアにまで、クラウドファンディングの考え方が広がってきています。

米国スタンフォード大学では、Kickstarterの成功プロジェクトから、その手法を学ぶコースができたそうです。これからはクラウドファンディングそのものがますます広範囲に広まるとともに、クラウドファンディング的考え方、クラウドファンディング的アプローチの展開が期待されます。SD



米国スタンフォード大学「d.school」

gadget 1

NFC Ring

http://www.kickstarter.com/projects/mclear/nfc-ring

NFCタグ搭載指輪

ありそうでなかったNFCタグ機能を搭載した指輪。昔あったJava Ringを思い浮かべている人もいるかもしれません。指定サイズのNFCリングと、専用アプリのセットで22ポンド(一般への市販の際には価格が変わる可能性があります)。NFC Ringでスマートフォンのロックを解除したり、名刺交換の代わりにNFC Ringで連絡先を読み取ってもらったりできます。

gadget 2

WaterColorBot

http://www.kickstarter.com/projects/1894919479/super-awesome-sylvias-watercolorbot-0

水彩画プロッター

WaterColorBotは水彩絵の具と水を使って、自動で水彩画を描くことのできるデジタルプロッターです。筆が平面上を動くことで、人間が描くように動作する機械をコンピュータで制御し、SVG形式の絵柄を紙に描くことができます。何より驚きなのは、当時12歳のSylviaさん(http://sylviashow.com/)が中心となったプロジェクトであること。295ドルで一般販売(予約)も始めたようです。

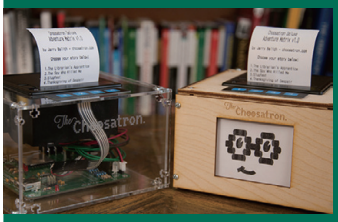
gadget 3

The Choosatron

http://www.kickstarter.com/projects/jerrybelich/the-choosatron-interactive-fiction-arcade-machine

紙芝居的アドベンチャーゲーム専用プリンタ

The Choosatronはユーザの選択で次々とプリントされる文章が変わっていく、自動紙芝居的なプリンタです。昔流行った、選択肢を選びページをめくっていくゲームブックのようなものです。野外で位置情報を活用したロケーションゲームや、宝探しのようなアトラクションも考えているそうです。木製バージョンが159ドル。シースルーバージョンもあり。プリントアウト予定のアドベンチャーブックの電子書籍版も用意されるそうです。



gadget 4

Structure Sensor

http://www.kickstarter.com/projects/occipital/structure-sensor-capture-the-world-in-3d

iPad活用3Dスキャナー

Structure SensorはiPadを活用して、3Dスキャナを安価に実現するデバイスです。赤外線ライトが搭載され、iPadのLightningコネクタでデバイスを接続します。約0.4〜3.5メートルの距離にある物体の形状を、iPadの画面で確認しながらデータ計測ができます。SDKやAPIも公開予定とのこと。3Dスキャンした家具のデータを自宅でAR技術を使って再現することを考えているそう。想定額の10倍の資金を集めました。予価349ドル。

結城 浩 YUKI Hiroshi
http://www.hyuki.com/
Twitter：@hyuki

結城浩の再発見の発想法 7

Random Sampling

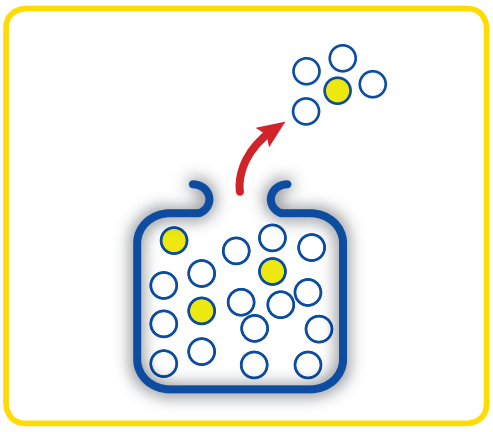
Random Sampling ——ランダムサンプリング

ランダムサンプリングとは

ランダムサンプリング(random sampling)とは、調査すべきものが多数あって、そのすべてを調査できないときに、少数をランダムに抜き出す調査のことです(**図1**)。日本語では「無作為標本抽出」といいます。

「**料理の味見**」は身近なランダムサンプリングです。たとえばスープの味加減を調べたいとします。鍋のスープをすべて飲んでしまえば、確実に味はわかります。でも、それではスープがなくなってしまいます。そこで、スープをよくかき混ぜてから一口だけ味見をします。これがランダムサンプリングです。

▼図1　ランダムサンプリング



「**世論調査**」でもランダムサンプリングはよく使われます。どれだけの国民が現在の内閣を支持しているかを調べたいとき、国民全員に聞き取り調査をすれば正確にわかりますが、それはたいへん困難です。そこでたとえばランダムに電話番号を生成し、その番号にかけて調査をします。これもランダムサンプリングの例です。

なぜランダムサンプリングを行うのか

どうしてランダムサンプリングを行うのでしょうか。それは、**すべてを調査しては無意味**だったり、**すべてを調査するのにコストや時間がかかり過ぎ**たりするからです。

「料理の味見」では、鍋のスープをすべて飲むことは不可能ではありません。でもそれでは食卓に供するぶんのスープがなくなってしまいます。だからランダムサンプリングである「味見」を行うのです。同様に「対象が破壊されるまで強度を調べる調査」でも、ランダムサンプリングはよく用いられます。

「世論調査」では、国民全員を調査することは困難です。有限の人数ですから原理的には不可能とはいえませんが、そのためにかかるコストや時間を考えるとランダムサンプリングを行うほうがはるかに良いでしょう。

ランダムサンプリングの注意点

ランダムサンプリングでは、**全体は適切か**という点に注意が必要です。世論調査の例では「ランダムに電話番号を**生成する**」という方法(RDD

法：Random Digit Dialing法)が一般的です。この方法は「ランダムに番号を電話帳から**選択する**」とは違います。もし電話帳から選択する方法をとると、電話帳に番号を掲載していない人は対象外になってしまい、調査結果に偏りが生じてしまうからです。

ただし「ランダムに電話番号を生成する」という方法でも、電話を持たない人は対象外になっていますね。極端な例としては、この方法で電話の普及率を調べることはできません。

また、**無作為に選択しているか**という点にも注意が必要です。選択の際に調査する人の「作為」が入り込んではいけません。スープの味見をする前に、よくかき混ぜよということですね。

また、ランダムな数が必要な場合には、サイコロやコンピュータのプログラムなどで機械的に作る必要があります。人間はランダムな数列を頭で作り出すことは得意ではないからです。

社員の意識調査に生かす

会社の経営者は、自社の状況を常に把握したいと思っています。経営指標として現れるものはさておき、社内の雰囲気や社員のモチベーション、抱えている問題を把握するのは数字として表しにくく調査も難しいでしょう。

そこでもランダムサンプリングが使えます。ランダムに少数の社員を選び、その社員に対してインタビューなどを含めた深い調査を行うのです。社員全員に対して行えないような深い調査でも、ランダムサンプリングで人数を少なくすれば実行が可能になります。調査にかけられるコストや時間に応じて、柔軟に対応することも可能です。

本の購入に生かす

本の購入のような日常の生活にもランダムサンプリングは生かせます。たとえば、書店で技術書を買うかを判断するとしましょう。とても厚くて高い技術書なので買うか迷います。目的に合うだろうか、読みこなせるだろうか、知っていることだけなら無駄になる……そのように考えますね。

本のすべてのページを調査できれば確実ですが、それには時間もかかりますし、そもそも店頭でそんなに立ち読みはできません。そこでランダムサンプリングです。

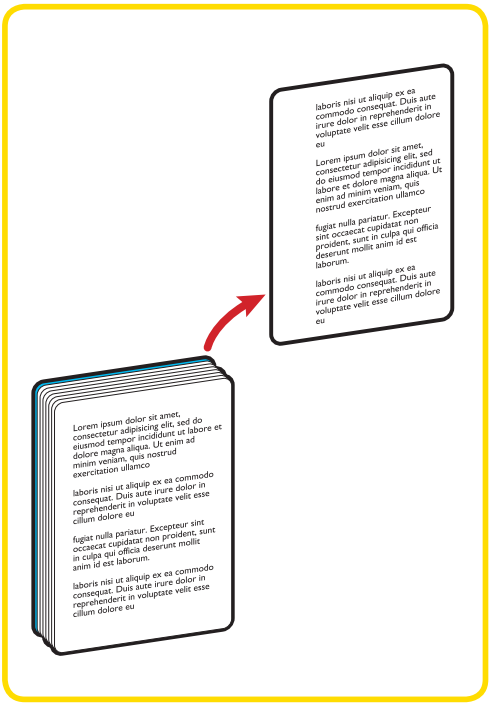
ランダムなページを開き、そのページに書かれていることをじっくりと調べるのです。目的に合うか、やさし過ぎないか、難し過ぎないか……本全体を調べる代わりに、ランダムに選んだページに調査の労力を注ぐのです(**図2**)。

本の全体像をつかみ、カバーしている範囲を知るためには目次や索引を見るのも良い方法です。しかし、目次を見るだけでは肝心の本文はわかりません。そのためにランダムサンプリングは良い方法です。

◆ ◆ ◆

あなたの周りには、数が多過ぎて取り扱いにくいものはありますか。ランダムサンプリングが使えないか、ぜひ考えてみてください。SD

▼図2　本を調査するランダムサンプリング



enchant ～創造力を刺激する魔法～

(株)ユビキタスエンターテインメント　清水 亮　SHIMIZU Ryo
URL http://www.uei.co.jp

第8回 プログラマが主役になるコンテスト『9leap』

2011年4月。enchant.jsがリリースされました。

そして5月1日からスタートしたHTML5ゲームコンテスト「9leap(ナインリープ)」には、開始後わずか10日間で100本もの作品が投稿されました。そして2013年現在、投稿されたゲーム作品は述べ3,000を超え、いまだに投稿され続けています。Yahoo!ゲームとの連携もスタートし、スマートフォンのYahooポータル画面には大手メーカーが作ったゲームと並んで、9leapの作品が紹介されています。

「give mac」キャンペーン

enchant.jsが出来上がりつつあった2011年の4月、僕はこれをどうしていこうか思案しました。長年この業界で生きてきた僕からみても、若い才能たちが集まって創りだしたenchant.jsは非常におもしろく、楽しいものに仕上がっていました。この楽しさをどう伝えて行くか……。

実は9leapのスタートにさかのぼる数ヵ月前、僕はブログ上である実験を行ったのです。それは「give mac」キャンペーンと銘打ったものでした。または「寄付自慢コンテスト」とも呼びました。当時、ちょうどタイガーマスクの伊達直人を名乗る人が、孤児院にランドセルを寄付していて話題になっていたのです。

日本には寄付文化が根付いていないことを常々疑問に思っていた僕は、日本でもやり方によってはこういう寄付が成り立つのかということにとても刺激を受けました。なぜ寄付をするのか——日本人の美徳として、寄付などの善行は人知れず行うもの、という考え方があります。この考え方はそれはそれで一理あると思います。しかし、あえて寄付をしたぞと喧伝することで人々を刺激し、また、寄付をすることが"カッコイイ行為"だとみなされる欧米のような価値観が広まれば、日本の閉塞した状況を解決できるのではないかと思ったのです。

ワルシャワで開催された世界初のHTML5ゲームカンファレンスで9leapを紹介する筆者

そこで、まずは実家のある新潟県長岡市の孤児院に寄付を申し出ました。快諾いただいたので僕は2台の新品のノートPCを寄付し、それをブログに書きました。すると、はてなブックマークでブーイングを受けたのです。僕は相手に確認をとって寄付をしているのに、それ自体が迷惑行為だと指摘されることに非常に疑問を覚えました。

そこで次に、「欲しい」と言っている青少年に小論文を書いてもらい、その内容によってMac Book Airをプレゼントする「give mac」キャンペーンというものを思いつきました。条件は18歳以下の学生であること。というのも、中高生でプログラミングやコンピュータを活用した仕事に興味があっても、今の子供は昔の子供よりもコンピュータを買い与えてもらう、あるいは譲り受ける機会が減っていると感じていたためです[注1]。結果的に、コンピュータリテラシーを学ぶ機会がケータイやスマートフォンに限られてしまいます。これでは未来のプログラマは育っていきません。

このgive macキャンペーンには71名の応募があり、ほかに志に賛同してくださった方からの寄付も含めて、合計6名の方々にMacBook Airや図書カードをプレゼントしました。

このキャンペーンをやって得た印象は、僕の想像以上に、21世紀の少年たちは自分専用のPCを持っていないということでした。これは幼少期を自分専用のPCと過ごしてきた僕にとってはとても大きな衝撃でした。ということは、21世紀には僕と同じ年齢で僕と同じキャリアを持った少年はほとんど現れないということになってしまいます。

もっと効率的に少年少女たちの才能を見抜き、持続的にPCを配る方法を考えたい、と思いました。また同時に、審査も1つのテーマや小論文でやると優劣がつきにくく非常に難しいということも解ってきました。ですがこの子供たちの前向きなパワーは、どこかうまく刺激してやればすごい成果を産み出してくれそうな予感がしたのです。

僕はスポンサーを募り、MacBook Airを毎月1台ずつ少年たちに効率的に分配する方法を模索し始めました。give macキャンペーンを単発のイベントで終わらせず、コンピュータの未来を描き出す少年少女たちを鼓舞しつづける方法が何かないだろうか。そう思ったのです。

コンテスト・ドリブン・プロジェクト

僕の頭のなかにうっすらとイメージされたのは「ロボカップ」でした。ロボカップとは、1997年から毎年開催されているロボットによるサッカー大会です[注2]。僕が大学に入学したのが94年ですから、まさにロボカップがスタートするそのときに大学生をやっていました。

当時、僕の恋人は電気通信大学の機械工学科に通っていました。彼女が「今度、すごい大会が開催されるらしい」と何かのニュース記事のようなものを持ってきました。

――西暦2050年までに、人間のサッカーの世界チャンピオンチームに勝てる、自律型の人型ロボットチームを作る――

そんな馬鹿な、と僕は目を疑いました。97年に2050年といえば、半世紀後です。半世紀後を目指して行うコンテストなんてものが成立するのかと度肝を抜かれました。ロボカップの目的は、こうした具体的な目標を提示することで、人々の目を同じ方向へ向け、そこに対して必要な課題を洗い出して行くというものでした。

これは凄いアイデアだ、と僕は唸りました。こういうものをコンテスト・ドリブン・プロジェクトと呼ぶのだそうです。F1レースが自動車の世界の進歩に貢献するのと同様、ロボカップもロボットの世界の進歩に貢献することを目指して企図されたものでした。

では僕はgive macキャンペーンを通じて何がしたかったのでしょうか。自分なりに考えてみると、僕は、未来の世界を創りだすプログラマやクリエイターを発掘したかったのだ、と思

注1）現在コンピュータの世界で活躍している人は子供のころ、親が買って投げ出し、片隅で埃をかぶっているPCを事実上の自分専用機として使えた、という状況に多かれ少なかれあったのではないかと思います。一方、今の親からすると、コンピュータはネットやゲームといった子供の気をそらせるものが多すぎて、自由に使わせることを避ける傾向にあるのではないでしょうか。

注2）現在ではサッカーだけでなく、災害救助を想定した「レスキュー」、日常生活を想定した「@ホーム」、小学生から参加できる「ジュニア」などが行われている。

い至りました。ならば、僕はプログラマを育てるようなコンテストをやろうと思いました。しかも、単にプログラミングの技巧だけを競うコンテストではなく、ドラマが生まれるようなコンテストを作りたいと思いました。

ゲームプログラマの高校野球

そこで考えた題材が、ゲームです。

ゲームは、プログラマだけでなく、企画、グラフィッカー、サウンド、シナリオ、すべてが重要な総合芸術です。つまりチームプレイができるということです。個人技ではなく、チームがあればドラマが生まれます。そこで出会いや別れ、葛藤もあることでしょう。

なお良いのは、ゲームは、いくらでもテーマが設定できることです。テーマは、いわばコンテストの選考基準です。選考基準がなければ選べません。また、選考基準が常に画一的では、単に記録を争うだけのものになってしまいます。その瞬間、瞬間の盛り上がりが生まれ、ライバルが生まれ、知略を巡らし、勝利をつかみ取る——そういう名場面を演出するには、画一的、数値的な審査基準ではなく、総合的な審査基準で厳しく判断する、ゲームのような総合芸術になっていたほうが望ましいと思いました。

アマチュアがゲームをつくる、となれば、主役はプログラマです。ゲーム開発において、プログラマはピッチャーのようなものです。彼がいい球を投げなければ、試合には勝てません。プログラマが主役になれるような競技を作りたかった、というのも1つの大きな理由です。

僕はスポンサーを探すべく、まず真っ先に旧知のD2C[注3]を訪れました。D2Cの本間さんと会った僕は、プログラマが主役になるようなスマートフォン向けゲーム開発コンテストを提案します。D2Cは独自に「アプリ甲子園」というアプリ開発を競わせる大会を企画していたのですが、本間さんは「話はよくわかった」と仰ってくださり、なんとアプリ甲子園とは別に、9leapも共同で主催することに同意してくださいました。

「それで清水さん、そのコンテストの名前は何にしましょう？」

「ナイン・リーダーズ・エデュケーション・アンド・アチーブメント・プログラム、略して9leapとしたいと思います。9はあと1足せば桁が変わります。頭字語のleapは大いなる飛躍、大ジャンプを意味します。このコンテストをきっかけに若い才能が花開き、大きく飛躍していく、そういう願いを込めた名前です。そしてナインは野球チームも意味します。一人ではなく、みんなで力を合わせればもっといいものが作れる。それを学ぶ場としての大会でもあります」

こうして9leapはスタートしました。

9leapへの挑戦を通じて、若きプログラマがいろんな仲間と出会い、青春の汗を流す。そうして作った成果物が、受賞作としてHTML5で配布され、級友や後輩たちに遊んでもらえる。それ自体がさらに周囲を刺激し、9leap全体を盛り上げていく……。そんな世界の実現を目指して、プログラミングテクニックを競うのではなく、あえてゲーム開発そのものをコンテストにしてみました。その結果、9leapは非常にユニークな存在になれたと思います。

enchant.jsと9leap

9leapについての構想を温めているころ、別の場所では田中諒がenchant.jsを開発していました。HTML5をベースとしたこの2つのプロジェクトが相互に強い関連性を持つことになったのは、むしろ必然と言えました。そしてenchant.jsの機能を充実させ、HTML5ベースのゲーム開発をより簡単にすることで、相乗的な効果を上げることをねらいとしました。

注3）D2Cはドコモと電通のジョイントベンチャーで、世界で最初の携帯電話専業の広告代理店。

enchant.jsを学習すると、その先には9leapへの挑戦が用意されているのです。これはenchant.jsを学習するモチベーションエンジンとして大きく寄与しました。ちょうどObjective-CやiOSを学ぶことが、その先にApp Storeでの販売というゴールが見えているのと似ています。

9leapそのものはenchant.jsのためのコンテストではありません。実際、受賞作の中にはenchant.jsを使っていないものもあります。それでいいのです。enchant.jsはあくまでHTML5ゲームを作るうえでの選択肢の1つであり、プログラマは自由に自分の作ったライブラリを選択することができるようになっています。

これはenchant.js自体が他のHTML5向けのゲームエンジンと競う場でもありました。ライバルがいなければさらなる高みを目指すことはできません。Processing[注4]やtmlib.js[注5]など、ほかにも優れたエンジンが多く使用されました。

「勝負できる舞台が欲しかった」

そして2011年5月1日から9leap第1回大会がスタートします。驚くほど凝ったゲームが数多く投稿され、僕は個別の作品の作者に時間を作っては会いに行きます。

第1回大会で最優秀賞を受賞した一人の少年に僕が会うと、実はこれは同じ大学の研究室の同級生との合作だと言います。プログラマと企画、グラフィック、そういう分担で作ったのだと。「どうしてこのコンテストに応募しようと思ったの？」

僕はそう聞かずにおれませんでした。すると彼らは答えました。

「自分たちの力を試したかったんです。しかし勝負する場所がなかった。誰かほかの人たちと勝負できる舞台が欲しかった。それが9leapでした」

9leapとenchant.jsチームの開発風景

「合コンクエスト」の作者、戸谷さんがARCに来訪したときの写真

このときほど、僕はこのコンテストを実施してよかったと思ったことはありませんでした。今は彼らももう立派な社会人として社会で活躍しています。若い人たちにそういうきっかけを与えられたことを、僕は素直に嬉しく思います。

さらに多くの人々へ

enchant.js、そして9leapによって、僕たちはゲーム開発を簡単にし、しかもそれに情熱を傾けてもらうモチベーションエンジンの構築にも成功しました。こうなると、僕たちは必然的にその先を考えることになります。より広く、より多くの人にプログラミングの楽しさを知ってもらい、ものを作る感動を知ってもらう、そのためにはどうすればいいのか。

考え抜いた僕たちはついには、新しいプログラミング言語を作り出すことになります。SD

注4）Casey Reas氏とBenjamin Fry氏によるオープンソースプロジェクトで、画像処理やアニメーションといったビジュアルデザイン、インタラクションデザインをしやすくしたプログラミング言語および統合開発環境。

注5）phi氏が開発したJavaScriptライブラリ。「JavaScriptをより使いやすく、より便利に、そしてより豊かに」をモットーとして開発されている。

コレクターが独断で選ぶ!

偏愛キーボード図鑑

株式会社 創夢
濱野 聖人 HAMANO Kiyoto
khiker.mail@gmail.com
Twitter：@khiker

第8回

KVMスイッチでお気に入りのキーボードを無線化しよう

CS533

写真1　CS533

はじめに

今回は、本筋のキーボードとは外れますが、USB to Bluetooth KVMスイッチであるCS533（**写真1**）を紹介します。CS533はPCとiPhone/iPadで1つのキーボード／マウスを共有するための製品です。今回はこのCS533を用いて、USBキーボードをBluetooth化して使う方法を取り上げます。

CS533

CS533はATEN社が販売する製品でKVMスイッチの1種です。一組のキーボードとマウスを、PCとiPhone/iPad間で切り替えて利用できるようになります。先に海外で販売されていました。現在では、日本でも販売されているため技適マークもついています（**写真2**）。

写真2　技適マーク

利用方法

CS533はPC、キーボード／マウスを次のように接続します（**図1**）。

- CS533とキーボード／マウスをUSB接続
- CS533とPCをUSB接続
- CS533とiPhone/iPadをBluetoothでペアリング

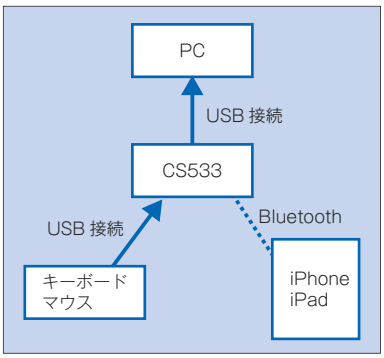


図1　CS533の接続例①

PCとiPhone/iPadとの切り替えは、CS533にあらかじめ設定されているホットキーで瞬時に行えます。ホットキーはいくつか存在し、PCとiPhone/iPadを切り替えるキーは Alt Alt（Alt の2度連打）です。主要なホットキーを**表1**に挙げます。

筆者は、iPhone/iPadは所持していませんので、その代替としてSurfaceRTを用いて試してみましたが、スムーズにPC、Surface間を切り替えでき、とくに問題を感じませんでした（**写真3**）。

なお、ホットキーなどの詳しい情報は製品のマニュアル[注1]を参照してください。

Alt Alt	別のデバイスにスイッチする
Alt ＋ F1	USB接続のPCにスイッチする
Alt ＋ F2	Bluetooth接続のデバイスにスイッチする
Alt ＋ F6	Bluetoothペアリングをクリアする
Alt ＋ F12	Bluetooth接続を切断する

表1　ホットキーの一例

USBキーボードのBluetoothキーボード化

CS533のUSBに接続するデバ

注1）http://www.atenjapan.jp/products/productItem.php?model_no=CS533

イスはPCである必要はなく、給電ができれば何でも良いです。そのため、ポータブルUSB電源を代わりに使うこともできます。

これをふまえて、次のようにiPhone/iPadの代わりにPCを接続して、USBキーボードをBluetooth化して使えます(**図2、写真4**)。

- CS533とキーボード／マウスをUSB接続
- CS533とポータブルUSB電源をUSB接続

写真3　CS533とSurface RT

写真4　CS533とHHKB

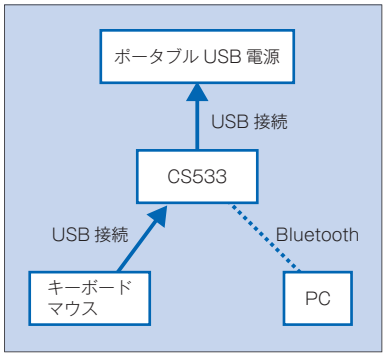


図2　CS533の接続例②

- CS533とPCをBluetoothでペアリング

PCとCS533をペアリングしたあとにキーボードで Alt ＋ F5 、 1 、 Enter と入力してキーボードマッピングをiPhone/iPad用からQwerty配列に変更する必要があります。これを行わなければ、 Esc や Home 、 End などの一部のキーが使用できません。

以上で、USBキーボードをBluetoothキーボードとして使用できます。ただし、次の点に注意する必要があります。

①Alt を2度連続して押すと、USB接続されたデバイスとBluetooth接続されたデバイスの切り替えが実行される
②一部の Alt ＋ファンクションキー操作が使えない
③メニューキーが使えない
④日本語キーボードと接続した際、 変換 、 無変換 、ひらがなカタカナキーが使えない[注2]

①は、もう一度 Alt を2度押すと使えるようになります。どうしてもキーボード入力ができなくなった場合、CS533と接続している電源を引き抜き、再接続を行ってください。

②は、CS533にホットキーとしてキーストロークが奪われてしまうためです。一例は**表1**で挙げたキーストロークです。 Alt とそれ以外の組み合わせである Alt ＋ a 、 Alt ＋ Tab などは使えます。また、 Alt ＋ F4 はホットキーとして登録されていないので、これも使えます。

③は、試した限りでは「=」が入力されるだけでした。

④は、Linux PCで試したところ、 変換 、 無変換 、ひらがなカタカナキーはまったく別のキーとして認識されているようでした。それ以外のキーは不都合なく入力できました。

入手方法

Amazonなどで購入できます。値段は7,000~8,000円ほどです。eBayのようなオークションサイトで出品されていますが、海外のCS533では技適マークがついていない可能性があるので、オークションでの購入は控えたほうが良いでしょう[注3]。

◆ ◆ ◆

USBキーボードをBluetooth化したいという要望はよく聞きますが、それを達成するための製品は今までありませんでした。有線キーボードを無線化するという観点であれば、ワイヤレスUSBハブとポータブルAC電源を利用するという方法もありましたが、これは大がかりになりがちでした。CS533なら多少制限はありますが、比較的簡単にUSBキーボードをBluetooth化できます。ワイヤレスキーボードの購入を考えるのであれば、今回紹介した制限が許容できるのかどうかを考えたうえで、CS533で使い慣れたキーボードをBluetooth化することを考えてみても良いかもしれません[注4]。SD

注2）半角／全角 は使えます。
注3）日本販売前に海外でのみ販売されていたCS533に技適マークはついていませんでした。
注4）なお、海外ではCS533のほかにBluetooth Keyboard Adapterという製品もあるようです。http://handheldsci.com/kb

秋葉原発!

第38回

はんだづけカフェなう

Maker Faire Rome

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/

はじめに

去る10月3日から6日にかけ、ローマのPalazzo dei Congressiという会議場で、Maker Faire Romeが開催されました。イタリアといえばArduino誕生の地で、Arduino Teamがパートナーとなってかなり力の入ったMaker Faireでした。**写真1**はセットアップの日に撮影したものですのであまり人が写っていませんが、会期中は本当に多くの人で混雑し、発表された来場者数は3万5千人ということでした。

イタリアはいろいろと寛容な国のようで、犬がテクテクと飼い主と一緒にリードなしで歩いていたり（**写真2**）、会場の外の至るところで喫煙をしている人がいたりと、たいへん混雑している中でも、とても自由な空気のMaker Faireでした。

筆者にとって、初めてのヨーロッパでMaker Faire以外に観光も楽しんできてしまったのですが、そんなMaker Faire Romeで見てきたものを紹介させていただきます。開催初日の3日はカンファレンスだったのですが、この日に新たなArduinoのボードが2つも発表されました。GalileoとArduino TREです。

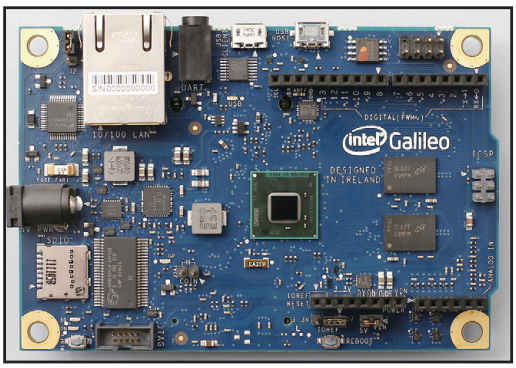
▼写真1　Palazzo dei Congressi

▼写真2　犬の来場者

Galileo

Galileoは、Intelが出すArduino Certifiedなボードです（**写真3**）。先だって発表されていた、Intel Quark X1000という低消費電力プロセッサが搭載されています。このQuark X1000は、32bitのPentium命令セットという懐かしい仕様ですが、Atomの1/10程度の低消費電力が特徴です。また、SoC（System-on-a-Chip）ということで使うために必要になる周辺回路が少なくなっています。写真を見ると、

▼写真3　Galileo



PentiumクラスのCPUが載っている基板なのに、真ん中のCPUと、その右のDDR3メモリ、Ethernetコネクタの左のPHY（Ethernetの物理層を担当するチップ）くらいしかx86のマザーボードを構成するチップがありません。

ボード自体はLinuxが走るようになっており、Arduinoのスケッチはユーザランドで動くアプリケーションにコンパイルされます。“Certified”ということでArduinoそのものではないためか、IDEは従来のArduinoのものとは別にIntelが配布するものを使用するようになっています。もちろん、従来のArduinoのシールドを搭載できるようにコネクタが配置されています。Arduinoといえばたいてい5Vですが、この連載でも触れているように最近の半導体は3.3Vのものが多くなってきています。Galileoは5Vと3.3Vが切り替えられるようになっており、とても便利そうです。このボードは11月下旬あたりをターゲットに発売される模様です。

Arduino TRE

Arduino TREの“TRE”はイタリア語で3を意味します。Uno、Dueがそれぞれ1、2を意味し、つまり3番めのArduinoということになります。TREは、1つのボードの上に2つのArduinoが搭載されているという変わり種です。1つはATmega32u4という、従来のArduino Leonardoにも搭載されていたプロセッサで互換性を確実なものとしています。もう1つはSitara AM335xというARM Cortex-A8のプロセッサで、従来のATmega32u4などの100倍程度の処理能力を有します。かなり乱暴な表現ですが、Raspberry PiとArduino Leonardoを1つのボードに搭載したようなものです。

来年の春にリリースされるという話のTREですが、会場でデモが行われていました（**写真4**）。HDMIがあるのでモニタをつなぎX Windowを動かしていました。

▼写真4　Arduino TREのデモ

Open Wheels

会場にいると、向こうからセグウェイのような乗り物に乗っている人が現れました（**写真5**）。聞くと、出展しているFutura Group srlというイタリア企業[注1]の人たちが作ったオープンソースのセグウェイクローンだそうです。まだプロトタイプということで、ソースは公開されていませんが、近日中に公開するとのことでした。

注1）　http://www.open-electronics.org/

▼写真5　Open Wheels

筆者も操縦させてもらいましたが、スムーズに動き、かなりの完成度でした。ソースが公開されたら、Maker Faire会場内用に自分の分を作ってみたくなっています。

OSVehicle

OSVehicle[注2]は、その名のとおりオープンソースの自動車です(**写真6**)。CC BY-SA[注3]ライセンスで公開されています。図面も公開されていますし、車の部品もそれぞれコンポーネントごとにオンラインで販売されています。電気自動車にするためのバッテリとモーターなども販売されています。筆者も……と思いましたが、組む場所はおろか、走って遊べるような私有地がありませんので断念しました。

The light cryptalk

The light cryptalk[注4]は、送信機と受信機の間で暗号化した光通信を行える装置です(**写真7**)。ご存じの読者も多いでしょうが、第二次世界大戦中にナチス・ドイツが用いたエニグマという暗号装置と同じ換字式の暗号方式を採用しています。Arduinoを使って作られているのですが、驚くべきはこの装置を14歳のイタリア人の少年、Lizzit君が独力で作ったということです。

Lizzit君は英語が話せないので、一緒に出展をしていたおじさんに通訳してもらいながら会話をしました。おじさんがArduinoをLizzit君にプレゼントしたことがきっかけでハードウェア開発を始めたそうです。話を聞いていて、筆者はなんだかとてもうれしい気持ちで一杯になったのですが、Make:誌の編集者も同じような感想を抱いたようで、彼と彼の作品は受賞していました。写真に写っている青いタグがその証です。

3Dプリンタ

3Dプリンタブームは世界中で起きています。Maker Faire Romeでも数多くの3Dプリンタが出展されていました。とくに筆者の印象に残ったものをいくつか紹介します。

写真はWASProjectという3Dプリンタを販売している企業が展示していた、人の身長よりも高い巨大な3Dプリンタです(**写真8**)。これは売り物ではなく、同社の3Dプリンタが売れて得られた利益を、このプリンタに投入してより巨大なプリンタにアップグレードしていくということでした。このプリンタは材料を出すノズルが上から複数のアームで支えられているパラレルリンク機構というものを採用しています。

会場を見ていると3色のフィラメントで出力をしているプリンタが目につきました。2色は

注2) http://www.osvehicle.com/
注3) クリエイティブ・コモンズ・ライセンス(表示-継承)。
注4) http://www.lizzit.it/

▼写真6 OSVehicle

▼写真7 The light cryptalk

たまに見かけますが、3色は初めてです（**写真9、10**）。Immagina e Crea[注5]が参考出品しており、Webに詳細な情報が載っていないかと期待して見てみたのですが、今のところこのプリンタに関する詳しい情報は見つけられませんでした。

筆者はアルミフレームを使った3Dプリンタを作りたいと思っているのですが、そんな3Dプリンタも見かけました。Hackerbot[注6]は、OpenSourceHardware.itというFabLab[注7]ピサに近しい人々のグループから生まれたプロジェクトです（**写真11**）。オープンソースで、このプリンタのソースはGitHubで公開されています。筆者は手元で抱えているプロジェクトが片付いてきたら、部品を集めてHackerbotを組んでみようと考えています。

来年も

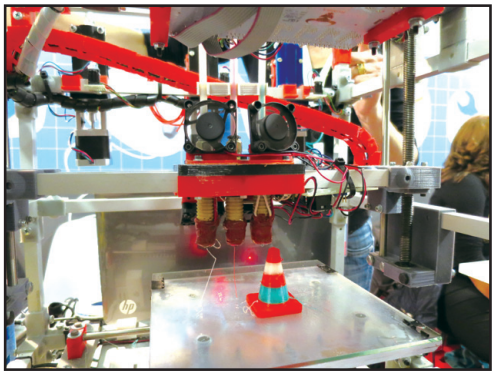
誌面の都合で紹介できませんでしたが、ほかにもおもしろいものが山盛りのMaker Faire Romeでした。どうやら来年も開催する予定のようですので、今回の旅程では訪れることができなかったフォロ・ロマーノ（古代ローマの遺跡）も見がてら、来年も参加したいと思います。

SD

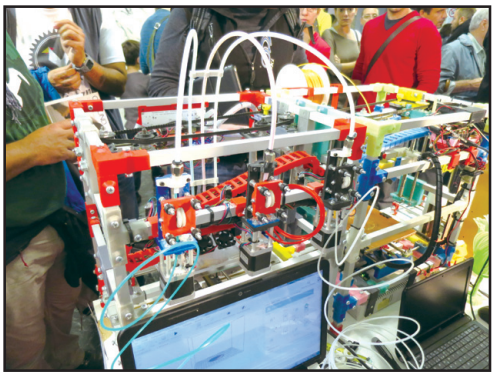
注5） http://www.immaginaecrea.it/
注6） http://hackerbot.org/
注7） Fabrication Laboratory。世界中に存在し、多様な工作機械を備えた市民のための工房（ラボ）。

▼写真8　巨大3Dプリンタ

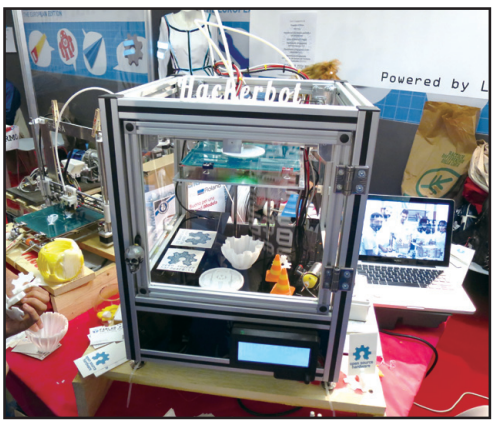
▼写真9　トリプルホットエンド



▼写真10　トリプルエクストルーダー



▼写真11　Hackerbot



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 12 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 My Book 2TB

1名

動画、音楽ファイル、写真の保存に最適な USB 3.0/2.0 対応の大容量外付けハードディスクドライブです。自動バックアップソフトウェア「WD SmartWare Pro」を搭載しており、My Book ドライブにバックアップできるだけでなく、Dropbox と連携してクラウドへのバックアップも可能です。

提供元 ウエスタンデジタルジャパン
URL http://www.wdc.com/jp

02 ウイルスバスター クラウド 1年版

2名

SNS での適切なプライバシー保護設定をお勧めする「プライバシー設定チェッカー」が Facebook に加え、Twitter、Google+ にも対応。リアルタイムスキャンの性能も向上しました。Windows 8.1/8/7/Vista(SP2 以上) /XP(SP3 以上)、Mac OS X Lion/Mountain Lion に対応。

提供元 トレンドマイクロ **URL** http://www.trendmicro.co.jp

03 エルゴノミクス アームレスト MR-TOKERG1

1名

椅子の肘掛けや机の端に取り付けて使うアームレスト兼マウスパッド。肘を肩から自然に下ろしたリラックスした姿勢でマウスを操作できます。本誌 p.183「エンジニアの能率を高める一品」で試用した品をプレゼントとして提供します。 ※本製品にマウスは付属しません。

提供元 サンワサプライ **URL** http://www.sanwa.co.jp

04 マスタリング TCP/IP OpenFlow 編

5名

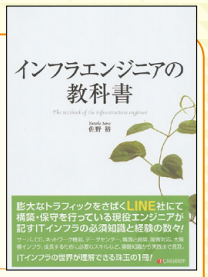
あきみち、宮永 直樹、岩田 淳 著／
B5 判、264 ページ／
ISBN = 978-4-274-06920-8

OpenFlow プロトコルの全体像を、OpenFlow 1.0 から 1.3.2 まで解説しました。プロトコルとメッセージだけにとどまらず、システムを構築するうえでのパターンについても解説します。

提供元 オーム社 **URL** http://www.ohmsha.co.jp

05 インフラエンジニアの教科書

2名



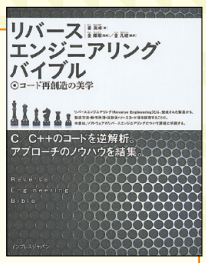
佐野 裕 著／
A5 判、184 ページ／
ISBN = 978-4-86354-133-7

膨大なユーザを抱える LINE のインフラの管理／運用に携わっている著者が、インフラエンジニアの仕事内容、求められる知識やスキルについてわかりやすく解説します。

提供元 シーアンドアール研究所 **URL** http://www.c-r.com

06 リバース エンジニアリング バイブル

2名



姜 秉卓 著、金 凡峻 訳、金 輝剛 監修／
B5 変形判、416 ページ／
ISBN = 978-4-8443-3479-8

リバースエンジニアリングは、セキュリティ分野や悪性コードの分析だけではなく、開発にも欠かせない技術。本書では C/C++ をベースにリバーシングからアンチリバーシングまでを徹底解説します。

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

07 レベルアップ Objective-C

2名

沼田 哲史 著／
B5 変形判、360 ページ／
ISBN = 978-4-7741-6076-4

Xcode 5 を効果的に利用するための技術を解説。Xcode 5 でのデバッグ、リファクタリング、バージョン管理や、最新の Objective-C によるデータ処理、Core Data、iCloud、メモリ管理などを取り上げます。

提供元 技術評論社 URL http://gihyo.jp

第1特集

SDN/OpenFlowの流れを総まとめ!

SDN/OpenFlowで幸せになれますか?

▶▶▶ 仮想ネットワークをどう使うか ◀◀◀

昨年よりもSDN/OpenFlowの認知度が上がり、ビジネスへの適用もジワジワと進みつつあります。概念的だった技術がさまざまな機器やアプリケーションソフトウェアとして実装されていくと、それにともない現実で起こりうる問題も、現場レベルでようやく想像がつくようになってきます。本特集では、SDN/OpenFlowの新技術の流れを紹介します。そして現場のエンジニアが何を考え、SDNにどのように対処していくのか語ってもらいます。SDN/OpenFlowのメリット・デメリットをあらためて確認することで、主体的に技術に取り組む手がかりを得られるようになるでしょう。幸せになれるかは、使いこなし次第!

CONTENTS

CHAPTER 1

仮想化通信技術の発展 SDN/OpenFlowとNFVの最新動向

Writer 佐宗 大介(さそう だいすけ)／オープン・ネットワーキング・ファウンデーション(ONF)

OpenFlowの登場に端を発したSDNの潮流は、いまやOpen Networking Foundation(ONF)を中心に基盤技術の標準化が行われるまでに成長しました。また、最近ではNFVという新たな概念も出てきています。ONFの活動を見ながら、現在のSDNの状況について俯瞰してみましょう。

SDNの概要

この数年、従来のネットワークの構造では、複雑化する通信事業者や企業の要求を支えられなくなってきました。そんな中、SDNやOpenFlowが新しい通信技術や通信フレームワークとして注目を集めています。また、Open Networking Foundationが業界を代表してSDNに取り組むなど、SDNがネットワークアーキテクチャを変えつつあります。

SDNとはSoftware Defined Networkの略であり、新しいネットワーク構造のフレームワークです。SDNはそれぞれのネットワーク機器内に実装されていたデータ転送と制御を分離しました。これまでネットワークデバイス上にあった制御の機能を外部のコンピューティングデバイスへと移行することによって、ネットワークのインテリジェンス(サービス収入を拡大し運用を簡素化するための、ネットワーク機器の管理／制御)がコントロールレイヤに一元化されました。アプリケーションやポリシーエンジンから見れば、インフラストラクチャレイヤがバーチャルな1つのエンティティとして扱えるようになったのです。

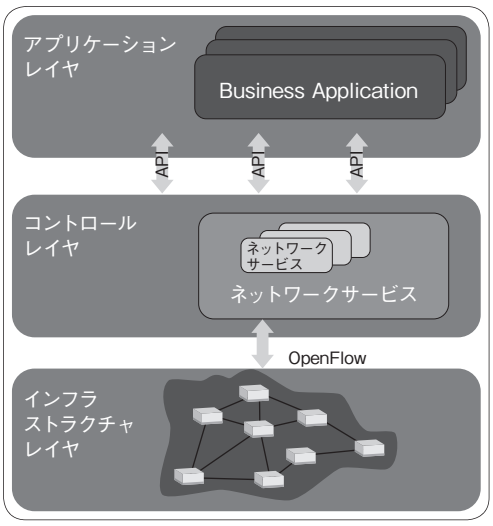
その結果、上位のアプリケーションやネットワークサービスに対してデータ転送機能が抽象化され、アプリケーションからネットワークの機能を制御できるようになり、ネットワークへのプログラマビリティが実現しました。

そして、複雑なネットワークの設計や制御により付加価値の高いサービスを展開していたサービスプロバイダのサービス競争力と収益性を向上するしくみとして、このSDNが注目を集めています。

SDNのフレームワークは、3つのレイヤ(層)と、その3つのレイヤをつなぐ2つのインターフェースから構成されています[注1](**図1、表1**)。

SDNフレームワークの3つのレイヤ間には2

▼図1 SDNのフレームワーク



注1) ONF発行のホワイトペーパー『Software-Defined networking: The New Norm for Networks』(https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf)より。

▼表1　SDNフレームワークの3つのレイヤ

レイヤの名称	役割
アプリケーションレイヤ	機能やサービスとしてのアプリケーションが動作
コントロールレイヤ	おもにSDN/OpenFlowコントローラとして、ネットワークサービスを実現する
インフラストラクチャレイヤ	データ／フロー転送を行うネットワーク機能

つのインターフェースが存在します。コントロールレイヤとインフラストラクチャレイヤの間のインターフェースは、サウスバウンドインターフェース(アプリケーションでは「サウスバウンドAPI」)と呼ばれています。ここにはベンダ独自のプロトコルや標準化されたプロトコルがあります。SDNとOpenFlowがよくセットになって登場しますが、OpenFlowはサウスバウンドインターフェースの役割を担い、最初に標準化されたプロトコルです。

一方、アプリケーションレイヤとコントロールレイヤの間は、ノースバウンドインターフェース(アプリケーションでは「ノースバウンドAPI」)と呼ばれています[注2]。

◆ ◆ ◆

SDNの登場で、アプリケーションやサービスはネットワークの実装にまつわるさまざまな束縛から解放されます。そして、これからはSDNでアプリケーションに合わせてリソースを最適化したネットワークも実現できると期待されています。このような新しいコンピューティングとネットワーキング環境実現への期待が後押しし、2017年には全世界でのSDN市場規模は244億ドル(約2兆4,000億円)にまで成長すると見込まれています[注3]。

SDNにおけるOpenFlowとONFの役割

Open Networking Foundation(以下ONF)はユーザ主導の団体であり、オープンな標準化活動を通じてSDNの利用促進とプロモーションを行っています。企業におけるネットワーク運用に劇的な変化をもたらすSDNとその基盤技術の普及と商用化が、ONFの掲げるミッションです。

ONFはオープンで利用者の視点に立った共同開発プロセスを重視します。ONFのワーキンググループではSDNに対する要望を分析し、OpenFlowの商用利用のためのニーズや、SDNをより活かすための新しい標準の必要性に関するリサーチを進めています。ONF発足当時は23のメンバー企業でしたが、現在では世界中から112のONFメンバーが参加しています。日本を中心に活動する企業も7つあります。

ONFの今日までの最も主要な業績は、転送プレーンをリモートからプログラミングできるOpenFlowを発表したことです。OpenFlowはSDNにおける最初の標準化プロトコルです。

ONFはOpenFlowの誕生と発展の歴史に深く関わってきました。OpenFlowは米国スタンフォード大学にて考案され、研究が進められてきました。その後2007年に、スタンフォード大学を中心としてOpenFlowコンソーシアムが立ち上げられ、OpenFlowスイッチの企画の標準化が開始されました。

現在の最新のバージョンは、OpenFlow 1.4です。2014年中のリリースを目指して、Extensibilityワーキンググループを中心としてOpenFlow 1.5への着手が開始されました。トンネルの取り扱いの向上、フローのローカルでの制御の改善、NFV(後述)が要求する機能への着手などの実現を目指しています(**表2**)。

OpenFlowコンソーシアムを引き継ぐかたちで、2011年3月にONFが設立されました。ONFはOpenFlowを含むSDNに関して、その商用化のサポートやプロモーションを行ったり、

注2) これらはとくにSDNに特化した用語ではありませんが、SDNの中心となるコントローラから見てそれぞれ上(北側)と下(南側)であることから英語ではこう呼ばれています。

注3) http://www.sdncentral.com/sdn-market-size/

SDNを利用したネットワーク構築を促進したりするための団体です。もともとのOpenFlowコンソーシアムはベンダを中心に構成されていましたが、ONFのボードメンバーはサービス事業者や通信キャリアなどの利用者を中心に構成されており、Yahoo、Google、Microsoft、Facebook、Verizon、Deutsche Telekomに加え、日本からはNTTコミュニケーションズが参加しています。ボードメンバーはONF内の決定事項の最終承認を行う重要な役割です。

ONFはボードメンバー、各ワーキンググループ、テクニカルアドバイザリーグループ(TAG)、チップメーカーアドバイザリーボードから構成されています。ベンダ中心ではなく、利用者側の視点や、ソフトウェアやアプリケーション、ネットワーク機器ベンダ、シリコンベンダなど幅広い視点を取り入れ、SDNの市場を活性化するうえでエコシステムを最大化かつ最適化する組織構成をとっています。

ONFの活動

ONFのワーキンググループでは、SDNにおけるフレームワーク、アーキテクチャ、標準化などの業界で最も重要な課題に取り組んでいます。2012年に「Forwarding Abstractions」と「Migration」が正式にワーキンググループとして承認され活動を開始しました。2013年10月には「Northbound API」と「Mobility & Wireless」が新たにワーキンググループとして認められました。その結果、現在10のワーキンググループがONFとして活動しています(**表3**)。

▼表2　OpenFlowのバージョン

バージョン	概要	策定日
OpenFlow 1.0	現在最も実装が進んでいるバージョン。シングルのフローテーブルと12のタプルを設定	2009年12月
OpenFlow 1.1	OpenFlow 1.0と互換性のあるバージョン。マルチテーブル、グループ、VLANとMPLSのサポート	2011年2月
OpenFlow 1.2	最初のONFからのリリース。1.1の改訂版。マッチパターンとリライトでの拡張やIPv6への対応	2011年12月
OpenFlow 1.3	ステーブルリリース(長期のサポートをコミット)。メター、PBBのサポート、イベントフィルタなど	2012年4月
OpenFlow 1.4	バンドル、オプティカルポート、フローのモニター、Eviction	2013年8月
OpenFlow 1.5	トンネルの取り扱いの改善、ローカルフローの制御の向上、L4/L7機能の追加など	2014年後半(予定)

▼表3　ONFのワーキンググループ一覧

グループ名	活動内容
Architecture & Framework	SDN全体のアーキテクチャの観点から解決すべき課題を明確にし、SDNの標準化をサポートする。L4-7についてもこのワーキンググループで検討中
Configuration & Management	OpenFlowプロトコルの標準化に伴う運用、管理(OA&M)の中核的な課題を解決する。OF-Configの仕様策定
Extensibility	OpenFlowの標準化と機能拡張を行う
Forwarding Abstractions	HAL(ハードウェア抽象レイヤ)の仕様検討／策定
Market Education	OpenFlowを基盤としたSDNと、ONFの標準への適合の推進
Migration	従来のネットワークからSDNへのスムーズな移行について検討する
Northbound API	SDNのアプリケーション開発を加速するため、ノースバウンドインターフェースのフレームワーク、モデル化
Optical Transport	オプティカルトランスポートネットワークにおいて、SDNやOpenFlowが標準ベースの制御へ適応することを検討
Testing & Interoperability	テストと認証の標準化方法を確立。PlugFestの開催などを通じてベンダ間の相互接続性を高める役割を担う
Mobility & Wireless	モバイル領域におけるOpenFlowの適応を検討

新設のワーキンググループ

Forwarding Abstractions では、OpenFlowなどのサウスバウンドインターフェースプロトコルのネットワーク機器への実装を簡素化したり、ハードウェアの性能を十分引き出すための機器実装モデル、フレームワークの策定を手がけたりするなど、インフラストラクチャレイヤのSDN環境への移行をスムーズに進める活動を展開しています。同時に、チップメーカーアドバイザリーボードを通じてシリコンベンダ各社との議論を行い、ハードウェアとソフトウェア環境がSDN市場においていち早く融合するような活動も実施しています。

Migrationでは、従来のネットワーク環境からSDN環境へ利用者がスムーズに移行するための手順と、それに必要なツールの提供を目指しています。キャンパスネットワーク、通信事業者のエッジ、データセンター間接続の3つの領域を中心として、移行のユースケースの策定と移行をサポートするツールを来年までに提供することが目標です。

Forwarding Abstractions と Migration は、SDN関連機器の提供者と利用者双方のネットワークの利便性を向上させるニーズが高まったことにより、ワーキンググループとして設立されました。まさにSDN市場の活性化が設立の背景にあります。

サーティフィケーションプログラム

2013年の7月にOpenFlow 1.0をベースとしたサーティフィケーションプログラムが発表されました。400以上のテスト項目を満たしている機器に与えられます(**図2**)。このサーティフィケーション試験は、インディアナ大学のInCenter、ニューハンプシャー大学、中国のBeijing Internet Instituteで実施できます。2013年10月16日に、NECのプログラマブルフロースイッチのPF5240とPF5248が世界で初めて、このテストにパスしました。

今後、新たに400以上のテスト項目を追加したOpenFlow 1.3をベースにしたサーティフィケーショングログラムを、Testing & Interoperabilityワーキンググループで開発しています。

SDNにおける新しい動き

ONFでは年に2回、すべてのワーキンググループのメンバーが一同に会する「メンバーワーキングデイ」と呼ばれるミーティングが開催されます。今年も10月8～9日にサンフランシスコ近郊にて開催されました。

今回も市場のニーズに基づき新たな2つのワーキンググループが承認されました。Mobility & Wirelessにはすでに55のONFメンバー企業から200人を超えるメンバーが議論に参加しています。携帯通信事業者のパケットコアやバックホールへのSDNの適応や、企業や大学のキャンパスネットワークでのSDNの活用と適応についての議論が始まっています。まずはこれらの領域におけるユースケースの策定をしながら、必要な機能の抽出から着手していきます。すでに15のユースケースが候補としてあがっています。SDNのモバイル環境への適応の期待の高まりが感じられます。

Northbound APIも今回のワーキングデイのボード会議で承認されました。今後、ノースバウンドインターフェースについて20以上のコントローラの取り組みを調査しながら、ワーキンググループの具体的な目標を設定していきます。年内に一度会合を開き、ワーキンググループのより明確なゴールを設定していきます。

▼図2　OpenFlowサーティフィケーションプログラム認定マーク

出典：https://www.opennetworking.org/sdn-resources/onf-specifications/openflow-conformance

SDNのエコシステムの形成に向けて

ONFはさまざまな関連団体と連携しながらSDNの発展を手助けしています。各団体ごとにリエゾンと呼ばれる窓口担当者が設置され、共同イベントや、標準化プログラムの策定、ユースケースやホワイトペーパーの共同執筆など、密接な活動を展開しています。その中で、日本で最近注目を集めているNFVとの活動について紹介します。

NFVの概要

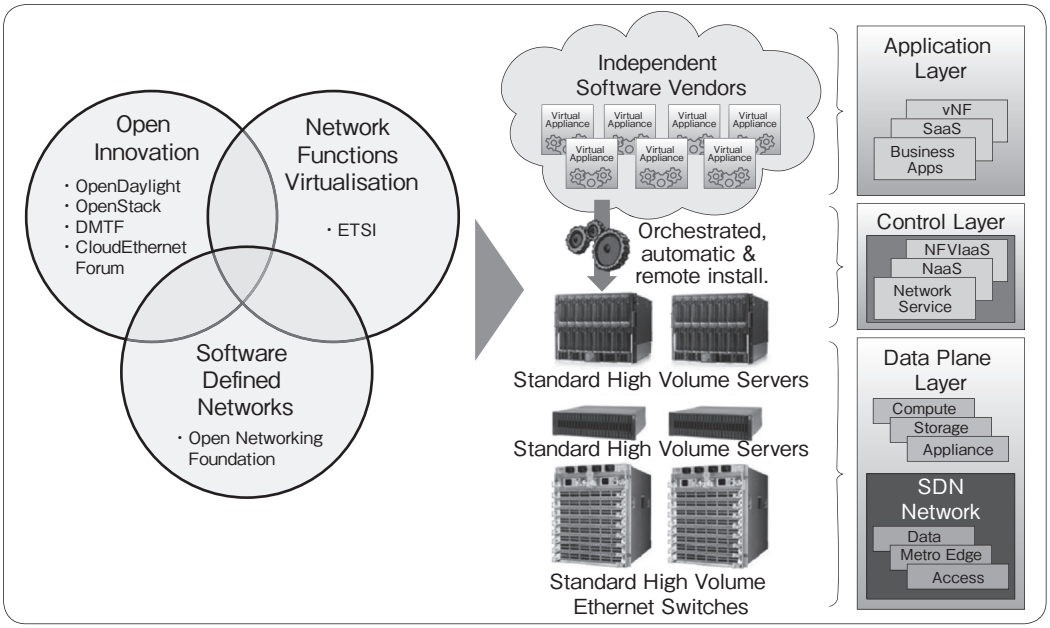
Network Function Virtualisation(以下NFV)とは、通信事業者の理想的なフレームワーク、モデル、アーキテクチャのユースケースを策定し、通信事業者視点での仮想通信技術に関する「ニーズの整理」を行おうというものです[編注]。2012年のETSIの業界標準グループ(ISG)の会合で、NFVの要件を定義するため、約20の通信事業者が集まりました。現在では30を超える通信事業者と、100以上のベンダが参加しています。

現状の通信事業者のネットワークには、いくつかの課題があります。競合他社との競争を勝ち抜くためにはネットワークのコストを抑えながら、差別化可能なサービスをネットワーク上で展開し、収益性を確保しなくてはなりません。この大きな課題を解決する1つの方法が、柔軟性、オープン性、速いイノベーションサイクル、ソフトウェアとハードウェアの分離など、クラウドコンピューティング環境における利点をネットワークに適応するアプローチです。NFVではこのアプローチに着目しています。

NFVが関心をよせるこのアプローチは、オープンかつ広くベンダ間で採用を狙うONFが標準化を進めるOpenFlowと非常に類似しています。両者ともソフトウェアのイノベーションを最大限に活かしながら、これまで実現が困難であったスケーラビリティと運用の効率性を同時に解決します。**図3**のようにネットワークの機能がVMにマイグレーションされ、SDNがレイヤ4-7を意識したコネクティビティを提供し

編注) NFVという用語は本稿とは別の意味(もっと広い意味)で用いられることもあります。コラム「ネットワーク機能の仮想化としてのNFVと仕様としてのNFV」を参照。

▼図3　NFVとSDNマッピング



ようとしています。

NFVとSDN

NFVにもONFのリエゾンがおり、密接な活動を展開しています。ONFは必要に応じて、NFVの要望をSDNのアーキテクチャの構想や標準化のプロセスに取り込む活動を展開しています。ここではNFVが策定した2つのユースケースについてSDNの適応を考えてみます。

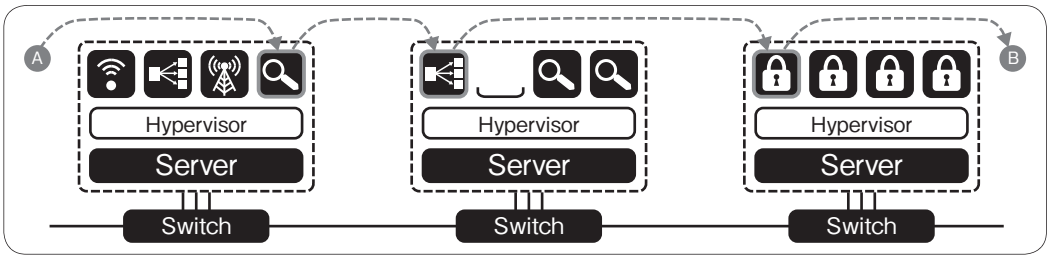
まず1つ目が「サービスチェイニング」として知られている「VNF(Virtual Network Function) Forwarding Graph」です(**図4**)。利用者の要求を満たすサービスを提供するためには、別々のサーバにある機能を利用しなくてはならないこともあります。通信事業者がこのようなサービスを効率よく提供するためには、非常に複雑な設定のプロビジョニングが要求されます。

OpenFlowを利用すれば、コントローラで動作するサービスのアプリケーションからインフラストラクチャレイヤにあるネットワーク機器のフローの制御を行うだけで、このサービスモデルを簡単に実現できます。設定の人的ミスを削減するだけでなく、サービス開始までの時間も短縮でき、品質の高いサービスの提供を通じてより多くの収益が期待できます。

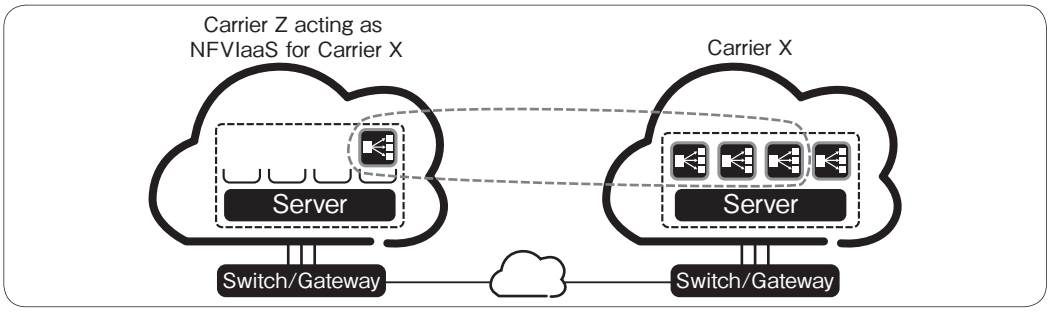
もう1つのユースケースは、「NFVIaaS」です(**図5**)。通信事業者Xが複数の地点でサービスを提供しているような例です。通信事業者Xが拠点を持たない地点でのサービス提供を利用者から求められた場合、その地域にある通信事業者Zが通信事業者Xに対してコンピュータリソースを提供し、通信事業者Xのサービスとして提供するシナリオです。この2つの拠点を結ぶ方法はさまざまですが、異なる通信事業者間ではトンネリング、IPv6/IPv4、NAT、QoSのポリシーなどが異なる環境で、この仮想的な通信経路を実現することも少なくありません。

このような環境の中でもリアルタイムでの物理/仮想エンティティのプロビジョニングが必要です。OpenFlowはオープンスタンダードであり、コントローラから各物理/仮想デバイスの制御が可能であり、異なるネットワーク構造を持つネットワークでもこのユースケースへの適応が可能です。このソリューションの詳細は、ONFのWebサイト(http://www.opennetworking.org/)にて公開されています。興味のある方はぜひソリューションペーパーをダウンロードしてください。SD

▼図4　VNF Forwarding Graph



▼図5　NFVIaaS



Column

ネットワーク機能の仮想化としてのNFVと仕様としてのNFV

Writer 高嶋 隆一(たかしま りゅういち)／ミドクラジャパン㈱ http://www.midokura.jp/

「NFV」って何だろう?

NFVは「Network Function Virtualization」の略語であり、直訳すると「ネットワーク機能の仮想化」となります。今年になって耳にすることが多くなってきた単語ですが、「NFVって何ですか?」という質問に答えることは、「SDNって何ですか?」という質問に答えるのと同じくらい難しいのではないでしょうか。

NFVという単語が使われる場合、今日では2通りの意味で使われています。1つは文字どおり「ネットワーク機能の仮想化」というグループに分類される技術を指す広義のNFVです。一方で、NFVという特定の仕様を指す固有名詞としても使われています。この関係は、「SDN」という分類と「OpenFlow」の関係に非常によく似ています。

広義のNFV

まず、広義のNFVについて解説します。この場合、「Virtualization」を起点として考えていくと理解がしやすいかと思います。「仮想化」というとまず思い浮かべるのはサーバ仮想化ではないでしょうか。従来は、「物理的な装置＝サーバ」であったものが、サーバ仮想化環境ではハードウェアは隠蔽され、メモリ、CPU、ストレージなどの計算機資源のプールとして提供されています。仮想化することにより資源の共有化やオンデマンドでの提供が可能になったわけです。

一方で、ネットワーク機器は依然として物理的に存在しているケースが多いようです。それはなぜでしょうか? サーバ機器では、仮想化する対象と実行する環境は同じものです。一方で、ネットワーク機器では目的に特化したハードウェアが利用されています。それらを汎用的なサーバ上で仮想化する場合には、特化したハードウェアに比べて著しく性能が劣化するため、従来は高性能を必要とされない小規模なルータ、スイッチなどに限定されてきたわけです。しかし、今日では汎用的なサーバの高性能化が進んでおり、適用領域によっては十分な性能を発揮できるようになってきています。そこでネットワーク機器についても仮想化しようという発想が出てきました。

NFVの実装

では、NFVと言われている具体的な実装はどんなものなのでしょうか。1つには「ネットワークそのもの」、つまりL2、L3の仮想化が挙げられます。仮想ルータや仮想スイッチなどはすでにみなさん利用されているでしょう。

また、つい最近ですが、NECとテレフォニカ社が家庭用通信機器の仮想化についての実証実験のプレスを行いました[注1]。従来、物理的に存在していた宅内装置を回線の先の通信事業者の局舎内に仮想化しておいてしまおうという試みです。一度客先に設置してしまうと更新が難しいハードウェアに比べて保守性が向上しますし、汎用機器の多重収容による機器コストの削減も狙っているわけです。

次に、ロードバランサ、ファイアウォール、

注1) http://jpn.nec.com/press/201310/20131011_03.html

VPNといった高レイヤの機能の仮想化です。じつは、NFVという言葉が出てくる前からこれらの実装は規模に応じて利用されてきました。従来はニッチな市場であったのですが、現在ではクラウド環境での要求もあり、大手メーカーからも「Software ○○○」や「Virtual ○○○」のような名称で汎用サーバ上で動作したり、VMとして動作したりするものも増えてきています。

実装例：MidoNet

では、実際の使われ方としてはどうなるのでしょうか。例として、筆者の所属するミドクラの製品である、MidoNetの場合を説明してみます。MidoNet[注2]はクラウド向けのネットワークスタックです。基本的な機能としては、顧客のVMをL3、L2で分離するネットワーク仮想化機能を提供しています。

ここでエンドユーザが「ファイアウォール機能を使いたい」と言った場合、どうなるでしょうか。従来の物理ネットワーク上の実装であれば、一度ネットワークを切断し、サーバネットワークの手前にそれらのアプライアンスを挿入する必要が出てきます。一方、仮想化されたネットワークスタック上であれば、ユーザ向けのネットワークを切断することなく、動的に機能を追加することが可能です。MidoNetの例では**図1**のような機能を提供済み、もしくは提供予定です。

仕様としてのNFV

次に、仕様としてのNFVについて解説します。こちらは通信事業者の標準化団体であるETSI(欧州電気通信標準化機構)により、策定が行われています。本稿執筆中の2013年10月14日に初版が公開されたできたてホヤホヤの仕様です[注3]。

こちらのNFVの特徴は2つあります。1つ目は既存仕様との棲み分けです。ETSI版NFVでは、ONFが考えるSDNを強く意識しており、基本的なL2、L3といった機能はSDNに任せ、そのうえで動作する高レイヤの機能を対象としています(**図2**)。

2つ目は実装対象となる機能です。広義のNFVの実装では、ロードバランサ、ファイアウォール、VPNといったエンドユーザ向けの機能を実装しているのに対し、ETSI版NFVでは、携帯網固有の機能、Carrier Grade NAT、監視機能など通信事業者固有の機能を仮想化しています。

◆ ◆ ◆

NFVをひとことで言うと「汎用サーバ上でネットワークの機能を実装する」ということになりますが、意味合いが非常に広く関連する技術も多岐に渡ります。単純に「NFV」と一括りにするのではなく、個々の技術について必要に応じて調査していくことが正しい理解につながるでしょう。

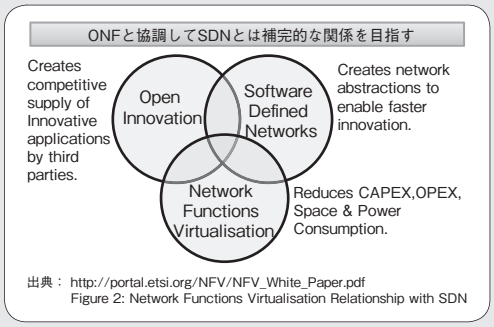
SD

▼図1　Network Virtualization "Function"の例：MidoNetの場合

L2 separation	スイッチによるL2レベルでの分類を動的に制御
L3 separation	ルータによるIPレベルでの分離を動的に制御
FWaaS	FireWall 機能を動的に追加・削除
LBaaS	L4/L7 Load Balancer機能を動的に追加・削除
VPNaaS	IPsec VPN 機能を動的に追加・削除

MidoNetのような仮想ネットワークスタックでは、従来データセンタ内で使われていた機能が動的に追加／削除できることが求められる

▼図2　ETSI版NFVの特徴



出典：http://portal.etsi.org/NFV/NFV_White_Paper.pdf
Figure 2: Network Functions Virtualisation Relationship with SDN

注2）http://www.midokura.jp/midonet/
注3）http://www.etsi.org/technologies-clusters/technologies/nfv

CHAPTER 2

オープンソースのSDNフレームワーク OpenDaylightとは何か?

Writer 佐藤 哲大(さとう てつひろ)／シスコシステムズ合同会社

The Linux Foundationは2013年4月に、オープンソースのSDNフレームワークを開発するプロジェクト「OpenDaylight」を発表しました。その最初の公式リリース(コードネームHydrogen)が2013年12月9日に実施されます。本章では、一足早くOpenDaylightの概要とその利用方法を紹介します。

OpenDaylightとは

OpenDaylightはSDNコントローラフレームワーク[注1]のオープンソースプロジェクトです。SDNコントローラのような基盤技術においては各々が個別に開発するよりもその経験と知恵を集積して共同で開発し、各ベンダは付加価値の創造に直結するSDNを用いたインテグレーションやアプリケーション開発に注力できるようにとの狙いから、国内外の多数の有力なネットワーク関連ベンダが支援しています。

OpenDaylightの大きな特徴の1つは、マルチSDNプロトコル対応ですが[注2]、誰かが新しいSDNプロトコル対応を追加すると、ほかの人もそれを利用できるというのは、コントローラを共通化するからこそ得られる利点です。

OpenDaylightのアーキテクチャ

OpenDaylightのアーキテクチャを簡単に概説しておきましょう(**図1**)。下から上に向かって、まず、SDNプロトコルを使ってコントローラとデバイスがコミュニケーションするためのサウスバウンドAPIがあります。

その上位にサウスバウンドAPIに対応するプリミティブな処理を提供するSAL(Service Abstraction Layer:サービス抽象化レイヤ、**表1**)があります。加えて、コントローラ上で典型的に必要とされる処理をまとめたサービスファンクション(**表2**)があります。SALとサービスファンクションの違いは、提供するサービスの抽象度の高低で、どちらもコントローラ上のアプリケーションから直接利用できるJava APIです[注3]。

さらに上位には、コントローラの提供するサービスを外部のアプリケーションが利用するためのRESTベースのノースバウンドAPIがあります。併せて、これらのインターフェースを用いて開発された特定用途向けのコントローラアプリケーションも提供される予定です。

環境セットアップ

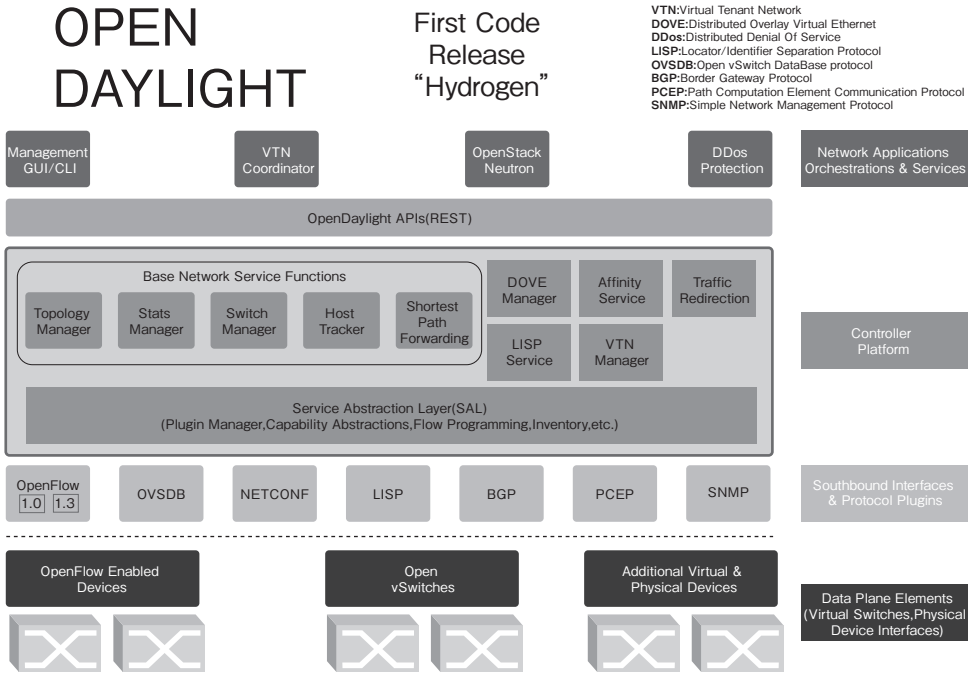
実際に、OpenDaylightの環境をセットアップしてみましょう。事前に、SELinuxとファイアウォールを無効にして、JDK 1.7およびMaven3をインストールしておいてください。はじめに、適当なディレクトリにOpenDaylightのソースを公開Gitリポジトリからダウンロー

注1) 「コントローラ」には、フレームワークとしてのコントローラとサービスとしてのコントローラとの両方の意味がありますが、本稿ではこれ以降、文脈から区別できる限り、どちらもコントローラと表記します。

注2) たとえば、OpenFlowのほかにも、トラフィックエンジニアリングのためのPCEPやスイッチ管理のためのOVSDBなど複数のSDNプロトコルのサポートが予定されています。

注3) SALはサービスファンクションよりも抽象度が低いインターフェースです。SALにおける抽象とは、複数のSDNプロトコルをサポートするうえで、それぞれのプロトコルに共通するサービスに対して統一されたインターフェースを提供することです。

▼図1　OpenDaylightアーキテクチャ



出典：http://www.opendaylight.org/announcements/2013/09/opendaylight-project-releases-new-architecture-details-its-software-defined

▼表1　主なSAL（現在はOpenFlowに対応する機能が中心）

API	説明
sal.core	ノード、ノードコネクタ、エッジ、パスなどの抽象データを定義
sal.packet	生のパケットの受信。パケットデータのエンコードおよびデコード
sal.match	OpenFlowマッチに相当する処理
sal.action	OpenFlowアクションに相当する処理
sal.topology	トポロジの更新を通知。エッジの更新、超過使用およびその平常化を通知
sal.flowprogrammer	OpenFlowフローの追加、削除、更新
sal.reader	統計情報の取得。統計情報の更新を通知

▼表2　主なサービスファンクション

API	説明
スイッチマネージャ	NWデバイス情報（デバイス名、システムMAC、OpenFlow対応など）を取得。NWデバイスの追加、削除、変更を通知
トポロジマネージャ	NWトポロジ情報を取得。NWエッジの更新を通知
スタティスティックスマネージャ	デバイスの統計情報（デバイスごと、フローごと、ポートごと）を取得
ホストトラッカー	ホストの情報（スイッチID、ポート、IP、MAC、VLANなど）を取得。ホストの追加、削除、変更を通知
フォワーディングルールマネージャ	フローエントリの作成。フローエントリのインストール、更新、アンインストール
クラスタリングマネージャ	ユーザアプリケーションをクラスタに対応させるためのサービスを提供

▼図2　OpenDaylightのソースをダウンロード

```
$ git clone https://git.opendaylight.org/gerrit/p/controller.git
```

▼図3　MAVEN_OPTSとJAVA_HOMEの環境変数を設定

```
$ export MAVEN_OPTS="-Xmx1024m -XX:MaxPermSize=256m"
$ export JAVA_HOME="<path_to_jdk>"
```

▼図4　Mavenでビルド

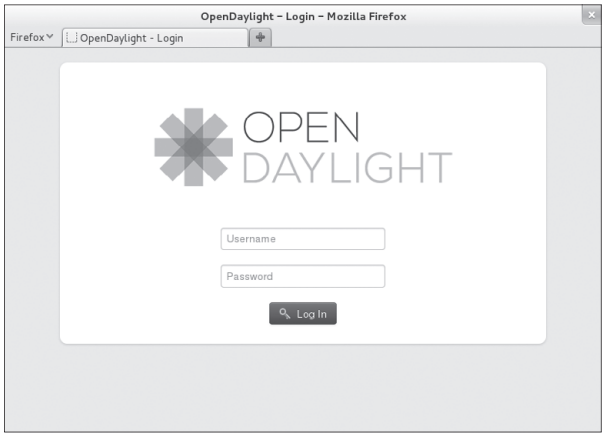
```
$ cd controller/opendaylight/distribution/opendaylight/
$ mvn clean install
 もしくは 、以上がうまくいかないとき
$ mvn clean install -DskipTests
```

▼図5　OpenDaylightを起動

```
$ cd target/distribution.opendaylight-0.1.0-SNAPSHOT-osgipackage/opendaylight/
$ ./run.sh
```

ドします(**図2**)。また、MAVEN_OPTSおよびJAVA_HOMEの環境変数を設定します(**図3**)。次に、ディレクトリを移動して、Mavenでビルドします(**図4**)。もしテストの際にメモリリークでビルドに失敗するようなら、テストをスキップしてビルドしてみてください。最後に、ビルドの成果物ができたディレクトリに移動して、OpenDaylightを起動します(**図5**)。ブラウザからhttp://<controller_ip>:8080にアクセスできたら成功です(**図6**)。初期ユーザ、初期パスワードともにadminでログインできます。

▼図6　ログイン画面



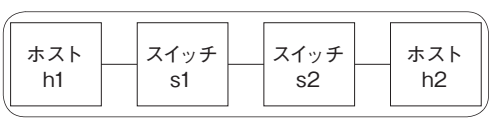
Mininetセットアップ

試験環境にはMininetを使います。MininetはOpenFlowコントローラを試験するための仮想OpenFlowスイッチ、仮想ホストからなる仮想ネットワークを提供してくれます。Mininetのサイト(http://mininet.org)でMininet VMがOVA形式で提供されるので、これを利用するのが手っ取り早いです。Mininet VMのシェルからmnコマンドで仮想ネットワーク環境を起動します(**図7**)。これにより、2台のスイッチの両サイドに1台ずつホストが接続した仮想ネットワークができます(**図8**)。ブラウザで再度読み込んで(Ctrl + F5)、2台のスイッチが見えたら、コントローラへ無事登録できたことになります(**図9**)。

▼図7　仮想ネットワーク環境を起動

```
$ sudo mn --controller=remote,ip=<controller_ip> --topo=linear,2
```

▼図8　Mininetトポロジ



OpenDaylightでコントローラアプリケーションを作る

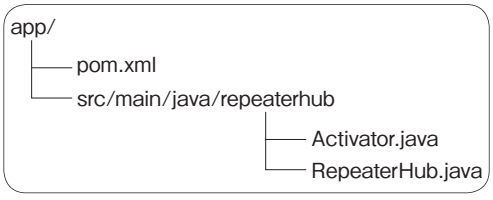
次に、OpenDaylightを使ってみます。前述のとおり、OpenDaylightは開発フレームワークであり、直接ソリューションを提供するものではありません。コントローラアプリケーションの開発やシステムインテグレーションに活用するのが基本的な使い道です。

利用できるインターフェースは、RESTベースのノースバウンドAPIとJava APIがありますが、今回はOpenDaylightアーキテクチャの理解にも役立つように、より柔軟な機能を持つJava APIを使って、OpenFlowベースのリピータハブアプリケーションを作ってみます。今回作成するリピータハブは、次のように動作します。

(1) フローテーブル中にマッチするフローエントリがなければ、OpenFlowスイッチ(以下、スイッチ)は受信パケットをコントローラへPacket-Inする
(2) コントローラはPacket-Inされたパケットの送信MACアドレスを参照し、そのMACにマッチしたら、Flood(入力ポート以外のすべてのポートへ出力)するフローエントリをスイッチに作成する[注4]
(3) (2)でフローエントリの作成を誘発したPacket-Inされたパケットは、Floodするように Packet-Outする

注4) 後続する同じ送信MACアドレスのパケットはこのフローエントリにマッチし、コントローラにPacket-Inされることなくスイッチ上で直接転送される。

▼図10　コントローラアプリケーションのファイル構成



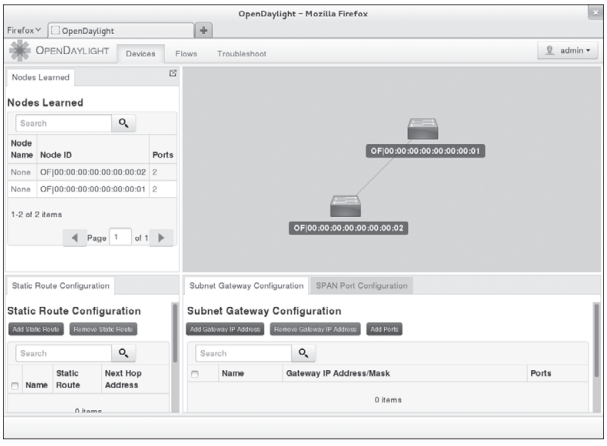
コントローラアプリケーションの構成

もうひとつのOpenDaylightの大きな特徴として、拡張性の高いモジュラーアーキテクチャが挙げられます。これを実現するために、OpenDaylightはEclipse Equinoxと呼ばれるOSGiフレームワークを実行環境としています。OSGiは、現在のJava言語仕様では提供されないモジュール(その実態はOSGiバンドル)の依存管理、バージョン管理、動的なプラグイン(追加、削除、起動、停止)を可能にします。サウスバウンドAPI、Java API、ノースバウンドAPIはすべてOSGiバンドルです。また、Java APIを使って開発されるコントローラアプリケーションも同様にOSGiバンドルです。OpenDaylightの実態は、OSGiランタイム上で動作するOSGiバンドルの集合体といえます。

OSGiバンドルとしてのコントローラアプリケーションは、次の3種のファイルから成り、ファイル構成は**図10**のようになります。

・ユーザアプリケーション(たとえば、RepeaterHub.java)
・バンドルアクティベータ(Activator.java)
・POMファイル(pom.xml)

▼図9　トポロジ画面



RepeaterHub.java(**リスト1**)は、まさにこれから作るリピータハブのユーザアプリケーションです。このアプリケーションでは、パケットの受信とデコードのためにSALのパケットサービス(sal.packet)、転送ルールの作成のためにサービスファンクションのフォワーディングルールマネージャを利用します。コントローラ上でPacket-Inしたパケットを処理させるため、IListenDataPacketインターフェースを実装します(**リスト1-**①)。また、このクラスではパケットをデコードするためにIDataPacketService、転送ルールの適用のためにIForwardingRulesManagerを利用します(**リスト1-**②)。これらのサービスの依存性注入は、後述するバンドルアクティベータによって行われます。receiveDataPacket()メソッドの中で、実際にPacket-Inした際の処理を定義します(**リスト1-**③)。receiveDataPacket()メソッドにはRawPacketオブジェクトが引数として渡されるので、IDataPacketServiceを使って、これをパケットの各フィールドへアクセスするため

▼リスト1　RepeaterHub.java

```
public class RepeaterHub implements IListenDataPacket {                    ①

    private IDataPacketService dps;                                         ②
    private IForwardingRulesManager frm;
    ...中略...
    @Override
    public PacketResult receiveDataPacket(RawPacket inPkt) {                ③
        Packet pak = dps.decodeDataPacket(inPkt);                           ④
        if (pak instanceof Ethernet) {
            // action
            List<Action> actions = new ArrayList<Action>();                 ⑤
            actions.add(new Flood());
            // match
            Match match = new Match();                                      ⑥
            byte[] srcMac;
            srcMac = ((Ethernet) pak).getSourceMACAddress();
            match.setField(MatchType.DL_SRC, srcMac);
            // flow
            Flow flow = new Flow(match, actions);                           ⑦
            Node node = inPkt.getIncomingNodeConnector().getNode();
            String mac = convertToString(srcMac);
            FlowEntry fEntry = new FlowEntry("Policy" + mac, "Flow" + mac, flow, node);
            // install flow
            Status status = frm.installFlowEntry(fEntry);                   ⑧
            if (status.isSuccess()) {
                logger.trace("Flow entry was installed successfully");
            }
        }
        // packet out
        floodPacket(inPkt);                                                 ⑨
        return PacketResult.IGNORED;                                        ⑩
    }
    private void floodPacket(RawPacket inPkt) {                             ⑨
        Node node = inPkt.getIncomingNodeConnector().getNode();
        NodeConnector allNodeConnector = NodeConnectorCreator
                .createNodeConnector(NodeConnectorIDType.ALL,
                        NodeConnector.SPECIALNODECONNECTORID, node);
        inPkt.setOutgoingNodeConnector(allNodeConnector);
        dps.transmitDataPacket(inPkt);
    }
    ...中略...
}
```

にPacketオブジェクトに変換します[注5](**リスト1-④**)。アクションの定義はリピータハブなのでFloodになります(**リスト1-⑤**)。マッチ条件の定義は送信MACでマッチさせるようにします(**リスト1-⑥**)。次にアクションとマッチ条件を使って、フローおよびフローエントリを作成します(**リスト1-⑦**)。IForwardingRulesManagerを使って、フローエントリをインストールします(**リスト1-⑧**)。Packet-Inしてきたパケットは、FloodとしてPacket-Outしてあげます(**リスト1-⑨**)。最後に、同様にPacket-Inしているほかのバンドルでのパケット処理を可能にするため、戻り値としてPacketResult.IGNOREDを返します(**リスト1-⑩**)。

Activator.java(**リスト2**)は、モジュールとしてのユーザアプリケーションを動的にプラグインするためのバンドルアクティベータで、OSGiフレームワークが提供するBundleActivatorインターフェースを実装するComponentActivatorAbstractBase抽象クラスを継承します(**リスト2-⑪**)。configureInstance()メソッドの実装の中で、アプリケーションが実装するIListenDataPacketインターフェースを指定します(**リスト2-⑫**)。これにより、実際にPacket-Inした際にアプリケーションにパケットが渡されようになります。また、アプリケーションが利用するIForwardingRulesManagerおよびIDataPacketServiceのインスタンスをユーザアプリケーションへ依存性を注入します(**リスト2-⑬**)。そのほか、コードの掲載は省略しましたがユーザアプリケーションの起動および停止時に実行されるメソッドを定義します。

POMファイルには、Mavenでビルドするために、作成するアプリケーションが依存するほかのOSGiバンドル、それらのリポジトリの情報などを規定します。紙幅の都合上、POMファイルの掲載は割愛しますが、今回のすべてのソースコードは、本誌サポートページ(http://gihyo.jp/magazine/SD/archive/2013/201312/support)からダウンロードできますので、詳細に関心がある方はご覧ください。

アプリケーションができあがったら、POMファイルを配置したディレクトリ(ここではapp/)に移動して、Mavenでビルドしてください(**図11**)。app/target/ディレクトリにビルドの成果物(repeaterhub-1.0.0-SNAPSHOT.jar)

注5) マッチ処理の個所にあるように、Ethernetフレームの各フィールドへアクセスするには、さらにPacketクラスのサブクラスであるEthernetクラスへキャストしてあげる必要があります。

▼リスト2　Activator.java

```
public class Activator extends ComponentActivatorAbstractBase {                    ⑪
    ...中略...
    @Override
    public void configureInstance(Component c, Object imp, String containerName) {
        if (imp.equals(RepeaterHub.class)) {
            Dictionary<String, String> props = new Hashtable<String, String>();
            props.put("salListenerName", "ListenDataPacket");
            c.setInterface(new String[] { IListenDataPacket.class.getName() }, props);  ⑫
            c.add(createContainerServiceDependency(containerName)
                    .setService(IForwardingRulesManager.class)
                    .setCallbacks("setForwardingRulesManager",
                          "unsetForwardingRulesManager")
                    .setRequired(true));                                            ⑬
            c.add(createContainerServiceDependency(containerName)
                    .setService(IDataPacketService.class)
                    .setCallbacks("setDataPacketService", "unsetDataPacketService")
                    .setRequired(true));
        }
    }
}
```

ができます。

OSGiによるモジュラリティのおかげで、ビルドされたアプリケーションは、OSGiランタイム上で動的に追加、削除、起動、停止できます[注6]。OpenDaylightの起動時に現れるOSGiコンソールからビルドされたアプリケーションを追加します(**図12**)。追加時のコンソールの出力もしくはssコマンドから追加したアプリケーションのバンドルIDを見つけてください(**図13**)。バンドルIDは環境によって異なります。このバンドルIDを指定して、アプリケーションを起動させます(**図14**)。もう一度ssコマンドを実行して、ステータスがACTIVEに変わっていたら成功です。Mininet起動時に現れるMininetコンソールからホスト間のPingを実行すると疎通に成功するはずです(**図15**)。

OpenDaylightの特徴

Javaを採用していることで、OpenDaylightのJava APIを用いたコードは少し野暮ったい印象を受けます。しかしながら、ラージエンタープライズやサービスプロバイダでの利用を前提とする豊富な機能、高パフォーマンス、高スケーラビリティ、高可用性は、Javaの採用によってもたらされています。

本稿で紹介したモジュラリティとその恩恵であるマルチSDNプロトコル対応に加えて、Javaならではの機能として、InfinispanとJGroups[注7]によって実装されているクラスタリング機能が挙げられます。このクラスタリング機能は、当初はコントローラの冗長性を実現するためのものですが、将来的には、コントローラの水平スケーリングを実現するためにも利用できるようになる予定です。そのほか、マルチコアCPUを有効活用するための並列処理機能の充実やプログラマ人口の多さもJava採用の恩恵と言えます。

◆ ◆ ◆

本稿で紹介したコントローラアプリケーションは、お世辞にも実用的とは言えない代物でしたが、OpenDaylightを利用した開発について、イメージを掴んでいただけたのではないでしょうか。興味をもっていただいた方は、OpenDaylightプロジェクトのサイト(http://www.opendaylight.org)および開発者向けWikiサイト(https://wiki.opendaylight.org)を訪れ、そこの資料に目を通してみてください。また、OpenDaylightを活用したコントローラアプリケーションの開発にもぜひ挑戦してみてください。SD

注6) もしくは、OpenDaylightがビルドされたディレクトリにあるplugins/ディレクトリにアプリケーションを配置しておけば、OpenDaylightが起動した際に、アプリケーションも自動で追加、起動します。

注7) InfinispanはオープンソースJavaインメモリキャッシュ。JGroupsはInfinispanノードのクラスタリングをするうえで信頼性の高いグループ間通信を提供。

▼図11 コントローラアプリケーションをMavenでビルド

```
$ mvn package
```

▼図12 コントローラアプリケーションを追加

```
osgi> install file:/<path_to_dir>/
target/repeaterhub-1.0.0-SNAPSHOT.jar
```

▼図13 バンドルIDを調べる

```
osgi> ss repeater
"Framework is launched."

id      State       Bundle
178
INSTALLED   repeaterhub_1.0.0.SNAPSHOT
```

▼図14 コントローラアプリケーションを起動

```
osgi> start 178
```

▼図15 疎通確認

```
mininet> h1 ping h2
```

CHAPTER 3

エンジニアが本音で語る! SDN/OpenFlowをやる・やらない理由

本章では、現役のエンジニアがどのようにSDN/OpenFlowに向き合い、自らのビジネスに活かして／活かそうとしているのか7人の論客が思うところを書き下ろしました。読者の皆さんも、SDN/OpenFlowに対してどういう態度で接するか考えてみてください。

第1の論客 使わなそうだから関係ない、んなこたぁない

Writer 谷口 有近(たにぐち ありちか) ㈱FIXER 技術本部長／シロッコ情報技術研究所 代表

ブログに書いてもPVが伸びないSDN/OpenFlow

SDNやNFVという言葉の背景にある“ネットワークの仮想化”がもたらす価値は、適用対象となる領域もさまざまで多種多様です。では、すべてのエンジニアにとって、その価値の獲得、実践に向けて、このトレンドを必死こいて検証し、自身の業務において適用範囲を無理矢理探してまで取り組む価値があることかと言えば、そんなことはないよね、と。

といいますかこれ、現時点では大規模データセンターのほうがその利点その価値を引き出しやすいわけで、中小でこぢんまりと仕事してたらほぼ使わんですわな。「やってみた」ブログ書いても、現実に業務でこれら技術を使うはめになる技術者の絶対数が少ないわけで、残念ながらPVもいまいち伸びないでしょう。

技術を取り巻く環境はそんな感じで認識してますが、されどされど、ソフトウェア制御や汎化(はんか)・モジュール化、そして集中管理といったこのトレンドの背景は、クラウドなノリと併せて理解しておかないと、あとあと「使えない技術者」と言われてしまい結構マズくなるんじゃね、というのが、これまでの流れを見てきたうえでの本音です。

サーバとネットで縄張り争いしている場合か

FPGAが金融の業務電文を脊髄反射してるような時代に、何がソフトウェアで何がハードウェアなのか定義することさえも面倒になってるわけです。そのうえで仮想化トレンドの根幹に流れる何かをあえて定義しようと試みるわけですが、「これまでとは違う実装方法を用いて仮想化したらウマー」=「コスト構造がロックアップした古い抽象化を捨てて新たなレイヤで抽象化をやり直す」っていう観点だと認識してみるに、これ、サーバ担当とネットワーク担当がその責任を擦り付け合いしてる世界では、なかなか出てこないだろうな、と。

サーバ技術者が抱えていた課題をネットワーク技術の領域で改善しちゃう系の、別レイヤでの改良が何かの課題を解決するという、天文学と物理学と数学の蜜月のようなアプローチ、全体最適での観点と言いますか、斜に構えた言い方なら、人の仕事を奪って自身の評価にする、というスタンスは、いかにもイノベーティブで、競争社会での技術開発だなぁ、と実感してます。

高みを目指すならばやるべし!

やる理由もやらない理由も筆者には存在しません。パブリッククラウドを利用していれば必然的に、ネットワーク仮想化の恩恵を受けているでしょう。Web上のボタンをクリック1つでプログラムそのままにジオレプリケーションしちゃう時代ですよ？　ガンガン構成を変えて新デバイスと高容量化に対応していく通信事業者

に感謝しているならば、その体験は、どこかでネットワーク仮想化の技術革新に支えられてるはずです。そんなわけで、自身の業務で、その実装の恩恵に与りたくなれば、何かしらの形で活用するでしょうし、不要なら実践もなくスルーしていきます。不要だと認識していても、すでに無意識に活用してるでしょう。ゆえに、高みを目指すなら、知っていたほうが良い。

ネットワークを専門にしていると自負するエンジニアが、この大きなトレンドをフォローしていないのはね、それどうかと思います。ですが、このトレンドをフォローしてない開発者が、自分で作ったオレオレフレームワークの中でシステムの分散化と動的なノード構成管理を実現してたりするのを見たりすると、専門って言葉を、ほかの領域に関心を持たないことの言いわけに使っちゃダメだよねぇ、ということも思っちゃったりしてまして、近ごろなかなか複雑なパケットフローになっています。

第2の論客 SIerがSDN/OpenFlowをやるワケ

Writer 前田 繁章(まえだ しげあき)NTTデータ㈱ 基盤システム事業本部 システム方式技術BU 課長代理

SDN/OpenFlowを推進する3つの理由

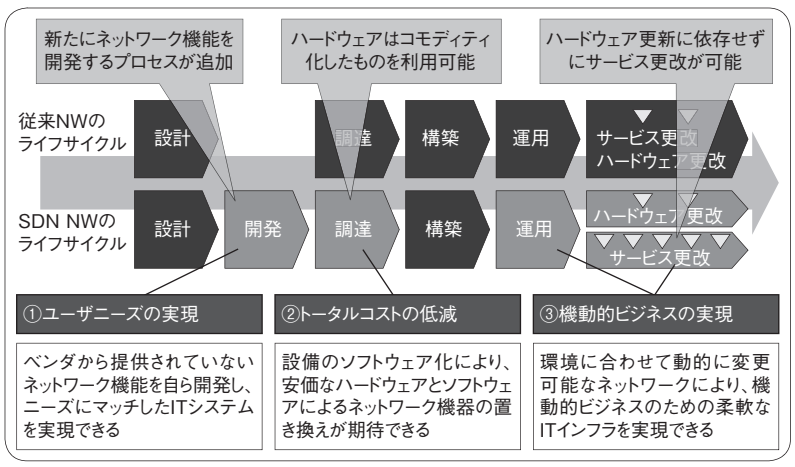
SDN/OpenFlowのメリットについて、まずはユーザの立場から、ネットワーク構築・運用のライフサイクルを考えながら、従来との対比で説明します。

図1は、従来のネットワークのライフサイクルと、SDN/OpenFlow環境でのネットワークのライフサイクルとを比較し、その違いについて示したものです。工程では、ネットワークの機能を自分で作る開発工程が新たに発生します。このほか、調達・運用・更改の各工程で大きな変化が起こります。この変化によって得られるメリットを3つ、順に説明したいと思います。

ニーズの実現

1つめは、ユーザニーズの実現です。これまでは、ベンダから提供されていない機能についてはベンダ対応を待つか、その機能を利用することを諦めるかする必要がありました。SDN/OpenFlowでは、ソフトウェアの領域で自由に機能を実現できるようになります。これにより欲しい機能を欲しい時に実現できるようになります。**図2**は従来と比べてアーキテクチャにどのような変化が生じるかを示したものです。このように作りこみ可能な範囲が増えることで、よりニーズにマッチした(特別な要件にも対応した)ネットワークが実現できるようになります。

▼図1 SDN/OpenFlowによるNW(ネットワーク)の構築とライフサイクルの変化



コストの問題

2つめは、トータルコストの低減です。従来では一部の機能のみ利用したい場合でも、不要な機能まで搭載された高機能なネットワーク機器を利用することがありました。SDN/OpenFlowでは、安価

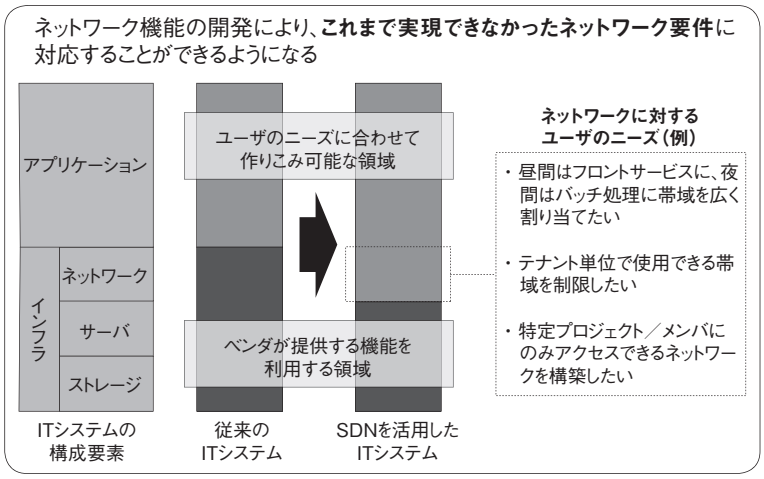
なハードウェア＋必要な部分のみを実装したソフトウェアに置き換えることでコストの低減が期待できます(ただし、ソフトウェア開発の費用が必要なため、従来装置のほうが結果的に低コストの場合もよくあります)。

ダイナミックなネットワークの実現～SIerの意義

3つめは、機動的ビジネスの実現です。従来はハードウェア更改にあわせてサービス公開をするため、必ずしもビジネス環境の変化にベストなタイミングでサービス変更ができるとは限らないという問題がありました。SDN/OpenFlowでは、ハードウェアはそのままに、ビジネス環境の変化に応じて必要となるサービスをソフトウェアでタイミングよく実装しサービス更改することが可能になります。これにより、よりビジネスに即したネットワークの実現が期待されます。

上記のメリットを享受したいユーザが必ずしもソフトウェア開発が得意とは限りません。そのためにSIerがいるのです。ユーザニーズを実現するため、業務アプリケーションだけではなく、ネットワークそのものもSIerのビジネス領域に加わることになります。このようにユーザだけでなく我々もメリットを享受できるためSIerがSDN/OpenFlowに取り組んでいます。

▼図2　SDNによるITシステムの構造変化



第3の論客　SDNの起源に見るOpenFlowとの付き合い方

Writer　中井 悦司(なかい えつじ)　レッドハット㈱

相対性理論の起源とは?!

まったくの余談から入りますが、アインシュタインの「相対性理論」は、どのようにして生まれたかご存じでしょうか？　天才アインシュタインの突然の閃きで生まれた理論……というわけではありません。相対性理論の背後には、「高速移動する物体は、空間を満たすエーテルの影響で長さが縮む(ローレンツ収縮)」という、奇妙な理論がありました。これは、「光の速度は、観測者の状態によって変化しない」というマイケルソン・モーリーの実験結果を説明するために、ローレンツ博士が考え出した苦肉のアイデアでした。

その後、アインシュタインは、相対性理論(時間と空間の相対性)によって、「エーテル」のような怪しげなものを想定しなくても、ローレンツ収縮を理論的に説明できることを示したのです。一見、誤った考えでも、それを頭から否定することなく、その背後に隠された真実を見いだす力、それがアインシュタインの才能だったのかもしれません。

これは、今回のテーマ「SDN/OpenFlow」にもあてはまります。SDN/OpenFlowは、これまでにない画期的な新技術……というわけではなく、ある意味において、既存技術の自然な延長と考えることができるのです。

SDNにも起源あり

みなさんは「SDNのしくみを教えてください」と言われたら、どうしますか? 「ソフトウェアでネットワークを柔軟になんたらかんたら……」とお茶を濁す手もありますが、筆者は**図3**を紹介するようにしています。これは「ONF(Open Networking Foundation)」が公開している資料からの引用に、少しコメントを追加したものです[注1]。端的に言うと、図の中段にあるソフトウェア(コントローラ)が、図の下段にある複数のネットワーク機器を集中制御するしくみにすぎません。

このとき、下段のネットワーク機器が担う本来の通信経路に加えて、コントローラと各ネットワーク機器が通信する、裏の管理ネットワークが必要となります。SDN業界では、これら2つの通信経路をそれぞれ「データプレーン」「コントロールプレーン」と呼んでいます。

実は、ここに、SDNの起源が隠されています。業務用のネットワークスイッチは、もともと、その内部にデータプレーン(制御部)とコントロールプレーン(処理部)と呼ばれるしくみを持っています。ポートからポートへとパケットを移動する単純な処理を担うのが「データプレーン」で、ASICなどによるハードウェア処理が行われます。一方、パケットの内容をチェックしてポート間の移動ルールを決定するなどの複雑な処理は、スイッチに搭載された組込みソフトウェアが実施します。このソフトウェア処理部分が「コントロールプレーン」です。

これでおわかりのように、従来はネットワーク機器に個別に搭載されていた「コントロールプレーン」を「外出し」にして、1つのソフトウェアで複数のネットワーク機器を同時にコントロールできるようにしたのが「SDNのアーキテクチャ」の起源です。

注1) https://www.opennetworking.org/sdn-resources/sdn-library/whitepapers

SDN/OpenFlowで何ができるのか?

エンジニア仲間で会話していると、「SDN/OpenFlowで結局何ができるの?」というのはよく議論になるところです。先の話を振り返ると、本質的には「複数のネットワーク機器を同時にコントロールする」ことで何ができるのか、ということになります。裏を返すと、各機器が個別のコントロールプレーンで独立に動作するという、これまでの「自立分散型」の限界を見極めるということでもあります。残念ながら、筆者はまだ、この点に関する明確な答えは持ちあわせていません。まさに、これからの研究課題です。

ところで、OpenFlowの入門書では、1台のOpenFlowスイッチを使って、「スイッチングハブ」や「ルータ」など、従来のネットワーク機器と同じことを実現する方法が説明されていることがあります。複数のネットワーク機器をコントロールするというSDNの本質から見れば、「つまらないお遊び」のようにも感じられます。

しかしながら、「つまらないお遊び」に隠された真実を見つけるのが、天才ではなかったのでしょうか? 1台のOpenFlowスイッチ(もしくは、2〜3台程度)でできることについて、真面目に考えてみるのもおもしろいかもしれません。とくに、これまでのネットワーク機器では、コントロールプレーンの実装はハードウェア内部の組込みソフトウェアとして固定化されていました。OpenFlowを利用すると、たとえ1台のスイッチでもその機能を自由に組み立てることができます。

複数のネットワークポートを持つPCに、LinuxとOpen vSwitchをインストールすれば、それだけで、OpenFlowスイッチの代わりになります。処理性能を気にしなければ、DPI(ディープ・パケット・インスペクション)のような、高価なネットワーク機器の機能を自宅で利用できるようにもなります。ソフトウェアを入れ替えることで、ネットワーク機器の機能を個人で自由に実装できるというのは、今までのネット

ワーク機器にはなかったことです。これは、意外と語られていないSDN/OpenFlowの活用エリアかもしれません。

そういえば、本誌2012年1月号のOpenFlow特集記事の中に、「OpenFlowやSDNに限らず本質をとらえずして技術を評価することは、エンジニアとして恥ずべきことなのだ」という一節がありました。新規技術は、まずはとにかく触って確かめる、動作ロジックを実際に追いかけて確認する——これが筆者の信条です。

ビジネス誌の記事では、SDN/OpenFlowを利用した壮大な活用例が解説されることもあるようですが、前述のように、「本質的にSDN/OpenFlowでなければできないこと」はまだまだ未知の領域です。SDN/OpenFlowに興味があるならば、まずは、個人の「お遊び」として使い倒しながら、その実体を体で理解していくことが必要でしょう。

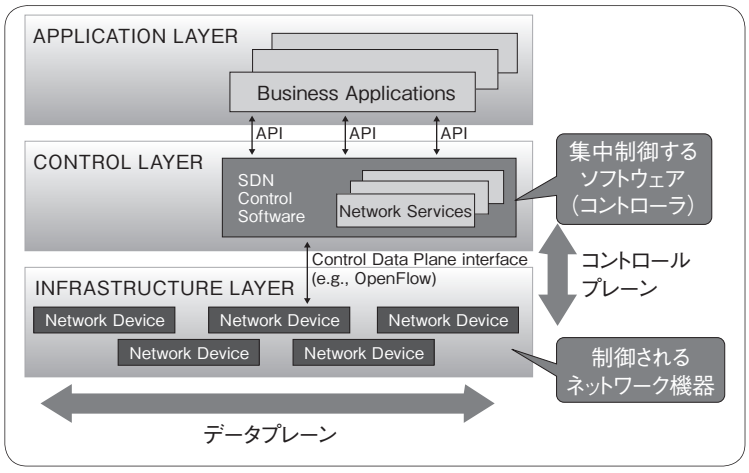
現場を助ける道具としての活用方法

これまでのITの歴史を振り返ると、個人の趣味的な取り組みが草の根で世の中に広がっていき、やがては企業システムを支える技術へと育った多数の例があります。1990年代には、自社のWebシステム開発責任者が無償のWebサーバを利用しているのを見つけて、IBMのシニアマネージャが驚愕したという逸話があります[注2]。その後、この「Apache」と呼ばれるWebサーバの完成度と人気の高さに気づいたIBMは、ついには自社製品にこれを組み入れる決断をしました。

先に触れたように、筆者は、「個人が自由に機能を実装できる」という点にSDN/OpenFlowの可能性を感じています。「こんなことができたら便利なのに……」、そんな現場の小さな困りごとを解決する道具としての活用法が広がり、さまざまなアイデアが共有されていくことを期待しています。もちろん、そのためには、技術の本質を根本から理解して、適切な利用シーン、活用方法を判断できるようになることが求められます。

個人のお遊びから生まれたアイデアが、いつか、SDN/OpenFlowの本質とも言える活用エリアを切り拓くことを期待しつつ、筆者も自作OpenFlowスイッチの構築に励みたいと思います。

▼図3　SDNのアーキテクチャ



注2）「The World Is Flat - Release3.0」
Thomas L. Friedman. Picador(2007)

第4の論客

「ネットワーク仮想化」によるIaaSネットワーク基盤の抽象化とSDN/Openflow

Writer　田中 洋(たなか ひろし)　ヴイエムウェア㈱　ネットワーク＆セキュリティ事業部
ソリューションズアーキテクト　Twitter @hirtanak　Mail：htanaka@vmware.com

IaaS基盤における既存ネットワーク技術のアプローチの限界と課題

ネットワーク仮想化、SDN/OpenFlowはすでに大規模に稼働したサービス基盤もあり各社の製品も発売されています。現状の課題からその必要性を考えてみましょう。

現在企業内でもデータセンターでマルチテナントのIaaS基盤を構築することは一般的になっています。この中でネットワークは既存技術の組み合わせで技術的に可能なものの設計、実装、運用で膨大な作業を要しており、すでに拡張や変更が容易な規模ではありません。この点を少し具体的に考えてみましょう。

あるテナントを作成する場合、実際ネットワーク基盤でどんな作業が生じているか

「VLANの作成、各スイッチへの設定」、「アクセススイッチへのQoSの設定」、「ロードバランサ、ファイアウォールのルールやテンプレート設定、セグメントへのルーティング設定」といった基本作業に加え「重複IPアドレスのために仮想ルーティングテーブルの検討」「VLANの枯渇に対応するためにロジカルスイッチドメインやファブリックの導入」などさまざまで面倒な作業が発生します。実際には、さらにアクティブVLAN数、仮想ルーティングテーブル数、ロジカルポート数など各種ハードウェアの制限も含めて考える必要があり、これらの設計、実装、運用は多大な労力を必要とし、さらに日常的な変更もあります。

IaaSネットワーク基盤全体を抽象化する「ネットワーク仮想化」とは

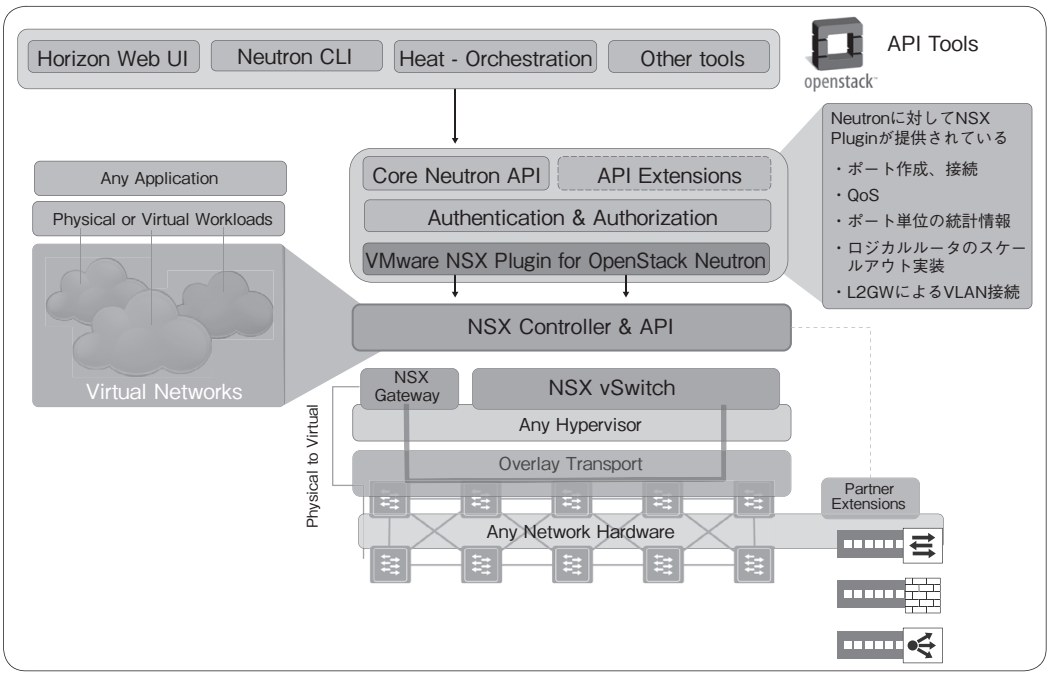
ネットワーク仮想化は、パケット転送の操作には重点を置かずオーバーレイ方式で転送します。ポート作成の粒度でAPIを提供でき、QoS、ファイアウォール、ACLなどネットワーク機能に関するAPIを提供することによって外部からIaaSネットワーク基盤全体を抽象化して操作できる点が特徴です。

VMware NSXを例に取れば、OpenStack Neutronのプラグインが提供されており、OpenStackからポートを作成・接続、QoSの適用やロジカルルータを作成すれば、NSX側でもポートの作成・接続、QoSの適用、ロジカルルータが作成されます(**図4**)。

ネットワーク仮想化とSDN/Openflowの関係

SDN/Openflowの要素は重要なのですが、あくまで便利なツールとしてです。抽象化の要素

▼図4　VMware NSXとOpenStack'Neutron'とのAPI連携の例



なしに既存のネットワーク技術で構成される基盤の問題を解決できません。また、物理ネットワーク機器を操作する場合は抽象度は上げにくく、既存のネットワーク機器に対するスクリプトと同様、誰かがワークフローを作成する必要があります。実際にワークフローの定義や管理は難しいので、SDN/Openflow単体ではIaaS基盤ではなく、よりパケット転送用途にあった適用例を考える必要があります。

VMware NSXはNicira時代からネットワーク仮想化のコンセプトでエンドユーザから多大な支持を受けて大規模な導入にいたっており、既存ネットワーク技術のアプローチの限界を考えるとネットワーク仮想化+SDN/Openflowの導入を考えない手はありません。

第5の論客 ネットワークの仕事とはつなぐことだ

Writer 井上 一清(いのうえ いっせい) ㈱IDCフロンティア 技術開発本部R&D室

SDN導入に二の足を踏んでいませんか?

SDNに関してはここ1、2年でさまざまな議論や記事が出ていると思いますが、まず、Software Defined Networkという言葉自体が非常に広義な意味合いを持つため、ユーザから見ると今ひとつSDNの具体的なメリットがわかりづらく、また商用での実事例もまだ少ないため、どのように取り組んでいけば良いのかわからず、多くの企業が二の足を踏んでいる状況です。

SDNをやる・やらない、という考え方ではなく、どうしたらその企業のネットワークを最適化できるか、どうしたらエンドユーザに対してメリットや付加価値を最大限に提供することができるのか、ということを徹底して考える方がSDNの扱い方やそのメリットに対しての理解が早くなるのではないでしょうか。

ユーザメリットを考えるとSDNになる

ネットワークの仕事はつまるところ、「つなぐこと」です。

当社に関してはIaaS事業とDC事業を提供しており、仮想マシンとお客様サーバの運用を行っていますが、これらをシームレスにオンデマンドでつなぐことで、必要なITリソースやサービスをユーザが自由自在に組み合わせられることがメリットにつながると考えており、「OneNetwork」という構想でプロジェクトを進めています(**図5**)。

これはSDN/OpenFlowの新技術のみならず、既存技術や社内ノウハウの蓄積、またそれらの応用によって、顧客利益、当社利益ともにバランスし新しい価値や構想を実現できる可能性が見えてきたからです。

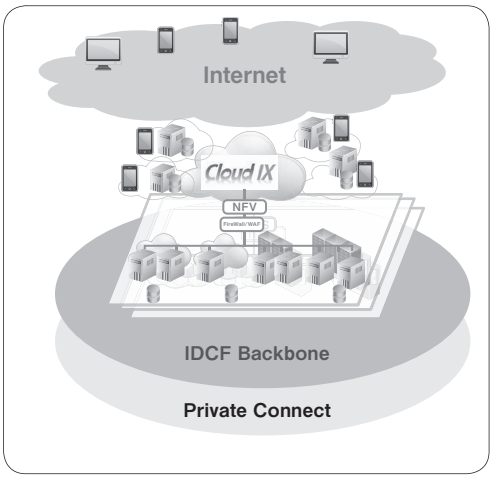
また、お客様目線で見るとネットワークの一番のボトルネックはLoadBalancerやFirewallなどのGateway部分(外部ネットワークとの出口部分)がほとんどです。このGateway部分もお客様の利用量や使い方に応じて柔軟にネットワークサービスを提供していくことで、お客様はネットワーク構成やボトルネックというものをあまり意識せずにコア事業に集中することができるようになります。

そのようなユーザ目線、サービス目線でネットワークを考えると、SDNでどのような機能が必要になってくるのかという要件が必然的に見えてきます。当社の場合には、上述のようにクラウド上の仮想マシンもお客様サーバもシームレスに接続できるようなネットワークや、柔軟にスケールアウト、機能追加できるようなネットワークサービスを提供できるようにしていくことがSDNの要件となってきます。

既存ネットワークと組み合わせて相乗効果

SDNはプログラマブルであることが最大の

▼図5　IDCフロンティアのOneNetwork構想



特徴ですので、機能面においては発想次第で何でもできてしまうものですが、どうしても最初の提案時はテクノロジーベースで内容も広がり過ぎてしまうことが多いため、テクノロジー部分から考えるのではなく、ユーザ側がどのようなネットワークを創りたいのか、という要件に合わせて、既存技術とうまく組み合わせてSDNを使っていくという考え方が良いのではないかと思います。

第6の論客　理論から現場で使える技術になるのか今後が楽しみ

Writer　湯澤 民浩(ゆざわ たみひろ)　さくらインターネット㈱　基盤戦略部バックボーンチーム

SDNが来ておもしろい状況になった

とにもかくにもネットワークの世界に久々に登場した破壊的技術(というか概念?)であることはたしからしい、という認識にいたってから、まだ2年足らずです。

見聞きする限りではSDNってそもそも、複雑系になってしまったコントロールプレーンの規律を正したい、というアカデミックな動機に端を発していると理解しています。そこから、機器やチップベンダをはじめとする供給サイドの思惑と、IaaS事業者をはじめとする需要サイドの期待と関与が深化している状況は、見ているだけでおもしろいです。

新技術対応しつつ、既存技術も活かす

では自分ではやるの、やらないの?——というお題をいただきましたが、データセンター・ホスティング事業会社での私の管掌業務は、特定のサービスに関わるものではなく、すべてのサービスがのっかる基幹ネットワークおよび対外接続の運用です。

基幹で稼働する機器には、ハードウェアの世代やファームウェアバージョンのばらつきが常に存在します。ときにはバグにも悩まされます。ベンダーから「OpenFlow対応しました、ハイブリッドで動作します」といったお話をいただいても、そうした機器をOpenFlowのデータプレーンとして使うことは想定できません。対応レベルや性能の把握が困難であること以前に、そもそも必然性がありません。

年月をかけてノウハウを積み重ねて、機器たちの性能をひきだせるよう網構成まで改善しつつ、世代交代とバージョンアップを続けながら既存のスイッチングやルーティングプロトコルで頑張ってくれてますので、当分その流れは変わらないと思います。

ただし、基幹ネットワークでのパケット転送処理がすべて思いどおりにできているというわけではなく、既存のルーティングプロトコルでは実現困難な経路で配送させたいケースもあります。当社では実際にそうしたケースの1つで、専用のOpenFlow対応ハードウェアスイッチを使った特殊な実装を構築中です。その過程で、アプリケーションからコントローラへ処理を一元化できるOpenFlowならではのメリットを実

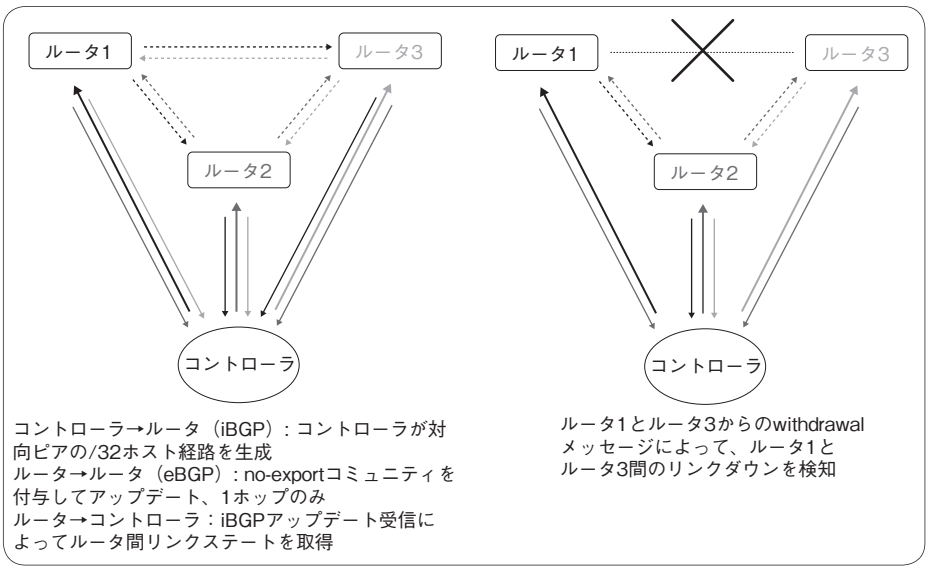
感しています。

アカデミックと現場の狭間で考える

で、この先またやることあるのかなぁと考えたときに、OpenFlowに限らずSDN的なアプローチでいくにはやはり転送処理におけるプロトコル対応が前提になるから、そっちがどうなるかしだいかなぁ――という思いにとらわれていました。しかし、最近IETFにドラフトが公開されたCentralized Routing Control in BGP Networks Using Link-State Abstractionのしくみを拝見し、コントロールの一元化って知恵をしぼれば既存のプロトコルだけでもやればできるんだ、と偏狭な考えを修正中です(実装の一端を**図6**にしました)。

そうはいっても、こちらのドラフトにしろ当社のユースケースにしろ、冒頭でふれたコントロールプレーンの「規律」を正すという動機とは縁が薄い、あくまで実践的な問題解決手段かと個人的には認識しています。逆にアカデミックな世界の人達は、こうした実装をはたしてSDNとみなすのかどうか、興味があります。

▼図6　Centralized Routing Control in BGP Networks Using Link-State Abstractionのしくみ



第7の論客

SDN/OpenFlowは課題解決の強力なツール

Writer　藤原 亜希子(ふじわら あきこ)　NTTコミュニケーションズ㈱　ネットワークサービス部

通信事業者がSDN/OpenFlowを使う理由

皆さんもご存じのとおり、SDN/OpenFlowは現状何か困っている課題を解決するための手段の1つです。ですので、SDN/OpenFlowを選ぶ理由をお話しするために、まずは私達が目指したい姿をお話ししたいと思います。

NTTコミュニケーションズでは、すでにSDN/OpenFlowを利用したサービスを実用化しています。Bizホスティングエンタープライズクラウドに SDN/OpenFlowを導入しています。今回はその発展として「VPNとクラウドの相互接続点における課題」を解決する手段として、SDN/OpenFlow に注目している例を取り上げます。

今や時代はクラウド全盛期、ユーザは簡単に

サービスを利用開始し、不要になればいつでもサービスを利用停止できるようになっています。このようなクラウドを支えるデータセンター内のネットワークは、柔軟性やスケーラビリティ、さらにはそれらが迅速に反映されることが求められ、その要求に応えるように、この数年でネットワークの仮想化が進んでいます。

しかし、少し視点を広げてみると、データセンター内でのネットワークの変更内容を、IP-VPNのようなキャリアのネットワークへ反映するために、キャリアがこれまでどおりの個別申込みによる対応を行っていては、ユーザはクラウドサービスを利用するメリットを十分に享受することができません。これこそが、解決したい課題です。

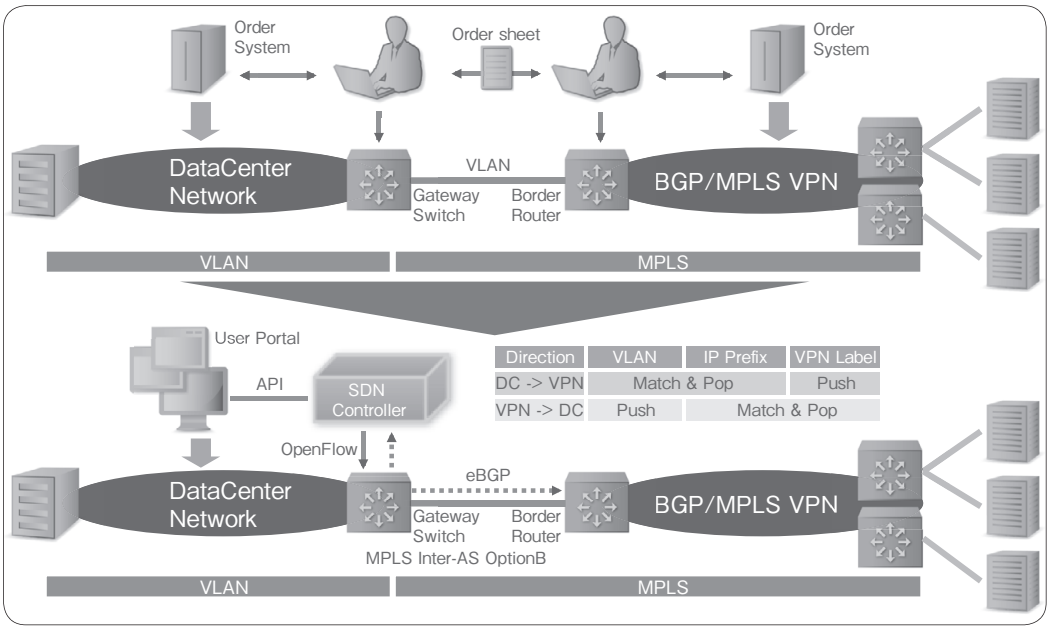
「これから」と「これまで」のネットワークをつなぎたい

急激なスピードで「これから」のネットワークへと進化するクラウドの世界と、「これまで」のネットワークであるキャリアネットワークの世界をシームレスにつなぎたい——これは、すなわちデータセンター内の仮想化されたネットワークと、BGP/MPLS IP-VPN(RFC4364)ネットワークとの間の相互接続の実現です。

実際、これまではクラウド側でネットワークが追加・削除されたりする度に、VPN側ではエッジルータのコンフィグをアップデートする必要がありました。しかし、クラウドとVPNの相互接続点にSDN/OpenFlowを導入した、この新しいモデルでは、SDNコントローラでクラウド側のテナントネットワークとVPNをマッピングし、経路のアップデートをBGPで広告することができるため、VPN側のエッジルータをコンフィグする必要がなくなります(**図7**)。

技術的なお話は、またの機会があれば詳しく紹介したいのですが、このように「これから」のネットワークと「これまで」のネットワークをつなぐ手段としてのSDN/OpenFlowは、通信事業者にとっても非常に有用なものであると考えています。SD

▼図7　NTTコミュニケーションズ㈱によるSDN/OpenFlowの実現例



CHAPTER

4 Undocumented『マスタリングTCP/IP OpenFlow編』

Writer あきみち／Geekなページ（http://www.geekpage.jp/）

10月に行われたネットワークスペシャリストの国家資格試験で午後Ⅱの問題が丸ごとOpenFlowだったのは多少びっくりしましたが、ここ2年ぐらいOpenFlowが話題です。そんななか、SD 12月号がOpenFlow特集ということで、『マスタリングTCP/IP OpenFlow編』の共著者として原稿執筆依頼をいただきました。拙著をベースにOpenFlowについて語っても良いとのことだったのですが、単純にOpenFlowそのものを解説してもおもしろくないと思ったので、今回は書籍に書いていないことを多めに書こうと考えました。

OpenFlowが日本で注目される理由

まず、最初にOpenFlow本を執筆するに至った経緯から紹介します。多少乱暴な表現ではありますが「OpenFlowが日本でだけ妙にバズっているのは恐らくSI案件として使いやすいからではないか？」といったブログ記事を書いたことがトリガーでした。「OpenFlowが日本で流行る理由」というタイトルのブログ記事をオーム社開発局の方が読み、「OpenFlowの本を書きませんか？」というオファーをいただきました。ただ、私は標準化にかかわっているわけでも、OpenFlow製品開発にかかわっているわけでもなく、実情に即した書籍を書き上げる自信がなかったので、世界で初めてOpenFlow対応ハードウェアを発売し、かつ、執筆開始段階において恐らく世界で最もOpenFlow導入実績があるNECの方々に共著をお願いしました。ということで、1つのブログ記事がOpenFlow本が産まれる背景になったわけです。OpenFlowが日本で「話題になる理由」であって「流行る理由」じゃないという反省点はあるものの、私がそのブログ記事で書いたような感想を持ったのは、OpenFlowに次のような特徴があると考えているためです。

- 柔軟性が高く、いままでやりにくかったことが可能
- ただし、案件にあわせて作り込む必要あり
- 案件に適した設計コンサルティングが必要
- 作った人しかメンテナンスできない可能性
- OpenFlowコントローラの実装や設定方式によっては、ロックインが可能

そのうえで、発祥地である米国ではなく日本での話題のほうが盛り上がった理由として、日本と米国の環境の違いがあると考えています。日本では自社開発ですべてを解決しない会社が多く、とくに中小企業はIT系エンジニアを自社で抱えずに社内システムなどをSIerに依頼する傾向があります。

一方、米国はIT系エンジニアを雇用する傾向があります。もちろん外注も行いますが、技術を理解しつつ意思決定可能なIT系エンジニアが雇われていることが日本と比べると非常に多い印象です。

こういった違いから、「個別のお客様のために作り込みができて、かつ、お客様を逃がさない」というしくみが魅力的なのだろうという感想です。

Interop Tokyo 2013での取り組み

拙著の執筆が佳境に差し迫っていた6月に、幕張メッセでInterop Tokyo 2013が開催されていましたが、毎年行われている主催者企画であるShowNetで、非常におもしろいOpenFlow

活用法が展示されていました。OpenFlowを利用して、ShowNetバックボーンの任意の場所から、任意のトラフィックを監視用セキュリティ製品へ転送するというデモでした(**図1**)。

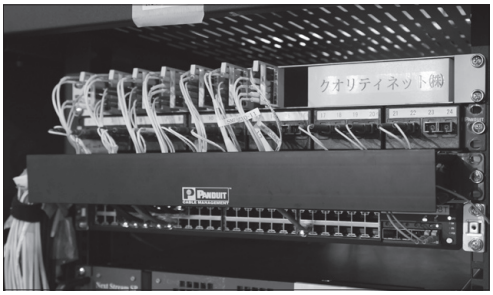
OpenFlowを利用して、単にパケットを転送するだけであれば、数年前から同様の取り組みが行われていますが、今年のShowNetで特徴的だったのが実現規模とそれによる柔軟性だったと思います。現場の方が「ネットワーク全体に対して指定可能なtcpdumpのフィルタみたいなもの」と表現されていましたが、まさにそういった感じです。

「このスイッチの○番ポートとこのスイッチの×番ポートのTCP 80番トラフィックだけを監視したいなぁ」と思えば、そういった設定ができてしまいます。ネットワーク全体を詳細に解析し続けるのではなく、必要なときに必要なだけ監視することが可能な環境を物理的な設定変更をせずに行えるのは非常におもしろいと思いました。

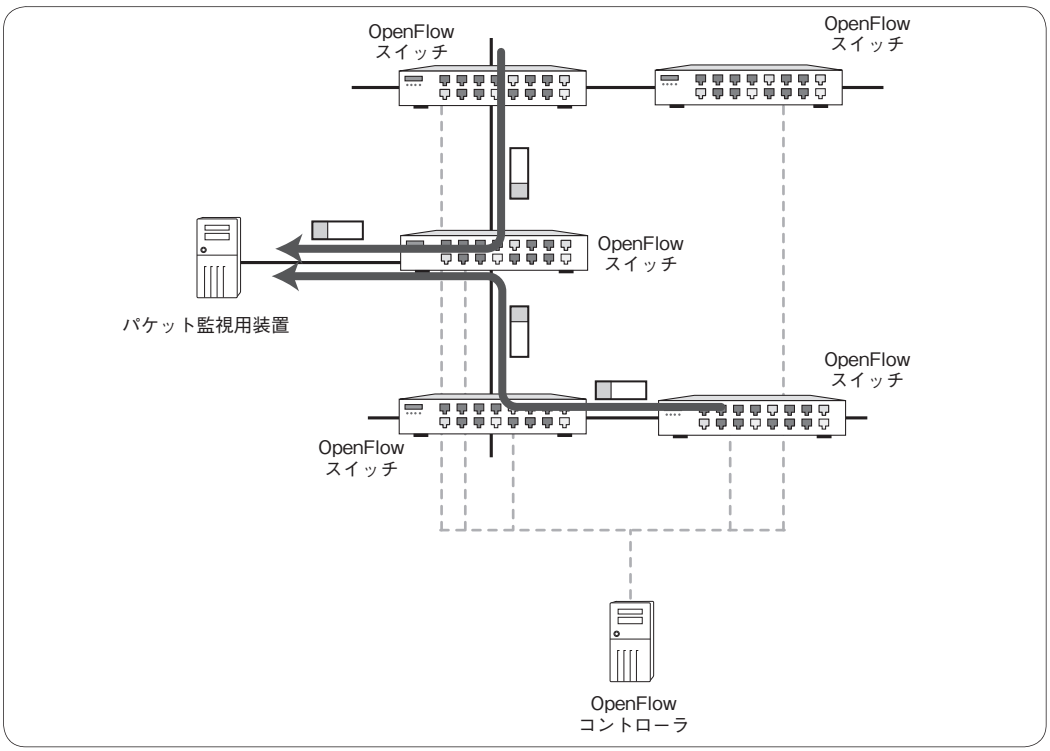
このSDN Security企画の肝は、ShowNetバックボーンネットワークに大量のファイバータップが設置された環境であるという点です(**写真1**)。ファイバータップによってコピーされたパケットは、OpenFlowスイッチを経由して監視装置へと転送される構成で運用することによって、実トラフィックにまったく影響を与えずに監視を行うことができていました。

実は、これ以外にもShowNetでOpenFlow

▼写真1　ファイバータップによるパケットコピー



▼図1　複数のOpenFlowスイッチから特定のフローだけを監視装置へ



企画はありました。その内容の紹介としては、SDN Japan 2013で行われた発表が非常にわかりやすいので興味のある方は、動画アーカイブをご覧いただくことをお勧めします[注1][1]。

OpenFlow 1.3

OpenFlow 1.0策定後、OpenFlowの仕様は、急激な勢いでバージョンアップしました。OpenFlow 1.0が2009年12月、約1年後の2011年2月にOpenFlow 1.1が策定され、約1年半後の2012年6月にはOpenFlow 1.3が策定されました。バージョン番号としては0.1ずつの上昇ですが、各バージョンアップでドラスティックな変更が加えられています。あまりに急激に仕様が変わったため、ハードウェアベンダーの実装が追いつかず、OpenFlow対応ハードウェアの多くはOpenFlow 1.0対応のままでした。そのため、OpenFlow 1.3標準化後に、OpenFlow標準化活動を行っているONF(Open Networking Foundation)でOpenFlow 1.3を相互接続性確保を行うバージョンにするという合意がなされ、OpenFlow 1.3での十分な相互接続性が確保されるまでは、さらなる仕様追加はしないとしていました(ただし、1.3.Xというマイナーバージョンアップは行われています)。

このような背景があり、OpenFlow 1.3は、ハードウェア実装間の相互接続性確保のためのバージョンという位置づけになりました。OpenFlow 1.3に関して大きな動きがあったのが、今年6月です。米国のインディアナ大学で行われた第3回プラグフェストでのOpenFlow 1.3相互接続試験と、幕張メッセで行われたInterop Tokyo 2013のOpenFlow 1.3.1相互接続性試験が、偶然だとは思いますが、ほぼ同時期に行われました。これらのイベントで、OpenFlow 1.3が相互接続が可能な状態まで実装されていることが世に示されました。その後、各社がOpenFlow 1.3対応機器を正式発表し、徐々にOpenFlow 1.3製品が増えてきました。

OpenFlow 1.4

OpenFlow 1.3で十分な相互接続性が確保されるまで凍結されていたOpenFlow新仕様も、OpenFlow 1.3相互接続性試験の成功とほぼ同時に再開されました。ただ、新仕様の策定は表面的には凍結されていたものの、議論そのものはONF内で行われていました。『マスタリングTCP/IP OpenFlow編』出版の次の月である、今年8月にOpenFlow 1.4の草案が公開されました。OpenFlow 1.3の相互接続性試験から2ヵ月後です。その後、10月15日に正式版として発表されました。

OpenFlow 1.4の代表的な変更はいくつかありますが、個人的に最も着目したのがOpenFlowコントローラとOpenFlowスイッチの間の通信チャネルであるOpenFlowチャネルのデフォルトTCP/UDPポート番号変更です。これは、2013年7月にIANA(Internet Assigned Numbers Authority)がOpenFlow用としてTCPおよびUDPのポート番号として6653番を割り当てた、という外的要因によるものなのです。OpenFlow 1.0から1.3.2は6633番をデフォルトのポート番号としていましたが、OpenFlow 1.4からは6653番に変わります。もう1つの大きな変更として、光デバイス関連の仕様追加が挙げられます。OpenFlow 1.4では、光ポートにおけるレーザー光の周波数やパワーの設定や監視が可能となる機能が追加されています。この機能はEthernetだけではなく、光回線交換機でも利用可能な仕様になっています。

OpenFlow 1.4/1.5は実装されるのか?

OpenFlow 1.4が出たばかりですが、年内にもOpenFlow 1.5が公開される予定であると言

注1)「INTEROP Tokyo 2013 ShowNetにおけるSDNの実際」Interop Tokyo 2013 ShowNet NOCチーム:中村遼(http://www.youtube.com/watch?v=MiRI6Trh3SA)電子情報通信学会の論文としても発表されています。興味のある方は、そちらもぜひご覧ください。

われています。OpenFlow 1.5の具体的な内容は、本稿執筆時点ではまだ公開されていませんが、これまでOpenFlowのバージョンが0.1上昇するたびに大きな変更があったので、OpenFlow 1.5でも何らかの大きな変更がありそうな気がします。そうなると、これらのバージョンがいつどのように実装されるのか、もしくは、そもそもこれらのバージョンが実装されるのかという点が気になります。過去を振り返ると、OpenFlow 1.1と1.2は、ほとんど実装されませんでした。個人的な感想としては、OpenFlow 1.4が実装されるかどうかはさておき、デフォルトのポート番号は6653番に変更されていくのではないかと考えています。IANAが決めたわけですから、おそらく、多くのベンダーがそれに従うだろうという感じです。

ただ、後方互換性を考慮して6633番と6653番の両方のポート番号を試してみるような実装が行われるかもしれないとも思います。もう1つは、光デバイス系の部分は比較的早期に人知れず評価機ができあがるか、もしくはすでに世界のどこかにあるのではないかと推測しています。光デバイス系のスペックは、光海底ケーブルや、米国国内でのデータセンター間通信など、長距離での用途を想定しているような気がしています。そういった案件は、件数としては少なくても、1件の金額が大きい傾向があるので、そういった案件専用に作り込まれて表には出ない可能性も考えられるためです。OpenFlow 1.4の光デバイス系のスペックに限らずですが、OpenFlowのスペックって、「この部分って実際の案件を前提として仕様を足しているんじゃないか?」と思える項目があるという感想を私は持っています。

これからのOpenFlow

OpenFlowはまだまだ変化し続けています。2年前に「OpenFlowの欠点」と言われていたような話の多くはOpenFlow 1.0を想定して語られていましたが、その後の変化で状況が変わった話も多いので、OpenFlowに関して何かを調べるときには、バージョンを意識するとともに、最新バージョンで何ができるようになったのかに関しても十分な注意が必要そうです。一方で、OpenFlowが10年後も残っているのかどうかは、現段階ではわからないというのが筆者の感想です。ただし、たとえ途中で消えてしまったとしても、日本でのOpenFlow熱や、その後にバズになっているSDN熱が何も残さないわけではないとも考えています。OpenFlowという考え方や、それから派生して発展したSDNは、今後のネットワーク関連商品に大きな影響を与えると思われます。これは、どちらかというとOpenFlowが起こした流れというよりも、仮想化やクラウドといった非常に大きな流れにOpenFlowがうまく乗れたからではないでしょうか。そういった意味では、OpenFlowそのものの仕様を理解しつつ、なぜそれが大きな注目を浴びたのかという部分にこそ本当のおもしろさが潜んでいると思える今日このごろです。

最後に書籍紹介

最後に『マスタリングTCP/IP OpenFlow編』を紹介させてください[注2]。拙著は、「OpenFlowがどのようなプロトコルであるか」や「OpenFlowで何ができるのか」をできるだけわかりやすく伝えることを目指すと同時に「OpenFlowで何ができないのか?」も適切に紹介することを意識しました。何か新しいプロトコルが登場し、それが実際に使えるのかどうかを判断するには、「何ができるのか?」も大事ですが、それよりもむしろ「何ができないのか?」を意識しないととんでもない落とし穴にハマってしまう可能性があるためです。

OpenFlow 1.3.2までを解説しているのも、

注2) DRMなしPDFの電子書籍版もあります(http://estore.ohmsha.co.jp/titles/978427406920P)。

拙著の大きな特徴です。本書が出版された時点では、OpenFlowの最新バージョンは1.3.2でしたが、同書はOpenFlow 1.0を骨子としつつ、OpenFlow 1.3.2までを解説するという形式にしました。OpenFlow 1.0を基本としているのは、OpenFlow 1.0でOpenFlowの初期設計思想が色濃く反映されているため、OpenFlowを理解するにはOpenFlow 1.0から入るのがわかりやすいと考えたためです。最初からOpenFlow 1.3で全部を解説することも検討しましたが、そうしてしまうと、なぜ各機能が追加されていったのかを理解しにくい書籍になってしまうのではという懸念がありました。

実際、OpenFlow 1.0までは「既存のハードウェアを活かしつつソフトウェアの変更でおもしろいことをする」という思想がありましたが、OpenFlow 1.3は、OpenFlowの旧バージョンではできないことを実現するために拡張されていった結果であるという側面も感じられます。プロトコルに対する変更は、必要に応じて各所で個別に言及するとともに、OpenFlow 1.1、1.2、1.3の各バージョンについては、それぞれ1章分を割いて解説しています。逆に言うと、それぞれ1章を割けるぐらい、各バージョンの違いが大きいとも言えます。ということで、拙著をお楽しみください。ありがとうございます。

SD

●参考文献

[1]IPバックボーンにおけるOpenFlowを使ったモニタリングネットワークの構築 Interop Tokyo 2013におけるShowNet SDNの取り組み　著者：田原裕市郎（KDDI）・齋藤修一（NEC）・石井秀治（NICT）・関谷勇司（東大）　論文情報：信学技報, vol. 113, no. 256, IA2013-48, pp. 33-36, 2013年10月発表 URL（http://www.ieice.org/ken/paper/20131022UBGT/）研究会 - IPバックボーンにおけるOpenFlowを使ったモニタリングネットワークの構築・Interop Tokyo 2013におけるShowNet SDNの取り組み（www.ieice.org）

CHAPTER 5

SDN/OpenFlowに欲しい機能／要らない機能

Writer 伊勢 幸一(いせ こういち)／Twitter@ibucho／㈱データホテル

IT(Information Technology)とICT(Information and Communication Technology)の溝がIT業界にはあるのではないでしょうか。SDN/OpenFlowをどう利用するのか、そこに現場で事業者が戸惑いとためらいを感じるところです。データセンター事業者側と使用するユーザのメリットの乖離、それも理由の1つです。しかしながらSDN/OpenFlowの実装においてPLUMgrid Platformというフレームワークが生まれています。本稿ではその評価も踏まえ現状から近い将来への展望を紹介します。

求めているのはConnectionではなくFunction

筆者が聞きおよぶ限り、通信キャリアやISP、IXPといったネットワークサービスプロバイダでは、盛んにSDNの導入を検討し、さらにサービス環境への配備も積極的に進められています。ただし現段階では、SDNとはいってもOpenFlowデータプレーンとコントローラによるSDN構成が多く、VXLAN(Virtual eXtensible LAN)やSTT(Stateless Transport Tunneling)、NVGRE(Network Virtualization using GRE)といった2011年以後に提案され、おもにクラウドIaaS基盤向けのトンネルプロトコルを適用しているケースは少ないようです。

また、SDNソリューションを提供しているベンダーによると、本来、通信キャリアやISPなどの通信サービス事業者以上に、データセンタ、ホスティング、クラウドIaaS事業者による採用を推進すべく、各事業者と評価検証などを進めてはいるそうですが、なかなかこれら事業者へのSDN導入には至らないようです。なぜ、SDNソリューションはすでに通信サービス事業者には導入されているにもかかわらず、データセンターなどのサーバインフラ事業者には受け入れられないのでしょうか?

ITとICTの混同

問題はICT業界において通信サービスとサーバインフラサービスを混同していることが原因ではないかと考えます。2000年以前、我々は情報処理技術をIT(Information Technology)と呼び、コンピュータやLAN、オペレーティングシステム、半導体、ストレージ、アプリケーションに関連する産業をIT産業、IT業界と称していました。このころ情報と通信は別々のカテゴリであり、どちらかと言うと通信は無線や放送とともに「放送通信業界」を形成していたと記憶しています。ところがインターネットの発達に伴い、従来からの無線放送界より、有線回線によるTCP/IP通信接続はサーバやLANと同じIT業界との親和性が高いという理由で、情報通信技術、すなわちICT(Information and Communication Technology)というカテゴリとして再編されることになります。このカテゴライズが現在に至るまでさまざまな技術やソリューション、そして製品開発などへの齟齬と混乱を招くことになりました。

コミュニケーションの意味の履き違いがありませんか?

そもそも2000年前後にWWW(World Wide Web)システムを基盤とした検索サービスやブログ、SNSなどがインターネットの主流コンテンツとなり、ICTという技術カテゴリが囁か

れ始めたとき、ICTのC(Communication)は英語でいう「通信」ではなく、ラテン語のCommunication(分かち合う)、つまり「共有」という意味で用いられ、それは前述のような検索エンジンやソーシャルメディアなどによって情報を共有する技術を指すものとされています。つまり、コンピュータやLANなどのハードウェアインフラだけではなく、さまざまなアプリケーションやサービスによって情報を共有する技術をも統合する技術カテゴリとしてICTが定義されたのです。ところが、そのICTというキーワードが各国に拡散する過程でCommunicationが「通信」と翻訳され、通信回線やインターネット接続サービスとコンピュータ、LAN、Webサーバ、HTMLアプリケーションを同じカテゴリとして議論することになってしまったのでしょう。

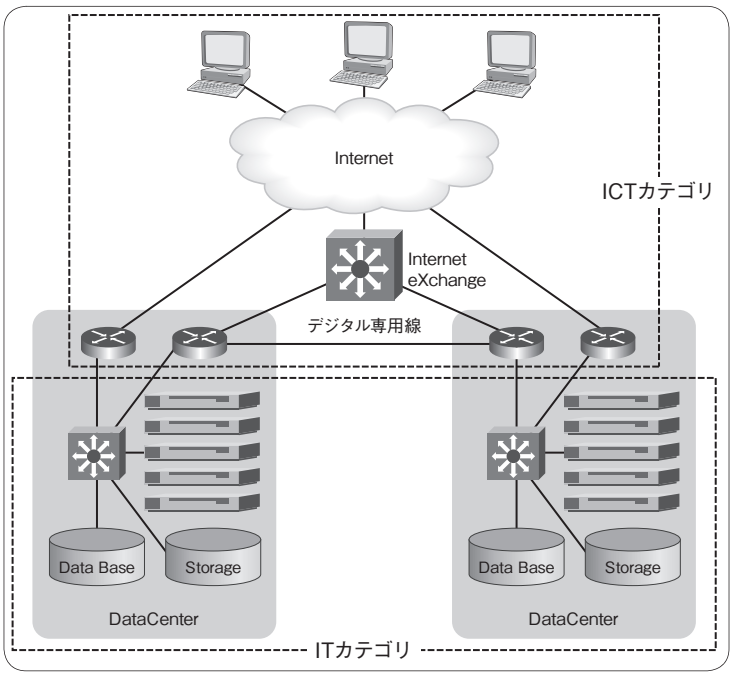
しかし、いわゆるコンピュータ、サーバ、ストレージ、LAN、アプリケーションという旧来ITとしてカテゴライズされてきた分野と、デジタル専用線、フレームリレー、ATMといった中長距離回線によるインターネット相互接続、そしてそのインターネットバックボーンなどの通信環境を提供する分野とでは、必要とされる技術、システム規模、そしてビジネスモデルに大きな差異があり、同じカテゴリとして論じることができない場合が多々あります。

とくに仮想ネットワークやSDNといった技術は同じネットワークであっても通信事業者とデータセンタ、ホスティング事業者では用途用法、とらえ方、機能的要求がまったく異なります。すなわちITは本来の意味でのICT、つまり情報とそれを共有する技術を指し、ICTは情報を対象とした通信技術そのものを指すと解釈したほうが良いでしょう。政府の担当省庁の役割分担で言うと、総務省の担当範囲がICTであり、経産省の担当範囲がITです。その証拠に総務省のHPでは「情報通信技術」を「ICT」で示し、経産省のHPでは「IT」と記述されています(**図1**)。

したがって、IT事業者とはインターネットとWebブラウザなどによりEコマースやオンライン予約販売などのサービスを提供したり、デジタル画像や動画、オンラインゲームなどのコンテンツ提供したり、ブログ、SNSといったソーシャルメディア、ニュースサイトの運営、そしてそれらインターネット上のコンテンツと利用者のマッチングを行う検索サイトなどを示します。また、それらサービスを提供するためのアプリケーションやミドルウェア、フレームワーク、プラットフォーム、そしてコンピュータとストレージ、コンピュータ間を接続するLANといったコンポーネントをリソースとして提供する事業者も同じカテゴリに属するでしょう。

本稿ではアプリケーションやミドルウェアではなく、おもにサーバインフラについて議論を

▼図1　ITとICT



進めていくので、データセンタ、ホスティング、レンタルサーバ、クラウドIaaSプロバイダのようなインフラ事業者をIT事業者とし、そのようなIT事業者に対してインターネットへのグローバルコネクティビティやサイト間の広域接続などを提供する通信キャリア、ISP、IXPといった通信サービス事業者をICT事業者と呼ぶこととします。

ビジネスモデルと投資

両者のビジネスモデルの違いを一言で言うと、IT事業者はコンピューティングリソースを提供し、ICT事業者は通信環境を提供することで対価を得るということです。現在議論され提案されているSDN製品、もしくはサービスとして提供されているSDNソリューションのほとんどは、仮想ネットワーク技術などを駆使し、コンピューティングインスタンス間にコネクティビティを形成することを目的としています。前述のようにインスタンス間の通信環境、つまりコネクティビティを提供することで対価を得るICT事業者にとって、SDNを導入することにより敏速かつ柔軟なコネクティビティサービスを提供し、かつ広域に配置された多数の機器を一極集中型で管理することで運用コストを軽減することができればより大きな対価を得られる可能性があり、SDNに投資する意義があります。

しかし、サーバやプラットフォーム、ストレージなどを提供して対価を得るIT事業者には、サーバ間、ラック間といったコンピューティングインスタンス間の接続性を課金するサービスメニューは存在しません。一般的にデータセンタやホスティング事業ではサーバ数やサーバのスペック、メモリ、ストレージ容量に対して課金しますが、サーバ間やラック間、フロア間の接続性品質に対して異なる料金設定をしている事業者はありません。通信環境に対する課金はインターネットへのグローバル帯域に関する料金設定のみです。つまり、SDNによってデータセンタ内のネットワークを仮想化し、敏速かつ柔軟なトポロジーに対応したところで、顧客への請求書に加える項目が存在しないのです。基本的に企業が実施する投資は何らかの手段で回収するために行う行為であり、回収の見込みがない対象への投資はありえません。したがって、現段階でICT事業者向けソリューションとして提供されているSDNソリューション、製品やサービスがIT事業者に受け入れられないというベンダーの悩みは当然といえば当然のことなのです。

例外として現時点ではなく将来的事業展望に対しての人材育成やエンジニアリングインキュベーションとして先進的技術を研究検証目的で導入する可能性はあります。またサービス運営会社がクラウドIaaS基盤の設計と実装をアウトソースしたところ、納品されたシステムのネットワークがSDNによって形成される設計になっていた場合もあるでしょう。しかしIT事業者がこれらSDNを実際に提供しているサービスに導入するメリットはまったくありません。

IT事業者の価格設定

たとえIT事業者であっても、コンピューティングリソースを提供するにはネットワークが必要です。ルータやスイッチなどのネットワーク機器や物理的にそれらを接続するメタルケーブル、ファイバーなどの敷設コストを負担しなければなりません。また、サーバ上のアプリケーションがインターネットに向けてデータを送信するため、パブリックピアリング、プライベートピアリングを行うための接続回線やトランジット費用もかかります。インターネットトランジットサービスのように従量制で課金されるサービスであれば、そのまま顧客が利用するアップリンク回線帯域に則した課金を設定できますが、どの顧客がどれくらい消費しているかを分析することが困難なTORスイッチ、エッジスイッチやバックボーンルータの費用を顧客別に料金

メニュー化することは現実的に不可能です。通常、これらの費用はサーバ台数やラック本数に比例した按分によってサーバ料金に重畳し提供価格を決定しています。したがって、ホスティングサービスを提供するうえで、物理ネットワークに加えSDNを導入してサーバ間ネットワークを強化したとしても、その投資分を回収するためサーバ料金を途中から値上げすることなどできません。サーバ間接続が物理なのか仮想なのか、もしくはどのベンダーのネットワーク機器を使っているかということはサーバの利用者にとってどうでも良いことであり、それらはすべてIT事業者側のリスクであり負担でしかないのです。

IT事業者にとって有効な投資になっているか

それではIT事業者が有償で提供するネットワークサービスはないのでしょうか。実は、スペック別のサーバやストレージ容量、インターネット帯域以外にも一般的なIT事業者がオプションとして有償提供しているネットワークサービスがあります。

ほとんどのIT事業者ではサーバやインターネット接続に加え、テナントドメインに対するファイアウォール機能、NATやNAPT設定、VPN、IPS/IDS、そしてサーバ間やサーバストレージ間のボンディング(冗長)化、バックボーンに対するLAG化など、それぞれオプションとして有料メニューに記載されています。つまり、単純なインスタンス間のコネクティビティではサーバ利用料金以外にオプションとして課金することはできませんが、接続性ではなく、ネットワークの機能であれば有償で提供できます。したがって、ネットワーク機能を提供するSDNソリューションであればIT事業者にとって投資対象となり得ます。このように、IT事業者が求めるSDNとはネットワーク接続(Connectivity)ではなく、ネットワーク機能(Function)をプログラマブルかつソフトウェアで動的に定義するソリューションなのではないでしょうか。

ベンダー実装はどうなっているのか

現状、さまざまなベンダーがSDNソリューションを提供していますが、シスコシステムズやジュニパーネットワークスのようなルータ・スイッチ機器メーカーでは各社の機器によって構成する仮想ネットワークセグメントやパス、ルートシグナリングをコントローラから定義することを主旨としており、SDNソリューションとしてマルチベンダーのデータプレーンをサポートすることを前提とはしていません。マルチベンダデータプレーンとのインターフェース、すなわちSouthbandフリーとしてSDNソフトウェア・パッケージを提供するベンダーもあります。ストラトスフィアのSSPにはvFirewallという機能が提供されていますが、これらもVMwareのNVPやミドクラのMidonetと同じく、主たるソリューションは物理ネットワーク上に仮想ネットワークパスをプログラマブルに形成することを目的としています。

また、ヴィエムウェアのvCenter/vCloudやMicrosoft の System Center Virtual Machine Managerといったクラウド IaaS基盤コントローラでは、仮想マシンやストレージだけではなく、仮想ネットワークの形成も担い、当然のことながらDHCPサーバ機能やロードバランサ機能、MTU Pass Discoveryレスポンス機能なども有してはいます。しかし、それらはIaaSコントローラとしての機能だったり、仮想ネットワークプロトコルに潜在するMTUフラッグメント問題に対する解決策として実装されている機能であり、当然のことながらプログラマブルかつダイナミックにネットワークを形成するものではありません。また国産では、あくしゅが提供するwakame-vdcでも同様のネットワーク機能を提供してはいますが、それらもIaaSコントローラの機能の一部として実装されているにすぎません。

PLUMgrid Platformに注目する理由とは何か

純粋なSDNソリューションとしてネットワーク機能を提供している製品の1つにPLUMgridのPLUMgrid Platformがあります。PLUMgridは2011年に北米のサニーバレーで設立され、その後ステルスモードにありましたが、今年2013年の5月にそのステルスモードが解かれ、彼らの製品をトライアルで利用できるようになりました。PLUMgridという社名はParse、Lookup、Update、Modefyというネットワークインスタンスがパケットに対して行う処理の頭文字を取って命名されたそうです。

PLUMgrid Platformは一言でいうと仮想ネットワークインフラストラクチュアであり、一般的なSDNソリューションと同じく、クラウド基盤上で展開されたマルチテナントグループに対して仮想ネットワークを提供します。特徴的なのは単に仮想ネットワークセグメントを提供するだけではなく、ファイアウォール、ロードバランサ、NAT、VPN、DHCPといった従来IaaSコントローラやクラウドサービスのオプションとして個別に提供されていたネットワークサービス機能を網羅しています(**図2**)。しかも、PLUMgridの製品は単なる仮想ネットワークコントローラではなく、ネットワーク機能をサードパーティが実装し、配備するためのSDKフレームワークが用意され、PLUMgridから提供されていない機能をユーザ側で追加していくことを可能としています。

たとえば、もしテナントがサーバ間接続においてIPv4ではなくIPv6を利用したい場合、PLUMgridからは提供されていないIPv6 RA(Router Advertisement)サーバやDHCPv6サーバをサードパーティがSDKによって実装し、配備できます。

PLUMgrid Platformを構成するコンポーネントは次のとおりです。

1)Director

IaaSコントローラに相当する部分でOpenStackで言うところのNovaです。

2)IO Visor

仮想ネットワークオーバーレイを提供するエンジンでありIaaS基盤におけるハイパーバイザに相当します。

3)Network Functions

IO Visor上で動作し、各Virtual Domainsごとにネットワーク機能を提供するモジュール群です。

4)Virtual Domains

IO Visorによって形成されるテナントごとの仮想ネットワークセグメントです。

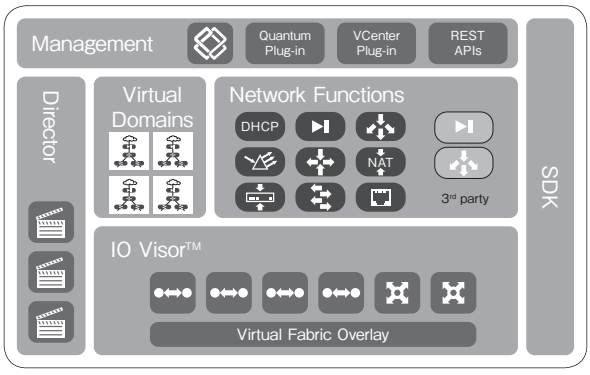
5)Management Interface

Northband APIであり、IaaSクラウド基盤全体を制御するコントローラ、オーケストレータとのインターフェースと、コマンドラインコンソールインターフェースが用意されています。現段階でサポートしているIaaSコントローラはOpenStackとヴイエムウェアのvCenterです。

6)SDK

カスタムネットワーク機能を実装し配備するための開発フレームワークです。

▼図2　PLUMgridコンポーネント



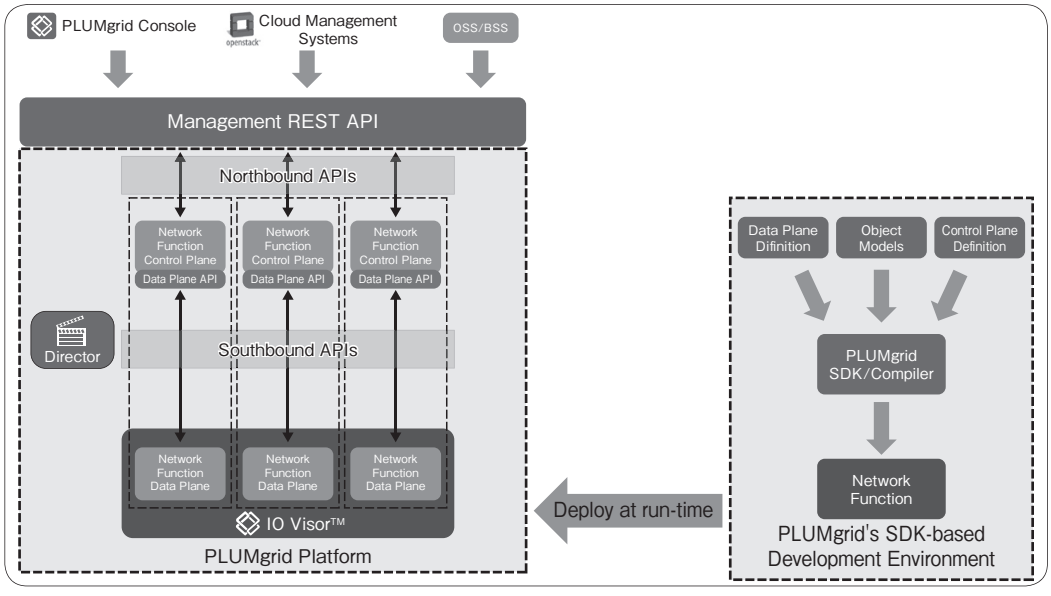
Southband

従来、SDNコントローラのSouthband APIの活用はOpenFlowのデータプレーンであるOpenFlowスイッチとOpenFlowコントローラとの通信に限られていました。OpenFlow以外のデータプレーンとはネットワーク機器メーカーが提供するハードウェアや、KVM、Xen、vSphereといったハイパーバイザであり、これらの機器メーカーやベンダーがサードパーティ製SDNソリューションの提供するAPIを用いて製品を開発するとは考えにくく、多くの場合、自社開発のSDNコントローラ専用インターフェースを実装するだけにとどまっています。また、たとえ利用者がSDNを導入したとしても仮想ドメインに提供するネットワーク機能はこれらハードウェアやハイパーバイザが提供する機能に限定されるため、コントローラのSouthband APIを活用して新たなネットワーク機能をSDNに実装するというケースもほとんどないでしょう(**図3**)。

このようにSDN側で独自にSouthband APIによってネットワーク機能を拡張し、仮想ドメインに新たな機能を提供するというパラダイムはあまり検討されていませんでした。しかしPLUMgridはそのハードウェアやハイパーバイザの上にIO Visorという独自のネットワークオーバーレイと、そのオーバーレイに新たなネットワーク機能を配備する開発フレームワークを提供することで、ユーザやサードパーティがネットワーク機能を拡張していくことを可能にしています。

PLUMgridはステルスモードを解除したばかりのまだ若いプロダクトであり、現在引き続き開発と実験が継続されている状況にあります。しかし、IO Visorをハードウェアスイッチやカーネルに実装し、ハイパーバイザ上のVMだけではなく、既存の物理サーバも仮想ドメインネットワークに統合するベアメタルソリューションも用意し始めています。従来のSDNソリューションが通信キャリア志向の製品であったため、データセンター・ホスティングといったIT事業者にはなかなか受け入れられないプロダクトでしたが、IT事業者の状況や事情をよく理解したPLUMgridソリューションは今後注目していきたいソリューションの1つでしょう。SD

▼図3　ネットワーク機能の実装と配備



CHAPTER 6

データセンター技術から見たSDN/OpenFlow技術の影響とは

Writer 杉田 正(すぎた ただし)／㈱LXスタイル

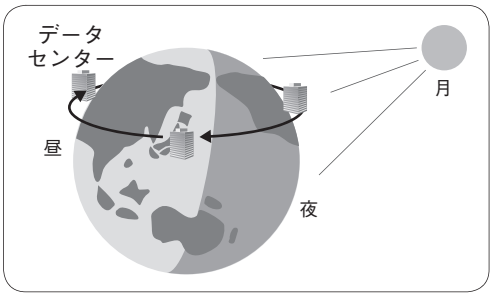
SDN/OpenFlowの普及は、世界レベルで大規模なWebサービスを運営しているGoogleなどから始まっているようです。本稿では、データセンター開発者の視点からSDN/OpenFlowがどのように使われていくのか、電力消費やハードウェアの性能向上などさまざまな要素から解説をします。

究極の省電力対策「月を追いかける」

Googleは月を追いかけています。同社はデータセンターからサーバ、アプリケーション、エンドユーザサービスまで1社で構築し、それらを全体最適化して省電力化に努めています。たとえば検索を100回すると約60Wの電力を消費するという試算があります。このことからわかるように膨大な電力を消費します。なお、物理的なサーバ台数は100万台近くとも言われています。そのためデータセンターを省電力化すれば、大きな利益を生み出します。

電気料金体系は、使用するほど高い料金を払うしくみです。サーバを収納するデータセンターが世界各地にあれば、発電能力が余っている地域の「夜間の安い電力」を利用できます。夜間電力を契約し「夜の地域のデータセンター」から「昼間の地域」へネットワーク経由でさまざまなサービスを提供できます。つまり低コストでデータセンターを運用できるのです(**図1**)。Googleは、昨年OpenFlow/SDNを導入したと発表しました。それは「月を追いかけるデータセンター」を実現したのでしょう。

▼図1　地球と月(安価な電力を求めてデータセンターをコントロールする)



ますます増加するデータセンターの電力消費

日本でも電力の2～5%を"データセンター"だけで消費するという予測があります(**図2**)。しかし、これには「スマートフォンの登場」が含まれていません。スマートフォンのバッテリーは小さいものですが「検索」を使えば、データセンターでサーバが稼働し大きな電力が消費されます。iPhoneのSiriのような音声認識技術は、データセンターで処理をしなければ実現できません。

Ericsson社 の「Ericsson-Mobility-Report」では今後5年間でモバイルによる通信量が6倍になると予想しています(**図3**)。

仮想化が生み出す、高密度なサーバ環境

データセンターの省電力化は、サーバの仮想化が手始めです。当初4分割程度から始まった仮想サーバも、SSDやPCI-Expressを使う超高速なストレージの登場により50分割、メモリを1TBや2TBほど大量搭載すれば200分割も可能などと言われてきました。さらにLXコンテナの超軽量仮想環境技術によって、1台の

サーバに1万台の仮想サーバという従来にはなかった環境が実際に稼働しています。

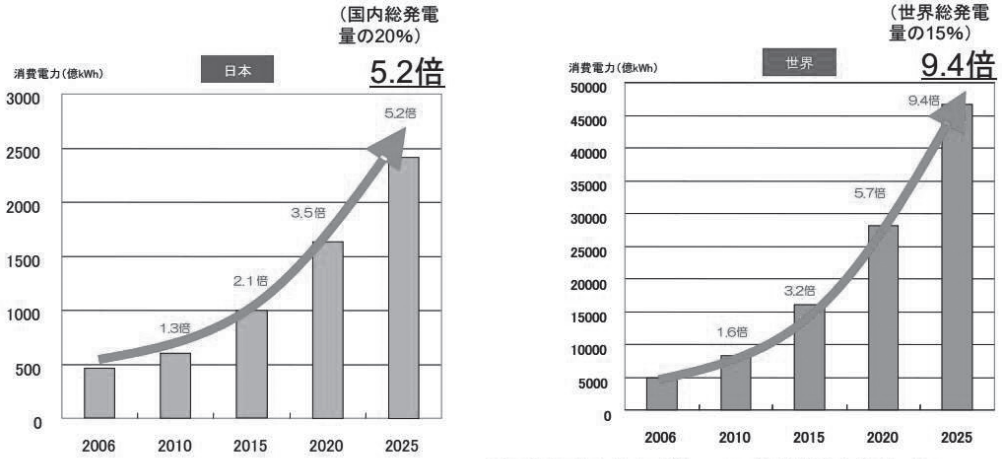
これらのサーバ環境を構築するためにさまざまな仮想化サーバ構築ツール、OpenStackやCloudStackなどが登場しています。仮想化が進むと、サーバ稼働を停止せずに引越しできるライブマイグレーションが使えるようになります。

クラウド事業者は、WANをLANのように使う

世界のクラウドサービス提供者は、ハードウェアを仮想化してSDN型データセンターを構築し、1つのラックに異なるサービスを行うサーバやスイッチを収納して効率的かつ適材適所なサービス実施ができるインフラを整備してきました。

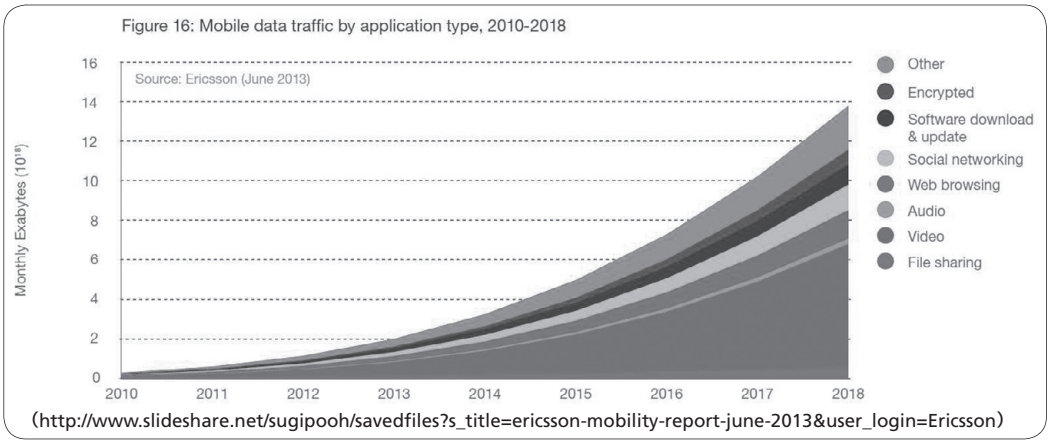
GoogleやMicrosoftのWindows Azureは言うまでもなく、Facebookも100万台に到達し、AWS(Amazon Web Service)やAppleなども巨大データセンターへの莫大な投資を続けています。彼らが対象とするユーザ(クライアント)は、PCに加えてスマートフォンです。Amazon Kindleは、バッテリー駆動時間を延ばすためにデータセンターで処理したデータを受け取るブラウザを搭載しています。これらクラウド事

▼図2　2010年経産省資料



（出所）経済産業省/グリーンIT推進協議会試算(2008)
(http://www.meti.go.jp/committee/materials/downloadfiles/g80520c03j.pdf)

▼図3　ericsson-mobility-report



(http://www.slideshare.net/sugipooh/savedfiles?s_title=ericsson-mobility-report-june-2013&user_login=Ericsson)

業者は、世界中でサービスを行うため、世界中にデータセンターを構築しています。データセンター間を太い回線で接続してWANをLANのように使い、電力やサーバ、ストレージ資源を効率よく運用しなければならないのです。

SDNとデータセンター運用

SDNという言葉を聞いたとき、インターネットの基幹部分が変わると感じました。従来からのネットワーク機器であれば、物理的に区切られた場所にサーバを置き、事前に設定されたL3ルータなどからEthernetケーブルを接続しているだけなので、L3ルータを2重化(冗長化)するか、あらかじめ設定された予備機を準備しておけば、LAN機器の故障では予備機に交換する作業だけで短時間に復旧対応できました。

SDN構築後「ネットワークがつながらない」「ネットワークが遅い」など障害が発生した場合、短時間に復旧できるスキルがSDN構築担当者や運用担当者にあるのでしょうか? また、メーカーや代理店も含めて1分1秒を争う時間で復旧支援が可能なのでしょうか? Googleの事例のように、SDNはこれからのネットワークに欠かせない技術と思われます。しかし、規模が小さなデータセンター事業者にとっては障害発生時の対策コストは大きく、SDNは当面リスクの高い導入メリットの少ない技術に見えます。

データセンターでSDNを使うメリット

従来のように時間をかけてネットワークを構築するのではなく、ソフトウェアでネットワーク構成を変更できるのであれば、故障対策用HUBも同一製品のストックだけでよくなります。

サーバの種類には省電力型から大容量メモリ・高速ストレージを使うハイエンドなものまであり、サーバ群をクラスタ化して利用するHadoopのようなソフトウェアも普及してきました。これは常時フルパワーで利用されることも少ないと思われ、未使用の高性能サーバから省電力のサーバに仮想環境で多数の待機サーバを移動しておけば待機電力を削減できます。

仮想環境状況の変化とともにネットワークも変更できれば、マイグレーション時間も短縮可能です。このような場合は、データセンター内をSDNで構成するメリットが出てきます。このメリットをサポートするため、従来からの大型コアスイッチがバージョンアップして、SDN機能を持つ製品も増えてきました。

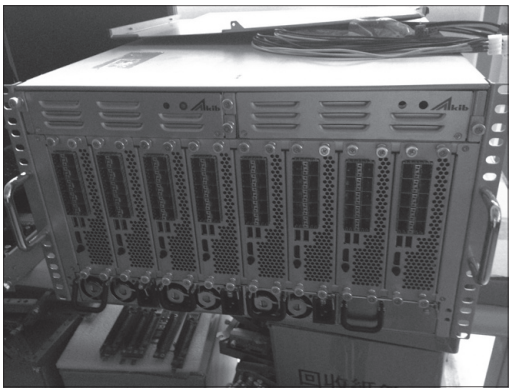
OpenFlowの問題点とは

NECなどが早くからOpenFlow対応製品を出荷し、Arisuta、BicSwitchなど専業メーカーから、Ciscoなど各コアルータメーカーまで対応してきました。台湾在住の日本人によるベンチャー会社であるAKIB SYSTEMSが開発した10GbE(SFP+)が80Portも装備され、トラフィックを高速処理できるBonet!などもOpenFlowに対応しています(**写真1**)。

しかし、マスターサーバに障害が出て、運用エンジニアは対応できるのだろうか? 運用障害の復旧を想像すると、考えるだけでとてもおそろしいです。その際は、各メーカーのサポートが大事です。RAIDはHDD障害によるデータロスを防ぐ技術ですが、実際には使うパーツが増えるので安定度が下がります。安定度を維持する製品設計は経年変化に耐える設計を行うため高度な技術が必要です。

同様なことが、OpenFlow機器にも存在します。内蔵するマザーボードからメモリの安定度などコントローラにネットワークを管理する情報が集中するので、大規模事故が発生するリスクは高くなり、クラスタ構成サーバでOpenFlowコントローラを稼働させたいものです。エッジスイッチと呼ばれるサーバ近くのスイッチが故障したときには、予備スイッチを通る経路にソフトウェアで対応できるようになりますが、確実な切り替えも必須です。安定度が高く

▼写真1　AKIB SYSTEMS Bonet!



▼写真2　Googleマップ「データセンター内部見学」
Explore a Google data center with Street View（http://www.youtube.com/watch?v=avP5d16wEp0）

なってゆくのはこれからと思われます。そのためOpenFlowは、サーバを大量に保有し技術要員のレベルが高く、HUB台数を大幅に減らすことができる可能性もあり、運用コストメリットの出るキャリアや大手クラウドサービス会社から導入が進むと考えられます。

L2化の問題を解決

GoogleのようにバックアップLANが世界中をめぐる（？）とか、データセンター内でHadoopのような大量なクラスタサーバ運用で大きなトラフィックが発生する場合、経路を変えて効率を上げるのは大きなメリットがあります。

L3スイッチHUBは、L2スイッチHUBに比べ高額で低速なので、L2スイッチHUBをOpenFlowで中央から制御できれば大量のトラフィックを効率よく運用できます。OpenFlowに対応した40GbE L2スイッチはL3スイッチより安価に高速回線を実現し、故障などのときOpenFlowで経路切り替えが可能です。

サーバ高性能化による小型化

仮想化ハイパーバイザサーバをグループ化し、OSをどのサーバでも自在にインストールできる時代です。ハードウェアが故障した場合には複数サーバでライブマイグレーションできるようにセットアップしておけば、冗長化サーバとなり、サービスを停止させないクラスタサービスになります。

Googleが公開しているデータセンター内部の写真では、1ラックにはぎっしりサーバが搭載されていますが、ラックは連結されずに間を離して設置されています。まさにラックが1台のサーバのように稼働しているようです。Googleマップで「708 Lynhaven Dr, Lenoir, NC, アメリカ合衆国」を検索すると、ストリートビューで内部を見学できます（**写真2**）。

SDN/OpenFlowは大規模データセンターで必須な技術

データセンターにおけるラック電源容量で、1ラック2KVAはなくなり、6KVA-8KVA/1ラックが主流になってきました。eBayとDELLは、40KVA/1ラックをコンテナモジュールを使い、1年以上砂漠で稼働させています。まさにスパコンをビックデータ処理に使っているように思います。サーバをたくさん使って高性能化し、ラック単位を1システムとする「ラックスケールアーキテクチャ」をインテルが提唱しています。

ラック内ネットワークやラック間の高速化低価格化を狙い、加えて故障対策の点でSDN/OpenFlowは最適に思えます。ラックを1台のPCと考え、多数のラックを接続する太くて速く冗長化できるネットワークを作ることができると思われます。SD

CHAPTER 7

OpenFlowスイッチ自作で見えた新たな利点 ワイヤレスとSDNの意外な関係

Writer 川井 浩陽(かわい ひろあき)／㈱ストラトスフィア
Twitter@kwi　Mail：kawai@stratosphere.co.jp

これまで、SDN/OpenFlowはデータセンターやエンタープライズ領域でこそ効果を発揮すると言われてきました。しかし、最近になってオフィスなどのLAN環境、とくに無線LANにおける適用も検討され始めています。本章では、無線LANでのOpenFlowの適用例を紹介します。

無線LANの課題をOpenFlowで解決

近年、無線LANを使うデバイスは急速に増えており、「BYOD(Bring Your Own Device)」という単語も出てくるご時世です。オフィスに設置してある無線アクセスポイントにつながりにくい、おまけに無線LANセグメント用のDHCPアドレスブロックも使い果たした、という状況はどこでも発生してきていることでしょう。筆者たちストラトスフィアも例外ではなく、OpenFlowを含むSDN周辺技術を使ってこれを解決できないかと考えました。具体的にはDHCPのアドレスブロックを動的に追加するといった方法が考えられます。

オフィス環境にOpenFlowのしくみを持ち込むうえで、ハードウェアの導入についてはずっと悩まされてきました。OpenFlowについて調べてみたら、OpenFlow対応の製品はエンタープライズ向けの高価なものばかりでなかなか試すことができない。そんな経験はないでしょうか？　ご存じのとおりOpenFlow自体は単なる仕様でしかなく、ある特定分野の用途に限られた技術ではありません。しかし、現時点ではオフィス環境にOpenFlowスイッチを新たに買いそろえて導入するというのは、価格の面でも難しいでしょう。

そんな状況の中、解決のきっかけになったは「自宅ラック勉強会 #2.5 秋葉原出張編」というイベントでした。㈱バッファローのAirStation WHR-G301N(市場価格数千円)のファームウェアを書き換えて、ソフトウェアベースのOpenFlowスイッチの入ったLinuxにして遊ぶという内容で、OpenFlowのプロジェクト内にあったpantou[注1]というプロジェクトの成果を使っています。pantouとは、OpenvSwitchの祖先であるOpenFlow 1.0リファレンス実装をLinux上で動作させるOpenWrtのパッケージです。

筆者たちは、より現代的なOpenFlow 1.3の実装が必要になったので、trema-edgeにあるC言語で書かれたOpenFlow 1.3スイッチをOpenWrt上で動作させて使い始めました[注2]。これにより、いわゆるOpenFlowスイッチはずっと安価に使えます。開発に便利なのはもちろん、価格的な面でも可能性を感じてもらえるため、デモンストレーションにも有効です。

もともとのpantouでは無線を使うことはとくに示されておらず、背面にあるLANポート4つをVLANで区切って有線のOpenFlowスイッチとして使っていました。ところが、AirStationにははじめから無線リンクが存在するので、これをOpenFlowに使おうと考えるのは自然な成り行きでした。

試していくうちに無線LANを使ったほうが

注1) http://archive.openflow.org/wk/index.php/Pantou_:_OpenFlow_1.0_for_OpenWRT
注2) https://github.com/iHiroakiKawai/trema-openwrt

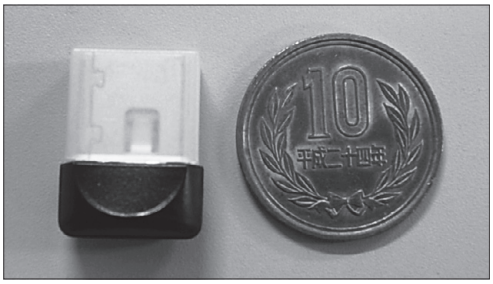
便利な点も見えてきました。有線のOpenFlowスイッチは一般的にポート数が多くなると製品は高価になります。無線LANではアクセスポイント1つに対して複数台接続できて、また接続状態は各個に管理されています。LANポートが4つしかないから4回線までしか制御できない、ということはありません。また無線であるがゆえに、有線でのEthernetケーブルの抜き差しのような動作をソフトウェアで制御することもできます。

OpenFlowによる無線LAN構築

ここからはOpenFlowスイッチを構成しつつ、弊社ストラトスフィアの製品「Omnisphere」のエッセンスを紹介したいと思います。実際にAirStation WHR-G301NにOpenWrtを入れるところから始めたほうが、ハッカー的な刺激があって楽しくお勧めなのですが、まずはお手軽にPCで試してみてはどうでしょうか。

Ubuntu 13.10 Serverを入れたサーバにUSB無線モジュールを挿すだけで同等の環境を作れます。AirStationよりも単価は高くなりますが、試験用途としては十分安価でしょう。たとえば

▼写真1　WLI-UC-GNM（バッファロー）



▼図1　ネットワークインタフェースを確認

```
# ip link
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: p5p1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP mode DEFAULT qlen 1000
    link/ether e8:40:f2:09:91:8c brd ff:ff:ff:ff:ff:ff
21: wlan0: <BROADCAST,MULTICAST> mtu 1500 qdisc mq state DOWN mode DEFAULT qlen 1000
    link/ether 80:1f:02:41:fc:1b brd ff:ff:ff:ff:ff:ff
```

筆者の開発環境では、**写真1**のようなUSBドングル型の無線LANアダプタを使っています。これを手元のUbuntu PCに刺すとwlan0という名前で認識されました（**図1**）。名前は環境によって異なる場合があります。

より詳細な無線の性能を調べるにはiwコマンドを使います（**図2**）。ポイントはAP modeがサポートされているかどうかです。無事AP modeがサポートされていれば、基地局のプログラム「hostapd」をインストールします（**図3**）。

Ubuntuではhostapd.confは最初は用意されていないので、自分で用意します。設定方法については、同梱されているhostapd.confにコメントで詳しく解説されています（**図4**）。最小の設定例は**リスト1**のようなものでしょうか。こ

▼図2　AP modeの確認

```
# iw list
Wiphy phy1
（略）
        Supported interface modes:
                 * IBSS
                 * managed
                 * AP
                 * AP/VLAN
                 * WDS
                 * monitor
                 * mesh point
        software interface modes (can always be added):
                 * AP/VLAN
                 * monitor
        valid interface combinations:
（略）
```

▼図3　hostapdをインストール

```
# apt-get install hostapd hostap-utils
```

▼図4　hostapd.confを参照する

```
# zcat /usr/share/doc/hostapd/examples/hostapd.conf.gz
```

の状態はOpen System認証という、暗号化が何もない状態です。運用時にはWPAなどを有効にしてください。

デバッグオプションを付けてhostapdを起動すると詳細なログが出力されます(**図5**)。接続が確認できたら/etc/default/hostapdのDAEMON_CONF を /etc/hostapd/hostapd.confにして保存します。

次はOpenFlowスイッチです。簡単にやるために、OpenvSwitchを使ってみましょう。インストールするだけで準備完了です(**図6**)。

無線LANとVXLANを結んでみましょう。VXLANはVLANと似ていて、L2レベルの仮想隔離ネットワークをIPレイヤの上に構築できます。有線インターフェースp5p1上にVXLANインターフェースを作りOpenFlowスイッチに接続するには、**図7**のようにします。openvswitchはコントローラが接続されていないときには、組込みの機能でスイッチングハブとして動作します。OpenFlowコントローラと接続するには**図8**のようにします。接続先は適宜読み替えてください。コントローラと接続したあとは、br0はコントローラの指示に従って動作するようになります。

wlan0の先につながっているホストをネットワークから切断するには、アソシエーションを解除します(**図9**)。hostapd_cliを使って動作中のhostapdに指示を出せます。

さらにhostapdを使ってマルチSSIDやダイナミックVLANなどさまざまな構成が取れますので、オリジナルの無線ネットワークをデザインしてみてはいかがでしょうか。SD

▼リスト1　hostapd.confの最小設定

```
/etc/hostapd/hostapd.confとして保存する
interface=wlan0
ssid=TestAP
hwmode=g
channel=1
ctrl_interface=/var/run/hostapd
```

▼図5　デバッグオプションでhostapdを起動

```
# hostapd -dd /etc/hostapd/hostapd.conf
```

▼図6　OpenvSwitchをインストール

```
# apt-get install openvswitch-switch
```

▼図7　VXLANを作りOpenFlowスイッチに接続

```
# ip link add type vxlan id 5 group 239.0.0.1 dev p5p1
# ovs-vsctl add-br br0
# ovs-vsctl add-port br0 wlan0
# ovs-vsctl add-port br0 vxlan0
# ovs-vsctl show
17c37cda-c518-41bd-a38d-8c004b4446c7
    Bridge "br0"
        Port "vxlan0"
            Interface "vxlan0"
        Port "br0"
            Interface "br0"
                type: internal
        Port "wlan0"
            Interface "wlan0"
    ovs_version: "1.10.2"
```

▼図8　OpenFlowコントローラと接続

```
# ovs-vsctl set-controller br0 tcp:127.0.0.1:6633
```

▼図9　アソシエーションを解除

```
# hostapd_cli disassociate <target_station_mac_address>
```

第2特集

エンジニアの伝わる図解術
～下手でも好印象で効果絶大～

本誌2012年12月号では「なぜエンジニアは文章が下手なのか」と題して文章を書くときに心がけておきたい内容について解説しました。今回は「図解術」にスポットを当てます。エンジニアの仕事の中には「書類を書く」ことが多くあります。文章にすると複雑な内容も、「図解」を取り入れるだけで理解しやすくなり、訴求力も上がります。

本特集、第1章では「図解術」を始めるためのヒントについて3つのステップで紹介します。第2章では、応用力が効く「よくある型」について紹介します。そして第3章では図解をするうえで陥りがちな側面と何のための図解かを意識する方法について説明します。

あなたも「図解術」をマスターして「伝わる文章」への第一歩を踏み出しませんか。

● 開米 瑞浩（かいまい みずひろ）

将棋と数学と小説に明け暮れる高校時代のあと、東大理一に入学するが半引き蘢りとなり3年で中退。IT技術者を始めてプログラミングに明け暮れる日々を過ごしつつ、技術情報をビジュアルにわかりやすく表現する方法を求めて個人的に研究、試行錯誤を重ねる。その経験をもとにしたIT専門誌の連載記事が大人気になったことから、独自開発した「読解力・図解力」を軸とする文書化能力向上プログラムによる企業研修業務を展開中。

第1章 図解術上達までの3ステップ

ITエンジニアの仕事の1つに仕様書や報告書などの「書類を書く」ことがあります。これは場合によってはプログラムを書くよりも労力がかかる場合も少なくありません。そんなときに役立つのが「図解術」です。複雑な情報ほど図解を取り入れることによってよく伝わるようになります。本章では、図解術上達に向けて「とにかく始める」ための3つのステップを紹介します。

はじめに

「ITエンジニア」というのは、情報系・勘定系・組み込み系・その他さまざまなバリエーションはあっても、何らかのコンピュータで動くシステムを作ることを仕事にしている人々です。そしてコンピュータはコード(＝プログラム)で動きます。したがって「ITエンジニアの仕事はコードを書くことである」……と言いたいところですが、残念ながら必ずしもそうとは言えません。「必ずしもそうとは言えない」どころか、コードを書くよりも書類を書く方に大半の労力を費やしているケースも少なくないようです。

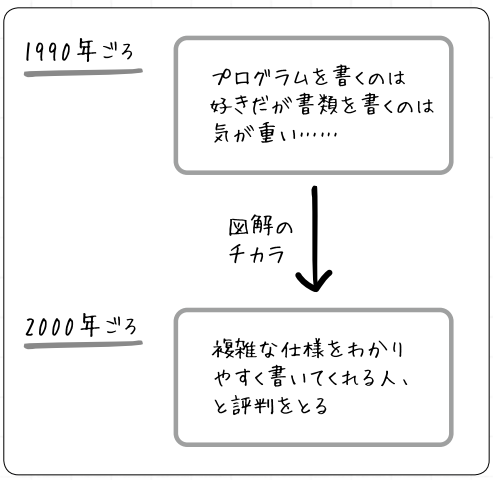
仕様書、報告書、提案書、リリースノートなどなど、現代のITエンジニアは「書類」と格闘しながら仕事をしています。ところがこの「書類を書く」ことを苦手にしているエンジニアが多いのもまた事実。できれば書類なんか書きたくない、という気持ちはありつつも、書かないと顧客や上司、ほかのメンバがわからないので、やむを得ず溜息をつきながら仕様書を書いている方も多いのではないでしょうか。

実はこの原稿を書いている筆者自身が、昔はそんなエンジニアの1人でした。プログラムを書くのは好きでしたが、種類を問わず、書類を書くこと、それを人に説明することには気が重くなりました。それが1990年ごろの話です。

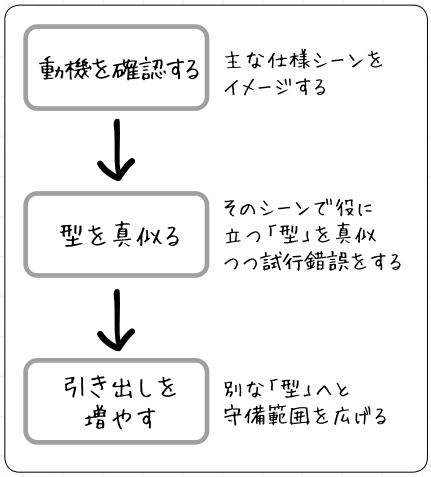
ところがそれから10年経った2000年ごろ、状況は一変していました。「めんどくさい複雑な話をわかりやすく説明したいなら、開米に頼め」と評価されるようになり、技術資料作成だけで仕事の依頼が来るようになっていました(**図1**)。

その変化の原動力になったのが「図解」です。複雑な情報は図に書くほうがよく伝わる、わかりやすく書ける、ということをつくづく感じていた筆者は、何かというと図解して説明するようになり、その場数を重ねるうちに上達し、いつの間にか人に通じる書類が書けるようになっていたのです。ちなみに「書ける」ようになったことで、「話す」ほうも楽になりました。10年

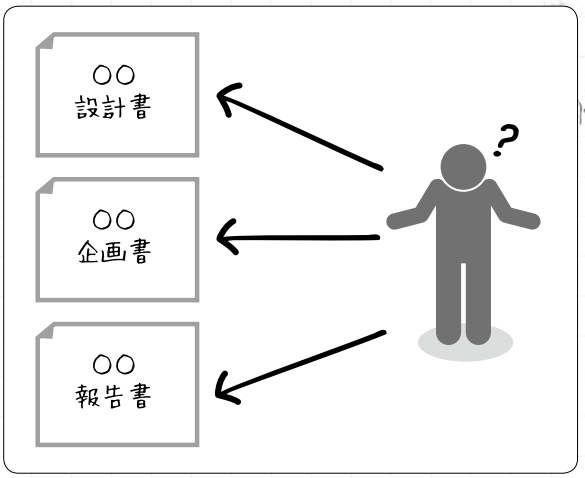
▼図1　図解のチカラがもたらした変化



▼図2　図解力向上を目指す3ステップ



▼図3　どんな種類の文書に困っているかをイメージする



前(‘90年以前)にはかなりのコミュニケーション下手で半分引きこもっていたような人間だったのですが、変われば変わるものですね。

そこで、今を去ること20数年前の筆者のように「書類書きは苦手」で悩んでいるITエンジニアのために本特集では「エンジニアの伝わる図解術」を紹介します。そのためにまず筆者が訴えたいのは、「とにかく、始めよう」ということです。言うまでもないことですが、やってみないことには上達しません。泳げるようになるにはプールに行く必要があるし、英語が話せるようになるには喋ってみる必要があります。当たり前ですね。

その場合、「とにかく始める」といっても大まかに3ステップで考えると良いでしょう。具体的には「動機を確認する」「型を真似る」「引き出しを増やす」の3つです(**図2**)。

それぞれいったいどういうことなのかを今から説明します。

ステップ1：動機を確認する

「ITエンジニアの図解術」というこの特集記事を、今まさに読んでいるあなたは「文書を書く」ことに何かと悩まれていることと思います。ただ、IT分野で業務上必要な文書を書くと言ってもいろいろあります。どんな技術を習得するにも、初心のうちはどうしても「うまい人のお手本を真似る」というところから入るものですが、たとえばコールセンターで使う顧客情報管理ソフトの画面仕様書と、組み込み機器の制御シーケンス設計書では、使用する用語も見た目の印象もまったく違うものを書くことになります。同じお手本が通用するはずはありません。

そこでまずは「自分はどんな種類の文書を書くときに困っているか」を思い返してみてください(**図3**)。

候補を3つぐらい挙げてその中から1つ「最初にこれをやってみよう」というのを選ぶと良いでしょう。選ぶ基準は何でもかまいません。「一番簡単そうだから」でも、「この種類の文書を一番よく使うから」でも、「これは顧客に提出するものだから恥を掻きたくない」でも、なんでもいいんです。大事なのは、「これをうまく書けるようになりたい」という動機がハッキリしていることです。動機がハッキリしている努力は継続できます。

銀の弾丸を求めず地道な継続を!

ここで1つ念を押しておきたいのは、「あっという間にうまく書ける方法など存在しない」ということです。「誰でも1日でプレゼンが圧

倒的にうまくなる3つの方法」のようなタイトルの書籍をよく見かけますが、どんな分野でも実際にはそんな都合のいい方法はありません。ソフトウェア工学の世界では昔から「(すべてをあっという間に解決してしまう)銀の弾丸など存在しない」という格言があります[注1]。「これさえチョイチョイっと覚えて使えば絶対うまくいく、そんな方法を教えてください」と言われても(実際それに近いことを筆者は数年前に図解力の研修で講師をしているときに受講生から要望されましたが……)、そんなものはないのです。

安定的に「うまくやれる」ようになるにはどうしてもある程度の熟練、努力の継続が必要です。だから、動機を確認しておきたいわけです。動機がハッキリしている努力は長続きします。それは別な見方をすると、「自分ができていないことを自覚する」ことですから、正直言ってある意味不愉快な現実ですが、その壁を超えた人だけが先のステージに進めるのです。

絞り込めば効率よく成果が挙がる

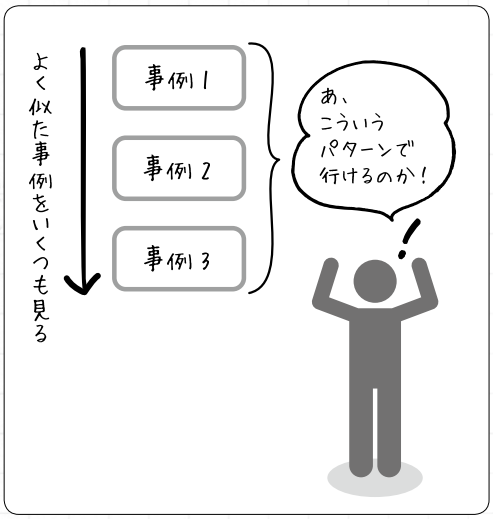
「努力の継続が重要」とは言っても、できるだけ効率よく成果を得たいのは当然ですね。そのために重要なのが、「絞り込む」ことです。

プログラマ向けの画面仕様書と、経営者向けの提案書を読み比べても勉強にはなりません。同じ分野の違う事例をいくつか見れば、良し悪しを比較できます(**図4**)。悪いところを避けて、良いと思ったところを取り入れればそれが自分の実力として身につきます。ですから、最初は1つの分野に絞ってやってみてください。そうすれば最も短時間で「うまく書ける」ようになります。もちろん、選んだその分野についてできるようになるだけですから、この段階では業務で必要な千差万別な文書のうちごく一部が書けるだけです。

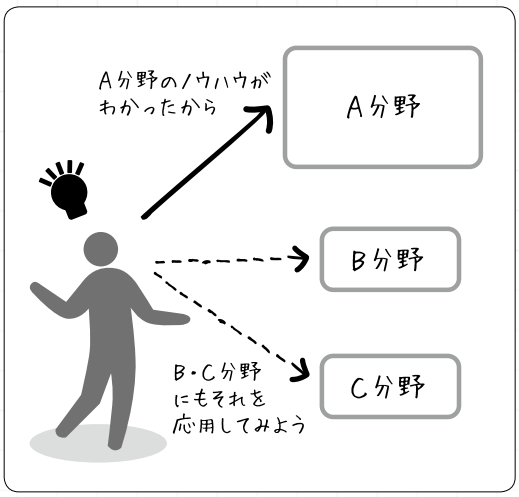
しかし、1つの分野で自分自身が考えて見つけた教訓、ノウハウは意外にほかの分野でも応用できることが多いものです(**図5**)。ですので、最初は絞り込んでください。類似事例をいくつも見て、共通する特徴を自分で見つけてください。それが、「短時間で成果を得て、それを広く拡大する」ための方法です。

ではここで1つ練習問題を出してみましょう。今まで出てきた5枚の図のうち、**図1**、**図2**、**図4**を見比べてください。何か共通点がありま

▼図4　類似事例を重ねることで「ノウハウ」がわかる



▼図5　1つの分野で得た教訓はほかの分野でも応用できる



注1）いわずと知れた、フレデリック・ブルックスの1986年論文『銀の弾などない ― ソフトウェアエンジニアリングの本質と偶有的事項』中の主張。

せんか？　実はとても単純なことですので、3枚を見比べて直観的に「これか？」と思ったことがたぶん正解です。

解答はもう少し後に書くことにして、ここでステップ2の話に移りましょう。

✳ステップ2：型を真似る

動機を確認したら、次のステップは「型を真似る」です。良い事例を真似して書こう、ということです。1人の発想にはどうしても限界があるので、他人の「うまいやり方」はどんどん取り入れていくべきなのです。初心のうちはとくにそれが重要です。

たとえばこんな例文を考えてみましょう。

❖ 例文1：GPS機能

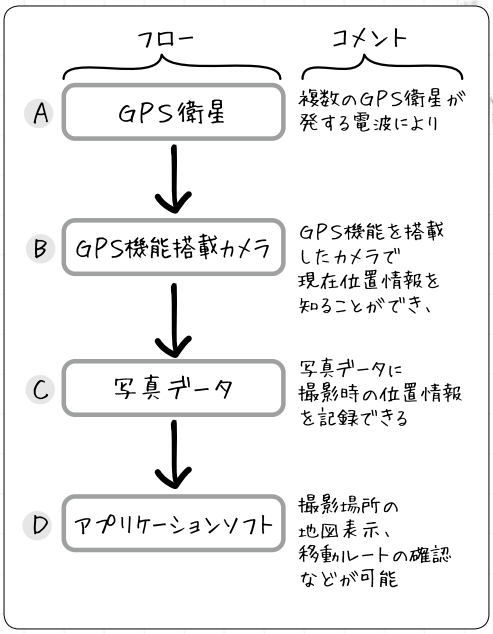
> GPSとは、複数のGPS衛星が発する電波により現在位置(場所)を知るしくみ。GPS機能を搭載したカメラで撮影した画像データには撮影時の位置情報を記録できる。付属するアプリケーションソフトを使えば撮影場所の地図表示や移動ルートを確認できる機能なども持つ。

短いので文章だけでもだいたいの内容はわかりますし、わざわざ図を書く必要もないのでは、と思われるかもしれませんが、筆者がエンジニア向けの図解研修で出題した感触からいうと、この程度の文章でも図解しようとすると苦労する方が意外に多いものです。普段やり慣れていないことは難しいんですね。

この例文「GPS機能」の図解案としてはたとえば**図6**のようなものがあります。

このようなパターンを筆者は「フロー＆コメント」型と呼んでいます。見てのとおり、左右が大まかに2列に分かれています。左側の、上から下へ真っ直ぐ順番につながる「フロー」の列に出てくるA～Dの各要素に「コメント」欄で解説を加えるパターンです。単純ですがこれが実に応用範囲が広いのでぜひ身近な文書で使えそうなものを見つけて試してみてください。

▼図6　GPS機能説明文の図解案



ITエンジニアは本来この型には馴染みがあるはずです。「フロー」という言葉はもちろんIT業界の伝統ある記法「フローチャート」から取ったものですし、プログラムにはたいてい「コメント」を書きますよね。構造化設計技法のデータフローダイアグラムやUMLのアクティビティチャートのように、システム開発をするにはフローを示すチャートは必要不可欠です。

ところが、**例文1**のようなものをいきなり見せられてもなかなかそこで「フロー」を読み取って書くことができないものです。そして解答例を見せられて「なんだ、これかよ！」とその単純さに悔しがる、そういう方を筆者は数多く見てきました。そういうものなんです。知っている手法であっても、人間は「今はその手法が使える場面だ」ということを知らないととっさに応用できないのです。だから、「これさえチョイチョイっと覚えて使えば絶対うまくいく、そんな方法を教えてください」という願いはけっして叶

えられることはありません。難しいのは、「方法」を覚えることではなく、「この場面で適切な方法は何なのか」を選択することだからです。

図7でいう「手法」の1つが「フロー＆コメント」型です。手法としては別に難しいものではないので、簡単に覚えられます。しかし、それを使うべき場面できちんと気がつくのは難しい。だから、「分野を絞り込んで集中的に類似事例を見ていく」ことが重要です。それをすることで、その手法が使える場面がわかるからです。

「まっすぐ順番につながる型」はどこにでもある

と、ここまで書いたところで前の出題の解答といきましょう。**図1**、**図2**、**図4**の共通点とは何か。それは、

まっすぐ順番につながる部分がある

ということです。「まっすぐ」「順番につながる」というこの2点、実は図解をするうえで東の横綱クラスに重要です。といっても、2003年に図解研修を始めたときも筆者自身はその重要性がわかっていませんでした。あまりに単純な話で自分ではハッキリ意識せずにやっていたぐらいですので、こんなものがノウハウとして必要だとは思わなかったのです。しかし現実に図解研修をやってみると、この「単純な話」を知らずに壁にぶつかって「うまく書けない」と悩んでしまう人が非常に多かったのが事実です。ぜひここは知っておいてください。

そしてもう1つ大事なのが、

同じ種類の情報をそろえる

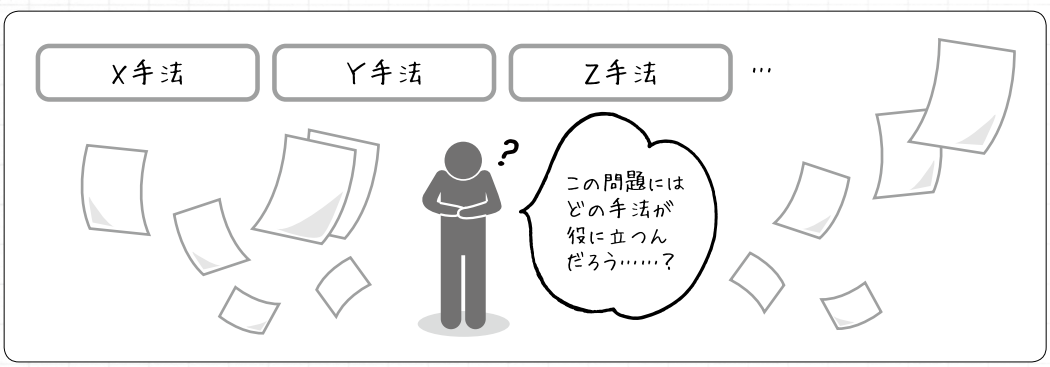
ということです。たとえば**図2**の「フロー」部分を見ると「動機を確認する」「型を真似る」「引き出しを増やす」と、動詞でそろえています。ここでもし最後が「引き出し」で終わっていたら、かなりの違和感がありますね。「同じ種類の情報をそろえること」もまた、複雑な情報をわかりやすく図解するためには、西の横綱クラスに重要です。これもあまりに単純過ぎて多くの人が軽視している、そんな種類のノウハウなのです。

なお、「フロー」といっても必ずしもIT業界でいうところの「処理のフロー」ではありません。**図2**は動詞でまとめているので「処理のフロー」の一種ですが、**図1**は「状態」の変化を表していて、**図6**は「GPS衛星」「GPS機能搭載カメラ」のようにある種の「装置」を表しています(実は「写真データ」や「アプリケーションソフト」も「装置」とみなすことができます。どちらもハードウェアではないので「装置」とは呼びにくいですが)。

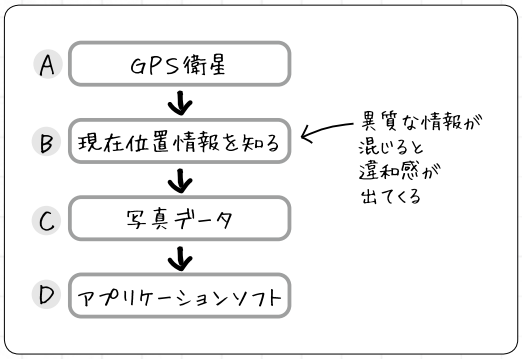
試しに「GPS機能」のフロー部分を**図8**のように書いてみましょう。

こんなふうに異質な情報が混じるとどうしても違和感が出てきます。とはいえ、「絶対にやってはいけない」というわけではありません。プログラムはコンピュータを動かすものですので

▼図7　「どの手法が役に立つのか?」の見極めが難しい



▼図8　同じ種類の情報で揃えることが重要



厳密に書かなければ動きませんが、図解は人間が読むものですので、要は人に通じるなら何をやってもいいのです。「不自然だけれど、あえてこうする！」と確信をもってするならそれは十分「あり」な選択です。しかしその確信がないまま、なんとなく書いていたら**図8**のように「異質情報混在になってしまった」というのはよくないので、避けてください。

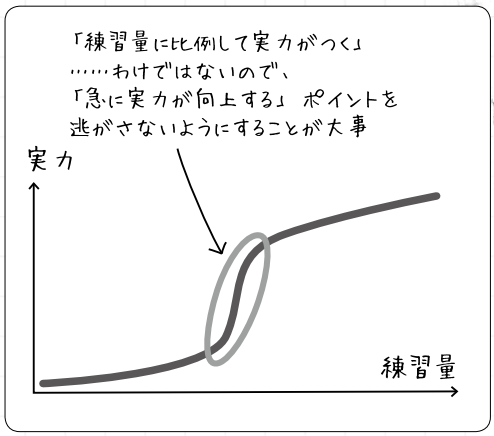
ステップ3：引き出しを増やす

突然ですが野球のピッチャーにもいろいろなタイプがいます。かつてメジャーリーグで活躍した野茂英雄や佐々木主浩は速球とフォーク、カーブというごく限られた球種で戦っていましたが、現役のダルビッシュ有や松坂大輔は多彩な変化球を駆使します。後者のようなタイプを「技の引き出しが多い選手」と言いますね。

そこで「エンジニアの伝わる図解術」ステップ3は「引き出しを増やす」ことです。

始めたばかりのステップ2の段階では「1つの分野に絞って類似事例を数多く見る／書く」ほうがいいのですが、それがある程度できるようになったら、別な分野へ広げてみてください。ただしその場合も、「絞り込む」のは忘れずに。1分野に絞って類似事例をいくつも見て／書いてみるほうが上達が早いです(**図9**)。だいたい、人が何らかの技術を習得するとき、「練習量に比例して実力がつく」ということはあまりありません。

▼図9　実力は練習量には比例しない



練習を始めても最初のうちはなかなか上達しませんが、ある程度積み重ねたところで急に「使える」ようになり、その後またなかなか伸びない「停滞」の時期が来るのが普通です。こうした「急上昇と停滞」を何度か繰り返して、人のスキルは伸びていきます。この件、興味のある方は「学習曲線とプラトー」というキーワードで調べてみてください。「プラトー」の本来の意味は「高原」ですが、スキルが伸びない「停滞」の時期が「高原」地形のように見えるのでこの名前で呼ばれています。

練習量と技術習得の間にはこのような関係があるので、「急上昇」の手前で止めてしまったらせっかくの練習も無駄になります。少なくとも最初の「急上昇」の時期までは集中的に練習量を確保してください。そのためには「絞り込み」が必要なのです。使える時間は有限です。その時間を2つ3つと分散させると、どれも中途半端で終わってしまいます。1つ壁を越えたらまた次の壁、と各個撃破作戦で行ってください。

必要なのは「情報」であって「文章」ではない

以上ここまで「動機を確認する」→「型を真似る」→「引き出しを増やす」という「図解術習得のための3ステップ」はおわかりいただけたでしょうか。

よしわかったやってみよう！と思った方のためにもう1つ言っておきたいのが、「文章は書かなくても良い」ということです。実は、書類を書くことに対して感じがちな苦手意識のかなりの割合は「文章を書く」ことについての苦手意識だったりします。ですが、我々は文学者ではなくエンジニアです。エンジニアの書く書類については、

必要なのは「情報」であって「文章」ではない

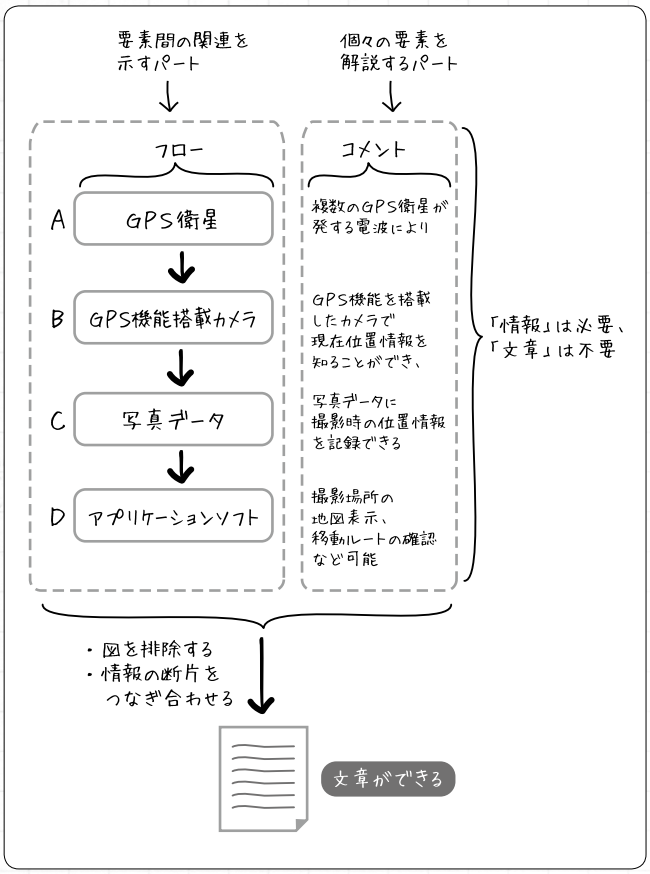
というケースが非常に多くあります。きれいにまとまった文章を書く必要はないこと、断片的な「情報」があれば十分役に立つことは知っておいてください。

実は、「文章」を書くのは図を書くよりも難しいのです（**図10**）。図解というのは「要素間の関連を示すパート」と「個々の要素を解説するパート」の大まかに2種類のパートでできています（フロー＆コメント型ではこれが視覚的にも2分されていますが、そうでないパターンもあります）。重要なのは、どちらのパートでもそこに書かれているのは「情報」であって「文章」ではないということです。

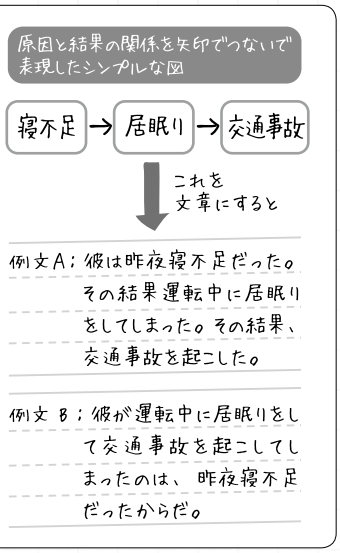
私達が国語教育の世界で教えられてきた「文章」というのは、図解から図を排除して、残った情報の断片に、「つなぎ合わせる表現」を加えてできるものなのです。図を排除するので、図によって表現されていた「要素間の関連」を言葉で表現しなければならないわけで、それを担うのが「つなぎ合わせる表現」です。このあたりの関係については**図11**をご覧ください。

この「寝不足」から「交通事故」までの図は、原因と結果の関係を矢印でつないで表現しただけの、ごくごくシンプルなものです。これを文章にした例をAとBの2種類挙げておきました。Aは因果関係を愚直にたどったストレートな文章、Bは結果から原因にさかのぼるトリッキーな文章ですね。

▼図10　「文章」を書くのは図を書くより難しい



▼図11　図の代わりに「つなぎ合わせる表現」を使う



この例から言えることの1つは、「文章のほうが書き方のバリエーションが多い」ということです。寝不足から交通事故までの因果関係を示すのに、図解なら**図11**の書き方以外はほとんど考えられませんが、文章だとA、Bのようにまったく違う2パターン(実際はもっと)があります。その分、文章を書こうとするとどうしても「迷い」が増えます。それが文章を書くことを難しく感じさせる要因の1つなのですが、そもそも「交通事故への因果関係を説明する」という目的に対してはその迷いは何の役にも立っていない、タダの無駄です。そんな無駄をわざわざかける必要はありません。

「良い文章」はかえって読者を誤解させやすい

ところで例文AとBではBのほうが「文章を書き慣れている」ように見えませんか？　実は「文章」を書く上でよく指導されることの1つに、「同じ表現の繰り返しを避ける」というものがあります。Aでは「その結果」という表現が2回重なっていますが、Bではそれがありません。「同じ表現の繰り返し」があると文章が幼稚に、書き手が馬鹿っぽく見えるので、それを避けるのが「良い文章」を書くための基本的テクニックの1つなんです。

ですが、ハッキリ言いましょう。そのテクニックは**「情報を正確に伝える」目的に対してはかえって有害**です。たとえば例文Bは確かに「文章を書き慣れている」ように見えますが、真の因果関係とは違う「居眠り」「交通事故」「寝不足」という順番でキーワードが登場しているため、かえって誤解を招きやすい表現なのです。

「情報」を「文章」で伝えようとした途端、この種の問題が起きやすくなります。必要なのは情報であって文章ではない、と筆者が言うのにはこんな理由もあります。無理に「文章」を書こうとせず、必要な「情報」を断片的でも良いので書き出すことから始めてください。

新聞や雑誌の記事をお手本にするのは大きな勘違い

文章の話がらみでもう1つ書いておくと、新聞や雑誌に載っている記事を「良い文章のお手本」と考えるのは大きな勘違いです。ITエンジニアがシステム開発の業務上、必要な関係者とコミュニケーションを取る、という目的に対しては、新聞や雑誌の記事はけっして良いサンプルにはなりません。

筆者が今書いているこの特集自体もそうですが、雑誌のような商業メディアの記事というのは「読者が興味を持って楽しく読んでくれる」ようにということを狙って書いています。純粋な情報提供系の記事なら別かもしれませんが、少なくとも筆者はそんな配慮をして「文章」を構成しています。

この配慮が実はクセモノです。システム開発のために書かれる文書の大半は「必要な情報が短時間で正確に伝わる」ことが最も重要なのに、「楽しく読める」配慮はそれに対して有害なのです。先ほど「良い文章」を書くテクニックのところで触れた同じ問題があるわけですね。

図10に書いたとおり、「文章」というのは図解から図を排除して情報の断片を「つなぎ合わせて」作ります。この「つなぎ合わせ」にいろいろなテクニックを駆使すると、「書き手の頭が良さそうに見える」そして「読み物としておもしろく楽しく読める」そんな文章ができあがります。しかしそこで必要な「テクニック」の大半はシステム開発の現場では不要などころか有害でさえあります。筆者が今書いているこの文章も、雑誌の特集記事だからこういう書き方をしているのであって、仕様書や企画書のお手本にはまったくなりません(論理構成は参考になるかもしれませんが)。

以上、「エンジニアの伝わる図解術」第1章はここまでです。「動機を確認する」「型を真似る」「引き出しを増やす」の3ステップで「伝わる図解術」の習得を目指すということ、ご理解いただけたでしょうか？ SD

第2章 IT関連文書の図解事例

第2章ではIT関連文書の図解事例で、「よくある型」がわかるものを何種類か紹介します。「よくある型」ですから、読者の職務上でも応用が効く可能性が高いものです。ぜひ実際に試してみてください。

✳フロー&コメント型

「フロー&コメント型」はすでに**第1章の図6**で出てきました。真っ直ぐ順番につながる「フロー」と、それを解説する「コメント」で構成されるのがフロー&コメント型ですが、必ずしも**第1章の図6**のようにフローとコメントが左右に2分されていなければならないわけではありません。この型の本質は、「フロー」部分がハッキリわかるように書くことです。文章でフローを書くと、たとえば「Cをするのは、Aの結果を受けてBを終えてからでなければならない」のように、前後が入れ違った表現をしてしまっている例もありますが、こういうものは箱と矢印で前後の関係を示すだけで誰にでも一目瞭然です。

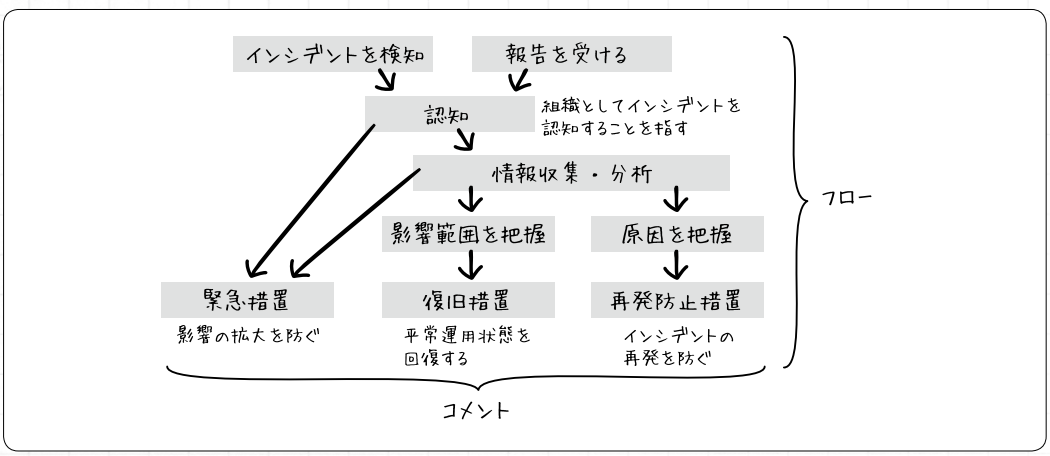
下記の例文も「フロー」を明示するだけで非常にわかりやすくなります。

❖例文2：インシデント対応活動

インシデント対応(活動)とは、インシデントを検知し、あるいはその報告を受けることにより認知し、影響の拡大を防ぐとともに、情報を収集して分析を加え、インシデントの全体像や原因について把握し、復旧措置や再発防止のための措置を取る一連の活動のこと。

(出典：JPCERTコーディネーションセンター「組織内CSIRTの必要性」)

▼図1　箱と矢印で「フロー」を明示する



この例文に対して「フロー」を図解すると**図1**のようになります。**第1章の図6**のGPS機能の例と違って「フロー」が1本ではなく枝分かれしています。「フローは絶対に真っ直ぐ1本につながらなければならない」といった制限は別にないので、枝分かれするなら枝分かれを、ループするならループを、そのまま書いてください。

また、この例では最下段の3要素と途中の「認知」にのみ「コメント」を付しています。コメントは必要なところに付ければよく、書く位置もとくにルールはありません。**第1章の図6**のようにフローが真っ直ぐ1本になるときは、すぐ横にコメントを書いていけば済みますが、この例のようにフローが1本にならないときはコメントを書く位置には工夫が必要です。要するに「どの要素へのコメントなのかが読み取れればそれでいい」ので、要素のすぐ横に書くなり、引出線を使うなり、ちょっとした工夫をしてみてください。

この際、多少見てくれが悪くなっても気にしないことです。エンジニアが書く文書は「センスよく美しくカッコイイ」ものである必要はありません。必要な情報がきちんと書かれていて、すばやく理解されることが何よりも大事ですので、ビジュアル的には下手でもまったくかまわないのです。美しく見せる必要があるなら、それはその仕事が得意な人にやってもらいましょう。

✳「書けるところから書いていく」ために付箋紙を活用しましょう

文書を書くことに苦手意識を感じていると、「書き始めるのが遅くなる」ことがあります。あるとき図解研修の受講生が、最初の一言を書き出すのもおっかなびっくりという感じだったので、少し聞いてみたところ、「間違ったことを書いてはいけない」というような意識が筆者の予想以上に強いようでした。

「えっ、間違ったら直せばいいだけですよ。心配いらないです」というのが筆者の本音なのですが、なかなかそう割り切れるものでもないのでしょうね。

そんな心配から自由になるために、ある程度有効なのが「付箋紙を使う」という方法です。フロー＆コメント型の「フロー」部分はほとんどキーワードだけですので、付箋紙にキーワードを次々書き出してから、「どういう順番につなげればいいか」は付箋紙の並べ替えで試行錯誤すればいいわけです。これなら最初から正しい答えが出なくても何の問題もありません。

もちろん付箋紙の代わりに、たとえばパソコンでPowerPointのような図形を書きやすいソフトウェアを使っても同じことができます。ただし、パソコンを使うのは「すでにある程度わかっている問題を考えるとき」に向いていて、「全然見当もつかないような問題を考えるときは付箋紙に手書きのほうがうまくいく」傾向があります。これは筆者の経験上の感想であって、絶対に誰もがそうだという気はありませんが1つの参考にしてください。

✳全体と部分の関係を表すツリー型

フロー＆コメントの次に、これも非常によく使われるのが、全体と部分の関係を表すために役に立つ「ツリー型」です。たとえばこんな例を見てみましょう。

❖例文3：パソコン用インターフェース

パソコンには周辺機器と接続するためのさまざまなインターフェースがある。パラレルインターフェースは複数の信号を並列に伝送する方式で、ハードディスク接続に使われていたIDEやSCSI、プリンタに使われていたセントロニクスI/Fが代表的。シリアルインターフェースは信号を直列的に伝送するものでRS-232Cが代表的。IrDAは赤外線を使ってデータ交換をするための規格、USBはキーボード、マウス、モデムなどを統一的に接続するための規格、Bluetoothは無線通信によりデータ交換をするための規格である。

この例文を元に、一部不足している情報を補ってツリー構造で図解すると**図2**のようになります。

図中の左側にある「幹」あるいは「根」と呼ばれる側に大分類を置き、右側（末端、枝葉側）へ進むにしたがって分解・分類していきます。

パソコン用インターフェースは大まかに有線／無線に大別され、有線はさらにパラレルとシリアルに、無線は赤外線と電波に分かれ、と階層的に細分化されていく様子を表現するにはこのツリー構造が最強です。ちなみに、技術的基礎知識の乏しい新入社員にこの種の図を書かせると、階層を1つ飛ばしたり、「USB」を「シリアル」と同列に書いたりといった間違いが多発するため、「何をわかっていないのかがすぐわかる」形式でもあります。

また、この種の図解パターンはロジカルシンキングの分野では「ロジックツリー」と言われて、非常によく使われています。「分類」を説明しようとするときには役に立つことが多いので、自分が馴染みのある技術分野を選んで試してみると良いでしょう。**図2**は「パソコン用インターフェース」という分野の例ですが、「プログラム言語」であったり、「通信プロトコル」であったり、分野を限定して掘り下げていくことで、自分の知識の棚卸しができる効果もあります。ぜひ使ってみてください。

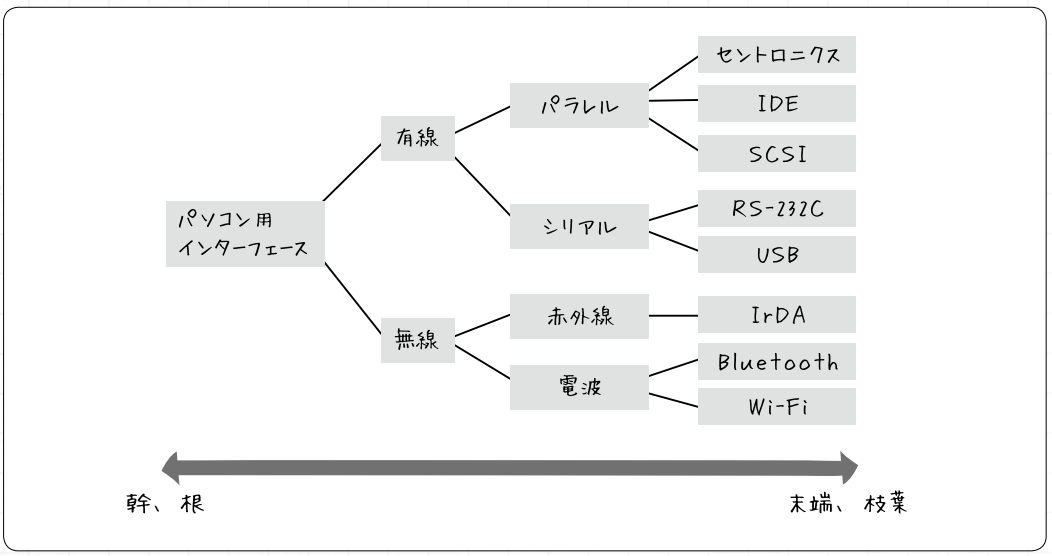
✳ツリー構造で「要約力」を上げよう

ちなみに、このツリー構造の図を末端から幹のほうに進むと「要するにこういうこと」という「要約」になります。これができると、非専門家、つまりはユーザ側や経営者とのコミュニケーションが非常に楽になります。たとえば次の例文を見てみましょう。

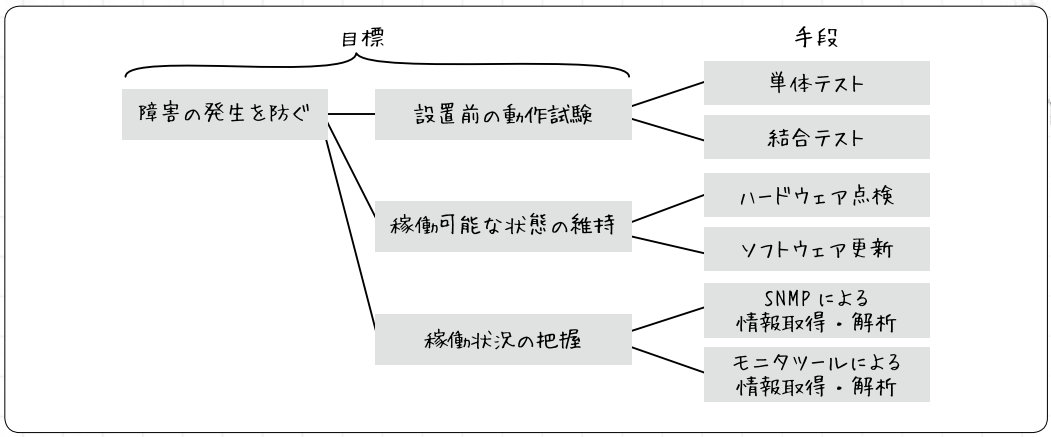
❖ 例文4：通信機器の障害防止策

通信機器で障害が起きることを防ぐためにまず必要なのが単体・結合での動作試験です。動作試験をパスして設置してもそれで終わりではなく、ハードウェアを定期的に点検する必要がありますし、ソフトウェアは更新する必要があります。さらに、SNMPを使って機器本体から、あるいは外部のモニタツールを使って情報を取得、解析して稼働状況を把握しなければなりません。

▼ 図2　ツリー構造図（パソコン用インターフェース）



▼図3　目標と手段、あるいは明細と要約の関係



この例文の情報をツリー構造で図解したものが**図3**です。「稼働可能な状態の維持」という文言は例文中にはありませんが、ツリー構造を作るために補ったものです。

こうしてみるとこのツリー構造は「目標○○を達成するために、手段××を取る」というパターンで解釈できることがわかりますね。このパターンで書き直した説明文を見ていただきましょう。

❖ 例文5：通信機器の障害防止策（改訂）

通信機器で障害が起きることを防ぐために必要なことはおおまかに言って、設置前の「動作試験」、設置後に「稼働可能な状態を維持すること」、そして「実際の稼働状況を把握すること」の3点です。
「設置前の動作試験」のために単体・結合でのテストを行います。
「稼働可能な状態を維持する」というのはハードウェアの定期的な点検とソフトウェアの更新のことです。
「稼働状況を把握する」ために、SNMPを使って機器本体から、あるいは外部のモニタツールを使って情報を取得、解析します。

さて、**図3**の「目標」の部分と「手段」の部分を比べてみると、通信機器に関する非専門家でも理解しやすいのはどちらでしょうか？

言うまでもなくそれは「目標」のほうですね。ということは非専門家と話をするときは「目標」のほうから話をしないと通じないのです。素人に対して「手段」という技術的細部の話ばかりしていると、「コミュ力低いエンジニアめ……」といった冷たい視線が飛んできます。

それを防ぐために有効なのが、「つまりこういうことなんです」と、要約をする習慣です。「障害の発生を防ぐ」という、より上位の大目標を達成するために役に立つ下位目標を分解し、それに関連付けて「手段」を語ることができれば、「技術もわかるし話もできる、こいつは頼りになるエンジニアだ！」という評価を得られます。素人に迎合する必要はありませんが、アシストはするべきです。「要約」はそのための非常に有益な方法であり、ツリー構造の図解はそれを習慣化するとても良いツールなのです。

✳マトリックス型

ここまで、フロー＆コメント型とツリー型を紹介してきましたが、いずれにしても図解「手法」としては極めて単純なもので、UMLやDFDあるいはER図のように記号を覚える必要もありません。記号を覚える必要がない、ということは、そのまま素人にも通じる、ということです。

そんな「記号を覚える必要がない」図解手法と

▼図4　マトリックスで2つの対象を比較する例

	ウィルスメール	スパムメール
定義	ウィルスを含むメール	迷惑な広告を含むメール
被害の性質	システム破壊・情報漏洩	性能低下・資源消耗・不愉快さ
被害の起きる条件	1通でも感染したとき	大量のスパムを受信したとき
防衛上の目標	1通のこらず食い止める	大量の到来を食い止める

して、フローやツリーと同格の広い応用範囲を持つものが「表形式」です。フローやツリーに合わせてカタカナでマトリックス型と見出しを付けておきましたが、要するにタダの表です。今度は次の例文を見てみましょう。

❖例文6：ウィルス、スパムメール対策

メールを経由してコンピュータウィルスが感染する例は非常に多い。ウィルス入りのメールが人間に届く前に食い止めることはセキュリティ上の重要な課題である。ウィルスはたった一通でも取り逃がすとシステム破壊や情報漏えいをもたらす危険な存在だからだ。

だが、システムへの脅威はウィルスだけではない。スパムメールも頭が痛い問題だ。

スパムメールとは、広告を含む迷惑メールのことを言う。コンピュータウィルスとは違って、1通のメールが直接的に被害をもたらすことは少ないが、一度に大量に発信されるため、インターネットのトラフィックやサーバの処理能力を圧迫する。また、メールの受信者がいちいちスパムと正規メールを振り分ける手間も馬鹿にならない。ウィルスと同様、スパムからも情報システムを防衛しなければならない。

この例では「ウィルス入りメール」と「スパムメール」の2種類を比較して語っています。人間は、「違うところ」を手がかりにすると問題を理解しやすいので、このように「AとBを比較して説明する」ケースは非常に多いですね。その場合は単純ですが表を作ってしまいましょう。なんだ、それだけかよ、と思われるかもしれませんが、その「たったそれだけ」をやっていない文書がITエンジニアの仕事の現場では実に多いのです（非エンジニアの職場でも同じですが）。単純な手法だからといって軽視しないでください。逆に、**単純な手法というのは広く応用できるので、意識的に使うことで大きな成果を得られる方法**でもあります。

実際に例文6の内容を元に図解すると**図4**のようになります。

この図については左端の列に注目してください。定義、被害の性質、被害の起きる条件、防衛上の目標、という4項目の見出しがついていますが、元の例文6にはこのような文言はありません。マトリックス自体は「タダの表」ですが、大事なのはこの**「比較項目を切り分けて、適切な見出しを付ける」**ことです。これはそう簡単にはできないので、一見すると「タダの表」であっても実際に作るのは簡単ではありません。

しかしながらこうした「比較項目」は、同じパターンが何度も出てきます。経験が次回に生き

るので、類似事例を数多くやってみてください。

✳前状態・処理・後状態型

では今度はこの例文を見てみましょう。

❖例文7：バージョンアップ手順書

製品Aを Ver.2以降にバージョンアップする場合、まずは製品AのVer.1のインストーラにてアンインストールしなければなりませんが、それだけでは、Ver.1に対するアップデート集約パッチはアンインストールされません。

これがアンインストールされていない状態で製品AのVer.2以降をインストールすると、集約パッチに含まれていたファイルがシステム上に残っているため、インストールに失敗します。

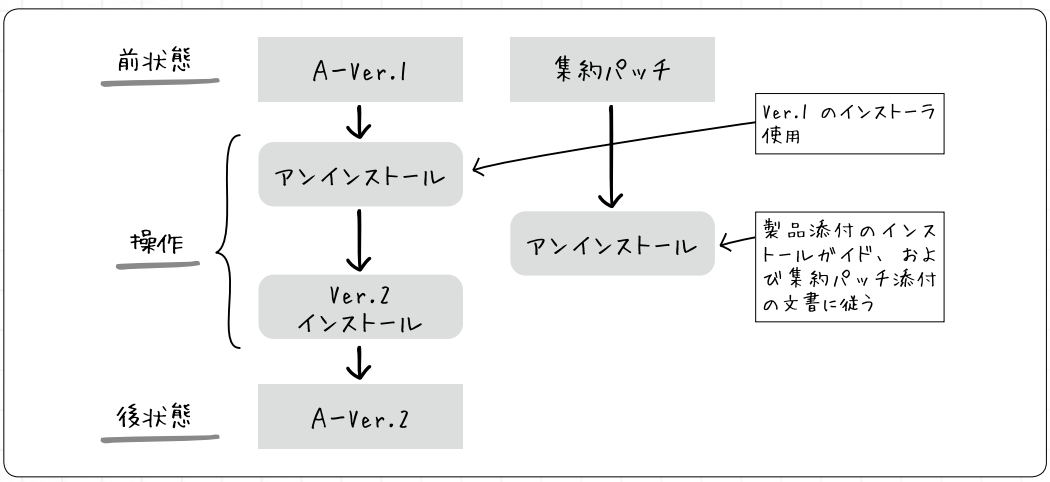
集約パッチをインストールしているシステムでは、製品A添付のインストールガイド、および集約パッチ添付の文書に従って、集約パッチの関連ファイルをアンインストールしてください。

IT製品を「バージョンアップ」するための手順書の一例です。バージョンアップはまっさらな状態からインストールするよりも手順が複雑になることが多く、実際、集約パッチの処理に注意してください、ということがこの例文では書かれています。この手順書は「○○をしてから××をして……」という「処理」の手順を書いているので、「フロー＆コメント型」をベースにちょっと工夫してこんな書き方をしてみましょう（**図5**）。

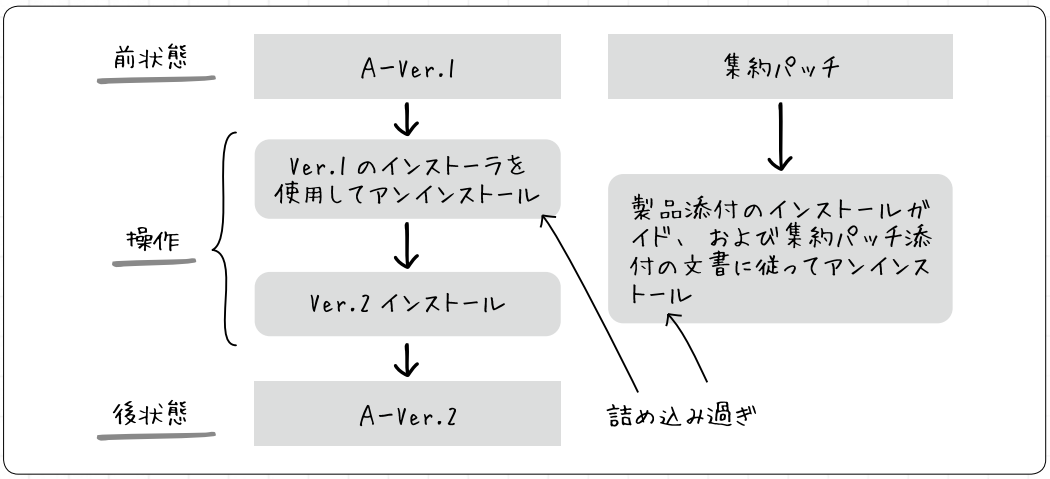
「フロー」というのは「操作」の手順なのですが、この図では操作の前後に「前状態」と「後状態」を追加しています。道案内に例えるなら「後状態」というのは目的地、「前状態」が現在地であり、「操作」が道順に該当します。現在地と目的地がハッキリしないのに道案内などできるわけがないので、それを明示したのが**図5**です。こうしてみると、前状態は「製品AのVer.1と集約パッチの双方がインストールされている状態」、後状態は「製品AのVer.2のみがインストールされている状態」だということがハッキリします。最初の例文ではそれが明記されておらず、「操作」部分だけが説明されていました。前後の状態を明示することで操作手順がわかりやすくなる例は非常に多いので、手順書を書くときに参考にしてください。実際、前・後状態を明示せずに「操作」だけを書いている手順書は非常に多いです。

なお、「Ver.1のインストーラ使用」とか「製品添付の……文書に従う」といった細かいところは

▼図5　前状態 ― 操作 ― 後状態モデル



▼図6　一度に多くの情報を詰め込むと混乱しやすい



「フロー」の中に入れずにコメントとして切り出し、引出線を引いてあります。仮にこれをフローの中に書いてしまうと**図6**のようになります。

予備知識のない状態でこのバージョンアップ手順書を読む人の「理解の順序」として理想的なのは、

【概要】まず製品Aのver.1と集約パッチをアンインストールして、それからAのVer.2をインストールするんだな。
【詳細】で、アンインストールの方法は……なるほど

と、概要→詳細の2段構えで理解してもらうことです。そのためには概要と詳細を分離したほうがいいので、それを想定して書いたのが**図5**です。**図6**のように書くと「概要」の理解に手間取ることになるため、お勧めできません。この種の文書を「書く」側はすでに知っていることだから**図6**のように書いても楽に読めるのですが、読む方は「一度に多くの情報を詰めこまれると混乱しやすい」のです。この点はあらゆる「説明書」に共通する法則として意識しておいてください。

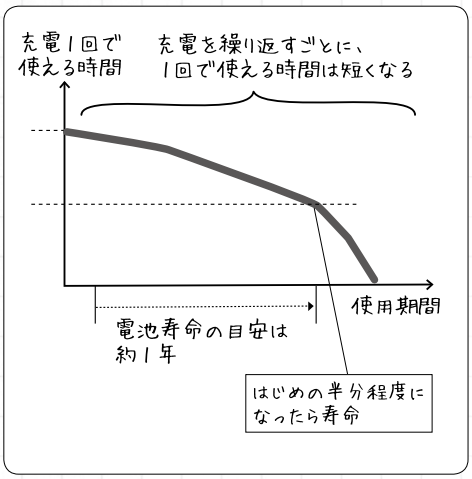
✳数値の性質にあったグラフを使う

今度はちょっと趣向を変えて「グラフ」を使ってみましょう。システムを適切に運用していくためには、さまざまな「数字」をもとにした意思決定を迫られる場面があり、そのための根拠となる資料を出せ、と経営者に求められた経験のある方も少なくないはずです。「現在の利用者増加ペースでいくと、半年後にサーバの容量が不足します」といったことを説明するにはやはりグラフがあったほうが話しやすいですね。しかし、ITエンジニアでグラフの使い方のトレーニングを受けたことのある方は少ないのではないでしょうか。そこで一例を紹介します。ある携帯電話の電池の持ち時間に関する説明文と思ってください。

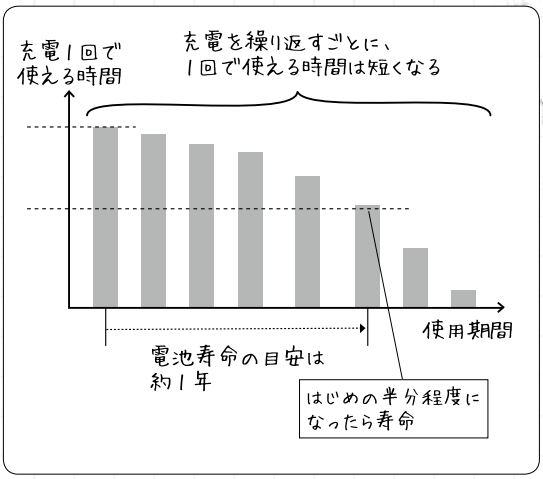
❖例文8：電池の持ち時間

1)電池パックは消耗品です。
2)どのような充電式電池も、充電を繰り返すごとに1回で使える時間は次第に短くなっていきます。
3)充電1回で使うことのできる時間が、使い始めたときに比べ半分程度になったころが、電池パックの寿命です。
4)電池パックの寿命の目安は、約1年です。
5)ただし、使用頻度が高ければ寿命はそれより短くなります。

▼図7　電池の持ち時間（折れ線グラフ版）



▼図8　電池の持ち時間（棒グラフ版）



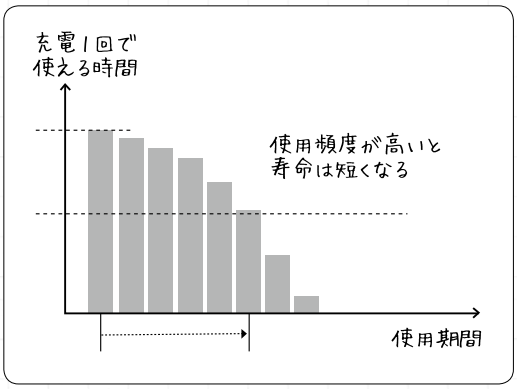
携帯電話の電池の話ですので誰にとっても身近な話題だと思いますが、文章だけだと少々イメージしにくいですね。そこで、この例文を理解するために役に立つグラフを書いてみましょう。

筆者がある図解研修の場でそんな出題をしたところ、2種類の解答が出てきました。折れ線グラフ版（**図7**）と、棒グラフ版（**図8**）です。違うのは折れ線か棒か、それだけです。さて、そのうえで質問が2つあります。

- 質問1：この2種類のグラフを見比べて、「携帯電話を長期間使ううちに電池が劣化していく」という情報を表すにはどちらのほうがより適切かを考えてみてください。
- 質問2：**図7、8**のどちらも、実は例文の1～5番のうちの1つを表現していません。どれを表現していないか、わかりますか？

まずは質問1の解答からいきましょう。折れ線か棒か、どっちでもいいんじゃないの？と思うかもしれませんが、実はちょっとした違いがあり、結論としては棒グラフのほうが適切です。理由は、「棒グラフであれば、棒のひとつひとつで、満充電したときの持ち時間をイメージできる」からです。1回充電して使い切り、また次の充電をして使い切り、という繰り返しを表すには棒グラフのほうがふさわしいわけです。

▼図9　電池の持ち時間（使用頻度が高い場合）



次に第2の質問の答えは、“5)ただし、使用頻度が高ければ寿命はそれより短くなります”を表現していない、です。そして、これを表現するのには棒グラフのほうがハッキリ便利です。「棒」の間隔を縮めることで「使用頻度が高い」を表現でき、それがそのまま「寿命が短くなる」ことまでの理解に直結するからです。折れ線グラフではこうはいきません。

要するに、棒グラフのほうが携帯電話の実際の使用シーンと性質が近い表現なのです。だからこそ、「使用頻度が高い」といった実際の状況を直観的に表現できます（**図9**）。ですので、できるだけ対象物（この場合は携帯電話の電池）が使われる状況に合ったグラフの種類を選んでください。単に数値がわかれば良い、というもの

ではないのです。

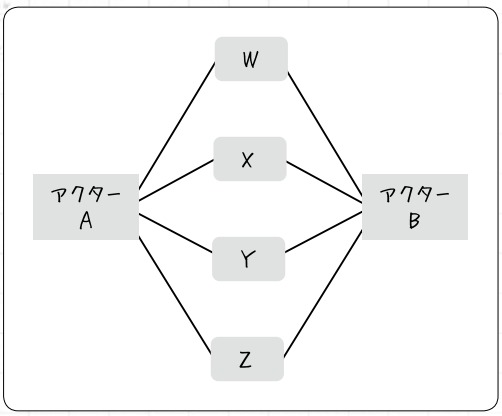
アクター間インタラクション・モデル

それでは第2章の最後に少し複雑な事例を紹介しましょう。下記例文は、個人情報保護法の簡略なガイドラインです。

❖ 例文9：個人情報保護法ガイドライン

A)個人情報を取り扱う場合は、あらかじめ利用目的をできる限り特定し、その利用目的の達成に必要な範囲内でのみ取り扱わなければならない。
B)個人情報は適正な方法で取得し、取得時に本人に対して利用目的の通知・公表などをする。
C)個人データについては、正確・最新の内容に保つように努め、安全管理措置を講じ、従業者・委託先を監督する。
D)あらかじめ本人の同意を得なければ、第三者に個人データを提供してはいけない。
E)保有個人データについては、利用目的などを本人の知り得る状態に置き、本人の求めに応じて開示・訂正・利用停止などを行う。
F)苦情の処理に努め、そのための体制を整備する。

▼図10　アクター間インタラクション・モデル



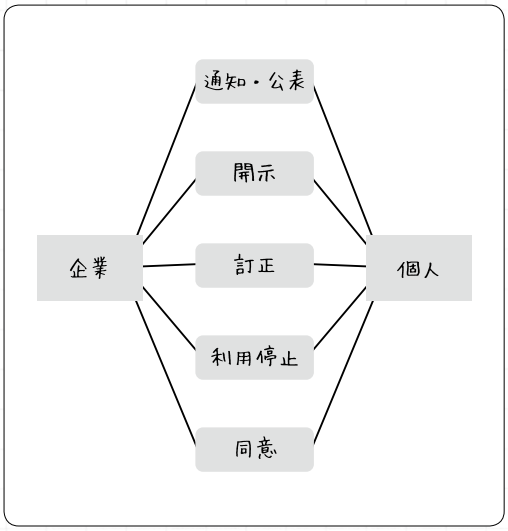
実際の個人情報保護法については各省庁が関連業界向けに特化したガイドラインを発行していて、はるかに情報量が多く複雑です。上記例文はバッサリと簡略化したものですが、それでもちょっとうんざりするような文章ですね。こういうものを文章のままで読んで理解しようとするのはやはりたいへんです。図解しましょう。

こういう場合、まずざっくりと書いてみます(図10)。まずはこのパターンを参考にしてください。

複数のアクター(人や組織など、動作の主体となる存在を表す。図中のA、B)がいて、アクター間でなんらかのインタラクション(W～Z)がある、というパターンです。こういうパターンでは「アクター」が見つけやすいのでまずそれを列挙し、さらにアクター間のインタラクションを列挙してみるとだんだん構図が見えてきます(図11)。

こうして大まかな枠組が見えてきたら、細部に手を付けます。たとえば「通知・公表」は「利用目的」に関わるアクションであり、「開示」は企業が持っている個人情報に関するアクションです。そのへんの区別をどんどん書き足していくと最終的にこんな図が書けます(図12)。

▼図11　個人情報保護法ガイドラインの主要なアクターとインタラクション

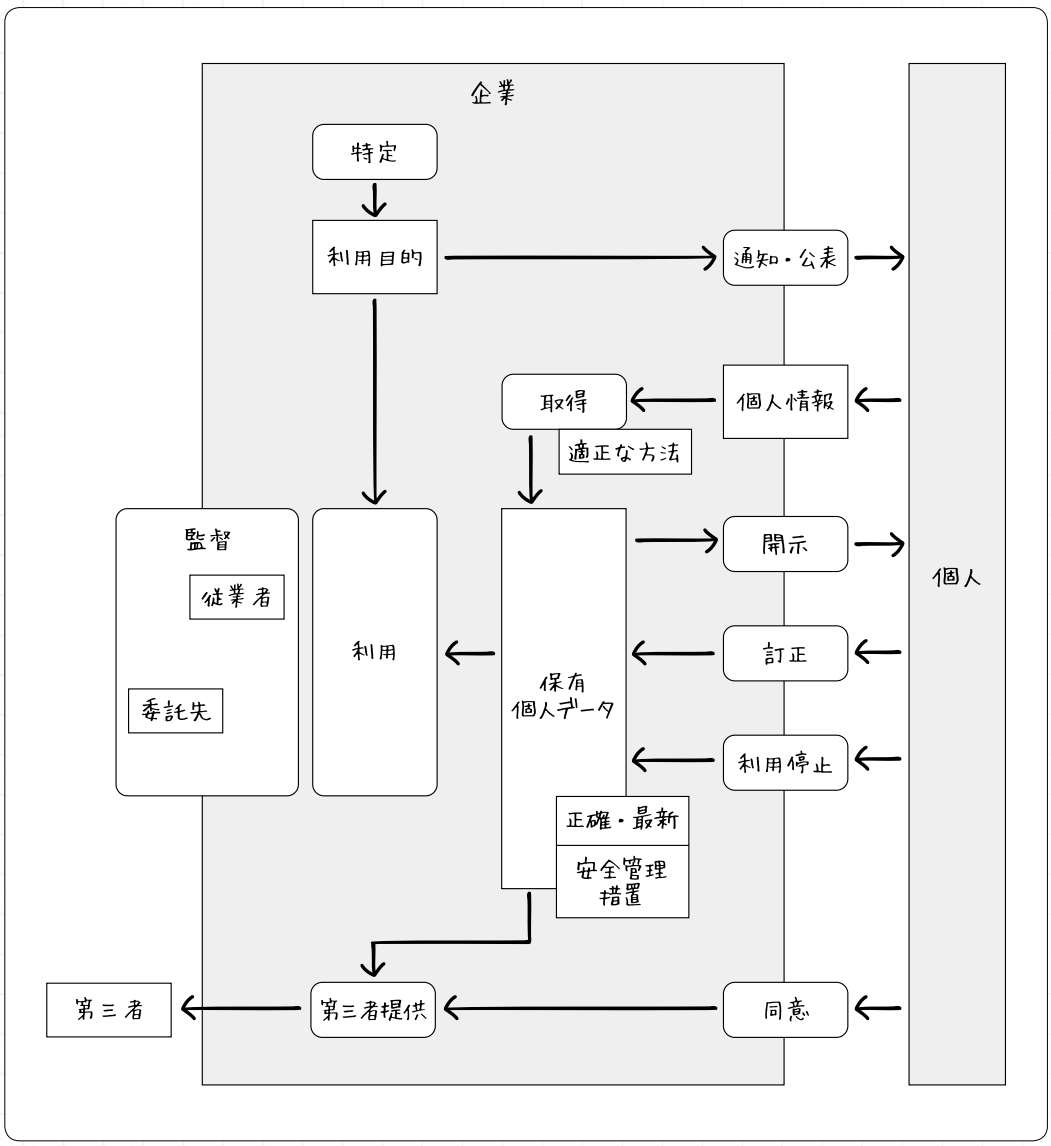


特定、取得、開示など隅の丸い箱はアクションを表し、利用目的、従業者、委託先など隅が直角の箱は企業や個人、データや状態を表します。

ずいぶんと複雑だなあ、と思いませんか？確かに図解したほうが理解しやすくなるにしても、自分でここまで書けるようになるだろうか……と不安に思うかもしれませんが、大丈夫です。本来、このぐらいの図解は3,000行のプログラムのメンテをするよりもずっとはるかに単純な作業でできるのです。今までできていないとしても、それは「やったことがないことはできない」というそれだけのこと。やり始めれば、いずれできるようになります。その際、学習効率を上げるために、「絞り込んで、類似事例を数多く見て試す」ことを心がけてください。

まずは、やってみましょう。すべてはそこから始まります。SD

▼図12　個人情報保護法ガイドライン：図解例



第3章　文書作成のワークフローを意識しよう

ここまで図解の手法や考え方を紹介してきました。しかし、図解すれば伝わる、図解はシンプルにすること、という考えをお持ちだとしたらちょっと待ってください。最後の章では図解をしていくうちに陥りがちな危うい側面と、脳内での情報処理のしくみを紹介します。これらを知ったうえで何のための図解なのかをしっかりと意識することで、あなたの図解はぐっと生きてきます。

その文書／図解の目的は何か

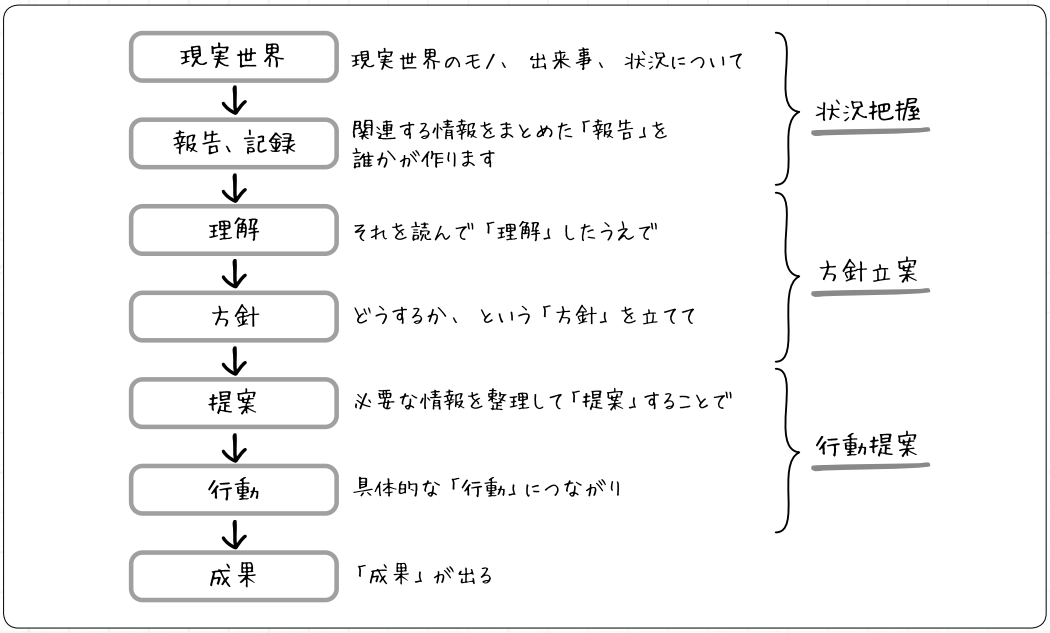
第3章はおそらく今までほとんどの方が聞いたことがないであろう「文書作成のワークフロー」について書きます。「図解」に限らず、さまざまな目的で文書を書くために共通して知っておいたほうが良いことです。

まずは**図1**からご覧ください。

ITエンジニアが書く文書にも、仕様書、報告書、提案書などさまざまなものがありますが、いずれにしても業務上使われるものである以上は何らかの「成果」を期待して書くはずです。そこで、「文書」が「成果」を上げるまでにどんな工程があるかを簡単にまとめたのがこの図です。

まず「現実世界」のモノ、出来事、状況について、関連する情報をまとめた「報告、記録」を誰かが作り、また別な誰かがそれを読んで「理解」したうえで、どうするかという「方針」を立て、必要な情報を整理して「提案」することで組織が

▼図1　文書作成のワークフロー



動き、具体的な「行動」につながって「成果」が出る、という流れです。このワークフローでは細分化しすぎだ、と感じるようなら上から順に「状況把握」「方針立案」「行動提案」のざっくり3フェーズと考えてもかまいません。

仕事を1人でやっている場合はとりたてて「文書」を書く必要はありませんが、組織を動かすためにはどうしても報告書や提案書を書かなければなりません。そして問題は、ワークフローのどの段階かによって、文書に求められる性質、要件が違うということです。

図解をすれば解決するのか？

「文書を書くのが苦手」というITエンジニアの苦手意識がどこから来るのかを考えたとき、「書くこと自体が苦手」というケースももちろんあるにせよ、「書いてもろくに読んでくれない」という面も大きいはずです。せっかくエンジニアの立場で問題点を発見し、それを報告書に書き、解決策を提案書に書いてもまともに読んでくれない、採用してくれない、という事態が続けばそりゃ、やる気もなくなることでしょう。

しかしそれは「文章だけで書いているからわかりにくくて読んでくれないのであって、図解すればわかりやすくなるから読んでくれるようになる」という種類の問題なのでしょうか？　必ずしもそうとは限らない、ことは認識しておく必要があります。

「理解」から「方針」を生むパターン

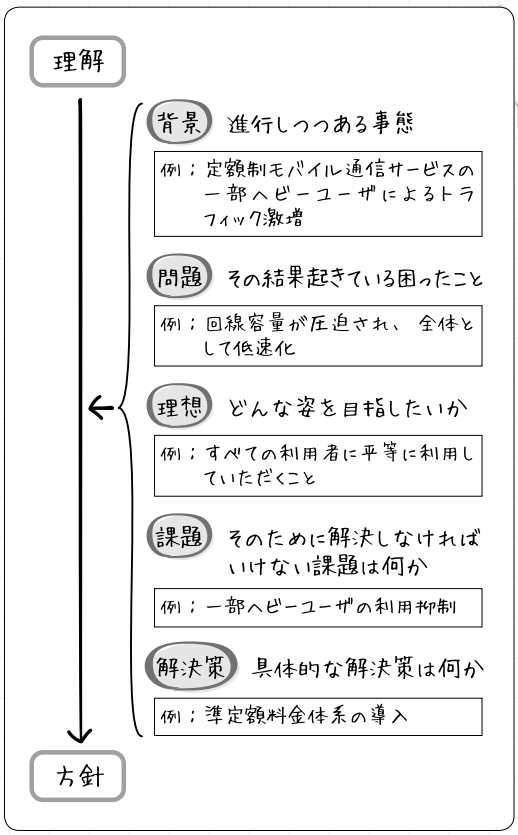
たとえば、Windows XPのサポート終了期限が来年に迫っている昨今、

経営者　「PCが使えなくなるだって？　どういうことだ？　こんな短期間に更新する予算なんかないぞ！！」

エンジニア　「2年も前からさんざん言ってたじゃないですか……」

というようなやりとりが発生している会社／団体も少なくないことでしょう。こういう事象は複合的な原因で起きているので、誰かを悪者にして責任を負わせることはできません。という念を押したうえで、ありうる「原因」の1つを指摘すると、「理解と方針の間をつなぐロジックができていなかった」ということが考えられます。

▼図2　理解と方針の間をつなぐロジックのパターン



「理解と方針の間をつなぐロジック」の典型的な例は**図2**に示す「背景→問題→理想→課題→解決策」というパターンです。

「背景」というのは通常、一定の方向に進行しつつある事態のことで、「問題」はその結果起きている困ったことです。図中の例のほかに、たとえば社会問題では「背景：社会の高齢化」「問題：健保・年金財政の悪化」などが一例です。この「背景」と「問題」は自分自身の意志とは関係なく現実世界で起きていることなので、「理解」する必要があります。

「問題」は解決しなければなりませんが、ではどの方向に解決するのか、というところで「理想」という「自分自身の意志」が問われます。理想がハッキリすると、その理想を実現するために乗り越えなければならない「課題」が明らかになり、課題が明らかになると具体的な「解決策」が浮かんできます。こうして「理想」「課題」「解決策」が決まるとそれが「方針」になります。

「方針」を立てなければ組織は動かない

ここで重要なのは、「方針」を立てなければ組織は動かない、ということです。「背景」と「問題」をいかに理解していても、問題だ問題だ問題だと騒ぐだけでは何も変わりません。理想、課題、解決策という「方針」を示せないままで「問題」の指摘だけをする人間は、だんだんと組織の中で疎んじられてしまうことがあります。もしそれが原因で「提案が通らない」「書いてもろくに読んでくれない」のであれば、いくら「文章」を「図解」にしてわかりやすくしてもその問題は改善されないでしょう。

つまりこの話は「図解」の問題ではないので、本来はこの特集の守備範囲ではないのですが、「人に通じる、受け入れられる文書を書きたい」という願いにとっては重要なことです。これに気づかず一生懸命に「図解」で解決しようとしてしまうことを防ぐためには、ぜひ意識しておいてください。「理解」から「方針」の間では「図解」はあまり役に立ちません。

では、役に立つのはいつなのでしょうか？

✳「simple is best」が常に正しいわけではない

図解が役に立つのは、「理解」と、「提案」そして「行動」の場面です。ただし、その3種類で役に立つ図解の特徴には少し差があります。どんな差があるかというと、「simple is best」という格言が通用するかどうかの差です。

「私達は複雑な問題に直面していた。解決するためにありとあらゆる複雑な技術を考えた。けれど最後にたどり着いた答えは極めてシンプルなものだった。とことん考え抜けば、シンプルな真実が見つかるのだろう」……といった主張を聞いたことはありませんか？　この主張は、筆者も9割ぐらいは正しいと思います。しかし、いついかなるときも正しいわけではないので、鵜呑みにしないようにしたいものです。

なぜこれを書くかというと、一般にビジネス書コーナーで発売されている「図解ノウハウ」本ではたいていの場合「simple is best」の原則が強調されているからです。だいたい、こんな原則を指導している場合が多いです。

――伝わる図解のコツは、シンプルに書くことである

――シンプルに書くためには、情報量を徹底的に削減しなければならない

実はこの原則が常に成り立つのは、「提案」段階で使う図についてだけです。「理解」と「行動」の段階で使う図については必ずしも正しくありません。

「理解」の段階では、自分自身がまだわかっていないことをわかるようにするために書くものなので、**何がわかっていないかを自覚する**のが図解をする主要な目的です。そのため、この段階では情報を詰めこんだ細かい図が必要になることがあります。わかっている情報を全部盛り込んで整理してはじめて「わかっていないことが浮かび上がってくる」という場合があるため、情報量を安易に削減することは勧められません。

一方、「提案」の段階で使う図は、**わかっていることのポイントだけを理解させる**のが主要な目的のため、たいていの場合、情報量を極力削減するほうが良い結果を産みます。つまりここでは「simple is best」が成り立つわけです。

最後の「行動」の場面では、たとえば「作業手順書」を作る場合があります。手順書のような書類では、「手順を漏れなくわかりやすく」示す必要があり、細かな手順だからといって省略できません。提案書なら「重要なポイントは3つです」と言って3つだけ語って後は省くという

手法が使えますが、手順書ではそれはできないわけです。

結局のところ、「simple is best」が常に成り立つのは「提案」の段階です。「理解」や「行動」の段階でも、シンプルにできるものはしたほうがいいですが、「提案」のときと違ってそれが不可能な場合がどうしても存在します。そんなときまで「simple is best」を絶対原則のように信じ込むのは有害なので、ここは気をつけておいてください。

ちなみに、なぜ一般の図解ノウハウ本のほとんどが「simple is best」を強調しているかというと、その種の本はたいていプレゼン資料などの「提案」の場面で使うことを想定して書いているからです。それは、**自分がよく考えてわかったポイントだけを、あまり考えていない人にわかった気にさせるための図**であり、その前提のもとでは「simple is best」は正しいです。が、自分自身が現実世界をよく理解していない段階で、とことん考えて理解しようとしているときにまでsimple is bestが通用するとは思わないようにしてください。

現実をありのままに表現した図が必要なときもある

別な観点でいうと、simple is bestが通用するのは「主張」がある図の場合です。「半年後にファイルサーバが溢れます」とか「○○処理がボトルネックになっています」といった「一言で表現できる、主張したいポイント」がある場合は、シンプルな図を書けるし、書くべきです。しかし、「図解」というのはそういう用途のものだけではありません。「主張」のない図が必要な場合も世の中にはあります。

たとえば典型的なものは「地図」です。地図というのは、道路、鉄道、建物、地形といった現実世界の状況がありのままにわかるように表現することに意味があり、「主張」を持ちません。「主張する」というのは、1点を強調してそれ以外を省く作業であり、地図でそれをやったら地図の意味がなくなってしまうのはおわかりでしょう。

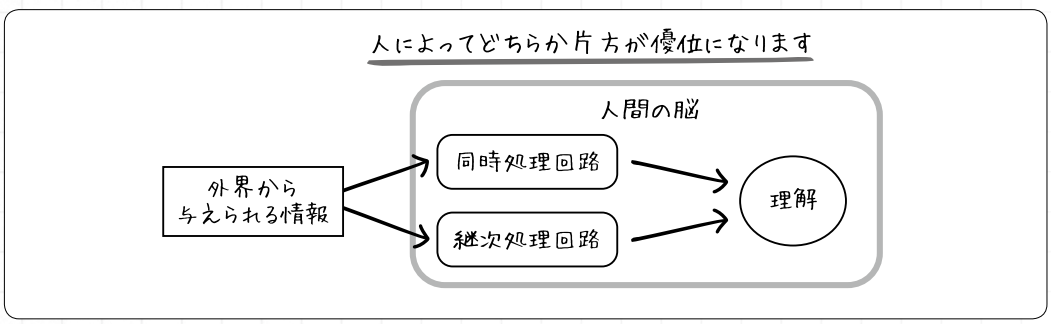
「地図」のように、現実をありのままに表現した図が必要なときも実際にはあります。一般に「設計図」と分類される図はその種のものが多いですね。IT業界ではクラス図やER図などがその例です。

✳同時処理型・継次処理型の違いを知っておく

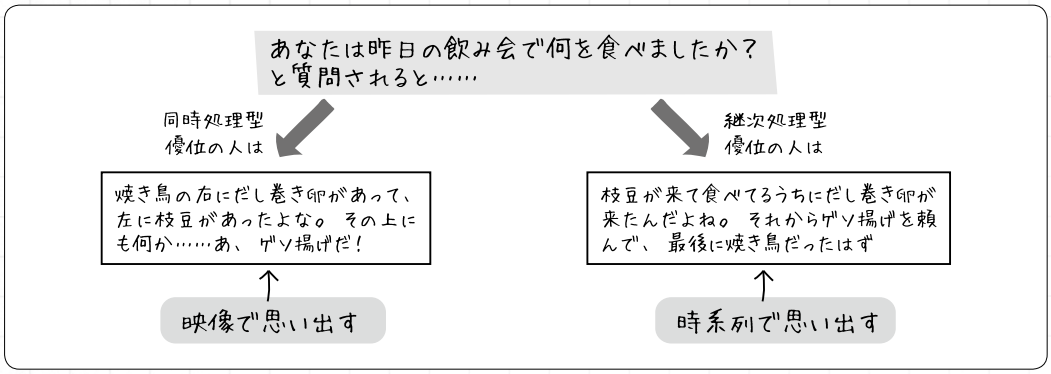
ところで、「図解」が必要だという理由の一番大きなものは「図解したほうがわかりやすく書ける」ということではないでしょうか。そこで最後に「わかりやすく書くために知っておいてほしい、人間の認知情報処理スタイルの個人差の話」をしたいと思います。

人間は外界から与えられる情報を理解するために、脳内で情報処理を行います。この「情報処理」を電子回路のようなものに例えると、人間の脳内には性格の違う2種類の情報処理回路があるということが知られています。それが「同時処理型」と「継次処理型」です(**図3**)。

▼図3　同時処理と継次処理



▼図4　同時処理型は映像で、継次処理型は時系列で記憶する



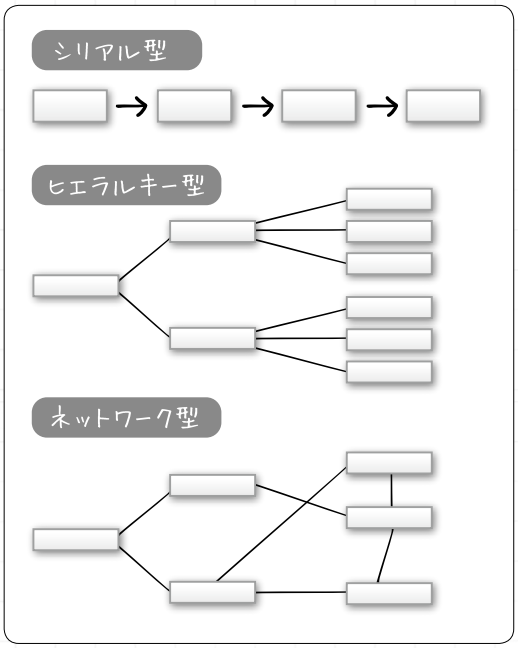
同時処理型というのは「情報の全体像を把握してその中に細部を位置づけるのが得意」なタイプの回路であり、継次処理型というのは「情報の細部を順番に着実にたどっていくことが得意」なタイプの回路です。どんな人の脳内にもこの2種類の回路が存在しますが、どちらがより優位かについては個人差があります。たとえば「あなたは昨日の飲み会で何を食べましたか」と質問されたときに、同時処理型の人はテーブルの上の映像を思い出し、継次処理型の人は食べた時系列で思い出す、といった違いが出ます(**図4**)。

継次処理型優位の人は「順番がハッキリしているものを順番どおりたどっていくことに強みを発揮」するので、たとえば電話口で道順を案内されてもあまり苦にせず、そのまま覚えることができますが、同時処理型優位の人はそれを苦手とします。逆に、同時処理型の人は地図を見て自分でルートを探して行くのが得意で、継次処理型の人はそれが苦手、といった差が出ます。

結局、人は自分が優位とするスタイルの情報のほうを「わかりやすく」感じるので、ある人にとってわかりやすい図だからといって別な人にもわかりやすいとは限りません。

ただ、1つ言えるのは「文章」というのは継次処理型の情報処理向けの表現しかできない形式だということです。文章や音声のように「順番に読んで／聞いて」いくことを前提とした形式は、同時処理型優位の人にとってはそもそも苦手なものです。

▼図5　情報の関連態様

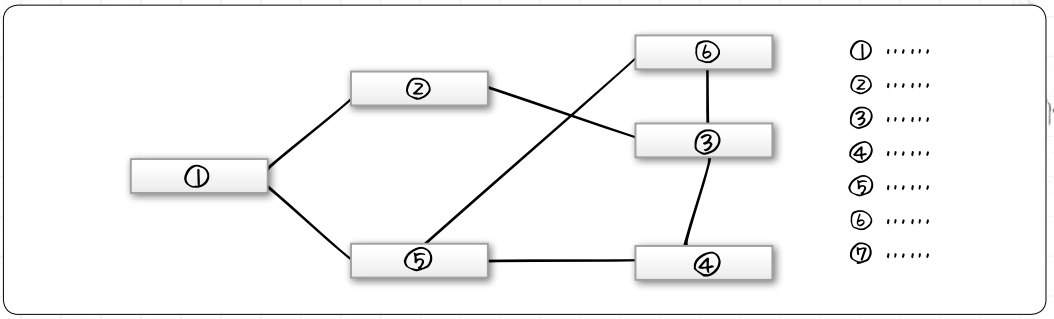


ネットワーク型の構造は図解でなければ表現できない

ここで、情報の関連態様には3種類ある、という話をしておきましょう。

複数の情報の間に何らかの関連性があるとき、その関連性のパターンを図に書くとシリアル型、ヒエラルキー型、ネットワーク型の3種類に分類できます。シリアル型というのは真っ直ぐ1本につながるもの、ヒエラルキー型は階層的に細分化していくツリー構造、ネットワーク型はそれ以

▼図6　ナンバリング&コメントでネットワーク型図解を補足するa



外のすべて、要は「なんでもあり」です(**図5**)。

「文章」は「シリアル型」を表現できる形式ですが、それ以外には本来向いていません。しかし、ヒエラルキー型までは文章でも工夫すればなんとか対応できます。「ポイントはおおまかに言うとAとBです。Aさらにを詳しくいうと3つあって……」という構造を守れば、文章でもヒエラルキー型をわかりやすく書くことはある程度可能です。

しかし、ネットワーク型になると文章ではお手上げです。この種の構造を表現するのは図解でなければ不可能だというのはご理解いただけることでしょう。地図やER図はこのタイプになります。

そこで、ネットワーク型の情報については図解を使う、のは良いのですが、ここで困るのは「継次処理型の人はこの種の構造の理解を苦手としている」ということです。ネットワーク型の構造にはハッキリした「順番」がありません。ヒエラルキー型ならまだなんとか1.1.1、1.1.2……のように階層化しての順序づけができますが、ネットワーク型だとそれも不可能です。同時処理型の情報処理が優位な人は苦にしない部分ですが、継次処理型の人にもネットワーク型の構造が「わかりやすく」伝わるようにするためには、何らかの順番をつけてやってください。

そのために筆者がよく使うのは**図6**のような方法です。要するに番号をつけてコメントを書き、どの順番に読めばよいかを明示するわけです。

ちょっとした工夫ではありますが、それだけで「図解」することでかえって継次処理型優位の人が感じやすいストレスを軽減することができます。ぜひ参考にしてください。

なお、もう1つ補足しておくと、**第1章の図6**で例示したような「フロー&コメント」型は、同時処理型・継次処理型のどちらの人にとっても楽に読める方法です。同時処理型の人は最初に全体像を見て取ることができ、継次処理型の人もどの順番に読むかを迷わなくて済むわけです。その意味でも「フロー&コメント型」は使いやすい手法でお勧めなのです。

✳バラバラにして真っ直ぐ並べることから始めよう

さて、長きにわたって書き連ねてきました本特集もいよいよ終わりです。最後に筆者があらためてアピールしたいのは、

バラバラにして真っ直ぐ並べることから始めよう

というこの一言です。文章を読んだら、その中に含まれている情報をバラバラにして「真っ直ぐ並べて」みてください。しっかり理由のある形で「真っ直ぐ並べる」ことができれば、そのまま「フロー&コメント」型で書ける可能性が高いです。そうならなくても、図解のヒントが得られるケースがよくあります。

「バラバラにして真っ直ぐ並べる」、これをやってみるだけでも「わかりにくい文章をわかりやすく図解する力」の向上に役立ちます。1日に数分でもいいので、ぜひやってみてください。SD

特別企画

基礎の基礎だけど案外知らない?

LinuxとFreeBSDのファイルシステムの良い・悪いところをご存じですか?

しくみを知って使いこなそう!

Writer 後藤 大地(ごとう だいち)
(有)オングス 代表取締役

UNIXをビジネスで使い日々運用するときに、システムを健康に保つ土台となるのがファイルジャーナリングシステム(以降ファイルシステム)です。本稿では、基礎の基礎だけど案外知らないファイルシステムのしくみを学びます。そしてストレージが磁気ディスクからSSDになる昨今、筆者自ら行ったIO Zoneによるテスト結果を通し、ファイルシステムのあり方を考察します。

OSに欠かせないファイルシステム

オペレーティングシステム(OS)を構成する欠かせない要素の1つが「ファイルシステム」です。ユーザやソフトウェアはファイルシステムを通じてデータにアクセスし、データを保存します。どのようにデータを整理するか、その考え方や実装方法は多種多様で、世の中にはさまざまなファイルシステムが存在しています(**図1〜4**)。同じ名前のファイルシステムでも、実際にはカーネルのバージョンが変わればファイルシステムを扱うコードも変わっています。ファイルシステムやそれに関するサブシステムについてよく知ることは、OSを使いこなすうえで1つの肝となります。

Linuxディストリビューションがデフォルトのファイルシステムとして採用することが多いのはext4です。これまでUNIX系のOSで広く採用されていたUFSというファイルシステムがありますが、Linuxが初期のバージョンで採用していたext2がUFSの設計思想に影響を受け

▼図1 Ubuntu 13.04のデフォルトファイルシステム[ext4]

```
Filesystem     1K-blocks    Used Available Use% Mounted on
/dev/sda1       26704124 3269568  22071408  13% /
none                   4       0         4   0% /sys/fs/cgroup
udev             2015196       4   2015192   1% /dev
tmpfs             404944     744    404200   1% /run
none                5120       0      5120   0% /run/lock
none             2024704      76   2024628   1% /run/shm
none              102400      28    102372   1% /run/user
# mount
/dev/sda1 on / type ext4 (rw,errors=remount-ro)
proc on /proc type proc (rw,noexec,nosuid,nodev)
sysfs on /sys type sysfs (rw,noexec,nosuid,nodev)
none on /sys/fs/cgroup type tmpfs (rw)
none on /sys/fs/fuse/connections type fusectl (rw)
none on /sys/kernel/debug type debugfs (rw)
none on /sys/kernel/security type securityfs (rw)
udev on /dev type devtmpfs (rw,mode=0755)
devpts on /dev/pts type devpts (rw,noexec,nosuid,gid=5,mode=0620)
tmpfs on /run type tmpfs (rw,noexec,nosuid,size=10%,mode=0755)
none on /run/lock type tmpfs (rw,noexec,nosuid,nodev,size=5242880)
none on /run/shm type tmpfs (rw,nosuid,nodev)
none on /run/user type tmpfs (rw,noexec,nosuid,nodev,size=104857600,mode=0755)
gvfsd-fuse on /run/user/daichi/gvfs type fuse.gvfsd-fuse (rw,nosuid,nodev,user=daichi)
#
```

▼図2 Btrfsで構築したFedora 19

```
[daichi@localhost ~]$ df
ファイルシス    1K-blocks     使用    使用可 使用% マウント位置
/dev/sda3       26864640 3604156 21533412   15% /
devtmpfs         2018660       0  2018660    0% /dev
tmpfs            2025020      88  2024932    1% /dev/shm
tmpfs            2025020     728  2024292    1% /run
tmpfs            2025020       0  2025020    0% /sys/fs/cgroup
tmpfs            2025020     372  2024648    1% /tmp
/dev/sda3       26864640 3604156 21533412   15% /home
/dev/sda1         487652   82756   379296   18% /boot
[daichi@localhost ~]$ mount | grep sda3
/dev/sda3 on / type btrfs (rw,relatime,seclabel,space_cache)
/dev/sda3 on /home type btrfs (rw,relatime,seclabel,space_cache)
[daichi@localhost ~]$
```

▼図3 UFS2で動作するFreeBSD 9.2-RELEASE

```
root@frccbsd92:~ # uname -a
FreeBSD freebsd92 9.2-RELEASE FreeBSD 9.2-RELEASE #0 r255898: Thu Sep 26 22:50:3
4 UTC 2013     root@bake.isc.freebsd.org:/usr/obj/usr/src/sys/GENERIC  amd64
root@FreeBSD92:~ # df
Filesystem  1K-blocks    Used    Avail Capacity  Mounted on
/dev/ada0p2  28434076 3504648 22654704    13%    /
devfs              1       1        0   100%    /dev
root@FreeBSD92:~ # mount
/dev/ada0p2 on / (ufs, local, soft-updates)
devfs on /dev (devfs, local, multilabel)
root@FreeBSD92:~ #
```

▼図4 ZFSで構築されるPC-BSD 9.2-RELEASE

```
[daichi@pcbsd] ~% gpart show
=>       63  62914497  ada0  MBR  (30G)
         63         63        - free -  (31k)
        126  62914383     1  freebsd  [active]  (30G)
   62914509         51        - free -  (25k)

=>        0  62914383  ada0s1  BSD  (30G)
          0  58699776       1  freebsd-zfs  (28G)
   58699776   4194304       2  freebsd-swap  (2.0G)
   62894080     20303          - free -  (9.9M)

[daichi@pcbsd] ~% mount
tank/ROOT/default on / (zfs, local, nfsv4acls)
devfs on /dev (devfs, local, multilabel)
procfs on /proc (procfs, local)
linprocfs on /compat/linux/proc (linprocfs, local)
tank/tmp on /tmp (zfs, local, nfsv4acls)
tank/usr/home on /usr/home (zfs, local, nfsv4acls)
tank/usr/home/daichi on /usr/home/daichi (zfs, local, nfsv4acls)
tank/usr/home/gyomu on /usr/home/gyomu (zfs, local, nfsv4acls)
tank/usr/jails on /usr/jails (zfs, local, nfsv4acls)
tank/usr/jails/.warden-template-9.2-RELEASE-amd64 on /usr/jails/.warden-template-9.2-RELEASE-amd64 (zfs, l
ocal, nfsv4acls)
tank/usr/jails/Jailbird on /usr/jails/Jailbird (zfs, local, nfsv4acls)
tank/usr/obj on /usr/obj (zfs, local, nfsv4acls)
tank/usr/pbi on /usr/pbi (zfs, local, nfsv4acls)
tank/usr/ports on /usr/ports (zfs, local, nfsv4acls)
tank/usr/ports/distfiles on /usr/ports/distfiles (zfs, local, nfsv4acls)
tank/usr/src on /usr/src (zfs, local, nfsv4acls)
tank/var/audit on /var/audit (zfs, local, nfsv4acls)
tank/var/log on /var/log (zfs, local, nfsv4acls)
tank/var/tmp on /var/tmp (zfs, local, nfsv4acls)
/usr/home/gyomu on /usr/home/gyomu (pefs, local)
[daichi@pcbsd] ~%
```

て実装されています。ext4はext2からいくつもの機能拡張を経たバージョンで、ジャーナリングの機能を備え大容量ストレージに対応しています。機能も豊富です。

最近ストレージシステムのファイルシステムとして採用されることが多いのがZFSというファイルシステムです。SolarisやOpenSolarisを源流とするOSで広く採用されています。ZFSを採用しているOSで最もシェアが多いのがSolarisまたはOpenSolarisを源流とするOS、第2シェアがFreeBSDと言われています。最近はオープンソースでのZFS開発を進める取り組みが「OpenZFS」として正式に発足され、開発は今後さらに活発に継続される見通しです。

FreeBSDがデフォルトで採用しているファイルシステムはUFS2(Unix File Sytem 2)です。UFSの設計思想をそのままに機能を拡張したファイルシステムです。FreeBSDではファイルシステムのジャーナル化をするのではなく、Soft updatesと呼ばれる機能を導入して処理性能を引き上げるとともに、Soft updatesそのものをロギングの対象とするという方法で利便性の引き上げを実現しました。

Linuxで使用されるファイルシステムは多岐に及びますが、ZFSに締めて言及しておいたほうがよいファイルシステムとしてBtrfs(B-tree file system)を取り上げておきます。BtrfsはZFSの設計思想に影響を受けて開発されているファイルシステムです。Linuxでネイティブに動作するZFSが登場したことでBtrfsが今後デフォルトのファイルシステムとしての立場を確保するかどうかはわからない状態ですが、UFS系のファイルシステムに代わるものとして期待されてきました。

ファイルシステムに影響を与えるもの

ファイルシステムの処理性能はさまざまな要因に影響を受けます。同じファイルシステムであってもフォーマット時のオプションを変えるだけで大きく性能が変わりますし、マウント時に指定するオプションによっても性能が変化します。ハードウェアの特性やオペレーティングシステムが提供するキャッシュレイヤなどの影響も受けます。ファイルシステムからハードウェアに至るまでのレイヤの影響も受けます。

・ファイルシステムの操作方法
・マウント時のオプション
・フォーマット時のオプション
・ファイルシステムサブシステムの実装
・ストレージハードウェア
・そのほかのハードウェア

ソフトウェアの構成をまったく同じにしたとしても、使用するハードウェアが変わればファイルシステムの性能も変わります。ファイルシステムの性能を発揮させるには、実際に使用するハードウェアで試験を実施して、そのハードウェア向けの設定やオプションを探すしかありません。

不整合への3つ異なるジャーナリングアプローチとは

ext4のジャーナリング対応

ファイルシステムは実際にデータが書き込まれている「データブロック」と、どのデータブロックがどのファイルを構成しているかを記録している「メタデータ」という2つの情報を持っています。UFS系のファイルシステムであればメタデータは「i-node」と呼ばれています。i-nodeにファイルサイズや実際のデータブロックを指し示すデータが収められています。こうしたファイルシステムにおいて、ファイルにデータを書き込む作業というのは、

①データを書き込む
②メタデータを更新する

という作業になるわけですが、①と②の作業の間にシステムがパニックしたり、電源の不具合

が理由でシステムが再起動するなどすると、実際のデータの状態とメタデータの状態が一致しない状況が生まれます。

UFS系のファイルシステムではこの不整合を解消するためにfsck(8)というツールを使ってメタデータを実際の状況と合うように書き換えます。場合によってはその過程でデータが失われることもあります。fsck(8)のようなツールはそれぞれのオペレーティングシステムで専用のツールとして提供されています。

fsck(8)が問題になりはじめたのは、とくにストレージデバイスの容量が増えてきてからです。fsck(8)はファイルシステムのすべてのメタデータとデータブロックをチェックして不整合を見つけ出しそれを修正します。このため、ストレージサイズが大きくなり、含まれているファイルの数が増えてくると、fsck(8)が完了するまでに長い時間がかかります。

この問題に対処する方法の1つが、ファイルシステムに「ジャーナル」と呼ばれる領域を確保し、ここにデータを書き込む前にメタデータを書き込むようにするという方法です。不慮の再起動などに遭遇した場合でも、ジャーナルにはメタデータが記録されていますので、このメタデータ情報を使ってファイルシステムを整合性がとれた状況へ戻します。fsck(8)でファイルシステムをすべてチェックするといったことをする必要がなくなりますので、結果的に高速に整合性化処理を終えることができます。ext4にはこのジャーナル機能が実装されています。UFS系ファイルシステムでは広く採用されたアプローチです。

Soft updates+SUJ

一方、ジャーナルを作らずにジャーナルと似たような効果を実現させながら、さらに処理性能の向上を実現した技術がFreeBSDで実装された「Soft updates」です。

Soft updatesはメタデータの非同期書き込みを実現する技術です。データおよびメタデータの書き込み順序を整理していくと、メタデータを同期書き込みしなくとも整合性を保つことができる手順があることがわかりました。Soft updateはそれを実装したものです。

このしくみではどの段階でシステムが再起動しても不整合が発生しません。ただし、唯一使われていないのに「使われている」とマークされたデータブロックが発生してしまうので、これをクリアにするためにバックグラウンドでfsck(8)を実行するといった操作が実施されます。これはファイルシステムの処理性能を引き上げる効果も持っています。UFS系のファイルシステムはその構造上、絶対的にi-nodeへの書き込みが発生します。Soft updateではこのi-nodeへの書き込みを非同期に実施します。同期書き込みの場合と比較して書き込み回数そのものを減らすことができますので、結果としてディスクへの書き込み回数が減り高速性が増します。

Soft updatesはジャーナル領域を使う方法とは異なり、メタデータをジャーナル領域に書き込むといったことはしません。既存のファイルシステムに変更を加えることなく使用できます。このあたりもほかのジャーナル機能を持ったファイルシステムと異なる特徴です。ただし、最近はSUJ(Soft updates Journal)と呼ばれる機能が実装され、ext4のようにジャーナルに似た領域も使えるようになりました。Soft updatesで使われるデータをジャーナルの対象とする機能です。この機能を使うとfsck(8)の時間が劇的に短くなります。最近のFreeBSDではSoft updatesとSUJの双方が最初から有効になっています。

ZFS

ZFSではext4のジャーナル機能や、UFS2のSoft updates+SUJとも違うアプローチが取られています。ZFSはファイルシステムの設計そのものをUFS系のファイルシステムのようにするのではなく、そもそもそういった状況が発生しないように根本的に設計を変えました。どちら

かといえばデータベースの設計に似ていると考えるとよいかもしれません。

ZFSの構造はUFSと比べるとかなり複雑です。i-nodeとデータという関係はかなりシンプルでわかりやすいモデルですが、ZFSのデータ構造はもっと混み入っています。

ZFSは複雑な構造をしているため、速度の面ではどうしてもUFS系のファイルシステムよりも遅いところがあります。さらにメモリを大量に必要とするという特徴も抱えています。反面、その機能はとても豊富です。ファイルシステムだけでなくボリューム管理機能も提供していますし、ZFSの便利さはZFSを使っている管理者であれば日々感じているところでしょう。

FreeBSDのSoft updates

機能のON/OFFでの違い

簡単な例を見てみましょう。最近のFreeBSDはデフォルトでSoft updatesとSoft updatesのジャーナリングを有効にしたUFS2フォーマットをデフォルトにしています。使用されているファイルシステムはmount(8)コマンドを引数なしで実行することで確認できます(**図5と図6**)。

▼図5　FreeBSD 9.2-RELEASEでdf(1)を実行

```
# df
Filesystem  1K-blocks    Used    Avail Capacity  Mounted on
/dev/ada0p2  28434076 3233492 22925860    12%    /
devfs               1       1        0   100%    /dev
/dev/cd0      2493970 2493970        0   100%    /mnt
#
```

▼図6　mount(8)でファイルシステムを確認

```
# mount
/dev/ada0p2 on / (ufs, local, journaled soft-updates)
devfs on /dev (devfs, local, multilabel)
/dev/cd0 on /mnt (cd9660, local, read-only)
#
```

▼図7　dumpfs(8)の出力は数百行(ここでは700行以上)におよぶなど多岐に渡る

```
# dumpfs / | wc -l
     716
#
```

▼図8　dumpfs(8)はファイルシステム情報を詳細に出力する(先頭から20行分を表示)

```
# dumpfs / | head -20
magic   19540119 (UFS2) time    Fri Oct 11 14:57:04 2013
superblock location     65536   id      [ 5250413f 5a2ec3c ]
ncg     46      size    7340016 blocks  7108519
bsize   32768   shift   15      mask    0xffff8000
fsize   4096    shift   12      mask    0xfffff000
frag    8       shift   3       fsbtodb 3
minfree 8%      optim   time    symlinklen 120
maxbsize 32768  maxbpg  4096    maxcontig 4     contigsumsize 4
nbfree  787335  ndir    73647   nifree  3313575 nffree  1467
bpg     20035   fpg     160280  ipg     80256   unrefs  0
nindir  4096    inopb   128     maxfilesize     2252349704110079
sbsize  4096    cgsize  32768   csaddr  5056    cssize  4096
sblkno  24      cblkno  32      iblkno  40      dblkno  5056
cgrotor 0       fmod    0       ronly   0       clean   0
metaspace 6408  avgfpdir 64     avgfilesize 16384
flags   soft-updates+journal
fsmnt   /
volname         swuid   0       providersize    7340016

cs[].cs_(nbfree,ndir,nifree,nffree):
#
```

より詳しいファイルシステムの情報はdumpfs(8)コマンドで確認できます。Linuxではdumpe2fs(8)で同様の情報をチェックできます(**図7、図8**)。

Soft updatesの有効になったUFS2は細かいファイルを作っては削除するといった操作や、ファイルやディレクトリの削除操作が高速という特徴があります。この状態で17万弱ほどのファイルとディレクトリで構成されたPorts Collectionを削除するのにかかる時間を計測してみます。**図9**では19秒ほどかかっています。

次に、システムをシングルユーザモードで再起動して、このファイルシステムにおけるSoft update機能を無効にします(**図10、図11**)。

この状態で先ほどと同じようにPorts Collectionを削除すると、**図12**のように71秒ほど時間がかかります。

簡単な試験ですが、このように機能の有効化/無効化だけでも大きく性能が変わります。この手の機能に万能というものはなく、何かの性能を引き上げたり何かの機能を有効にすれば、ユーザにとって良い方向へも悪い方向へも影響がでます。どの機能を有効にするかではなく、求めている性能に対してどの機能の組み合わせがベストか、それを設定するというのが肝ということになります。

キャッシュによる動作の違いはどの程度か?

ファイルシステムは設定のみならず、使い方でも大きく性能が変わります。次の簡単な例を見てみましょう。まず、**図13**のようなデータファイルを用意したものとします。このファイルは1GBほどのテキストファイルです。

図14では、このデータファイルから該当するデータをgrep(1)で取り出す操作を実施し、その実行時間を計測してみます(Linuxで同じ実験をするときはtime(1)の引数に-vオプションを指定してください)。

表1でとくに注目すべきはブロック入出力回数の違いです。1回目は8405回ですが、2回目以降は0回です。ブロックの転送単位は128KBですので、8405回実行されてだいたい1GBほ

▼図9　Ports Collectionを削除[その1]

```
# /usr/bin/time -p rm -r ports
real 19.10
user 0.16
sys 6.40
#
```

▼図10　Soft updatesのジャーナル機能を無効化

```
# tunefs -j disable /
Clearing journal flags from inode 4
tunefs: soft updates journaling cleared but soft update still set.
tunefs: remove .sujournal to reclaim space
tunefs: file system reloaded
#
```

▼図11　Soft updates機能を無効化

```
# tunefs -n disable /
tunefs: soft updates cleared
tunefs: file system reloaded
#
```

▼図12　Ports Collectionの削除[その2]

```
# /usr/bin/time -p rm -r ports
real 71.29
user 0.06
sys 12.60
#
```

▼図13　1GBほどのテキストデータファイル

```
# head DATA
0000000001
0000000002
0000000003
0000000004
0000000005
0000000006
0000000007
0000000008
0000000009
0000000010
#
```

どになります。この数は計算どおりです。

カーネルはメモリが余っているならば、可能な限りキャッシュとして活用しようとします。1回目のgrep(1)ですべてのデータがキャッシュに乗ります。2回目以降はキャッシュからデータを持ってくるため、ディスクまでアクセスしません(**図15**)。すべての処理はオンメモリで実行されるので、その分高速化されているというわけです。

このように、ファイルシステムの使い方しだいで大きな処理速度の高速化ができますので、ファイルシステムそのものの性能にも注目する必要があるのですが、その活用方法にも注目したほうが良いことになります。

デバイスによる特性の違いはどの程度か?

ファイルシステムの処理性能は利用するストレージデバイスの性能にも左右されます。たとえば次にIO Zone(iozone)というファイルシステムの特性を調査するソフトウェアによる計測結果を掲載します。

IO Zoneはファイルシステムにおけるシーケンシャルなデータの転送速度を計測するソフトウェアです。ファイルシステムにおけるデータの入出力に関与するシステムコールをレコードサイズを変更しながら、特定のファイルサイズに到達するまで処理を繰り返し、その処理のデータ転送速度を計測します。ファイルシステム、カーネルにおけるシステムコールの実装、ファイルシステムからデバイスまでのレイヤ、ハードウェア(ディスク、プロセッサ、バスなど)など、さまざまな要素に影響を受けた値を計測できます。

HDDがハードディスクを使ったときの結果、SSDはそのままSSDを使った場合の結果です。たとえば、**図16**のHDD writeのグラフを見てみましょう。X軸がファイルサイズ、Y軸がレコードサイズ、Z軸がデータの転送速度です。単位はすべてKBです。このグラフを見ると、このハードディスクをUFSでフォーマットして使っている場合、write(2)システムコールの傾向として、書き込み速度が高速になるスポットが存在していることがわかります。レコードサイズとファイルサイズが特定の範囲内に収まっている間は転送速度が出ていますが、その範囲

▼表1　ディスクキャッシュの実験

	1回目	2回目以降
実行時間	1.72秒	0.75秒
ブロック入力回数	8405回	0回
自発的コンテキストスイッチ回数	81回	5回
強制的コンテキストスイッチ回数	8789回	30回

▼図14　実行1回目

```
# /usr/bin/time -lph grep 0123456789 ↵
DATA > /dev/null
real 1.72
user 0.77
sys 0.62
      1916  maximum resident set size
        74  average shared memory size
      1745  average unshared data size
       114  average unshared stack size
        95  page reclaims
         0  page faults
         0  swaps
      8405  block input operations
         0  block output operations
         0  messages sent
         0  messages received
         0  signals received
        81  voluntary context switches
      8789  involuntary context switches
#
```

▼図15　実行2回目

```
# /usr/bin/time -lph grep 0123456789 ↵
DATA > /dev/null
real 0.75
user 0.53
sys 0.21
      1916  maximum resident set size
        84  average shared memory size
      1988  average unshared data size
       129  average unshared stack size
        95  page reclaims
         0  page faults
         0  swaps
         0  block input operations
         0  block output operations
         0  messages sent
         0  messages received
         0  signals received
         5  voluntary context switches
        30  involuntary context switches
#
```

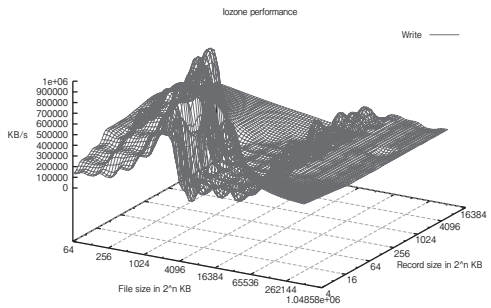
を超えると一気に転送速度が低下しています。

図17のSSDをUFSでフォーマットして使っている場合のwrite(2)システムコールのグラフを見てみましょう。ハードディスクと異なり書き込み速度が全体的に高速であるほか、高速な書き込みが実施されるスポットエリアが広範囲に及んでいます。このように両者の特性が異なっていることがわかります。このように同じファイルシステムを使っていても、ストレージデバイスの特性が異なれば、その性能に違いが現れます(図16〜25)。

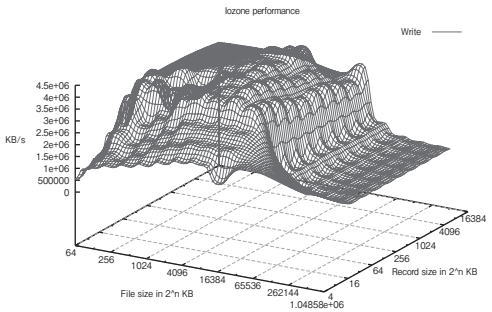
SSDは使い続けると特性が変化する

ハードディスクは回転する円盤へデータを書き込みます。UFS2はもともとハードディスクでの使用を想定して設計されたことから、このデバイスで性能が発揮できるしくみになっています(ただ、現在のハードディスクはUFSが設計された当初からはだいぶ中身が変わっているので、設計どおりの動作にはなっていません)。

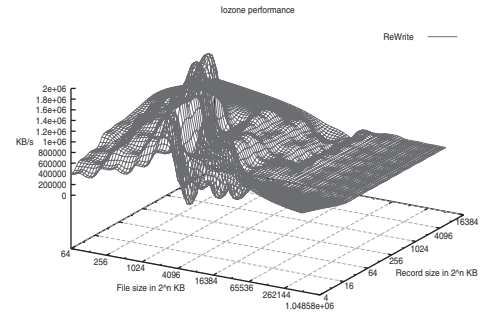
▼図16　HDDのWrite(書き込み)結果



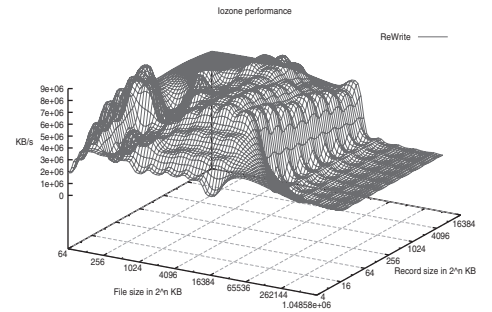
▼図17　SSDのWrite(書き込み)結果



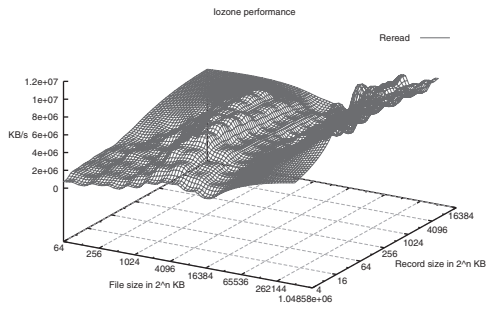
▼図18　HDDのRewrite(再書き込み)結果



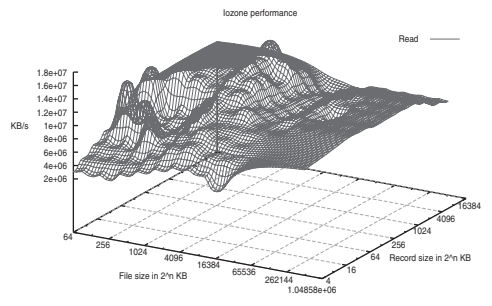
▼図19　SSDのRewrite(再書き込み)結果



▼図20　HDDのRead(読み込み)結果



▼図21　SSDのRead(読み込み)結果



一方、SSDはハードディスクとは大きく構成が異なります。この結果が上記の結果に現れています。またここには掲載しませんが、SSDのほうは使い続けることで特性が変化するという特徴があります。SSDで使用されているファームウェアがどのような動きをしているか知るすべを持たないので想像するしかないのですが、処理の最適化やバランスを取るためのしくみによって、動作が変化するのだと思われます。ファイルシステムの性能はこのようにハードウェアそのものの特性にも左右されます。

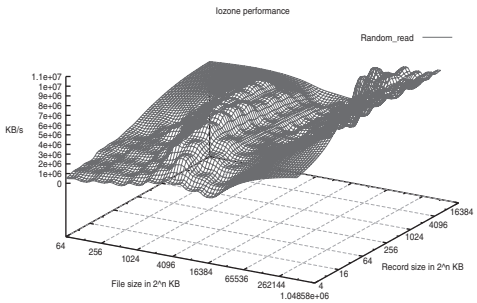
性能を見極めて利用していますか?

ファイルシステムの性能はファイルシステムの設計や構造からある程度予想できます。ZFSはボリューム管理からファイルシステムまですべてに渡ってかなり柔軟な機能と設定を提供しますが、そのつくりのため細かいファイルの作成や削除、読み込みなどを繰り返す作業がUFSと比較して遅くならざるを得ないこともわかります。メモリも大量に消費します。

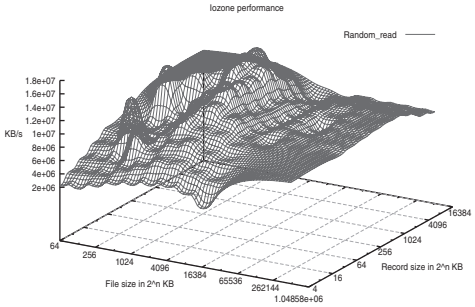
しかし、ここに紹介したようにファイルシステムはファイルシステムの構造だけでなく、関連するサブシステムやデバイスの特性にも影響を受けますし、使い方しだいで性能も大きく変化します。ファイルシステムの特性云々を調べる前に、まずは実際に使って性能を測定してみることが大切です。

知りたい性能が定まっているのであれば、その処理を実際に実施して計測して比較してみましょう。知りたい性能があいまいな状態であれば、Bonnie++などのベンチマークソフトウェアを使って性能を比較してみるというのもアリだと思います。まずは使ってみて、マウントオプションを変えたりフォーマット時のオプションを変えてみて比較してみる、これを繰り返すことでいろいろなことが見えてきます :) SD

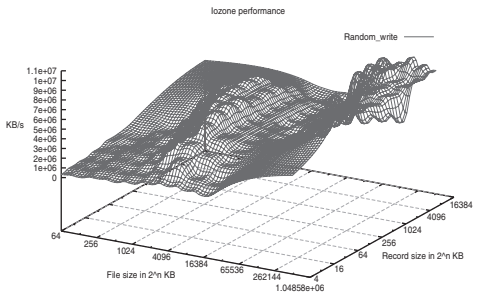
▼図22　HDDのRandom Read(無作為読み込み)



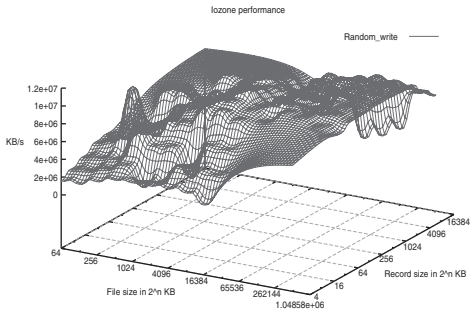
▼図23　SSDのRandom Read(無作為読み込み)



▼図24　HDDのRandom Write(無作為書き込み)



▼図25　SSDのRandom Write(無作為書き込み)



未来を見透す ヘッドマウントディスプレイ

「Mirama」a.k.a. VIKING

後藤 大地(ごとう だいち)
(有)オングス

Google Glassの発表によって、メガネ型のディスプレイを使った情報機器への関心が高まっています。ブリリアントサービスが開発を進めている「Mirama」および「Mirama OS」は、そんなメガネ型情報機器と専用OSです。本稿は開発担当者への取材を通し、Mirama OSが採用する技術とその理由、Mirama OSのしくみや今後の展開ロードマップなどを紹介します。

スマートフォンは時代遅れなのか?

デザインが刷新されたiOS 7がリリースされました。iOS 7と指紋認証機能、64bitプロセッサを採用して高機能・高速化されたiPhone 5sの販売も開始されています。NTTドコモでもiPhoneの取り扱いがはじまったこともあって、スマートフォンの需要とシェアはますます広がる傾向。電車の中はスマートフォンを操作している人ばかりです。

こうしたコンシューマ市場を尻目に、企業における研究開発は透過型のヘッドマウントディスプレイ(HMD)を眼鏡型のウェアラブルコンピュータへ適用した、いわゆる「スマートグラス」の研究開発へ向かっています。スマートグラスでは眼鏡のレンズ部分がディスプレイとレンズの双方の機能を持っています。つまりレンズを通して辺りが見えつつ、ディスプレイとしてそこに映像をオーバーラップさせるといったことができるしくみになっています。

これまでに製品化されたHMDはいくつもありますが、現在のスマートフォンのように広くコンシューマ市場に受け入れられる製品はありませんでした。しかし、現在は事情が異なり、当時普及のハードルと考えられていた問題点のいくつかがクリアされています。たとえば、プロセッサの消費電力がきわめて低くなり、ダウンサイジングも進んでいます。必要になる回路をすべて単一のチップで実装することができ、眼鏡型のデバイスでPCと同レベルの演算能力を持つことが可能になりつつあります。また、液晶ディスプレイ技術も進んでいます。バッテリーに関してはさらなるブレイクスルーが必要ですが、スマートグラスに必要となるハードウェア技術は整いつつあり、今後数年でコンシューマ市場に一気に広がる可能性があるという状態です。

とはいえ、スマートフォンに慣れた方からすれば、眼鏡型デバイスの普及については懐疑的ではないでしょうか。操作が曖昧そうですし、使いものにならないような気がします。実際に使ってみないことには、一過性のものなのか、今後モバイルデバイスに変革をもたらすものなのか、実感としてつかめないと思います。

2013年9月、英国ケンブリッジで開催された開発者会議「BSDCam 2013」において、HMD向けオペレーティングシステム「Mirama OS」の開発者であるJohannes Lundberg(ヨハネス・ルンベリ)氏と議論する機会を得ました。Lundberg氏はFreeBSD Vendor SummitにてMirama OSの発表をするため、また、FreeBSD開発者会議に参加するためにBSDCamに来ていました。このときにプロトタイプである試作機マークIを

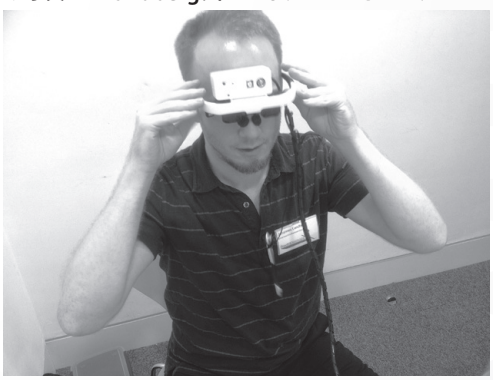
使わせてもらうなど興味深い体験ができました（**写真1**）。Lundberg氏からMirama OS誕生について採用した技術とその理由、Mirama OSのしくみや今後の展開ロードマップなどについてうかがいましたのでご紹介します。

新しい体験、ゼロからのブレイクスルー「Mirama OS」

Mirama OSは株式会社ブリリアントサービスが2013年2月26日に発表したコンシューマ市場向けHMDオペレーティングシステムです[注1]。FreeBSD 10-CURRENTをベースにしたオペレーティングシステムで、独自開発の認識エンジンを搭載し、眼鏡前部に設置されたカメラから得られる画像から人間や風景を理解し、ジェスチャによる操作や見ている風景に地図を重ねて表示するといったことができます。

注1）発表当時はVIKINGおよびVIKING OSという名称が使われていましたが、商標の関係で後日MiramaとMirama OSへ名称が変更されました。発表当時はVIKINGでしたが、本記事では変更後の名称に統一してあります。

▼写真1　Lundberg氏によるデモンストレーション



Miramaのコンセプトモデル（**写真2**）はフレーム部分、レンズ部分、耳あて部分などで色が違います。これはたとえば側面の右のフレームにはプロセッサ、左のフレームにはバッテリー、レンズは透過液晶ディスプレイというように個別のパーツが入っており、部品ごとに交換してバージョンアップが可能になっている、というコンセプトを表現しています。現在は試作機の段階ですのでこうはなっていませんが、いずれはこのような製品が市場に出てくるかもしれません。

試作機マークIを使ってみた

使わせてもらった試作機はMiramaマークIと呼ばれていました。カメラ、透過型ディスプレイ、ノートPCの組み合わせになっています（**写真3**）。カメラで景色を画像として取り込み、ノートPCで処理、処理結果を透過型ディスプレイに出力するというしくみです。本質的にはこれが基本構成で、ダウンサイジングを進めてこれをすべて眼鏡型のフレームにはめ込むと、コン

▼写真2　Miramaコンセプトモデル

Mirama a.k.a VIKINGの名前の由来

Mirama OSはもともと「VIKING OS」という名前で発表されました。VIKING OSの“VIKING（Lundberg氏はバイキングではなくヴァイキングだと説明してくれましたけれども）”の由来が気になるところです。Lundberg氏が言うには「僕がスウェーデン人だから、ジョークでつけたんだ」とのことで、「商標の関係で名前は変わると思う」とも説明してくれました。そのとおり後日、名称は「Mirama OS」へ変更されました。

Lundberg氏は「Mirama」という名前は、「見て驚く」という意味の日本語と英語から作った造語だと説明してくれました。「見て驚く」→「見る＋Amazed.」→「Miru＋Amazed.」→「Mir(u)＋Ama(zed.)」→「Mirama」ということです。見て驚くからMirama。このデバイスの特徴を表しているように思います。

セプトモデルのような最終形態になります(**写真4**)。

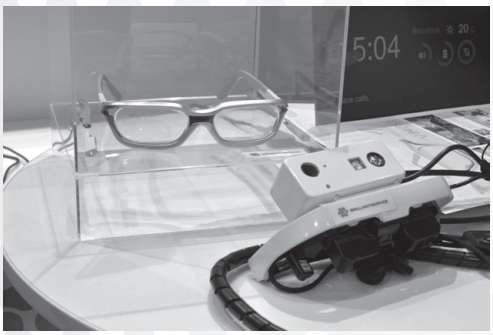
実際に使わせてもらった試作機の表示部は**写真5～8**のようなものでした。カメラと透過型ディスプレイが結合してHMDのような作りを構成しています。

ノートPCが画像データを取得して認識処理を実行、それに合わせてかぶせて表示する画像データを生成し、リアルタイムに透過型ディスプレイに出力しています(**写真9**)。ノートPCにはディスプレイ越しに見えているものが表示されるしくみになっています。ノートPCの電源を入れるとFreeBSD 10-CURRENTが起動し、X Window Systemが起動してきて、いくつものターミナルが起動してから動作モードに移ります(**写真10、11**)。

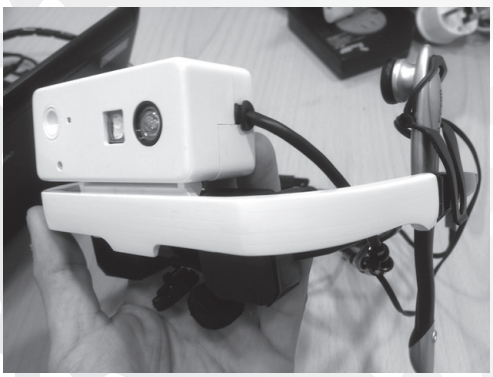
HMDをかけている本人には、周りの風景に対

▼写真3　試作機のイメージ画像：カメラ、ディスプレイ、PCを組み合わせている

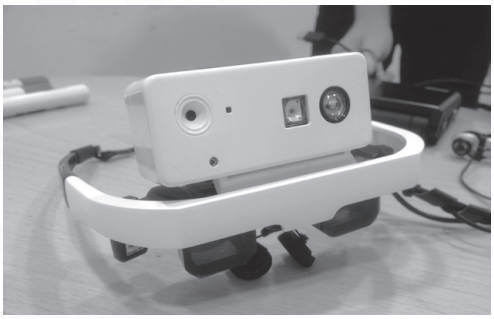
▼写真4　試作機は最終的にコンセプトモデルのようなデバイスへダウンサイジング



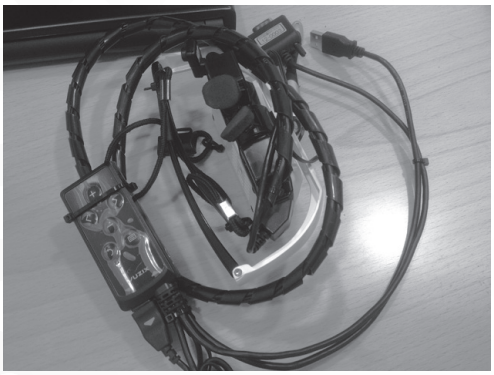
▼写真5　カメラとヘッドマウントディスプレイが一体化



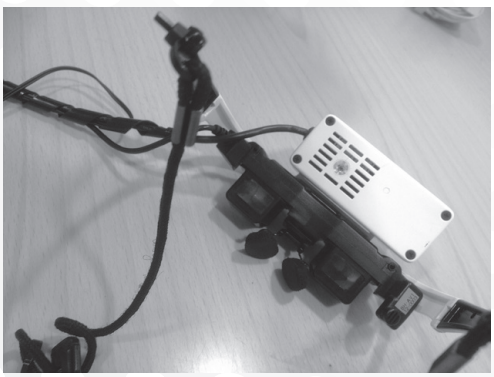
▼写真6　斜め方面から見たところ



▼写真7　上部から見たところ



▼写真8　内側から見たところ

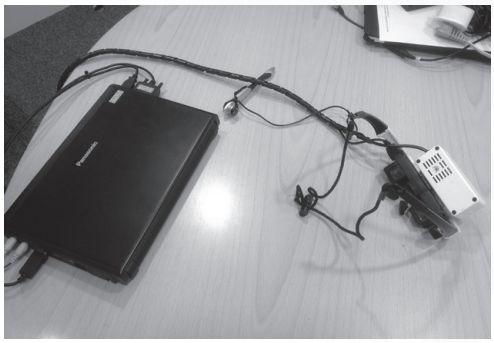


して地図やランドマークがかぶさるように表示されています(**写真12**)。目の前にはアイコンが表示され、人差し指で指すとアイコン全体がクルッと光って選択できます(**写真13**)。顔を上へ向けると別の情報が表示されます。ジェスチャでビューの高さを変更したり、アプリを起動することができます(**写真14**)。

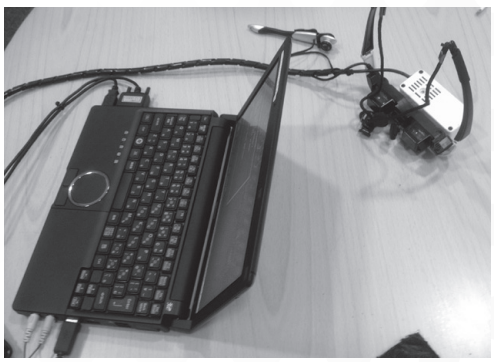
不思議に思ったのは、実際には存在しないアイコンを操作するのに違和感を感じなかったことです。指先での触感が得られなくても、視覚的な変化が得られるとごく自然なこととしてストンと受け入れてすぐに慣れるようです。

現在は(筆者も)スマートフォンを操作しながらナビゲーションを実施して目的地へ移動しています。このナビゲーション作業を眼鏡を通してやれるようになれば、かなりのストレス削減

▼写真9　カメラ・ディスプレイユニットとノートPCをUSBケーブルで結合



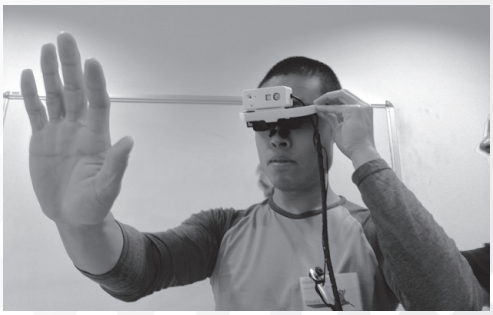
▼写真10　ノートPCではFreeBSDとX Window System、独自の認識エンジンとアプリケーションが動作



▼写真11　Mirama OSの起動画面

▼写真12　ノートPCには本人が見ているものと同じ画面が展開されている。壁を超えて向こう側にあるランドマークや地図がオーバーラップで表示されている

▼写真13　本人は表示されているアイコンを操作している



▼写真14　顔を上げると別の画面(情報表示)が見える

になると感じました。スマートフォンを操作しながらのナビゲーションは画面と景色の間で視線を移動させる必要があります。眼鏡越しにかぶさるように表示されればその必要がなくなります。

FreeBSDとObjective-C採用の理由とは

Mirama OSはベースのオペレーティングシステムにFreeBSD 10-CURRENT、コンパイラはLLVM Clang[注2]、アプリケーションフレームワークはGNUstep[注3]を採用しています。決定要因はいくつかあるそうですが、要点をまとめると次のようになります。

- Android向けのアプリケーションも開発しているが、Android上で開発できるシステムには制限があり、スマートグラスといったようなまったく新しいユーザ体験を提供するには、新しい環境を最初から構築する必要性を感じていた
- 最終的にどのようなビジネスモデルになっていくかは検討を続けているが、ハードウェアとともに販売することを考えると、オペレーティングシステムやコア部分のソフトウェアのライセンスは保護可能なライセンスである必要がある
- Intelプロセッサの省エネ化とダウンサイジングが進んでおり、HMDの開発において十分に採用できるレベルになってきた
- iOSおよびMac OS X向けのアプリケーション開発者が増加し、Objective-Cに慣れ親しんだ開発者が増えている。このためObjective-Cをアプリケーション開発に採用したい。その場合アプリケーションフレームワークにはCocoa互換のGNUstepの採用が妥当である

Objective-Cコンパイラとしては LLVM Clangが有力候補です。Intelプロセッサで動作し、LLVM Clangがベースシステムにマージされており、GNUstepが利用できて、バイナリのみの配布も可能なライセンスで提供されているオペレーティングシステムということになると、FreeBSD 10-CURRENTが第1選択肢としてあがってきます。

Objective-CとGNUstepを選択した背景には、iOSやMac OS Xのアプリケーションデベロッパが慣れていることに加え、こうしたデベロッパがMirama OS向けのアプリケーションを開発することでエコシステムを実現していきたいといった狙いもあるそうです。

余談ですが、Objective-Cでソフトウェアを開発したいからLLVM Clangを使うといったシーンがMiramaに限らず増えています。その際、LLVM Clangが扱いやすい環境であるということからFreeBSD 10-CURRENTを採用するケースも多々見られます。2013年中のリリースが予定されているFreeBSD 10.0-RELEASEですが、10.0がリリースされたあとはこの傾向が加速するものとみられます。

Linux版ドライバをFreeBSDで使うテクニック

Lundberg氏に開発の話を聞きました。試作機に採用したカメラはUSB接続になっていますが、USB経由で画像を取り出すためのデバイスドライバがプロプライエタリで、Linux版とWindows版しか提供されていなかったということです。このため、Linux版のデバイスドライバをFreeBSDからLinuxバイナリ互換機能経由で利用したということです。

技術的におもしろかったのは、Linux版のデバイスドライバをどうやってFreeBSDから利用できるようにしたか、という部分でした。実はこのテクニックは*BSD系のOSでLinux向けのデバイスドライバを利用する場合にはよく使われているものです。簡単に説明しましょう。

注2) C/C++/Objective-C/Objective-C++向けのコンパイラフロントエンドのClangとバックエンドのLLVMで構成されるオープンソースのコンパイラ。
注3) NeXTのOPENSTEP Objective-C ライブラリ(フレームワーク)、ウィジェット・ツールキット、アプリケーション開発ツール群をフリーソフトウェアとして実装したもの。

提供されているLinux版のドライバはlibusb経由でUSB系の操作を実施していたそうです。libusbはFreeBSDも提供している共有のインターフェースです。Lundberg氏はこのドライバを利用するために、FreeBSDとLinuxの双方から利用できる「カスタムされたlibusb」と「Linux版ドライバを使用するユーティリティソフトウェア」を開発します。

開発の方法は次のとおりです。まず、Lubuntu 12.10にFreeBSD 10-CURRENTのソースコードをダウンロードします。freebsd-makeを使ってFreeBSDをLubuntu上でビルドします(ビルドできるようにlibusbに若干の変更は加えたりヘッダを追加したりしたそうです)。このビルドされたlibusbとLinux版ドライバを使って画像データを取得し、FIFOファイルへデータを出力する小さなユーティリティソフトウェアを開発して、同じくLubuntuでビルドします。

Lubuntuでビルドしたlibusb、ユーティリティソフトウェア、そのほか依存するライブラリなどをホストとなるFreeBSD 10-CURRENTの/compat/linux/以下へデプロイします。このLinuxバイナリはもともとFreeBSDのソースコードで、かつ、Linuxでビルドされたものです。FreeBSDのLinuxバイナリ互換機能で問題なく動作します。

Linuxバイナリのユーティリティは FreeBSD上で動作し、FIFOファイルへ画像データを出力し続けます。FreeBSD 10-CURRENTではFIFOファイルから画像データを吸い出して認識処理をするネイティブアプリケーションを動作させます。こうすることで、Linux向けのデバイスドライバを問題なく活用できます。つまり、次のような流れを経てデータが取得されていることになります。

> デバイス→カーネル(FreeBSD カーネル)→libusb(Linux バイナリ)→ユーティリティ(Linux バイナリ)→[FIFO ファイル]→FreeBSD ネイティブアプリ(FreeBSD バイナリ)

FreeBSDのLinuxバイナリ互換機能は、Linuxバイナリも FreeBSDバイナリと同じように扱います。適切な状態で用意すればこのように組み合わせて利用することができます。

Mirama OSマスタープラン、2016年プロダクト目指す

Miramaは2016年にプロダクトの出荷を目指しているということです。試作機はマークIIとマークIIIまでを予定しており、次の流れで製品へ進めていくと説明してくれました(マークII、マークIIIといった名称は映画『アイアンマン』シリーズを彷彿とさせますね)。

- 2014年モデル:試作機マークII。よりタイトなヘッドマウントデバイス。ノートPCはFreeBSD on MacBook Airへ差し替え
- 2015年モデル:試作機マークIII。ノートPC部分をペンダントサイズの小型PCへ置き換え、Bluetoothを使ってヘッドマウントデバイスと通信
- 2016年モデル:すべての装置を眼鏡型デバイスに搭載

当初は産業界向けに提供することになるだろうと説明してくれましたが、最終的にはコンシューマ市場向けに広く提供できるプロダクトを目指すということでした。

試作機マークIIに関してはすでにBTOによる事前注文が開始されています。本誌が販売されているころにはマークIIも販売されている見通しです。2014年の2月には、スペインのバルセロナで開催されるMobile World Congress 2014でExhibition Prototype IIが発表される見通しです。

Mirama OSに限らず、眼鏡型の新しいモバイルデバイスの開発は世界中の大企業からベンチャー企業まで、多くの開発者が取り組みを継続している、または新しい取り組みをはじめています。今後数年の間は草創期としてエキサイティングな時期になるのではないかと思います。

SD

iOS 7アプリの魅力はどうやって引き出すのか

従来機とのデザイン調整、ゲーム開発を促すSprite Kit、進化したデバッグ&テストツール……

フラットデザインへの変更がクローズアップされているiOS 7。ユーザの使い勝手が変わるということは、アプリ開発者にとってはユーザビリティの見なおしを迫られているということ。より良い体験、今までになかったサービスを提供するために、ぜひとも取り入れたいフレームワークや開発ツールを紹介しましょう。

中野 洋一（なかの よういち）　ドリームガーデンソフトウェア　URL http://www.dreamg.com/

いまこそiOS 7アプリ開発へ

2007年に初代バージョンのiPhoneが発表されて以来、iOSは年を重ねるごとに機能や操作性を向上させ、Appleにとって看板的な商品になりました。そのiOSが9月にバージョンアップしてiOS 7となりました。画面デザインを一新し、過去最大とも言えるバージョンアップを成し遂げ、未来を先取りするような多くの新技術が盛り込まれました。

また、iPhone 5s/5cもOSとほぼ同時期に発売され、さらには10月にiPad AirとRetinaディスプレイ版のiPad miniが発表となり、iOSデバイスのラインナップが一気にアップデートしました。通信キャリアに関してもNTTドコモの参入により、日本におけるiPhoneの販売合戦もヒートアップすることは間違いなさそうです。いずれにせよ、iOSを取り巻くビジネスは今後も盛り上がり、その勢いは加速していくと期待しています。

iOSプログラマにとっては、iOS 7のリリースが今年最大のインパクトのあるニュースでした。ユーザインターフェースを刷新し、数多くのフレームワークやAPIが追加され、さらにiPhone 5sの登場にあわせてOSの64bit化も成し遂げました。これだけ一度にOSが進化してしまうと、プログラマとしてはうれしい反面、新しい技術を習得することへの不安を感じてしまうこともあるでしょう。Appleの技術資料に目を通したり、既存のアプリをiOS 7へ対応させたりと、目の前にたくさんの課題が積み上げられたような気分になります。さらに、iOS 7のフラットデザインも理解して、アプリのユーザビリティ向上にも目を向けなければなりません。

iOSプログラマとしては困難な状況かもしれませんが、次の時代を見据えたアプリ開発を目指すには、一歩ずつ着実に各フレームワークの知識を習得し、その能力を引き出すことが重要だと考えています。そこで本稿では、iOS 7の中から筆者がとくに気になった技術を取り上げ、その特徴をご紹介いたします。誌面の関係ですべてのフレームワークに触れることはできませんが、iOS 7の魅力を少しでも感じ取っていただけたらうれしいです。

一新したデザイン

スキュアモーフォズムからフラットデザインへ

iOS 7では、今までの写実的なデザインからシンプルなデザインに変わりました。専門的な用語では、前者にスキュアモーフォズム、後者にフラットデザインという言葉を使っていますが、実際に使ってみると操作性も変化していることがわかります。

ホーム画面を見ると、アイコンの並びは同じですが、デザインのテイストや雰囲気は大きく

変わりました(**図1**)。フラットデザインに対しては賞賛する声や落胆するユーザとで賛否両論ですが、しばらく使って慣れてしまえばフラットデザインの良さが理解されていくと思います。アイコンに限らず、ボタンやスライダー、ナビゲーションの背景など、至るところでスキュアモーフィズムは取り除かれました。フラットデザインを採用した理由はいろいろな見解があるようです。ユーザがiOSそのものに十分慣れ親しんだ結果、あえて写実的な表現をしなくても操作を理解できるようになったとか、アプリ自体を引き立てるためにユーザインターフェースは脇役として控えめにするため、などいろいろです。

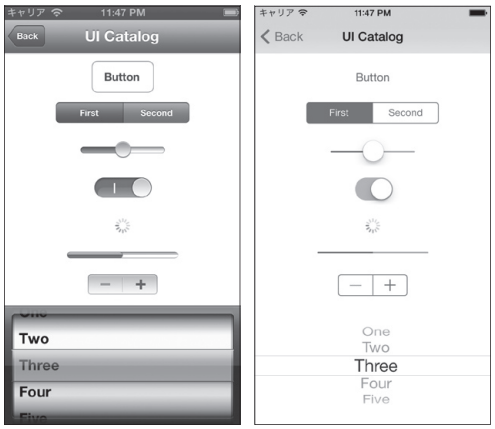
ここでデザインの話を続けると、フレームワークまでたどり着かないので、話題をプログラミングに移します。Xcode 5をインストールして新しいアプリを作ってみると、さまざまなユーザインターフェース(UIKit)のデザインがiOS 6から変わっていることがわかります。**図2**の一番上に配置されたボタンは周りの枠がなくなり、文字だけになりました。もし枠のような装飾をしたい場合は背景に画像を設定します。その下のセグメンテッドコントロールは、以前は3種類のスタイルから選択できましたが、iOS 7では1種類だけになりました。以降は割愛しますが、見た目やタップしたときのアニメーションなども変わっています。ただ、各オブジェクトの大きさは同じですので、以前に作ったアプリの画面が崩れることはなさそうです。

問題はiOS 6と7の両方に対応したアプリの開発です。標準のユーザインターフェースを使ったアプリならば自動的にスタイルが変わるので問題ありませんが、独自の画像やオブジェクトを使っている場合は対応が必要になるかもしれません。たとえば**図3**のナビゲーションバー上に配置した「もどる」ボタンのように、iOS 6用に光沢のある画像を背景に使った場合、iOS 7で実行するとデザインがちぐはぐになります。この場合はiOS 7用にもボタンの背景画像を用意して、OSのバージョンによって切り替えることが望ましいと言えるでしょう。バージョンの判定はNSFoundationVersionNumberを使うと簡単に実装できます(**リスト1**)。

すべてのiOSデバイスがiOS 7にアップグレードされるのはまだ先の話ですので、それまでは過去のOSバージョンにどう対応するのかが、当面の課題になりそうです。

▼図1　iOS 6(左)とiOS 7(右)のホーム画面

▼図2　iOS 7のUIKitオブジェクト



▼図3　iOS 7では不釣り合いな「もどる」ボタン

▼リスト1　OSバージョンの判定

```
if ( floor( NSFoundationVersionNumber ) <= NSFoundationVersionNumber_iOS_6_1 )
{
    // iOS 6.1以前の環境で実行したい処理
}
else
{
    // iOS 7.0以降の環境で実行したい処理
}
```

ステータスバーの違い

画面の一番上にある、電波の状態やバッテリーの残量を表示するステータスバーの扱いがiOS 7になって変わりました。今まではステータスバーとアプリ画面の領域は明確に分かれていたのですが、iOS 7ではアプリの上に覆い被さるように(オーバーラップ)表示されます(**図4**)。アプリは常にフルスクリーン表示になり、ステータスバーの下には背景を表示しなければなりません。

その際、ステータスバーの文字と近い色の背景に設定すると、読みづらくなるケースもあります。iOS 7では、ビューコントローラごとにステータスバーのスタイルを変更できるので、画面に表示する内容に合わせて調整することができます(**リスト2**)。

▼図4　iOS 6(左)とiOS 7(右)のステータスバー

また、上端に配置したオブジェクトがステータスバーと被る問題にも対処しなければなりません(**図5**)。iOS 6ではステータスバーの下に配置されたテキストラベルが、iOS 7ではフルスクリーンとなってステータスバーの領域まで拡がるためにこの問題が起こります。テキストラベルをステータスバー分下げる処理をプログラムに加えるのも手ですが、iOS 6から採用されたAuto Layoutを利用すると、ステータスバーとの距離が適切に保たれるので、OSの違いによる画面の調整を自動的に処理してくれます。

なお、ナビゲーションバーもステータスバーと同様に、アプリ画面の上にオーバーラップして表示されるようになります。iOS 6のようにナビゲーションバーとアプリ画面を分けたい場合は、UIViewControllerに、iOS 7から追加

▼図5　iOS 7でステータスバーと重なってしまったラベルテキスト

▼リスト2　ステータスバーの表示状態を決めるメソッド

```
// ステータスバーの表示状態を決めるUIViewControllerのメソッド
- (BOOL)prefersStatusBarHidden {
    return YES; // ステータスバーを隠す
}

// ステータスバーのスタイルを決めるUIViewControllerのメソッド
- (UIStatusBarStyle)preferredStatusBarStyle {
    return UIStatusBarStyleLightContent; // ステータスバーの文字色を明るい色に設定
                                         // (背景が暗い時スタイル)
}
```

▼リスト3　アプリ画面の領域を設定するプロパティ

```
- (void)viewDidLoad
{
    [super viewDidLoad];
    // ビューをナビゲーションバーやタブバーの領域まで拡大しないように設定
    self.edgesForExtendedLayout = UIRectEdgeNone;
}
```

されたedgesForExtendedLayoutプロパティにUIRectEdgeNoneを設定します(**リスト3**)。

ポイントまとめ

- iOS 7のフラットデザインにマッチしたユーザビリティを検討しよう。
- iOS 6にも対応させる場合は両者のデザイン思想を損なわぬよう、それぞれの画像を用意することが望ましい。
- ステータスバーを含めたさまざまな画面サイズに自動対応させるため、Auto Layoutを積極的に使おう。
- アプリの背景色に応じて、ステータスバーのスタイルを適切なものに調整しよう。

創造力をかき立てるフレームワーク群

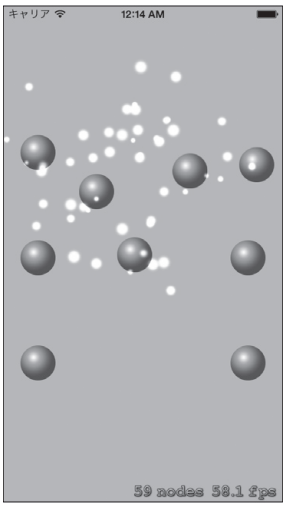
新しい描画エンジンSprite Kit

iOSでは当初から、2Dの画面描画にCore Graphicsフレームワークが使われてきました。もちろんiOS 7でも使うことはできますが、ゲームのような速度を要求されるアプリには不向きです。カーレースやシューティングゲームのようなリアルタイム性の高いアプリには、これまでOpenGLを使うケースがほとんどだと思います。また、物理計算を必要とするようなゲームでは、cocos2dのようなサードパーティの描画エンジンを使って多くのアプリが開発されています。

iOS 7で登場した「Sprite Kit」は、2Dのグラフィックを高速に描画でき、物理計算も容易に実装できる強力な描画エンジンです。Angry Birdsやビリヤードのようなゲームに必要な落下運動や衝突判定などの処理がOS標準のフレームワークとして提供されることになり、iOSのゲームプログラマにとって大きな味方になるでしょう。

フレームワークの使い方はとても簡単で、画像の表示、アニメーション、サウンドの再生、動画の再生、Core Imageによるエフェクトなど、2Dゲームの開発に必要なAPIがひと通りそろっています。さらに、炎や爆発のシーンに使えるパーティクル効果なども演出できます(**図6**)。物理計算は個々のオブジェクトに対する係数や画面全体の重力を設定することで、現実のように物体が跳ね回ったり、衝突して方向を変えるなどのアニメーションを実装できます。また、オブジェクト同士をバネのようにつなげたり、条件に合ったオブジェクトだけを衝突させるなど、さまざまなケースに対応できるよう柔軟な設計になっています。

▼図6　Sprite Kitのパーティクル効果



ただし、Sprite KitはiOS専用のフレームワークです。Androidとのクロスプラットフォームを考えた場合は、移

植性がゼロですので合理的とは言えません。その点を考慮すると、cocos2d、Unity、OpenGLなどは複数のプラットフォームをサポートしているので、Sprite Kitと比べて移植性にアドバンテージがあります。もしiOS専用のゲームアプリを考えるならば、Sprite Kitはとても魅力的でパワフルなフレームワークですので、多くのプログラマにお勧めできるiOS 7の目玉と言える技術です。

表現力豊かなUI実装がより簡単に

iOSの見た目はフラットデザインによって大きく進化しましたが、Appleはさらに一歩進んだユーザインターフェースを作るしくみを我々に提供しています。それが「UIKit Dynamics」フレームワークで、その名のとおりユーザインターフェースに動的な要素を加えられます。

画面上のボタンやキーボードなどは、アクションに対するフィードバックによってユーザに直感的な操作性をもたらします。今まではCore Animationフレームワークを使って、画面上のさまざまなオブジェクトにアニメーション効果を与えることができました。

UIKit Dynamicsでは、Sprite Kitのように落下運動や衝突計算のような物理計算をユーザインターフェースに応用することができます。1つの例を挙げると、iOSのロック画面では右下にカメラのアイコンがあり、そこを上にフリックするとロック画面が持ち上がって下からカメラアプリの画面が見えてきます。途中でフリックをやめると、ロック画面が落ちてきてドスン！という感じでカメラ画面が隠れます(**図7**)。このような効果がUIKit Dynamicsを使うと容易に実装できます。

▼図7　ロック画面で使われているUIKit Dynamicsの使用例

Sprite Kitとほぼ同じような機能なのでおそらく同じエンジンを使っているものと思いますが、あくまでもユーザインターフェース用なのでゲームに使うことは推奨していません。果たして動的なユーザインターフェースはどのような場面で効果があるのか、まだ筆者には答えが思い浮かびませんが、使い方次第ではおもしろいアプリができそうです。

用途が広がったマルチタスキング

iOSのマルチタスク機能は、バックグラウンドで自由にプログラムを実行できるものではありません。バッテリーの消費量を抑えるため、音楽の再生、IP電話、位置情報の取得など、限定された機能にしぼってバックグラウンドでの動作を許可しています。

iOS 7でも基本的な考え方は変わりませんでしたが、バックグラウンドで動作する機能が2つ追加されました(**図8**)。いずれも使い方次第では利便性の高いアプリを開発できるでしょう。

1つは「Background fetch」と呼ばれるモードで、定期的にデータをダウンロードするような目的で使用され、SNSのようなアプリで事前に最新のデータをダウンロードする場合に利用できそうです。もう1つは「Remote Notification」というモードで、プッシュ通知によって実行するようなケースで使用します。

いずれも内部で独自の処理を実行するようなしくみではありませんので、OS Xのアプリのようにバックグラウンドでレンダリングさせたり、科学計算のような重い処理を実行させるような用途には使えません。WWDC直後の一部の報道では、iOS 7で完全なマルチタスクを実現したような表現がありましたが、残念ながら現実は違っていたようです。すべてのアプリが好き勝手にバックグラウンドで実行されれば、

▼図8　Xcode 5におけるマルチタスクモードの設定

いくら大きなバッテリーを積んだiPhoneでもたちまちなくなってしまうでしょう。自由度の高いプログラムの実行環境とバッテリーの消費量は、互いに相反する関係ですから、将来的にもiOSのマルチタスクに対する姿勢は変わらないと予想しています。

新しいフレームワークで新しい体験を提供しよう

これらのほかにもiOS 7で追加された興味深いフレームワークは数多くあります。高度なテキスト描画をサポートする「Text Kit」、近距離の無線通信を実現する「iBeacon」、3D操作へのアクセスが可能になった「Map Kit」など、プログラマの創造力をかき立てる技術がiOS 7の中に詰まっています。

また、iPhone 5sにはM7と呼ばれるモーションセンサー用のコプロセッサが内蔵されています。それに伴い、モーションセンサーの値を取得するためのCore Motionフレームワークには、カウントした歩数を得るためのAPIなどが実装されています。現在はiPhone 5sに限定されますが、歩数計やライフログのようなアプリが簡単に開発できる環境が用意されたことによって、アプリ開発の可能性がさらに拡がりました。

iOS 7の中に埋まっている宝のような技術をアプリ開発でいかに活用できるかは、プログラマの腕の見せどころと言えるでしょう。

●●●●○ 100%

ポイントまとめ

- Sprite Kitはゲームに限らず、その特性を生かしてグラフィック描画を必要とするあらゆる用途で活用しよう。
- UIKit Dynamicsを使って動的要素を加えた新しいユーザ体験を開発しよう。
- マルチタスキングは機能が限定されるが、使い方次第でユーザビリティを高める効果を発揮できる。
- 新しいハードウェア技術に応じたフレームワークに注目し、iPhoneならではの機能をアプリから引き出そう。

より生産性を高めた開発ツール

操作性の向上と進化したデバッグツール

統合開発環境のXcodeは、iOS 7の登場と同じタイミングでバージョン5になり、機能や操作性が大きく拡張されました。画面構成や基本的な操作体系は以前のバージョンから変わっていませんが、iOS 7のようにプロジェクトウィンドウはすっきり整理されました(**図9**)。機能の面ではデバッガやテストツールが拡充され、より生産性の高い開発環境をプログラマに提供しています。

ソースレベルデバッガでは、クイックルック機能により変数の内容と同様にUIImageの画像も見ることができるようになりました(**図10**)。また、デバッグゲージ機能により、アプリが使用するメモリ消費量やCPUの占有率も、Xcode上で確認できるようになりました(**図11**)。今までは別アプリ(Instruments)を起動して確認していたので、アプリを切り替えることなくプログラムの状態を知ることができるのはとても便利です。

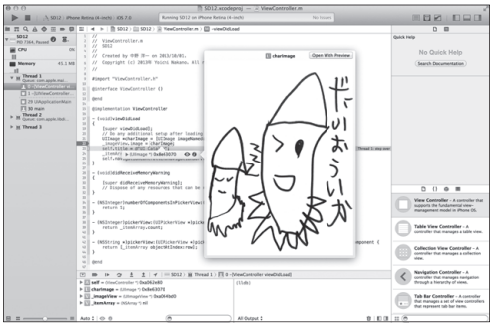
画面設計の機能では、iOS 6/7のユーザイン

ターフェースを設計段階で確認できるプレビューモードが追加されました(**図12**)。ビルドして実行しなくてもアプリの画面を確認できるので、効率よく過去のOSに対応したアプリを開発できます。

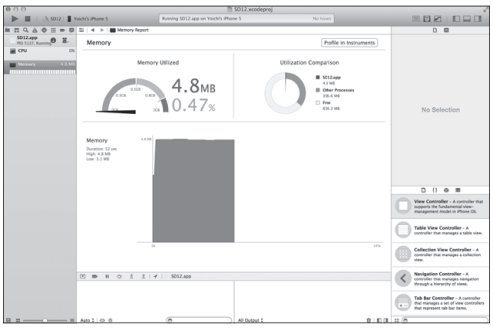
筆者はまだ試していませんが、OS X Serverと組み合わせてリモートでテスト環境を構築できるようです。Botと呼ばれるしくみでプログラムの解析や一連のテストを実行し、バグの発見を手助けするそうなので、ぜひ導入してみたいと思いました。

▼図9　Xcodeのプロジェクトウィンドウ

▼図10　デバッガのクイックルック機能



▼図11　デバッグゲージ

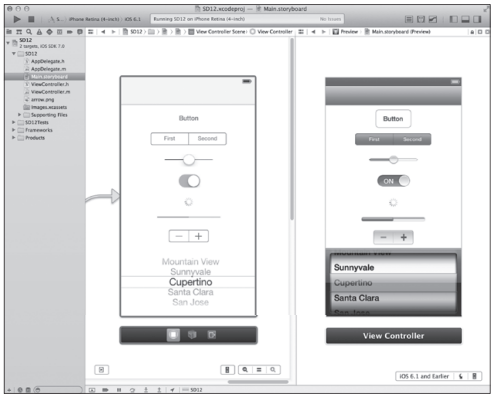


アカウント管理機能が便利に

そのほかにもプログラマにとって便利な機能が追加されています。すべてをご紹介できませんが、アカウント登録は複数のApple ID(デベロッパID)を持つ筆者にとって望んでいた機能です。環境設定であらかじめApple IDを登録しておけば、開発に必要なプロビジョニングプロファイルを、Appleの開発者サイトと同期してくれます(**図13**)。

以前のバージョンにも同様の機能がありましたが、複数のIDを登録できなかったり、うまく同期できないこともあって、あまり活用できていませんでした。プロビジョニングプロファイルは開発するアプリが増えると管理が大変なので、地味ではありますがうれしい機能です。

▼図12　プレビューモード



▼図13　アカウント機能

画像が一元管理できるイメージアセット

イメージアセットはXcode 5で追加された画像ファイルの新しい管理方法です。アプリを開発する際、画像ファイルは通常の解像度とRetinaディスプレイ用に倍解像度の2種類を用意するのが一般的です。倍解像度にはファイル名の後ろに「@2x」を付けるルールになっていて、画像の数が増えるとファイル管理に苦労します。

イメージアセットは、フォルダに入れた画像ファイルとJSONファイルで構成されています。Xcodeのエディタ上ではわかりやすくリストで表示されますので、アプリで使用する画像は常に整理された状態で管理できます。アプリのアイコンや起動画面もイメージアセットで管理されていますので、不足している画像も一目でわかるようになっています(**図14**)。

ポイントまとめ

- Xcode 5の豊富なデバッグ機能やプログラム解析機能を使って、効率よくアプリ開発を進めよう。
- アカウント機能を使えば複数のプロビジョニングファイルを包括的に管理できるので、プロジェクトの設定ミスを未然に防げる。
- イメージアセットを積極的に利用し、アプリで使用する画像を見通しよく管理しよう。

おわりに

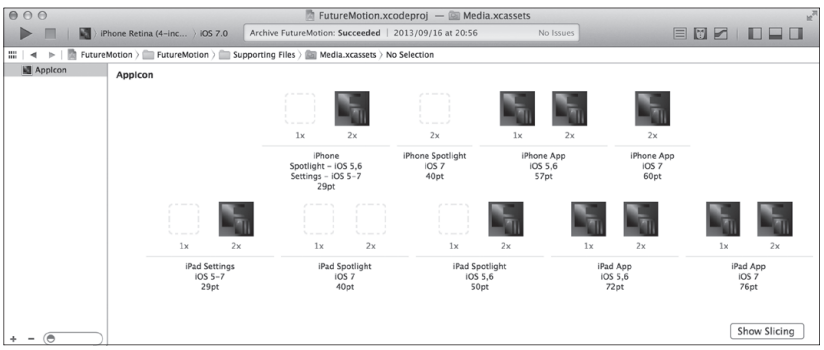
一歩抜け出したアプリを開発するには

すでにiOSのアプリはものすごい数がリリースされています。よほど魅力的な機能や価値がない限り、ユーザの目に触れることすらないくらいに、アプリは飽和状態にあります。玉石混交のiOSアプリ市場でキラリと光るアプリを開発するには、iOS 7の最新技術をいかに取り入れ、iPhoneやiPadの魅力を引き出すことに成功するかどうかが鍵を握るでしょう。また、iOS 7のフラットデザインによって、よりユーザインターフェースデザインの重要性が認められようとしていますので、画面の設計やユーザビリティに高いコストがかかる時代になったと言えます。

このように考えると、今までよりもアプリ開発のハードルが上がったように筆者は感じていますが、それだけビジネスチャンスも拡がったと前向きにとらえるべきかもしれません。また、Appleのエコシステム(iCloudやGame Centerなど)をうまく活用して、Macと連携したアプリを開発することで、iOSの限界を超えたサービスを生み出すことができるかもしれません。

いずれにせよ、引き続きスマートフォン市場を牽引していくAppleの製品と技術は、これからも目が離せないでしょう。SD

▼図14　イメージアセット



より速く、より莫大に、より高みへ！

新連載

サーバマシンの測り方

ベンチマークを極める実践テクニック

第1回　HDDやFlashのI/O性能を測る

本誌2013年7月号で、サーバのベンチマークについて入門記事を書かせていただいた藤城です。
入門記事とのことで、もしかしたらベンチマークのおもしろさが伝わらなかったかもしれません。
筆者がベンチマークを楽しく思う瞬間は、ベンチマークの結果からそのサーバやデバイスの速さを実感できたときです。
そこで、この連載では速さに焦点を当て、筆者が業務上でよく使うベンチマークツールを用い、その使い方や結果を紹介したいと思います。1回目では、ディスクのI/O性能についてもう少し掘り下げていきます。

Writer ㈱IDCフロンティア　ソリューションアーキテクト　藤城 拓哉（ふじしろ たくや）／Twitter@tafujish

ioping で I/Oレイテンシーを測る

IOPSはもう古い!? 今流行のI/Oレイテンシーとは

ディスク性能を見る主な指標として秒間転送量、IOPS、レイテンシーの3つがあります。

1つめの指標として、1秒あたりの転送量（Byte/sec）があります。これは大容量のデータを扱うときに参考になる数値です。たとえば、`hdparm -t`コマンドで簡単に計測でき、MB/sのような単位で結果が出力されますので、昔からよく使われてきました。

しかし、データベースなどのサーバでは、処理容量よりも処理数が重要になってきます。つまり、1つのI/Oリクエストのサイズ（ブロックサイズ）が小さくランダムアクセスのため、1秒あたりの処理数を見たほうがわかりやすいということです。これが2つめの指標のIOPS（Input/Output Per Second）です。そのため、昨今ではFlashドライブやストレージのI/O性能として、IOPSが使われることが多くなってきました。ちなみに、転送量とIOPSには次の関係性があります。

転送量B/s ÷ ブロックサイズByte ＝ IOPS

または、

IOPS × ブロックサイズByte ＝ 転送量B/s

たとえば、転送量が1MB/sでブロックサイズが4KBのとき、

1MB/s ÷ 4KB ＝ 256 IOPS

※1M=1024K

となります。

ベンチマークを実施しIOPSを測定したとき、そのスコアはシステムの限界に近い値です。しかし、実際のシステムを動かすときに限界ギリギリまで性能を出すことは少ないでしょう。

たとえば、ピーク時に100 IOPSになるデータベースサーバがあったとき、ローカルに接続されたSSD（1,000 IOPS）とネットワーク接続されたiSCSIストレージ（2,000 IOPS）で、どちらのほうが速く処理できるでしょうか。IOPSで考えるとiSCSIストレージのほうが高いですが、両方とも要件は満たしています。次に考えるポイントが、ローカルに接続されているよりネットワーク接続のほうが、ネットワーク通信分遅いのではないかということ。

そこで3つめの指標のレイテンシーです。I/Oリクエストしてから処理が完了し、データが返ってくるまでの時間です。単純に時間なので、μsec（マイクロ秒）やmsec（ミリ秒）という単位となりこの値が小さいほど高速ということになります。このレイテンシーは、性能が良いかどうかというだけでなく、時間内にI/Oが返って来ているかという安定性の指標としても使えます。

ioping

ではI/Oのレイテンシーをどう見るかですが、**ioping**がお勧めです。その名のとおり、**ping**の感覚でI/Oレイテンシーが簡単に計測でき、シンプルなベンチマークも実行できます。

インストールは簡単で**図1**のとおりです。基本的な使い方としては、

```
ioping オプション 対象
```

となります。

```
ioping -h
```

でヘルプが出てきます。

たとえばI/Oレイテンシーを計測する場合は、

```
ioping -c 4 -D /tmp
```

指定した/tmpディレクトリから4KB(デフォルト値[-s])でデータを読み出したときの時間が-cで指定した回数分1秒間隔(デフォルト値[-i])で繰り返されます。複数回繰り返すときは1MB(デフォルト値[-S])の範囲でアクセスされます。最後にpingのように最小値や平均値といった結果も計算してくれます(**図2**)。併せてIOPSやMB/sの結果も表示されますが1秒間隔でのI/Oのため、ベンチマークを目的とした値としては不十分です。そのため、IOPSを計測したい場合は後述の-Rオプションを参考にしてください。ちなみに対象は、ディレクトリのほか、ファイルの指定や/dev/sdaのようにデバイスを直接指定することもできます。また、-Dを付けることで、常にディスクへ直接読みにいきますので、ディスクやストレージの値を見たいときはこのオプションを付けるとキャッシュを利用しないので指定しましょう。

これを定期的に繰り返し、ディスクやストレージの安定性をモニタリングするのも良いでしょう。**図3**はストレージに負荷がかかっている状態で計測したレイテンシーです。負荷が大きくなるとレイテンシーも大きくなりました。数百msecかかると遅く感じる、数秒かかるとサービスに影響が出るといった目安にもなります。

また、時間指定(-w)、リクエスト間隔(-i)、リクエストサイズ(-s)、リクエストする範囲のサイズ(-S)を指定できるのでちょっとしたベンチマークにも利用できますし、次のようにベンチマークとして簡単に利用するオプションもあります。

▼図1　ioping(https://code.google.com/p/ioping/)のインストール

```
$ sudo yum -y install libaio-devel make gcc
$ curl https://ioping.googlecode.com/files/ioping-0.7.tar.gz | tar zx
$ cd ioping-0.7
$ make
$ sudo make install
```

▼図2　ioping(複数回実行し、平均値を計算できる)

```
$ ioping -c 4 -D /tmp
4.0 kb from /tmp (ext4 /dev/mapper/VolGroup-lv_root): request=1 time=84 us
4.0 kb from /tmp (ext4 /dev/mapper/VolGroup-lv_root): request=2 time=80 us
4.0 kb from /tmp (ext4 /dev/mapper/VolGroup-lv_root): request=3 time=77 us
4.0 kb from /tmp (ext4 /dev/mapper/VolGroup-lv_root): request=4 time=79 us
                                                          ↑ レイテンシー

--- /tmp (ext4 /dev/mapper/VolGroup-lv_root) ioping statistics ---
4 requests completed in 3.0 s, 12.5 k iops, 48.8 mb/s
min/avg/max/mdev = 77 us / 80 us / 84 us / 2 us
                   ↑ レイテンシーの平均値
```

```
ioping -R 対象
```

でIOPS、

```
ioping -RL 対象
```

で256KBでアクセスされるのでMB/sの結果を見るのに適しています。

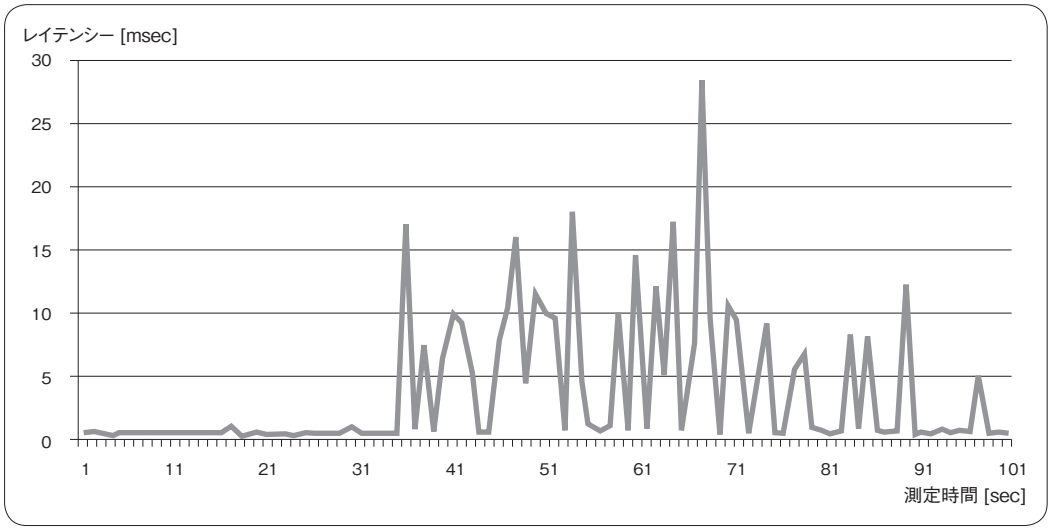
HDD、ioDrive、メモリ、一番速いのは?

では、実際に**ioping**を使ってレイテンシーを計測し結果を比較します。ここでは、一般的なディスク構成ということで、SAS 15000rpmのHDDと、このHDDを4本でRAID10を組んだRAIDアレイ(キャッシュあり)、低レイテンシーが売りであるFusion-io ioDrive2 365GB MLC、最速だろうということでメモリをtmpfsにて計測し比較します。また、計測元のサーバとは別にNFSサーバを用意し、計測元のサーバにてNFSマウントしたものとも比較します。なおNFSサーバ内では先述のioDrive2をマウントしています。

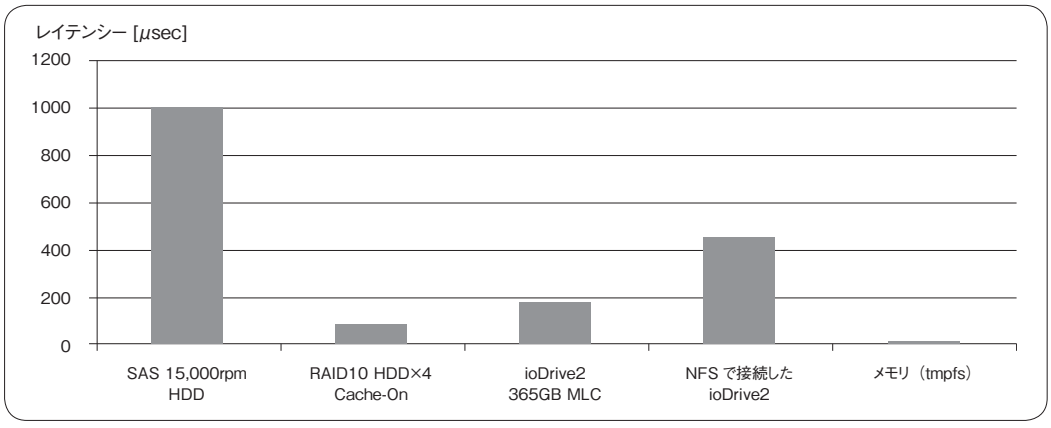
```
ioping -c 100 -D 対象
※tmpfsでは-Dオプションを外しています
```

を実行したレイテンシーの平均の値は**図4**です。

▼図3　ストレージの負荷状態を計測



▼図4　フラッシュ、ハードディスク、それぞれの速度比較



メモリ(4 μ sec)、RAIDアレイ(80 μ sec)、ioDrive2(174 μ sec)、NFS(445 μ sec)、HDD(1000 μ sec)の順で速い結果となりました。やはり一番レイテンシーが小さいのはメモリで、レイテンシーの小ささに定評があるioDriveは、HDDと比べて圧倒的に高速でした。

ioDrive2よりもRAIDアレイのほうがレイテンシーは小さい結果となりましたが、これはRAIDコントローラのキャッシュすなわちメモリが応答しているため高速でした。たとえば、RAIDアレイにてキャッシュを無効にするとHDDと同じくらいのレイテンシーとなりました。

次に同じioDrive2でも、ローカルに接続されているioDrive2のほうがNFS接続先のioDrive2よりレイテンシーが小さい結果となりました。計測元のサーバとNFSサーバは同一ネットワークにつながっており、通信のレイテンシーは百数十 μ sec程度でした。併せてNFSの処理にかかるオーバーヘッドもあるため、NFS接続のioDrive2のほうがレイテンシーは大きい結果となりました。

以上からI/Oレイテンシーの速さは、ドライブの種類や性能だけでなく、RAID(SATA/SAS)コントローラの性能やキャッシュの有無やネットワーク接続といったさまざまな要因が考えられます。

fioで継続的にI/O性能を測る

ioping は簡単に扱えましたが、高機能なfio(http://freecode.com/projects/fio)でも同じような計測をさせることができます。

```
ioping -c 60 -D /mnt/fio
```

と実行したときと同じようにI/Oレイテンシー計測ができるジョブファイルが**リスト1**で、テキスト出力された実行結果が**図5**です。レイテンシー計測以外にも使える便利なオプションを紹介します。

rate_iopsで指定したIOPSまでしかI/Oを出

▼リスト1　fioのジョブファイル

```
[ioping-like]
rw=read                  ← iopingはデフォルト読み込みI/O
bs=4k                    ← iopingはデフォルト4KB(-s)
size=1m                  ← iopingはデフォルト1MB(-S)
runtime=60
direct=1                 ← ioping -D に相当
directory=/mnt/fio       ← 対象
rate_iops=1              ← iopingはデフォルト1秒間に1I/Oリクエスト
write_iops_log=iopinglike
write_lat_log=iopinglike ← レイテンシーのログを出力
iopsavgtime=1000
```

▼図5　fioによるレイテンシー計測の実行結果

```
$ fio iopinglike.fio ← fio実行(ジョブファイルiopinglike.fio)のとき
……(省略)……
$ cat iopinglike_lat.log
0, 131, 0, 4096
1000, 131, 0, 4096
2000, 102, 0, 4096
3000, 129, 0, 4096
4000, 128, 0, 4096
       ↑
  レイテンシー結果(μ秒)
……(省略)……
```

しません。たとえば、100 IOPSだけ負荷をかけ続けたいといった場合や、100 IOPSの負荷がかかっている状態のレイテンシーを計測したい場合に使えます。

write_iops_logは、iopsavgtime(ミリ秒)で指定した間隔でIOPSを算出しテキストに出力します。結果は、時間(ミリ秒)、IOPS、Read(0)かWrite(1)か、ブロックサイズ(Byte)の順です。IOPSのベンチマーク中に、IOPSの値が大きく変動することがあればこのログからグラフ化すると動きが見やすくなります。ストレージが安定していないと気づいたり、何らかの負荷がかかったタイミングがわかったり、キャッシュから溢れたタイミングがわかったりするための一助となります。

write_lat_logもwrite_iops_log同様で、結果のIOPSのところがレイテンシーになります。latのほかにslat(submission latency)とclat(completion latency)のログファイルも出力されますが、アプリケーションとカーネルのやりとりの時間と、ドライブにて処理を完了する時間のため、最終的にはこれらの合算であるlatの値を参照すれば十分です。

OS上からIOPSを見る

ここまで「IOPS」を何度も使ってきました。また、IaaSなどのクラウドサービスやFlashドライブのカタログスペックにもIOPSをよく見かけるようになりました。ところで、自分のシステムが何IOPS出ているかご存じでしょうか。その計測方法は簡単で、次のとおり**iostat**で計測できます。

```
iostat -mx 1 60
```

-xオプションを付け、1秒ごとに計測することでIOPSを確認できます。ここでは60回繰り返し実行しています。また、-mを付けると転送量(MB/s)も出てきます。「r/s」が読み込みのIOPS、「w/s」が書き込みのIOPSです。実行例は**図6**です。

実際のシステムのIOPSを確認したいときや、ベンチマーク実行中にiostatを実行し本当にベンチマークツールが言うIOPSが出ているかの確認に使えます。

次回予告

今回はiopingとfioを使ってI/Oレイテンシーを計測しました。ディスクやストレージのベンチマークの際には、IOPSと併せてI/Oレイテンシーも計測されてはいかがでしょうか。

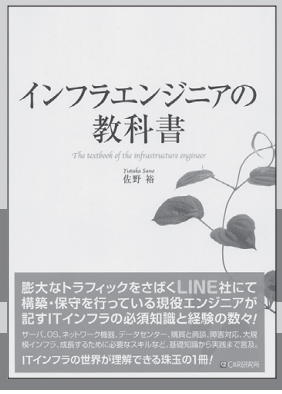
レイテンシーの影響を抑える方法の1つに並列でのアクセスがあります。次回はディスクに並列アクセスしやすいデータベースをベンチマークのテーマに、Fusion-io ioDriveの性能について掘り下げていきます。SD

▼図6　iostatによるIOPSの観測

```
avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           0.03    0.00    0.19    2.88    0.00   96.90

Device:         rrqm/s   wrqm/s     r/s     w/s    rMB/s    wMB/s avgrq-sz avgqu-sz
await  svctm  %util
sda               0.00     0.00    0.00    0.00     0.00     0.00     0.00     0.00     0.00
0.00   0.00
sdb               0.00     0.00 2779.00 2769.00    10.86    10.82     8.00    31.97     5.80
0.18 100.00
```

BOOK no.1 インフラエンジニアの教科書

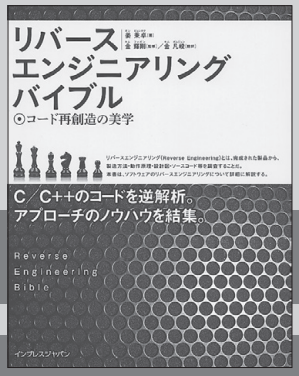
佐野 裕 【著】
A5判、 184ページ／価格＝2,100円 （税込） ／発行＝シーアンドアール研究所
ISBN＝978-4-86354-133-7



インフラエンジニアを目指す人の羅針盤となる1冊。本書はさらっと読める分量ながら仕事の全体像が見渡せる（なんと将来のことまで考えてくれている！）ことに加え、一歩踏み込んで実務に即した視点を与えてくれることが最大の魅力。インフラエンジニアには「技術力」と「責任感」に加え、めまぐるしく変化する状況に対応するための「情報収集力」と「決断力」を備える必要があるという主張のもと、必須知識としての専門技術の解説をしつつ、それらの技術に対してインフラエンジニアならどういったことに着目したら良いのかが、大人気サービスのLINEを支えている現役エンジニアの豊富な経験から語られている。新人教育担当者にもぜひ。

BOOK no.2 リバースエンジニアリングバイブル
コード再創造の美学

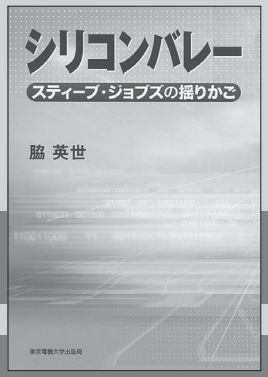
姜 秉卓 【著】 ／金 凡峻 【訳】 ／金 輝剛 【監修】
B5変形判、 416ページ／価格＝4,200円 （税込） ／発行＝インプレスジャパン
ISBN＝978-4-8443-3479-8



その名のとおりリバースエンジニアリングの手法を解説する。基本とも言えるアセンブラの読み方の解説から始まり、C、C++における基本文法がアセンブラではどう表現されるかという視点で、もとのソースコードを把握する手法を手ほどきする。内容は複雑で高度だが、文体は軽く読みやすい。よくあるパターンのリバーシングについて学んだら、次はリバーシングを防ぐ手法を学ぶ。本書が徹底していると感じるのは、その防御を回避する方法まで解説しているところだ。「防御したいなら、攻撃を研究しろ」というのはセキュリティにおける基本姿勢とのことだが、本書もその姿勢に立ち、高度なリバーシング手法を惜しげもなく示してくれる。

BOOK no.3 シリコンバレー
スティーブ・ジョブズの掴りかご

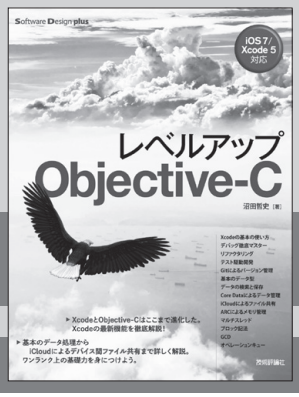
脇 英世 【著】
四六判、 528ページ／価格＝2,625円 （税込） ／発行＝東京電機大学出版局
ISBN＝978-4-501-55210-7



シリコンバレーの歴史を書いた1冊。スタンフォード大学創立からインテル社の創業までのことが書かれている。真空管を用いたアマチュア無線技術が、やがてトランジスタやICに発展し、用途も無線からコンピュータなどに変わっていく過程が描かれる。その成長の背景にあったのは、「互いに競争しつつも、成果を情報共有しながら開発する」というアマチュア無線家たちの独自の文化と、軍用電子機器の需要だ。多くの技術者がそんな時代背景の中で、しのぎを削って研究、開発を行う様子がよく描かれている。この地は今もトップエンジニアが集い、起業が盛んだ。本書はその文化を育んだ一時代を知ることができる貴重な資料と言える。

BOOK no.4 Software Design plus シリーズ
レベルアップObjective-C

沼田 哲史 【著】
B5変形判、 360ページ／価格＝3,360円 （税込） ／発行＝技術評論社
ISBN＝978-4-7741-6076-4



本誌2011年5月号から2012年の4月号までの連載「極めるObjective-C」から1年半。本書は同著者によるiOS 7とXcode 5を使ったObjective-Cの中級者向けのレベルアップ指南書である。

Xcode 5で追加されたナビゲータの解説や、ユニットテスト、Gitによるバージョン管理などの実践的な使い方や、現代的なオブジェクト指向のプログラミングについても解説されている。

また、Core DataやiCloud、ARC（Automatic Reference Counting）によるメモリ管理、マルチスレッドプログラミングなど、アプリ開発のための最新の技術を身につけることができる。

分散データベース「未来工房」

第6回

Riakアプリケーションとしての Riak CS(1)

――基本機能から内部構造までじっくり解説

Writer 上西 康太(うえにし こうた)　Bashoジャパン株式会社　kota@basho.com

SQLを持たないデータベースでのアプリケーション設計は、基本的には個々のデータベースの特性を考慮しなければならず、一般的なデータモデリングの考え方をそのまま適用できないどころか、トランザクションを使えない場合もある。Riakも同様で、アプリケーションを設計するときの方法はこれまでのRDBMSとはまったく異なる。そこで、本稿ではいくつかの典型的なパターンを、実際のアプリケーションを用いて2回に分けて解説する。
注)本稿は、筆者の意向により常体で表記している。

はじめに

RiakはSQLのインターフェースとデータモデル、トランザクションの機能を持っていないために、通常のデータモデリングや設計技法がそのまま適用できない。具体的には、トランザクションがなくてもデータが壊れないように、また効率的なデータアクセスができるように、Riakへのアクセスパターンやデータ構造を、これまでとは異なる考え方で設計しなければならない。RDBMSに慣れている場合には少し敷居が高いが、その対価として、スケーラビリティや可用性などRiakを使うことのメリットを得ることができる。

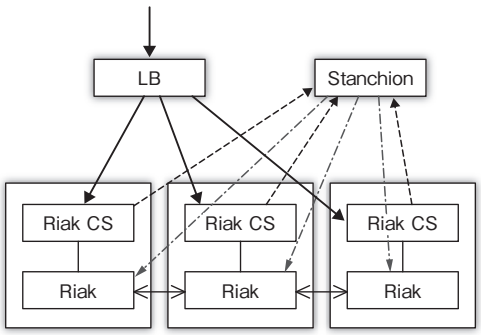
一方、本誌9月号の記事で説明したように、Riak CS(Cloud Storage)は、下回りのデータストアにRiakを使用している(だからこそRiak CSにも、Riakとほぼ同様の可用性、拡張性、運用性がある)。少し視点を変えると、Riak CSは単なるHTTPのAPIサーバであり、Riakをデータベースにした Web3層のシステムそのものである(**図1**)。つまり、Riak CSはRiakというデータベースを使ったアプリケーションそのものであり、このデータ構造や設計を調べていけば、通常のアプリケーションを設計する際に参考になるものが、たくさんあることが期待される。ここでは、その具体例をいくつか取り上げて、「Riakでどうやったらアプリケーションが設計できるの?」という疑問への解答としたい。

Riak CSの基本機能とデータモデル

Riak CSは「クラウドストレージ」と呼ばれる分野のソフトウェアで、コモディティサーバを複数台並べてスケールアウトできるストレージであり、REST APIを持っている。詳しくは9月号の記事を参照するとよいが、AWS S3互換のフロントAPI[注1]と、いくつかの独自の管理

注1) 1.4から、Keystone認証をはじめとするSwift互換のAPIも持っている。

▼図1　Riak CSのプロセス構成



RiakとRiak CSのプロセスが同じマシンに同居しているが、別マシンで動作させることもできる。また、Stanchionという管理プロセスがいるがロードバランサ、アプリケーションサーバ、データベースサーバが3層になっている。

APIを持っている。HTTPのインターフェースを持つので、そのまま静的コンテンツをホストすることもできるし、大きなバックアップのデータをまとめて保存することもできる。容量が不足してくればサーバを追加するだけでスケールアウトできるし、容量が余っているようであればサーバを減らしてコストを削減できる。

Riak CSに限らず、クラウドストレージと呼ばれる類の製品は次のような特徴や機能を持っている。

・MBからGBオーダーの大きなオブジェクトの保存・読み出し・削除
・マルチテナントなデータ空間とユーザ管理
・ユーザごとにも、ディレクトリ構造を模した名前空間
・ユーザの利用統計

また、こういったものを実現するためには、次のようなモデルによる構成が一般的で、Riak CSでもそうしている(**図2**)。

・オブジェクト
・ユーザ
・バケット
・アクセス集計
・ストレージ使用量の履歴

Riak CSのAPIはAWSのS3とほぼ同じ機能なので、ユーザに見えるデータモデルは基本的には同一だ。運用側の機能として、ユーザ管理、アクセス集計とストレージ使用量の統計はRiak CS独自のものだ。

最初は通常の設計と同じで、これらひとつひとつにRiakのバケットを用意する[注2]。

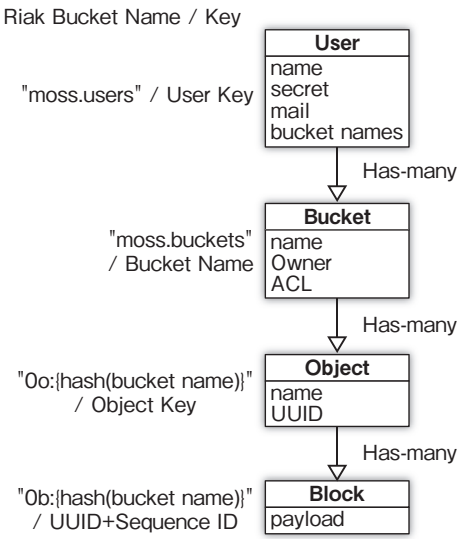
まず、**ユーザ**は個々のユーザのアカウントである。ここには、ユーザの認証情報、メールアドレス、ユーザ名などに加えて、ユーザが保持しているバケットの一覧もここに保持しておく。

次に、**バケット**は、個々のユーザが管理する、複数のオブジェクトをまとめて管理する単位である。この単位でアクセス権限などを管理する。バケットの名前はグローバルに一意でなければならず、別のユーザが同じ名前のバケットをあとから作成することはできない。

オブジェクトは、個々のユーザが保存したいファイルのことを指す。Riak CSでは最大で5TBまでのサイズのオブジェクトをアップロードできる。また、オブジェクトの名前で、パス区切りの文字(通常は'/')を指定して模擬的にディレクトリ階層を構成することもできる。ユーザに理解しやすい「バケットにオブジェクトを格納する」という考え方を実現するために、オブジェクトは、必ず1つのバケットに属していなければならない。

アクセス統計と、ストレージ使用量の履歴については、ここでは割愛する。

▼図2　Riak CSの要素間の関係



左側にはRiak上でどのように格納されるかを示した。

注2) Riak CSのバケットと、Riakのバケットは、どちらも同じBucketという単語であるが、明確に区別する言い回しは決まっていない。本稿ではなるべく区別しやすいように注意するが、呼称が明確に区別される用法があるわけではない。文脈によって変わるのでWeb上の情報やメーリングリストでは慎重に読み進める必要がある。

ユーザ作成：Stanchionでのシリアライズ

なぜシリアライズが必要なのか?

まず、ユーザの作成や操作などについて解説する。ユーザ作成は、RiakなどのCAS(Compare-And-Swap：条件付き更新要求)を持たないデータストアにはもっとも難しい処理である。別の人物が同じユーザ名でアカウントを同時に作成しようとした際に、必ずどちらかを成功させてどちらかを失敗させなければならない。トランザクションを持ったデータベースであれば、ユーザ名のカラムにユニーク特性を持たせるだけでよい。

ところが、Riakを用いた場合には、単なるPUTの処理だと、あとからRiakに到着した作成のリクエストで簡単に上書きを許してしまう。したがって、先にアカウントを作成できたほうのユーザのアカウントが無効になってしまう[注3]。

RiakにはCASのような、いったんデータを読んでから安全な状態でデータを更新するような操作はないため[注4]、何らかの排他機構を用いて、排他的にユーザアカウントを作成・更新する手段がなければ、ユーザ管理を実現することはできない。

だが、ここでよく考えると、アカウントの発行は**頻繁に起きる操作ではない**。アカウントを発行したあとに行われるオブジェクトの操作のほうがはるかに多く、実際に利用する場合に、ここをとくにスケールアウトさせる必要はない。そこで、Riak CSではこういった排他的な処理を担当するプロセスをStanchionという名前で別途用意することとした[注5]。Stanchionの内部では、ネットワークI/Oやプロトコルパーサーなどは並列で処理されるものの、最終的なデータ構造の操作は必ず同時に1つしか実行されないため、擬似的にロックサーバのような機能を果たしていることになる(**図3**)。

トランザクションなど必要なかった

Stanchionのアイデアの優れているところは、よくあるMaster-Slaveの構成と異なり、状態をいっさい保持せず、ステートレスに動作する点である。もしもMaster-Slaveのような構成でデータを保存してレプリケーションを行った場合、それに伴い運用が非常に複雑になる。StanchionはRiakに対するデータ更新を順番に並べるだけなので、Stanchionプロセスのいるマシンやディスクが故障しても、問題を解決して再び起動するだけでよく、データのリカバリなどの操作は不要だ。ひとたびStanchionが起動すれば何もなかったかのように処理を再開できる。また、Stanchionが落ちている間も、アカウント作成・更新(と、バケット作成・更新・削除)以外のRiak CSの機能は通常どおり利用できる。

こうすることによって、Stanchionに関して運用者が気を付けなければならないことをたった2つに集約できる。それは、1)すべてのRiak CSのプロセスが必ず1つのStanchionにアクセスする、2)何らかの理由でStanchionプロセスが失われたら、1)の範囲内でStanchionを(再)起動する、の2つだけだ。ディスクがどうなったとか、データがなくなったとかそういった考慮はRiakに関するものだけでよい。

運用しやすさとのバランス

本来なら、サーバの種類を増やすのは運用上

注3) バケットも同様で、バケット名がグローバルにユニークでなければならないという制約を守るために、同様にStanchionを利用するしくみになっている。

注4) これがRiakがAP(可用性と分断耐性)のデータベースだと言われるゆえんである。これを分散データベースで同期的かつ安全に実行する、よい実装はあまりない。Riakの次期リリースにはこういった機能が入るかもしれないと聞いているが、真偽のほどは定かではない。

注5) ソースコードはhttps://github.com/basho/stanchionで公開している。Stanchionとは、英語で「梁柱」という意味だが、切符売り場などで行列を管理するための柵を支える柱のことを指す。つまり、Riak CSに対するリクエストを並ばせて順番に受け付ける。

は非常に好ましくないことであるが、ユーザ管理を実現するためのトランザクショナルなデータ更新のしくみをRiakに作り込むのはかなりのコストがかかるため、Stanchionに依存する機能を最小限にしてようやくサーバを1種類増やした。しかしサーバの種類はなるべく減らしたい。そこで、Bashoの開発者たちも、最終的には、StanchionをなくしてRiak CSとRiakの2種類だけで動作するようなしくみにしたいと考えており、実際にそういった分散合意プロトコルのプロトタイプはいくつもつくっているので、ご期待いただきたい。

具体的な実装を知りたい場合は、ユーザ情報は`moss.users`という名前のRiakのバケットに格納されるので、そこを操作するRiak CSのソースコードとStanchionのソースコードを追うとよいだろう。ユーザ情報の内容はRiak CSのレポジトリの`include/riak_cs.hrl`というファイルの`#rcs_user_v2{}`というレコードにあるので、ぜひご覧いただきたい。Erlangの場合は1つの軽量プロセスに処理を集約することによって処理のシリアライズを実現できるが、その他の言語であれば、Mutexなどの排他制御機構を用いるのが一般的だろう。

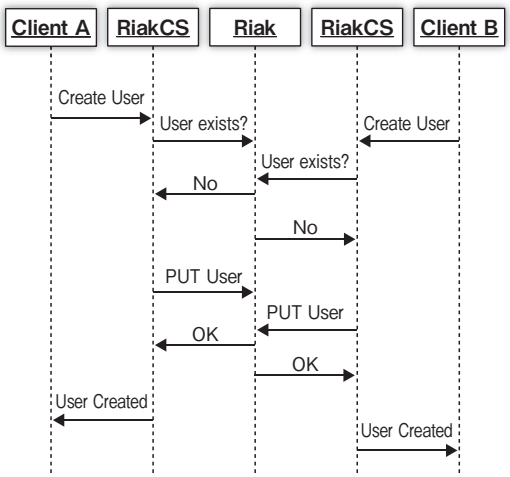
ともあれ、ある種のシステムでは、データを読んでからの更新を排他的に行う処理が必要だがそこまで性能が要求されないことがある。その場合は、Stanchionのように処理をシリアライズするしくみを使うとよいだろう。Stanchionのようにすべての更新が一ヵ所で整列されてしまって性能が足りない場合には、ZooKeeperの分散ロックを利用するのもよい。こちらであればもっと性能がでるはずなので、世間のたいていのケースはカバーできるだろう。

マニフェストの安全な更新：オブジェクトの作成とアップロード

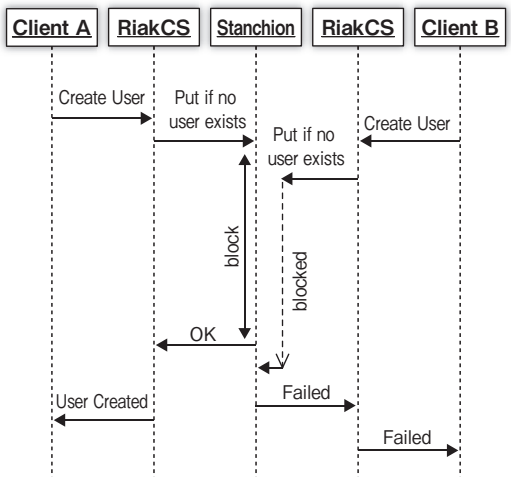
オブジェクトは、Riak CSの内部でマニフェストと呼ばれるメタデータと、1MBごとに分割されたブロックから構成され、それぞれ別のバケットに保存される。メタデータだけが必要なAPIは、マニフェストだけを取り出せばよい。

Riak CSは、S3と同様、一度のHTTPリク

▼図3　Riak CSでのユーザ作成のシーケンス



(a)Stanchionなしで直接Riakを更新してユーザを作成する設計の場合：ユーザのキーがRiak上にあるかないかを確認してからユーザを作成するまでの間に、他のクライアントからもユーザの有無の確認が入った場合、あとからユーザを作成したほう(Client B)が、先にユーザの作成に成功したほう(Client A)を上書きし、無効化してしまう。



(b) Stanchionを経由してユーザを安全に作成する場合(実際の動作)：Stanchion内部の排他機構(Erlangの軽量プロセス)が、あとからユーザ作成をリクエストしたほう(Client B)のリクエストをブロックしておくので、先に成功したほう(Client A)のデータを破壊しない。Client Bは失敗のレスポンスを得た後にリトライすればよい。

エストでアップロードできる最大サイズは5GBである。オブジェクトは最大で5TBまでをサポートしているが、そのサイズをアップロードする際にはマルチパートというAPIを用いて、複数回のHTTPリクエストに分けてアップロードする必要がある。もっとも、5GBものデータを一度のHTTPリクエストでアップロードするのも現実的ではないため、通常は5〜50MB程度の単位でアップロードする。

とはいえ、これだけインターネット回線が高帯域になった時代でも、数MBのデータを送信するにはいくらか時間がかかる。Riak CSの内部ではデータを受け取りながら1MBのチャンクに分割し、Riakに保存するため、データをクライアントから受信しながらRiakに書き込むように実装している。もし、Riakに書き込んでいる途中に、クライアントからの送信が途絶えたり、リクエストを受け付けているRiak CSのプロセスがハードウェア故障などで落ちてしまった場合に、中途半端なデータが見えないようにしなければならない。つまり、オブジェクトのアップロードの処理をユーザに対してアトミック[注6]になっているように見せなければならない。また、ディスク容量がリークしないために、アップロード途中のデータは最終的に回収されなければならない。

一方で、Riak のAPIはプリミティブなものが多く、そういったことを簡単には実現できない。マニフェストと複数のチャンクを同時にアトミックに更新するためには、分散トランザクションが本来なら必要なところだが、Siblingsと呼ばれるしくみと、巧妙なデータ構造の設計でこれを擬似的に実現している。

マニフェストの状態遷移

これを可能にしているのが、Riakが持っている`allow_mult`という設定項目だ。これによって、Riak内で必要なバージョンをすべて保持することができるようになる。さらに、データを読み込む際にクライアント側のロジックで競合を解決して必要な形式に変換する。これが、トランザクションがないデータベースでデータを失わずに安全にデータを更新していくための基本的な考え方だ。

ソースコードから読み解くのはかなりの根気が必要だが、Riak CSもこの考え方を採用している。まず、マニフェストのライフサイクルを状態遷移とともに定義する。**表1**にあるとおり、状態を定義しておく。

オブジェクト作成〜削除までの流れ

このように状態を定義しておくことにより、アップロード中、削除済み、物理削除待ちなどの状態を管理できる。本来であれば、状態遷移はトランザクションを使って「ある状態でこの条件がそろったら次の状態へ遷移する」といった、いったん読んでから更新する処理を安全に行わなければならない。また、Riakにはトランザクションがないため、同じオブジェクトに対する操作が並列に実行された場合でもデータ構造が壊れないようにしなければならない。たとえば、オブジェクトを削除する操作とオブジェクトを作成する操作が同時に実行された場合に、状態を決定しなければならない。これらの状態を安全に更新するのではなく、1つのマニフェストに対して複数のバージョンが存在したときにそれを解決するロジックを提供すること

▼表1　Riak CSのマニフェストの状態一覧

状態	意味
none	マニフェスト未作成
writing	オブジェクトをアップロード中
active	アップロード完了
pending delete	論理削除済み
scheduled delete	GC待ち状態
deleted	マニフェスト削除完了（※）

（※）実際にはnoneとdeletedは何もない状態なので、区別できない。

注6） データの更新が成功しているか、すべて失敗しているかのどちらかになっていること。

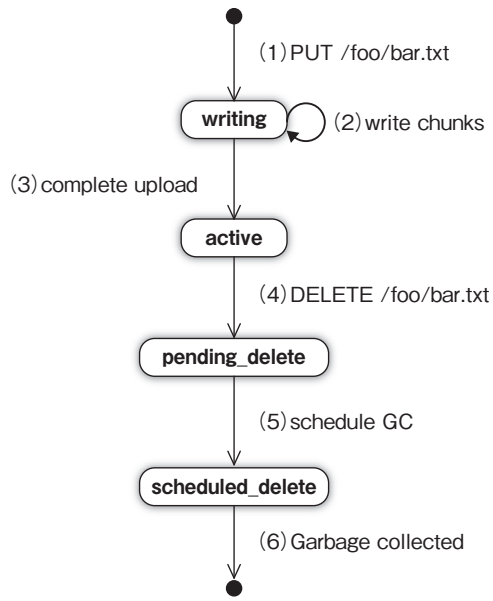
によって、オブジェクトの更新の並行制御を行う。

実際のオブジェクトの状態遷移の流れは**図4**のようになる。

(1)オブジェクトの作成開始時に、UUIDとともにマニフェストを作成し、writing状態でRiakへPUTする
(2)writing状態でマニフェストが作成できたら、個々のチャンクをRiakに書き込んでいく。途中でユーザからのデータが途絶えた場合などはこのまま放置してよい
(3)すべてのチャンクの書き込みが成功したら、active状態のマニフェストをあらためてPUTする[注7]。このときRiak上には、同じUUIDについてwriting 状態のマニフェストとactive状態のマニフェストが併存していることになる。これは後述する読み込み時の解決によって、最終的にactive なものが得られるようになる

注7) このUUIDで同じ名前のオブジェクトの複数のバージョンを区別する。

▼図4　オブジェクトの状態遷移の流れ



(4)削除するときは、pending_delete状態のマニフェストをあらためてPUTする。これも読み込み時の解決によって、pending_deleteが優先されるようになる
(5)pending_delete状態に変更した直後に、GCバケットにもエントリを追加し、scheduled_delete状態に変更する
(6)GCバケットに入れておくことにより、定期的にGCが走査しRiakでの削除操作が行われる

オブジェクト読み出し時の競合の解決

このように、とりあえずすべての状態のマニフェストを保存しておく。最終的にあるべき状態を決定するのは、オブジェクトを利用する(読み出し)時である。実際にクライアントからGETなどのリクエストがきたときには、`writing`、`active`、`pending_delete`、`scheduled_delete`などのマニフェストのSiblingsがまとめて得られることになる。基本的にはUUIDごとにタイムスタンプが新しいものが優先されるが、最新のものがwritingであった場合は、直近の`active`、`pending_delete`、`scheduled_delete`のものが選ばれる。activeであれば、オブジェクトは存在するのでその情報を使ってオブジェクトのデータを返却したり、削除したりする。他のものであれば、最後に削除されているはずなので、オブジェクトは存在しないことになり404がクライアントに返る。詳しいロジックはRiak CSの設計Wiki[注8][注9]に書かれているので確認するとよいだろう。

図5の例では、あるオブジェクトについて4種類のUUIDを持ったマニフェストが存在しているが、aaaのものはGC済み、bbbのものは

注8) https://github.com/basho/riak_cs/wiki/Object-Chunking-and-Garbage-Collection
注9) https://github.com/basho/riak_cs/wiki/Manifest-Sibling-Resolution

削除待ち状態である。cccのものはwriting状態でとまっており、どこかで失敗をしたままになっているだろう。この中でdddのものはactive状態になっており、かつタイムスタンプがもっとも新しいdddが正しいオブジェクトと判断され、リクエストに対してはUUID=dddのものが返る。

一方向の状態遷移(状態遷移図にループがない)であれば、sblingsの解決はこのように自明である。

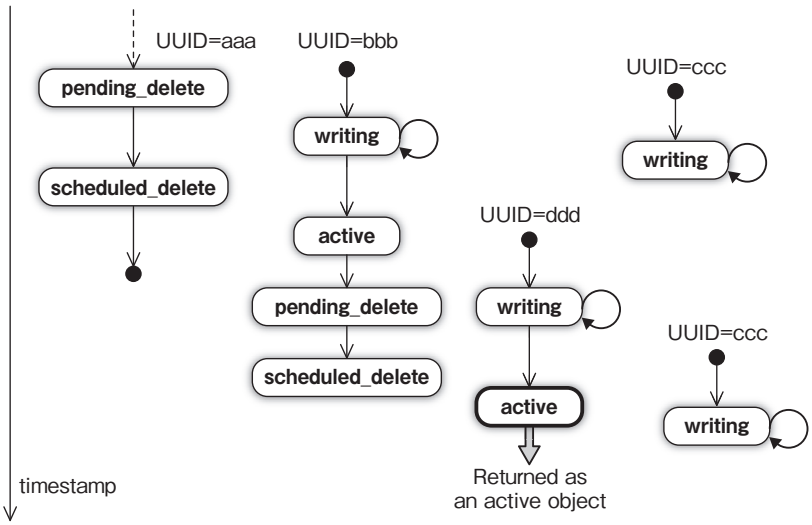
まとめ

ここまで、Riakのアプリケーションとしての Riak CSのデータ管理のしくみや考え方を、おもに更新系の操作について解説してきた。1つめは、排他的な一貫性がどうしても必要な、ユーザやバケットの操作を、専任の1つだけのプロセスを用意して排他的に管理できるようにする方法。これは、読者のアプリケーションにすぐにも適用できるタイプのものだと思う。いったんこの手の設計をしてしまえばスケーラビリティで悩むことはもうないだろう。2つめは、オブジェクトのアップロードを例にして、長時間かかる処理を安全に管理・変更する方法。最後のデータ削除のところがまだ残っているが、次回の読み込み関連の設計方法と合わせて、持ち越しとしたい。こちらは、読者のアプリケーションにすぐに適用できるものではないが、安全性を保証しながら状態をうまく遷移させていく方法は複数のバケットを安全に更新したい場合にアイデアを応用できるだろうと思う。

Riakを用いたアプリケーションでとくに重要なのは、ノードの故障を考慮に入れること、何をスケールアウトさせたいかを慎重に選ぶことである。Riak CSではオブジェクトに関する操作をスケールアウトさせる対象として慎重に設計してあるので、どのタイミングでノードが故障してもデータが壊れない、なくならない、消し漏れて残らないかどうかを、ぜひ検証してほしい。

次号では、本稿に収まらなかった、Riak CSにおけるオブジェクトの一覧表示や、インデックス(RDBでいうリレーションや外部キーに近い考え方)を解説する。SD

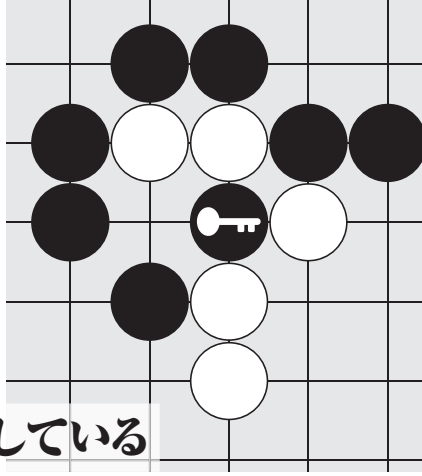
▼図5　並列に作成・削除されたマニフェストの選択条件



この状況であればUUID=dddのものが選ばれる。

セキュリティ実践の基本定石

すずきひろのぶ
suzuki.hironobu@gmail.com



みんなでもう一度見つめなおそう

【第六回】攻撃は常に、そして頻繁に発生している

インターネット接続しているサーバの管理者ならば、日常の経験から、攻撃は常に、かつ頻繁に発生している状況を知っているでしょう。しかし、管理者でなければ、そういった現状を知るチャンスは多くありません。そこで今回は、攻撃の状況がどのようなものか、サーバを中心に具体的に述べてみます。

攻撃は日常風景

2014年4月9日のWindows XPのサポート終了に伴い、自治体などの古いシステムをどうするのかという話題が、先日、新聞に取り上げられていました。そこに、とある自治体の担当者の言葉として次のようなことが載っていました。

「サイバー攻撃はめったにあるものじゃないし、別に不安はない」[注1]

定期的なシステムのセキュリティアップデートやサードパーティーアプリケーションの脆弱性対策が実施され、メールに添付されて侵入を試みようとするマルウェアのチェックや侵入予防／侵入検知システムなどが用意されている。そして、それらが効果的に機能を発揮している場合には、具体的な攻撃が成功するのは難しいかもしれません。

一方で、今どきのマルウェアは昔のような愉快犯ではなく、攻撃ボット端末として密かにパソコンの資源を使う、コンピュータ内部にある組織内の情報を窃取する、あるいは個人のパスワードやサーバへアクセスするための秘密情報を窃取するといった目的で使われています。そのため、すでに攻撃が成功しているにもかかわらず、ただ単にそれを認知していないだけ、ということもありえます。

パソコンのエンドユーザには実感がないかもしれませんが、サーバを管理していると、1つだけ確実に言えることがあります。それは

「攻撃は日常的かつ頻繁に発生している」

ということです。

デフォルト設定を狙う

理解しやすさを考え、前回、取り上げたWordPressを今回も取り上げたいと思います。これから述べることは、WordPressがとくに脆弱というわけではなく、コンテンツマネージメントシステム一般に、だいたい共通して言えることです。

WordPressは海外での普及度は高く、2011年において米国で新しく立ち上がるドメインサイトの22%がWordPressを動かしているということです[注2]。普及しているプラットフォームはターゲットとして選ばれやすいと言えます。また、オープンソースですので、とりあえずダウンロードしてみて、ネット上にあるインストールの経験談の情報を読みながらインストールしてみた、という人も多いのではないでしょうか。

WordPressのインストールには、気になることがあります。インストールを進めていくと、WordPressの管理者アカウント名がデフォルトでadmin

注1）「自治体低い危機意識『サイバー攻撃めったにない』XP期限切れ問題」 読売新聞、2013年10月6日、朝刊、35面
注2） State of the Word, Posted August 19, 2011 by Matt Mullenweg http://wordpress.org/news/2011/08/state-of-the-word/

となっているのですが、それを直さず、とりあえず進めていくパターンが非常に多いです。しかも、この重要な管理者アカウントのパスワードは人間が入力するようになっています。WordPressのほかの利用者については自動的にパスワードを生成し、メールで通知するというWordPressの便利な機能を使うケースがほとんどですが、管理者アカウントはパスワードを必須で求められているように見えるので、人間がパスワードを入れてしまうケースが多いのです。

本連載の第1回目（本誌2013年7月号）で、人間がパスワードを選ぶ場合、ある一定の割合で弱いパスワードを選ぶ傾向があることを紹介しましたが、この場合も同様です。しかも、とりあえずのインストールで事前計画性が怪しいですし、たぶん、将来的にも、とりあえずインストールしたままの状態である可能性が高いでしょう。

wp-login.phpへのアクセスを分析してみる

WordPressはログインをするときにwp-login.phpを呼び出します。筆者のサイトにはWordPressをインストールしていますが、HTMLドキュメントルートの直下にはインストールしていません。セキュリティのためにそうしたのではなく、WordPressのようなコンテンツマネージメントシステムはサイト運用開始の時期から比べるとずいぶん後になってから現れたので、たまたま、インストールする先がHTMLドキュメントルートの直下になっていないだけです。

ですが、そんなことはおかまいなしに、とにかくサイトのトップにWordPressがあるものとして、世界中から存在しないwp-login.phpにアクセスされます。平均すると、3日に1回のペースでアクセスされます。

実際には存在しないドキュメントルート直下にあるwp-login.phpへアクセスした記録を分析すると、どのような結果が現れるのか。興味があったので、過去1年間（2012年10月～2013年9月）のログデータをもとにして調べてみました[注3]。IPアドレスから発信元の国をリストアップした結果は**表1**、**図1**のとおりです。このログを取った筆者のサイトは都内の仕事場に設置し、国内のISPに固定IPアドレスで接続しているサーバです。

繰り返しになりますが、存在していないドキュメントルート直下にあるwp-login.phpへのアクセスは、WordPressへのログインへの試行を目的とした意図的なアクセスとして判断するのが最も妥当です。狙いはadminやguestといったありきたりなアカウントに対して弱いパスワードを試みての侵入、あるいは無条件にユーザを登録できるモードのまま運用しているようなサイトを見つけることでしょう。

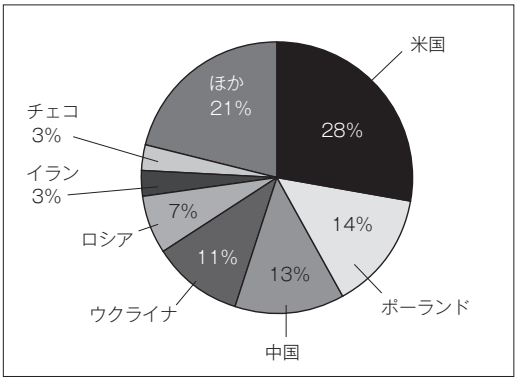
高度な技術やツールは必要ありません。侵入確率は低いですが、アングラサイトからダウンロードし

◆表1　wp-login.phpに対するアクセスの各国別の割合

国名	割合
米国	28%
ポーランド	14%
中国	13%
ウクライナ	11%
ロシア	7%
イラン	3%
チェコ	3%
ほか ※	21%

日本国内からのアクセスはなし。
※IPアドレスの中で14%が判別不可能だっので、それを除外しています。

◆図1　wp-login.phpに対するアクセスの各国別の割合（グラフ）



注3）筆者の研究プロジェクトであるWCLSCAN分析ツール群を利用しています。 www.wclscan.org

たツールを使って絨毯爆撃的に行えば良いのです。たとえわずかな確率であっても、攻撃の全体数が多いので、結果として、それなりの数のサイトを見つけることができます。人間のミスやうかつさを根絶するのは、たいへん難しい問題だということの証かもしれません。

さて、**表1**でインターネットのユーザ数が多い米国と同様に、ユーザ数が多い中国が上位に顔を出すのは妥当かと思います。一方で、日本、インド、ブラジル、ドイツといったインターネットユーザ数の上位国は現れません。その反対に、ポーランド、チェコ、ウクライナといった東欧の国が顔を出します。ロシアとイランはインターネットのユーザ数でいえば世界の10位前後ですので、そこのレベルで現れるのは妥当かと思われます。ユーザ数ではロシアやイランと同じレベルの韓国は、この攻撃には現れません。

東欧のクラッカーたちの不正アクセス技術は、アジア地域のそれよりも洗練されていると言われていたり[注4]、また、今年に入ってからも、Apple、Facebook、Twitterなどの大手が東欧のクラッカー集団のマルウェアのターゲットになっているといった報道がなされています[注5]。それを考えると、ポーランド、チェコ、ウクライナそしてロシアといった国々から日本も含めた世界中をターゲットとして不正アクセスの攻撃を試みられていることは、なんら特別なことでも、不思議なことでもないことがわかります。手元にあるログを少し分析してみるだけで、こんなことがすぐにわかります。

攻撃が失敗していても負荷は上がる

経路を暗号化して安全にリモートログインするsshのサーバに不正にログインを試みるような攻撃に関して言いますと、sshユーザにパスワード利用を許可している場合などでは、何時間も時間をかけて何万という数のユーザIDとパスワードの組み合わせを手当たり次第に試していくブルートフォース攻撃を平気でしかけてきます。

筆者のWCLSCANプロジェクトの経験から言えば、これもとくにターゲットを絞っているわけではなく、インターネット上のサーバを網羅的に探して、そして見つけたら攻撃しています。

sshdのデフォルト設定では、rootはログインできないという安全な設定になっています。もしrootをssh経由でログインさせたい場合は、設定ファイルに明示的に記述しなければなりません。しかし、rootで直接外部からログインできるということ自体がたいへん危険な状況を招く可能性があるので、この設定はしないというのが基本です。

サーバ上にある既存のユーザアカウントうち、特定のアカウントだけssh接続を許可するAllow Usersの指定により明示的に利用ユーザを限定できますし、その際に、どのユーザがどこのIPアドレスからアクセスできるのかといった限定もできますので、より安全性を高められます。公開鍵接続のみ可能な設定にできればいいのですが、なかなかそうもいきません。

余談ですが、sshdの設定をリモートから行った際に記述を間違えると自分がネットワーク経由でログインできなくなってしまうので、この手の設定をするときは、まず1つの端末でssh接続のセッションは続けている状態で、設定を変更しても自分が外部からログインできるかを確認します。

さて、ここでsshdの特徴なのですが、ログインを間違えた場合でも、ログインができない場合でも、相手にその違いを悟られないように、エラーメッセージはまったく同じものが出力されます。sshの不正ログインスクリプトを作成する側／利用する側にしてみれば、rootのパスワードを間違えているのか、それとも設定でrootのログインを許していないのかの差を区別できないので、勢い何百回も何千回も、あるいは何万回もパスワード探しをしてしまう結果になります。

注4）Report Examines Eastern European Hackers Vs. East Asian Hackers **http://www.securityweek.com/report-examines-eastern-european-hackers-vs-east-asian-hackers**

注5）米アップルのマルウェア感染、東欧ハッカー集団が関与-関係者 **http://www.bloomberg.co.jp/news/123-MIHWYT6K50YW01.html**

安全なパスワードを使っているという前提で、一応のセキュリティは保たれているとしても、このような攻撃下では、sshに接続し通信するトラフィックは明白に増えます。サーバの計算資源やネットワーク資源を無駄に消耗するのは気持ちのいいものではありません。

sshへの接続を制限する

この手の対策をするときは、筆者は2つのパターンを使います。1つはsshのデフォルトのポート22番を変更してしまうこと。もう1つはiptablesのhashlimitを使い、繰り返して接続を行う通信元を一時的にアクセスさせないようにすることです。

前者は単純にデフォルトのポート番号22の代わりに、ポート番号が20000～50000台で使われていないランダムなポートを選んでsshdに使うものです。欠点は、クライアントが使っているネットワークのファイアウォールが、アウトバウンド方向にポート番号22しか許しておらず、sshの通信が届かない場合があることです。

後者はiptablesのhashlimitの機能を使います（**図2、表2**）。

ハッシュリミット（hashlimit）の機能は、規定時間内にどれだけのパケットを受理するかというものです。高い頻度で接続を試みようとするホストは、一定時間抑制することになります。--hashlimit-htable-expireは、その単位時間を設定します。単位はミリ秒です。**図2**の例だと60000ミリ秒＝60秒＝1分を設定しています。この1分という時間が一定の区切りとなる規定時間（クォンタム）となります。1分間に同一のIPアドレス（同じホスト）から10回以上SYNパケットが到着したら（sshへの接続が10回以上されたら）制限をかけ、あとは捨てることになります。1分が過ぎると、パケットのカウントは破棄され、また同じ繰り返しをします。

--hashlimitの値は、これ自体はあまり意味はありませんが、必須パラメータですので、とりあえず1分ごとに1パケットを通過させるという値を設定しておきます。

iptables -Lで設定状況を確認できます。**図3**のような設定にすることによって接続を制御し、ブルー

◆図2　iptablesでhashlimitを設定する

```
# iptables -I INPUT -p tcp -m state --state NEW --dport 22 -i eth0 ➥
--tcp-flags SYN,RST,ACK SYN -m hashlimit --hashlimit-burst 10 --hashlimit 1/m ➥
--hashlimit-mode srcip --hashlimit-htable-expire 60000 --hashlimit-name sshattack -j ACCEPT

# iptables -A INPUT -p tcp -m tcp --dport 22 --tcp-flags SYN,RST,ACK SYN -j DROP
```

◆表2　図2上のiptablesのオプション

オプション	説明
-I INPUT	入力にフィルタリングする
-p tcp	プロトコルはTCP
-m state	ステートモジュールを使う
--state NEW	新しいステートを生成
--dport 22	ポート番号22（ssh）
-i eth0	インターフェースはeth0
--tcp-flags SYN,RST,ACK SYN	対象とするTCPのフラグ
-m hashlimit	ハッシュリミットモジュールを使う
--hashlimit-burst 10	規定時間内に10パケットが来たらリミットを効かせる
--hashlimit 1/m	リミット時は60秒に1パケットを上限とする
--hashlimit-mode srcip	ソースIPで状態を区別する
--hashlimit-htable-expire 60000	リミットを決める規定時間（60秒）
--hashlimit-name sshatack	hashlimitで区別に使う名前（sshatack）
-j ACCEPT	通過させる

トフォースによるsshサーバへの多数の接続数を低減し、負荷を下げることが可能となります。

hashlimitのステート状態はファイル/proc/net/ipt_hashlimit/sshatackに収められていますので、この内容を観察することで、どのような状態なのかがわかります。

sshへのブルートフォース攻撃

今回の原稿のためにちょっとしたログ収集をして分析してみました。このサイトは先ほどのWord Pressのサイトとはまったく関係のないサイトで、日本国内にはありますが、IPアドレスやISPは異なり、上位のバックボーン接続も異なる、まったく別の空間です。

デフォルトの22番ポートで運用し、iptablesなどの対策をせず、接続を試み、パスワードを試みたものは、すべて接続させました。期限は4日間という短い期間でしたが、攻撃元はユニークIPアドレスで15を数えました。

一度に100回以上のパスワードを試みている危険な攻撃は8ヵ所からありました。その内訳は**表3**のとおりです。最も頻度が高かったのは韓国から行われた攻撃で517秒間に1312回のパスワードを試していました。これはネットワーク的に近く、通信のレスポンスが速いからかもしれません。興味深いのは、ここでもウクライナが顔を出しているという部分でしょう。東欧はやはり活発です。

今もこの瞬間に攻撃されている

パソコンのエンドユーザに、今、この瞬間にも攻撃されていることを実感してもらうのは、なかなか難しいと思いますが、実際にインターネット上では日常の風景として攻撃が発生しています。何度も繰り返しての説明になりますが、これらの攻撃は、自動化されており、ツールはネット上で簡単に手に入るので、中学生程度のパソコンの知識があれば誰にでもできるレベルになっています。

パスワードのブルートフォースという最も単純な部類の攻撃ですら、低いとはいえ、ある一定の確率で成功していますし、インターネット全体で考えてみると相当数のサイトが侵入可能と考えるべきです。また、そこが隠れ家になり、ほかのサイトへの攻撃の拠点にもなります。

もし、管理しているサーバがあるのならば、そのログからどのような攻撃が行われているのか、一度、精査してみるのもいいかもしれません。攻撃が失敗しているとはいえ、思っている以上に攻撃されていることがわかるはずです。SD

◆表3　一度に100回以上のパスワードを試みた攻撃

国名	回数
中国	3
米国	2
韓国	1
インド	1
ウクライナ	1

◆図3　iptablesの設定状況

```
# iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source               destination
ACCEPT     tcp  --  anywhere             anywhere            state
NEW tcp dpt:22flags: SYN,RST,ACK/SYN limit: up to 1/min burst 10 mode
srcip htable-expire 60000
DROP       tcp  --  anywhere             anywhere            tcp
dpt:22flags: SYN,RST,ACK/SYN

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
```

プログラム知識ゼロからはじめる
iPhoneブックアプリ開発

第8回

インタラクティブ性を持たせよう(その2)アニメーション

GimmiQ(ギミック;いたのくまんぼう&リオ・リーバス)
URL http://ninebonz.net/
URL http://www.studioloupe.com/
イラスト●中川 悠京

プログラミングをしたことのない方にもアプリを作る楽しさを味わってもらいたい本連載。今回はページ内の物をアニメーションさせる方法を解説します。

もっともっと「アプリ」らしく

今月も先月に引き続き紙の書籍ではできないこと、アプリならではの要素を追加しましょう。今月追加するのは「アニメーション」です。

ページに表示されている物に触るとその物が動くようなものを作りましょう。これができるとインタラクティブな絵本のようなアプリを作れるようになります。

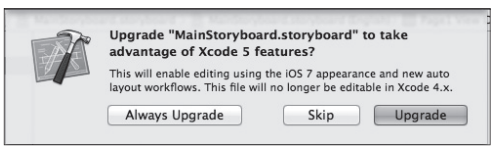
その前に

今月の講習に入る前に、開発環境を最新のものにしましょう。これまではXcode 4.6系、iOS 6を対象にしていましたが、Apple社から正式にXcode 5とiOS 7がリリースされましたので、最新の環境に対応するために今月から当連載もXcode 5、iOS 7を対象にしたいと思います。

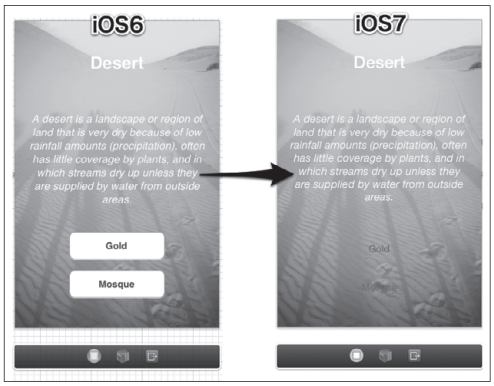
アップデートしたXcodeでこれまでの連載で作ったプロジェクトファイルを開くと**図1**のように「プロジェクトファイルをXcode 5用にアップデートするか?」と確認が出ますので[Upgrade]をクリックしてください。これでプロジェクトファイルは変換され、ストーリーボード上の各UIパーツのデザインもiOS 7準拠のものになったかと思います。とくに大きく見た目が変わったのはUIButtonのパーツです。ボタンの枠がなくなり文字だけの表示になりました(**図2**)。

それでは、ここで1回実行してみましょう。今回からはiOS 7を対象とするためにシミュレータの実行設定も変更しましょう。Xcode左上の

▼図1　アップデートを確認するダイアログ



▼図2　見た目が変わる!



▼図3　シミュレータの設定変更

[実行設定]をクリックし[iPhone Retina (3.5-inch)]→[iOS 7.0]を選択してください(**図3**)。

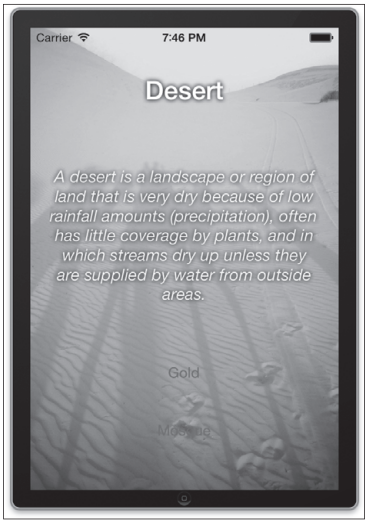
実行すると**図4**のようになったかと思います。よく見ると消していたはずのステータスバーが表示されるようになってしまい、時刻やバッテリー残量が表示されています。これはiOS 7からステータスバーを非表示にする方法が変更になったためです。書籍アプリでは邪魔になりますので改めて非表示にしましょう。

Xcode左の「Project Navigator」でプロジェクト設定用のplistをクリックして内容を表示してください。このplistの名称はプロジェクトごとに自動で名付けられています。作例では「MyBook2-Info.plist」となっています(**図5**)。

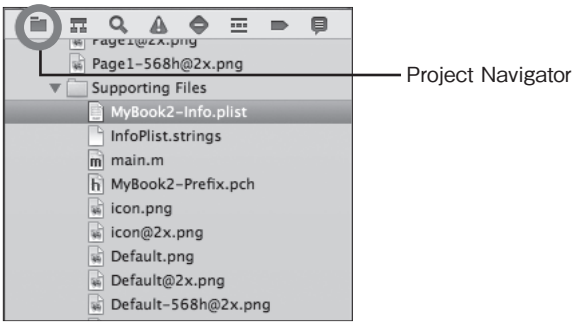
どの行でもよいのでマウスカーソルを合わせると、**図6**のように行追加用のボタンが出てきますので[+]をクリックし、新しい行(設定)を追加します。「Key」欄に「View controller-based status bar appearance」と入力し、「Type」欄は「Boolean」、「Value」欄に「NO」を設定してください(**図7**)。

実行すると**図8**のようにステータスバーが非表示になっているかと思います。すっきりしたところで、今月のメインテーマであるアニメーションを作っていきましょう！

▼図4　シミュレータの実行画面

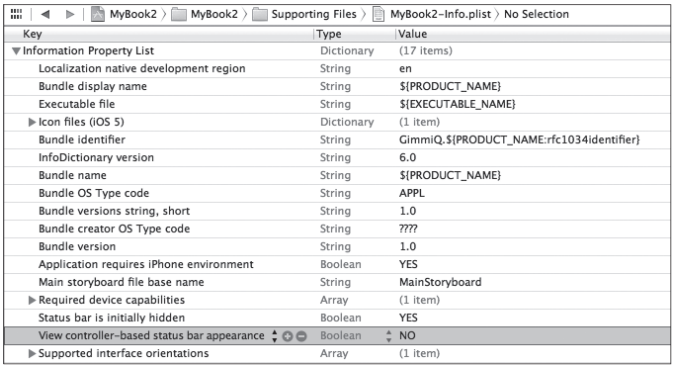


▼図5　plistの選択



▼図6　新しい設定行の追加

▼図7　ステータスバーを非表示にする設定



▼図8　ステータスバーが消えた

車を走らせよう！

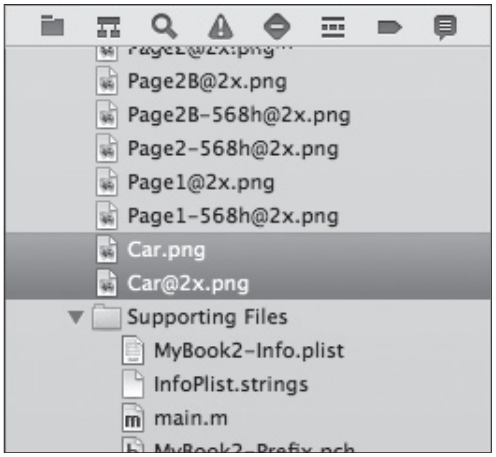
まず、今回つくるアニメーションの完成形を見てみましょう。ページに車のイメージを配置し(**図9**)、この車をタップすると**図10**のように右方向に走り去っていくアニメーションを作ります。

ステップ1

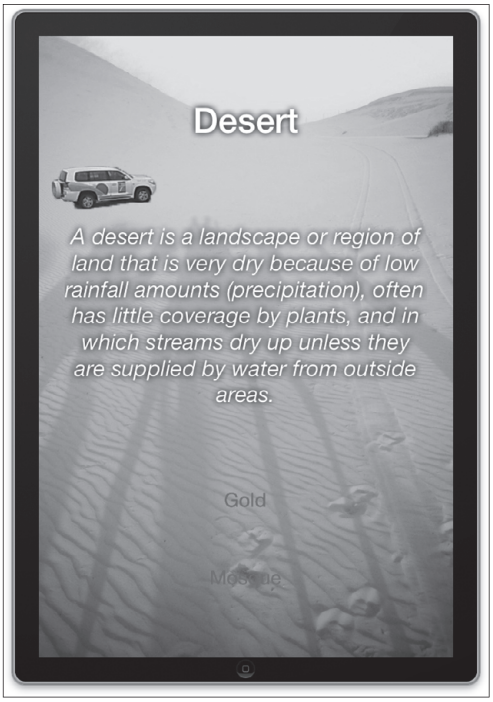
まずは車の画像をプロジェクトに追加しましょう。用意した画像(作例ではCar.pngとCar@2x.png)[注1]を「Project Navigator」にドラッグ&ドロップして、画像をプロジェクトに追加します(**図step1**)。

注1) 作例で使用している画像は本連載サポートページのほうに用意しましたので、よろしければお使いください。

step1



▼図9　ページに車のイメージを配置



▼図10　右方向へ走り去っていく

ステップ2

画像をページに配置しましょう。Xcode右下ペインで「Media library」のアイコンを押し、画像一覧から先ほど追加した車の画像を1ページ目へドラッグ&ドロップしましょう(**図step2-1**)。

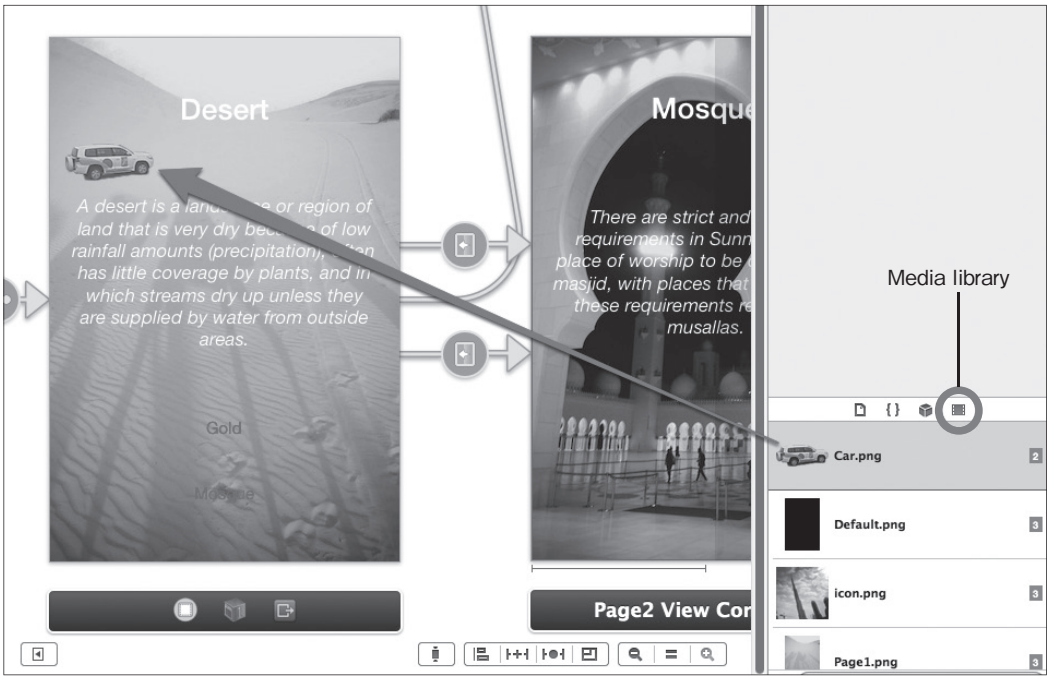
Xcode右上ペインで「Size Inspector」のアイコンを押し、画像のサイズ(Width、Height)と位置(X、Y)を**図step2-2**と同じ値にしてください。

画像を表示するのは「UIImageView」というパーツを使用するのですが、このように「Media library」から画像を配置することで自動的にUIImageViewに画像が設定された状態になります。画像が直接置かれているのではなく**UIImageViewというパーツに画像が乗っている状態**になっているということを意識しておきましょう。

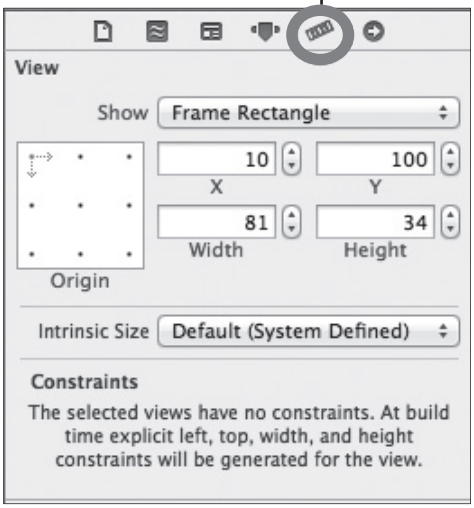
車を右方向に走らせるというのは、コード上では「車の場所を右に変更する」ということになります。

車の位置を変更するというのは、このUIImageViewの「座標」を操作することになります。そのためにストーリーボード上のパーツをコードから操作できるようにする方法を覚えましょう。

step2-1



step2-2



ステップ3

ステップ2で配置したUIImageViewをコード中で扱えるように関連付けましょう。1ページ目に配置した画像ですので「Page1ViewController.h」にコードを追加します。次の1行を**図step3-1**のように追記してください。

```
IBOutlet UIImageView *carImage;
```

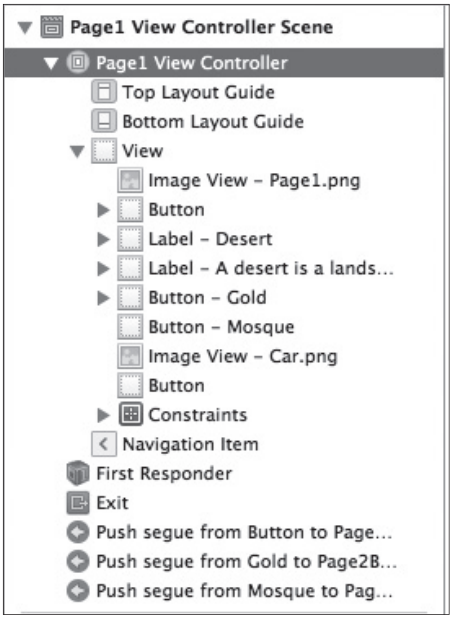
step3-1

```
#import <UIKit/UIKit.h>
#import <AVFoundation/AVFoundation.h>

@interface Page1ViewController : UIViewController{

    IBOutlet UIImageView *page1Image;

    IBOutlet UILabel *titleLabel;
    IBOutlet UILabel *bodyLabel;

    IBOutlet UIImageView *carImage;

    AVAudioPlayer *sound;
}
```

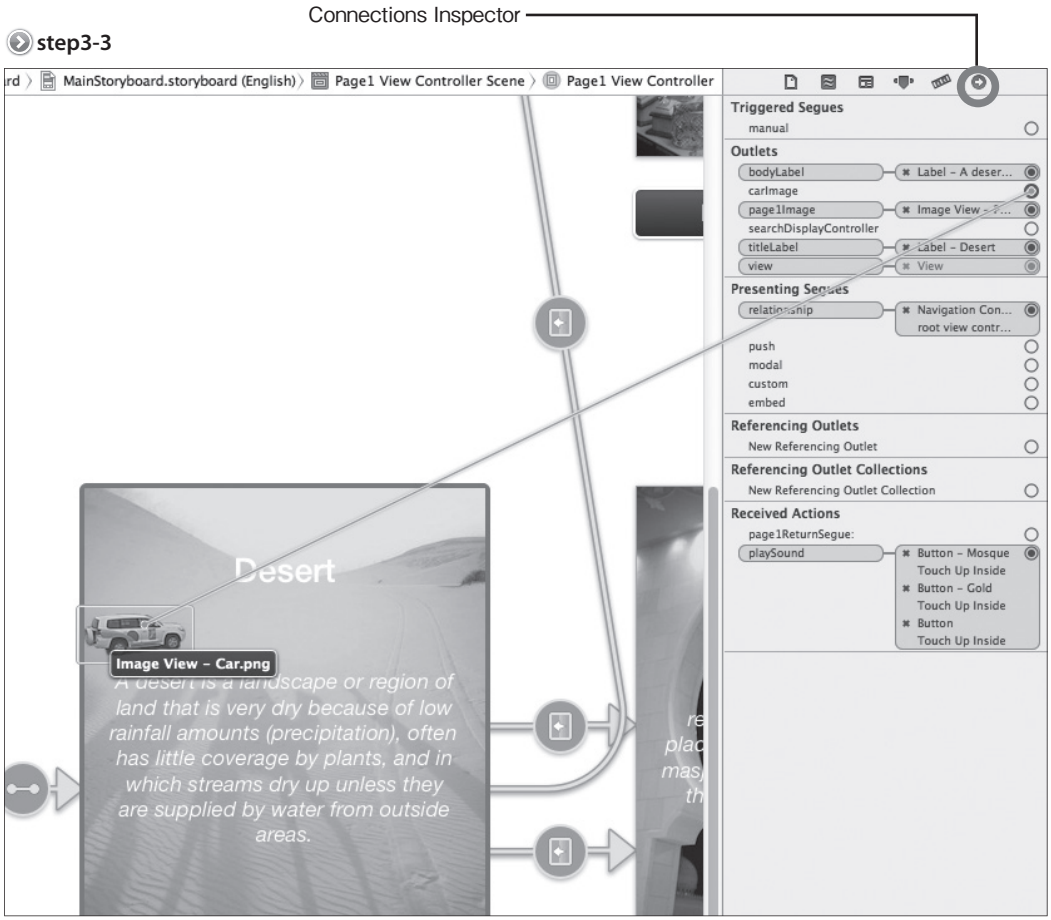
コードを記述したら、今度はストーリーボードを開いて1ページ目のViewController（Page1 View Controller）をクリックし選択状態にします（**図 step3-2**）。続いてXcode右上ペインで「Connections Inspector」のアイコンを押してください。「Outlets」の中に先ほどコードに追記した「carImage」というラベルがあると思います。ラベルの右にある「○」をドラッグ＆ドロップし、伸びてきたラインを車のUIImageViewまでつなげます（**図 step3-3**）。

これで、コード中のcarImageと車のUIImage Viewが関連付けられ、carImageという名前で車のUIImageViewを扱えるようになりました。

step3-2



step3-3



ステップ4

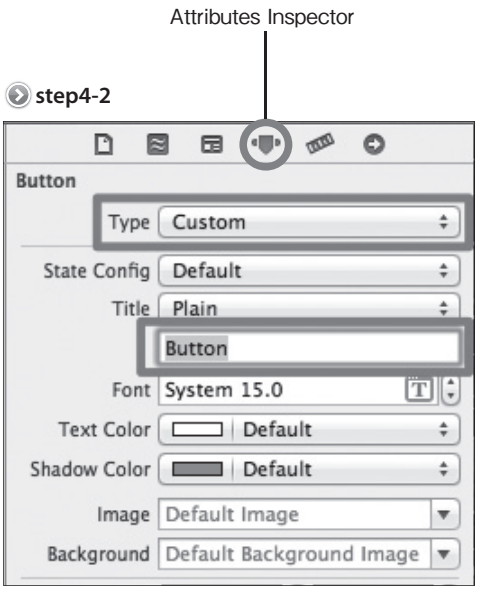
次に、この車の画像をタップしたことを感知するためのボタンを配置しましょう。Xcode右下ペインで「Object library」のアイコンを押し、「Button」を車のUIImageViewと同じ場所にドラッグ＆ドロップしてください（**図step4-1**）。

ステップ2と同様に「Size Inspector」でボタンのサイズと位置を、車のUIImageViewと同じ値に設定します。

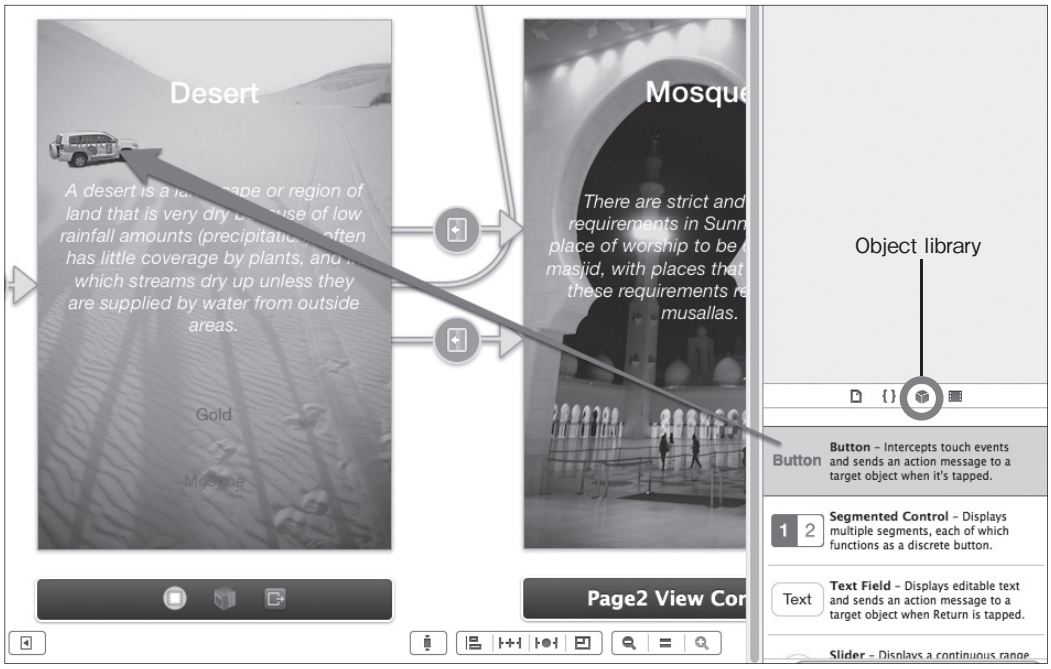
また「Attributes Inspector」でボタンの「Type」欄を「Custom」に変更します。ボタンに表示される文字も変更しましょう。デフォルトでは「Button」という文字が設定されていますので、これを削除して空の状態にしてください（**図step4-2**）。

これで、車のUIImageViewの上に同じ大きさの透明なボタンが乗っている状態になりました。このボタンを押したときに車が走り出すようにしたいと思います。そのためにはボタンを押したときに呼び出される処理をコードに記述する必要があります。その処理の中で、carImageの座標をアニメーションしながら変更するようにします。

step4-2



step4-1



ステップ5

いよいよアニメーション用のコードを記述します。「Page1ViewController.m」に**リスト1**のコードを追記してください。

このコードはmoveCarImage{ }; という関数を作り、その中にcarImageをアニメーションさせる処理を記述しています。[UIView animateWithDuration:5.0f]; という部分は「5秒かけて以下の処理を実行する」ことを意味しています。そして5秒間にする処理はanimations:^{ }の中に記述されています。carImage.center = CGPointMake(370, 100); という部分が実際にcarImageの動きを定義している部分です。carImage.centerはcarImageの中心座標を意味しています。ですので、中心座標をCGPointMake(370, 100)に変更するという処理になります。

画面上の座標を表すには「CGPointMake(X座標, Y座標)」という形式で記述します。iPhoneの画面は横320ドットしかありませんので、370というのは画面外を示しています。

▼リスト1　Page1ViewController.m

```
- (IBAction)moveCarImage {

    // 車のイメージをアニメさせる
    [UIView animateWithDuration:5.0f //5秒かけてアニメを再生
                     animations:^{
                         carImage.center = CGPointMake(370, 100);  // 車のイメージを画面外に移動
                     }];
}
```

ステップ6

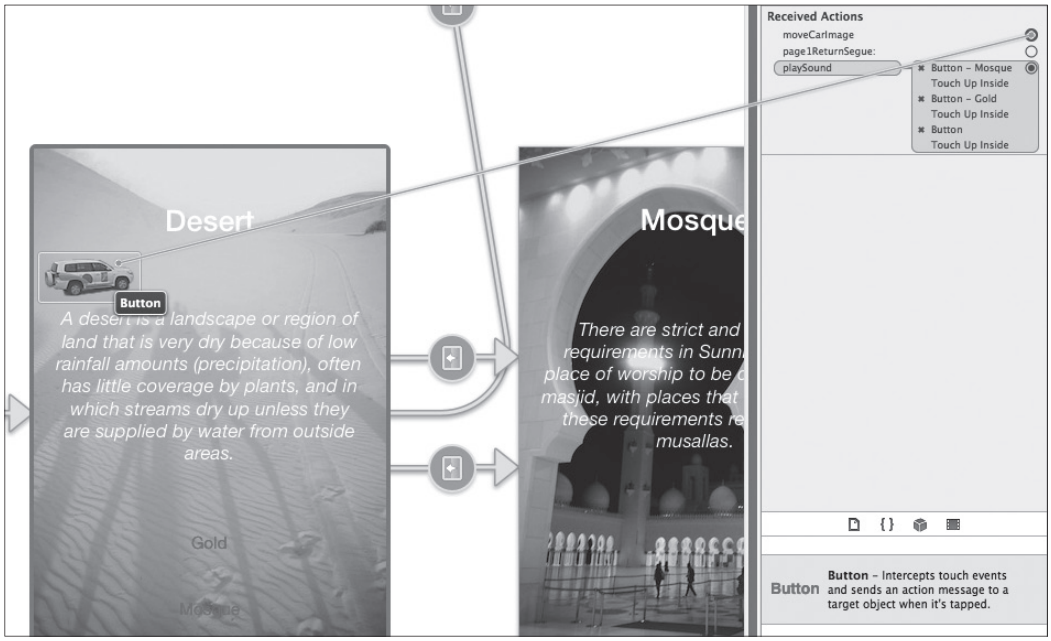
ステップ5の処理をボタンに関連付けましょう。ステップ3と同様に「Connections Inspector」を開いてください（次頁**図step6-1**）。「Received Actions」にPage1ViewController.mへ記述したアニメーション用の関数「moveCarImage」が表示されていると思います。「moveCarImage」の右にある「○」をドラッグ＆ドロップし、伸びてきたラインをボタンまでつなげてください。

ドロップ時に表示されるメニューは「Touch Up Inside」を選択してください（**図step6-2**）。これは、関連付けられた関数を呼び出すタイミングの設定になります。「Touch Up Inside」はボタンを押してその指がボタン内で離されたときを表しています。

以上でボタンをタップするとcarImageが横方向に移動するようになったはずです。実行して確認してみましょう。無事に車は砂漠の彼方に走り去っていったでしょうか？

step6-2

step6-1



可能性は無限大

いかがでした？　これでユーザの操作に応えて画面上の何かが動くような処理を作れるようになりました。先月号の「分岐」による物語の変化と組み合わせるといろいろなアイデアを形にできると思います。

たとえば、写真に隠されているアイテムを探してタップすると、そのアイテムが写真から飛び出し、そして物語が変化していく、というようなアプリだってもう作成可能なのです！

いろいろなアイデアを思いついたら、とりあえず作り始めてみましょう。そうやって手を動かしていると自然と理解も深まって、また新たなアイデアを生み出すきっかけにもなります。SD

本連載のサポートページで記事の補足説明をしています。あわせてご活用ください！
› http://www.gimmiq.net/p/sd.html

いたのくまんぼう／Itano Kumanbow
Twitter @Kumanbow

神奈川工科大学非常勤講師。リオさんとはGimmiQ名義で「MagicReader」（手を使わずにページがめくれる電子書籍ビューワ）をリリース。個人ではNinebonz名義で「江頭ジャマだカメラ」（無料総合1位獲得）、「i列車の車窓から―そうだ! 京都行こう!―」（バーチャル旅行アプリ）などをリリース。アプリ紹介サイト「あぷまがどっとねっと（http://appmaga.net/）」の技術サポータ。

リオ・リーバス／Leo Rivas
Twitter @StudioLoupe

iOSアプリ開発を中心に電子絵本作家・漫画家として活動中。代表作は、KDDI株式会社に社内導入され、世界で40万ダウンロードを記録する革新的な電卓アプリ「FusionCalc」と、国連主催のWSA Mobile 2013を受賞した、顔の動きで電子書籍が読める「MagicReader」。電子絵本はiBookstore/Kindleストア共に児童書カテゴリ総合1位を獲得。

第43回

[アプリ開発2013] 4 Intentを使えるようになろう

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！

重村 浩二　SHIGEMURA Koji
日本Androidの会 中国支部長
Mail k-shigemura@android-group.jp

Siriライクなアプリを作ろう

［アプリ開発2013］と題した前回までの3回では、開発環境構築とアプリ開発に必要な重要要素であるActivityなどの解説を通じて簡単なアプリ開発の基礎を見てきました。4回目となる今回からは、筆者が実際に公開するアプリを開発するときの流れを見ていくことで、読者の皆さんの開発の参考としてもらえたらと思います。

題材として作成するのは、音声認識を使ったアプリの開発です。iPhoneに搭載されているSiriは読者の皆さんもご存じでしょう。今回はそのSiriライクなアプリの開発を行います。

アプリ開発の流れ

Androidのアプリ開発をする際には、どのように開発を進めるのがよいのでしょうか？　ソフトウェア開発の世界では、先人たちのノウハウの蓄積によって開発の流れがモデルとしてまとまっています。ウォーターフォール・モデルやプロトタイプ・モデル、スパイラル・モデルなどさまざまな開発モデルがありますが、Androidアプリの開発ではアジャイル開発が適していると筆者は考えています。これはAndroidアプリに限ったことではなく、iPhoneアプリなどを含むモバイルアプリの開発全般に言えることで、すでにさまざまな企業でアジャイル開発を用いてアプリ開発が行われています。

モバイルアプリの開発では単納期・高品質が求められ、求められる仕様も時が過ぎるごとに頻繁に変わっていきます。そのため、ウォーターフォール・モデルのように要求定義→概要設計→詳細設計→プログラミング→単体テスト→機能テスト……と進めていこうとすると、プログラミングを始めたタイミングに概要設計の仕様変更が必要だ！となってしまい、どんどんコストがかかってしまうことになります。場合によっては、求められている成果物が作り出せない事態となってしまうことも発生してしまうでしょう。

アジャイル開発を採用すると、ポイントごとに追加の要求を受け入れることができるために、変化に強いシステム開発が可能となります。

ここまで、アジャイル開発を採用するメリットを企業で開発する立場の方向けに述べてみましたが、個人のアプリ開発者が採用するメリットはただ1つ！　モチベーションを維持するためです。一人で黙々とアプリ開発をしているのはなかなかつらいでしょう。短いスパンで成果物を作り、さまざまな人からのフィードバックを得る形を作れるアジャイル開発のほうが、モチベーションも維持しやすいと筆者は考えます。

以降では、実際のアプリの開発の流れに沿って、1つ1つ手順を見ていきましょう。

アプリの要件（要求仕様）を考えよう

まずはアプリの要件を考えましょう。どのよ

うなことができて、どんな人向けのアプリなのかがまとまっていないと、作る成果物もぼやけたものとなってしまいます。すべてが決まっている必要はないので、要点がきちんと押さえられていれば問題ないでしょう。個人でアプリ開発する際は、実装したい機能を箇条書きでメモに洗い出すところから入るのでも十分です。

最初の仕様は紙に書こう

要件を考えたら、実際に要求仕様として仕様書に起こしましょう。会社でしたら、一般的にMicrosoft WordやExcelを用いて、社内で定められた様式に沿って仕様書を作ることになるかと思います。ただ、個人で開発を進めていくのであれば、まず何よりも先に、次に出てくる動くもの(モック)を作ることが最優先となりますので、紙1枚に起こしたような簡単な仕様でも問題ないと思います。

今回のSiriライクなアプリ(以降はLikeSiriと呼ぶことにします)を開発するために、筆者も**写真1**にあるような簡単な仕様を紙に書いて、どのような動きをするアプリなのかをまとめてみました。**写真1**を見ていただけたらわかるとおり、画面のイメージと遷移も一緒に表現してしまえば、より直観的にどんなアプリを作るのかを想像しやすくなります。

今回の仕様は、アプリ起動後に表示した画面上で❶電話をかけたいことを伝えると、❷電話番号を聞かれるのでかけたい番号を伝え、❸TEL画面に遷移できるという仕様です。地図を表示する場合の遷移も、**写真1**を見るだけでイメージしてもらえるのではないでしょうか。

必要最低限の機能しか載せていないので、エラー時の処理などはどうするのだ？といった疑問が出てくると思いますが、この段階ではその点については置いておきましょう。まずは動くアプリを作り、周りからのフィードバックを得ることを重視します。アプリを何度も何度もブラッシュアップしながら機能を強化していけばよいのです。

仕様に合わせたモックアプリを作ろう

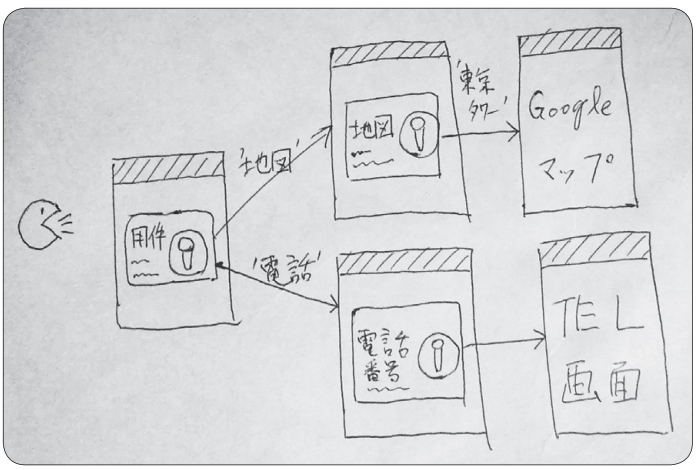
簡単な仕様書ができたら、さっそく作る工程に入りましょう。最初に構築するものは要求仕様で実現したい機能をすべて盛り込んでいる必要はありません。最低限の画面遷移ができて、どんなことができるのかを確認できればよいでしょう。

次項では、LikeSiriを作成するのに必要な知識の習得を行いましょう。

アプリを作るために必要な知識

今回開発するLikeSiriでは、Androidのアプリ開発を行ううえで重要な機能の1つとして挙げられるIntentを多用しています。実際のアプリ開発でも、要求された機能を実現するために必要な知識は随時入手しながら、テストをしつつ実装を進めていくことになります。それでは、LikeSiriの開発で重要となるIntentについて、概要と具体的な活用方法を学んでいきましょう。

▼写真1　LikeSiriの仕様



Intent概要

Intentとは、“○○”を“△△したい”という要求をActivity(およびアプリ)間でやり取りするために用意されたAndroidの基本機能です。この機能を用いることで、アプリ内で画面遷移を行ったり、別のアプリを呼び出すといったことを実現しています。

具体例として**リスト1**のようにすることで、アプリ内のTargetActivityに画面遷移することができます。Intent生成時の第一引数はContextクラスのインスタンスを指定します。ここではthisとし、第二引数に「画面遷移したいActivity名+.class」を指定します。例では、TargetActivity.classと指定しています。

そして、startActivityメソッドに生成したIntentのインスタンスを指定して呼び出すことで、対象のActivityに遷移することができます。追加されたActivityの場合でしたら、AndroidManifest.xmlにActivityの定義を追加することを忘れないようにしてください。

明示的Intentと暗黙的Intent

Intentの呼び出しには、大きく分けて2種類の動作があります。前項で例示した動作は**明示的なIntent**と呼ばれ、呼び出したい処理を行ってほしいActivity(アプリ)を直接指定しています。よく画面の遷移などを行いたいときに使われる手法です。

それとは違う動作として用意されているのが、**暗黙的なIntent**と呼ばれるしくみです。こちらは、処理してほしい対象を指定せずに、“△△したい”というアクションに注目し、その処理が行えるアプリをAndroidのシステム側が自動的に判別し、実行してくれるというものです。

リスト2は、暗黙的Intentを用いて技術評論社のサイト(http://gihyo.jp/)を呼び出す例です。Intent生成時の第一引数で、“データを表示したい”というアクションを表すIntent.ACTION_VIEWを指定し、第二引数に表示したいデータ(ここでは技術評論社のサイトアドレス)を指定しています。

この条件でstartActivityメソッドの引数に生成したIntentのインスタンスを渡して実行すると、Androidのシステム側が自動判別してWebブラウザを起動し、技術評論社のサイトを表示します。標準以外のWebブラウザ(OperaやFirefoxなど)をGoogle Play Storeからインストールしていた場合は、実行可能なアプリが一覧で表示され、その中からユーザが起動するアプリを選択します。

LikeSiriの実装を行おう

アプリ実装に必要な基礎知識を得たところで、いよいよLikeSiriの実装に入っていきます。今回のプロジェクトのサンプルを公開していますので、そちらも参考にしてください[注1]。

画面のレイアウト設計

まずはプロジェクトを作成してください。プロジェクトの作成方法は[アプリ開発2013]の2回目[注2]を参考にしてください。

注1) サンプルは本誌サポートサイトからダウンロードできます。URL http://gihyo.jp/magazine/SD/archive/2013/201312/support

注2) 本誌2013年10月号に掲載の本連載第41回のこと。

▼リスト1　Activityの呼び出し

```
Intent intent = new Intent(this, TargetActivity.class);
startActivity(intent);
```

▼リスト2　暗黙的Intentでの呼び出し

```
Intent intent = new Intent(Intent.ACTION_VIEW, Uri.parse("http://gihyo.jp"));
startActivity(intent);
```

プロジェクトを作成したら、最初は画面のレイアウト設計から入るのがよいでしょう。これは各イベントのトリガーが画面上のボタンなどを起点とするためです。画面レイアウトを作った後、“機能1を作る→テスト、機能2を作る→テスト……”と実装していけば、小さな単位で進捗を確認しながら進めていくことができるので、いきなりすべてを作ろうとするよりもモチベーションも維持しやすいでしょう。

LikeSiriの最初のバージョンは音声認識機能を実装することを目的としているので、機能を呼び出すためのボタンを画面上の真ん中に、**図1**のように設置する仕様としています。

図1の画面を実現するために、**リスト3**にあるようなレイアウトを準備します。ボタン上に画像を表示するために「ImageButton」を利用しています。このボタンがクリックされたことをトリガーとしてイベントを処理しますが、そのためのメソッドの指定として、android:onClick属性に"onClickEvent"としている点に注目してください。ここで指定された名称が次で説明するボタンクリックのイベントを受け取る際のメソッド名となります。

音声認識機能の実装

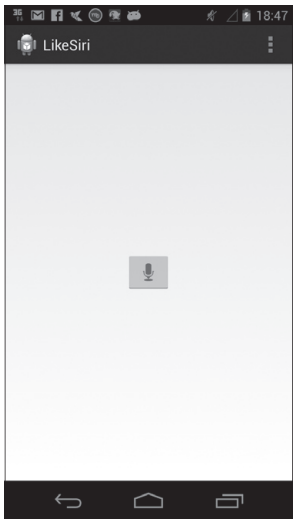
LikeSiriの画面レイアウトが実装できたら、メインとなる音声認識機能の実装を行いましょう。前項で出てきたandroid:onClick属性に指定したのと同じ名前で、**リスト4**のようにメソッドを実装します。

ボタンが押されたときに行っているのは、音声入力画面を呼び出すIntentの発行を行うsendSpeechIntentメソッドの呼び出しです。sendSpeechIntentメソッドでは、音声入力画面を呼び出すため**リスト5**のように音声認識のための引数を指定したIntentを生成し、発行を行います。

ここで注目してもらいたいのは、Intentの発行で利用しているメソッドが、startActivityResultメソッドとなっている点です。このメソッドは呼び出し先のActivityで処理を実行した結果を受け取ることができます。

LikeSiriの仕様では、ユーザが端末に向けて話しかけた言葉をAndroidシステム上の音声認識機能で処理し、次の動作を実施する必要があ

▼図1　LikeSiriの画面



▼リスト3　LikeSiriの画面レイアウト

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".MainActivity" >

    <ImageButton
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:onClick="onClickEvent"
        android:layout_centerInParent="true"
        android:src="@android:drawable/ic_btn_speak_now" />

</RelativeLayout>
```

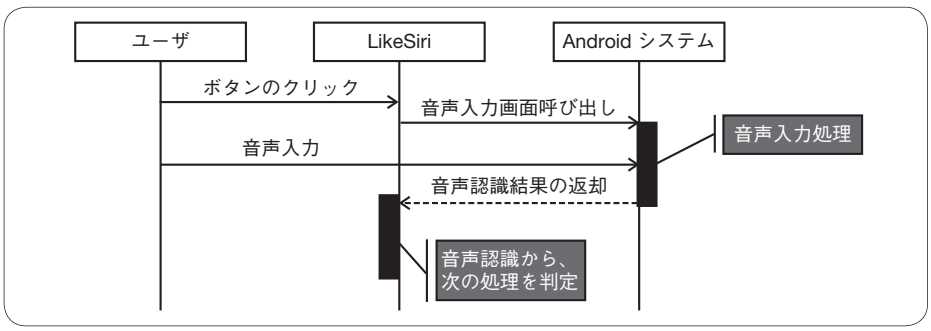
▼リスト4　メソッドの呼び出し

```
// ボタンが押されたときに呼び出される
public void onClickEvent(View v) {
    String msg_start = getResources().getString(R.string.msg_start);
    sendSpeechIntent(REQCODE_START, msg_start);
}
```

▼リスト5　音声入力画面呼び出しのためのIntent発行

```
// 音声入力画面を呼び出すIntentの発行
public void sendSpeechIntent(int reqCode, String prompt) {
    try {
        // インテント作成
        Intent intent = new Intent(RecognizerIntent.ACTION_RECOGNIZE_SPEECH);
        intent.putExtra(
            RecognizerIntent.EXTRA_LANGUAGE_MODEL,
            RecognizerIntent.LANGUAGE_MODEL_FREE_FORM);
        intent.putExtra(
            RecognizerIntent.EXTRA_PROMPT,
            prompt);    // 音声入力時に表示するテキスト

        // インテント発行
        startActivityForResult(intent, reqCode);
    } catch (ActivityNotFoundException e) {
        // このインテントに応答できるアクティビティがインストールされていない場合
        Toast.makeText(this,
            "この環境では音声認識機能を利用することができません。", Toast.LENGTH_SHORT).show();
    }
}
```

▼図2　ユーザのボタンクリックから音声認識によって次の処理に移るまでの流れ



ります。そのため、**図2**のような流れとなるように、処理結果を受け取れるよう実装する必要があるのです。

呼び出し先のActivityからの結果受け取りは、onActivityResultメソッドを利用します。**リスト6**にあるように、第三引数dataからIntentで受け取った文字列リストの結果をgetStringArrayListExtraメソッドで取得し、その中に特定のキーワード(電話、地図)が含まれているかどうかを判定します。含まれていれば再度音声入力画面呼び出しを行い、電話番号の入力や地図で検索する場所の入力を促すようになっています。

ポイントとなるのは、今回の実装は同一のActivity内で実装しているため、onActivityResultメソッド内で処理の分岐を行っている点です。第一引数で渡されてくるrequestCodeで要件確認時の要求結果なのか、電話番号の要求結果なのか、地図検索の要求結果なのかIntent発行時のリクエストを判別しています。

これで、LikeSiriのバージョン1.0が完成しました。実際に動かしてテストしてみると、さまざまな点が足りないことに気付くでしょう。本文では触れませんでしたが、サンプルのコード上で行っているnull判定処理などがなぜ必要なのかは、ぜひデバッグモードで動かして、どのようなときのための実装なのかを解析してみてください。

まとめ

今回はLikeSiriの実装によって、Intentのし

▼リスト6　音声認識結果の判別処理

```
// アクティビティ終了時に呼び出される
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    Intent requestIntent = null;
    ArrayList<String> result = null;
    Resources res = getResources();     // 文字列リソース取得用

    if (data == null) {
        // 結果が取得できなかった
        Toast.makeText(this, R.string.msg_unknown, Toast.LENGTH_SHORT).show();
        return;
    } else if (data != null) {
        // 結果を文字列リストで取得
        result = data.getStringArrayListExtra(RecognizerIntent.EXTRA_RESULTS);
    }

    // 要件の音声認識結果処理
    if (requestCode == REQCODE_START && resultCode == RESULT_OK) {
        // キーワードが含まれているかどうかを判定し、処理を分岐
        String keyTel = getResources().getString(R.string.key_tel); // キーワード(電話)
        String keyMap = getResources().getString(R.string.key_map); // キーワード(地図)
        if (result.get(0).indexOf(keyTel) != -1) {
            // 電話
            String msgTel = res.getString(R.string.msg_tel);
            sendSpeechIntent(REQCODE_TEL, msgTel);
        } else if (result.get(0).indexOf(keyMap) != -1) {
            // 地図
            String msgMap = res.getString(R.string.msg_map);
            sendSpeechIntent(REQCODE_MAP, msgMap);
        } else {
            // 不明
            Toast.makeText(this, R.string.msg_unknown, Toast.LENGTH_SHORT).show();
        }
    // 電話番号の音声認識結果処理
    } else if (requestCode == REQCODE_TEL && resultCode == RESULT_OK) {
        requestIntent = new Intent(Intent.ACTION_VIEW, Uri.parse("tel:" + result.get(0)));
        startActivity(requestIntent);
    // 地図検索の音声認識結果処理
    } else if (requestCode == REQCODE_MAP && resultCode == RESULT_OK) {
        requestIntent = new Intent(Intent.ACTION_VIEW, Uri.parse("geo:0,0?q=" + result.get(0)));
        startActivity(requestIntent);
    }

    super.onActivityResult(requestCode, resultCode, data);
}
```

くみの基礎を取り上げました。Intentの機能を利用することで、音声認識を利用したアプリを比較的簡単に実装することが可能です。そのほかにも、応用がしやすいシンプルな設計となっているため、さまざまなアプリとの連携が簡単に可能です。しくみは今回学んだ内容を理解すれば、きっと使いこなせるようになると思いますのでぜひ挑戦してみてください。

次回は「データの保存」をテーマに取り上げたいと思いますので、お楽しみに！SD

重村 浩二（しげむら こうじ）　日本Androidの会 運営委員 中国支部長

日本Androidの会にて運営委員、中国支部長として毎月山口県・広島県を中心に自作アプリの発表やハッカソン、ハンズオンなどを中心とした勉強会を精力的に開催。個人としても、雑誌への寄稿などを中心に活動中。
URL http://buildbox.net/　Twitter @shige0501

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15回 PCIパススルー その1「PCIパススルーとIOMMU」

浅田 拓也（ASADA Takuya）Twitter @syuu1228

はじめに

前回・前々回は、総集編として2回にわたり仮想化システムの全体像をふりかえりました。

今回は、PCIパススルーについて解説していきます。

PCIのおさらい

PCIパススルーの解説を行う前に、まずは簡単にPCIについておさらいしましょう。

PCIデバイスが持つID

PCIデバイスはBus Number・Device Numberで一意に識別され、1つのデバイスが複数の機能を有する場合はFunction Numberで個々の機能が一意に識別されます。

Linuxで"lspci -nn"を実行したときに出力の左端に表示される「aa:bb:c」のうち、aaがBus Number、bbがDevice Number、cがFunction Numberにあたります（**図1**）。

さらに、そのデバイスがどこのメーカーのどの機種であるかという情報はVendor ID・Device IDで表され、この情報によってOSはロードするドライバを選びます。Linuxでlspci -nnを実行したときに出力の右端に表示される「dddd:eeee」のうち、ddddがVendor ID、eeeeがDevice IDにあたります。

PCIデバイスが持つメモリ空間

これらのデバイスはPCI Configuration Space、PCI I/O Space、PCI Memory Spaceの3つのメモリ空間を持ちます。PCI Configuration Spaceはデバイスがどこのメーカーのどの機種であるかを示すVendor ID・Device IDや、PCI I/O Space・PCI Memory Spaceのマップ先アドレスを示すBase Address Register、MSI割り込みの設定情報など、デバイスの初期化とドライバのロードに必要な情報を多数含んでいます。

PCI Configuration Spaceにアクセスするには、次のような手順を実施する必要があります。

①デバイスのBus Number・Device Number・Function Numberとアクセスしたい領域のオフセット値をEnable BitとともにCONFIG_ADDRESSレジスタ[注1]にセットする（CONFIG_ADDRESSレジスタのビット配置は**表1**のとおり）

②CONFIG_DATAレジスタ[注2]に対して読み込みまたは書き込みを行う

OSはPCIデバイス初期化時に、Bus Number・Device

注1）PCではCONFIG_ADDRESSレジスタはI/O空間の0xCF8にマップされています。

注2）PCではCONFIG_DATAレジスタはI/O空間の0xCFCにマップされています。

▼図1 lspci 実行例

```
$ lspci  -nn|grep IDE
00:1f.2 IDE interface [0101]: Intel Corporation 82801JI (ICH10 Family) 4 port SATA IDE
Controller #1 [8086:3a20]
00:1f.5 IDE interface [0101]: Intel Corporation 82801JI (ICH10 Family) 2 port SATA IDE
Controller #2 [8086:3a26]
```

Numberをイテレートして順にPCI Configuration SpaceのVendor ID・Device IDを参照することで、コンピュータに接続されているPCIデバイスを検出できます。

PCI I/O SpaceはI/O空間にマップされており（**図2**）、おもにデバイスのハードウェアレジスタをマップするなどの用途に使われているようです[注3]。

PCI Memory Spaceは物理アドレス空間にマップされており、ビデオメモリなど大きなメモリ領域を必要とする用途に使われているようです[注4]。

どちらの領域もマップ先はPCI Configuration SpaceのBase Address Registerを参照して取得する必要があります。

PCIデバイスにおけるDMA機能

PCIデバイスが持つメモリ空間に対して読み書きを行うことで、OS上のデバイスドライバとデバイスの間でデータをやりとりできます。具体的には、I/O空間に対してならin/out命令を発行、物理アドレス空間ならアドレスに対してmemcpy()のような処理を行うことでデータを転送します。しかしながら、この方法ではブロックデバイスや高速なNICなど、短い時間に大量のデータを転送する用途ではデータ転送を行う度にCPUが占有されてしまい、十分な性能が得られません。このようなPCIデバイスでは**DMA機能**が使用されます。

例として、82583V GbE Controller(e1000e)のパケット送信機能におけるDMAの使われ方を見てみましょう。**図3**に82583Vの送信用リングバッファと4種類のハードウェアレジスタ（TDBA、TDLEN、TDH、TDT）の関係を示しました。

TDBAはリングバッファの先頭アドレスを指します。TDLENはリングバッファ長を示します。TDHはNICがリングバッファのどこまで送信完了したかを示します。TDTはドライバがリングバッファのどこまでパケットを積んだかを示します。

次にパケット送信の手順を示します。手順で用いる番号は**図3**の番号に対応しています。

1. パケット送信処理を行う直前のレジスタおよびリングバッファの状態
2. ドライバはパケットを受け取り、TDTの次のエントリにパケットバッファのアドレスを書き込む
3. ドライバはTDTを1つ進めて、NICへ送信可能な

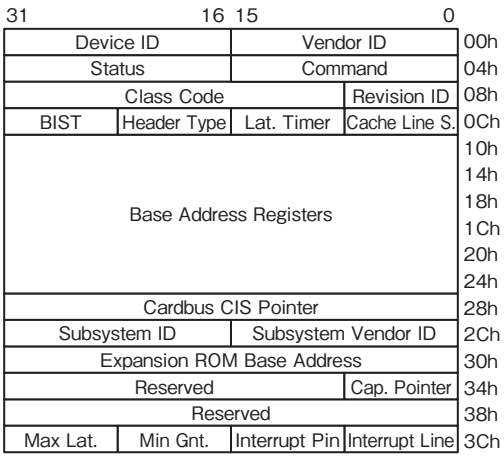
注3） PCではI/O空間にマップされますが、ほかのアーキテクチャではメモリマップされる場合もあると思われます。

注4） 必ずしもこのように使い分けられているわけではなく、ハードウェアレジスタのマップにPCI Memory Spaceを使用するデバイスも存在します。

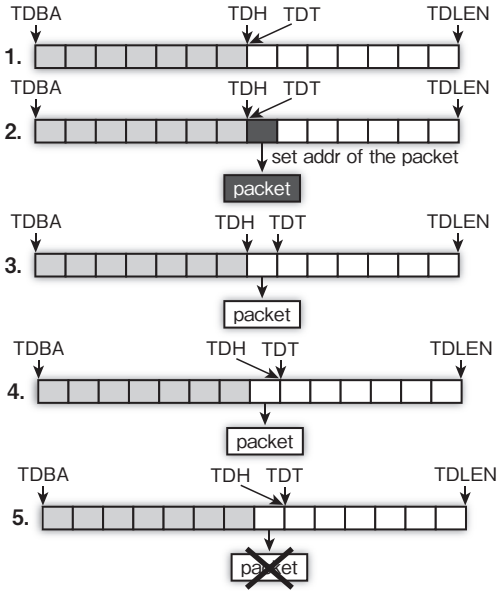
▼表1　CONFIG_ADDRESSレジスタ

bit	name
31	Enable Bit
30-24	Reserved
23-16	Bus Number
15-11	Device Number
10-8	Function Number
7-2	Register offset

▼図2　PCI Configuration Space



▼図3　82583V GbE Controllerのパケット送信機能



パケットがあることを伝える

4. NICはTDTへの書き込みを受けて、パケットをメインメモリからNIC上のバッファへDMA転送し、送出する。送信が終わったらTDHを1つ進めて送信完了割り込みを起こす
5. ドライバは送信完了割り込みを受け、送信済みパケットバッファ（TDHより手前にあるパケットすべて）に割り当てられたメモリを解放する

上述の例では話をわかりやすくするため、登場するパケットは1つだけにしましたが、実際にはNICの送信状況にかかわらずドライバはどんどんリングバッファへパケットを積んでいきます[注5]。NICは順にパケットをDMA転送して、送信が完了したら送信完了割り込みを行います。ドライバがTDTへ書き込んでからNICが送信を完了するまでの間、OSは送信完了を待つ必要はありません。この間、OSはほかの処理にCPUの時間を使えます。送信に使ったパケットバッファを片付けるために、OSはNICからの送信完了割り込みを使います。

PCIデバイスのパススルー

PCI Configuration Spaceのパススルー

ゲストOSによるCONFIG_ADDRESSレジスタとCONFIG_DATAレジスタへのアクセスをVMExit Reason 30（I/O Instruction）のVMExitを用いてVMMでハンドルします[注6]。

VMMはCONFIG_ADDRESSレジスタに設定されたBus Number・Device Number・Function Numberからどのデバイスへアクセスが来たのかを識別し、パススルー対象デバイス宛てであれば、実デバイスのConfiguration Spaceの指定オフセットへアクセスを行います。書き込みであればCONFIG_DATAレジスタの値を実デバイスへ書き込み、読み込みであれば実デバイスから読み込んでCONFIG_DATAレジスタから返します。

PCI I/O Spaceのパススルー

ゲストOSによるパススルーデバイスのPCI I/O Spaceへのアクセスも、同じくVMExit Reason 30（I/O Instruction）のVMExitを用いてVMMでハンドルします。このとき対象デバイスが持っているI/Oポートの範囲はPCI Configuration SpaceのBase Address Registerで定義されており、VMMはこの値を記憶しておくことでどのデバイスへのI/O命令であったか判別できます。

VMMはゲストからデバイスのポートへI/O命令を受け、同じポートへI/O命令を発行し結果をゲストに返します。あるいは、ゲスト側とホスト側でI/Oポートの番号が一致している場合[注7]は、前述の方法をとらずにVMCSのVM-Execution Control FieldsにあるI/O-Bitmapで対象となるポート番号のビットを0に設定することでVMExitを起こさずにPCIデバイスへアクセスできます。

PCI Memory Spaceのパススルー

ゲストOSによるPCI Memory Spaceへのアクセスは、ゲストマシンの物理メモリ空間のページ割り当てを設定することで実現します。本連載の『第3回I/O仮想化「デバイスI/O編」』にて、EPTによるメモリマップドI/Oのハンドリング方法として、デバイスがマップされたアドレスへのアクセスが発生したときにVMExit reason 48（EPT violationで、VMExitさせる方法）を示しました[注8]。

このように無効なページを作ってVMExitさせるのではなく、デバイスがマップされたページを有効なページとして設定[注9]し、実デバイスのメモリマップされたエリアをマップします。これにより、ゲストOSはVMExitされることなく直接実デバイスのPCI Memory Spaceに対してアクセスできます。

割り込みのパススルー

実PCIデバイスからの実割り込みはVMExit

注5）ただし、リングバッファが一杯になってしまったら一時的に送信を抑制するなどの措置をとります。
注6）詳しくは第3回 I/O仮想化「デバイスI/O編」を参照。
注7）VMMが独自のBase Address Registerを用意しておらず、実デバイスのものをそのままゲストに見せている場合。
注8）シャドーページングのときも同じ要領で設定ができますが、今回は解説を割愛します。
注9）Read accessビット・Write accessビットをともに1にします。

reason 1（External Interrupt）を生じさせ、VMMによって[注10]ハンドルされます。VMMはこの割り込みを処理するためのパススルー専用割り込みハンドラをあらかじめ登録しておく必要があります。割り込みハンドラはこの割り込みを受け付け、ゲストOSへ割り込みが届いたことを伝えるために、ゲストマシンのLocal APICレジスタを更新します。そしてVMCSのVM-entry interruption-information fieldへ割り込みをセットしてVMEntryします[注11]。これにより、VMMで受け取った割り込みがゲストへ伝えられ、ゲストOSが割り込みハンドラを実行できます。

PCIパススルーでDMA転送時に生じる問題

ここまで、PCIメモリ空間や割り込みは一定の手順を踏めばパススルーできるということを示してきましたが、DMAでは1つ困った問題が生じます。

PCIデバイスはDMA時のアドレス指定にホスト物理アドレスを使用します。通常、物理メモリ領域の全域にアクセスできます。もちろん、PCIデバイスはデバイスを使用しているOSが仮想化されていることなど知りません。

この状態で、**図4**のゲストA上のドライバがDMA先アドレスとしてゲスト物理ページの2番を指定すると何が起こるでしょうか？　PCIデバイスはDMAリクエスト元のOSが仮想化されていることを知らないので、ホスト物理ページの2番へDMA転送を行います。結果、PCIデバイスはゲストAのメモリ領域の範囲外にデータを書き込んでしまいます。

これにより、別のゲストマシンやVMMのメモリ領域を破壊してしまいます。かといって、ゲストOSは自分が持つゲスト物理ページとホスト物理ページの対応情報を持っていないので正しいページ番号をドライバに与えられません。たとえ持っていたとしても、ゲストOSが悪意を持ってゲストマシンに割り当てられている範囲外のページ番号を指定することで他のゲストマシンやVMMのメモリ領域を破壊できるという問題が解決しません。

そこで、**物理メモリとPCIデバイスの間にMMUのような装置を置きアドレス変換を行う**方法が考え出されました。このような装置を**IOMMU**と呼びます。

DMA転送時にアドレス変換を行うことで、**図4**の例ではゲストAのメモリ領域の範囲外へデータを書き込んでしまっていたパススルーデバイスが正しいページへデータを書き込めるようになります（**図5**）。

Intel VT-dは、このような機能を実現するためにチップセットへ搭載されたIOMMUです。VT-d対応PCでは、VMMがIOMMUへ変換テーブルを設定してアドレス変換を行い、ゲストマシンへの安全なデバイスパススルーを実現できます。

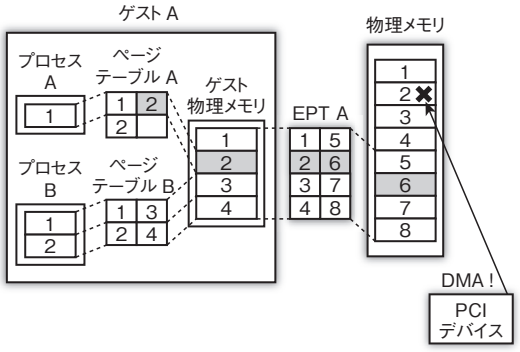
まとめ

今回は、PCIのおさらいとPCIパススルーの実現方法、IOMMUについて解説しました。次回は、Intel VT-dの詳細について解説します。SD

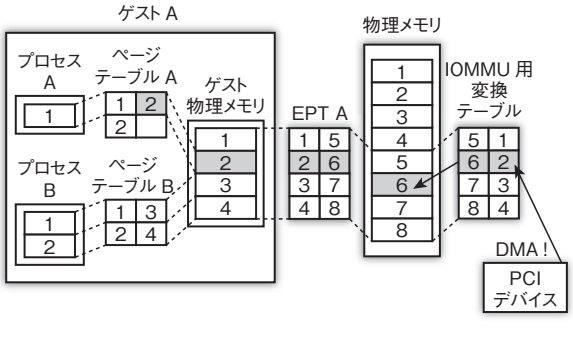
注10） KVM・BHyVeのようにホストOS上でVMMが動く場合はホストカーネルによって違います。

注11） 詳しい仮想割り込みのセット方法は第5回 I/O仮想化「割り込み編・その2」を参照。

▼図4　PCIパススルーでDMA転送時に生じる問題



▼図5　IOMMUを用いたDMA時のアドレス変換



テキストデータならお手のもの

開眼シェルスクリプト

USP友の会／産業技術大学院大学　上田 隆一 UEDA Ryuichi　Twitter @ryuichiueda

最終回

コマンドを定期的に実行させる ――cronとシェルスクリプトの合わせ技

はじめに

「みなさん、定時出勤、定時退社してますか？」などと最初から挑発ともとれるような発言で炎上しそうですが、今回の本連載は定時に何かをマシンにさせるcron、具体的にはcrontabの使い方を扱います。どうか、右手にお持ちの日本刀を鞘に戻してください。

cronは、実行したい日時を指定しておけば、その日時にプログラムを自動実行してくれるしくみです。Macを含むUNIX系のOSではほとんどの場合、最初から動いており、crontabというコマンドを使って定期的に動作させたいプログラムを指定するだけで、そのプログラムが動くようになっています。

本連載では2012年の10月号で使った例があります。そのときはさらっと説明しましたが、cronには少し癖みたいなものがあるので、落とし穴にはまるとなかなか思うようになりません。この癖はシェルスクリプトを間に挟むことで緩和できます。今回はほんのさわりだけですが、基本的な方法を示します。

ところで定期実行というと、昔、カントという人物がいて、あまりに時間に忠実で、カントが散歩で家の前を通る時刻に合わせて家の人が時計を合わせたなどという逸話が残っています。cronもそれくらいは正確です（当たり前）。ついでにこの人物が言い放ったことを書いておきます。

我が行いを見習えと、誰にでも言い得るよう行為せよ。――イマヌエル・カント

先生、無理です。

話が逸れたついでに報告しますと、本連載は今回で終了です。当初、ネタが地味なだけに6回連載して様子をみるという話でしたが、シェルスクリプト＝UNIXという筆者の勝手な拡大政策によって気づけば今回で24回目（ちょうど2年）です。現在の風潮である、みんなで一所懸命に馬鹿でかいブラックボックスの使い方を勉強し、みんなで愛と情熱をもってブラックボックスのバージョンアップ地獄をフォローしていくという、おおよそ工学的アプローチとは思えない騒ぎからのコペルニクス的転回を狙い、今後も寒風の中裸一貫、地味にシェルスクリプト活動をしていく所存です。あ、「コペルニクス的転回」も、カントの言った言葉です。うまくまとまりました。

cronとcrondとcrontabの関係

さて、まず用語の整理から。自分でインストールしなくても、だいたいの環境においてはcronというしくみが動いています。cronはしくみの名前であって、実際には、crondというサービスが最初から後ろで動いています。crondというのは、cronのデーモンという意味ですね。ただ、どの環境でもcrondの名前がcrondであるとは限らず、**図1**のようにcronになっていたりします。ややこしや。

crontabは、cronで定期実行するプログラムを確認したり、設定したりするためのコマンドです。crontabで設定した内容はどこかにファイルで保存されているのですが、このコマンドを通している限りはどこにあるか気にしなくてかまいません。あるユーザでcrontabを呼び出し、実行内容を書き込むと、その実行内容はそのユーザで動作します。そしてcrontabに書いた自動実行のリストは、crondを再起動しなくてもすぐに有効になります。いつもcronを使っている人でも、crondを再起動したという経験は少ないかと思います。筆者もありません。

crontabの使い方

crontabの使い方ですが、まずは設定内容の編集方法から説明します。

```
$ crontab -e
```

と打つと、エディタ(viかVim)が立ち上がります(Macの場合は注意があるので後述します)。たとえばここに次のように打ってみましょう[注1]。

```
* * * * * touch /tmp/aaaa
```

これで普通のviの操作で保存して終了します。ちゃんと登録されているかどうかは、**crontab -l**で確認できます。

```
$ crontab -l
* * * * * touch /tmp/aaaa
```

1分くらい待って/tmp/aaaaができたらcronがうまく働いています。さらに、1分ごとにlsを打ってみると、タイムスタンプが変化している様子がわかります(**図2**)。なぜそうなるかは後から説明しますが、cronが1分ごとにtouchを起動して/tmp/aaaaのタイムスタンプを更新しているからです。

今度は設定を消してみます。**crontab -e**で編集しても良いのですが、**crontab -r**とやると、設定が全部消えます。

```
$ crontab -r
$ crontab -l
crontab: no crontab for ueda
```

これは「**crontab -r**で消しましょう」と言うよりは、「**crontab -r**を押すとたいへんなことになるぞ！」という注意の意味で紹介しました。

ファイルでcrontabの内容を管理

さて、crontabはMacでも使えますが、**crontab -e**で編集した内容が反映されないという現象が発生します。どうもVimと相性が悪いようで

▼図2　1分ごとにタイムスタンプが変化している

```
$ ls -l /tmp/aaaa
-rw-r--r-- 1 ueda wheel 0  9 22 16:24 /tmp/aaaa
$ ls -l /tmp/aaaa
-rw-r--r-- 1 ueda wheel 0  9 22 16:25 /tmp/aaaa
```

▼図1　環境によってcrondが動いていたり、cronが動いていたりする

```
CentOS環境
$ ps aux | grep cron | grep -v grep
root      1397  0.0  0.0  20408   452 ?        Ss   Feb04   7:25 crond

OS X環境
$ ps aux | grep cron | grep -v grep
root           10058   0.0  0.0  2432784    156   ??  Ss   日03PM   0:00.39 /usr/sbin/cron

Ubuntu環境
$ ps aux | grep cron | grep -v grep
root       748  0.0  0.0  19112   848 ?        Ss   Sep18   0:01 cron
```

注1) viが使えないと苦労しますが、その場合はvimtutorという練習用コマンドがありますので、まずはそちらを練習しましょう。

す[注2]。Vimの設定ファイルをいじると解決するようですが、ここではもうちょっと確実な方法を示しておきます。

まず、名前はなんでも良いので、次のようなファイルを自分で作ります。筆者はホームの下にetcというディレクトリを作って、その下にcrontab.confという名前で作りました。

```
$ cat crontab.conf
* * * * * touch /tmp/aaaa
```

crontabにこのファイルを読み込ませます。

```
$ crontab crontab.conf
```

-lオプションで確認しましょう。

```
$ crontab -l
* * * * * touch /tmp/aaaa
```

こうやれば、crontabからVimを呼び出したときに起こる不具合とは無縁です。また、設定を書くのに好きなエディタを使えます。

これを応用すると、**図3**のようなこともできます。これは同じものを書き出したり読み出したりしているだけで、まったく意味がないのですが、これを知っていると、別のサーバや別のユーザにcronの設定を簡単に移せます。**crontab -r**をやらかしても、またファイルを読ませれば復旧できます。ただし、この方法には欠点が1つあって、crontab.confを書いて満足してしまい、読み込ませることを忘れがちになります。ご注意を。

注2）http://d.hatena.ne.jp/yuyarin/20100225/1267084794やhttp://d.hatena.ne.jp/shunsuk/20120122/1327239513などで調査しました。

▼図3　crontabでリストを出し入れ

```
crontabの内容を書き出す
$ crontab -l > hoge
crontabに書き出した内容を戻す
$ crontab hoge
```

▼図4　crontabの書き方あれこれ

```
$ crontab -l
5 * * * * touch /tmp/aaaa   ←①
*/5 * * * * touch /tmp/bbbb   ←②
15,30 * * * * touch /tmp/cccc   ←③
30 14-20 * * 1 touch /tmp/dddd   ←④
```

コマンドの前の記号の意味

次に時刻の指定の方法を説明します。書式は**man 5 crontab**で調べられますので、ここでは最小限の説明をします。先ほどcrontabで指定した*** * * * * touch /tmp/aaaa**ですが、この*** * * * ***の部分が時刻の指定部分です。順番に、分・時・日・月・曜日の指定で、*はそれぞれ毎分、毎時、毎日……ということになります。つまりはワイルドカードです。この例では、毎分、**touch /tmp/aaaa**を行うという意味になります。最小単位が分ですので、最小の周期は1分ということになります。

時刻の指定の例を**図4**に一気に示します。上から順に、①毎時5分に実行したい、②5分ごとに実行したい、③毎時15分と30分に実行したい、④月曜日の14～20時まで毎時30分に実行したい、という意味になります。

曜日の数字は、日曜から土曜までを0から6で指定します。日曜は7と書いてもOKです。結局、次の点を押さえれば良いということです。

- スラッシュの後ろに数字を書くと、その数字の周期で実行
- カンマで数字を並べると、その数字に該当するときに実行
- ハイフンで数字をつなぐと、その範囲内で毎回実行

ハイフンについては、n-mと書いたら、nとmも含まれます。また、ハイフンとスラッシュの併用もできます。

Twitterへの自動ツイートにcronを使う

cronを使って何か作ってみましょう。今回もMacで試します。**図5**に環境を示します。

今回はTwitterで自動ツイートを行うプログラムを作りましょう。と言っても一からシェルスクリプトでbotを作るとたいへんですので、できあいのコマンドを使います。また、筆者が試した環境はOS X ServerではなくMacBook Airですので、サスペンド状態だったりネットワークに接続されていなかったりするとツイートできません。ただ、今回の内容をほかのUNIX環境に移植するのは簡単です。

ツイートコマンドの準備

https://github.com/ryuichiueda/TomoTool/blob/master/Twitter/usptomo-tweetから、筆者が作ったつぶやきコマンド(シェルスクリプト製!!)usptomo-tweetをダウンロードします。ダウンロードの方法がわからなかったら、画面をコピー&ペーストしてusptomo-tweetファイルに保存して、

```
$ chmod +x usptomo-tweet
```

としてください。コマンド名が長いので、別にtwと変更してもかまいません。

usptomo-tweet内部ではいろいろなコマンドを使っています。ほとんど標準的なものですが、nkf、curl、openssl コマンドあたりはインストールされているか確認してください。しょせん書きなぐりのシェルスクリプトですので、何か動かなかったら自分でログを見て直すくらいの気持ちでお願いします。

次に、鍵やトークンというものの設定を行います。https://dev.twitter.comに行って、ツイートしたいアカウントでログインします。ログインしたら「My applications」の画面、「Create an application」の画面に進み、必要事項を入力してください。アプリケーション名は何でも大丈夫です。必要事項の入力後、登録のボタンを押すと、「Consumer key、Consumer secret、Access token、Access token secret」が取得できます。普通に取得すると、Consumer keyもAccess tokenも、「Read only」になっているはずです。画面の指示に従って「Read and write」というアクセスレベルで再取得してください。ここら辺はややこしいのですが、説明し出すと長くなってしまうので、うまくWeb上で方法を見つけながらやってみてください。

取得できたら、**図6**のようなファイルをホームの下に置きます。これはusptomo-tweetに読み込ませるシェルスクリプトの一部ですので、シェルスクリプトの文法で書き、ファイル名も間違えないようにします。ホーム下に置くのが嫌なら、usptomo-tweetの中を書き換えます。

ツイートしてみる

これで準備OKです。テストしてみましょう。**図7**のように打ってみます。うまくいけば、**図8**

▼図5　実験環境

```
$ uname -a
Darwin uedamac.local 12.5.0 Darwin Kernel Version 12.5.0: Mon Jul 29 16:33:49 PDT 2013;
root:xnu-2050.48.11~1/RELEASE_X86_64 x86_64
```

▼図6　キーとトークンを書いたファイル

```
$ cat twitter.key
CONSUMER_KEY="aaaaaaaaaaaaaaaaaaaaa"
CONSUMER_SECRET="bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb"
ACCESS_TOKEN="000000000-ccccccccccccccccccccccccccccccccccccccccc"
ACCESS_TOKEN_SECRET="ddddddddddddddddddddddddddddddddddddddddddddd"
```

▼図7　端末からテストツイート

```
$ ./bin/usptomo-tweet 'test: 東東東南南南西西西北北北白白'
```

のようにネット上に投稿がツイートされます。

cronから使う（環境変数に気をつけて）

cronとusptomo-tweetを組み合わせて使ってみましょう。とは言うものの、cronには慣れていても慣れていなくてもいろいろな落とし穴があって、なかなか一発でうまくいきません。粘り強くいきましょう。ここではよく起こるミスを1つだけ、デバッグしながら紹介します。

まず、**図9**のようにしかけてみましょう。時刻は直近のものに合わせます。数分余裕を持ってしかけるようにと書いてあるサイトがありますが、おそらく余裕を持たせなくても大丈夫な環境がほとんどだと考えます。

時刻がきたら、/tmp/errorを見てみましょう。環境にもよりますが、筆者のMacでは**図10**のように失敗しました。「あれ？　nkfがインストールされていないのかな？」というところですが、先ほど端末から試したときにはうまくいっていたので、ここはパス（環境変数PATH）を疑います。crontabで**図11**のようにしかけます。ついでにLANGも調べてみましょう。

時刻がきたら/tmp/pathとtmp/langを見てみると、**図12**のようになっていました。nkfの場所は次のように/usr/local/bin/ですので、nkfが見つからずエラーが起きたようです。

```
$ which nkf
/usr/local/bin/nkf
```

cronで何かを動かそうとしてうまくいかない場合、たいていはパスが間違っているか、今の例のように環境変数が端末で使っているものと違うという問題に突き当たります。

さて、原因がわかったので対策を。まず、**図13**のようにcrontabに環境変数を設定する方法があります。

結果は掲載しませんが、これはうまくいきます。ただ、**図13**を見渡すとごちゃごちゃしていますし、usptomo-tweetのために通したパスがほかの設定にも影響します。筆者はあまり良い方向にいっているとは思えません[注3]。

ラッパーのシェルスクリプトを使う

ここでシェルスクリプトの出番です。環境変

注3）ついでに書いたMAILTO=""は、cronがログのメールを送ってくるのを防ぐための記述です。

▼図8　Twitterに投稿される（図7の実行結果）

▼図9　crontabにコマンドを直接書き込む

```
$ crontab -l
21 21 * * * /Users/ueda/bin/usptomo-tweet 'test: びろーん' > /dev/null 2> /tmp/error
```

▼図10　nkfが見つからないエラーが発生

```
$ less /tmp/error
（略）
/Users/ueda/bin/usptomo-tweet: line 33: nkf: command not found
（略）
```

▼図11　echoでcronに設定されている環境変数を調べる

```
$ crontab -l
23 10 * * * echo "$PATH" > /tmp/path
23 10 * * * echo "$LANG" > /tmp/lang
```

▼図12　環境変数の調査結果

```
$ cat /tmp/path
/usr/bin:/bin
$ cat /tmp/lang

（LANGには何も入っていない）
```

数やそのほかをすべてシェルスクリプトの中に押し込んでしまいます。ラッパーのシェルスクリプトは、ホーム下にbatchというディレクトリを作って、そこに置きます。**図14**に、例をお見せします。これはMacを閉じてしまうとツイートできないのを逆手にとって、夜にMacを開いていたら自虐ツイートするしくみです。

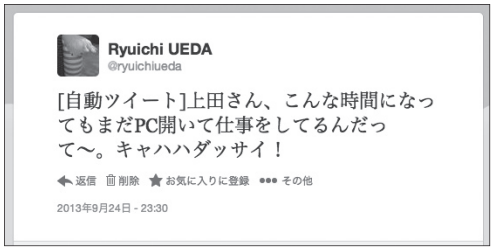
PATHにはnkfのある/usr/local/binと、usptomo-tweetのある/Users/ueda/binを指定します。また、**exec 2>**でこのシェルスクリプトの標準エラー出力、**exec >**で標準出力をリダイレクトしてファイルに残しておきます。**basename $0**は、このシェルスクリプトの名前(nightwork)になります。**図15**のようにちゃんと送信されました……。悲しいですね。もう寝ることにします。

終わりに

今回はcronとシェルスクリプトと組み合わせて、自動自虐ツイートを行う自動送信機能をMacBook Airに組込みました。シェルスクリプトという点では、最後の最後にちょっと出てきただけでしたが、PATHの明示的な指定など、これまでの連載で説明できなかったことを扱えました。最後に作ったnightworkを拡張していくと、たとえばブログの記事を紹介したり、リストから文言をランダムに選んでつぶやくボットを作ったりすることができます。ぜひ試していただければ。

冒頭でお伝えしたとおり、開眼シェルスクリプトは今回で最終回です。ご愛読、ありがとうございました。SD

▼図15　送信の確認(図14の実行結果)



▼図13　環境変数をcrontabで指定する方法

```
$ crontab -l
PATH=/usr/bin:/bin:/usr/sbin:/sbin:/usr/local/bin
MAILTO=""
35 10 * * * /Users/ueda/bin/usptomo-tweet 'test: びろーんぐ' > /dev/null 2> /tmp/error
```

▼図14　cronから呼び出すシェルスクリプトと設定

```
$ cat nightwork
#!/bin/bash -xv

PATH=/usr/local/bin:/Users/ueda/bin:$PATH
LANG=ja_JP.UTF-8

exec 2> /tmp/stderr.$(basename $0)
exec > /tmp/stdout.$(basename $0)

usptomo-tweet '[自動ツイート]上田さん、こんな時間になってもまだPC開いて仕事をしてるんだって〜。キャハハダッサイ！'

 実行できるようにしましょう
$ chmod +x nightwork
 crontabは次のようにセット
$ crontab -l
MAILTO=""
30 23 * * * /Users/ueda/batch/nightwork
```

Be familiar with FreeBSD.

チャーリー・ルートからの手紙

第2回 ◇ top(1) ~システム管理の第一歩、統計情報を読み取れていますか?~

◉著者プロフィール

後藤 大地(ごとう だいち)

BSDコンサルティング(株)　取締役／(有)オングス　代表取締役／FreeBSD committer

エンタープライズシステムの設計、開発、運用、保守からIT系ニュースの執筆、IT系雑誌や書籍における執筆まで幅広く手がける。カーネルのカスタマイズから業務に特化したディストリビューションの構築、自動デプロイ、ネットワークの構築など、FreeBSD/Linuxを含め対応。BSDコンサルティングでは企業レベルで求められるFreeBSDの要求に柔軟に対応。

システムモニタリングコマンド top(1)

システムの状態をモニタリングするためのコマンドの1つに**top(1)**があります(**図1**)。有名なコマンドですので、少なくとも一度は実行したことがあるかと思います。**top(1)**はインタラクティブに動作するモニタリング用のコマンドで、2秒ごとにシステムの統計情報とプロセスの動作状態一覧情報をアップデートしながら表示し続けます。デフォルトではプロセッサやメモリ使用量に関する情報を表示しますが、ディスク入出力の視点からのプロセス情報一覧を表示させることもできます。

top(1)にはさまざまなオプションが用意されています。コマンドにオプションを指定して実行することもできますし、インタラクティブに表示を切り替えることもできます。どのようなショートカットキーが利用できるかは[h]または[?]キーを押すことで表示させることができます。

top(1)コマンドはシステムに関する多くの情報を教えてくれますが、実は多くの方がプロセッサの使用率やメモリの使用量くらいしか注目しておらず、とくに上部に表示されているヘッダ部分の情報は気にしていないことが多いようです。**top(1)**のヘッダ部分に表示されているデータはシステムに関する多くのことを教えてくれます。今回はこのヘッダを説明します。

▼**図1　top(1)コマンドが出力するデータの例**

```
last pid:  9538;  load averages:  0.84,  0.86,  0.86     up 0+20:29:30  07:10:16
95 processes:  2 running, 93 sleeping
CPU:  0.2% user, 12.3% nice,  0.3% system,  0.0% interrupt, 87.1% idle
Mem: 907M Active, 3072M Inact, 10G Wired, 184M Cache, 1644M Buf, 1476M Free
ARC: 6125M Total, 99M MFU, 1768M MRU, 16K Anon, 164M Header, 4095M Other
Swap: 4096M Total, 4096M Free

  PID USERNAME    THR PRI NICE   SIZE    RES STATE   C   TIME   WCPU COMMAND
 1024 daichi        1  20    0  3266M 90332K select  6  10:12  1.17% Xorg
 1055 daichi       40  23    0   818M   325M uwait   6  31:04  0.00% firefox
  895 root          1  20    0 12088K  1396K select  1   4:40  0.00% powerd
 1086 daichi        1  20    0   269M 37220K select  0   4:13  0.00% lxpanel
 1045 daichi        4  20    0 65048K  9804K select  2   2:57  0.00% ibus-daemon
 1051 daichi        4  38    0   399M 52852K kqread  6   2:55  0.00% lxterminal
 1059 daichi        5  20    0   418M 63936K uwait   2   1:28  0.00% sylpheed
 1039 daichi        5  20    0   148M 96452K accept  1   1:08  0.00% mozc_server
 1047 daichi        3  20    0   709M 98208K kqread  2   0:58  0.00% pidgin
99744 daichi        6  20    0   515M 61304K pcmwrv  0   0:56  0.00% vlc
 1083 daichi        1  20    0   126M 16252K select  4   0:43  0.00% openbox
 1120 daichi        4  20    0   102M 10928K select  1   0:37  0.00% ibus-engine
 1070 daichi        1  20    0   343M 54296K select  1   0:35  0.00% python2.7
 1916 daichi        3  20    0   199M 51116K accept  1   0:33  0.00% mozc_render
 1119 daichi       10  20    0   116M 14812K uwait   5   0:30  0.00% VBoxSVC
  516 root          1  20    0 14264K  1464K select  6   0:24  0.00% moused
```

各コマンドのうしろについている括弧書きの数字は、manコマンドで見ることができるマニュアルに記載されている章番号を表しています。

top(1)コマンドのヘッダはカーネルの統計情報

top(1)コマンドのヘッダ(**図2**)は5行ないしは6行で構成されています。ZFSを使っているシステムでは6行、そうでないシステムでは5行で表示されます。

この部分はシステムの稼動時間、負荷状況、プロセッサの利用状況、メモリの使用状況、スワップの利用状況が表示されています。この部分をチェックすることで、対象のシステムがどのような負荷状況にあり、パフォーマンス上のボトルネックが存在する場合にどのハードウェアを強化すればよいかなどの判断材料とすることができます。次から各行を個別に見ていきましょう。

ロードアベレージとアップタイム

1行目にはロードアベレージとアップタイム(稼動時間)、それに現在時刻が表示されています(**図3**)。「last pid: 9538」はシステムで最後に実行されたプロセスのプロセスIDを示しています。「load averages: 0.84, 0.86, 0.86」がロードアベレージ、「up 0+20:29:30」がアップタイム、「07:10:16」が現在時刻です。

このヘッダからは「7時10分16秒におけるシステムの稼動時間は20時間29分30秒であり、ロードアベレージは0.85前後と軽めの負荷状態。最後に実行されたプロセスのプロセスIDは9538」ということが読み取れます。アップタイムの先頭が「0+」となっているのは日付を表しており、24時間が経過するとここが1になります。日数+時間という表記になっているわけです。

ロードアベレージというのは、システムにおいて実行可能状態(RUNNABLE)にあるプロセスの個数の平均値のことです。3つ表記されているのは、左から順に1分間の平均値、5分間の平均値、15分間の平均値となっているためです。この3つの値を比べることで、システムが一定の負荷で動作しているのか、負荷が高まる傾向にあるのか、逆に負荷が減る傾向にあるのかが判断できます。

平均値ですが、計算は指数加重移動平均で計算されていますので、その期間の平均値といっても、より現在に近いほうの値が優先的に反映されるしくみになっています。どの程度までロードアベレージが上がるかは、どのような用途でシステムを組んでいるかによって変わります。ちょっとしたシステムでも数百、大規模なサーバであれば1,000を超えるロードアベレージが出ることもあります。

ロードアベレージが常に高いようなシステムでは、プロセッサをより高速なものに交換したり、物理コア数の多いプロセッサに交換することで性能の向上が期待できます。

プロセスの数と実行状態

2行目にはプロセスの数と実行状態が表示されています。**図4**の表示であれば95のプロセスが動作しており、うち実行可能状態(RUNNABLE)のプロセスは2個、スリープ状態のプロセスが93個あることがわかります。ここから、このシステムはかなり余裕のある状態にあることが読み取れます。

ただし、ここで表示されているプロセスの個数は、top(1)コマンドが表示の対象としているプロセスの個数であることに注意してください。top(1)コマンドにオプションを指定して表示対象を変更する

▼**図4　プロセスの数と実行状態(図2の2行目)**

```
95 processes:  2 running, 93 sleeping
```

▼**図2　top(1)コマンドのヘッダ部分(図1より)**

```
last pid:  9538;  load averages:  0.84,  0.86,  0.86     up 0+20:29:30  07:10:16
95 processes:  2 running, 93 sleeping
CPU:  0.2% user, 12.3% nice,  0.3% system,  0.0% interrupt, 87.1% idle
Mem: 907M Active, 3072M Inact, 10G Wired, 184M Cache, 1644M Buf, 1476M Free
ARC: 6125M Total, 99M MFU, 1768M MRU, 16K Anon, 164M Header, 4095M Other
Swap: 4096M Total, 4096M Free
```

▼**図3　ロードアベレージとアップタイム(図2の1行目)**

```
last pid:  9538;  load averages:  0.84,  0.86,  0.86     up 0+20:29:30  07:10:16
```

と、ここの値も切り替わります。

プロセッサの使用状況

3行目にはプロセッサの使用状況が表示されています(**図5**)。「user」はユーザ空間で処理が実行された割合、「system」はカーネル空間で処理が実行された割合、「nice」はユーザ空間で処理された処理のうち、スケジューラプライオリティ値(nice値)がデフォルトの値から変更されたプロセスが実行された割合、「interrupt」は割り込みモードで実行された割合、「idle」はアイドル時間の割合となります。

カーネルはidleというカーネルプロセスをもっとも低い優先度で実行しています。実行するプロセスがなくなると、このidleというカーネルプロセスに処理が移ることになります。idleの実行時間を集計することで、アイドル時間(なにも処理していない時間)がわかることになります。

userやsystemの割合が高いシステムはプロセッサの性能を使いきっていることになります。こうしたシステムではプロセッサを高速なものに置き換えることで性能の向上が期待できます。

割り込みを処理するinterruptカーネルプロセスにはかなり高い優先度が割り当てられます。interruptの割合が高いシステムでは割り込み処理にかなりの時間が割かれていることになり、userやsystemへリソースを割り当てることができていない可能性があります。こうしたケースでは、プロセッサおよび周辺チップも含めて製品を交換することで性能の向上が期待できます。

メモリ使用状況

4行目には物理メモリの使用状況が表示されます(**図6**)。各項目が表している内容を**表1**にまとめます。単位表示は**表2**のとおりです。

一見するとFreeの値が大きいほうが余裕があるように思えますが、ハードウェアの性能を使い切るという観点からはFreeが少なくBufが大きいほうが適切な状態だと言えます。カーネルはメモリに余裕がある場合、可能な限りディスクキャッシュとして利用しようとします。

メモリを積めば積むほどシステムの起動には時間がかかるようになりますし、故障率も上がります。BufとFreeの値などを見ながら対象システムにおいて適切なメモリサイズを設定することができるようになると、上手にハードウェアの選定ができるようになります。

ZFS ARCのメモリ使用量

5行目に表示されているのはZFS ARCのメモリ使用量です(**図7**)。ZFS ARCはZFSで使用されるキャッシュメモリです。ZFSを使っていないシステムではこの行は表示されません。各項目が表してい

▼表1　物理メモリの状態

項目	意味
Active	アクティブに使われているメモリのサイズ
Inact	アクティブではないメモリのサイズ
Wired	固定化されているメモリのサイズ(BIOレベルでのキャッシュされたファイルデータページを含む)
Cache	ただちにリロケーション可能なデータキャッシュ用のメモリのサイズ
Buf	BIOレベルでディスクキャッシュのために使われているメモリのサイズ
Free	フリーなメモリのサイズ

▼表2　メモリサイズの単位

単位	意味
G	ギガバイト
M	メガバイト
K	キロバイト
%	パーセンテージ

▼図5　プロセッサの使用状況(図2の3行目)

```
CPU:  0.2% user, 12.3% nice,  0.3% system,  0.0% interrupt, 87.1% idle
```

▼図6　メモリの使用状況(図2の4行目)

```
Mem: 907M Active, 3072M Inact, 10G Wired, 184M Cache, 1644M Buf, 1476M Free
```

▼図7　ZFS ARCのメモリ使用量(図2の5行目)

```
ARC: 6125M Total, 99M MFU, 1768M MRU, 16K Anon, 164M Header, 4095M Other
```

る内容を表3にまとめます。

ZFSは便利なストレージシステムですが、そのしくみ上、多くのメモリを消費します。さらに性能をあげようとした場合にはARCをうまく活用してあげる必要があります。

スワップ使用量

6行目（ZFSを使っていない場合は5行目）にはスワップの使用状況が表示されます（図8）。「Total」がスワップとして割り当てられている表域のサイズ、「Free」が使われていないサイズです。

現在ではメモリを豊富に積むことができますので、基本的には「スワップは起こらないようにする」のがシステム構築の基本です。大きくスワップを消費している場合、常駐しているアプリケーションがメモリリークを起こしている（たとえばデータベースのメモリリークが原因で徐々にスワップの使用量が広がっていく）などの可能性があります。

カーネルがパニックした場合など、コア情報をスワップ領域に書き出しますので、スワップはそうした用途に使う領域だと割りきって設定するのも1つの方法です。

コアごとの情報

top(1)のヘッダにはプロセッサとしての平均値が表示されています。マルチコア／メニーコア時代の現在では、平均値ではなくコアそれぞれの値を知りたいことがあります。すべてのコアをうまく活用できているか知りたい場合などです。

top(1)コマンドに-Pを指定して実行すると、プロセッサとしての平均値ではなく、コアそれぞれの値を表示させることができます（図9）。

▼図8　スワップの使用状況（図2の6行目）

```
Swap: 4096M Total, 4096M Free
```

▼表3　ZFS ARCメモリの状態

項目	意味
Total	ZFS ARCのために固定化されているメモリのサイズ
MRU	もっとも最近使われたデータを保持しているARCメモリのサイズ
MFU	もっとも頻繁に使われたデータを保持しているARCメモリのサイズ
Anon	フライトデータを保持しているARCメモリのサイズ
Header	ヘッダを保持しているARCメモリのサイズ
Other	それ以外のARCメモリのサイズ

top(1)を使ってシステムをよく知ろう!

今回はtop(1)のヘッダ部分の読み方を紹介しました。この部分が読めるようになることで、カーネルやシステムがどのような状態にあるのかがよくわかるようになります。状態が把握できれば適切な対応もできるようになります。

FreeBSDはベースシステムにおいてシステム管理に必要になるほぼすべてのコマンドを提供しています。サードパーティ製のモニタリングソフトウェアの導入を検討する前に、まずこうしたコマンドの使い方を知り、システムをよく知ることがスマートなシステム管理につながっていくと言えます :) SD

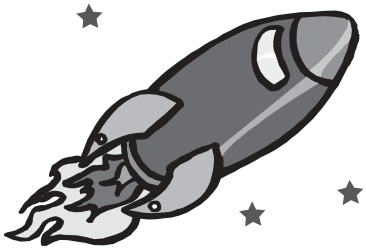
▼図9　コアごとの情報表示：top -P

```
last pid: 37703;  load averages:  1.10,  1.10,  1.08
up 0+21:30:23  08:11:09
95 processes:  2 running, 93 sleeping
CPU 0:  0.4% user,  0.0% nice,  0.0% system,  0.4% interrupt, 99.2% idle
CPU 1:  0.4% user,  0.0% nice,  0.4% system,  0.0% interrupt, 99.2% idle
CPU 2:  0.4% user,  2.4% nice,  0.0% system,  0.0% interrupt, 97.2% idle
CPU 3:  0.0% user, 14.7% nice,  0.4% system,  0.0% interrupt, 84.9% idle
CPU 4:  0.4% user, 68.9% nice,  2.8% system,  0.0% interrupt, 28.0% idle
CPU 5:  0.0% user,  0.0% nice,  0.4% system,  0.0% interrupt, 99.6% idle
CPU 6:  0.0% user,  0.0% nice,  0.0% system,  0.0% interrupt,  100% idle
CPU 7:  0.4% user, 10.2% nice,  1.6% system,  0.0% interrupt, 87.8% idle
Mem: 840M Active, 3458M Inact, 10G Wired, 180M Cache, 1647M Buf, 929M Free
ARC: 6087M Total, 222M MFU, 1717M MRU, 107K Anon, 165M Header, 3983M Other
Swap: 4096M Total, 4096M Free
```

10

Debian Developer　やまねひでき　henrich@debian.org

GSoCの成果発表& Debian 8の動向



Debian Hot Topics

はじめに

かなり寒さが増してきましたが、みなさん体調など崩されていないでしょうか？　筆者は、気温の変化についていけなかったのか、ストレスのせいなのか、帯状疱疹（ほうしん）という病気になってしまって原稿の〆切をぶっちぎってしまいました。この原稿が12月号にちゃんと載っていたら、それは担当編集氏の忍耐の賜物です。

夏を振り返る ～GSoCの結果

この夏も熱かったFLOSSのイベントといえば「Google Summer of Code」（以下、GSoC）[注1]です。今年もDebianプロジェクトが参加し、16個のプロポーザルのうち15個を達成という、かなり良い成績を収めています。ここではそのうちのいくつかを取り上げてみましょう。

★PTS rewrite

Debianには「Package Tracking System」（以下、PTS）[注2]という、Debianの各パッケージについて各種情報を一元的にまとめて参照できるようにしているサイトがあります。Debianの長い歴史の中で、このサイトは古い旅館のように増築に増築を重ねたような状態になっていたのですが、これをPythonのWebフレームワークDjangoを使ってスクラッチでモダンなサイトに書き直そうというのが、今年のGSoCで採択されたのでした。

現在動作中の旧PTSで参照可能な情報のうちの一部はまだ欠落しているものの、ほぼ問題なく動作している新システムがhttp://pts.debian.netで参照できます。機能以外では動作に関してPerlで書かれた現状のサイトよりも消費メモリが大きいことが悩みのようです。しかし、大きな問題ではないので、そのうち書き換えられた新しいPTSサイトの運用が始まるのが期待されます。最終的には「distro-tracker」という名前で、tracker.debian.orgとして動作する予定です。

★scan-build the debian archive

このプロポーザルの目的は2つです。

- 標準のGCC以外のコンパイラ（clang）でもパッケージのビルドができるようにする
- コードの静的解析を行って潜在的なバグを洗い出していく

clangでのパッケージビルドのチェックはすでに行われています[注3]が、現在は「clangのバージョンアップに併せてフルリビルドを行って問題を発見する」という運用で、パッケージがリポジトリ

注1）さまざまな団体が課題を出し、それに応募した学生を指導し、最終的に成果が出たらGoogleがスポンサーとなって賞金を払うというプログラム。

注2）URL http://packages.qa.debian.org

注3）URL http://clang.debian.net/

に入ってから実際にclangを用いてリビルドされるまでに若干のタイムラグが生じます。これをリアルタイムで処理するように改善してフィードバックと修正のサイクルを早めよう、というのが今回のGSoCでの目標の1つです。

もう1つの目的は、clangのstatic analyzerを利用してコンパイラが見つけられなかった問題を洗い出して[注4]修正していくことで、潜在的なバグを見つけてソフトウェア自体の品質を高めるというものです。Debianのみならず Upstreamプロジェクトにも良いフィードバックができることが期待されています。すでに動作するようにはなっているようですので[注5]、うまくインフラに組み込んで標準的に利用されるようになることが期待されます。

★ZFS on Linux integration

Sun MicroSystems社がSolaris用に開発したファイルシステム「ZFS」のオープンソース実装であるOpenZFS[注6]の1つ「ZFS on Linux (ZoL)」をDebianに取り込もう、というプロポーザルです[注7]。

ZFS on Linuxはローレンス・リバモア国立研究所を運営するローレンス・リバモア・ナショナル・セキュリティが主体となって開発しており、GPL2で提供されるSolaris Porting Layer(SPL)とCDDLのNative ZFS for Linuxの組み合わせで動作します。Linuxカーネルの上にSPLが載ってLinuxとSolarisの違いを吸収し、さらにその上にZFSが載る形になります(**図1**)。ZFSのコードがLinuxのGPL2とは互換性がないCDDLで提供されているため、Linux本体にはコードがマージできないのですが、ZFS on Linuxプロジェクトの見解では「カーネルと独立したバイナリモジュールとして配布するのには制限はない」とのことです[注8]。

Debianでは、このプロポーザルの成果がzfs-linuxパッケージとしてすでにNew Queueにアップロードされており、リポジトリに入るためのチェックを待っています[注9]。

★The OpenJDK and Debian

Java 7への移行のためにOpenJDK 6/7でのビルドを実施し比較して問題点を洗い出して修正していく、という作業を第一の目的として行われたプロポーザルです。結果として、無事主要なアーキテクチャはOpenJDK 7への移行を果たしました[注10]。

そして、もう1つの目的である「OpenJDK 8のパッケージング」が困難なため、findbugsを利用した静的解析とその関連ソフトウェアのパッケー

注4) とはいえ、関係者らのtweetを見ると同様の機能を持つ商用のCoverityなどと比べればまだまだだのようです。

注5) 現状の処理の様子は URL http://debile.debian.net/で確認できます。

注6) ZFS自体はOpenSolarisの一部としてCDDL(Common Development and Distribution License)のもとで公開されていました。しかし、OracleによるSunの買収後、OpenSolarisはクローズされ、ZFSもプロプライエタリソフトウェアとしてのみ提供されています。そこで、CDDLで提供されていたコードをベースにOpenSolaris派生ディストリビューションやFreeBSD、そしてその利用企業などが中心となってOpenZFSプロジェクトが立ち上げられました。詳細については URL http://open-zfs.org/を参照ください。

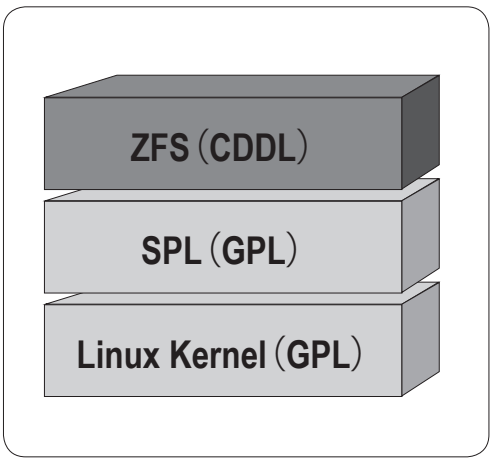
注7) なおFreeBSDではネイティブでZFSをサポートしていますので、現状でもすでにDebian GNU/kFreeBSDでZFSを利用できます。

注8) とはいえ、何かもめそうな気もするのですが……。

注9) URL http://bugs.debian.org/686447

注10) URL http://lists.debian.org/debian-devel-announce/2013/08/msg00000.html

▼図1　ZFS on Linux



ジングが実施されました。findbugsに若干のライセンス問題があるので、そちらが解決されればDebianへマージされることになるでしょう。

★Androidパッケージ

最後はちょっと軽めのものを。Android向けアプリケーション「DebianDroid」パッケージの作成が行われました。おもにメンテナ向けのソフトウェアで、手軽にパッケージ情報やバグの追いかけ、新規パッケージのチェックなどがスマートフォンからできるようになっています。パッケージはGoogle PlayやF-Droidリポジトリ[注11]からインストールが可能になっています。ソースコードもGitHub[注12]で入手できますので、気になった方はどうぞ。

Debian 8の開発フリーズ日が決定

Debian 8のリリースに向けて、来年2014年11月5日[注13]に開発をフリーズすることが発表されました[注14]。Debianでは、リリース前に新規のバージョンの流入を停止し、ひたすら既存のバグつぶしに入る「フリーズ」期間が存在します。これから推測するに、Debian 8「Jessie」は、おそらく2015年の2〜5月あたりがリリースだと思われます。

これまでの傾向として、リポジトリのパッケージ数の増大に伴って対応が必要なバグ数が増加してリリースが遅延することが問題となっていました。これに対する新しい取り組みとして前回でも取り上げた「package autoremovals」が実施され始めましたので、処置が必要な重大なバグがあるパッケージは、フリーズの実施時にはJessieにさほど存在していない可能性があります。バグ数が多くなければ、フリーズして間を置かずにリリースとなることを期待したいところです[注15]。

注意点としては、**バグがないパッケージでも、依存するパッケージにRCバグがあり、しかもキーパッケージではない場合、同様に削除される**ことがあります。場合によっては利用したいパッケージがいつの間にかDebian 8からは消えている……という問題が発生する可能性もありますので、今後の状況を見守りましょう。

Debian 8の目標

さて、そのDebian 8のリリースにあたって、開発目標にはどんな機能や変更が挙げられているのでしょうか。

- Native systemd support in every package with sysv scripts
- Hardening of ELF binaries(carry over from Wheezy)
- debian/rules to honor CC/CXX flags
- clang as secondary compiler
- piuparts clean archive
- Cross Toolchains in the archive
- Make the base system cross-buildable
- SELinux
- UTF-8

簡単にまとめると「クロスビルドで使いやすいように」「セキュリティを強化しよう」「systemdやclangなど、システムで取り得る選択肢を増やそう」というところでしょうか。毎度のことながら地味な作業が中心になります。ユーザの視点からは「どこが変わったの？」と思えるかもしれませんが、とくに意識することなく安心して利用できるのが安定版のいいところ、ということでご理解ください。

なお、これとは別にデスクトップとしては新しいバージョンのパッケージが導入されて各種

注11) Android用のFOSS(Free and Open Source Software)アプリケーションリポジトリ。URL https://f-droid.org/

注12) URL https://github.com/uberspot/DebianDroid

注13) 繰り返しますが「来年」です。お間違えのなきよう。

注14) URL http://lists.debian.org/debian-devel-announce/2013/10/msg00004.html

注15) バグが非常に多い場合、パッケージのバージョンが固定されたままリリースが延びて利用できるバージョンが古くなっていってしまうので、なるべく短いほうが望ましいです。

変更が起こるであろうことはご留意ください（unstableでの様子を見る限り、好むと好まざるにかかわらず、GNOMEはさらにシンプルな形になっていくと思われます）。

s390アーキテクチャの削除、ほかのアーキテクチャの今後

s390が削除されるアナウンスが出ました。とは言っても日本で使っている方はおそらく1人もいない、IBMのメインフレームで動くDebianの移植版です。しかも後継のs390xアーキテクチャがすでにありますので、実質の影響はほぼないでしょう。

また、これとは別にリリースに際して、各アーキテクチャごとに対応する人員の状況が整理されました（**表1**）。しかし、状況はあまり芳しくはないようです。通常、リリースの対象として確定するにはDDが5名必要ですので、ARMやx86アーキテクチャは問題ないのですが、その他のマイナーアーキテクチャは人員の不足が見て取れます。今後のリリースのためには、機材の充実もそうですが作業者の増加をどのようにして図るのか、ということがあらためて課題としてあぶり出されています。

Debianサーバ群のCDN対応?

「Possibly moving Debian services to a CDN」[注16]というメールで、Debianシステム管理チームからディスカッションが始まりました。Debianが提供しているいくつかのサービスでCDNを利用してはどうか、という内容です。技術的な問題はさほど大きくありませんが、「フリーではないインフラストラクチャーにDebianのサービスを委ねてしまうのか？」という倫理哲学的な問題は、なかなかうまい解決方法が見つからないようです。今後の議論に注目しましょう。

security.debian.orgミラーを日本に

現在、Debianではセキュリティアップデートを提供するのはsecurity.debian.orgとして応答するサーバのみですが、これをアジア地域向けとして日本にも置くのはどうか、という話を検討しています[注17]。

◆ ◆ ◆

来月は台湾で開催されるMiniDebConfレポートを中心にお送りする予定です。お楽しみに。SD

注16）URL http://lists.debian.org/debian-project/2013/10/msg00029.html
注17）URL http://www.debian.or.jp/project/donations.html

▼表1　アーキテクチャごとの対応人員数

アーキテクチャ名	DD	NM/DM	その他	合計
armel	5	0	2	7
armhf	6	1	2	9
hurd-i386	5	0	3	8
kfreebsd-amd64	5	0	2	6
kfreebsd-i386	5	0	2	6
ia64	0	0	3	3
mips	2	0	1	3
mipsel	2	0	1	3
powerpc[1]	0.5※	0	2	2.5※
s390x	1	0	1	2
sparc	1	0	0	1

DD：Debian Developer（公式開発者）、NM：New Maintainer（現在公式開発者に応募中）、DM：Debian Maintainer
※メイン担当ではない人については0.5換算とした

Red Hatの認定資格

田中 歩
Tanaka Ayumi
レッドハット㈱
教育サービス部
システムデザインエンジニア

システムデザインエンジニア……??

はじめまして。教育サービス部、システムデザインエンジニアの田中と申します。以前はインストラクターをしており、2年前に今の職種に異動しました。筆者の名刺を見せると、たいていの方が、「システムデザインエンジニア……??」という顔をなさるのですが、ここでいう「システム」は「教室内でトレーニング用に使用するシステム」と理解していただくとわかりやすいと思います。

レッドハット㈱のトレーニングは、コース・認定試験ともにすべて実技ベースで行います。筆者はおもに試験の際にインストラクターが使用するサーバや、お客様が使用するクライアント用のシステムの開発をしています。

Red Hatの認定資格たち

Red Hatではさまざまな認定試験を提供していますが、最も古くからあるのがRed Hat Certificed Engineer(RHCE)で、1999年にRed Hat Linux 5(Enterprise ではありません!)を対象として認定をスタートしました。その後、下位資格であるRHCT(現在のRHCSA)、上位資格のRHCA、仮想化資格のRHCVA、JBoss向けのRHCJAと、各製品に対応した認定資格が追加されていきました。現在日本国内で実施している認定試験は当社Webページ[注1]を確認ください。

実技試験のカラクリ

先ほども書きましたが、Red Hatの認定試験はすべて実技ベースで行われます。具体的にどのようなものかと言うと、与えられた試験問題の内容を満たすように、実機のセットアップ・修正を行うという、言葉で書くと非常にシンプルな内容です。

これは、実際の運用・構築環境を想定していて、たとえば新しくRed Hat Enterprise LinuxをOSとして使用したWebサービスを提供しようと思った場合、まずは提供するサービスの詳細を確認して仕様書を作成し、その仕様書に則ってシステムのインストール・構築作業をしていくことになります。試験問題はこの仕様書に相当し、仕様書に従って作業をするというわけです。そして、その採点も実際の環境に則した形で行われます。

たとえば、「Webサービスの設定は完璧!でも、ファイアウォールの設定忘れちゃった。」という場合は、結果的にクライアントはサービスを受けることができず、ブラウザから本来見られるであろうページを見ることはできません。クライアントにとって最も重要なのは、期待したサービスを受けられるかどうかです。認定試験においても「期待したサービスが提供されているかどうか」を基準に採点します。

試験環境を作る立場から申し上げますと、設定ファイル等を確認して採点する方がどれだけ楽なことか……(苦笑)。文字列処理はUNIX系OSの得意とするところですから、とてもシン

注1) http://jp.redhat.com/training/certifications/

プルに処理ができることでしょう。しかしあえてそれをせず、時にはとてつもなく長いコードを書きながら、粛々と作っています。

今後試験を受ける予定のある方は、構築などの作業だけではなく、どのようにそれを確認するか、採点はどのように行っているのかを、想像しながら作業するのもおもしろいかもしれません。あ、くれぐれも採点方法をインストラクターに聞いたりしないでくださいね。お答えすることはできませんので……。

認定番号のカラクリ

RHEL6から、認定番号のシステムが変更されました。以前は合格した試験ごとに、別々の認定番号が発行されていましたが、現在は1つの認定番号だけが発行され、そこに受験・合格した試験が紐付けされるようになっています。不合格だった試験も紐付けされるため、Certification Central[注2]にログインすると受験したすべての試験の記録を確認できます。

また、認定番号が正しいかどうかはVerify a Certificationのページ[注3]から確認できます。RHCEの認定番号がついた名刺を交換なさったときは、こっそりとここで確認してみてください。RHEL5以前の認定番号は15桁の数字で、下5桁がグローバルでの通し番号になっていました。

RHEL5以前で認定資格を取得した方は、ご自分の認定番号の下5桁を確認してみてください。全世界で何番目の合格者かがわかります。

Red Hatのオフィス物語

気がつけば社内でもかなり歴の長い方になってしまいました。ということで、Red Hatオフィスの今昔を少し紹介します。

入社したのは時をさかのぼること1X年前。オフィスは秋葉原でした。電気街口を出た先にはバスケットコートがあり、メイドカフェがちらほら増え始め、街にはパーツショップが溢れていたころです。トレーニング中にマウスが壊れても、お昼休みの間に調達できてたいへん助かりました。

その翌年にオフィスは神田に移転。神田明神のすぐそばのオフィスだったので、RHCE 試験の試験官をする日の朝は神田明神の境内を通りながら、「皆さん合格しますように」とお祈りしていました。

その2年後には六本木へ。日比谷線六本木駅からすぐの好立地。ほぼ傘いらずで助かりました。ビルに掲げてあるトレーニングルームという看板を見て、スポーツジムと勘違いされた方がいらしたのは土地柄？

さらにその2年後、現在の所在地である恵比寿に。千葉在住の私としては、移転する度に家から遠くなってしまってちょっと切ないのですが、個人的には今までの中で一番好きな場所です。東京タワーと富士山が見える、すてきなオフィス！　スカイツリーは見える、のかな？　今度調べておきます。トレーニングルームは現在品川にありますが、こちらも14階で東京タワーやレインボーブリッジが見え、とくに夜景が素晴らしいです。トレーニングを受講された暁にはぜひご覧になってください。

現在、11月末までのキャンペーンを実施しています。Red Hatのトレーニングの受講を検討されている方はお見逃しなく！　詳しくはこちらキャンペーンページ[注4]で確認ください。SD

注2）https://www.redhat.com/wapps/sso/login.html?redirect=%2Fwapps%2Fcertifications%2Ftx%2Fonboard
注3）https://www.redhat.com/wapps/training/certification/verify.html
注4）http://jp-redhat.com/training/campaign/

第44回

Ubuntu Monthly Report

Linuxのサウンドサブシステムと HDA

Ubuntu Japanese Team
坂本 貴史　SAKAMOTO Takashi
Mail o-takashi@sakamocchi.jp　Twitter @takaswie

今回は、Linux標準のサウンドサブシステムであるALSAについて、
身近なサウンドデバイスであるHDAとともに、そのしくみを解説します。

ALSA

そもそも「コンピュータを使う」ということは、ユーザが何らかの入力をコンピュータに与え、処理された結果を出力として得ることです。この入出力を行うためのデバイスは「入出力デバイス」と呼ばれます。入出力デバイスにはキーボードやモニタ、あるいはプリンタなどがありますが、今回はその中でも音声を扱うサウンドデバイスを取り上げます。

Linuxにおいてサウンドデバイスを使うためのサブシステムが、Advanced Linux Sound Architecture（ALSA）です[注1]。ALSAはUbuntuのようなLinuxディストリビューションやAndroidに使われています。ALSAの役割は、ソフトウェアとサウンドデバイスを仲介することです。音楽や動画の視聴、IP電話での通話だけでなく、スマートフォンにおける電話などでも利用されています。

HDA

ALSAが制御するサウンドデバイスのうち、最もよく見かけるものは「High Definition Audio（HDA）」と呼ばれるデバイスです。HDAとはIntel社が2004年に定めた、サウンドデバイスとその入出力インターフェースに関する仕様です[注2]。今日では、HDA準拠のサウンドデバイスはノートパソコンやデスクトップPCボードにほぼ標準で搭載されています。

HDAを説明するうえで重要となる要素を**図1**に示します。ソフトウェアはHDAコントローラを利用してHDAコーデックを操作し、音声のデジタル／アナログ変換や電気音響回路を制御します。制御のために「HDAコントローラがソフトウェアからコマンドを受け取って転送し、HDAコーデックからのレスポンスを受け取ってソフトウェアに返す」という処理を行い、実際の「音」のデータであるPCMデータは「ソフトウェアからDMA転送してHDAコントローラへ送り込み、HDAコーデックに届ける処理はHDAコントローラがやる」というのが基本的な処理の流れです。システムとHDAコントローラはPCI-Expressバスで、HDAコントローラとHDAコーデック間はHDA Linkと呼ばれる数本のシリアルバスで接続されます。

HDAはIntel社のICH6（I/O Controller Hub 6）で登場し、現在のPCH（Platform Controller Hub）でも変わらず使われています。また、HDAコントローラはNVIDIAのTegra3のような一部のSoC（System-on-a-Chip）にも集積されています。

注1）ただしBluetooth接続とFirewire接続のサウンドデバイスは、それぞれBluezとFFADOというプロジェクトがドライバやライブラリ、ユーティリティを開発しています。

注2）HDA以前にもIntel社はAC'97という仕様を定めています。しかしHDAのコンセプトにはAC'97との互換性は含まれていません。

規格上はHDAコーデックにベンダ独自の制御コマンド拡張を付加して、HDAコーデックに特殊な機能を持たせることもできます。また、同じベンダのHDAコーデックでも、それを実装するハードウェアベンダーが機能の取捨選択を行うことがあります。加えて、ハードウェアベンダによって電気音響回路との接続が異なることがあります。HDAコーデックベンダ独自コマンドとハードウェアベンダの実装の違いが、ドライバ開発者の悩みの種です。

ALSAのカーネルランド

ALSAの実装はユーザランドとカーネルランドの両方にあります。まずはカーネルランドを見ていきます。ALSAのカーネルランドはCoreと各デバイスドライバで構成されます。Coreはさまざまな役割を持つので、ここではドライバを説明するという観点から2つだけ述べます。1つはデバイスへの共通したインターフェースをユーザに提供すること、もう1つは各ドライバが実装するべき手順を規定することです。アプリケーションプログラマはALSAのインターフェースを実装することだけ、ドライバ開発者は手順に合わせてデバイス固有処理を実装することだけを気にすれば良いようにしています。

ALSAのHDAドライバ

それではALSA Coreが規定する手順に従い、HDAドライバがどのような処理をしているのかを簡単に見ていきましょう。ただし、ALSAはいくつかのキャラクタデバイスを設けますので、ここではPCMデバイスの処理に限定します。

❶ open
ハードウェアのケーパビリティを登録します。

❷ hw_params
ケーパビリティとユーザランドからの要求を調整し、ハードウェアのパラメータを決定します。最後に、DMA用メモリページを確定します。

❸ prepare
メモリページとI/Oポートの間のDMA転送設定、コーデックの準備、コマンド・レスポンス用バッファの準備を行います。

❹ trigger
データ転送の開始や終了、サスペンドやレジュームに伴う処理を行います。

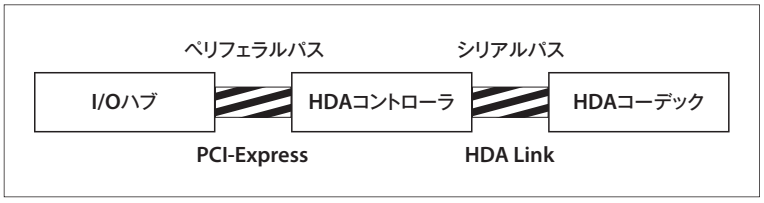
❺ pointer
buffer上のポインタ位置を返します。

❻ hw_free
DMA転送設定の解除、コーデックのリセット、メッセージバッファのクリアを行います。

❼ close
PCMデバイスを終了します。

メモリページを確保し、I/OポートへDMAを用いてデータ転送するというのが処理の要です。このとき、ALSAはメモリページをbufferとperiodという単位で管理します。通常、メモリページはプラットフォームのMMUを利用して4,096バイト単位で管理されますが、確保したページのうちALSAが実際に使う領域がbufferです。periodはbufferを分割したもので、サウンドデバイスとのDMA転送の単位として用いられます。DMAコントローラはperiod分を転送し終えるとハードウェア割り込みを発生し、それをハンドルしたドライバは次のperiodを転送するよう制御します。buffer内の最後のperiodを転送し終えると、最初のperiodに戻って転送を続けます。

図1　HDAの要素



ALSAのユーザランド

それでは次に、ユーザランドを見ていきましょう。ひとくちにユーザランドと言っても、ライブラリや設定ファイルパーサ、ユーティリティまでを含む大きなソフトウェアとなっています。ALSAは1998年に始まったプロジェクトで、今日に至るまで大量のコード変更が行われています。その中にはALSAのキャラクタデバイスに対するファイル操作の仕様変更も含みます。ALSAはライブラリを提供することで、ALSAのバージョンの違いを吸収し、そしてさまざまな機能も提供します[注3]。

PCMノード

ライブラリの役割の1つは、ランタイムへPCMノードを設けることです。PCMノードはPCMのキャラクタデバイスを隠蔽し特定の機能を付与するもので、aplay -L（またはarecord -L）で一覧を取得できます（**図2**）。

plughwはドライバが対応できるPCMデータの諸要素とアプリケーションが要求するそれの食い違いを自動で調整します。frontやsurroundといったPCMノードは、plughwをスレーブとして使ってチャンネルの違いに対応します。hwはこれらのスレーブとなるPCMノードで、直接キャラクタデバイスを使います。これらはALSAのPCMプラグインと、ランタイム設定ファイル[注4]によって実現されています。

PCMインターフェース

2001年あたりのALSAはPCMキャラクタデバイスへのread/write、つまりcatなどのツールを使ってPCMデータの読み書きができました。しかし現在は、ライブラリが提供するPCMインターフェースを利用します。ALSAのライブラリ（asound）はC言語のインターフェースを提供します。ALSAのPCMインターフェースを使うには、**リスト1**のように手続きを記述します。なお、ここでは処理の流れがわかりやすいよう、mmapを使わず、また重要なAPIだけを抜粋しています。

❶ハンドルの取得

snd_pcm_open()の引数にPCMノード名とフラグを与え、ハンドルを取得します。フラグではデータの入力か出力か、などを指定します。このときカーネルランドのドライバはopenを処理します。キャラクタデバイスは使用中となります。

注3） なお、Androidのユーザランドの場合、ALSAの代わりとなるソフトウェアスタックを開発しています。

注4） Ubuntuであれば/usr/share/alsa以下のファイルになります。実はALSAがリリースしている設定ファイルでは、aplay/arecordがdmix/dsnoop/hw/plughwの各PCMノードを表示することはありません。Ubuntuではlp#652035のバグに対応するパッチが適用されており、それらを見ることができます。

図2　PCMノードの一例

```
$ aplay -L
default
    Playback/recording through the PulseAudio sound server
front:CARD=PCH,DEV=0
    HDA Intel PCH, CONEXANT Analog
    Front speakers
surround40:CARD=PCH,DEV=0
    HDA Intel PCH, CONEXANT Analog
    4.0 Surround output to Front and Rear speakers
dmix:CARD=PCH,DEV=0
    HDA Intel PCH, CONEXANT Analog
    Direct sample mixing device
dsnoop:CARD=PCH,DEV=0
    HDA Intel PCH, CONEXANT Analog
    Direct sample snooping device
hw:CARD=PCH,DEV=0
    HDA Intel PCH, CONEXANT Analog
    Direct hardware device without any conversions
plughw:CARD=PCH,DEV=0
    HDA Intel PCH, CONEXANT Analog
    Hardware device with all software conversions
```

❷ハードウェアパラメータの設定

ハンドルに対しsnd_pcm_hw_params()を実行し、ハードウェアパラメータを設定します。パラメータはあらかじめsnd_pcm_hw_params_alloca()などでアロケートし、snd_pcm_hw_params_set_xxx()という一連の補助関数を使いセットします。カーネルランドのドライバではhw_paramsを処理し、PCMデータやバッファに関する諸要素を決定します。続いてドライバのprepareがコールされます。

❸PCMデータの書き込みあるいは読み出し

ハンドルに対しPCMデータの書き込みあるいは読み出しを行うAPIを実行します。詳しい処理内容は後述します。

❹bufferの書き出し

たとえbufferにPCMデータが残っていたとしても、アプリケーションを終了すると破棄されてしまいます。snd_pcm_drain()はbufferすべてを転送し終えるまでアプリケーションの実行をブロックします。なお、この処理は必須ではありません。

❺後片付け

ハンドルに対しsnd_pcm_close()を行います。ドライバ側ではfreeとcloseが順次処理されます。キャラクタデバイスの使用が解除されます。

PCMデータの書き込みや読み出しの処理をもう少し見ていきます。書き込みや読み出すデータはPCMのサンプルです。ALSAでは同じ再生タイミングを持つサンプルの集合をframeと呼びます。ステレオ2chであれば、1frameは2サンプルになります。このとき、サンプルが16ビットで量子化されていれば、1frameは4バイトになります。ALSAのAPIはこのframeを単位としてサンプルを扱います。

APIを使って初めて書き込みあるいは読み出しを行うと、ランタイムのステータスが変わり、ドライバのtriggerがコールされデータ転送が開始されます。ユーザランドからは順次書き込み、あるいは読み出しを行ってください。このときバッファへの書き込み位置はライブラリが調整するため、意識する必要はありません。

書き込みあるいは読み出し中、ALSAはユーザランドとカーネルランドのポインタを監視し、アンマッチを検出するとXRUNというエラーを発生して転送を中止します。たとえば、ユーザランドからの書き込みが遅過ぎて、まだ書き込まれていないperiodをカーネルランドでDMA転送してしまう、あるいはユーザランドの読み出しが遅過ぎて、まだ読み出していないperiodをカーネルランドのDMA転送が上書きしてしまう、などです。原因はアプリケーション実行速度と比較すると、DMA転送のほうが圧倒的に早いことにあります。この場合はsnd_pcm_recover()を実行します。内部ではsnd_pcm_prepare()をコールし、ドライバのprepareが実行されて転送が再開されます。

snd_pcm_close()を実行するとランタイムのステータスが変わり、ドライバのtriggerがコールされてデータ転送が終了します。その後hw_free、closeと順次処理されます。

まとめ

ALSAは大きなソフトウェアスタックです。今回扱ったPCM機能のほかにも、サウンドデバイスのミュートなどを行うcontrol機能や、MIDIメッセージ送受信を行うMIDI機能などがあります。この記事がALSAの理解のきっかけになればうれしいです。SD

リスト1　ALSAライブラリAPIの使用例（抜粋）

```
#include <sound/asound.h>
snd_pcm_open(&handle, "default", SND_PCM_STREAM_PLAYBACK, 0); ……❶
snd_pcm_hw_params(&handle, &params); ……❷
while (1) {
    frames = fill_buffer_with_samples(buffer);
    err = snd_pcm_writei(handle, buffer, frames); ……❸
    if (err < 0)
        err = snd_pcm_recover(handle, frames, 0);
    if (err < 0)
        printf("%s", snd_strerror(err));
    else if (frames != err)
        printf("short write¥n")
}
snd_pcm_drain(handle); ……❹
snd_pcm_close(handle); ……❺
```

第21回

Linux 3.12の新機能
～btrfsのdedupとdm-statistics～

Text：青田 直大　AOTA Naohiro

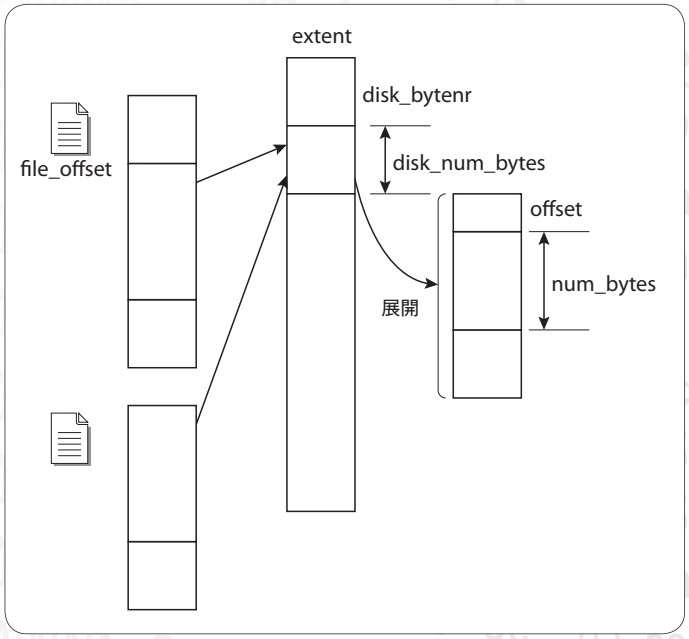
LinuxCon Europeが行われていることもあってか、Linux 3.12のリリースはもう少し遅くなりそうです。今回もLinux 3.12に入る予定の「btrfsのdedup」と「dm-statistics」の紹介を行います。

btrfsのdedup

豊富な機能を備えるファイルシステムbtrfsに、3.12ではまた新しく機能が1つ追加されます。3.12にはdedupといって、同じデータを持つファイルを「まとめる」機能が実装されました。

btrfsではファイルのデータは、「あるファイルの何バイト目(file_offset)から何バイト(num_bytes)」が「extentアドレスのどこ(disk_bytenr)から何バイト(disk_num_bytes)を(データがbtrfsによって圧縮されている場合は)展開したデータの先頭何バイト(offset)から何バイト(num_bytes)」というように管理されています(**図1**)。このようにデータが管理されているので複数のファイルが同じデータを持つ場合にはデータを共有させることで、ディスク上に保管しなければならないデータサイズを削減できます。

▼図1　btrfsのファイルデータ構造



duperemoveの使い方

では、実際にdedup機能を使用してみましょう。新しくbtrfsのファイルシステムを作り、"btrfs fi df"コマンドでディスクの使用状況を見てみます。問題となるのは"Data:"の部分で初期状態で256KBが使用されています(**図2**)。

サイズ128KB×10のファイルfooを作り、さらにfooの内容を2つ分持つファイルbarを作ります(**図3**)。Dataの使

用量(used)はfooを作ったあとは“1.50MB”で、barを作ったあとは“4.00MB”と順当に増加しています。しかし、barのデータはfooと完全にかぶっているので、このように無駄に容量をとってほしくはないですね。

では、dedupしてみましょう。dedupにはduperemove[注1]を使います。duperemove自体はbtrfs以外のファイルシステムでも動作し、データに重複があるかどうかを見つけることができます。使い方は単純にdedupしたい対象のファイルを指定するだけです。ディレクトリを指定した場合には、その直下のファイルがチェックされます[注2]。デフォルトでは重複チェックを行うだけでdedupは行われません。実際にdedupをさせるには、“-D”を指定してください。fooとbarを指定して(verboseの“-v”と“-D”も指定して)duperemoveを実行します(**図4**)。

❶重複のチェックは128KB単位で行われています。これは“-b”を使って変更できます。
❷重複したデータ領域が1種類見つかっています。fooの0ブロック目から(＝0B目)と、barの0ブロック目からとbarの10ブロック目(＝128×10KB)からとが重複部分で、合計2621440(＝128×10×2KB)のデータが重複と出ています。
❸実際にdedupをbtrfsに指示している部分です。“foo”の0バイト目から1310720バイトを指すように、“bar”の0バイト目からと、1310720バイト目からがdedupされます。

注1) https://github.com/markfasheh/duperemove
注2) “-r”で再帰的処理になります。

▼図2　ファイルシステムを作成し容量を確認

```
# mkfs.btrfs /dev/vg2/test
# mount /dev/vg2/test /mnt/btrfs
# cd /mnt/btrfs
# btrfs fi df .
Data: total=8.00MB, used=256.00KB
System, DUP: total=8.00MB, used=4.00KB
System: total=4.00MB, used=0.00
Metadata, DUP: total=1.00GB, used=28.00KB
Metadata: total=8.00MB, used=0.00
```

dedupが終わって“btrfs fi df”を使い、再び使用量を見てみると、fooを使った直後と同じく“1.50MB”となっています。確かに重複部分がdedupされて、ディスク容量を節約できたこと

▼図3　無駄のある状態

```
# dd if=/dev/urandom of=foo bs=128K count=10
10+0 レコード入力
10+0 レコード出力
1310720 バイト (1.3 MB) コピーされました、
0.1006 秒、13.0 MB/秒
# sync
# btrfs fi df .
Data: total=8.00MB, used=1.50MB
System, DUP: total=8.00MB, used=4.00KB
System: total=4.00MB, used=0.00
Metadata, DUP: total=1.00GB, used=28.00KB
# cat foo foo > bar
# sync
# btrfs fi df .
Data: total=8.00MB, used=4.00MB
System, DUP: total=8.00MB, used=4.00KB
System: total=4.00MB, used=0.00
Metadata, DUP: total=1.00GB, used=28.00KB
Metadata: total=8.00MB, used=0.00
# ls -lh
合計 3.8M
-rw-r--r-- 1 root root 2.5M 10月 25 05:48 bar
-rw-r--r-- 1 root root 1.3M 10月 25 05:46 foo
```

▼図4　dedup後の使用量を確認する

```
# duperemove -v -D foo bar
Using 128K blocks                                    ……❶
csum: foo
csum: bar
Found 1 instances of duplicated extents              ……❷
(dext: 0x0x12a7f50) 3 extents had length 10
(1310720) for a score of 2621440.
foo     start block: 0 (0)
bar     start block: 0 (0)
bar     start block: 10 (1310720)
Calculated 2621440 bytes of duplicated data.
Deduping data...                                     ……❸
3 extents had length 10 (1310720) for a score of 2621440.
foo     start block: 0 (0)
bar     start block: 0 (0)
bar     start block: 10 (1310720)
Running dedupe.
Dedupe from: "foo"      offset: 0       len: 1310720
"bar":  offset: 0       deduped bytes: 1310720  status: 0
"bar":  offset: 1310720 deduped bytes: 1310720  status: 0
Deduped 2621440 bytes of data
# sync
# btrfs fi df .
Data: total=8.00MB, used=1.50MB
System, DUP: total=8.00MB, used=4.00KB
System: total=4.00MB, used=0.00
Metadata, DUP: total=1.00GB, used=28.00KB
Metadata: total=8.00MB, used=0.00
```

がわかりますね。

dedupのscore

次にduperemoveが表示している“score”について見てみましょう。**図5**のようなデータを持ったファイルfoo、bar、bazを作ります。このときdedupする領域の候補はいくつかありますが、どの部分をdedupするのかを決定するのがscoreになります。デバッグ出力をする“-d”を付けてduperemoveを実行してみましょう(**図6**)。

❶まず各ファイルの各ブロックのハッシュが計算されます。ここでは、それぞれのハッシュを持ったブロックが一覧表示されています。**図5**の中の番号でいえば、“2”、“3”、“1”、“4”の順番に出力されています。

❷このハッシュを使ってdedupの候補が出力されます。**図5**に示してある3ヵ所になっています。これら3つの領域はどれも互いに重複した領域を含んでいます。この3つからどれをdedupするのかを選択するのが次のステップです。

❸重なったdedup候補領域から1つを選択するために使われるのがscoreの値です。これはdedupによって削減できるバイト数そのものになっています。結局、なるべく多くのバイト数を削減できるような部分を選び出している、ということです。

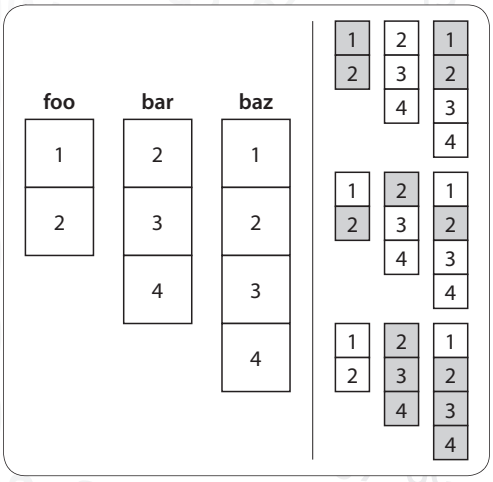
dedupのioctl

btrfsへのdedupの指示は、dedupしたいすべてのファイルを開いた状態で、新しく追加されたioctlであるBTRFS_IOC_FILE_EXTENT_SAMEを呼び出して行います。ioctlの引数は**図7**のようになっていて、ioctlの対象の<logical_offset>バイト目から<length>バイトで、<info>以降に指定されているファイルのそれぞれの<logical_offset>バイトからをdedupするようになっています。カーネル側では渡されたファイルとoffset情報から実際にデータが等しくなっているかを検証し、ファイルのextent参照を書き換えます。<info>内の<bytes_deduped>に何バイトが削減されたかと、<status>にdedupに成功すれば0、それ以外ならなんらかのエラーコードが返ってきます。

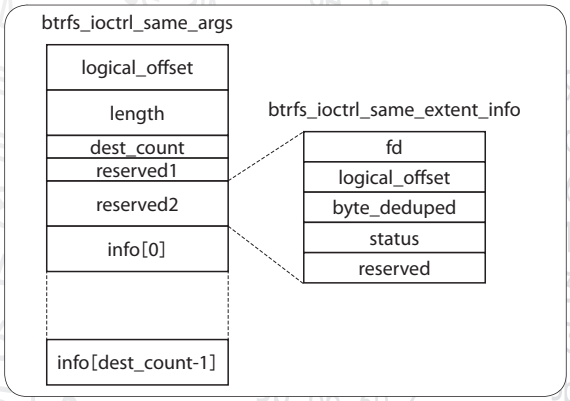
online dedup

ここまで見てきたように今回のリリースでbtrfsに入るdedupはユーザランド側で、どのファイルのどの部分と、どのファイルのどの部分と

▼図5　実行前のファイル構造



▼図7　ioctlの引数



▼図6　実行結果

```
# duperemove -d -v -D foo bar baz
added file: foo
added file: bar
added file: baz
Using 128K blocks
csum: foo
csum: bar
csum: baz
Block hash tree has 4 hash nodes and 9 block items            …… ❶
All blocks with hash: 041b197045cb128dfa51d62039e8e734fb18dc682a3fadb54ce019dbcf3d2b2f
foo     loff: 131072 lblock: 1
bar     loff: 0 lblock: 0
baz     loff: 131072 lblock: 1
All blocks with hash: 06271c1a4a3b3f8bb6bf39c4979b940f28afcbd2f60f025a14da5cb6acb9c8df
bar     loff: 131072 lblock: 1
baz     loff: 262144 lblock: 2
All blocks with hash: ab1821187a893dd8a83ee005e5c1d001217cd9690033a73e7dbc0cecec6fd19a
foo     loff: 0 lblock: 0
baz     loff: 0 lblock: 0
All blocks with hash: e85697c2f8e814e9c7689a4ca0bd0e245b2d5744de78c4946e0854795d23021b
bar     loff: 262144 lblock: 2
baz     loff: 393216 lblock: 3
Duplicated extent of 1 blocks in files:
foo             bar
1-2             0-1
Duplicated extent of 2 blocks in files:
foo             baz
0-2             0-2
Duplicated extent of 3 blocks in files:
bar             baz
0-3             1-4
Found 3 instances of duplicated extents                       …… ❷
(dext: 0x0x10a06f0) 2 extents had length 1 (131072) for a score of 131072.
Hash is: 9a62716de2e3babdd33a8d6217f02702988d84116426f98d0f50cb637940a1bc
foo     start block: 1 (131072)
bar     start block: 0 (0)
(dext: 0x0x10a0810) 2 extents had length 2 (262144) for a score of 262144.
Hash is: 8b829c30c92c9a7c052b4bdbaf2bfb985a4ac820afe88ea42810bdb6578337c7
foo     start block: 0 (0)
baz     start block: 0 (0)
(dext: 0x0x10a0930) 2 extents had length 3 (393216) for a score of 393216.
Hash is: 3a8e427c614ceaa633ffef01e602785210135010eaaa6edf5f8cfcd64b9b548e
bar     start block: 0 (0)
baz     start block: 1 (131072)
Calculated 786432 bytes of duplicated data.


Removing overlapping extents                                  …… ❸

Found 1 instances of duplicated extents
(dext: 0x0x10a0930) 2 extents had length 3 (393216) for a score of 393216.
Hash is: 3a8e427c614ceaa633ffef01e602785210135010eaaa6edf5f8cfcd64b9b548e
bar     start block: 0 (0)
baz     start block: 1 (131072)
Calculated 393216 bytes of duplicated data.
Deduping data...
2 extents had length 3 (393216) for a score of 393216.
Hash is: 3a8e427c614ceaa633ffef01e602785210135010eaaa6edf5f8cfcd64b9b548e
bar     start block: 0 (0)
baz     start block: 1 (131072)
Running dedupe.
Dedupe from: "bar"      offset: 0       len: 393216
"baz":  offset: 131072  deduped bytes: 393216   status: 0
Deduped 393216 bytes of data
```

が重複しているのかを調べ、見つけた重複部分をカーネル側に通知しdedupしてもらうものです。これをファイル書き込み時に自動的にdedupが行われるのではない、という意味で“offline dedup”と呼んでいます。

これに対してファイルへの書き込み時に自動的に書き込み内容のハッシュ値をとり、同じハッシュ値を持つextentがすでにあれば新しいextentを作成する代わりに見つけたextentを参照するようにしたものです[注3]。こちらは対比して“online dedup”と呼ばれます。

online dedupにはoffline dedupと比較して自動で行われるため、ユーザランドでコマンドを起動する手間がなく、どのあたりのデータが重複しているのかをユーザが考える必要がないメリットがあります。一方で、書き込み時にハッシュ値計算を行っているので新しいデータに対してはパフォーマンスが悪化するデメリットがあります。もちろん同じハッシュを持つデータばかりであれば、ディスクへのI/Oが削減され劇的に速くなることもありえます。

もう1つのデメリットとして、dedupする単位がofflineと比較して小さいことも挙げられます。online dedupの現在の実装ではチェックできるサイズの範囲が4KBから128KBに限られています。offline dedupであればより広い範囲を1つのextentとして共有できます。

dm-statistics

次はdm-statisticsです。これは名前に反して新しいDevice Mapperターゲットではなく、Device Mapperによるデバイスの統計情報をとるためのしくみです。これによってデバイスのどの部分にどれだけの読み書きがあるのかなどを知ることができます。

では、実際に統計情報をとってみましょう。dm-statisticsの操作には“dmsetup message <device> <sector>”を使います。ここではsectorは“0”になります(**図8**)。

❶まずは“@stats_crate <range> <step>”を使って統計をとるための領域を設定します。1つめの例では“-”を使ってデバイス全体を、“/100”によって100の部分に分割する領域を設定しています。<range>には“<start_sector>+<length>”の形で、領域全体の始まりのセクタと長さを、<step>には数値を指定することで個々の領域の長さを設定できます。この2つのパラメータ以外に<program_id>と<aux_data>という2つの引数を領域の識別用に付け加えることもできます。dmsetupが“0”と出力していますが、これが領域のIDとなります。このIDを使って統計情報を取得したい領域を指定します。

❷“@stats_list”を使うことで作成されている領域の一覧を見ることができます。ここに<program_id>

注3) http://thread.gmane.org/gmane.comp.file-systems.btrfs/29125

▼図8　dm-statisticsの実行

```
# dmsetup message vg2-test 0 @stats_create - /100  …… ❶
0
# dmsetup message vg2-test 0 @stats_create 0+209716 ➡
100000 344 foobar
1
# dmsetup message vg2-test 0 @stats_list           …… ❷
0: 0+20971520 209716 - -
1: 0+209716 100000 344 foobar
# dmsetup message vg2-test 0 @stats_print 0        …… ❸
0+209716 0 0 0 0 0 0 0 0 0 0 0 0 0
209716+209716 0 0 0 0 0 0 0 0 0 0 0 0 0
419432+209716 0 0 0 0 0 0 0 0 0 0 0 0 0
629148+209716 0 0 0 0 0 0 0 0 0 0 0 0 0
838864+209716 0 0 0 0 0 0 0 0 0 0 0 0 0
...
19923020+209716 0 0 0 0 0 0 0 0 0 0 0 0 0
20132736+209716 0 0 0 0 0 0 0 0 0 0 0 0 0
20342452+209716 0 0 0 0 0 0 0 0 0 0 0 0 0
20552168+209716 0 0 0 0 0 0 0 0 0 0 0 0 0
20761884+209716 0 0 0 0 0 0 0 0 0 0 0 0 0
# dmsetup message vg2-test 0 @stats_print 1
0+100000 0 0 0 0 0 0 0 0 0 0 0 0 0
100000+100000 0 0 0 0 0 0 0 0 0 0 0 0 0
200000+100000 0 0 0 0 0 0 0 0 0 0 0 0 0
```

と<aux_data>が表示されます。

❸“@stats_print <ID>”を使って統計情報を見ることができます。とくに何もしていないのでまだ“0”が並んでいるだけです。

このデバイスのファイルシステム上でカーネルソースを展開してから、再度統計を見てみましょう(**図9**)。1行目を例にとるとセクタ0から209715について**表1**のものを計測しています。

この中で4、8、11番めと10、12、13番めの区別がわかりにくいですね。**図10**のようにA、B、Cと3つのI/Oがあったときに4、8、11番めはAmsec + Bmsec + Cmsecの合計であり、10、12、13番めはTmsecの部分になります。

dm-statisticsにはほかにも統計のクリアなどの操作もあります。詳しくはDocumentation/device-mapper/statistics.txt[注4]を参照してみてください。

まとめ

今回はbtrfsの新機能であるファイルの重複データをまとめてデータ容量を削減するdedup、そしてDevice Mapper上のデバイスの統計情報をとれるdm-statisticsというLinux 3.12の2つの新機能を紹介しました。SD

注4）http://lwn.net/Articles/566273/

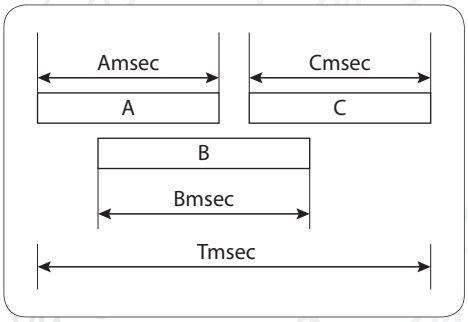
▼図9　カーネルソースを展開し統計を確認

```
# tar xf /usr/portage/distfiles/linux-3.11.tar.xz
# dmsetup message vg2-test 0 @stats_print 0
0+209716 88 0 1208 72 1796 0 161996 728812 0 2217 728884 72 2145
209716+209716 0 0 0 0 3139 0 125316 584215 0 6158 584215 0 6158
419432+209716 0 0 0 0 2441 0 85192 314876 0 4114 314877 0 4114
629148+209716 0 0 0 0 0 0 0 0 0 0 0 0 0
838864+209716 0 0 0 0 0 0 0 0 0 0 0 0 0
...
20342452+209716 0 0 0 0 0 0 0 0 0 0 0 0 0
20552168+209716 0 0 0 0 0 0 0 0 0 0 0 0 0
20761884+209716 15 0 120 43 0 0 0 0 43 43 43 0
# dmsetup message vg2-test 0 @stats_print 1
0+100000 0 0 0 0 317 0 13072 2111 0 72 2111 0 72
100000+100000 0 0 0 0 2 0 16 39 0 39 39 0 39
200000+100000 0 0 0 0 0 0 0 0 0 0 0 0 0
```

▼表1　計測する項目

1	88	読み込み完了数
2	0	読み込みがマージされた数
3	1208	読み込まれたセクタ数
4	72	各I/Oが読み込みに使ったミリ秒数の合計
5	1796	書き込み完了数
6	0	書き込みがマージされた数
7	161996	書き込まれたセクタ数
8	728812	各I/Oが書き込みに使ったミリ秒数の合計
9	0	現在実行中のI/O
10	2217	I/O実行中全体のミリ秒数
11	728884	各I/Oが実行中であったミリ秒数の合計
12	72	読み込みに使われたミリ秒数
13	2145	書き込みに使われたミリ秒数

▼図10　I/Oの実行時間



December 2013

NO.26

Monthly News from jus

Japan UNIX Society

日本UNIXユーザ会　http://www.jus.or.jp/
高橋 昭子　TAKAHASHI Shoko　shoko@jus.or.jp
法林 浩之　HOURIN Hiroyuki　hourin@suplex.gr.jp

暑い夏の熱い講演２連戦

今回は8月に島根、9月に札幌で行った研究会の模様を報告します。

島根大会

■Thinking RealSpace：
みんなの情報を正しく集め活用しよう!!!
【日時】2013年8月24日（土）10:40～10:55
【会場】松江テルサ 4F 大会議室
【講師】砂原 秀樹（日本UNIXユーザ会／慶應義塾大学）

砂原先生を講師にお迎えし、個人情報を正しく集め活用することを目指すプロジェクト「情報銀行コンソーシアム（仮称）」について紹介いただきました。OSC 2013 Shimaneのセッションとして行ったこともあり、76人もの方に参加いただきました。

まず、あちこちで個人のプライバシーが盗まれている、という実例の紹介がありました。各自のカーナビゲーションからの走行情報によって渋滞情報を作っているものの、果たして個人情報がどのくらいまで使われているか、実ユーザとしてはわかっていないこと。逆に震災でどこが通れるかがわかるといった役に立つ例もあったこと。また、もっと如実にわかる例として震災の日の人間の動きを携帯電話のデータをもとに表している動画の紹介がありました。

講師からは、「最近、勝手に情報を売って問題になった会社のように、新しいことやおもしろいこと、商売になりそうなことができるとはいえ、自分が出したデータをちゃんと管理しながら使ってくれるのか、ユーザ自身にはそもそも不明で気持ち悪いことが問題である」という意見が出されました。

個人情報はきちんと集め、いったん集めたものをできるだけ壊さずに取っておくことが個人情報保護法上も大切です。しかし、誰もがその条件を満たすのは難しい。そこで、きちんとした組織が情報を集めて管理するしくみをオープンに作るため、東京大学の柴崎先生を助けてコンソーシアムを作ることにしたそうです。

- 情報を預けて（取られるのではなく積極的に出す）
- 運用し（アイデアを結集して活用する）
- 利益を還元する（提供者に還元）

という運用のしくみを2、3年で作り、その後は誰かが会社を起業してくれたらそちらに移管していきたいとのことです。

今後、コンソーシアムでは、2つのワーキンググループを中心に技術仕様書やソースコードなどさまざまなアウトプットを提供していきたいとの報告がありました。技術WGでは基礎プロトコルの開発や、基礎技術の標準化、実証実験（会員のみ）を実施し、社会受容性WGでは、情報を出してもらうための法的問題の検討や、認証監査のメカニズム、資産管理などの検討を予定しているそうです。9月30日に慶応義塾大学三田キャンパスで行われるオープンニングシンポジウムへのお誘いもありました。

会場からは、2つの質問や指摘がありました。「オープンデータ／ビッグデータはどこが一番お金になるのか？」という質問に対しては、「運用する（＝どうやって使うかを考える）ところが一番お金になる。た

だし、儲かったら総取りではなく、提供者にも少し還元してほしい。気持ちよく出してもらうためには、どこに情報を取られているかがわかって理解してもらえるしくみが大切だが、ビジネスとしてはどう利用するかが本質」との回答がなされました。

「情報を取られることに対して心のバリアがある」という指摘に対しては、「何のために情報が取られているか、情報がどこで利用されたのか、情報を消したときに本当に消えたのか、といったことを説明できるように監査のしくみなども整備して、個人が積極的に情報を出せるようにしたい」とのこと。そのためのDBの作り方や暗号化の方法も検討している最中なので、興味がある人はぜひ参加してほしいと呼びかけられました。

講演後の懇親会でもさまざまな質問が出ていたことからも、やはり個人情報を適切に扱うためのしくみ作りは、個人としてもビジネスとしてもたいへん興味を持たれている分野であると感じました。本件に興味がある方は、info-bank-inq@kmd.keio.ac.jp まで連絡してみてはいかがでしょう。

札幌大会

■キミも実機をさわって
インフラエンジニアにならなイカ？
【日時】2013年9月14日（土）13:00～13:45
【会場】札幌コンベンションセンター 104会議室
【講師】深町 賢一（千歳科学技術大学）

講師にお迎えした深町さんは、メーリングリスト管理ツールfmlの作者としても知られますが、今回は大学で行っている実習の紹介と、それを通して学生に何を教えたいのかについての考えをうかがいました。参加者は18人でした。

実習の背景として、深町さんと同世代の人々はインターネットのインフラ作りをおもしろいものととらえていますが、若い世代はなかなか興味を持ってくれないこと、それから仮想化環境の普及により実際のハードウェアを触る機会が少ないため、学校にいる間に実機を触ることを体験させたいと考えていることを掲げました。そこから生まれた実習内容は次のとおりです。

- システム開発の授業のうちの４週間で実習を実施
- 作業内容はOSのインストール、ネットワークの設定、SSHによるリモートログイン、Apacheの設定、PostgreSQLの設定とデータベース作成、CGIの作成とデータベース検索
- ２人１組で作業
- 深町さんが公開しているNetBSD構築ガイド（http://www.nsrg.fml.org/）をマニュアルとして使用
- トラブル対応も体験させるため、学生に提供するPCにはわざと不良品を混ぜている

実習結果としては、マニュアルどおりに作業を進めれば良いので全員期間内に一通りの作業はできたとのことです。ただし作業の意味を理解できたかどうかは別で、積極的に取り組んだと思われる学生は全体の5%ぐらいだそうです。

NetBSDを選択したのは、深町さんが詳しく知っているという事情もありますが、新製品や試験製品の評価においてはメッセージが英語しかなく、GUIも用意されていないことは珍しくないので、そうした状況でも作業できるように訓練しておくべき、という理由によるものです。また、実機を使うことにこだわる理由は、それが本来どういった役割の機材や構成なのか、ハードウェアのトラブルにはどんな種類があるのかを体験させたいというものです。

今後の構想として、「インフラエンジニア虎の穴」の紹介がありました。いわばインフラエンジニアの養成機関です。いろいろなベンダーの機器を用意し、それらを学生や若手エンジニア向けの研修ができる施設として提供したいと考えているそうです。

深町さんは現職に就く前はISPで仕事をされていたので、その経験に基づいてインフラエンジニアに求められる能力の養成を考えていることがうかがえる、たいへん興味深い講演でした。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スタッフ
鎌田 篤慎　KAMATA Shigenori
Twitter @4niruddha
佐伯 幸治　SAEKI koji
Twitter @widesilverz
高橋 憲一　TAKAHASHI Kenichi
Twitter @ken1_taka

第24回 今年の夏もアツかった! ～石巻ハッカソン開催～

社会的課題をテクノロジで解決するためのコミュニティHack For Japanの活動をレポートする本連載。今回・次回は復興支援イベントとして宮城県石巻市で開催された石巻ハッカソンについて紹介します。

「震災10年後の2021年までに1,000人の開発者を石巻から生み出す」ことを目的に開催

7月26～28日にわたって「第2回 石巻ハッカソン」が開催されました。このハッカソンは昨年に開催された第1回に続くイベントで、地元の若者たちにアプリ開発を体験してもらいプログラムの楽しさを伝え、ゆくゆくはITによる地域活性化を担う人材を生み出すという大きな目標があります。このイベントを主催しているのはイトナブ[注1]という石巻のコミュニティです。イトナブとは「IT」×「営む」×「学ぶ」の造語で、「石巻の次世代を担う若者を対象にWebデザインやソフトウェア開発を学ぶ拠点と機会を提供し、地域産業×ITという観点から雇用促進、職業訓練ができる環境づくりを目指している」というのが活動の趣旨です。

今回はイトナブ主催者である古山隆幸氏を中心に石巻の若者たちが運営に携わり、古山氏がめざす「震災10年後の2021年までに1,000人の開発者を石巻から生み出す」という理念のもと、Hack For Japanのスタッフがプログラムを教える講師役としてこのイベントをサポートしました。今回の石巻ハッカソンでテーマとして掲げられていたのは「石巻の若者を触発させる」でした。“開発の経験を通して、あらゆる若者たちが彼ら自身の行動や考えに触発される何かを手に入れてほしい”というイトナブ古山氏の熱い想いが込められていたように感じました(**写真1**)。

◆写真1 「石巻をHackしてやる」気合い十分の参加者

石巻ハッカソンスタート!

7月26日の午後からハッカソンがスタート。古山氏からは「石巻には若者たちが活躍できるフィールドが少ないが、ITには無限の可能性がある。エンジニアのスキルを学べば石巻から世界に挑戦できる人材が生まれると信じて邁進している。この3日間で何かを得てほしい。全国からプロのエンジニアが参加しているので、彼らと接してぜひ自分の学びにしてください」といった開催宣言が行われました。

続いてHack For Japanスタッフ及川からは、ハッカソンそのものについて、Hack For Japan活動の説明がありました。さらに各自の自己紹介を経て、それぞれの部門に分かれて石巻ハッカソンが開始となりました。ハッカソンは石巻工業高校を使って行われていたため、部門ごとに各教室に分かれての作業となります。教室内の黒板を使ってのアイデア出しなども見られ、学校での開催ということで、

注1 http://itnav.jp/

いつものハッカソンとは違った雰囲気で行われ、3日間をかけてそれぞれが最大の成果をめざして開発に取り組んでいきました。

今回の石巻ハッカソンでは通常のハッカソンとは違い、さまざまな人が参加できるようプログラムの習熟度別に「IT Boot camp部門」、「チャレンジング部門」、「どや部門」と3部門を設け、同時並行で開催されました。

「IT Boot camp部門」は、開発の経験がない人でも3日間でアプリを実装するところまで体験してもらうことを目的とした部門です。

◆写真2　IT Boot Camp部門ではハンズオン形式で講習が行われた

「チャレンジング部門」は、開発を学んでまだ日が浅い若者を対象にした部門です。この部門の参加者で注目されていたのは、昨年のIT Boot campで初めて開発を経験した第1回石巻ハッカソンの卒業生や東北で定期的に開催されている開発講習イベント「東北TECH道場」に通いスキルを着々と身に付けてきた若者たちでした。

「どや部門」は、開発経験者で仕事として携わっている、あるいはすでに何個もアプリを開発しているなどといった高いスキルを持った方たちが参加する部門です。「石巻の若者たちに3日間で最高にすごいものを作ってレベルの高さを見せつけ、触発する」という意味合いから「どや部門」と命名されました。次からぞれぞれの部門について具体的に紹介します。

初めてのプログラム！ IT Boot Camp部門

皆さんは初めてプログラムに触れて、動くものが作れたときの感動を覚えているでしょうか？「IT Boot Camp部門」は中高生を中心に、開発経験のない人でも3日間の開発を通して実際に動くAndroidアプリを実装し、プログラムの楽しさを知ってもらう部門です（**写真2**）。

この部門では、本誌2012年11月号で紹介した昨年の石巻Bootcampと同様に「Corona SDK[注2]」という、誰でも高度なスマートフォンアプリケーションが簡単に開発できる開発キットを採用しました。プログラム経験がない、あるいは浅い中高生への教育で利用実績があり、また評判も教育効果も非常に高かったためです。

Corona SDKについて改めて紹介しますと、サンフランシスコのCorona Labs社が開発、販売しているAndroid/iOSをターゲットとしたマルチプラットフォームアプリケーション開発のためのフレームワーク、およびSDKの総称です。CoronaはOpenGL ESのグラフィックス処理とLua言語によるスクリプティングにより、画面へのコンテンツ描写が処理の中心となるゲームなどの2次元アプリケーションの開発に適しています。これによりスマートフォンのタッチスクリーンを活かしたゲームアプリケーションが比較的容易に実装でき、初めてプログラミングを経験する中高生にプログラムの楽しさを知ってもらうにはうってつけの開発環境だと言えます。

プログラムが動くことに目を輝かせる生徒たち

IT Boot Camp部門の講師陣は石巻工業高校より

注2　http://www.coronalabs.com/

阿部吉信先生、Corona SDK Ambassadorの小野哲生氏、Hack For Japanからは及川卓也と鎌田篤慎が参加しました。参加者は地元の高校生が中心でしたが、茨城から参加した中学生、大学生なども参加し、約14名のクラスとなりました。この部門では、阿部先生による講義からその講義に基づいた実装を行い、小野氏、及川、鎌田の3名で参加者の中高生をフォローしていくという流れでした。

初日は石巻ハッカソンの開会式の後、教室で講師陣の紹介を終えて、さっそく講義に入りました。まずはCorona SDKを動かすための開発環境の準備です。生徒の皆さんの席に配置されたパソコンへのCorona SDKのインストールのほか、開発に必要な各種環境をインストールしました。まだこうした作業に不慣れな生徒も多く、講師陣でひととおりの準備をサポートする必要がありましたが、中には自分の持ってきたノートパソコンに環境を構築しようとする意欲のある生徒もいました。

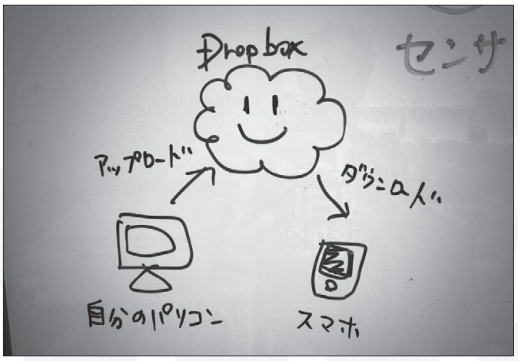
この部門では、実機で動作するAndroidアプリケーションを開発することを目的としていたため、参加する生徒全員にAndroid端末が用意されました。自分のスマートフォンがAndroidの生徒は自分のものを、持っていない生徒はこの日のために寄贈されたAndroid端末を利用しました(**写真3**)。

ようやく全員の開発環境が整ったところで、今回の開発の導入としてCorona Wiki[注3]に掲載されているサンプルコードを実装し、アプリ上での文字や図形の描画と、それらにタッチして動作する物理エンジンの使い方を学んでもらいました。実装したプログラムから生成したapkファイルをDropboxにアップロードし、各自のAndroid端末にDropbox経由でインストールすると、自分で実装したプログラムがスマートフォンでタッチして動かせることに目を輝かせて喜ぶ生徒もいました(**写真4**)。こうした感動がプログラムを学ぶうえで一番大切なことです。

実際、Corona Wikiのサンプルコードはよくできており、順を追って実装していくと開発されるアプリケーションが拡張され、より動きのあるものができあがっていきます。生徒たちが自分のプログラムのステップアップを体感する、その片鱗を講師として実感しながら、初日は無事に終えました。

◆写真3　スマートフォンで動作を確認する参加者

◆写真4　Dropboxからスマートフォンへのインストールの説明図



オーサリングツールを使ってリッチなゲームアプリにも挑戦

IT Boot Camp部門の2日目は、初日のサンプルコードの拡張を続けていたのですが、ひととおりのサンプルコードの実装を終えた一部の生徒から、自発的にオリジナルのプログラムを実装し、自身のアプリケーションにサンプルにはない動きを盛り込む生徒も現れはじめました。こうしたところからハッカーと呼ばれる人種は生まれるものですから、講師として喜びを感じる一幕でもありました。

注3　http://corona.xpf.io/index.php/%E3%83%A1%E3%82%A4%E3%83%B3%E3%83%9A%E3%83%BC%E3%82%B8
注4　http://www.nerderer.com/Gumbo/

2日目の後半は「Gumbo[注4]」というオーサリングツールを使うことで、GumboとCorona Wikiのサンプル素材を利用したAngry Birdsのようなよりリッチなアプリケーションの開発に取り組みはじめました。Corona Wikiに用意されているさまざまなキャラクターを含むサンプル素材は動作する成果物をより魅力的にしてくれ、こうした物理エンジンを活かしたゲームアプリケーションを初めて開発する者の学習意欲をさらに高めてくれる材料となります。初めてプログラミングを経験した中高校生は、自分が実装したプログラムがAngry Birdsのようなゲームになったことに、初日のサンプルコードが動いたとき以上に目を輝かせてるようでした。そういう意味で、今は私たち大人がプログラムを始めた若かりしころより、はるかに子供たちがプログラムを学ぶのに適した時代だと言えます。

◆写真5　発表前、参加者のひとりが作ったアプリを全員で確認

しかし生徒によっては、まだ不慣れなキーボード入力作業と、まだ未学習な英語の単語がたくさん現れるコードを見ながらの開発では、スペルミスやサンプルコードの実装ミスなど、ちょっとしたところでつまずいてしまいます。そうしたところを講師陣がフォローしつつ、脱落者が出ない形で2日目も無事に終えることができました。

2日間の学びを活かして自主開発！

そして他部門の発表も控える最終日では、Corona Wikiで公開されている素材やこれまでに学んだ知識を使い、生徒の自主性に任せて2日目に作ったゲームアプリケーションに改良を加えていくスタイルで開発のフォローを行いました。初めてプログラムに触れた生徒たちも3日目となると、自分の発想をプログラムに反映させることが少しずつできるようになっていきます。また、自分が作ったAndroidアプリケーションを自分の端末にインストールする作業も手慣れたものです（**写真5**）。

◆写真6　IT Boot Camp部門生徒によるアプリの例

こうなってくるとプログラムは、がぜん楽しくなっていきますし、講師としては“しめしめ”といったところです。このIT Boot Campが終わった後も、自分のパソコンを使ってプログラムにチャレンジする生徒も出てくるかな？　と予想していたところ、「家でもプログラムをしようと思いましたか？」という質問に元気に手をあげる生徒の数も多く、3日間にわたったIT Boot Camp部門の講師を務めた講師陣もやりがいを感じた瞬間でした。

3日目の成果発表でIT Boot Camp部門の生徒たちはどのようなアプリを披露するのでしょうか（**写真6**）!?　次号では他部門の紹介と合わせてその内容をお伝えします。SD

温故知新
IT むかしばなし

第28回

記憶に残る製品

北山 貴広　KITAYAMA Takahiro　Twitter：@kitayama_t　kitayamat@gmail.com

はじめに

今回は今まで本連載の中で取り上げなかった、成功しなかったけれど筆者の記憶の中に印象深く残っている製品について振り返ってみたいと思います。

Apple III

1980年にアップルから発売された「Apple III」は、その名のとおりApple IIの後継製品としてビジネス用途に特化して製作されました。1977年に発売されたApple IIが成功した後、1984年にMacintoshが世に出るまでの間に出た製品はApple IIIとLisaがありました。Apple IIはゲームやプログラミングなどのホビー用途に加えてキラーアプリのVisiCalc[注1]などでのビジネス用途でも売れ、多数の拡張スロットとオープンな仕様から生まれた拡張性も手伝って大成功しました。

Apple IIIはスティーブ・ジョブズが外見デザインを重視して静音性を高めるために筐体にファンを取り付けなかったため、内部が高温になりコンピュータの動作が不安定になりました。また高価でIBM-PCがすでに普及していたこともあり成功に至りませんでした。

PC-100

「PC-100」は1983年に日本版の「Alto[注2]」を目指して、PC-8800シリーズの部隊が開発しNECから発売されました。同じころ、NECからはPC-9801Fという2DDのフロッピーディスクが装備されたモデルが発売されていました。720×512ドットの高解像度ブラウン管ディスプレイ（PC-9801シリーズは640×400ドット）が縦置き横置きの両方に対応しているという当時としては珍しい機能をもっており、マウスを備えた先進的なグラフィカルユーザインターフェースを備えてました。PC-100自体はN-5200やPC-9800シリーズとの競合や高い価格もあって製品としては成功しませんでしたが、GUIを用いた先進的な製品はアップルのLisaと同じだったのかと思います。

また本製品にバンドルされていた「JS-Word」は後にPC-9800シリーズでベストセラーとなった一太郎のベースとなりました。本製品にバンドルされていた「Multiplan[注3]」が三菱のMULTI16用に開発されていた歴史と同じような軌跡をたどりました。

Lisa

1983年にアップルから発売された「Lisa」は、ジョブズの娘の名前からとったエピソードで有名です。当時としては最高水準のグラフィックス機能とマウスを使った先進的なユーザインターフェースでした。ただ非常に高価で、求めた機能に比べてアプリケーションの動作などは快適ではなかったと想像しますので、個人が所有するパソコンとしての値段ではなかったのだろうと思います。しかし本製品は後のMacintoshのベースとなり、先進的な製品として印象深

注1） 世界初のパソコン向け表計算ソフト。

注2） ゼロックスで作られた、パソコンの試作機。ワークステーションの元祖。

注3） MicrosoftのGUI式表計算ソフト。Excelの前身。

いものでした。

MZ-2861

1987年にPC-9801をソフトウェアでエミュレーションするという画期的な製品「MZ-2861」がシャープから出ました。このころはEPSONが国民機と称してPC-286などのコンパチ機を発売してNECとやりあってました。ソフトウェアをエミュレーションで動かすというのはかなり大胆な出来事だと思います。この製品は専用ワープロの「書院」としても使える点でもすごく珍しい製品です。

ソフトウェアのエミュレーションで、そもそもどのソフトがキチンと動くのか、どのくらいの遅さなのか使い物になる程度のスピードなのか情報がなかったこともあり、中途半端な製品となってしまって最後のMZシリーズと結果的になってしまったようです。

BeOS

1990年にアップルで開発責任者を務めたジャン＝ルイ・ガセーらが設立したBe社で、ハードウェアの「BeBox」とOSの「BeOS」が開発され、1995年にPowerPCベースのBeBoxとBeOSが公開されました。BeOSはとても洗練されたOSで技術者の間ではとても人気がありました。

1996年にPowerMacintoshに移植され、Macintoshの次期OSの有力候補となりました。すでに買収されていたNeXT社のNEXTSTEPと次期OSの座を争いましたが敗れて、その後Intelプラットフォームへ移植され、最後は2001年にPalm社に知的資産を売却されBe社は解散しました。OSとしての機能は画期的でしたが、新規OSゆえ、アプリケーションはほとんどなく商売として継続して生き残るのはたいへんだと感じました。

3DO REAL

「3DO REAL」は1993年に北米で、1994年に日本で松下電器産業が発売したゲーム機です。3DO社は1990年にゲームメーカーのエレクトリックアーツの創始者の1人が設立した企業で、1993年に32ビットマルチメディア端末の規格「3DO」を発表しました。3DO REALは3DO規格を実装した単なるゲーム機ではなくマルチメディアプレーヤという位置づけで発売されました。ゲーム機としてはたいへん高価な価格付けで発売されたゲームも欧米ものの移植が多く、セガサターンやプレイステーションがその後に発売されたこともあり、本製品は普及しませんでした。

バンダイ ピピンアットマーク

1996年にピピン構想を実現した唯一の製品としてバンダイ・デジタル・エンターテイメントがアップルと共同開発したゲーム機です。モデムを使ってインターネットに接続できるマルチメディア機として発売されMacintoshのOS 7.5が動作しました。当時はまだアップルがMac互換機路線を取っていたころ（1995～1998年）でしたがアップルが非協力的であったことと高価な値段付けと魅力的なコンテンツが提供できなかったため1998年に撤退しました。GamePro.comサイトで、世界で最も売れなかったゲーム機ベスト10で第1位を獲得したと紹介されています[注4]。もし在庫処分で安く発売されているのを見かけたら、つい買ってしまいそうな製品でした。

終わりに

Apple IIIのデザイン重視は当時の製品の発熱量や集積度では実現できませんでしたが、後のiMacなどの製品では実現できているのだろうと思います。LisaとPC-100のGUIを採用した挑戦的で先進的な試みは、Macintoshを含めて今ではマウスを使ったGUIが当たり前になっています。MZ-2861のソフトウェアエミュレーションは当時のハードウェアの速度では無理だったでしょうが、現在の高速なハードウェアでは実用的な速度で使えると思います。ほかの製品についても、魅力的な機能や用途やコンテンツをほど良い価格とタイミングで提供できるかどうかが鍵を握っているなと感じました。SD

注4）http://gigazine.net/news/20070507_worst_selling_consoles/

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。

Service

さくらインターネット、「さくらのVPS」においてプラン拡充／機能追加を実施

さくらインターネット㈱は、「さくらのVPS」サービスにおいて複数のリニューアルを実施し、10月31日より提供を開始した。改訂内容は次のとおり。

- サーバ間のローカル接続による複数台構成を可能とする新機能「ローカルネットワーク」を提供開始
- 「VPSコントロールパネル」において、サーバ一覧表示機能、複数台サーバの一元管理／操作機能、「ローカルネットワーク」設定／管理機能を新たに提供
- マルチリージョン展開の拡充として、石狩／大阪リージョンに加え、東京リージョンが選択可能に。また、石狩リージョンにて新たに「4Gプラン」「8Gプラン」を追加
- 「さくらのVPS SSD」にさらにハイスペックな「SSD 8Gプラン」「SSD 16Gプラン」を追加
- IPv6アドレスを標準付帯。IPv6での通信が可能に

また今回のリニューアルを機に、「さくらのVPS 2G」の初期費用1,980円を無料とするキャンペーンを、2014年3月31日まで実施している。

CONTACT さくらインターネット㈱
URL http://www.sakura.ad.jp

Topic

LPI-Japan、学生向けLinux/OSS体験セミナー「Linuxを使ってみよう」寄付講座を実施

特定非営利活動法人エルピーアイジャパン（以下、LPI-Japan）は、高校生など若者向けの実践的なLinux/OSSのハンズオンセミナー「Linuxを使ってみよう」の寄付講座を実施するプログラムを発表した。

このプログラムは、若い世代にLinux/OSSに触れる機会を作り、将来の日本のIT人財の育成に貢献したいとの意図から、LPI-Japanが中学、高校、高専、専門学校、大学などにおいて、Linuxの体験講座を無償で実施するというもの。

本発表に先立ち、千葉県立商業高等学校と北海道旭川工業高等学校で寄付講座が実施された。講座の様子はLPI-JapanのWebサイトにて公開されている。

LPI-Japanでは、今後も教育機関からの要請に応えるかたちで、セミナー実施の希望がある教育機関と協力し、学生がLinux/OSSに実際に触れる機会を提供していくという。

■教育機関問い合わせ先

https://www.lpi.or.jp/school_contribution/

CONTACT 特定非営利活動法人エルピーアイジャパン
URL http://www.lpi.or.jp

Topic

リンク、「PCI DSS Ready Cloud」をビジネスモデル特許出願

レンタルサーバサービス「at+link」などを展開する㈱リンクは、クレジットカードのデータを取り扱う際に必須とされる国際的なセキュリティ基準「PCI DSS」の準拠を低コストでスピーディーに実現できるクラウドサービス「PCI DSS Ready Cloud」について、10月24日にビジネスモデル特許を出願した。

同サービスは、2013年5月から提供を開始しており、PCI DSS準拠に必要なリソースのすべてをクラウド上に用意しPaaSとして提供することで、低コストかつ短期間で準拠できることを特徴としている。

同社調べによると、他社でもPCI DSSに準拠をしているクラウドサービスは存在するが、前述のような形式で提供しているサービスはPCI DSS Ready Cloud以外にはないという。そのため、同サービスを独自性の高いクラウドサービスとして、競争優位性を維持し続ける目的で、特許出願することに決めたとのこと。

今後、クレジットカードを取り扱う事業者がますます増えるなか、同社はこれからもさまざまなサービスをPCI DSS Ready Cloud上で展開していくという。

CONTACT ㈱リンク
URL http://www.link.co.jp

Hardwere

ウエスタンデジタルジャパン、外付けデスクトップハードディスクドライブの新ラインアップを発表

ウエスタンデジタルジャパン㈱は、外付けデスクトップハードディスクドライブ「My Book」の新ラインアップを発表、10月18日より発売した。

高速USB 3.0対応の新「My Book」シリーズは、2TB、3TB、4TBの3モデルを用意し、大容量コンテンツの保存に最適な製品となっている。

また、今回の新製品では、より充実したデータバックアップ機能を提供している。まず、自動バックアップソフトウェア「WD SmartWare Pro」が「Dropbox」と連携することにより、クラウドへのバックアップが可能となった。次に「Acronis True Image WD Edition」に対応することで、システム全体の丸ごとバックアップもできる。

価格は、2TBモデルが14,800円、3TBモデルが17,800円、4TBモデルが25,800円（すべてメーカー希望小売価格（税込））となっている。

▲My Book

CONTACT ウエスタンデジタルジャパン㈱
URL http://www.wdc.com/jp

Hardware

ぷらっとホーム、中大規模ネットワーク向け高性能DHCPアプライアンスの新モデルを発売

ぷらっとホーム㈱はネットワークアプライアンス「EasyBlocksシリーズ」に、上位クラスの機能を実現した新型DHCPアプライアンス「EasyBlocks DHCP 1500モデル」と「EasyBlocks DHCP 3000モデル」を発売した。価格はいずれもオープン。

1500モデルは1500個のIPアドレスを割り当てでき、3000モデルは3000個を割り当てできる。本モデルから追加された新機能は次のとおり。

- Active-Active方式の冗長化に加え、Active-Standby方式にも対応
- 稼働状態に加え、冗長化構成時のActive側／Standby側を本体ステータスLEDにより可視化
- 不適切な設定を入力した場合には警告を出すことでトラブルを防止
- IPアドレスを払い出した部署名、個人名の入力／確認ができ、重複割り当てを防ぐ
- Webインターフェース上でログの視認やダウンロードが可能

CONTACT ぷらっとホーム㈱
URL http://www.plathome.co.jp

Software

アドバンスソフトウェア、Excelシートから HTMLファイルを展開する「Excel to HTML」を発売

アドバンスソフトウェア㈱は、Microsoft Excelのシートを HTMLに展開するソフトウェア「Excel to HTML」を10月24日に発売した。

同製品には、専用のユーザインターフェースで操作するアプリケーションとASP.NETで使用できるサーバコントロールおよびスクリプトが付属している。Excelシートの内容をHTMLテーブルとJavaScriptに展開する。単なるHTMLコンバータではなく、展開したHTMLでは入力ができ、スムーズなカーソル移動も実現している。罫線やフォント、背景色などのExcel上で設定した一部の書式や計算式もHTMLに反映される。

サーバコントロールではJavaScriptを使用して、HTMLで入力した内容をJSON形式で取得可能。これを利用して、HTMLに入力した内容でExcelに書き戻したり、データベースを更新したりできる。

価格はHTMLテーブル作成アプリケーションの「ExcelWeb Tool」が1ライセンス12,600円（税込）。ASP.NET用のコントロールおよびスクリプトをサーバ上で使用するには、別途サーバライセンスが必要。

CONTACT アドバンスソフトウェア㈱
URL http://www.adv.co.jp

Letters from Readers

SD編集部員のエディタ

10月号のVim、11月号のEmacsと2号続けてのエディタ特集はいかがでしたか？　身近なツールだけにアンケートのコメントはいつもより熱いものばかりでした。さて、編集部ではどんなエディタを使っているかというと、WZ EDITORを使っている人が一番多いです。編集作業をする際に、アウトライン機能や用語統一機能が便利なのです。ほかには、秀丸、Emacsを使っている人もいますよ。

2013年10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　Vim至上主義

多くのLinuxディストリビューションにも標準で搭載され、サーバエンジニアのデフォルトエディタとも言えるvi/Vim。そのVimを使いこなしている達人たちに使いこなす術を紹介してもらいました。

エディタというのはエンジニアにとってとても大切なツールのひとつ。エディタを個別に掘り下げて特集してもらえるのはとてもありがたい。11月号のEmacsにも期待しています。

長崎県／romeosheartさん

確かにどんな環境にもインストールされていて、起動も速く、GUIがなくても安心して使える。しかも、昔のviと違って多機能になった。20年以上に渡りEmacs派でしたが、浮気をしたくなりました。と、思いにふけっている中、11月号予告を見ると特集「我が友Emacs」となっている。どうすればいいのだ!!

福岡県／池内さん

VimでObjective-Cプログラミングをする記事などが参考になりました。ですが、『実践Vim』発売直後ということもあり、内容にもの足りなさを感じました。

東京都／ryogさん

今まで大学の課題や研究のためのコードを書くのに、GUI版のEmacsを使っていました。しかし最近、インターンでサーバサイドの開発をするようになり、Vimを使い始めようと思っていたところだったので、うれしい特集でした。基本の基本からちょっとした応用までが、わかりやすくまとめてあり、当分はこれを参考にしながら勉強しようと思います。

東京都／時武さん

viはコード入力と設定ファイル編集に使用しています。もっと精進したいです。

神奈川県／米太郎さん

効率的なキー入力の方法や、個別のカスタマイズなど、自分流にアレンジして使えるのが、エンジニアに好かれる所以でしょうか。そういう意味では11月号の特集のEmacsも同じ。みなさんからどんな声が聞けるか、楽しみです。

第2特集　ルータの教科書

ルータの基礎である経路選択のしくみやルーティングテーブルについて押さえつつ、トンネル制御、優先制御、BGPなどより深い知識について解説しました。

Ciscoルータのシミュレータの話やVyattaについては、ページ数がもっとほしかった。また別途特集で取り上げてほしい。

東京都／Blackbirdさん

久々にネットワークの勉強をできたという感じになりました。

東京都／村川さん

基礎的なことが多かったけど、BGPまであったのが良い。

東京都／hiddenさん

今号で特集したSDN/OpenFlowのネットワーク仮想化技術も、ルーティングなどの基礎知識があやふやでは、安心して導入／運用できません。しっかり理解しておきたいものです。

一般記事　Key Value Storeをゼロから創る

分散KVS「okuyama」の開発者自らが、okuyamaの機能、開発経緯、設計思想、啓発活動、今後の開発予定などについて紹介してくれました。

既存の資源からの移行、OSSとの連携、拡張性、耐障害性についても、もう少し言葉を並べてもよかったと思う。

東京都／HL.Tさん

NoSQL系データベースについては日本

語の資料が少ないので非常にうれしい。

東京都／ひろし@巣鴨さん

突っ込んだところまでは書けないとしても、苦労やとっかかりの部分の規模みたいなものが見えた。ほかの案件でもWBSを作ったりする参考になります。

千葉県／野崎さん

NoSQLデータベースも多くの種類がありますが、それぞれの特徴や違いを理解するのはなかなか難しいです。開発者自身による開発の動機、設計思想などについての説明は、そういう意味で参考になるのではないでしょうか。

短期連載　小規模プロジェクト現場から学ぶJenkins活用

Skypebotの制作方法や運用、そしてJenkinsからGUIツールを動かす方法について解説しました。

UWSCで回す苦労のところをもっと知りたかったです。GUI経由で動かすとうまくいってくれないこともあるので、その辺をどう対応されているのかなぁ、と思いました。

兵庫県／yoneさん

しくみや理屈はわかっているが、新規ではなく途中からプロジェクトに参画すると実際に利用するのはハードルが高いので、どちらかというとシステムとしての機能より活かし方がより知りたい。

東京都／ネムリンさん

Jenkinsは使ってみたいと思うけれど、いざ導入すると一筋縄ではいかないことが多いようですね。本連載が少しでもヒントになれば幸いです。

フリートーク

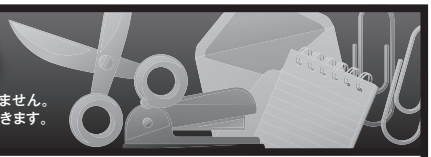
『Software Design総集編』を購入しました。デジタル版のバックナンバーとの違いはありますか。

大阪府／牧さん

総集編のご購入ありがとうございます。総集編に収録しているバックナンバーはPDFですが、Fujisan.co.jpなどで販売しているデジタル版はPDFでの配布ではなく、専用アプリでご覧いただく形式となっています。また、総集編は特集、一般記事、連載以外の記事や広告は未収録ですが、デジタル版は紙版と同じくすべてのページをご覧になれます。

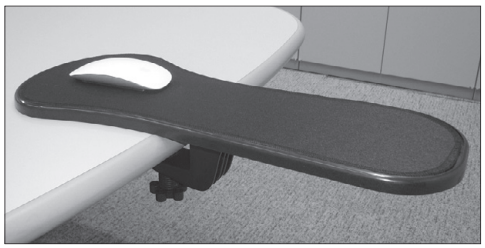
エンジニアの能率を高める一品

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。



エルゴノミクスアームレストMR-TOKERG1

3,675円（税込）／サンワサプライ㈱　http://www.sanwa.co.jp



◀写真2
肘あてが付いた椅子にも取り付けられる。肘あて上部の面積が幅7cmまで、厚み5cmまでの椅子で使用可能

今回試したのはマウスパッドも兼ねた肘置き台。PCを使うとき、マウスを自分の体のほうに引き寄せて操作していると、肘がだるくなってきますが、それを解消してくれる一品です。机に取り付けて使うと（写真1）、肘が支えられてとても楽です。今までは肘が不安定だったことで、肩や腕に余計な力が入っていたのだなあ、と実感しました。購入する前にはくれぐれも机との相性を調べてください。本製品は厚み4cmまでの机にしか取り付けられません。編集部の机は引き出しがあるので、それを取り外して試用しました。また、マウス操作が改善されると、次はキーボードを使うときに、同様に肘が浮いてだるいのが気になってしまいました。キーボード使用時に肘を支えてくれる「エルゴノミクス肘置き台」という製品もあるようですので、プログラマにはそちらのほうが合っているかもしれません。（読者プレゼントあります。p.16を参照）

▲写真1　机に取り付けて使用

10月号のプレゼント当選者は、次の皆さまです

①Samsung SSD 840 EVO ベーシックキット 250GB ..... 京都府　板東修平様
②Super Win Utilities 3 ..... 兵庫県　青木大我様

★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
January 2014

2014年1月号
定価1,380円　184ページ

12月18日発売

［第1特集］シェルがわかればシステムがわかる

あなたの好きなシェルは何ですか?

シェル誕生の歴史から振り返り、それぞれのシェルの設定ファイルの書き方を学び、UNIX（Linux・FreeBSD）をより深く理解します。それによって、仮想環境・クラウド環境でのシステム展開や応用、運用、ユーザ管理などが自信をもって実施できるようになるのが本特集のゴールです。

［第2特集］未来を作るITインフラ

10Gbを実現するケーブリング技術

GigaBitの高速ネットワーク技術をまるごと紹介します。クラウドコンピューティングの普及の影響で、サーバマシンの処理能力が高機能・高速化しています。そこでネットワークも広帯域・高速化を実現することが求められています。本特集では10Gbを支えるケーブリング技術についてその導入ポイントを紹介します。

［特別付録］法輪寺電電宮情報安全護符 Ver.2.0

［一般記事］魔法少女リリカルなのはINNOCENTの舞台裏（後編）

［新連載第1弾］小飼弾の「SDでSF」

［新連載第2弾］啓蒙Linux通信（1ページ読み切り特殊記事）

※特集・記事内容は、予告なく変更される場合があります。あらかじめご容赦ください。

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11月号

- P.16　「読者プレゼントのお知らせ」プレゼント02の製品名
  ［誤］ESET FAMILY SECURITLY
  ［正］ESET FAMILY SECURITY
- P.17　第1特集トビラページ「Chapter 4　いつもの環境がどこでも使える!　絶妙の引きこもり型エディタ」のページ番号
  ［誤］p.50
  ［正］p.39
- P.150　連載「Debian Hot Topics 第9回」注2
  cloud-packages@lists.alioth.debian.orgの前にURLマークが付いていますが、URLではなくメールアドレスでした。

休載のお知らせ

「システムで必要なことはすべてUNIXから学んだ」（第10回）、「IPv6化の道も一歩から」（第10回）、「＜ネットワークエンジニア虎の穴＞自宅ラックのススメ」（第8回）は都合によりお休みいたします。

SD Staff Room

●今期は『キルラキル』がおもしろい。一見ラフに見える画面でも作り込まれていて素晴らしい。『グレンラガン』も良かったが、さらに上を行く作品だ。そうなると思う。最初はちょっと敬遠しがちなストーリーとキャラだったけど、制作陣の懐の深さとか感じられていいなぁと思う。そういうふうにしたいな。（本）

●iPad AirとiPad mini Retinaがやっと発表になった。速度も上がって使い勝手はよさそう。全天球カメラのRICOH THETAもマニア心をくすぐる。Apple TVも導入したいが液晶TVが古くてHDMI対応していないので買い換えたい。検討事項の多い物欲の秋……。（幕）

●携帯電話をウィルコムのPORTUSに。iPod touch、Nexus 7、enchantMOONをつないでいます。不通の場所（地下が厳しい!　電話はつながるヨ）での対策として、Nexus 7にはDTIの490円SIMを搭載。月額4,000円弱くらいとサイフに優しくなりました!（キ）

●スマホ全盛の今でも、スケジュール管理は紙の手帳でやっています。自分流の使い方がすっかり定着しているので、毎年同じメーカーの手帳を購入。スケジュールだけでなくタスク管理にも使っています。昔の手帳を見れば、当時、自分がしていたこともわかるので、ちょっとした日記でもありますね。（よし）

●このたび実家暮らしから卒業することになりました。この号が出るころには新生活を始めている予定ですが、編集後記記入時は片づけや準備でバタバタ。色々なところから使えるものを集めていますが、年季が入っているものも多く、買い換えも必要。決めなければいけないことも多く、引っ越しは大変だと実感中。（ま）

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12月号

発行日
2013年12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

●編集アシスタント
松本涼子

●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。