special feature 1 LAMP環境強化
special feature 2 ネットワークとストレージを見定める

Software Design

2014年11月18日発行
毎月1回18日発行
通巻355号
（発刊289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

2014
11

安定の
LAMPを
変化に
強くする!

無理なくはじめる

special feature 1

Infrastructure as Code

Docker
Ansible
シェルスクリプト

ウェブ開発で知っておきたい構築テク

special feature 2

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］
ネットワークと
ストレージを極める

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

新連載

Raspberry Piに耽溺
おとなラズパイリレー

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71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。

MEANスタック

MEANスタックとは

近年、Webアプリケーション開発の現場で「MEANスタック」という用語が使われるようになってきました。MEANスタックとはJavaScriptを軸としたWebアプリケーションの開発およびデプロイメントのためのアプリケーションスタックを指す言葉で、次に挙げる4つのツールの頭文字に由来したものになっています。いずれもJavaScriptを活用した技術であり、サーバサイドからクライアントサイドまでを一貫してJavaScriptで開発できるのがMEANスタックの特徴となっています。

[1] MongoDB

MongoDBはMongoDB社によって開発・サポートが行われているデータベースシステムです。NoSQLに分類されるドキュメント指向のDBシステムであり、「ドキュメント」と呼ばれるJSONライクな構造的データと、「コレクション」と呼ばれるドキュメントの集合によってデータを管理します。RDBMSで行えるような高度な結合操作は得意ではない代わりに、大規模なデータに対して、追加や更新、削除などの操作を高速に行えるという強みを持っています。

[2] Express

ExpressはNode.js用のWebアプリケーションフレームワークです。Node.js上でWebアプリケーションを構築するために必要となるさまざまな機能を備えており、最低限の設定とコーディングでアプリケーションの基本構造を組み上げることができます。

[3] AngularJS

AngularJSは、JavaScriptを用いたWebアプリケーションの開発に対してMVCアーキテクチャを取り入れることを目的に開発された、オープンソースのJavaScriptフレームワークです。モデルとなるJavaScript内のデータオブジェクトとビューとなるHTMLとの間で、双方向のデータバインディングを極めて簡単に実装できる点が大きな魅力です。

[4] Node.js

Node.jsはサーバサイドで動作するJavaScript実行環境です。ノンブロッキングI/Oを利用することによってシングルスレッドベースでの非同期通信を実現しており、サーバとクライアント間でリアルタイムなデータ交換を行えるという点が大きな特徴です。Node.jsのようなサーバサイドのJavaScript実行環境は、サーバとクライアントで同じプログラミング言語を用いてシームレスな開発を行えることから人気が高まっています。

LAMPスタックとMEANスタック

MEANスタックという用語は、従来一般的に使用されてきた「LAMPスタック」に対比する形で生まれました。LAMPスタックは、Linux、Apache、MySQL、そしてPHP/PerlまたはPythonを軸にしたアプリケーションスタックであり、オープンソースのツールで構成され、Webアプリケーションに必要な一通りの要素を備えているという特徴があります。

ここで重要なのは、「LAMP」や「MEAN」という用語はあくまでもアプリケーションスタックの特性を象徴するものであり、ツールの種類を限定するものではないということです。LAMPスタックは「OS＋Webサーバ＋RDBMS＋スクリプト言語」という構成の象徴として使われてきました。それに対してMEANスタックは、「サーバサイドのJavaScript実行環境およびフレームワーク＋NoSQLデータベース＋クライアントサイドのJavaScriptフレームワーク」という構成を象徴しています。

MEANスタックのほうはNode.jsがOSおよびWebサーバのレイヤの役割を担っており、アプリケーション側から見たOSの差異はNode.jsが吸収しているという特徴があります。また、MEANではクライアントサイドでJavaScriptを活用した動的なUIの構築が想定されているという違いもあります。そして動的なUIのためにJavaScriptが必須技術となったことから、サーバサイドもJavaScript化して開発効率を上げるという近年のトレンドが反映されています。

最近ではMEANスタックの構築・デプロイをサポートするツールやサービスも登場してきました。MEANスタックは、Webアプリケーション開発の新しいトレンドを代表する要素の1つと言うことができるでしょう。SD

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

Contents

Software Design Nov. 2014

［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

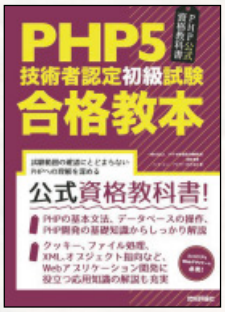
あなたを合格へと導く１冊があります！

これだけで突破［合格ライン］LPIC合格読本 レベル1攻略

中島能和、矢越昭仁　著

ISBN978-4-7741-6672-8　B5判／148ページ　定価（本体1980円+税）

LPIC（Linux技術者認定試験）取得希望者の多くは、資格取得を決めてからの準備期間が限られています。そこで本書では、LPICレベル1を対象に、「このポイントを押さえておけば合格ラインを突破できる」という最重要ポイントを厳選した解説と演習問題をお届けします。また、単なる試験対策の知識ではなく、現場で役立つ知識を最短距離で効率よく学習するには、どのようにすればよいか、仕事をしながらゼロから3週間で確実に合格するための学習法を詳解しました。さらに、LPIC取得者による合格体験記＋業務での活用レポート、そして2015年に予定される試験改訂情報も掲載しました。



PHP5技術者認定初級試験合格教本 PHP公式資格教科書

酒徳峰章、石本和大　著

ISBN978-4-7741-6692-6　A5判／336ページ　定価（本体2980円+税）

PHP技術者認定機構公式のPHP5技術者認定初級試験に対応した資格テキスト。各章末付属のPHP5技術者認定初級試験に準拠した練習問題を解くことで、試験対策はバッチリです。PHPの基本文法をはじめ、オブジェクト指向や各種ライブラリなど、PHPを使いこなす上で必須知識の解説も充実しています。

Ruby公式資格教科書 Ruby技術者認定試験 Silver/Gold対応

増井雄一郎、小川伸一郎、藁谷修一、川尻剛、牧俊男　著
Rubyアソシエーション、CTCテクノロジー　監修

ISBN978-4-7741-5001-7　A5判／512ページ　定価（本体3600円+税）

Ruby技術者認定試験Silver/Goldに対応した標準的な教科書です。Rubyの文法をはじめ、オブジェクト指向や各種ライブラリなどRubyを使いこなす上で必須の知識がしっかり身につきます。
巻末にRuby技術者認定試験Silver/Goldに準拠した練習問題が付属します。

最短突破 Cisco CCNA Routing and Switching /CCENT ICND1合格教本［200-120J, 100-101J対応］

エディフィストラーニング株式会社　著

ISBN978-4-7741-6269-0　A5判／560ページ　定価（本体3800円+税）

Cisco CCNA Routing and SwitchingおよびCCENT資格取得のための、一番新しい対策テキストです。2013年に改定された新試験にいち早く対応しました！ 本書は、ICND1試験（100-101J）と、CCNA試験（200-120J)のICND1に該当する出題範囲をカバー。わかりやすい解説と豊富な解説図、各章末に用意された練習問題で、資格の取得に必要な技術と知識を効率よく学習できます。スキルアップを目指すエンジニア必携の1冊です。

最短突破 Cisco CCNA Routing and Switching ICND2合格教本［200-120J, 200-101J対応］

エディフィストラーニング株式会社　著

ISBN978-4-7741-6584-4　A5判／688ページ　定価（本体3300円+税）

Cisco CCNA Routing and Switching資格取得のための、一番新しい対策テキストです。2013年9月から始まった新試験に対応しました！ 本書は、ICND2試験（200-101J）と、CCNA試験（200-120J)のICND2に該当する出題範囲をカバー。わかりやすい解説と豊富な解説図、各章末に用意された練習問題で、資格の取得に必要な技術と知識を効率よく学習できます。スキルアップを目指すエンジニア必携の1冊です。

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 Nov. 2014

検索エンジン自作入門

手を動かしながら見渡す
検索の舞台裏

●山田浩之、末永匡　著

ISBN978-4-7741-6753-4
A5判／224ページ
定価（本体2680円+税）

まいにち使っている検索エンジンがどうやって動いているか、知っていますか？
本書では、小さな検索エンジンを作りながら、ソースコードレベルで検索エンジンのしくみを解説。
Yahoo!Japanの検索エンジン開発チームを経て2008年度上期未踏IT人材発掘・育成事業において高性能分散型検索エンジンの開発によりスーパークリエータに認定された山田浩之氏と、全文検索エンジンSenna／Groongaの開発に携わってきた末永匡氏による、オンリーワンの1冊です。

\手を動かしながら学ぶ／

ビジネスに活かすデータマイニング

人気ブログ「銀座で働くデータサイエンティストのブログ」を運営する現役データサイエンティストである著者が、Rを使ったデータマイニングの基礎から最新の手法まで、ビジネス現場での具体例を交えながらやさしく解説します。技術評論社のサイトからダウンロードできるサンプルデータを用いて実際に手を動かしながら学習していく方式なので、データもRも「使える」力を確実に身に付けることができます。

●尾崎隆　著

ISBN978-4-7741-6674-2
A5判／224ページ　定価（本体1980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 Nov. 2014

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	101cats /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

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。

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

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

データベース技術[実践]入門
松信 嘉範 著
定価 2,580円+税　ISBN 978-4-7741-5020-8

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

最新刊！

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

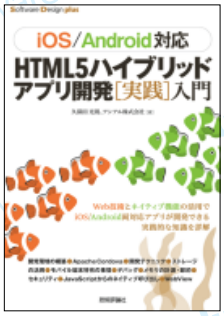
最新刊！

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

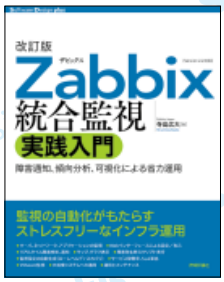
勝俣 智成、佐伯 昌樹、原田 登志 著
A5判・288ページ
定価 3,300円(本体)+税
ISBN 978-4-7741-6709-1

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

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



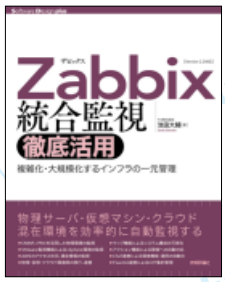
久保田 光則、アシアル㈱ 著
A5判・384ページ
定価 2,880円(本体)+税
ISBN 978-4-7741-6211-9



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



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

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

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

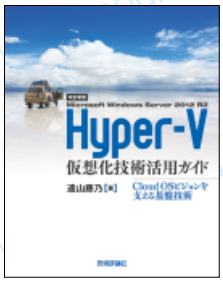
技術評論社

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

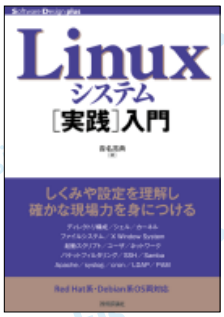
養成読本編集部 編
B5判・212ページ
定価 1,980円(本体)+税
ISBN 978-4-7741-6578-3

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



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

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

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

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

髙橋 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

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

〒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 191

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で開催［研究と展示編］

機械学習から クラウドソーシング活用まで

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SIGGRAPH 2014が8月10日から14日の5日間、カナダのバンクーバーで開催されました。今年は、米国開催の昨年よりも若干少ない、75ヵ国から14,045人の登録参加者があり、150社以上の展示が行われました。

41回目となる今年のSIGGRAPHのテーマは、“NATURALLY DIGITAL”。直訳すると「自然なデジタル」です。最新のデジタル技術を、ごく自然に、なにも特別なことではなく利用する。普通のこととしてデジタル技術を使う、浸透させていくといったイメージです。

今年のSIGGRAPHの発表論文の傾向として、機械学習と、クラウドソーシングを活用したものが多くみられました。

3DCGにおけるアニメーションの動きは、従来は特別なアルゴリズムで数値的に算出するか、熟練アニメーターがツールの助けを借りながら経験をもとに手作業で設定するのが通例でした。正確なアルゴリズムが選び出せない場合は、近似値や似通った結果を素早く算出できる方法が利用されてきました。

しかし最近では、データストレージやコンピューティングパワーが安価になったため、実際の動きを収録したデータを大量に集約し、データの固まりから法則性や細かなニュアンスなども含めて抽出して活用する、という流れになり始めています。数年前ならば手間と費用がかかりすぎて無理だった機械学習的なアプローチも、現実的な1つの手法として選ばれるようになったのです。

クラウドソーシングは、クラウド（群衆）とアウトソーシング（外注）を合わせた造語で、インターネットを介して不

SIGGRAPH会場となったバンクーバー・コンベンション・センター。隣りには冬季オリンピックの聖火台も

アートギャラリーと先進技術展示コーナーの入場口パノラマ写真

※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。

特定多数の人々に細かなタスク（仕事）をアウトソースするしくみです。クラウドソーシングサービスの1つ、Amazon Mechanical Turkでは、人間が行う作業を、数セント単位でクラウド上のアプリケーションの一部に組み込むことができます。とくにSIGGRAPHでは、人間にしかできないこと、人間にしか判断できない情報を大量に、かつ安価で正確な結果を得る手法として活用されています。たとえば、写真に写っている動物が猫なのか、犬なのか。人間であれば一瞬で、ほぼ間違いなく判別できますが、コンピュータの画像解析アルゴリズムだけではサンプル比較に限界があり、目鼻の抽出や形状などでは認識できない場合もあるでしょう。

クラウドソーシングを活用した研究としては、液体のシミュレーションにおいてユーザがよく使う事例をあらかじめ計算しておく方法、イラストの類似度を測るための評価、フォント種別の選び方を改善するツール、日光／蛍光灯／白熱灯などといった照明環境に依存しない本来の色をディスプレイで表現するための評価などに活用されています。とくに視覚の評価では、高齢になると青や緑の色が見えにくくなる事象を改善するため、若年層から高齢者まで広く被験者を集める必要がある場合にクラウドソーシングは重宝されます。

さらに最近の傾向として、文章で記述された論文だけでなく、実際に動作するリファレンス実装のソースコードや実行アプリが公開される場合が多く見受けられるようになりました。再現性や耐性が実用レベルかどうかの判断、チャンピオンデータ（良質な結果が得られる、ある特定のデータ）でしかうまくいかないとかいった疑問や不安を払拭する意味もあります。

先進的なアイデアとヒントの固まり。CG論文の数々

圧倒的なコンピューティングパワーにより進化した研究もありますが、数年来の事象を覆すような、まったく新しいブレイクスルーにはなかなか出逢えません。従来も同じようなことができた事象も、アルゴリズムや手法を工夫し、高速に利用できる「改善」的な研究も一定の評価を得られるようです。

また、普通に考えると面倒すぎたり、あまりニーズがないように思われるような事柄でも、逆転の発想による新しい考え方を取り入れた研究もいくつか見受けられました。最先端の技術開発ばかりではなく、身の回りや、生活の中から研究のヒントを見いだすものもありました。

旧来は新型デバイスそのものの開発が研究の1つでしたが、距離センサー「Kinect」、ヘッドマウントディスプレイ「Oculus Rift」やジェスチャーセンサー「Leap Motion」といった機材が、安価に手軽に入手できるようになり、それらを活用した応用に関する研究も増えてきた印象が強く感じられました。

これからのコンピュータグラフィックスの進化

SIGGRAPHの本分は学会であり、毎年多くの先進的な論文が発表されます。近年、純粋なコンピュータグラフィックスに関する論文だけでなく、デジタルファブリケーション系、音響や

液体のシミュレーション
「Self-Refining Games using Player Analytics」
http://graphics.cs.cmu.edu/projects/self-refining-games/

イラストの類似度を測る
「A Similarity Measure for Illustration Style」
http://webdiis.unizar.es/~elenag/projects/SIG2014_styleSim/

照明環境に依存しない本来の色をディスプレイで表現
「Simulating and compensating changes in appearance between day and night vision」
http://luminance-retargeting.bangor.ac.uk/

フォント種別の選び方を改善するツール
「Exploratory Font Selection Using Crowdsourced Attributes」
http://www.dgp.toronto.edu/~donovan/font/

毛足の長い絨毯を起毛させて絵柄を描く装置
「Graffiti Fur：Turning Your Carpet Into a Computer Display」
http://vimeo.com/102480456

指先のジェスチャでデジタル粘土細工をするツール
「Free FORM：Digital Sculpting With Adaptive Surface Topology and Seamless 3D Coordinate Systems」。Leap Motionを活用。後に商標の関係でSculptingに改名
https://airspace.leapmotion.com/apps/sculpting/

音、動画編集といったCG周辺の技術に関しても研究が進んでいます。

また、論文発表もデジタル化が進んでおり、今年は分厚い書籍の論文集の販売やDVDによる配布はなくなり、すべてオンラインで公開されました。SIGGRAPH会期中は無料で配布されていましたが、現在はSIGGRAPH Digital Library会員以外は有料です。とくに3DCG関連の論文は、論文だけでなく動画や画像でアピールすることも多く、論文の最初のページに研究の要(かなめ)となる画像が入るため、ぱっと見でも、研究の概要を把握しやすくなっています。

●SIGGRAPH 2014アンオフィシャル発表論文リンク集。動画やソースコードへのリンクも
http://kesen.realtimerendering.com/sig2014.html

●各論文の1ページ目が無料で公開されており、概要と主要画像が把握できる
http://s2014.siggraph.org/sites/default/files/firstpages.default.pdf

●SIGGRAPH 2014論文集（PDF、有料）
http://www.siggraph.org/learn/siggraph-2014-open-access-conference-content

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は日本の神戸で開催されることが発表になりました。

日本での開催は2009年に横浜で開催されたSIGGRAPH ASIA以来であり、パワーアップした日本ならではの内容が期待されます。開催日は2015年11月2日～5日の4日間。場所は神戸国際会議場および国際展示場です。規模は北米で開催される本家SIGGRAPHの半分ほどとなりますが、各種セッションや展示はたいへん充実したものになる予定です。SD

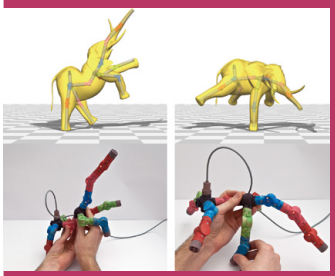
GADGET 1

Tangible and Modular Input Device for Character Articulation

http://igl.ethz.ch/projects/character-articulation-input-device/

アニメーション作成専用モジュール

この研究は、人間以外の四本足のキャラクタや恐竜、エイリアンなど、さまざまな形状のCGキャラクタの動きを着脱自在のモジュールを組み合わせて信号を送信し、データ作成するためのしくみです。デモでは、四つ足の象と、象の鼻をモジュールで表現し、手で動かしたモジュールの形状がCGの象に反映される例を見せていました。各モジュールはつなぎ合わせただけで、どのように接続されているのか構造を認識して扱えます。



GADGET 2

Cyberith Virtualizer

http://cyberith.com/

仮想空間を歩く機器

Cyberith Virtualizerはバーチャルリアリティの世界の中を歩くことができる、ゲーム用の周辺機器です。現在は事前予約のお知らせ中。滑りやすく加工された専用の靴、または靴下など足が滑りやすい状態でVirtualizerの中心に立ち、同じ場所で足踏みすると景色が変化したり、首をめぐらせて四方を自由に見渡したりすることができます。Oculus Riftのようなヘッドマウントディスプレイと併用するのが前提です。

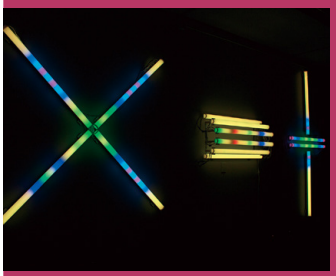
GADGET 3

XEPA

http://philipgalanter.com/art/xepa/images/

音と光の彫刻

XEPAは約6フィート（約2メートル）の筒が音に合わせて変化するデジタルな彫刻作品です。Philip Galanter氏によるアート作品です。個別に即興的に音を生成し、光のアニメーションを表現しています。形状は変化しませんが、テーマやムードで新しいタイプのデジタル表現がなされています。ハードウェアはArduinoをベースとしており、DMXと呼ばれるステージ照明用のプロトコルで、各辺の16個のLEDライトを遺伝的アルゴリズムにより色表現しています。



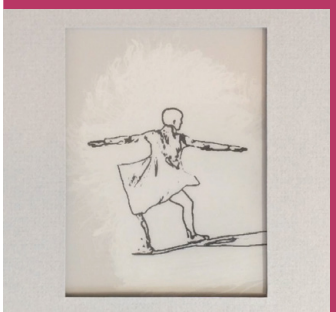
GADGET 4

Lineographs

http://farbrook.net/lineographs/

手書き風ディスプレイ

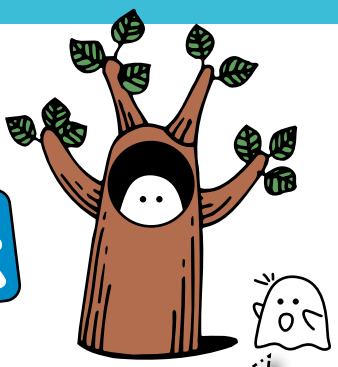
Lineographsは、Joseph Farbrook氏の作品で、Kindleのような電子ペーパーディスプレイで紙のような印象で表示され、かつリフレッシュレートが高くなめらかにアニメーションします。数百フレームの手書き静止画が連続表示され、額縁で飾られたそれは、アニメーションを見ているような、手書きの絵画を見ているような不思議な感覚の表示装置でした。



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

18

結城浩の再発見の発想法



Trigger

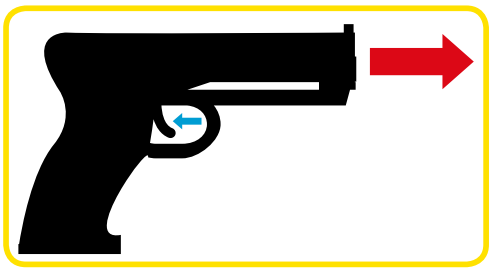
Trigger——トリガー

トリガーとは

トリガー(Trigger)とは、「大きな出来事を新たに引き起こすきっかけ」という意味です。トリガーは、もともと「拳銃の引き金」のこと。「指先で拳銃の引き金を引く」という小さな事象が、「弾丸を発射する」という大きな事象を引き起こすことから転じて、前述の意味が生まれたのでしょう(**図1**)。動詞としては「トリガーを掛ける」あるいは「トリガーが掛かる」と使います。場合によっては「トリガーを引く」ということもあります。

コンピュータの世界で「トリガー」という表現は、ハードウェアに対してもソフトウェアに対しても使われます。ハードウェアでは外部から与えられる信号全般をトリガーと表現しますし、ソフトウェアではシステムに送られてくるリクエストや発生したイベント(およびその要因)をトリガーと呼ぶことができます。

▼図1　拳銃の引き金(トリガー)

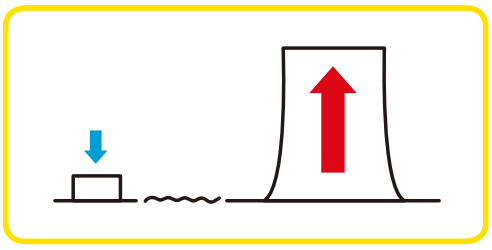


具体的にはユーザがボタンを押したり、特定の時刻になったり、温度が一定以上になったり、ネットワークが切れたり……そのような出来事はすべてトリガーと言えるでしょう。

トリガーはスイッチやフラグと似た意味を持ちます。スイッチは、オン・オフを行うものを総称したというニュアンスが強く、フラグは、二値的な状態の変化と記憶を行うものというニュアンスが強くなります。それらに対してトリガーは、スイッチが入ったことで何かが引き起こされるという部分が強調されています。

トリガーは、それによって引き起こされる事象の大きさに比べたら小さなものです(**図2**)。これは拳銃の引き金と同じですね。ソフトウェアの場合、たとえばボタンを押すというトリガーが掛かったときに何が起きるかは前もってプログラミングしておく必要があります。何段階にも渡る複雑な処理を前もって仕込んでおき、トリガーを一発掛けるだけで、その複雑な処理を実行させることになります。トリガーが掛かるまで、その複雑な処理は実行されずに静かに眠っているとも言えるでしょう。

▼図2　トリガーは小さくていい



日常生活とトリガー

日常生活で拳銃を撃つ機会はありませんが、自分に対してトリガーを掛けることはよくあります。

スマートフォンなどでアラームを掛けておくのは「自分にトリガーを掛ける」典型的な行動です。たとえば「目覚まし時計を鳴らす」というのは「朝、自分を起こす」という事象を引き起こすためのトリガーですね。

筆者のiPhoneは1日に10回以上、指定時刻にアラームが鳴ります。そのほとんどが「自分にトリガーを掛ける」ためのものです。予定表を確認する、実家の母にメールを送る、買い物がないかどうか妻に確認する……そのような「忘れがちな行動を思い出す」ためにiPhoneのアラームを使っています。月末には会計のためのトリガーが掛かり、〆切前には原稿を書けというトリガーが掛かります。

自分に掛かるトリガーは、指定時刻によるアラームだけではありません。iPhoneには特定の場所に来たときに通知してくれる機能がありますから、行きつけの書店の近くに来たときに「この本をチェックする」というトリガーを掛けることができます。

iPhoneが自分の代わりに仕事をしてくれれば便利ですが、そこまでいかなくても、自分にトリガーを掛けてくれるだけでも十分に便利です。特定の時刻、特定の場所、特定の出来事に合わせて「あの件はどうなってる？」と引き金を引いてくれるだけで、「ああそうだった」と用事を思い出すことができるからです。良いタイミングで掛かってくれさえすれば、引き金自体はちょっとしたことでかまいません。

「自分にトリガーを掛ける」という発想は重要です。「今度こういう事態になったら、こういうことをしよう」と懸命に自分の心に刻むのではなく、「今度こういう事態になったときのために、トリガーが掛かるようにしておこう」という発想ですね。

「トリガーが掛かる」ように仕込んでおけば、記憶しておく負担から解放されます。また、せっかく書店に立ち寄ったのに、この本をチェックするの、忘れてた！といった失敗を繰り返す心配がありません。特定の事象が起きたときの自分の行動を前もって仕込んでおくことで、いざというときにばたばたしないという効果もあるでしょう。

iPhoneなどのスマートフォンは高機能ですから、多彩な条件でトリガーを掛けることができます。IFTTT[注1]などのサービスを組み合わせると、さらに複雑な条件でトリガーを掛けることができます。たとえば「月曜日から金曜日のあいだ、天気予報が雨だったら、朝7:00に『傘を忘れるな』と通知する」などというトリガーを掛けることができます。

「雨が降りそうだったら、傘を持っていく」のはあたりまえですが、その前提となる「天気のことを気に掛ける」部分が不要になるということですね。

教育とトリガー

人間関係、とくに教育とトリガーは深い関係にあります。人に何かを教えるとき、毎回すべてを伝えるのが最適とは限りません。相手がすでに知識を持っている場合、くどくど伝えるのではなく「ほら、あれ、忘れているよ」あるいは「ほら、いまだよ」と一言いうだけで済む場合もあります。これは、教える人がトリガーを掛けていることになりますね。十分な仕込みが済んでおり、タイミングが効果的ならば、シンプルに一言掛けるだけで大丈夫なのです。

◆ ◆ ◆

あなたの周りを見回して、生活の中に使われているトリガーを探してみてください。そのトリガーは効果的なタイミングで掛かっているでしょうか。また、トリガーが有効に働くように、十分な仕込みができているでしょうか。ぜひ、考えてみてください。SD

注1）Twitter、GmailなどのWebサービス間でトリガーとアクションを指定して、サービス同士を連携させられるサービス。

新連載！

耽溺せよ電子工作 おとなラズパイリレー

第1回 「工作恐怖症のためのRaspberry Pi入門（前編）」

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＋工作をします。1年を通してどんなデバイスが出来上がるのか？　まずスタートを切るのは小飼弾さん。Raspberry Pi B+と遭遇します！

Writer 小飼 弾（こがい だん）
twitter @dankogai

組み立て好きの工作恐怖症

Raspberry Piを一目見て思ったのは、こうでした。

「これなら私でもいける」

工作恐怖症の、私でも。

はじめからそうだったわけではありません。ナイフで鉛筆だって削れますし、包丁で林檎だってむけます。ハンダ付けだって、母の内職の手伝いでやっていたぐらい。オモチャを買ってもらえなかった私の唯一の玩具はずっと油粘土。上手下手はさておき、これで工作が嫌いになるほうがおかしいというものです。

しかし、大学のとき、やっちゃったんですな。教授が入手したばかりのサンプルCPUをバシっと静電気で。心の傷は、発売初日に人様のiPhoneを落として割っちゃったガラスのヒビの256倍といったところでしょうか。その体験以来、私にとってハードウェアは「楽しいもの」であると同時に「怖いもの」ともなりました。簡単に壊れ、そして二度と戻らない。

それから四半世紀。この体験は今も私を苛（さいな）んでいるようです。どうも差し込み口に差し込んで終わる以上の電子工作は、今でも二の足を踏んでしまう。文字どおり火急の必要に迫られて、半田ごて片手にアプライアンスの電源のキャパシターを交換したことはあります。しかしその体験も「バシッ」のトラウマを克服するには至らなかったようで、私のハードウェアに対する態度ときたら、

① 何も挿さなくていいにこしたことはない
② どうしても抜き差しならぬなら、まともなプラグとソケットがついたものを
③ 要ハンダ付け？　電撃食らわしたるかゴラァ！

といったヘタレっぷり。

それでも時代は私のようなヘタレに追い風が吹いているようで、要ハンダ付けだったものにはコネクタが用意され、そしてコネクタは世代ごとに抜き差ししやすいものへと進化していきました。Magsafeには拍手喝采しました。SATAには狂喜乱舞しました。今はmicroUSBが挿しにくいです。

そんな私がはじめてRaspberry Pi Type Bを触れたときの感想は、こうでした。

「もしかして：PC」

裸のPC

いつの頃からでしょうか。「マイコン」という言葉と「パソコン」という言葉の示すものが別れたのは。この2つが別れる前、「マイコン」の「マイ」は「マイカー」のマイと同様、「自家用」を意味するものでした。それがいつの間に「マイコン」は「マイクロコンピュータ」という「部品」を指し

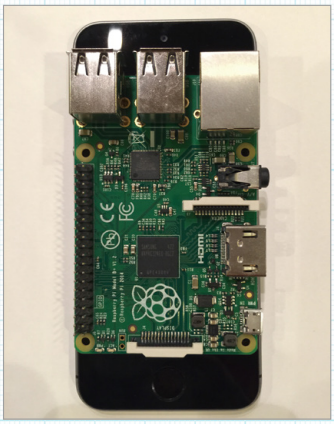
示すようになり。自家用コンピュータには「パーソナルコンピュータ」の略である「パソコン」という言葉が割り当てられ、さらにIBM PCの登場以降は英語同様「PC」と呼ばれるようになっていきました。

「マイコン」と「パソコン」、この2つの違いは、NECのTK-80（**写真1**）とPC-8001（**写真2**）に見ることができます。基板剥き出しの前者はいかにも「マイコン」という姿形をしていますし、立派な筐体を持つ後者はいかにも「パソコン」という姿形をしています。

それではRaspberry Piはどうでしょう。基板は剥き出しで、いかにも「マイコン」という姿形をしています（**写真3**）。

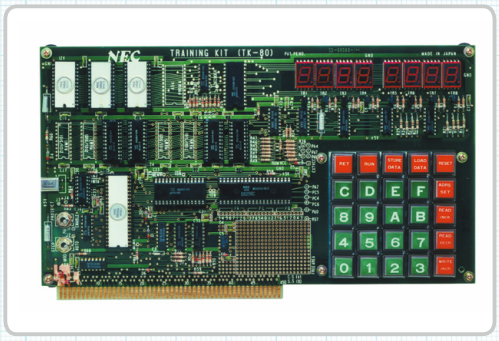
しかし、実際に動かしてみるとその印象は180度変わります。USBポートにキーボードとマウスを挿し、HDMIポートにディスプレイを挿し（TVでもかまいません。私はそうしました）。EthernetポートにLANケーブルを挿し、そしてSDカードスロットに「起動ドライブ」を挿し、そして最後にmicroUSBポートに電源を挿せば、あら不思議。普通のデスクトップPCの出来上がり（**写真4**）。最高1,080pの大画面で、Ajaxばりばりの今日日のWebもネットサーフィンできちゃいます（**図1**）。

そしてそこに至るまで、ネジ1本回す必要が

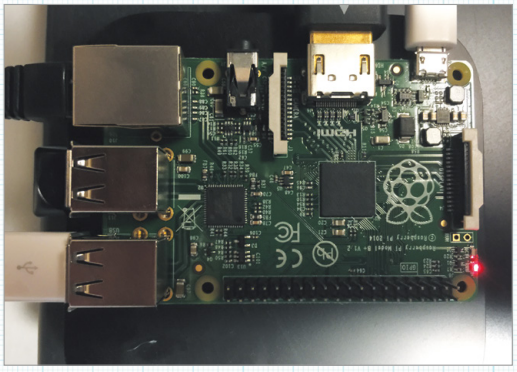


◀ **写真3** Raspberry Pi B+とiPhone 5s

▼ **写真1** NEX TK-80（写真提供　NEC）

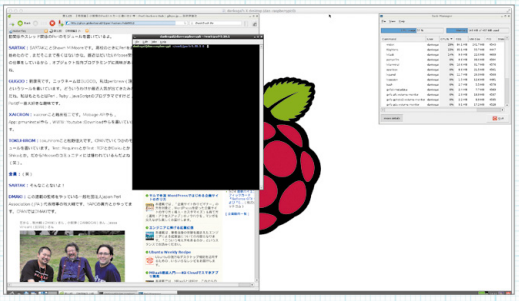


▼ **写真4** 周辺機材を接続（比較のためにApple TV）



▼ **写真2** PC-8001（写真提供　NEC）

▼ **図1** 十分ネットサーフィンできる!



ないのです。

足りないのは、筐体だけ。

実際に筐体を制作して「100%パソコン」を作っちゃう人も少なくありません。

写真5は「野生のSF作家」野尻 抱介さんのRaspberry Piですが、見事なものです。手前にあるのは、Arduino。マイコン開発の「母艦」としても使えてしまうのです。

それでは、パソコンとしての能力はいかほどのものでしょうか？

Model Aの256MBというのは21世紀のパソコンとして使うには小さすぎますが、Model B/B+は512MB。これは初期のネットブックと同じです。ただしIntelは入っていません。よってWindowsやOS Xは動きません。そのかわり、LinuxやFreeBSDをはじめ、オープンソースOSがごまんとあります。

ただしそのサポートレベルがまちまちなのもまた、オープンソースOSの常識どおり。「とりあえずLoginプロンプトが表示できます」というものから、ただちにネットサーフィンできるものまでまちまちです。迷ったらとりあえず「制式OS」であるRaspbianにしておくとよいでしょう。この場合、自分でディスクイメージをダウンロードしてSDカードに焼くという作業すら省略して、インストール済みのSDカードをふつうに購入することもできます。ますますもってパソコンです。

▼**写真5**　野尻 抱介さんのRaspberry Pi

なのに、お値段35ドル。円安にあえぐ今日日の日本でも、5,000円でおつりが来ます。スマートフォンの1ヵ月の使用料より安い。常識的な性能に、非常識な値段。これもまた、Raspberry PiをRaspberry Piたらしめている特長ではないでしょうか。

色男金と力はなかりけり

とはいえ、ふつうのパソコンだと思って使うと思わず足をすくわれるところもあります。それが最も顕著に現れるのは、ソフトウェアプロジェクトのビルド時。Perl 5.20.1を`make -j`する（コア数）のにかかった時間を**表1**にしてみました。

ふつうの1Uサーバであれば1分とかからない作業が、1時間半以上！　100倍以上の開きがあります。これでもLinux Zaurusの頃に比べればましになったんですよ。4時間以上かかりましたもの。

JK（常識的に考えて）、こういう場合はクロスコンパイルするのが定石ではあるのですが、やはり悔しいものは悔しい。フルHDのデスクトップパソコンなのに。ましてや、Raspberry Piの本来の目的は、教育。Macで開発してiPhoneやiPad（そして来年からはApple Watch）に配布するiOSの世界の真逆な立ち位置にあります。なんとかならないものでしょうか。

つぶさに見ていくと、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早速くても、ランダムアクセスが遅い。どれくらい遅いかというと、10MB/sの転送を確約しているClass 10でもランダムアクセスだと0.1Mbpsぐらいになってしまうのです。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。

▼表1　ビルド時間の比較

Vendor	CPU	Cores	OS	Build Perl
Raspberry Pi B+	Broadcom BCM2835 700MHz	1	Raspbian	1h35m20.15s
Serversman VPS	Xeon L5630 2.13GHz^	2	Ubuntu 12.04	13m53.57s
1U PC Server	Xeon E3-1270	4C/8T	FreeBSD	51.45s

普通の奴らの逆を行け？

本誌の読者であれば、今日日のトレンドが仮想化とクラウド化にあることはご存じかと思いますが、その本質は、比較的高額で物理的なハードウェアを論理的に分割して安価に提供することにあります。

Raspberry Piの特長を活かすには、逆に考える必要が出てくるでしょう。比較的安価なハードウェアを論理的に統合して、高性能なサービスを手に入れるというような。実際、Raspberry Piをクラスタにする人はすでに登場しています[注1]。

そうなってくると、“Raspberry Pi 64”への期待も高まってきます。64bit ARMはiPhone 5sを嚆矢（こうし）にすでに実用化されていますし、Ethernetポートもきちんとしたクラスタを構築するとなると10GBaseとまでは言わぬとも1Gbpsは欲しい。クラスターでなくとも、モバイルルータとして活用するともなればIEEE802.11acで無線が1Gbpsを超えた今となっては100Mbpsは時代遅れ。USBも3.0にしてしまえば速度が10倍で、起動ドライブのSSD化も視野に入ってくる……。

「教育用」を見下すのは、IT業界の悪癖の1つです。PascalやSchemeがそれでどれだけ不当に扱われてきたか心が痛みます。その一方、「教育用」がそのまま現場では使いにくかったのもまた悲しい現実で、PC自体32bit化されるまでは「おもちゃ」でした。

Raspberry Piは32bit。「現場」の64bitまではあと一歩です。

今日慣れておけば、明日普通の奴らの上を行けること請け合いです。

工作が怖い？　大丈夫。私にもできたのです。読者のあなたなら、もっとうまくできることでしょう。

Happy hacking! SD

Raspberry Piの入手方法

Raspberry Piの正規輸入販売代理店のアールエスコンポーネンツ㈱のWebにてオンライン販売がされています（**図A**）。Raspberry Piに関連した周辺デバイスや書籍も購入できるので、活用アイデアが閃いたらチェックするとよいでしょう。遠隔地にお住まいの皆さんにもお勧めです。

▶ **図A**　RSオンライン（http://jp.rs-online.com/）

注1）http://gihyo.jp/admin/clip/01/linux_dt/201402/19

軽酔対談

かまぷの部屋

第4献 ゲスト：戸倉 彩さん

㈲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

戸倉 彩さん（クラウディア・窓辺さん）

日本マイクロソフト勤務。公認キャラクタークラウディア・窓辺さんをこよなく愛し、クラウディアさんコスプレをし、Microsoft Azureのエバンジェリストとして活躍中。小さな頃からマイクロソフトで働くことが夢で、学生時代からSolarisなどをインストールしていた。実はセキュリティにも詳しい。
Twitter：@ayatokura

たくさんの「クラウディアさん」グッズ。

こんにちは。USP研究所のかまぷです。今月もほろ酔い気分で、お話を聞いてみます。

（鎌田）戸倉さんはクラウディア・窓辺さんとして有名ですが、クラウディアさんを知らない方に紹介したいので、自己紹介お願いします。

（戸倉）日本マイクロソフト株式会社でMicrosoft Azure、クラウドサービスのエバンジェリスト（製品技術の特長を多くの方に知ってもらう活動をする人のこと）です。私はおもに開発者向けの活動を軸にしています。そのクラウドサービスで、日本法人は独自のキャラクタを創っているのですが、その一番のアイコンが「クラウディア・窓辺」です。私は金髪にして、コスプレして、キャラになりきっています。彼女を通してクラウドサービスを知ってもらい、ユーザを増やすということが、会社から与えられたミッションなのですが、これがもう楽しくて、よく夢の中で生きているという表現をしています。マイクロソフトで働くことが小さなころからの憧れだったのです。クラウディアさんというキャラクタを日本マイクロソフトが創ったときに、私のなりたい女性像というのにマッチしていたのも驚きなのですが、偶然にも彼女と誕生日が一緒で、まるで運命のように感じています。

すごい偶然（必然？）ですね。社内でも「クラウディアさん」と呼ばれることが多いのですか？

そうなんです。逆に私の本名をみなさん知らないみたいで、会社やイベント会場に「クラウディアさんへ」と手紙が届くこともよくあります（笑）。以前に、「クラウディアさん、これもらってください！」と、大きい箱をプレゼントされたことがあるのですが、開けてみたらマザーボードでした。もちろんそのマザーボードは、ありがたく使わせてもらいました。それから、指輪のケースに入ったKOKUYOのプレゼンテーションポインタをいただいたこともあります。プレゼントはうれしいですし、そもそものファンとして慕ってくれる気持ちが一番うれしいです。たまに、クラウディアさんのキャラと「ピアスの形が違う」などといった、深いツッコミもいただくこともあります。みなさんよくみていますよね。

普段からコスプレはされているようですが、ほかに趣味はありますか？

サーバとか、もろもろです。本誌も愛読書で、学生時代には読んだ特集を手本にLibrettoにSolarisをインストールして遊んでいました[注1]。それと学生時代といえば、SPARCが個人ユーザ向けに販売されていたのですが、そのために貯金をすべておろして秋葉原へ買いに行ったことがあります。母には「電気代がかかり過ぎだ」としかられました。ほんと、とんでもない娘ですよね（笑）。父は航空無線のエンジニアで、プログラミングの学習はアセンブラを勧めてくれたのですが、私がみなさんにお勧めしたい言語はC#ですね。ちなみに私自身が今勉強している言語はPythonやF#などオープンソース系です。これは、エバンジェリスト

注1）1998年11月号に「Solarisといっしょ」という記事が掲載されていました。




として取り組んでいることに関連していまして、教育機関での臨時講師のお仕事をやらせてもらっているのですが、これに活かせればと思っています。Pythonはデータサイエンス学科などでビッグデータに使われている言語ですから。

いやぁ～すごいですね。貯金おろして数十万のサーバを買う女子学生はほとんどいないですよね。まわりの友達の反応はいかがでしたか？

学生のとき、いつでもサーバに触っていたかったのでWebサーバを持ち歩いていたのを自慢しましたが、まわりはドン引きですよね。「意味がわからない」と(笑)。当時はAzureのようなクラウドサービスがなかったですからね。

クラウディアさんのキャラクタに関する裏話とかはありますか？

MMDとかご存じですか？Miku Miku Danceの略なんですが、初音ミクの文化と一緒で2次利用ができるのです。さまざまなツールを使って、たくさんの方にクラウディアのMMDモデルデータを使って作品を創ってもらうんです。海外の方も参加しているので、おもしろいですよ。

コスプレやアニメ好きになった理由などは？

初音ミクの影響です。この文化が日本にあったことが大きいです。前職では海外勤務もしていて、米国でそれなりのキャリアを築けたのですが、初音ミクが理由で現在は日本にいます。初音ミクは2次創作、VOCALOIDで有名ですが、いろんな人がITを駆使して、すばらしい作品を作りますよね。キャラももちろん、その世界観が大好きなんです。

アニメ好きにどうしてなったかって、生まれつきですよね。もう物心ついたときからという感じでしょうか。

アニメや2次創作の世界ではなく、IT技術者としてすごいと思われる方はいますか？

技術力を持っている方はみんな尊敬します。弊社の萩原正義さんは、OSのコアな部分まで知っていて、プレゼンもとてもわかりやすいアカデミックな方です。たくさんエンジニアがいますが、長年の経験と豊富な知識でトップクラスのエンジニアとして位置づけされている人だと思います。彼のセミナーはすぐ満席になります。

マイクロソフト製品のパッチやリリースされる頻度について、戸倉さんはどんなふうにみていますか？

個人的に、パッチの頻度については問題意識をものすごく持っています。たとえば電化製品を買ってすぐ使えなかったら、怒りますよね。パッチを当てたことで動作しなくなるソフトが出てきたり、人間側が運用を変えないといけないことがあります。CEOが変わって会社の変化を感じています。サトヤ・ナデラ氏はインド出身で、技術者でもあります。スピードやリーダーシップなどの力に期待しています。

オープンソースからみたAzureについてはいかがですか？

意外に思う方もいらっしゃると思うのですが、Microsoft Azureの上にはLinux OSも載せられます。LinuxやBSDのOS、オープンソースをAzureの上でどんどん使ってもらいたいなって思っているんですよ。

最後に、これだけはお聞きしたかったのですが……彼氏はいらっしゃるんですか？(照)

募集中です。彼氏は、私とクラウディアさん両方が好きな人がいいです。あと、コミケや世界コスプレサミットに参加することに理解がある人。私は2.5次元にいる感覚なので、状況しだいで話が合わせられます。真剣にどこかにいい出会いがないか、探しているんですよ(笑)。マシンに「motokare」「imakare」って名前つけて可愛がってますけどね。

おきれいですから、その気になればすぐ恋人作れるんじゃないかなって思いますよ。今日は楽しいお話しありがとうございました。 SD




はんだづけカフェなう

mbedと無線LAN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/

はじめに

9月10日に、村田製作所から、「IoT（Internet of Things：モノのインターネット）機器への無線LAN組み込みを容易にする無線LAN Smartモジュール（Type-YD）と専用ライブラリ（mbed向けSNIC Interface）を提供開始」というプレスリリースが発表されました[注1]。従来、このType-YDという無線LANモジュールは、年間50,000個を購入する顧客向けに提供されていた製品です。昨今、Makerムーブメントを始めとして、プロトタイピング市場が活発になってきています。村田製作所もこのような市場の動向をふまえて、こういった製品の提供を開始するのでしょう。

BCM43362

Type-YDは、Broadcom Corporation[注2]のBCM43362という無線LANチップと、STマイクロエレクトロニクス[注3]のSTM32F205というARM Cortex-M3プロセッサを搭載したモジュールです（写真1）。

パソコンのEthernetインターフェースでお馴染みのBroadcomですが、ドライバソフトウェアを入手するのに苦労した読者の方も多くいらっしゃるでしょう。Broadcomのパソコン用ドライバソフトウェアはパソコンメーカー経由で提供されており、パソコンメーカーのWebサイトからでなければダウンロードできないのが一般的です。従来、Broadcomも、大量に生産する事業者向けにのみ製品を提供する半導体メーカーでした。たとえば、Raspberry Piに搭載されているCPUであるBCM2835の仕様書を一般のユーザが入手することはとても困難です。

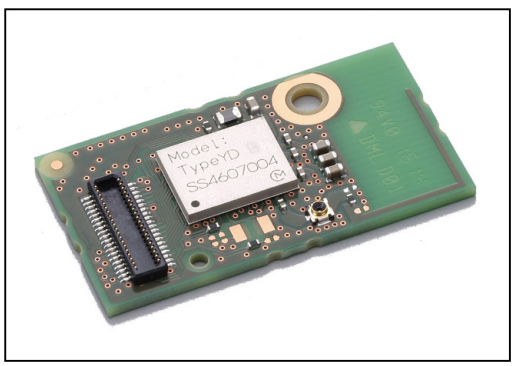
このようなビジネスモデルであったBroadcomですが、最近になってWICEDというブラ

注1）http://www.murata.co.jp/new/news_release/2014/0910/

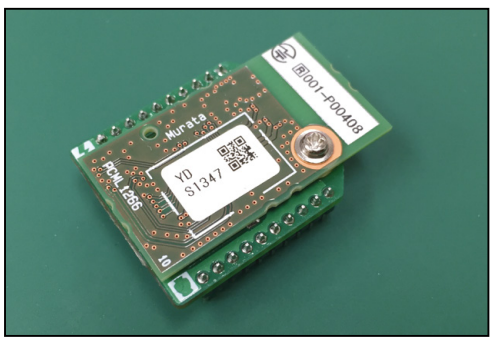
注2）ブロードバンド通信向けのチップで有名な、アメリカ カリフォルニア州の半導体メーカー。

注3）スイスのジュネーブに本社がある、世界トップクラスの半導体メーカー。

▼写真1　Type-YDモジュール



▼写真2　Type-YDモジュールの背面（Muratの刻印が見える）



ンドでWi-FiやBluetooth Smartのチップをリリースし、データーシートの提供を始めています。Type-YDに搭載されているBCM43362のデータシート（仕様書）も、BroadcomのWebサイト[注4]でユーザ登録すればダウンロードできます。

Type-YDモジュール

先述のとおり、Type-YDモジュールには無線LANチップと一緒に、マイコンが搭載されています。このマイコンにはWPAやWPA2などのサプリカント（認証クライアント）やTCP/IPプロトコルスタックが搭載されており、mbedなどのマイコンを手軽にWi-Fiに接続することができます。Type-YDには、“Murata Serial To Wi-Fi”というソリューションが提供されており、UART（非同期シリアル）でコマンドを送ることで、この無線LANモジュールを使うことができます（**写真2**）。

このType-YDを、mbedからSerial to Wi-Fiを使ってコントロールするライブラリがSNICInterfaceです。このSNICInterfaceライブラリを使用して、センサで得た値をXively（ザイブリー）というクラウドサービスに無線LANで送信するデモプログラムも公開されています。どちらも、Type-YDモジュールをmbedに接続しやすくする、スイッチサイエンスの「ムラタ 無線LANモジュール Type YD ブレークアウト」からリンク[注5]が張られています。

Xively

Xively[注6]は、IoTのためのクラウドサービスです。デバイス（モノ）からREST APIでデータを書き込み、それを蓄積できます。蓄積したデータはダウンロードしたり、グラフにして表示することができます。Xivelyは商用サービスですが、開発者向けに無償のアカウントを提供しています。ただし、記事執筆時点では順番待ちになっているようで、申し込みをしてからアカウントが作れるまでに時間がかかってしまうようです。

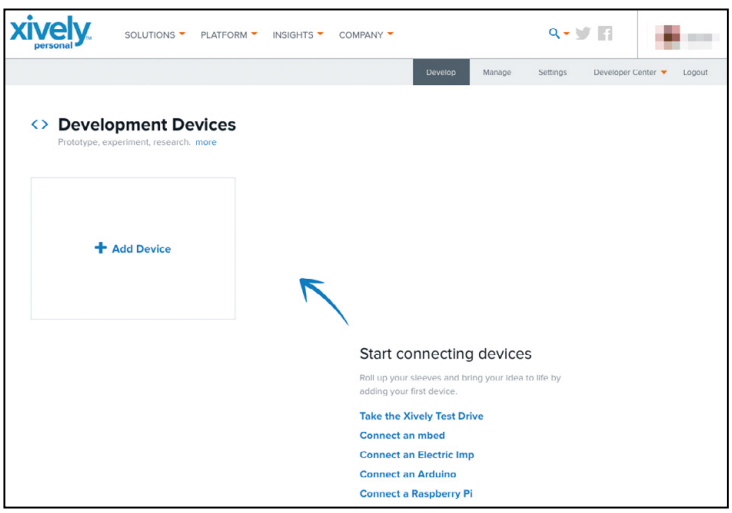
Xivelyのアカウントを作り、ログインするとコントロールパネルが表示されます（**図1**）。このDevelopタブでデバイスを登録する（**図2**）と、Feed IDとdevice keyが発行されます（図

注4） http://community.broadcom.com

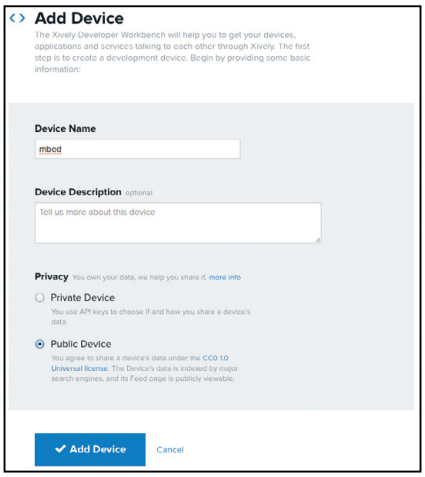
注5） http://ssci.to/1919

注6） http://xively.com/

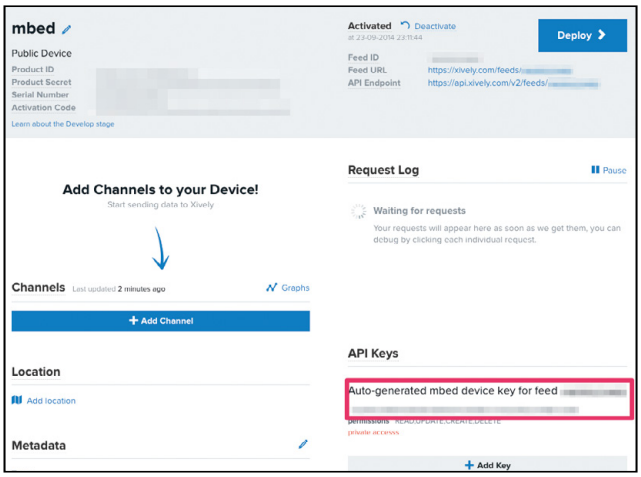
▼図1 Xivelyへログイン直後の画面



▼図2　デバイスを登録



▼図3　IDとkeyを取得

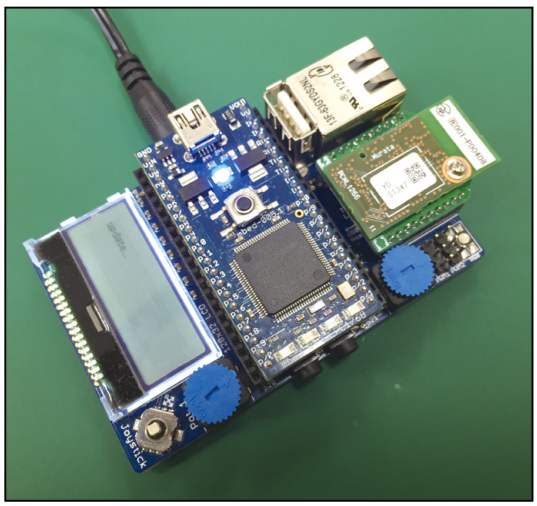


3)。このFeed IDとdevice keyを使い、デバイスはXivelyにデータをアップロードします。

試してみよう

今回は、このデモプログラムを使ってみることにします。mbedアプリケーションボードに搭載されているセンサの値を、Xivelyに送信してグラフにしてみましょう。先述の「ムラタ 無線LANモジュール Type YD ブレークアウト」はmbedアプリケーションボード上のソケットに挿して使います。また、mbedアプリケーションボードには、温度センサと加速度センサが搭載されていますので、そのセンサの値を読み取ります（写真3）。

▼写真3　デモを動かしているところ



mbed.orgのデモプログラムのページにアクセスし、“Import this program”をクリックすると、自分のオンラインコンパイラにデモプログラムをインポートできます。インポートを終えたら、オンラインコンパイラの画面右上にあるターゲットデバイスが“mbed LPC1768”になっていることを確認しましょう。

次に、main.cppを開き、先ほどのXivelyのFeed IDとAPI Keyを記入します。このとき、手元のWi-FiのSSIDと暗号方式、キーも一緒に設定するのを忘れないようにしましょう。なお、キーの長さも設定する必要があります。書き換えを終えたら、オンラインコンパイラの“Compile”ボタンをクリックして、自分用のバイナリをビルドします。コンパイルを終えると、ブラウザが自動的にバイナリファイルのダウンロードを始めます。バイナリのダウンロードを終えたら、mbedを接続したときに表示されるドライブにドラッグ＆ドロップして書き込みましょう。たったこれだけの手順で、IoTを始めることができます。

デモが動き出したら、Xivelyにブラウザでアクセスをしてみましょう（図4）。先ほどFeed IDとdevice keyを取得した画面にmbedから送

信したデータが追加されています。数秒おきに温度やアプリケーションボードの向きといった情報を送信できていることが確認できます。

しくみ

SNIC-xively-jumpstart-demoの構造を見てみましょう。XivelyのREST APIにアクセスする部分は、Xivelyの提供するlibxivelyが使われています。このライブラリは、mbed LPC1768のEthernetインターフェースでXivelyにデータをアップロードするデモプログラムでも使われているものです。

このデモプログラムでは、SNICInterfaceライブラリを使っています。mSNICwifiというインスタンスを作って、Type-YDモジュールの初期化や設定を行い、Wi-Fiネットワークに接続、IPアドレスをDHCPで取得してネットワークインターフェースを起こしています。REST APIはもちろん、HTTPの処理もlibxivelyに書かれています。libxivelyは、SNICInterfaceのTCPSocketConnectionクラスを用いてTCPでの通信を行っています。実際のTCP/IPスタックは、Type-YDモジュールに搭載されているSTM32F205というマイコンに搭載されています。SNICInterfaceライブラリは、このマイコンにUARTでコマンドを送り、モジュールにTCP/IPを用いた通信を代行させています。ですので、Type-YDモジュールに接続するマイコンにはTCP/IPスタックを搭載する必要がありません。ただし、記事執筆時点でのSNICInterfaceライブラリは、通信バッファを大きめに取っているため、32kBのRAMを搭載しているmbed LPC1768や、FRDM-KL46Zでのみ動作が確認されています。

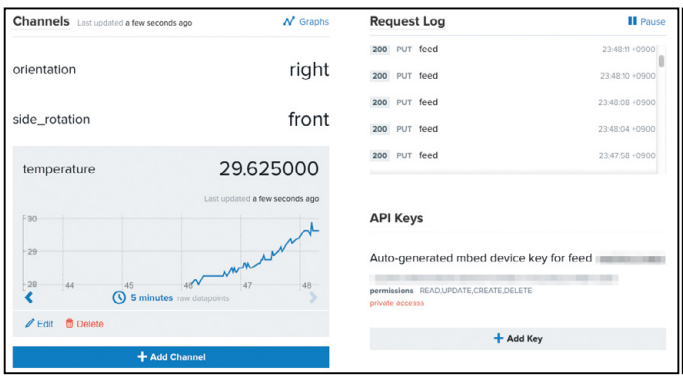
最後に

Wi-Fiは消費電力が大きいのが難点ですが、電源さえあれば、パソコンやスマートフォン、あるいはサーバと高速かつ簡単に無線で通信できます。手軽にマイコンをTCP/IPネットワークに接続できると、作れるモノの幅が拡がります。

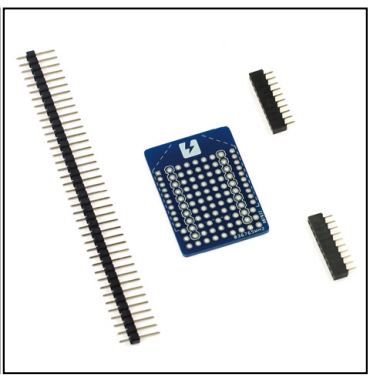
mbedアプリケーションボードは、グラフィック液晶やセンサが搭載されているため、少々値が張るのが難点です。Type-YDモジュールをmbedで使うには、アプリケーションボードを使わずに変換基板を使うという手もあります。たとえば、筆者は当初、「XBee用バニラカード[注7]」を使って、ブレッドボードに搭載してテストをしていました(**写真4**)。アプリケーションボードに搭載されているセンサを使うことはできませんが、自分で用意したセンサを使ったり、無線LAN接続をいろいろ試してみるという用途には十分です。SD

注7) http://ssci.to/1570/

▼図4 Xivelyでデータを表示



▼写真4 XBee用バニラカード用のキット



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 11 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 写真管理ツール「ジュエリーボックス」

1名

デジカメやスマホの写真・動画を、SD カード／USB 2.0 ／ネットワーク経由で本体／クラウドに保存できる、16GB の写真管理ツールです。保存したデータはテレビで見たり（HDMI 経由）、スマホやタブレット端末で閲覧したりできます。対応 OS は Windows、Android、iOS。なお、2 年目以降は、外出先からのデータ閲覧、クラウドへの保存といった機能には別途有償の「おもいでバックアップサービス」の契約が必要になります。

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

02 USB 3.0 Flash メモリ「Extream」16GB モデル

2名

最大で毎秒 245MB の読み出し、毎秒 50MB の書き込み速度を実現できる、キャップ収納型の USB メモリ。同梱の SanDisk Secure Access Software を使えば、メモリ上にパスワードで保護されたプライベートフォルダを作成することもできます。

提供元 サンディスク **URL** http://www.sandisk.co.jp

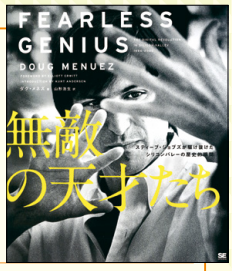
03 揉まれる肩・首スッキリピロー

1名

上に寝転ぶだけで凝り固まった肩と首周りを揉みほぐし、歪んだ姿勢をストレッチすることができる快適枕です。美容整体師の波多野賢也氏の施術理論を基に 14 個の突起が配置され、整体師の指圧を再現するよう設計されています。1 日 5 分の使用が目安です。

提供元 ドリーム **URL** http://mydream.co.jp

04 無敵の天才たち



Doug Menuez 著／山形 浩生 訳／
27.2cm × 23.6cm、200 ページ／
ISBN = 978-4-7981-3857-2

2名

1985～2000年のシリコンバレーでのデジタル革命を記録したフォトドキュメンタリ。スティーブ・ジョブズたちシリコンバレーの巨人たちを、黎明期から追い続けてきた写真家による 1 冊です。

提供元 翔泳社 **URL** http://www.shoeisha.co.jp

05 Amazon Web Services 入門

加藤 章 著／
A5 判、184 ページ／
ISBN = 978-4-8443-3647-1

2名

企業内で利用するエンタープライズシステムにおいて Amazon Web Services を利用する際に、理解しておきたい点について解説した本。開発面だけでなく、ビジネス面の情報も豊富です。

提供元 インプレス **URL** http://www.impressjapan.jp

06 アカマイ 知られざるインターネットの巨人

小川 晃通 著／
B6 判、225 ページ／
ISBN = 978-4-04-080017-2

2名

世界最大手のコンテンツデリバリネットワーク事業者「アカマイ・テクノロジーズ」の全容が、歴史・技術の面から明らかになる 1 冊。理解の助けになるインターネットの基礎知識も紹介しています。

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

07 内部構造から学ぶ PostgreSQL 設計・運用計画の鉄則

勝俣 智成、佐伯 昌樹、原田 登志 著／
A5 判、288 ページ／
ISBN = 978-4-7741-6709-1

2名

オープンソースのデータベース PostgreSQL について、その内部構造を参照しながら、基本、設計／計画、運用、チューニングの 4 章立てで解説した、中・上級者向けの解説書です。

提供元 技術評論社 **URL** http://gihyo.jp

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

安定のLAMPを変化に強くする!

いわゆる「LAMP」サーバ環境は当たり前のように安定して動作し、世界中のWebサービスの基盤でありつづけています。しかし、技術や市場の変化が激しい現在のスピードに合わせるかたちで、これまでのメンテナンスの考え方とは異なる「Infrastructure as Code」「Immutable Infrastructure」といった運用管理手法が定着しはじめています。

LAMPを題材にした本特集で仮想化技術の基本に触れ、変化に対応できる運用管理を行うきっかけにしてください。

CONTENTS

第1章 システム構築に与えた深い影響 なぜLAMPだったのか 過去から探る今の技術

Linux、Apache HTTP Server、MySQL、PHP/Perl/Pythonの頭文字で表される「LAMP」は、現在のシステム開発の主流です。ごく当たり前なシステム構成です。マーケット的な意味あいもありますが、1つの開発スタイルを代表する言葉になっています。あたかも空気のように使われるLAMPをそもそも論で振り返り、今のLAMPの流れを紹介します。

Writer 三島 匡史(みしま ただし)

2006年を振り返る

初代iPhoneが発売されたのが2007年でした。当時は、まだ今でいうスマホが存在せず、ガラケー(ガラパゴス携帯)が全盛期の時代でした。IT業界では言うまでもなくケータイアプリケーションの開発案件が大盛況でした。限られた市場でユーザの獲得と拡大のため、日々新しいアプリケーション、コンテンツの追加、メンテナンスが必要とされ、ケータイアプリケーションの開発では、より高速な開発が求められていました。

このような開発のスピード感とニーズに合わせ、従来よくあったコンパイル型の開発言語よりも実行確認が即時にできるスクリプト型言語での開発が主体となり、LAMP環境が標準的に使われるようになったのでした。

一方、以前は基幹システムのような信頼性が求められるものには、LAMPはまだまだ使えない、という声をよく聞きました。過去の資産を活用したり、開発するエンジニアの得手、不得手も影響しますが、最近は企業における基幹の業務システムであってもユーザインターフェースはブラウザというのが一般化しています。そうした流れの中でLAMP環境が使われていたりします。

LAMP環境もこの8年でだいぶ様変わりしました。実は筆者は8年前にLAMPの解説記事を本誌で執筆したのですが、今回は現時点のLAMPのあり方を考える前段として、当時を振り返りつつ、現在のLAMP環境を踏まえたLAMPの概要を解説していきます。

LAMPはオワコン?

過去に記事を執筆したときは「開発はLAMPで」という話をたびたび耳にしましたが、最近はそうした機会は激減しました。「オワコンか?(終わったコンテンツ)」と喧伝されますが、実はそんなことはありません。話題性について、Googleトレンドで LAMPのトレンドを見てみると、実際2010年ごろを境に検索数は減っています。一方でDrupalやWordPressなどのCMSで構成した、LAMP環境で動作するサイトは増えています[注1]。ゆえにLAMP環境自体もCMSの増加に合わせて増えています。LAMP環境としては、技術的な大きな変化は減りました。つまりシステム環境としては安定し、問題が起きにくくなったため、それ自体の話題は減ってきているものの、標準的に使われる環境になったといえます。

LAMPの構成の振り返り

LAMPはLinux、Apache HTTP Server(以下、Apacheと省略します)、MySQLとプログラミング言語のP(PHP、Perl、Python)の頭文字を組み合わせた言葉です。ご存じの方が多いとは思いますが、ここであらためてそれぞれについて説明しておきます。

注1) http://w3techs.com/

Linux

LinuxはLAMP環境の基礎となるOSである一方、今や多くの携帯はLinuxをベースにしたスマホとなり、Linuxを採用するテレビもたくさんあり、組込みOSとしても身近にさまざまなところで利用されています。

Linuxには、多くのディストリビューションがあり、以前はサーバ用途におけるシェアを争っていましたが、現在の日本国内ではRed Hat Enterprise Linux(RHEL)/CentOSがその大半を占めています。一方、海外に目を向けますと、Ubuntuがシェアを大きく伸ばしており、RHELのシェアを超えたというデータもあり、今後の注目の的になっています[注2]。

LAMP環境は物理サーバにLinuxを入れて環境構築するケースは減少し、最近は仮想環境上に導入することが当たり前になりました。また、LAMPでの開発環境は、個人のPCにVirtual Boxなどで仮想環境を構築し、その中でいくつもの仮想マシンを用意して、開発するというのが一般的になっています。仮想化でのLAMP環境について詳しくは第2章を参照ください。一方、仮想化するとコンピュータをエミュレートするための処理能力によるオーバーヘッドが発生し、物理サーバにインストールした際に比べ、パフォーマンスが低下します。そこで、最近は仮想サーバと物理サーバのイイトコどりの環境としてコンテナが注目を浴びています。Linuxのコンテナ技術として人気があるDockerについては第5章で紹介します。

Apache(Webサーバ)

ApacheはWebサーバとして、米国立スーパーコンピュータ応用研究所(NCSA)が開発したNCSA httpdをベースに1995年に「Apache Server」としてリリースされました。その後Webの利用者が増えるとともにその開発組織が大きくなり、1999年にApacheの開発組織としてApache Software Foundation(ASF)が設立されました。現在のApacheはASFで開発される1つのプロジェクトとなっており、正式名称は「Apache HTTP Server(httpd)」です。

Apacheは機能のすべてをモジュールとして構成しており、ベースとなるCoreモジュールにいろいろな機能を持ったモジュールを組み合わせてWebサーバを構成します。

これまで「Webサーバといえば Apache」というほど高いシェアを確保していましたが、近年では状況が変わりました。Webサーバのシェアについて、継続的に調査を行っているNetcraft社の調査によると、2012年半ばからMicrosoft社のIISが急速にシェアを伸ばし、Apacheに追いついて来ています。また本誌2014年7月号でも特集したNginxが注目をあびるとともにシェアを伸ばしています。

MySQL

MySQLはオープンソースソフトウェア(OSS)のデータベース(OSS DB)としてMySQL AB社によって開発されました。その軽快な動作から人気を博し、現在は多くのソフトウェアの標準的なデータベースとして利用されています。MySQL AB社は2008年にSun Microsystems社に買収され、そして2010年にOracle社に買収されました。現在はOracle社により開発が継続されています。Oracle社による買収後、MySQLのオリジナルの作者がMySQLを基に新たにMariaDBというデータベースを開発しています。最近はLinuxのディストリビューションに含まれるデータベースのパッケージがMySQLからMariaDBに置き換わってきており、LAMPのMはMariaDBというケースが増えていくでしょう。

国内でMySQLと人気を二分するOSS DBとしてPostgreSQLがあります。OSS DBにおいて、海外ではMySQLのシェアがかなり大

注2) http://www.markshuttleworth.com/archives/1072

きいのですが、国内ではPostgreSQLも多く使われています。なお、MySQLの代わりにPostgreSQLを採用した環境をLAPPと呼びます。

PHP or Perl or Python

LAMPのPはPHP、Perl、Pythonの3つのプログラミング言語を表します。この3つのプログラミング言語も時代とともに流行の変化がありました。

- Perl……WebアプリケーションとしてCGI（Common Gateway Interface）が使われていた。その当時はCGIを開発する言語としてPerlが流行。しかしながら、Perlは難しく初心者には敷居の高い言語だった
- PHP……携帯電話用のWebアプリケーションの需要増加のためWeb開発が流行した。Web開発に特化し、文法がわかりやすく、効率的に開発が可能なPHPが主流になった。当初は大規模開発には向かないと言われていたが、開発用フレームワークが充実するとともに、大規模開発にも使われるケースも増えた。FacebookやGREEといった巨大なWebサービスの開発にも用いられている。Web系のプログラミング言語として世界中で広く使用されている
- Python……Pythonは汎用言語としてLinuxのインストーラやゲーム、データ解析など幅広く使用された。根強い人気があり、OSSの開発やWebアプリケーションの開発に使われる機会も増えている

これらは筆者のイメージですが、Perl使いは昔からの玄人が多く、Python使いはプログラミングについてこだわりを持っている現役プログラマが多く、PHPは初心者から玄人まで幅広く使われています。

LAMP環境の基本動作の復習

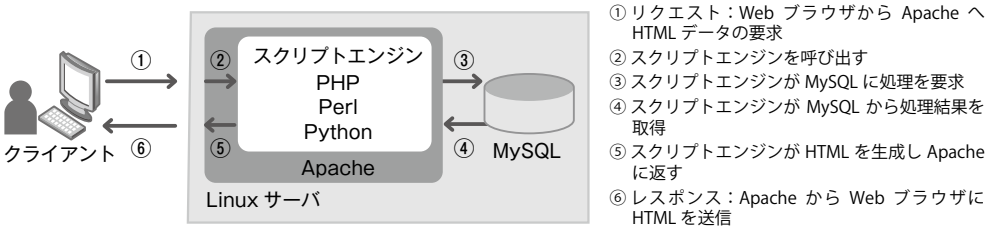
動的なWebページの生成

ブラウザからのリクエストに応じ、サーバ側でプログラムを実行し、表示内容を変えることができるWebページを動的なページと言います。LAMP環境はこの動的なページを作るために利用します。

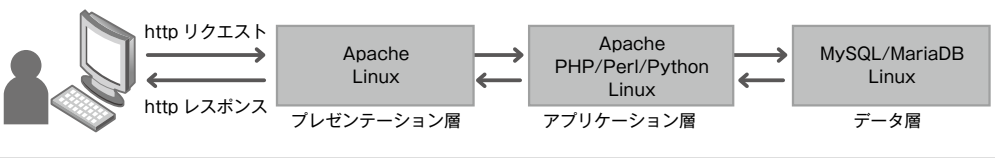
構成

LAMP環境をサーバ1台で構成すると**図1**のように動作します。負荷の軽い社内のちょっとしたサーバや、手元のPCの仮想環境上に開発環境を作る場合などにはこのような構成になります。

▼図1　LAMP動作



▼図2　Web 3層構成



基本動作

一方、本番環境において、ある程度の負荷が見込まれるLAMP環境におけるWebアプリケーションでは、Web 3層構成(**図2**)をモデルとしてシステムを構成します。基本的な動作は両方とも同じで、Web 3層構成は**図1**を各層に役割分担した形です。Web 3層構成を使って、LAMP環境の動作を示します。

プレゼンテーション層

ブラウザから送信されたhttpリクエストをフロントのApacheが受け取ります。Apacheはそのリクエスト内容を見て、静的なコンテンツ(画像、JavaScriptのソースコードなど)についてはデータを読み込み、ブラウザへ送信します。動的なコンテンツについてはアプリケーション層に処理を転送し、その応答を受けたあと、ブラウザに対し応答結果を送信します。

アプリケーション層

PHP/Perl/Pythonのスクリプトが動作し、動的なコンテンツを生成するアプリケーションサーバです。通常、スクリプトは単体ではなく、Apacheのモジュールとして動作させます。リクエストに応じてプログラムを実行します。また、必要に応じてデータ層にデータをリクエストし、そのレスポンスをプログラムにて処理します。

データ層

MySQLが動作し、データの管理を行います。アプリケーション層からの要求に対してデータベースの操作を行い、レスポンスを返します。

LAMPのインフラ

仮想化からクラウド化、IaaSとPaaS

LAMPのインフラも今や仮想化は当たり前になり、負荷の高い一部のサーバ以外は仮想化して集約度を高めるのが一般的です。そんな中、データベースサーバについては、負荷が高くなりがちなため、「データベースサーバだけは物理サーバで」という声はしばしば聞かれます。

仮想化したあとは移行がしやすくなるため、維持管理などのコストや可用性などを考慮し、既存システムをまるごとクラウド環境に移すような事例もあります。

クラウドというとハードウェアやネットワークを仮想化して提供するIaaSをイメージされる方も多いと思いますが、今後開発環境として期待されるのがPaaS(Platform as a Service)です。PaaSを使うと、リソースの追加や削除が容易に行え、OSやミドルウェアの運用管理の手間が減ります。

既存システムを仮想化してクラウドへ載せる、という流れがありますが、新規のシステムについては、PaaSを使って開発、構築を行うというケースの増加が見込まれます。LAMP環境はPaaSの中に含まれており、ユーザからは見えにくく、LAMPを意識せずに開発環境として自然に使うようになりますので、PaaSがより一般的になりますと、LAMPという言葉はさらに使われなくなるかもしれません。

パフォーマンスの検討

CPU、メモリ、ストレージなどがLAMP上で動作するアプリケーションのパフォーマンスに影響を与えます。LAMP環境で作るようなWebアプリケーションでは複数のアクセスを同時に受け付け、同時に処理するようプロセスが並列動作します。パフォーマンスについては、システムの動作を意識したリソースを検討しましょう。パフォーマンスは何か1つが良ければ良い、というわけではなく、何か1つボトルネックとなるものがあると、システム全体に影響するため、システム全体のバランスを考える必要があります。

CPU

Linux上では複数のプログラム(プロセス)が同時に動作しているように見えますが、実際は各CPUコア単位で、1つの処理の流れの中で細かく時間を区切りながら、複数のプログラムを順に処理しています。そのため、CPUのコア数を増やすことで並列処理する量を増やすことができるので、パフォーマンスの向上が見込めます。

メモリ

メモリとHDDはデータへのアクセス速度が10万倍くらい違うとされ、メモリのほうが圧倒的に高速です。最近はメモリの価格も下がり、100GB超の大容量のメモリを搭載するサーバも使われるようになってきています。その大容量のメモリを活かし、大容量のキャッシュを使ったり、データをメモリ上に置くなどして高速化を図ることができます。

ストレージ

ボトルネックになりがちなのがI/O性能です。対障害性、コスト、実効容量、性能などを考慮し構成を考えます。容量重視だとRAID 6で構成し、パフォーマンス重視だとRAID 10で構成することが多いようです。多くの商用利用においては障害時に備え、ホットスペアも必要とされます。

ストレージメディアとして使われているハードディスクドライブ(HDD)はインターフェースの違いでSATA、SAS、FCと3種類あります。最近ではパフォーマンス重視の環境についてはSSDも使われます。これらを用途、予算に応じて使い分けます。最近はFCはあまり聞かれなくなり、代わりにSSDが使われています。SATAのHDDと比較すると、SSDのほうが容量単価が10倍くらい高くなりますが、その分読み書きの速度が速いので、需要が高まっています。しかしながら、すべてSSDを使うととても高価なものになるため、重複排除機能によりデータ容量を減らしたり、データをハードディスクに貯めつつ、SSDをキャッシュに使うなど、効率的にSSDを使い、パフォーマンスを高める、ハイブリッドなストレージ製品などもあります。

オンプレミス、クラウドの選択

最近は「クラウドファースト」と言われ、自前でハードウェアを用意するか、クラウド上に構築するか、という点が検討されます。LAMPのインフラも同様に検討が必要です。クラウドにすれば安い、というイメージもありますが、実際はそうでもないケースがあります。クラウドは使い方によっては安くも高くもなります。Web上でのサービスで、アクセスの急増など、急激な負荷の増加に対しては柔軟に対応できるクラウドが向いていたりしますが、社内のシステムのようなサイジング可能なシステムについては、オンプレミスで構築したほうがコストを抑えられたりします。システムの用途、特性に応じ、適切に選択しないと、思わぬコスト増につながるため注意しましょう。

LAMPをこれから始める方へ

LAMPについてこれから詳しく学びたいという方には資格試験を活用したスキルアップとエンジニア同士のコミュニケーションによるスキルアップがオススメです。

資格試験

Linux

Linuxの技術認定試験にはいくつかありますが、中立性が高く最もメジャーな資格がLPICです。エンジニアの登竜門として、Linuxの基礎となる知識を身に付けるのに向いています。LPICはレベル1からレベル3まであり、高度な技術までカバーできるように構成されています。レベル1を習得し、実践で鍛えながらレベル2、3と資格取得を目指すことでエンジニアとしてのスキルアップが目標を持って行えます。

PHP 5技術者認定試験

一般社団法人PHP技術者認定機構が運営するプログラミングの知識力を認定する資格試験です。初級、上級、ウィザードとレベルがあります。

OSS DB

OSS DBはオープンソースデータベース(OSS DB)に関する技術力と知識を、中立的な立場で認定する試験です。本試験では現在、OSS DBとしてPostgreSQLが対象となっています。「MySQLがない！」という声が聞こえてきそうですが、LAMPエンジニアとしてデータベースの基本的な知識、スキルを身に付けるための登竜門として本試験はオススメです。

イベント、勉強会に参加しよう

最近は毎日何かしらの勉強会がそこかしこで開かれています。「IT勉強会カレンダー[注3]」を見ると日本全国で開催される勉強会を調べられますし、Connpass[注4]やAtnd[注5]などイベント管理サイトでも見つけることができます。大小さまざまなイベント、勉強会がありますが、その中でもLAMP関連で主だったものを紹介します。

オープンソースカンファレンス

たくさんのOSSコミュニティが一同に介し、ブースによる展示とセミナー形式のセッションを行います[注6]。初心者向けのセッションも多く、これから始める方にとってはとても有益な場だと思います。

PHPカンファレンス

2000年から毎年夏に開催されている、国内最大のPHPのイベント[注7]です。1日で初心者向けセッションから深い話までたくさんのセッションがあります。

カーネル読書会

横浜Linux Users Group(YLUG)が主催している勉強会[注8]です。開催回数は114回という老舗の勉強会です。第100回ではLinuxの生みの親であるリーナス・トーバルズ氏を招いています。深い話から最新動向まで話題は幅広く、また、最後にビアバッシュが行われますので、達人たちとの刺激的な交流もできます。

まとめ

LAMPはコモディティとなり、今や当たり前になりました。関連書籍もたくさんあり、ネット上の情報も充実していますので、今後も安定したシステム基盤として利用されていくでしょう。本章では、特集の最初として振り返りから今を考えてみました。SD

COLUMN

「OSSの開発組織」

LAMP環境はOSSの集合体であり、それぞれ個々に基盤となる開発組織があります。Linux FoundationとApache Software Foundation(ASF)はそれぞれLinux、Apache HTTP serverを開発していますが、そのほかにも開発プロジェクトが複数あります。Linux Foundationのプロジェクトとしては仮想化関連でOpenVirtualization Alliance やXen Project、SDN関連でOpenDaylightなどがあります。ASFは約150ものプロジェクトがあります。代表的なものとしましては、TomcatやOpenOffice.org、Hadoopなどがあり、ASFで最も開発が盛んなプロジェクトであるCloudStackや注目度の高いプロジェクトとしてSparkやMesosなどもあります。なお、ここでは省略してますが、ASFで開発されているソフトウェアの正式名称にはApache CloudStackのような形で、すべて先頭に「Apache」が入ります。

▼表1　OSSの開発組織

Linux	Linux Foundation
Apache	Apache Software Foundation
MariaDB	MariaDB Foundation
PHP	PHP Group
Python	Python Software Foundation
Perl	Perl Foundation

注3）http://goo.gl/DIJzCP
注4）http://connpass.com/
注5）https://atnd.org/
注6）http://phpcon.php.gr.jp/
注7）http://www.ospn.jp/
注8）http://www.ylug.jp/

安定のLAMPを変化に強くする!

第2章 Web開発必修項目 VirtualBox+CentOSでローカル開発環境を構築する

CentOS+Apache+MySQL+PHPのLAMP環境を仮想化ソフトウェア「VirtualBox」上で構築する手順をゼロから解説します。仮想マシンの設定から、各ソフトウェアのインストール・設定方法、ポートの開放、DBとの接続方法まで、詳しく見ていきましょう。

●アシアル㈱ URL http://www.asial.co.jp/
Writer 宇都宮 諒(うつのみや りょう) Twitter @ryo511 岡本 雄樹(おかもと ゆうき) Facebook j801com

LAMP環境の変遷

筆者らが働いているアシアル㈱は技術者を中心とした会社で、常に新しい技術を追いかけています。創業時はPHPを、そして今はHTML5によるモバイルアプリ開発に取り組んでいます。

また、受託開発やエンジニア向けの自社製品の開発を行う傍ら、ブログを通じた技術情報の発信やWeb業界で働くプロに向けたスクール事業を行っています。最近では1回3,000円で受講できる「アシアル塾」という低価格のサービスも開始しました。

スクールに参加される方にはいろいろな人がいるのですが、基本的には仕事でWebにかかわっている人です。デザイナの方やディレクターの方はもちろんのこと、広報の仕事をされている方も来られます。また、フリーランスの方や社長業を営んでいる方もよくいらっしゃいます。

最近では、仮想化が普及したこともあり、Web制作を行っている方々もLinuxを始めとしたサーバ技術を学び始めています。スクールでLinuxのコマンドライン操作を憶えたデザイナの方がそれをMacのターミナル上でも活用するといったケースもあり、積極的に学ぶ人は職種にとらわれず、活用できるものは何でも活用している状況です。

この記事では仮想化ソフトウェアであるVirtualBoxを利用してLinuxをデスクトップコンピュータにインストールし、ローカル上にLAMPの開発環境を構築する方法を紹介します。

Webの開発を行う場合に、開発環境の構築は避けて通れません。開発環境を簡単にインストールできるようなソフトウェアも存在しますが、要件にあった環境を自在に構築できるようになるためには、OSをインストールするところから学ぶ必要があります。

OSから環境構築をするときには、望みのソフトウェアがうまくインストールできなかったり、ネットワークがうまく設定できなかったりといったトラブルに見舞われる可能性もありますが、それは貴重な訓練です。本番で予期せぬトラブルに見舞われたとき、環境構築の経験がなければ何もわからない状態に陥ります。

Linuxは普遍的な技術ですので、今からでも学ぶ価値は十分にあります。仮想化ソフトウェアを使えばサーバ機を買う必要もありませんし、文献も多くて学びやすい状況にあります。それでは、一緒にLAMP環境をローカル上に構築する方法を学んでいきましょう。

事前準備

VirtualBoxのインストール

VirtualBox(**図1**)は、仮想化ソフトウェアの1つです。WindowsなどのOSの上に、Linuxなど、別のOSを動作させることができます。VirtualBoxの特徴は、オープンソースであり無料で利用できることと、Windows/Mac/Linuxといったマルチプラットフォームに対応していることです。次からのステップでは、Windows 7(64bit)を使用していますが、VirtualBoxの基本的な操

作手順はMacでもLinuxでも変わりありません。

VirtualBoxの公式サイト[注1]から自分のOSに対応するVirtualBoxのインストーラをダウンロードしてください。インストーラを起動してウィザードに従えば、とくに問題なくインストールできるかと思います。なお、本稿で使用しているVirtualBoxのバージョンは「4.3.14」です。

CentOSのダウンロード

CentOS(**図2**)は、Linuxディストリビューションの1つです。Red Hat Enterprise Linuxという商用Linuxのクローン(無償版)であり、高度なセキュリティと機能性を兼ね備えています。2014年10月現在、CentOSの最新安定版は7系ですが、7系は以前とは大きく変わっている部分があります。そのため、現時点ではまだ6系を標準としている専用サーバやVPS(Virtual Private Server)が大半です。本稿では、CentOS 6.5を使用します。

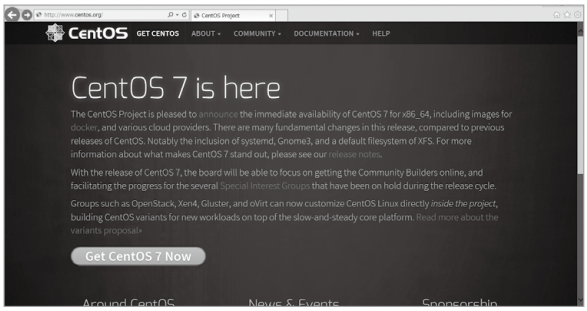
CentOSの各バージョンの一覧[注2]から、「CentOS6 X86_64」を選択し、ミラーサイトを選択して64bit版のCentOSをダウンロードします(**図3**)。CentOSのisoファイルには何種類かありますが、今回はサーバ用途なので最小構成(CentOS-6.5-x86_64-minimal.iso)を使用します。

VirtualBoxにCentOSの仮想マシンを追加する

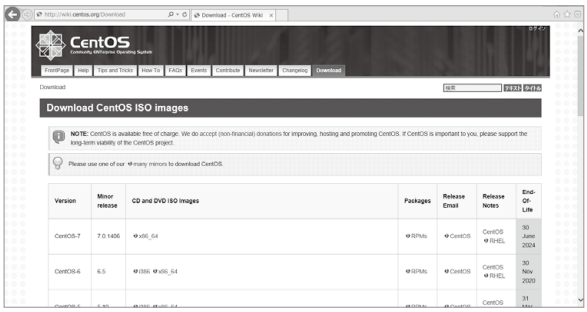
VirtualBoxを起動し、「新規」を選択して仮想マシンの作成ウィザードを立ち上げます(**図4**)。仮想マシンの名前は何でもかまいません。タイプは「Linux」、バージョンは「Red Hat(64bit)」にしておきましょう。設定はすべてデフォルトで作成していきます(**図5**)。

▼図1　VirtualBox

▼図2　CentOS



▼図3　CentOSのダウンロード



・メモリサイズ：512MB
・ハードドライブ：「仮想ハードドライブを作成する」
・ハードドライブのファイルタイプ：VDI
・ストレージ：可変サイズ
・仮想ハードドライブのサイズ：8GB

メモリや仮想ハードドライブのサイズは、ホストPCのスペックしだいでは、もう少し多めにしても良いでしょう。

注1）URL https://www.virtualbox.org/wiki/Downloads
注2）URL http://wiki.centos.org/Download

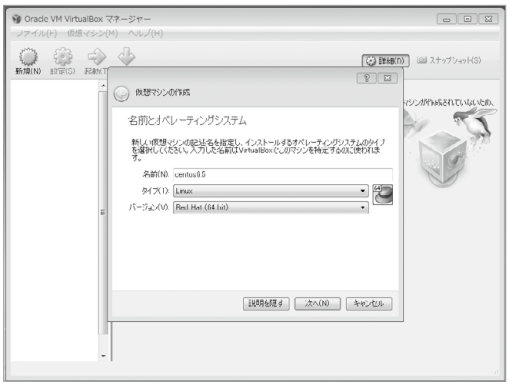
CentOSインストール

仮想マシンが作成できたら、「起動」をクリックして仮想マシンを立ち上げます。初めに、起動ハードディスクを選択するダイアログが出てくるので、先ほどダウンロードしたisoファイルをセットして「起動」をクリックします(**図6**)。次のとおりに設定をしていきます。

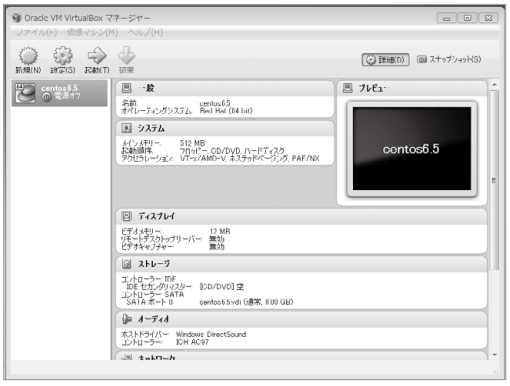
① インストーラのメニューが表示されるので、一番上の「Install or upgrade an existing system」を選択します。
② インストーラのディスクのチェックをするか確認が出てきます(図7)。「Skip」を選択します。なお、「OK」から「Skip」への切り替えにはkを使用します。
③「Welcome to CentOS!」と表示されたあと、最初に言語の選択を行います。Englishのままでもかまいませんが、「Japanese」にしておくとキーボードやタイムゾーンの設定が楽になります。
④ キーボードは、日本で一般的に使用されているJISキーボードの場合、「jp106」を選択します。
⑤ ハードディスクの初期化が必要な旨警告が出るので、「Re-initialize all」を選択します。
⑥ タイムゾーンは「Asia/Tokyo」で良いでしょう。
⑦ パスワードは強度の低いものを設定しようとすると警告が出ますが、「Use Anyway」を押せば強度の低いものでもそのまま設定できます。
⑧ パーティションについては、「Replace existing Linux system」とします。警告が出るので、「Write changes to disk」を選択します。
⑨ しばらく待つと、インストールが完了します。

「Reboot」を押して、CentOSを再起動します。

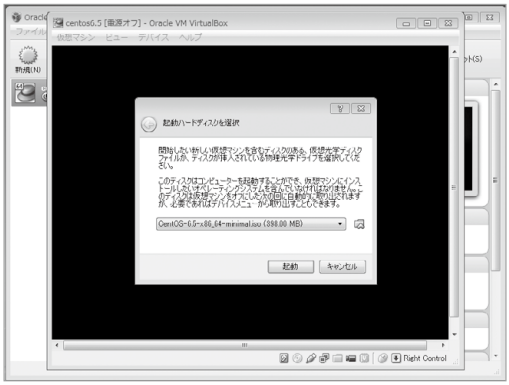
▼図4　新しいVMの追加



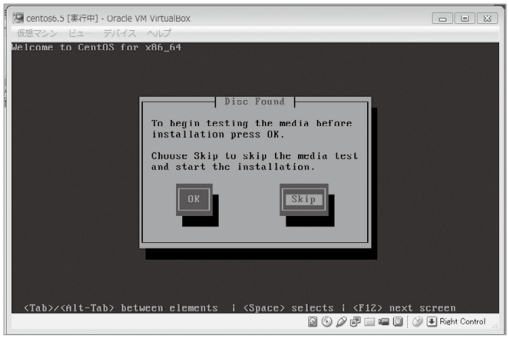
▼図5　VMの設定



▼図6　ディスク選択



▼図7　ディスクチェック



ログイン・ネットワーク設定

CentOSを再起動すると、ログイン画面が表示されます。localhost loginに「root」、Passwordに先ほど設定したパスワードを入力し、CentOSにログインします(**図8**)。

起動直後の状態では、ネットワークが有効になっていないので、その設定を行っていきます。

VirtualBoxのデフォルトで設定されているNetwork Address Translation(以下、NAT)の設定をviエディタを使っておこないます。

```
# vi /etc/sysconfig/network-scripts/ifcfg-eth0
```

ファイルを開いたら、「ONBOOT=no」となっている個所を、「ONBOOT=yes」に変更します。これで、eth0のネットワークがOS起動と同時に動くようになります。設定が終わったら、次のコマンドでネットワークの再起動を行います。

```
# service network restart
```

また、今回は必要ありませんが、OSの再起動方法も知っておきましょう。次のコマンドで行えます。

```
# shutdown -r now
```

パッケージのインストール

続いて、パッケージのインストールを行っていきます。まず、既存パッケージを最新版に更新しましょう。途中、「これらのパッケージを更新して良いですか？」と聞いてくるので、「Y」を選択します。

```
# yum update
```

インストールするパッケージについて

本稿ではLAMP環境構築を目指すため、Apache、MySQL、PHPをインストールします。CentOS 6.5のデフォルトでは、yumでインストールできるパッケージのバージョンは古めです。

たとえばPHPの場合、バージョンは5.3.3です。PHP5.3は、2014年8月でEOL(End of Life：サポート終了)を迎えており、今後公式なパッチはリリースされません。公式EOL後もOSのベンダによるサポートは続きますが、新規に構築する環境で古いPHPを使うメリットはありません。サードパーティライブラリの5.3系サポートも徐々に打ち切られているので、5.4以上の使用を強く推奨します。

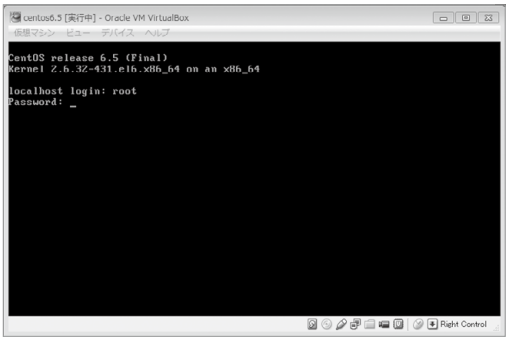
リポジトリの追加

デフォルトでは提供されていないバージョンのインストールに関しては、ビルド済みのバイナリを提供している非公式リポジトリを利用するのが手軽です。**図9**では、EPEL(Extra Packages for Enterprise Linux)と、PHPコア開発者であるRemi氏の提供しているリポジトリを追加しています。

Apache、MySQL、PHPのインストール

それでは、パッケージを実際にインストールし

▼図8　ログイン



▼図9　リポジトリの追加

```
# rpm -Uvh http://dl.fedoraproject.org/pub/epel/6/i386/epel-release-6-8.noarch.rpm
# rpm -Uvh http://rpms.famillecollet.com/enterprise/remi-release-6.rpm
```

ます。次のコマンドで「--enablerepo=remi-php56」の部分で特定のバージョンのPHPをインストールするよう指定できます。5.5なら「php55」と記述します。

```
# yum install --enablerepo=remi-php56 httpd ☑
mysql-server php php-mysql
```

インストール中に、Remiリポジトリの GPGキーをインポートするか聞かれるので、「y」を選択します。

インストールが完了したら、次のコマンドを実行して、インストールされたバージョンを確認しておきましょう。

```
# httpd -v
# mysql --version
# php -v
```

なお、筆者の手元の環境では、Apache 2.2.15、MySQL 5.1.73、PHP 5.6.0がインストールされました。

Apacheの初期設定

現在、VirtualBoxはNATで動作しています。ホストOS(Windows)から、ゲストOS(CentOS)のApacheにアクセスできるようにするため、「VirtualBoxのポートフォワーディング」、「CentOSのポート開放」の設定を行います。

VirtualBoxのポートフォワーディング

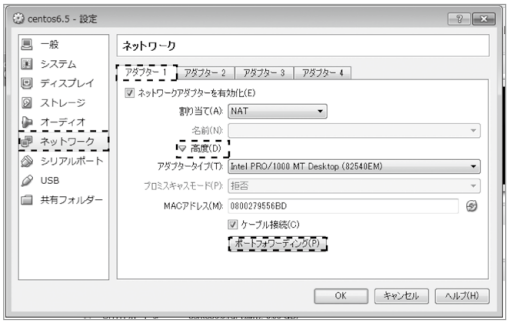
ホストOSのポートからゲストOSのポートへポートフォワーディングを行う設定をします。

まず、端末に「exit」と入力して、CentOSからログアウトします。そのあと、仮想マシンのウィンドウを閉じようとすると、3つの終了オプションが表示されます。「電源オフ」を選択して、仮想マシンを終了します。

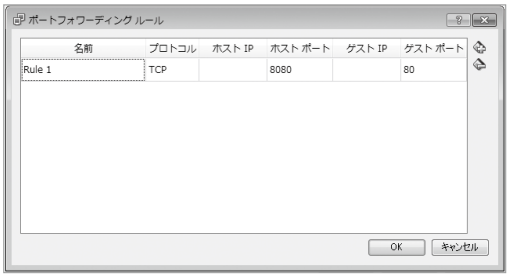
仮想マシンを終了したら、VirtualBoxマネージャーで「設定」-「ネットワーク」-「アダプター1」を選択し、「高度」をクリックして高度な設定を表示します(**図10**)。

高度な設定の中で、「ポートフォワーディング」をクリックし、ポートフォワーディングルールを追加します(**図11**)。ルールの名前は何でもかまいませんが、プロトコルは「TCP」、ホストポートは「8080(ホストOSで未使用のポートなら何

▼図10　ネットワーク設定



▼図11　ポートフォワーディング



▼図12　ポートの確認

```
# iptables -L --line-numbers
Chain INPUT (policy ACCEPT)
num  target     prot opt source               destination
1    ACCEPT     all  --  anywhere             anywhere            state RELATED,ESTABLISHED
2    ACCEPT     icmp --  anywhere             anywhere
3    ACCEPT     all  --  anywhere             anywhere
4    ACCEPT     tcp  --  anywhere             anywhere            state NEW tcp dpt:ssh
5    REJECT     all  --  anywhere             anywhere            reject-with icmp-host-prohibitedv
```

でもOK)」、ゲストポートは「80(ゲストOSのApacheがListenするポート)」を指定します。

設定ができたら、「起動」をクリックして仮想マシンを起動します。

CentOSのポート開放

CentOS 6.5(minimal)のデフォルトでは、22番ポート(ssh)だけが開放されている状態になっています。確認してみましょう(**図12**)。

それでは、Apacheが使用する80番ポートを開放してみます。

```
# iptables -I INPUT 5 -m state --state NEW -m tcp -p tcp --dport 80 -j ACCEPT
```

再度**図12**のように**iptables -L --line-numbers**を実行し、設定した内容がINPUTチェインの5番めに追加されたことを確認してください。設定ができたら、次のように設定内容を保存します。

```
# service iptables save
```

そのあと、iptablesを再起動することで、変更を反映させます。

```
# service iptables restart
```

Apacheの起動

次のコマンドでApacheの状態を確認できます。

```
# service httpd status
```

現在は「stopped」になっていると思います。次のコマンドで、Apacheを起動できます。

```
# service httpd start
```

ドメインネームに関する警告は出ますが、無事に起動できたと思います。ついでに、OS起動時にApacheが自動で起動する設定も行いましょう。

```
# chkconfig httpd on
```

これらの設定が終わったら、ゲストOS(Windows)のブラウザを起動して、「**http://localhost:8080/**」にアクセスしてみましょう。Apache 2 Test Page(**図13**)が表示されれば、Apacheが正常に動作しています。

MySQLの初期設定と起動

まず、MySQLを起動し、同時に自動起動設定も行います。

```
# service mysqld start
# chkconfig mysqld on
```

次に、MySQLの初期設定を行います。

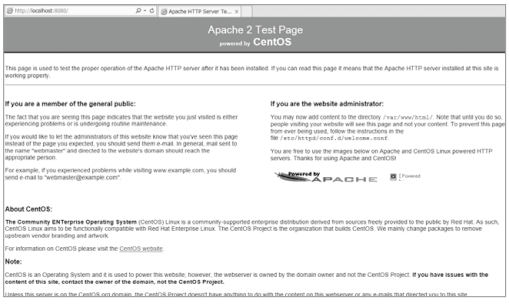
```
# mysql_secure_installation
```

次のとおりに設定をしていきます。

① MySQLのrootユーザのパスワードを設定します。
② anonymous(匿名)ユーザを削除するか聞かれるので、「Y」を選択します。
③ rootでのリモートログインを無効にするか聞かれます。これも「Y」でいいでしょう。
④ testデータベースも削除(「Y」)でいいです。
⑤ 「Reload privilege tables now?(権限テーブルを再読み込みするか?)」と聞かれるので、「Y」を選択します。

設定が終わったら、MySQLにログインできることを確認します。次のコマンドを入力すると、

▼図13　Apache 2 Test Page



パスワードを聞かれるので、先ほど設定したMySQLのrootユーザのパスワードを入力します。

```
# mysql -u root -p
```

ログインに成功したら、MySQLの初期設定は完了です。

PHPの動作確認

/var/www/htmlの下に次の内容で「info.php」ファイルを作成します。

```
<?php phpinfo();
```

次に、「http://localhost:8080/info.php」にアクセスします。インストールされているPHPの情報が表示されれば、PHPが正常に動作しています。

PHPとMySQLの連携

最後に、PHPからMySQLにアクセスして、データを取得する動作の確認を行います。まず、MySQLにログインして、テスト用のデータベースを作成します。

```
# mysql -u root -p
mysql > CREATE DATABASE test_db;
mysql > USE test_db;
```

次に、テスト用テーブルとデータを追加します。

```
mysql > CREATE TABLE users (id SERIAL, name
VARCHAR(255));
mysql > INSERT INTO users (name) VALUES
('hoge'), ('fuga');
```

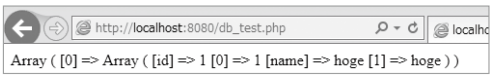
念のため、追加できていることを確認します。次のように表示されればOKです。

```
mysql > SELECT * FROM users;
+----+------+
| id | name |
+----+------+
|  1 | hoge |
|  2 | fuga |
+----+------+
2 rows in set (0.00 sec)
```

▼リスト1　db_test.php

```
<?php
$dsn = 'mysql:dbname=test_db;host=localhost;
charset=utf8';
$user = 'root';
$password = '';
$dbh = new PDO($dsn, $user, $password);
$sth = $dbh->prepare('SELECT * FROM users
WHERE name = :name');
$sth->execute(array('name' => 'hoge'));
$result = $sth->fetchAll();
print_r($result);
```

▼図14　db_test.phpのブラウザでの実行結果



無事に追加できていることが確認できたら、exitでMySQLを抜けます。

次に、/var/www/htmlの下に、リスト1の内容で「db_test.php」ファイルを作成します（$passwordには、MySQLのrootユーザのパスワードを設定してください）。

作成できたら、実行してみましょう。

```
# php /var/www/html/db_test.php
```

先ほどusersテーブルに追加したユーザの名前が表示されれば、PHPとMySQLの動作確認も成功です。ブラウザでの実行結果は図14のような画面が期待されます。

◆◆◆

以上、駆け足になりましたが、VirtualBoxでのLAMP環境の構築方法を見てきました。昨今はVagrantなど、仮想マシンを便利に使えるツールが充実してきているため、開発環境を作る際にOSのインストールから行うことは減ってきていると思います。しかし、独自の開発環境を作るには、ゼロから構築する方法を押さえておいたほうが、融通が利きます。この機会に、LAMP環境構築の基礎を見直してみると、新しい発見があると思います。SD

安定のLAMPを変化に強くする!

第3章 ツールなしでもここまでできる! bash＋シェルスクリプトでLAMP環境一発構築

Web界隈では、ChefやDockerといった構成管理ツールの話が盛り上がっています。しかし、そのようなツールを導入し、学習しなくとも、Linuxエンジニアが使い慣れているシェルスクリプトで十分な場合もまだまだあります。本章では、あらためてシェルスクリプトの良さを見なおしてみたいと思います。

Writer 竹原 俊広(たけはら としひろ) ㈱クラウドエイジア 代表取締役 URL http://www.cloud-asia.co.jp/

なぜいまさらシェルスクリプトでやるのか?

流行りのツールは万人向けか?

コンピュータの低価格化やクラウドの普及が進み、大量のコンピュータを構築・運用する場面が増えてきました。そして大量のコンピュータを管理する技術として、インフラをプログラミングしてコード化するInfrastructure as Codeの浸透が進んでいます。それを実現するツールとしてPuppet、Chef、Ansible、Dockerなどのツールがあります。これらの中でもChefは比較的注目度が高かったので、使ってみたことのある方は多いのではないでしょうか? しかし、使ってみたけれど難しく(すぐには理解できない)、使いこなすためには学習コストが高く、かつ機能が豊富すぎて複雑な面もあるため、導入には至らなかったケースも少なくないようです。筆者がサーバ構築を担当したお客さまの中にも、そういう状況の方がいらっしゃいました。

本章では、シンプルな構成であれば、シェルスクリプトを使えばそれなりに簡単な手法で同等のことができるということをご紹介したいと思います。なお本稿では、Apache HTTP ServerのことをApache、MySQL ServerのことをMySQLと略して呼称するものとします。

シェルスクリプトで行うメリット

ChefやDockerなどの便利なツールがあるのになぜ(いまさら)シェルスクリプトなのか、という疑問に対する「シェルスクリプトで書くメリット」を次に紹介します。

- シェルはどのLinuxでもパッケージインストールなどの必要がなく、そのままで使用可能。そのためOS上の構成変更の必要がない。とくに今回使用するbashはLinuxのシェルの中で最も広く普及しているため、さまざまな年代や地域(国)の方でも共通で使えるスクリプト言語である
- サーバ構築だけでなく、実際にシステムを構築／運用する際にもシェルを使う場面は多くあるため、その経験を活かして手軽に書ける
- 構築手順をスクリプト化しやすい。たとえば一度実行した一連の手順をhistoryコマンドでファイルに出力し、それを整えることでそのままシェルスクリプトにできる

シェルスクリプト化すると便利なケース

では実際にどのようなケースで便利なのか、以下にいくつか挙げてみます。

① 開発環境やテスト環境を同じ構成で立てることが多い場合。たとえばいつもWordPressのサーバを構築するなど(繰り返し)
② 複数の同じ環境を構成する必要がある場合。たとえば連携する複数台の同じサーバが必要な場合など(複数)

開発環境、テスト環境、本番環境といった環境を同じ構成にしたとき、場合によっては開発を進めていくと構成を変更(調整)する必要が出

てくることがあると思います。そのような構築運用をする場合には、次章のAnsibleなどのツールを使ったほうが良いと考えます。

サンプルスクリプトの概要

さて本題です。シェルスクリプトでLAMP環境を構築してみましょう。簡単に言うとPHPとMySQLが動くLinuxサーバです(**表1**)。オープンソースソフトウェア(OSS)で代表的なものですとCakePHPやWordPress、MediaWikiなどが動くサーバということになります。

本章で使用するシェルスクリプト(setting_lamp_server.sh)で行う作業は次のとおりです。

- OSの基本設定と必要パッケージ類のインストール
- ApacheとPHPのインストール
- MySQLのインストールと設定

このシェルスクリプトをGitHubに公開[注1]してあります。これをベースに解説を進めていきますので、手元にダウンロードして参照いただくとよりわかりやすいと思います。

なお、このシェルスクリプトの実行環境はCentOS 6.5 64bit版を前提にしています。ただし、本スクリプトは開発環境を前提としておりiptablesの無効化などを行っていますので、本番環境にはそのまま使用しないようにご注意ください。

▼表1　本章で構築を想定する環境

OS	CentOS 6.5 64bit版
Webサーバ	Apache HTTP Server 2.2.15
開発言語	PHP 5.3.3
データベース	MySQL Server 5.1.73
用途	開発環境

※上記は執筆時点でのyumでインストールされるバージョン

実行方法とデバッグ方法

LAMPを構築するシェルスクリプトの説明をする前に、シェルスクリプトの実行方法とデバッグ方法を説明しておきます。

シェルスクリプトの設置場所

このシェルスクリプトは単体で動作し完結するようになっています。そのため、ほかのファイルやディレクトリとの相対的な位置関係に影響を受けません。サーバのどこに置いても実行することが可能です。今回は説明のために例として**/usr/local/src**にシェルスクリプトを設置することとして説明します。

実行方法

図1のようにスクリプトの置いてあるディレクトリまで移動し、実行するスクリプト名の前に**bash**と付けます。ファイルパーミッションで実行権限が付いている場合は、最後の行を**図2**のようにしても実行できます。

▼図1　シェルスクリプトの実行

```
# cd /usr/local/src
# yum -y install wget
# wget https://raw.githubusercontent.com/nouphet/lamp_build_script/master/setting_lamp_server.sh
# bash setting_lamp_server.sh
```

▼図2　シェルスクリプトに実行権限を付けた場合

```
# ./setting_lamp_server.sh
```

▼図3　デバッグモードでの実行

```
# cd /usr/local/src
# bash -x setting_lamp_server.sh
```

注1）http://github.com/nouphet/lamp_build_script

デバッグ方法

図3のように-xというオプションを付与することで、実行中の状態を確認できるデバッグモードでの実行になります。例としてユーザ判定のところが実行されているときの中身を確認してみましょう。**リスト1**がデバッグ対象の処理で、**図4**がデバッグを実行した結果です。

whoamiコマンドで取得された現在のユーザが$USER変数に代入され、**['root' '!=' root ']**となっているのがわかります。その結果を元に処理を続行するか停止するかを判定させています。このように実行中、どのような値が変数に格納されているのかを見ることができるので、うまく動かない場合のデバッグができます。

▼リスト1　デバッグを実行するスクリプト文（実行ユーザの権限チェック処理）

```
# ユーザのチェック
USER=`whoami`

if [ "$USER" != 'root' ]; then
    echo "rootユーザでスクリプトを実行してください。"
    echo "Press Enter to finish"
    read Enter
    exit 1
else
    echo "rootユーザです。"
    echo "このまま処理を続行します。"
    echo "Press Enter to Continue"
    read Enter
fi
```

パラメータの設定

さて、ここからシェルスクリプト本体の解説に入ります。部分的にピックアップしながら処理の流れを説明します。

まずはMySQLのパラメータ設定です。インストール後にこのパラメータを適用します。そのパラメータを**リスト2**のようにあらかじめ設定しておきます。

ユーザ判定

このスクリプトで実行する処理はroot権限でないと実行できずエラーになってしまうものが多くあります。そのためシェルの実行ユーザを確認する処理を入れておきます(**リスト1**)。ちなみにrootユーザでない場合は、exitして処理を中断するようにしてあります。

必要な基本的ツール群のインストール

構築に必要になるパッケージをいくつかインストールします(**リスト3**)。ntpは時刻同期を

▼リスト2　MySQLのパラメータ設定

```
## パラメータの設定
### Set to MySQL Server
DB_NAME="sample-domain"
DB_ROOT_PASSWORD="P@ssw0rd"
```

▼図4　リスト1のデバッグを実行した結果

```
++ whoami
+ USER=root
+ '[' root '!=' root ']'
+ echo $'root 343 203 246 343 203 274 343 202 266 343 201 247 343 201 231 343 200 202'
rootユーザです。
+ echo $' 343 201 223 343 201 256 343 201 276 343 201 276 345 207 246 347 220 206 343 202 222 347 ☑
266 232 350 241 214 343 201 227 343 201 276 343 201 231 343 200 202'
このまま処理 を続行します。
+ echo 'Press Enter to Continue'
Press Enter to Continue
+ read Enter
```

してくれるパッケージです。vimはviの上位版で編集画面のカラー表示や拡張機能が多々あります。これはお好みで入れてください。

OSの基本構築

SELinuxを無効化

SELinuxは本格的なセキュリティを設定できるのですが、複雑なポリシーを組んでいくことになるため、ここでは無効化しておきます(**リスト4**)。

時刻同期デーモンの有効化

PHPプログラムに直接の関係はありませんが、開発環境の場合はログを確認する場合が多いと思います。ログの時間がずれているとデバッグがしづらいためntpを起動して時刻を同期しておきましょう(**リスト5**)。

ApacheとPHPをインストール

続いてApache(httpd)とPHP(5.3)のパッケージをインストールします(**リスト6**)。PHPに必要となる関連モジュールも同時にインストールしておきます。とくにphp-mysqlは入れ忘れてしまうとMySQLと接続できませんのでご注意を。

iptablesを停止

開発環境なので、iptables(Firewall)は不要として停止しておきます(**リスト7**)。

続いてPHPの情報一覧を出力するファイル

▼リスト3　基本的なパッケージのインストール処理

```
echo "cd /usr/local/src/"
cd /usr/local/src/

# 必要となる基本的なパッケージ類をインストール
echo "yum -y install wget ntp vim"
yum -y install wget ntp vim
```

▼リスト4　SELinuxの無効化処理

```
echo "# disable SELinux"
echo "## SELinux 設定変更前確認"
getenforce
setenforce 0
echo "## SELinux 設定変更後確認"
getenforce
perl -p -i.bak -e 's/enforcing/disabled/g' /etc/selinux/config
```

▼リスト5　時刻同期デーモンの有効化処理

```
echo "# 時刻同期デーモンの有効化"
date
chkconfig ntpd on
chkconfig ntpd --list
/etc/init.d/ntpd start
ntpq -p
NTP_IP=`ntpq -p |grep -v remote |grep -v "=====" |head -1 |awk '{print $2}'`
/etc/init.d/ntpd stop
ntpdate $NTP_IP
/etc/init.d/ntpd start
date
```

▼リスト6　ApacheとPHP関連のインストール処理

```
yum -y install httpd php php-cli php-curl php-pear php-mysql php-imagick curl imagemagick
chkconfig httpd on
chkconfig httpd --list
/etc/init.d/httpd start
```

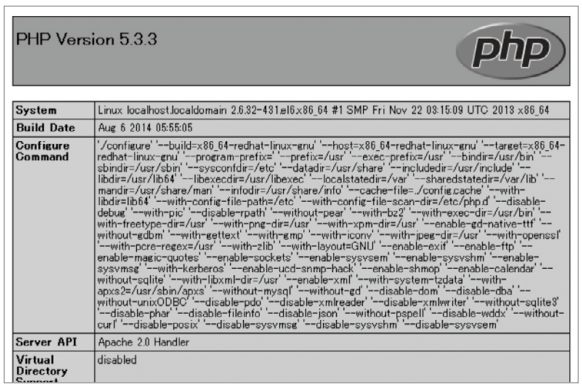
▼リスト7　iptablesの停止処理

```
echo "iptablesを停止します。"
/etc/init.d/iptables stop
chkconfig iptables off
chkconfig iptables -list
```

▼リスト8　PHPの情報一覧を出力するファイルを生成

```
cat << _EOT_ > /var/www/html/phpinfo.php
<?php
phpinfo();
?>
_EOT_
```

▼図5　シェルスクリプトで構築されたWebサーバの画面



▼リスト9　MySQLのインストールと初期設定処理

```
yum -y install mysql-server
chkconfig mysqld on
chkconfig mysqld --list

echo "MySQL Serverを初期設定します。"
/etc/init.d/mysqld start
# ここで初期設定スクリプトを実行
# パスワード設定などもここで
/usr/bin/mysqladmin -u root password $DB_ROOT_PASSWORD
/usr/bin/mysqladmin -u root -p$DB_ROOT_PASSWORD -h localhost.localdomain password $DB_ROOT_PASSWORD
```

を生成します(**リスト8**)。なおこのファイルは本番環境などではないほうが好ましいため、本番向けの場合は生成しない、または動作確認後削除するようにしてください。

MySQLのインストールと初期設定

リスト9で、MySQLのインストールとrootユーザのパスワード設定など、初期設定を行います。そのほかの設定を追加したい場合はここに追記してください。

PHPを動かしてみる

図1のようにシェルスクリプトを実行したら、Webブラウザに次のURL(<IPアドレス>部分には設定したアドレスを入力)を入れてアクセスしてみましょう。

```
http://<IPアドレス>/phpinfo.php
```

図5のような画面が表示されたら正常動作しています。

おわりに

ここまでででPHPとMySQLが動くLinuxサーバが完成しました。いかがでしたでしょうか？

このスクリプトを使えば数分でこの構成が完了します。素早く的確に何度でも同じ構成ができるので重宝するのではないでしょうか。

また、この後に構成変更などを行っていく場合は、「○○が入っていたらスキップして次の処理を行う」など、状態によって判断が必要な場合も出てきます。そのときは次章のAnsibleなどのツールを使ってみるのも良いでしょう。より便利に楽しく環境構築を行うために、みなさんもぜひチャレンジしてみてください！SD

安定のLAMPを変化に強くする!

第4章

中小規模のサーバ構築にオススメ! AnsibleでLAMPをコード化しよう

構成管理ツール(プロビジョニングツール)を使ってサーバをコードで管理するのがトレンドですが、話題のChefをコストをかけて導入するほどの規模ではないと判断される声もしばしば聞かれます。本章で紹介するAnsible(アンシブル)も構成管理ツールの1つですが、事前に必要な環境整備も少なく、特定の言語にも縛られない柔軟さが中小規模での活用に適しています。

Writer 橋本 猛(はしもと たけし) ㈱クラウドエイジア

構成管理ツールとしてのAnsible

Ansibleの説明に入る前に、まずは、構成管理ツールについて確認することにしましょう。実際の企業の中で構成管理の対象となるものとしては、ITインフラを構成するハードウェアやソフトウェア、それらに関連するドキュメントやプログラムはもちろんのこと、ITインフラを維持・管理するために必要な構築手順書などの作業文書も含まれているのではないでしょうか。

今回LAMP環境の再点検をするにあたり、サーバ構築作業に着目して、

・サーバ構築作業
・サーバ構築手順書などのドキュメントづくり
・構築作業者の暗黙作業

など、これまでサーバ構築担当者や運用管理担当者が手作業で行っていた内容を、Ansibleで「Infrastructure as Code」化していく手法を紹介します。

Ansibleを利用することで、

・サーバ構築作業の簡素化、標準化
・サーバ構築手順などの変更管理
・構築作業者の暗黙作業の可視化

を実現し、運用効率の向上を目指しましょう。

Ansibleとは何なのか?

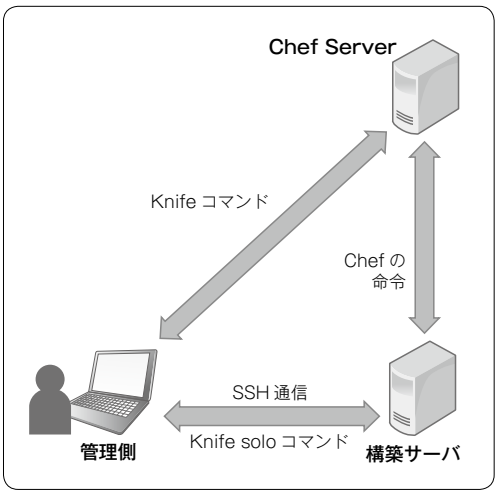
Ansibleとは、ソフトウェアのデプロイ・設定を自動化するツールです。開発者はMichael DeHaan。2014年8月6日の時点でAnsible 1.7がリリースされています。1.7ではWindowsのベータ版が含まれ、1.8ではSoftLayer、Windows Azureへの対応が予定されています。

Ansibleはソフトウェア構成管理ツールの一種で、PuppetやChefのようなツールと考えればわかりやすいでしょうか。

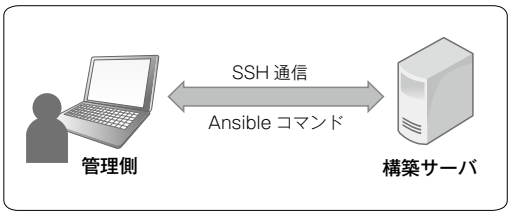
なぜAnsibleを使うのか?

Chefは Rubyにより構成されていますが、Ansibleは Pythonで書かれています。とはいえAnsibleでは幅広い言語でモジュール(後述)を構成できるようになっているので、運用時にPythonの知識が必ずしも必要になることはありません。また、実行時に構築サーバ側にChefクライアン

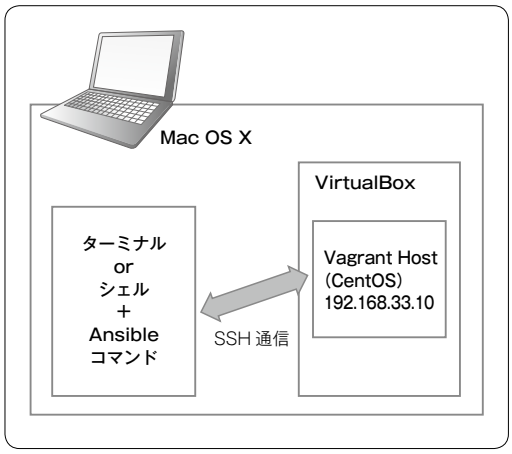
▼図1 Chefの動作イメージ



▼図2　Ansibleの動作イメージ



▼図3　動作テスト環境



トのインストールが必要なChef(**図1**)やChef solo、管理者側でインストールが必要なknife soloコマンドと違い、Ansibleは構築サーバ側にSSH通信を利用して直接命令を送り込む形式となるので、構築サーバ側へのインストールが不要です(**図2**)。

Ansibleは動作がシンプルなゆえに、Chefなどと比べて導入時の学習コストが低く、中小規模の実運用に導入しやすいツールとなっています。書籍やインターネット上での日本語情報はChefほど充実しているとは言えませんが、単構成のサーバや、小規模構成でのシンプルな運用に導入できる程度の情報は得られます。PuppetやChefの導入を検討したが、Chef + Rubyの学習に時間がとれない、Chef Server構成まで準備するほどの運用規模ではない、Puppetを導入したけど実運用できなかったといった方には、Ansibleでの構成管理を再検討してもらいたいです。

Ansible管理側必要条件

ここで言う管理側(**図2**参照)とは、Ansibleを実行し、サーバ構築担当者が操作する側を意味します。管理側のOSは次のものがサポートされています。

・Red Hat Enterprise Linux
・Debian
・CentOS
・BSD系

これに加えて、Python 2.6がインストールされていることが必要になります。Windowsで

Ansibleを利用する場合は、VirtualBoxなどで上記Linux系OSを仮想構築して、その中から操作したほうが簡単です。Windows上だとPythonがうまく動作しなかったり、Windowsのコマンドライン(あるいはPowerShell)からSSH接続するためのOpenSSHへのパスを設定する必要があったりと手間がかかるためです。

Ansible構築側必要条件

一方、構築側(**図2**参照)にはPython 2.4以上が必要になります。最近のLinux系OSではPythonが標準でインストールされている場合が多く、CentOS、Ubuntuなどで構築する際は、あらためてPythonをインストールする必要はありません。

Ansibleによる構成管理環境の準備

本稿では管理側のマシンをMac OS X 10.9とし、Homebrewを利用してAnsibleをインストールします。また今回の構築は同一マシン上で行うこととし、VirtualBox上でCentOS 6.5を立ち上げ、これを構築側マシンとします(**図3**)。

Homebrewをインストール

HomebrewとはOS X用のパッケージマネー

ジャです。パッケージの導入が容易になるため、まずはHomebrewをrubyコマンドでインストールします(**図4**)。詳しくは公式サイト[注1]で確認してください。なお、**図4**の2行目では、SSH認証に必要となるsshpassをインストールしています。

Ansibleをインストール

続いて、次のようにHomebrewを利用してAnsibleをインストールします。

```
$ brew install ansible
```

Ansibleの動作確認

インストールが完了したら、構築側への操作が行えるか確認します。まずはVirtualBoxとVagrantを利用して構築側の仮想マシンを立ち上げることからです。VirtualBoxとVagrantがインストールされていない場合は、それぞれ公式サイト[注2]からダウンロード&インストールをしておいてください。そしてVagrantbox.es[注3]から「CentOS 6.5 x86_64/VirtualBox」を利用して仮想サーバを準備します(**図5**)。

図6のように任意に作成したディレクトリで

▼図4　Homebrewとsshpassのインストール

```
$ ruby -e "$(curl -fsSL https://raw.github.com/mxcl/homebrew/go/install)"
$ brew install https://raw.github.com/eugeneoden/homebrew/eca9de1/Library/Formula/sshpass.rb
```

▼図5　vagrant boxの準備

```
$ vagrant box add centos65 https://github.com/2creatives/vagrant-centos/releases/download/v6.5.3/ ☑
centos65-x86_64-20140116.box
$ vagrant box list
centos65         (virtualbox, 0)   仮想マシンの構築を確認
```

▼図6　VagrantfileへのSSH設定

```
$ mkdir ansibleDemo
$ cd ansibleDemo
$ vagrant init
$ vi Vagrantfile   viでVagrantfileを開く
…省略…                                                    ┐
config.vm.box = "base"                                     │
 ↑この行を↓のように変更。図5で作ったbox名を指定する          │
config.vm.box = "centos65"                                 │
…省略…                                                     │
#config.vm.network  "private_network",ip: "192.168.33.10"  Vagrantの初期値設定IP   │ Vagrantfileの
 ↑この行を↓のように変更。コメントアウト(#)を外す            │                    │ 編集中
config.vm.network "private_network",ip: "192.168.33.10"    │
…省略…                                                     │
:wq                                                        ┘
$ vagrant up
$ vagrant ssh
# exit   SSH接続が確認できたら構築側からexit
$ vagrant ssh-config --host 192.168.33.10 >> ~/.ssh/config
 vagrantで起動した仮想サーバへのSSH設定をする
```

注1）http://brew.sh/index_ja.html
注2）VirtualBox URL https://www.virtualbox.org/wiki/Downloads
　　Vagrant URL http://www.vagrantup.com/downloads.html
注3）http://www.vagrantbox.es/

vagrant initを実行し、Vagrantfileを作成します。そのVagrantfileの内容を、管理側から構築側へSSH接続できるように書き換えます。設定が終わったらvagrant upし、SSH接続を試してみてください。

SSH接続が確認できたら、**図7**のようにAnsibleで構築するサーバを記述するhostsファイルを作成し、Ansibleの接続テストをします。Ansibleの接続チェックに使用しているコマンドのオプションは**表1**のとおりです。

hostsファイルの書き方

hostsファイルの記述では、**[]**付きでグループ名を記述することが可能です。Ansible構築時にグループを指定することで、複数の構築サーバを同時に構築することが可能です(**リスト1**、**図8**)。

Playbookについて

AnsibleのPlaybookは、構築サーバの構成内容を記述したものになります。Chefで言えばレシピ、Puppetで言えばマニフェストにあたる設定記述ファイルです。記述はYAML形式で行います。インストールするパッケージや処理、設定を「tasks:」の項目に記述していくことで、記述された処理が実行されていきます。

たとえば、**図7**で接続テストしたpingをPlaybookに記述すると**リスト2**のようになります。sample.ymlの各記述項目は**表2**のとおりです。

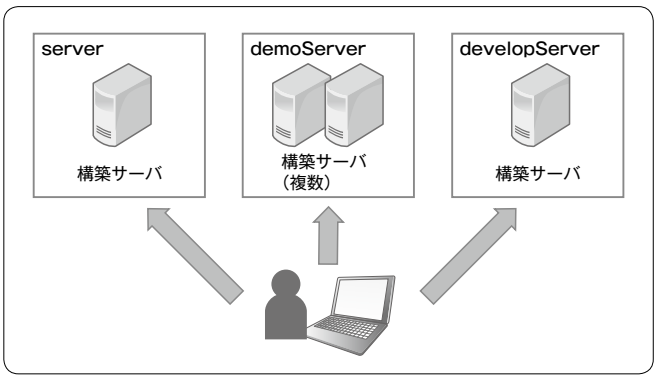
▼図7　Ansibleの接続確認

```
$ echo “192.168.33.10” > hosts
$ ansible -i hosts all -m ping -k
SSH password: vagrantパスワード  Vagrantの初期設定の場合、「vagrant」
192.168.33.10 | success >> {
    "changed": false,
    "ping": "pong"
}
```

▼表1　Ansibleの基本コマンドオプション

オプション	説明
-i --inventory-file=INVENTORY	対象になるサーバを記述したファイル(インベントリファイル)を指定
-m --module-path=MODULE_PATH	モジュールファイルを指定
-k --ask-pass	SSHのパスワードを要求

▼図8　hosts(リスト1)によるグループ設定のイメージ



▼リスト1　hosts記述例

```
[server]
192.168.33.10
[demoServer]
demoserver1.demo.com
demoserver2.demo.com
[developServer]
develop.demo.com
```

▼リスト2　pingを実行するPlaybook例(sample.yml)

```
- hosts: server
  user: vagrant
  sudo: yes
  tasks:
    - name: ping check
      action: ping
```

▼表2　sample.ymlの記述について

項目	説明
hosts:	対象のサーバグループを指定
user:	対象サーバで実行するユーザ指定
sudo:	対象サーバでsudoコマンドを使って実行するか
tasks:	実行する処理 - name: 　taskの処理の名前になり、任意の文字列を記述できる action: 　処理内容の指定(例では引数としてpingを指定)

Playbookを使った実行とデバッグ

それではPlaybook(sample.yml)を使ってAnsibleを実行してみましょう。そのまえに、Playbookに間違いがあると処理がスキップされたり、処理が途中終了するなど正常に構築できないといった問題が発生してしまうので、Playbookのデバッグは必ずしておきましょう。

ansibleコマンドに「--syntax-check」オプションを付けることでPlaybookの構文チェックを実施します(**図9**)。たとえば、「name:」項目にハイフンを付け損なっていた場合にこのコマンドを実行すると、**図9**のコメント以降のようなエラーが表示されます。

また、**図10**のように「--check」オプションを付けると、パッケージなどのインストールは実行されませんが、Playbookの実行動作を確認するデバッグを行います。

エラーがないことを確認したら、sample.ymlを実行しましょう(**図11**)。**図7**と同様pingが実行されます。

モジュールについて

-mオプションで指定するモジュールについてですが、Ansible Documentationでモジュール数を確認すると、記事作成時には200以上のモジュールが記載されていました[注4]。

モジュールとはAnsibleを使用する際に同じ処理をする操作を動作単位でまとめて、部品化してあるものです。このモジュールの作成言語は幅広く、Pythonだけでなくbashで書かれていても動作します。サーバ構築時にいつも行う処理をbashで書いていたなら、それを独自モジュールとして作成、登録しておくと便利です。具体的には、SELinuxを無効化してから、ntpパッケージを利用してサーバの時刻同期を実行するまでを1つの独自モジュールとして登録することで、まとめて1つの処理として利用することができます。

独自モジュールとすることで作業者の暗黙処理などを標準化したり、構築作業の漏れを防いだりできるので、ローカルルールのモジュール化に挑戦してほしいです。

▼図9　--syntax-checkでデバッグ

```
$ ansible-playbook -i hosts sample.yml --syntax-check
playbook: sample.yml   エラーがなかった場合の出力

 - nameのハイフンがなかった場合のsyntax-check結果
ERROR: Syntax Error while loading YAML script, sample.yml
Note: The error may actually appear before this position: line 6, column 13
     name: ping check
      action: ping
            ^
```

▼図10　--checkでデバッグ

```
$ ansible-playbook -i hosts sample.yml --check
playbook: sample.yml   エラーがなかった場合の出力
```

▼図11　sample.ymlの実行

```
$ ansible-playbook -i hosts sample.yml
…省略…
PLAY RECAP ************************************192.168.33.10 : ok=2 changed=0 unreachable=0    failed=0
```

注4) http://docs.ansible.com/list_of_all_modules.html

LAMP環境構築用Playbookの作成

最後に実践としてLAMP環境を構築するPlaybookを書きましょう。bash＋シェルスクリプトの構成と比較するために第3章で実行したコマンドをPlaybookに置き換えてみます。取り回しがしやすいように、5つのモジュールに分けて作ります。

1 必要な基本的ツール群のインストール

yumモジュールを利用してwget、ntp、vimパッケージをインストールします。Playbookは**リスト3**のように書きました。

「name:」は必須ではありません。ラベルのようなもので、日本語記述も可能です。「yum:」はyumモジュールを利用して、httpdの最新版をインストールする指定です。

2 SELinuxの無効化

セキュリティを考えれば重要なSELinuxですが、ここでは設定を簡単にするためひとまず無効化するモジュールを作ります(**リスト4**)。「command:」には、実行したいコマンド文を渡すことでコマンドが実行されます。「ignore_errors:」を付けることによって、すでにSELinuxがPermissiveの状態でエラーとなっても、処理を継続するようにしています。

3 ApacheとPHPをインストール

Apache(httpd)とPHP 5.3をインストールするPlaybookです(**リスト5**)。「vars:」では各変数を指定しています。「service:」で、httpdサービスを起動しています。「with-items:」を利用して、yumでインストールするパッケージを列挙してインストールします。

4 iptablesを停止

iptablesを停止させます(**リスト6**)。「service:」モジュールを使って停止の指示が出せます。

5 MySQLのインストールと初期設定

MySQLのインストールとrootのパスワード設定を行うPlaybookは**リスト7**のようになります。DBの追加などがある場合は、「tasks:」に追加することで、初期設定をすることができます。

◆◆◆

以上、5つのPlaybookを実行すれば第3章と同じ構成の環境が構築できます。

▼リスト3　必要な基本的ツール群のインストール

```
- hosts: server
  user: vagrant  SSHユーザを指定
  sudo: yes
  tasks:
# 必要な基本的ツール群のインストール
  - name: Install wget ntp vim package
    yum: name={{ item }} state=present
    with_items:  パッケージが増える場合はこの下にパッケージ名を追加する
      - wget
      - ntp
      - vim
```

▼リスト4　SELinuxの無効化

```
- hosts: server
  user: vagrant
  sudo: yes
  tasks:
  - name: SELinux Disable
    command: setenforce 0
    ignore_errors: True
  - name: Edit selinux config
    command: sed -i -e "s/^SELINUX=enforing/SELINUX=disabled/g" /etc/selinux/config
```

▼リスト5　ApacheとPHPのインストール

```
- hosts: server
  user: vagrant
  sudo: yes
  vars:
       php_version:5.3.3
  tasks:
    - name: install httpd
      yum: name=httpd state=latest  latestで最新版を指定
    - name: httpd is service start
      service: name=httpd state=running enabled=yes  enabled=yesで自動起動を指定
    - name: Install php and package
      yum: name={{ item }} state=present
      with_items:
        - php{{ php_version }}  vars:の変数にバージョンを指定
        - php-cli
        - php-curl
        - php-pear
        - php-mysql
        - php-imagick
        - curl
        - imagemagick
```

▼リスト6　iptablesを停止

```
- hosts: server
  user: vagrant
  sudo: yes
  tasks:
    - name: stop iptables
      service: name=iptables state=☑
stopped
```

▼リスト7　MySQLのインストールと初期設定

```
- hosts: server
  user: vagrant
  sudo: yes
  vars:
    mysql_root_pw: "passw0rd"
 tasks:
  - name: insall mysql-server
    yum: name=mysql-server state=installed
  - name: mysqld service start
    service: name=mysqld state=running enabled=yes
  - name: set mysql root password
    command: mysqladmin -u root password '{{ mysql_root_pw }}'
```

ここまで見てきて、シェルスクリプトと比較してもかなり見通しの良いコード管理ができそうに感じていただけたのではないでしょうか。

また、Ansible Documentationのサイトにはベストプラクティスが掲載されています。

ベストプラクティス
http://docs.ansible.com/playbooks_best_practices.html

Ansibleを使用する際には、Chefのようにディレクトリを分ける必要はありません。しかし多くのサーバを管理する場合には、Ansibleの運用についてベストプラクティスを参考にし、運用をアレンジしてみることを検討してみてはいかがでしょうか。SD

参考資料（Webサイト）　Ansible Documentation
http://docs.ansible.com/

安定のLAMPを変化に強くする!

第5章 サーバ運用の問題点を解決する! DockerでImmutableなLAMP運用
仮想化とコンテナの違いを押さえて再構築

Infrastructure as Codeという考え方が、Web構築・運用を仕事としてきたインフラエンジニアの間でも、その必要性が検討されはじめました。ひとつにChefによる大規模プロビジョニングなどの普及とその使用方法の啓蒙もあると思われます。コードで記述し運用の効率化を図る、そんな流れができつつあります。本章ではInfrastructure as Codeのキーテクノロジーであるを中心に既存のLAMP環境がどのように変化していくか紹介します。

Writer 田中 邦裕(たなか くにひろ)　さくらインターネット㈱ 代表取締役社長　Twitter @kunihirotanaka

Immutable Infrastructureとは何か?

サーバ運用の問題点を解決

最近、Immutable Infrastructureというインフラの考え方が注目を集めています。これは、一度立ち上げたサーバについては設定の変更をしないという考え方です。Immutableとは「不変の」という意味を持つ英単語です。

これまでの考え方では、はじめてサーバ起動したときに初期設定やテストを行い、本番稼働を開始し、本番稼働中においても必要に応じて都度本番環境に設定変更を行っていました。

しかしながら、当たり前と思われていたこのやり方には、いくつもの問題点があります。たとえば、設定変更後に突然サーバの動作が意図しないものになってしまうというものです。テスト環境を作って慎重にテストしたとしても、テスト環境と本番環境にバージョンや設定、構成の違いなどがあって、本番環境での動作がテスト環境のものと異なる結果となるのです。また、設定変更を重ねているうちに、どのような設定をしたのかが追跡できなくなってしまい、同じ設定を再現できないというケースも発生する傾向にあります。

Infrastructure as Codeという考え方

このような状況を改善するために生まれてきたのが、ChefやPuppet、Ansibleといった構成管理ツールで、サーバのあるべき設定や状態などをコードで記述するといったInfrastructure as Codeという考え方が出てきました。

これにより設定変更が履歴管理できたり、同じ設定を複数サーバに行ったりすることが簡単になったのですが、冪等性(べきとうせい)と呼ばれる“何回どのようなサーバに対して構成管理ツールを実行したとしても、同じ状態にしなければならない”という原則の担保は簡単ではありませんでした。

なぜかというと、サーバによってミドルウェア(ApacheやNginxなど)やライブラリのバージョンが異なっていたり、構成管理ツールの実行が途中で失敗したサーバがあるなどの理由で設定に差異が出たり、という状況を吸収しなければならないからです。

実際には、ライブラリや各ソフトウェアの依存関係がコンフリクト(衝突)して、すべてのサーバに同じ設定をしたつもりでも、立ち上げた時期によってサーバ状態が異なってしまうということが発生したり、サーバの再起動をすると異なった状態になってしまったりということも少なくありませんでした。

冪等性である必要がなくなる

これらの問題を解決するのがImmutable Infrastructureであり、常に新しいサーバを立ち上げることで冪等性が必要なくなり、設定変更によるトラブルもなくせます。以前なら、設定変更のたびにサーバを設置したり撤去したりという作業はきわめて非効率でしたが、IaaS型のクラウドやVPSの台頭によって、簡単に仮想サー

バが立ち上げられる環境が整ってきたことから、このような考え方が成立するようになりました。

Immutable Infrastructureの使い方

図1の例は代表的なImmutable Infrastructureの使い方を示しています。

10月25日(10/25)にアプリケーションの修正を反映させたServer 4とServer 5という新しいサーバを用意し、1月15日(1/15)にはセキュリティパッチを適用させたServer 6という新しいサーバを用意し、そして3月1日(3/1)にアプリケーションサーバの設定変更と増設を行うためにServer 7、Server 8、Server 9という新しいサーバを用意しています。

これらの切り替えはロードバランサで行い、使わなくなったサーバは一定期間経たあとで削除してしまいます。

この方法は、Blue-Green Deploymentとも呼ばれ、テスト環境を用意して検証してから本番環境を修正するのではなく、常に新しい設定をしたサーバを用意してロードバランサで切りかえるというものです。この方法にすれば、新しい環境がうまく動かないときには、ロードバランサで切り戻すということが可能になります。

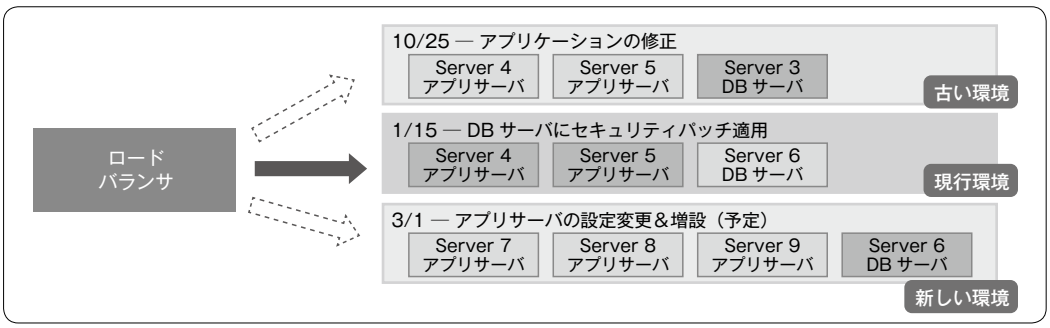
Docker ~加速するImmutable Infrastructure

完全仮想化とコンテナの違い

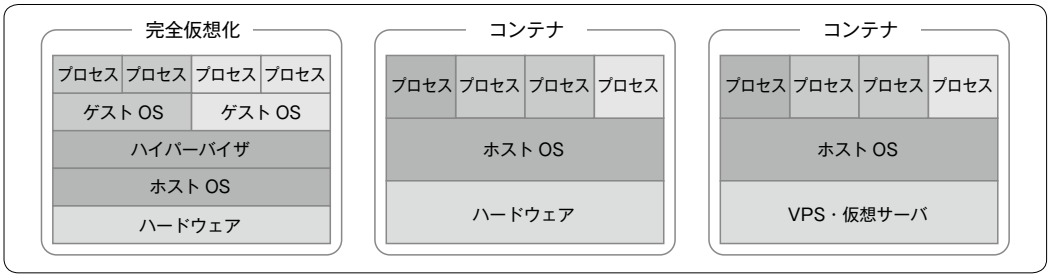
さて、ここまでImmutable Infrastructureの話をしてきましたが、簡単にサーバを作ったり壊したりということを実現するためのソフトウェアとして最近注目されているのがDockerです。DockerはLXC(Linux Containers)というLinuxのコンテナ技術を用いて、サーバ上に独立したOS環境を作れるもので、OS環境を保存しておいたり、ほかのサーバで稼働させたりと、柔軟なサーバ構築を実現できます。

ちなみに、コンテナ技術は仮想化技術と同列視されますが、KVMやXenなどの完全仮想化とはまったく異なるものです(**図2**)。KVMやXenなどがハードウェアのエミュレーションを行ってホストとは異なるOS空間になっているのに対して、コンテナの場合は単一のOSを見

▼図1　Blue-Green Deployment法



▼図2　完全仮想化とコンテナ型(Docker)の違い



かけ上分割しているだけで、ホスト側でpsコマンドを実行するとゲストのプロセスも含めてすべてのプロセスが表示されます。

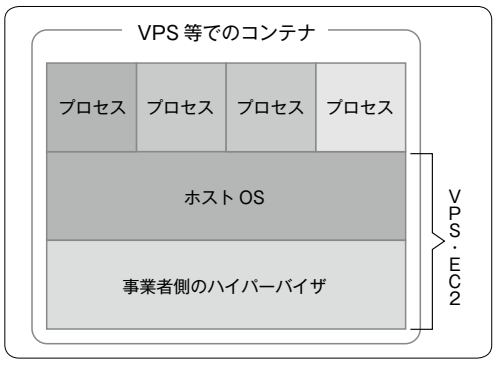
みかけ上の仮想化

Dockerでサーバを起動すると、Dockerの起動時に指定したLinuxのコマンドを新しいコンテナの中で実行します。Docker環境において別々に起動されたサーバは、お互いのファイルシステムやプロセス空間にはアクセスできず、見かけ上仮想化されたような形となります。

仮想化でのBlue-Green Deploymentの考え方

最近では自分のハードウェアを持つことは少なくなり、Amazon EC2やさくらのVPSといった仮想サーバサービスを利用する機会が多くなりました。これらのサービスは前述したゲストOSを借り受けるサービスであり、特殊な技術を使わない限り、自分自身でゲストOSを立ち上げることはできません。しかしながら、OSより上のレイヤで空間を分離するコンテナの場合には、仮想サーバサービスのうえでも利用することが可能になります(**図3**)。

▼図3　VPSとコンテナの関係



▼図4　Dockerのインストール(CentOS 6)

```
$ sudo yum update -y
$ sudo wget -P /etc/yum.repos.d http://www.hop5.in/yum/el6/hop5.repo
$ sudo yum install xz docker-io -y
$ sudo service docker start
Starting cgconfig service:                                 [  OK  ]
Starting docker:                                           [  OK  ]
```

たとえば、さくらのVPSにおいて月額980円のプランを借りたとして、10個のコンテナを立ち上げれば、1コンテナ当たりの月額費用は100円以下にできます。

言うまでもなく、10個のコンテナすべてで負荷をかけるとすると、1コンテナあたりのリソースは厳しくなります。しかしBlue-Green Deploymentの考え方の場合は、新しい設定のサーバを用意して、切り替えが無事に終わったら古いサーバは破棄するので、常にすべてのコンテナに負荷がかかるわけではありません。これまでは、一時的な切り替え用のサーバに対してもコストを負担しないといけなかったものが、仮想的に環境を切り分けられることでコストの一本化ができるというのも非常に大きな利点といえるでしょう。

CentOSにDockerをインストールする

それではCentOSにDockerをインストールしてみましょう。なお、Dockerのサポートは CentOS 6.5からですので、CentOS 6系でなければインストールができません。CentOS 6系であれば、6.5未満であってもyum updateをすれば問題ありません。

手順は簡単です。yum updateで最新バージョンにアップデートし、Dockerのリポジトリをダウンロードしてyum installするだけです(**図4**)。ちなみに本稿執筆時のDockerのバージョンは1.1.2でした(**図5**)。

通常のLAMP環境の管理とDockerによる管理との違い

さて、ここからはDockerを使ってLAMP環境を管理する方法を見ていきますが、Dockerの場合には原則としてサーバを立ち上げてログインして設定変更という方法をとりません。たとえば、Apacheでバーチャルドメインを設定す

ることを考えてみましょう。

図6に示すような従来のやり方であれば、新しいドメインを追加するには稼働中のサーバのApacheの設定ファイルを修正します。

図7に示すDockerの場合には、新しいドメインの設定を記述したサーバを新規に立ち上げて、前段のロードバランサで切りかえ、古い設定のサーバは破棄する形となります。

Dockerを使ってみる

Dockerイメージの取得

それでは、さっそくDockerを使ってみましょう。最初に行うのは、Dockerイメージを用意することです。Dockerイメージとはコンテナを立ち上げるためのテンプレートのようなものです。Dockerイメージは、Dockerレジストリから取得できて、標準ではDocker Hub Registryというレジストリ[注1]が指定されています。

Dockerをインストールするとdockerというコマンドが作成されますので、このコマンドにpullというオプションを使用すると、Dockerレジストリからイメージを取得できます(**図8**)。取得できたDockerイメージの一覧を見るには、imagesというオプションを使用します(**図9**)。これでCentOS 5、CentOS 6およびCentOS 7が取得できたことがわかります。標準でUbuntuも用意されており、**図10**のようにすればUbuntuのDockerイメージを取得できます。

Dockerイメージの実行

ここで用意できたDockerイメージを実行してみます。起動するには**run**というオプションを付け、**-t**で使用するDockerイメージのタグを指定します。そして、そのうしろのオプションで、コンテナ内で実行するコマンドを指定します。

図11の例では、bash経由で、CentOSのバージョン、コンテナ内のプロセス一覧およびインターフェース**eth0**の情報を表示しています。CentOSのバージョンは7.0.1406、プロセスはPIDが1のbashが1つだけ、IPアドレスは**172.17.0.2**であることがわかりました。

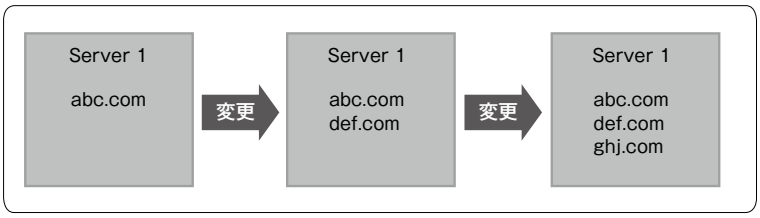
▼図5　Dockerのバージョンの確認など

```
$ sudo docker version
Client version: 1.1.2
Client API version: 1.13
Go version (client): go1.2.2
Git commit (client): d84a070/1.1.2
Server version: 1.1.2
Server API version: 1.13
Go version (server): go1.2.2
Git commit (server): d84a070/1.1.2
```

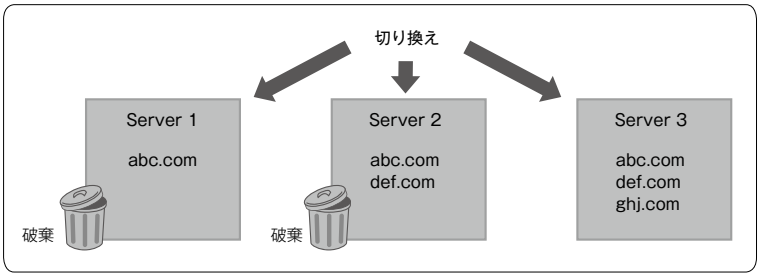
▼図8　Dockerレジストリからイメージを取得

```
$ sudo docker pull centos
Pulling repository centos
5a1ebaa356ff: Download complete
70214e5d0a90: Download complete
68eb857ffb51: Download complete
511136ea3c5a: Download complete
34e94e67e63a: Download complete
```

▼図6　従来の管理方法



▼図7　Dockerを利用した場合の管理方法



注1）https://registry.hub.docker.com/

ちなみに、-tのオプションでcentos:6と指定するとバージョンが6.5に変わっていることがわかります。また、IPアドレスが先ほどとは変更されて172.17.0.3となっています(図12)。

図11と図12でDockerイメージからコンテナを起動しましたが、1回目と2回目の実行で、それぞれ新しい環境が作られたことがわかりました。

図13で起動中のDockerのコンテナ一覧を表

▼図9　imagesオプションの実行

```
$ sudo docker images
REPOSITORY    TAG          IMAGE ID         CREATED         VIRTUAL SIZE
centos        centos5      5a1ebaa356ff     2 days ago      484 MB
centos        centos7      70214e5d0a90     7 days ago      224 MB
centos        latest       70214e5d0a90     7 days ago      224 MB
centos        centos6      68eb857ffb51     7 days ago      212.7 MB
```

▼図10　Ubuntuのイメージ取得

```
$ sudo docker pull ubuntu
Pulling repository ubuntu
245ce11c1f25: Download complete
3db9c44f4520: Download complete
…省略…
$ sudo docker images
REPOSITORY   TAG        IMAGE ID        CREATED       VIRTUAL SIZE
…省略…
ubuntu     14.04        826544226fdc    7 days ago    194.2 MB
ubuntu     14.04.1      826544226fdc    7 days ago    194.2 MB
ubuntu     latest       826544226fdc    7 days ago    194.2 MB
ubuntu     trusty       826544226fdc    7 days ago    194.2 MB
…省略…
```

▼図11　Dockerイメージの実行

```
$ sudo docker run -t centos /bin/bash -c "cat /etc/redhat-release && ps aux && ip addr show eth0"
CentOS Linux release 7.0.1406 (Core)
USER  PID %CPU %MEM    VSZ   RSS TTY STAT START   TIME COMMAND
root    1  0.0  0.0  11552  1360 ?   Ss+  06:37   0:00 /bin/bash -c cat /etc/redhat-release && ps aux ☑
&& ip addr show eth0
root   10  0.0  0.0  19696  1236 ?   R+   06:37   0:00 ps aux
98: eth0: <NO-CARRIER,BROADCAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state DOWN qlen 1000
    link/ether 5e:0e:94:40:4c:8b brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.2/16 scope global eth0
    inet6 fe80::5c0e:94ff:fe40:4c8b/64 scope link tentative
       valid_lft forever preferred_lft forever
```

▼図12　Dockerの実行とバージョンの確認

```
$ sudo docker run -t centos:6 /bin/bash -c "cat /etc/redhat-release && ps aux && ip addr show eth0"
CentOS release 6.5 (Final)
USER  PID %CPU %MEM    VSZ   RSS TTY STAT START   TIME COMMAND
root    1  0.0  0.0  11300  1288 ?   Ss+  06:37   0:00 /bin/bash -c cat /etc/redhat-release && ps aux ☑
&& ip addr show eth0
root    7  0.0  0.0  13364  1052 ?   R+   06:37   0:00 ps aux
101: eth0: <NO-CARRIER,BROADCAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state DOWN qlen 1000
    link/ether 22:4c:e3:7d:2d:47 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.3/16 scope global eth0
    inet6 fe80::204c:e3ff:fe7d:2d47/64 scope link tentative
       valid_lft forever preferred_lft forever
```

示してみます。確認するには、psオプションを利用します。しかし、この状態では何も表示されません。先ほど、2個のDockerコンテナを起動しましたが、コンテナ内で実行されたbashの終了とともに起動したコンテナは破棄されています。通常のLinuxであれば、プロセス番号1のプロセスは/sbin/initであり、デーモンプロセスの起動や停止の処理などを行うとともに、再起動するまではずっとこのプロセスが実行され続けます。

コンテナの場合には2種類の動作があり、/sbin/initを実行して通常のサーバのように起動するシステムコンテナと、アプリケーションだけを独立して実行させるアプリケーションコンテナというものがあります。Dockerの場合にはアプリケーションコンテナとして動作し、Docker起動時のコマンドラインオプションで指定されたコマンドを起動し、そのコマンドの実行が終了すると、コンテナ自体も破棄されることになります。

Dockerを通常のVMwareやKVM、Xenなどの仮想化と同列にみてしまうと理解が難しくなりますが、逆にアプリケーションコンテナというしくみが理解できると、Dockerの使い方は半分以上理解できたといえます。

ログが残らない落とし穴

Dockerはアプリケーションコンテナとして動作するため、コンテナの中でファイルへの書き込みが発生したとしても、プロセス自体が終了してしまうとファイルの書き込み結果は破棄され、再度実行したときには書き込みがなかった状態からシステムが開始されるということを意味しています。

たとえばApacheをDockerから起動したとしても、Apacheが終了するとコンテナ自体が終了してしまい、再度起動したとしてもアクセスログなどは残されていません。

コンテナイメージをコミットする

コンテナは完全消去されない

先ほど、Dockerで起動されたコマンドが終了すると、コンテナが破棄されると解説しましたが、完全に消去されるわけではありません。オプション-aを付けると、終了したコンテナが残されていることがわかります(図14)。

終了したコンテナから新しいイメージを作る

終了したコンテナをベースに新しいイメージを作成(コミット)できます。イメージを作成するにはcommitオプションを使用し、実行したあとにイメージ一覧を見ると、my:test1という新しいイメージが作成されたことがわかります(図15)。

この新しいイメージをベースに、再度コマンドを実行できます。図16の実行結果を見ると、my:test1というイメージを元にps auxが実行されたことがわかります。ちなみに、今回はbashを実行して実際にコンテナ内で作業をしてみます。この場合、-iオプションを付けて対話モードにしなければならないことに注意してください(図17)。

これで、コンテナ内での作業が可能です。コンテナを終了させるには、exitと入力するか^Dを押してください。コンテナ終了後にcommitを実行してイメージ化すれば、この作業が反映されたイメージを作成できます。

▼図13　Docker起動中のコンテナー覧表示

```
$ sudo docker ps
CONTAINER ID  IMAGE    COMMAND   CREATED   STATUS   PORTS   NAMES
```

▼図14　ps -aを実行してコンテナ残留を確認

```
$ sudo docker ps -a
CONTAINER ID IMAGE      COMMAND         CREATED   STATUS     PORTS      NAMES
774a4e8b0840 centos:centos6 /bin/bash -c 'cat /e 12 minutes ago Exited (0) 12 minutes ago
tender_morse
41b2bc26a2da centos:centos7 /bin/bash -c 'cat /e 12 minutes ago Exited (0) 12 minutes ago
jovial_bell
```

▼図15　新しいイメージをコミット

```
$ sudo docker commit 41b2bc26a2da my:test1
11833a8fa7f9ded35f5d91156a80ce23c1badbab18cb2d555423812a52d44990
$ sudo docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             VIRTUAL SIZE
my                  test1               11833a8fa7f9        8 seconds ago       224 MB
…省略…
```

▼図16　新しいイメージでコマンド実行

```
$ sudo docker run -t my:test1 /bin/bash -c "ps aux"
USER        PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root          1  0.0  0.0  17596  1024 ?        Rs+  07:19   0:00 ps aux
$ sudo docker ps -a
CONTAINER ID IMAGE    COMMAND              CREATED        STATUS                      PORTS   NAMES
44100667c5d0 my:test1 /bin/bash -c 'ps aux 13 seconds ago Exited (0) 12 seconds ago         cranky_
engelbart
```

▼図17　対話モードをつけて実行

```
$ sudo docker run -i -t centos:test /bin/bash
bash-4.1#
```

LAMP環境を作る

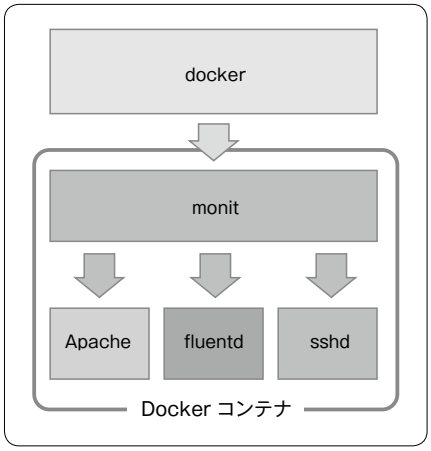
既存の環境を移行するにはいくつかの方法があります。最も美しいのは、サービスごとにイメージを作成して、機能別にコンテナを立ち上げる方法です。

Dockerで monitコマンドを実行

通常のサーバでは、1つのサーバでApacheやsendmail、NTPなどが同居していますが、Dockerの場合には、これらのサービスをコンテナごとに分離するのが望ましいでしょう。

ただ、この方法だとApacheのプロセスの実行状態をコンテナ内で把握することが難しく、最低でもSSHでコンテナ内にログインできたり、システムのログが見られたりしたほうが便利です。そのため、まずは既存のサーバと同じように、ApacheとSSHが同居したイメージを作成してみます。複数のサーバプロセスを実行するのに便利なのがmonitです。monitはサーバプロセスの監視ツールで、Dockerからはmonitをコマンドとして実行し、monitがApacheやSSHなどを起動するというモデルで作成します(**図18**)。

▼図18　monitによるDocker環境



Dockerfileの作成

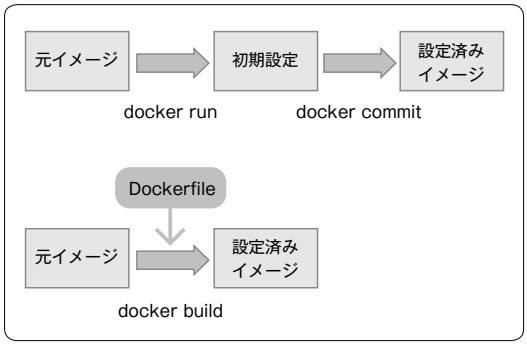
なお、イメージの作成にあたってはログインして初期設定を行うのではなく、Dockerfileというイメージを作成するための定義ファイルを作成して、その定義ファイルを元にイメージを作成してみます(**図19**)。Dockerfileの書式などを覚えるのは少し手間かもしれませんが、buildオプションを使用すれば簡単にイメージを作成でき、イメージの作成をコードでできるということから、ぜひ覚えてもらいたい機能です。

まず、今回の設定を行うにあたって、Dockerfileとそのほかの必要なファイルを用意するために、GitHubからクローンをします(**図20**)。

GitHubからクローンが完了すると、ローカルディレクトリに**docker-centos-lamp**というディレクトリが作成され、Dockerfileと関連する設定ファイルが作成されます。

Dockerfileには、**RUN**や**ADD**といったDockerfile用のコマンドが列挙されており、**RUN**はコンテナ内でのLinuxコマンド実行を、**ADD**はローカルディレクトリのファイルをコンテナ内にコピーすることを示しています。

▼図19　docker run、Dockerfileの実行



Dockerfileを元にイメージを作成

これからDockerfileを元にイメージを作成したいと思いますが、**ENV**でIPアドレスやパスワードを指定するため、ファイルの中を少し修正してもらう必要があります。IPは、monitやphpMyAdminへアクセスを許可する接続元IPアドレスを、PWはMySQLやログインのためのパスワードを、LOGSERVERはfluentdでログを転送する先のIPアドレスを示します(**リスト1**)。

また、**authorized_keys**というファイルもありますので、適宜sshの公開鍵を登録しておいてください。既存の環境から乗り換える場合には、**httpd.conf**も既存環境からコピーしておく必要があるでしょう。

ここまでくれば、あとはコマンド一発です(**図21**)。**docker build**をすれば構築が開始されます(最後のドットを忘れないようにしてください)。**docker images**で、作成したイメージが表示されれば、構築完了です。

LAMP環境を起動させる

これで新しいイメージが完成しましたので、さっそく起動してみましょう。**/usr/bin/monit**を起動すれば、SSH、MySQL、Apache、fluentd(td-agent)が起動されます(**図22**)。

docker psをすると、**monit**が起動してい

▼図20　Dockerの作成(GitHubからクローンを得る)

```
$ git clone https://github.com/kunihirotanaka/docker-centos-lamp.git
Initialized empty Git repository in /home/tanaka/docker-centos-lamp/.git/
remote: Counting objects: 32, done.
remote: Compressing objects: 100% (24/24), done.
remote: Total 32 (delta 11), reused 26 (delta 8)
Unpacking objects: 100% (32/32), done.
$ cd docker-centos-lamp/
$ ls
Dockerfile  README.md        monit.conf   monit.mysqld  monit.td-agent  td.repo
LICENSE     authorized_keys  monit.httpd  monit.sshd    td-agent.conf
```

▼リスト1　Dockerfileの修正

・変更前

```
FROM centos
MAINTAINER Kunihiro Tanaka

ENV IP __YOUR_IP_ADDRESS_HERE__
ENV PW __YOUR_PASSWORD_HERE__
ENV LOGSERVER __YOUR_LOG_SERVER_HERE__
```

・変更後

```
FROM centos
MAINTAINER Kunihiro Tanaka

ENV IP 192.168.50.3
ENV PW Jhd30Kwj
ENV LOGSERVER logserver.example.jp
```

▼図21　Dockerfileを元にイメージを作成

```
$ sudo docker build -t centos:lamp .
…省略…
Successfully built cd934830f515
$ sudo docker images
REPOSITORY        TAG             IMAGE ID        CREATED              VIRTUAL SIZE
centos            lamp            cd934830f515    About a minute ago   822 MB
```

▼図22　monitを起動してLAMP環境を立ち上げる

```
$ sudo docker run -d -t -p 12812:2812 -p 10080:80 -p 10022:22 centos:lamp /usr/bin/monit -I
65d63382ebac74066290024a096fc291cf054435e1d10331ee1a03ac7e5e0c65
```

▼図23　docker psでmonitの起動を確認

```
$ sudo docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              ☑
PORTS                                                                           NAMES
65d63382ebac        b3f874bbbd97        /usr/bin/monit -I   49 seconds ago      Up 48 seconds       ☑
0.0.0.0:10022->22/tcp, 0.0.0.0:10080->80/tcp, 0.0.0.0:12812->2812/tcp   prickly_lovelace
```

るのがわかります(図23)。

正常にmonitが起動しているのを確認できれば、ブラウザからアクセスしてみましょう。各サーバの起動には少し時間がかかるので、アクセスできないときは少し待ってみましょう。

なお、monitの起動時に**-p**というオプションを指定していますが、これはホストのポートをコンテナ環境のポートにマッピングするためのものです。コンテナ内のmonitは2812番ポートで実行されていますが、これをホスト環境の12812番ポートにマッピングしています。つまりコンテナ内のmonitへアクセスするためには、Dockerで動いているサーバの12812番ポートへアクセスすればよく、ホストのIPアドレスが**192.168.1.1**だとすると、**http://192.168.1.1:12812/**というURLでアクセス可能です。ユーザ名はadmin、パスワードはmonitで、これは**monit.conf**の中で変更できます(図24)。

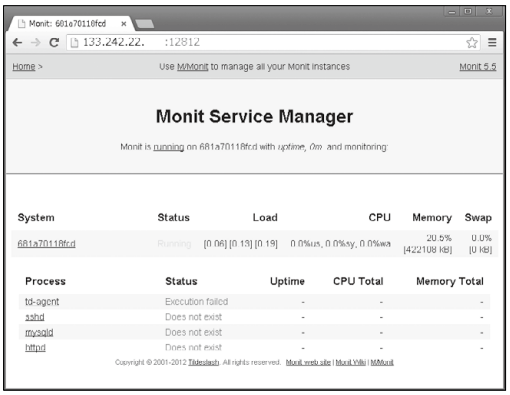
次に、phpMyAdminへもアクセスしてみます。URLは、dockerの動いているサーバの10080番ポートですので、**192.168.1.1**というサーバで動いていたとすると、**http://192.168.1.1:10080/phpmyadmin/**と入力するとアクセスできます(図25)。

sshを行う場合は、10022番ポートへアクセスすればできます。ログインするときは、ユーザ名lampで、先ほど指定したパスワードでできます。

起動したLAMP環境を終了するときはdockerコマンドに**kill**オプションを付けて実行します(図26)。

このときに終了したLAMP環境をもう一度利用する場合にはcommitしてイメージを作成し、その作成したイメージから再度起動すれば再開されます(図27)。ただ、実行し終わった環境を

▼図24　Monit Service Manager



▼図25　PHP Adminの稼働

▼図26　dockerの終了

```
$ sudo docker kill 65d63382ebac
$ sudo docker ps
CONTAINER ID    IMAGE       COMMAND     CREATED      STATUS     PORTS          NAMES
```

▼図27　dockerの再起動

```
$ sudo docker commit 65d63382ebac
$ sudo docker run -d -t -p 12812:2812 -p 10080:80 -p 10022:22 fa55a6cebb12 /usr/bin/monit -I
```

継続して使うというのは、本番環境で発生するサーバ状態の変化をイメージに引き連れてくることを意味しており、ともすれば今までのサーバ管理と同じように、同じ環境を後から作れないというケースになる可能性もあります。

できるだけImmutable Infrastructureを実現し続けるためにも、実行した本番環境をイメージ化するのではなく、常に本番ではないところでイメージの管理を行い、そのイメージから本番環境を作ることを心掛けるほうが良いでしょう。

今回はDockerfileを使った例を見てみましたが、通常のサーバのようにrsyncを使ったコピーの仕方もあります。また、今回はMySQLも同じコンテナ内で立ち上げましたが、Dockerと相性がいいのはロードバランサで、「えいやッ」と切り替えのできるWebサーバが中心になろうかと思います。ですので、MySQLサーバについては従来型の管理をしつつ、フロントエンドで設定変更頻度の多い部分のみをDockerに持ってくるというやり方がお勧めです。

また、Immutable Infrastructureとしての本分を忘れて、稼働中のサーバの変更をしてしまうのは絶対に避けるべきですが、Dockerfileの書き方でつまずいたり、イメージの管理方法に悩み過ぎて、結局Docker導入の敷居が上がってしまうのは本末転倒です。

イメージ更新用のコンテナを立ち上げて、コンテナ内にログインして今までどおりのサーバ設定変更のやり方で修正をしてイメージをcommitし、それから本番環境をrunするという使い方でも十分にImmutable Infrastructureと言えます。最終的には、Dockerfileを使ったり、サーバ構成管理ツールなどでイメージを管理したほうが美しいやり方ですが、まずは本番環境を毎回起動するというImmutable Infrastructureの考え方をDockerを通じて勉強してもらいたいと思います。SD

安定のLAMPを変化に強くする!

第6章 クラウドへの構築・運用テストに Microsoft Azureで作るLAMP環境

Microsoftのパブリッククラウドサービス「Microsoft Azure」は、Windowsテクノロジに強いのはもちろん、Linux OSをベースとするオープンソースソフトウェアでのサーバ構築・運用にも対応しています。これまでWindowsサーバしか使ったことがない方でも、LAMPを気軽に勉強できる環境です。

Writer 桜井 剛(さくらい たけし) ㈱pnop 取締役 Chief Software Architect Mail sakurai@pnop.co.jp

Azureでわざわざ LAMP?

オンプレミス環境を見なおそう

Windows Server 2003のサポート終了が2015年7月に迫っていることが話題になっています。サーバOSの移行はデスクトップOSのWindows XPよりも困難なことが予想されるため、Microsoftは早めの対応を呼びかけています。

Windows Server 2003を自社サーバとして運用している場合は、これまでの資産を引き継ぐことを考えれば最新のWindows Server 2012 R2に入れ替えることを検討されるでしょう。それが移行の第一歩ですが、Microsoftや移行支援サービスを提供するベンダーが声をそろえて提案するのがサーバのハイブリッド運用です。この機会に自社サーバの役割を棚卸しし、オンプレミスにこだわらなくてよいものは適材適所でホスティングサービスやクラウドサービスを利用する(ハイブリッド化)ことで、業務の効率化、製品やサービス開発力の強化を図ります。

また、これまでオープンソースソフトウェア(以下、OSS)を中心に自社サーバを構築してきた企業でも、上記のようなハイブリッド運用による攻めのIT活用を検討すべきときであるのは同様です。

本章ではパブリッククラウドの候補としてMicrosoft Azure(以下、Azure)を紹介します。Windowsテクノロジを中心にシステムが構築されている場合は移行がしやすいのはもちろん、後述するようにOSSを使った環境構築も可能です。検討のために、本章を読みながら無料評価版を使ったLAMP構築をやってみてください。

Microsoft Azureとは

あらためてAzureについて最近の動向を交えて説明しておきましょう。AzureはMicrosoftが提供するパブリッククラウドプラットフォームです。開設当初はWindows Serverをベースにした PaaS(Platform as a Service)でしたが、2012年に仮想マシン(IaaS:Infrastructure as a Service)サービスを発表してからはLinuxも利用できるようになりました。主要なディストリビューション(SUSE、Ubuntu、CentOS、Oracle Linuxなど)のイメージがあらかじめ準備されています。また、SQLデータベース、ストレージ、仮想ネットワーク、HDInsight(Hadoop)、キャッシュなどのサービスも順次提供されています。

2014年2月には日本データセンターも開設され、国内でのサービス提供も可能となりました。ほぼすべてのクラウドの操作を、Webブラウザから Azure管理ポータル[注1](以下、管理ポータル)へアクセスすることで行えますので、HTML5対応ブラウザであればどこからでも利用できます。

Microsoft Azureをはじめるには

Azureをはじめるには、まずAzureのサイト[注2]へ

注1) https://manage.windowsazure.com/
注2) http://azure.microsoft.com/

アクセスします。すでにMSDN[注3]やBizSpark[注4]を利用している場合は、月々決まった無料枠でAzureを利用できる権利があります。これらの特典を利用しない手はありません。また、試しにAzureを利用する場合には、1ヵ月間無料評価版[注5]があります。

詳しくは、無償利用特典一覧[注6]で確認できます。本稿ではこの無料評価版を対象に解説します。

無料評価版でAzureを使う

1ヵ月間無料評価版を利用するためには、まずMicrosoftアカウント[注7]を作成します。ここで登録したメールアドレスとパスワードで、Azureの無料評価版にサインアップします。なお、本人確認のためにクレジットカードの入力がありますが、デフォルトでは請求金額が0円を越える(つまり課金が発生する)と利用が制限される設定ですので支払いは発生しないようになっています。

Azureの無料評価版のサインアップが終わったら、いよいよLAMP環境の構築です。管理ポータルにサインインしてみましょう。サインインすると、**図1**の画面が表示されます。左側には提供されるサービスが並びます。

▼図1　管理ポータル画面

▼図2　管理ポータルでの新規作成

LAMP環境を構築する

仮想マシンを作成する

AzureでLAMP環境を構築するには、Linux仮想マシンを立ち上げてパッケージを導入するのが簡単です。今回は1台でLAMP環境を構築します。OSはUbuntuの場合で説明します。管理ポータルの左下にある「＋新規」をクリックすると**図2**の画面が表示されます。

「コンピューティング」−「仮想マシン」−「簡易作成」を選択します。DNS名にはホスト名を入力します(ここでは「gihyolamp1」としました)。すでに登録されているものは利用できませんのでご注意ください。

イメージは、Ubuntuですので「14.02 LTS」を選択することにします。サイズは仮想マシン

注3) Microsoft Developer Network URL http://msdn.microsoft.com/
注4) スタートアップ支援のためのしくみ URL http://www.microsoft.com/ja-jp/mic/bizspark/
注5) http://azure.microsoft.com/ja-jp/pricing/free-trial/
注6) http://msdn.microsoft.com/ja-jp/windowsazure/gg674969.aspx
注7) http://www.microsoft.com/ja-jp/msaccount/

のスペックですが、今回はデフォルトの「標準A1(1コア、1.75GBメモリ)」を選択します。

ユーザ名は「azureuser」で固定されています。パスワードは任意のものを入力してください。リージョン／アフィニティグループでは、「日本(東)」または「日本(西)」のどちらかを選ぶと日本データセンターに仮想マシンが作成されます。

すべて入力後、右下にある「仮想マシンの作成」をクリックして仮想マシンを作成します。準備ができると、一覧に作成した仮想マシンが**図3**の画面に表示されます。この状態で、sshクライアントから「azureuser」ユーザで仮想マシンへログインできます。

エンドポイントでHTTPへの接続を許可する

仮想マシンが作成された時点で接続が許可されているのは、SSH(TCPの22番ポート)のみです。HTTP(TCPの80番ポート)への接続許可の設定をしましょう。

管理ポータル画面左のメニューから「仮想マシン」をクリックすると仮想マシンの一覧が表示されるので、作成したホスト名をクリックします。ホスト名をクリックすると、作成した仮想マシンの設定が行えます。ホスト名の下にあるメニューの中の「エンドポイント」をクリックすると、設定の一覧が表示されます(**図4**)。

デフォルトで許可されているSSHが表示されています。HTTP接続を許可するには、下にある「追加」をクリックします。エンドポイントの追加で、「スタンドアロン エンドポイントの追加」を選択して右下の「→」をクリックします。

次の画面で、名前をクリックすると一覧が表示されますので、その中から「HTTP」を選ぶとプロトコルとポートが自動で入力されます。右下のチェックをクリックするとエンドポイントが設定されて、HTTP接続が許可されます。HTTPSやほかのポートを許可する場合も、同様に設定を行います。

UbuntuでLAMP

AzureでのUbuntu

Azureで提供されているUbuntuは、2014年9月現在「12.04 LTS」と「14.04 LTS」です。これらのイメージにはAzureで利用するためのツールがあらかじめ導入されており、インストール後の仮想マシンは実機と同様に使うことができます。

▼図3　仮想マシン一覧

▼図4　エンドポイント設定画面

最低限のセキュリティ対策をする

セキュリティ対策のためにapt-getコマンドでパッチを当てます(図5)。iptablesはデフォルトでoffになっているので、必要に応じて設定します。

Apache2、PHP5、MySQLを導入する

Apache2、PHP5、MySQLをapt-getコマンドで導入します(図6)。

sysv-rc-confコマンドで自動起動設定を行い、serviceコマンドでhttpdとmysqldを起動します(図7)。これでWebブラウザからHTTPが動作していることが確認できます。

次に、PHPが動作していることを確認するために、phpinfoの情報を表示するプログラム(リスト1)を/var/www/html/以下に保存します。Webブラウザをリロードすると、phpinfoの出力が確認できます。

最後にMySQLを使ったPHPプログラムのサンプルで、LAMP環境が正しく構築できていることを確認します。MySQLにテーブルを作るクエリ(リスト2)を図8のようにコマンドで実行します。MySQLへ値を読み書きするプログラム(リスト3)を/var/www/html/以下に置いてWebブラウザから確認します。動作していることが確認できれば、LAMP環境の設定は完了です。

▼図5　最新のパッチを当てる

```
$ sudo apt-get upgrade
```

▼図6　必要なパッケージの導入

```
$ sudo apt-get install apache2 php5 php5-mysql
$ sudo apt-get install mysql-server
```

▼図7　自動起動設定とdaemon起動

```
$ sudo apt-get install sysv-rc-conf
$ sudo sysv-rc-conf apache2 on
$ sudo sysv-rc-conf mysql on
$ sudo service apache2 start
```

▼図8　データベースとテーブルの作成

```
$ mysqladmin -u root create test1 -h localhost -p
$ mysql -u root -p test1 < sample.sql
```

より簡単にLAMP環境を実現

Azureが提供する機能を使ってみる

LAMP環境を運用するにあたって、サーバのメンテナンスや負荷が増えた場合のスケールアップ／スケールアウトなど課題がいくつもあります。Webデザイナーやコンテンツ制作者には敷居の高い作業が多いですが、AzureにはPHPアプリケーションを簡単に動かす「Azure Web Sites」という機能があります。

Azure Web Sitesによるサイト運用

Azure Web Sitesは、サーバを意識せずにWebアプリケーションを動かせるPaaSです。利用者はOSのバージョンやパッチ適用などの作業を意識することなくアプリケーションを運用できます。対応する言語は、ASP.NET、Java、PHP、Node.js、Pythonです。また、管理ポータルから、高負荷時の自動スケールアウトなどの設定が簡単に行えます。

アプリケーションはGitやFTPで直接デプロイしたり、GitHubやBitbucketなどのリポジトリを利用して管理が可能です。Azureが提供するデータベースやストレージなどの機能と連

▼リスト1　phpinfoを表示するプログラム(info.php)

```
<?php
phpinfo();
```

▼リスト2　テーブルを作成するクエリ(sample.sql)

```
create table test (
  id int not null auto_increment,
  name varchar(32) not null,
  primary key(id)
);
```

▼リスト3　データベースへの読み書きを行うプログラム（lamptest.php）

```
<?php
$mysqli = new mysqli("localhost", "root", "設定した管理者パスワード", "test1");

/* 接続確認 */
if ($mysqli->connect_errno) {
  echo "Connect failed<br/> n";
  exit();
}

/* DBへ値を保存 */
if ($mysqli->query("insert into test(name) values('" . uniqid() . "')") === FALSE) {
  echo "insert error<br/> n";
}

/* DBから読込み */
if ($result = $mysqli->query("select id,name from test")) {
  while($object = $result->fetch_object()) {
    printf("%s : %s<br/> n", $object->id, $object->name);
  }
  $result->close();
}

/* 接続解除 */
$mysqli->close();
```

携して、Webアプリケーションをほぼインフラを意識せずに構築運用できます。また、小さな規模のサイトであれば10サイトまで無料で利用できるところも利点です。

Azure Web Sitesを使う

Azure Web Sitesを使うには、管理ポータルより左下の「＋新規」をクリックして「WEBサイト」を作成します（**図9**）。「コンピューティング」−「WEBサイト」−「簡易作成」とクリックし、URL、WEBホスティングプラン、リージョンを入力して右下の「WEBサイトの作成」で作成されます。WEBサイト作成が完了したら左の「WEBサイト」をクリックすると、一覧に作成したWEBサイトが追加されます。

WEBサイトが追加されたら、Webアプリケーションをデプロイするためのユーザの設定を行います。設定したいWEBサイトをクリックして、「ダッシュボード」に移動します（**図10**）。右にある「概要」の下にある「デプロイ資格情報のリセット」をクリックします。ユーザ名とパスワードを設定すると、その情報でGitやFTPでログインできるようになります。

▼図9　WEBサイトの新規作成

次に、ソース管理の設定を行います。今回は、Azure上のGitで管理することにします。**図11**の下の「ソース管理からのデプロイの設定」をクリックします。デプロイ設定で「ローカルGitリポジトリ」を選択します。これでGitでソースの管理ができます。

WEBサイト名の下にあるメニューの「デプロイ」をクリックし、表示されているGitのURLをコピーして、開発環境にGitの設定およびサイトの初期化を行います(**図12**)。そして、phpinfoを表示するプログラム(**リスト1**)をデプロイします。メニューから「ダッシュボード」に切り替え、右側にあるサイトのURLをクリックすると、phpinfoによるPHPの情報が表示されます。このように、比較的簡単にWebアプリケーションを動作させることができます。

まとめ

この章ではAzureでLAMP環境を構築する方法を解説しましたが、実機やほかのクラウドと同様、簡単に構築できます。

また、Webアプリケーションを簡単に動かすAzure Web Sitesについても説明しましたが、手軽にLAMPベースのアプリケーションを試すことができますので、開発環境としての利用などに適しているのではないでしょうか。

Azureは今後も新しい機能が提供されて進化していきますので、公式サイトをはじめとした情報をチェックしてもらえればと思います。SD

▼図10　ダッシュボード

▼図11　作成されたWEBサイト

▼図12　クライアントのGit設定(ホスト名がgihyowebの場合)

```
$ mkdir gihyoweb
$ cd gihyoweb
$ git init
$ vi index.php
$ git add .
$ git commit -m "init"
$ git remote add azure https://Gitのユーザ名@gihyoweb.scm.azurewebsites.net:443/gihyoweb.git ←ここにコピーしたGitのURLを入れる
$ git push azure master
Password for 'https://Gitのユーザ名@gihyoweb.scm.azurewebsites.net:443':
↑デプロイ資格のパスワードを聞かれます
```

公式サイト

Microsoft Azure トップページ	http://azure.microsoft.com/
Microsoft Azure Blog	http://azure.microsoft.com/blog/
Microsoft Azure Japan Team Blog	http://blogs.msdn.com/b/windowsazurej/

第7章 安定にあぐらをかくことなかれ! 進化を続ける仮想化技術とインフラのアーキテクチャ

Webサービス構築において今日でも代表的な構成である「LAMP」ですが、時代の変遷とともに、そのアーキテクチャやインフラストラクチャも進化し続けています。新しいLinuxディストリビューション、サーバ、DBMS、プログラミング言語の登場から、仮想化基盤、クラウドサービスの利用まで、大きな変化の流れを振り返ります。

Writer 並河 祐貴(なみかわ ゆうき) Twitter @namikawa URL http://d.hatena.ne.jp/rx7/

LAMP環境の変遷

「LAMP」という言葉が誕生したのは、Wikipediaの記事[注1]によると1998年とのことで、今から15年以上前となります。おさらいしておくと、LAMPはオープンソースである「Linux+Apache+MySQL+Perl/PHP/Python」を使ったWebシステムのスタックを表す言葉になりますが、この数年でWebシステムのアーキテクチャのトレンドもさまざまな進化を遂げています。

筆者は、2000年ごろに大学の授業でたまたまLAMPを使ったシステム構築演習をする機会があり、Linux+Apache+MySQL+PHPを使って簡易的な在庫管理システムを作った記憶があります。内容はあまり覚えていない(笑)のですが、教科書のサンプルコードを参考にしながら実装し、「Webサービスは簡単なものなら意外と手軽に作れるな」と感じた記憶があります。当時は、まだ学生だったので多くのシステムを見る機会はなかったのですが、お手軽なWebシステムといえば、CGI/PerlやPHPが多かったのではないでしょうか。Javaサーブレット/JavaServer Pages(JSP)などもあったかもしれません。

それから10年以上が経ち、今や世の中には、そのころとは比較にならないくらい多くのWebサイトが存在します。Netcraft社の調べ[注2]によると、2000年1月時点で約993万だったWebサイト(hostnameベース)が、2014年8月時点では約9億9200万サイトと、約100倍の数にもなります。それに伴ってWebサイトの実装方法も進化を続けてきました。その中でLAMPと呼ばれた主流スタイルも、大きく変化しつつあります。

Linux OSこそ大きくは変わりませんが、そのディストリビューションもRed HatやDebianだけではなく、多くの選択肢が増えました。Web/APサーバ部分も、Apacheのシェアは年々下がってきており、かつてインターネットの全サイトの70%を占めていたApacheのシェアは、2014年8月時点で35%と半分になっており、Nginxなどのより軽量かつハイパフォーマンスなWebサーバがシェアを伸ばしています。

データベース部分も従来のオープンソースRDBMSであるMySQLはもちろん、PostgreSQLも変わらず進化を続け、加えて、高いスケーラビリティ(分散システム型)の確保やスキーマレスで柔軟に変更が加えられる(ドキュメント指向)ような、Key-Value Store／NoSQL型のデータベースも多く使われるようになってきました。

最後にプログラミング言語ですが、近年多くの言語でWebアプリケーションフレームワークの実装が進んでいます。中でも2005～2006年あたりでブレイクしたRuby on Railsは本当に多くのWebサービスの実装として今日も利用されています。こういったLAMPの変遷の中で利用されているプロダクトは書き始めるとキリがないので、その一部を**表1**に記しておきます。

注1) URL http://ja.wikipedia.org/wiki/LAMP
注2) URL http://news.netcraft.com/archives/2014/08/27/august-2014-web-server-survey.html

▼表1　Webシステムスタックの変遷

	過去	近年
OS	Linux（Red Hat、Debian）	Linux（Red Hat Enterprise Linux/CentOS/Fedora、Debian/Ubuntu、Gentoo、CoreOSなど）
Web／APサーバ	Apache	Apache、Nginx、lighttpdなど
データベース	MySQL、PostgreSQL	MySQL、PostgreSQL、MariaDB、MongoDB、CouchDB、Cassandra、HBaseなど
言語	Perl、PHP	Perl、PHP、Python、Ruby、node.js、Java、Scala、Golangなど

LAMP環境のインフラストラクチャと構築手法の変遷

LAMP……とは少し話がズレていきますが、Webシステムを支えるインフラストラクチャについても、時代の進化とともにさまざまなパターンが出てきています。また、そのパターンの進化にあわせて、環境構築の手法も多様化してきています。

インターネットサービスが増えることで、人々の生活に対してより便利なサービスが提供され、その結果インターネットトラフィックが大きくなり、多くのサーバリソースが必要となります。サーバが増えるとセットアップ作業量も増えることになりますが、そこがどのように変わっていったのかを見ていきましょう。

オンプレミスな物理環境

LAMPという用語が誕生した1998～2000年ごろからはIA（Intel Architecture）サーバが多く普及してきた時代で、比較的安価なIAサーバを並べ、水平分散させる形でスケーラビリティを確保していました。今でいうオンプレミスな環境での物理サーバが主流で、LAMP自体のセットアップについても、手動で行うこと自体は普通の話でした。セットアップを効率化する手段として、Linux OSのインストールやミドルウェアの初期セットアップについてはRed Hat系ではkickstartが、Debian系ではpreseedなどのツールを利用して自動化が進められてきました。コンフィギュレーションツールについては、CFEngineなどが古くからありましたが、日本ではそれほど普及していなかったように思います。このころは、自作のシェルスクリプトなどがセットアップで多く利用されていたのではないでしょうか。

サーバ仮想化によるリソース集約

2004～2006年ごろにかけて、さらなるハードウェアリソースの進歩やコスト減に伴い、リソースの有効活用を目的としてサーバ仮想化が注目され始めます。サーバ仮想化を活用することで、次のようなメリットが生まれます。

・サーバリソースの有効活用
　→サーバの集約が可能
・仮想マシン実行環境の標準化
　→物理マシンが変わっても仮想マシンには影響しない＝高いポータビリティを実現
・OS+アプリケーションをコンテナ化できる
　→仮想マシンを簡単にコピーしたり、別の物理マシンにマイグレーションできる

仮想マシンにLAMP環境（Webシステム）を構築することで、最初に仮想マシンというコンテナを作ってしまえば、仮想マシンのイメージをコピーして増殖させていくことで、同じ環境を作るのに何度もセットアップせずに済むため、簡単にスケールアウトを実現できます。また、仮想化することでライブマイグレーションと呼ばれるような、オンラインのまま仮想マシンを別物理マシンに移動できるポータビリティが実現され、より物理マシンのメンテナンスが行いやすくなりました。また2007年には、Linux Kernel に Kernel-based Virtual Machine（KVM）が組み込まれることになり、Linux OSでの仮想環境の運用がより身近なものになっていきます。

クラウドサービスの台頭

今やパブリッククラウドサービスの代表的な存在であるAmazon Web Services。Amazon Elastic Compute Cloud(EC2)とAmazon Simple Storage Service(S3)が2006年にリリースされ、2007～2009年にかけて世界中で大きく普及していきました。とくにB to C向けのWebサービスは、流行り廃(すた)りが早いことが多く、ヒットした際のトラフィックの跳ね上がり方が大きいこともあり、「すぐにリソースを調達できる(借りられる)こと」と「すぐに捨てられる(資産・在庫を持たない)こと」といったクラウドサービスの特徴との相性は抜群です。

また、クラウドサービスをコントロールするインターフェースとしてAPIが準備されており、さまざまなシステムリソースを制御できることから、「Infrastructure as Code」と呼ばれるような、インフラリソースをプログラマブルに扱う潮流が強くなりました。

クラウドサービスによっては、オートスケーリングと呼ばれる、設定した負荷量の閾値(いきち)をトリガーにして、サーバリソースの増減を自動的に行ってくれるしくみがあることから、クラウドサービスの普及に伴い、OSやミドルウェアのセットアップの効率性および自動化がより強く求められるようになってきました。

クラウドサービスも、基本的なアーキテクチャや概念はサーバ仮想化と同様であり、それと同じメリットが享受できます。そのため、仮想マシンのイメージを利用することで、スケールアウトの一環で同じ役割のサーバを増殖できるしくみがあります。前述の「Infrastructure as Code」を推す流れから、LAMPスタックのようなOSやミドルウェアの運用管理については、ChefやPuppet、Ansibleといったコンフィギュレーションツールが使われるようになりました。

また、クラウドサービスの普及に伴い、それらが提供するAPIと同じような形式をプライベートクラウドにも実現できるEucalyptusやOpenStackといったクラウド運用管理ソフトウェアも出現してきました。

Dockerの出現

「Docker」の詳細については本特集の別章で語られているため省略しますが、Dockerは、コンテナ化による高いポータビリティと、コンテナの構成管理(ミドルウェアのインストールやその設定)をコードで記述することができるため、最近のITトレンドである「Immutable Infrastructure」と「Infrastructure as Code」の両方を実現しやすいアーキテクチャと言えるでしょう。

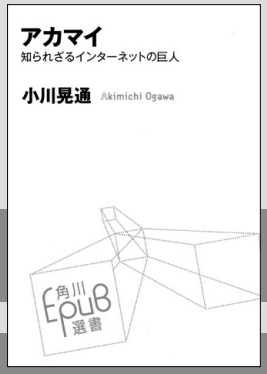
各クラウドサービスが、このDockerへの対応を進めており、これからさまざまなWebサービスの本番環境でも使われていくのではないでしょうか。LAMPスタックのレイヤの管理については、Dockerfileと呼ばれる設定ファイルに記載することがメインの手段となります。

おわりに

本章は、LAMPに特化した内容というよりは、LAMPスタックのようなOS・ミドルウェアレイヤの運用管理の変遷について簡単に紹介しました。LAMPといったくくりが登場して十数年以上経っていますが、当時のスタンダードなLAMP構成がそのまま利用されているシステムもあれば、そこから派生したWebシステムスタックもあるでしょう。また、それらを構成するインフラストラクチャも、アーキテクチャが変化していますし、これからも変化し続けるはずです。私たちエンジニアはそれらのトレンドを追い続け、目の前のWebサービス運用をどう改善させていくのかを、常に意識し続けることが大事です。SD

BOOK no.1 アカマイ
知られざるインターネットの巨人

小川 晃通【著】
B6判、225ページ／価格＝1,500円＋税／発行＝KADOKAWA
ISBN＝978-04-080017-2



本誌でも、過去何度かにわたって特集記事などで取り上げてきたアカマイ・テクノロジーズを、一般読者向けに解説する冒険をしたのが本書である。筆者は皆さんおなじみの、「あきみち」こと小川 晃道さん（Geekなページ：http://www.geekpage.jp/）。

インターネットを陰で支える巨人アカマイ。そもそもContents Delivery Networkのしくみを非技術者に向けて説明するのは至難の技だ。それはインターネットの幻想を打ち砕くことから始めねばならないからだ。アカマイのしくみを解き明かした結果、ネットワーク技術の良質な教科書という内容を持つ本になっている。新書の枠組みを超える力作。

BOOK no.2 Amazon Web Services入門
企業システムへの導入障壁を徹底解消

加藤 章【著】
A5判、184ページ／価格＝2,300円＋税／発行＝インプレス
ISBN＝978-4-8443-3647-1

自社システム（オンプレミス）のAWS移行を検討するとき、誰もが疑問に思うことがある。「データ漏洩が起きたら誰の責任？」「365日間システム無停止は可能？」などの疑問だ。本書は、AWSパートナー企業に勤める著者が、AWSの約款を示しつつ、これらの疑問に答えてくれる。ユーザには受け入れがたい約款があったとしても、安心してAWSを利用するための解決策を提示する。「可用性を上げるためにはAmazon Machine Imageやアベイラビリティゾーンをうまく活用する」「暗号化を用いてデータ漏洩対策を施す」などはその一例だ。本書は、AWSの理解を深めるだけでなく、AWS導入を人に説明するときの材料としても役に立つだろう。

BOOK no.3 Rubyによるクローラー開発技法

佐々木 拓郎、るびきち【著】
A5判、448ページ／価格＝2,980円＋税／発行＝SBクリエイティブ
ISBN＝978-4-7973-8035-4

自動的にWebサイトを巡回し情報収集する「クローラー」をRubyで開発する手法を解説する一冊。クローラーの概要から、Anemoneなどのライブラリを使った実装方法まで詳しく説明している。Rubyの基本や正規表現の記法についても説明があるので、Ruby初心者にも優しい。

第5章では、TwitterやAmazon.comといった具体的なサービスごとのクローラーの実装がサンプルスクリプトとともに解説されており、クローラーを自作するときに大いに参考にできる。

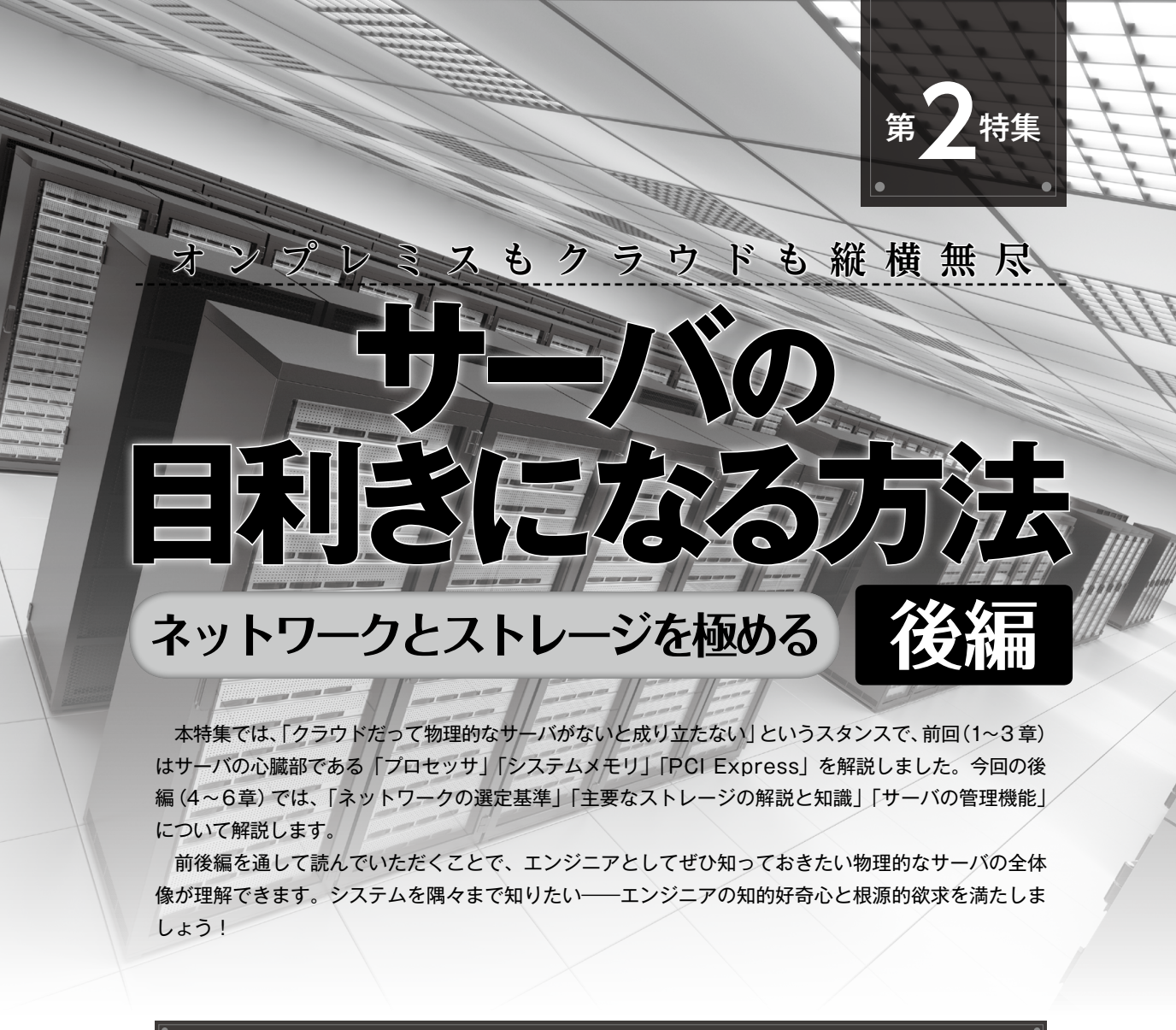
また、サイトの利用規約、robots.txt、著作権の扱いといった、クローリングにおいて守るべきルールについても詳しい説明があり、実際にクローラーを開発する際はぜひ確認してほしい。

BOOK no.4 内部構造から学ぶPostgreSQL
設計・運用計画の鉄則

勝俣 智成、佐伯 昌樹、原田 登志【著】
A5判、288ページ／価格＝3,300円＋税／発行＝技術評論社
ISBN＝978-4-7741-6709-1

オープンソースのデータベース「PostgreSQL」の解説書。本格的にPostgreSQLの運用管理や技術力の向上を図りたい中・上級者向けの本だが、基本設定などの基礎的な内容も充実している。

本書は基本編、設計／計画編、運用編、チューニング編に分かれているが、どのパートもPostgreSQLのアーキテクチャ・内部での挙動に触れながら解説しており、「実際に内部で何が起きているか」という深い視点から学べる。パート4では、現場への投入後に起きる性能の低下について、問題の切り分け方、実行計画の解析方法、チューニングの仕方が書かれているので、設計後のメンテナンスに利用してほしい。



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

ネットワークとストレージを極める 後編

本特集では、「クラウドだって物理的なサーバがないと成り立たない」というスタンスで、前回（1～3章）はサーバの心臓部である「プロセッサ」「システムメモリ」「PCI Express」を解説しました。今回の後編（4～6章）では、「ネットワークの選定基準」「主要なストレージの解説と知識」「サーバの管理機能」について解説します。

前後編を通して読んでいただくことで、エンジニアとしてぜひ知っておきたい物理的なサーバの全体像が理解できます。システムを隅々まで知りたい――エンジニアの知的好奇心と根源的欲求を満たしましょう！

CONTENTS

Writer 長谷川 猛（はせがわ たけし）

第4章 ネットワークの変遷と選定基準

Writer 長谷川 猛(はせがわ たけし)／Twitter @hasegaw

私たちの手もとにあるコンピュータやスマートフォン、タブレットなどの端末は、ネットワークに接続して利用しています。本章では、現在、サーバ間やクライアントーサーバ間の通信だけでなく、データセンターから家庭内まで広く使われているネットワーク技術「イーサネット(Ethernet)」を中心に取り上げます。

イーサネットの規格と歴史

サーバの多くはデータセンターに設置され、利用者は遠隔地からネットワークを介してサーバにアクセスします。このため、サーバのネットワークインターフェースは私たちがサーバに何かを要求したり、データを送受信したりするためには欠かすことができない、重要な役割を持ちます。最近では手もとのデバイス類——スマートフォン、タブレット、パソコンに限らず時計や自動車などあらゆるもの——をネットワークで接続し、情報を集めて活用しようという試みもあり、モノとネットワークをより密に接続しようという風潮は、ますます高まっています。

現在のイーサネット規格と言えば100Mbps、1Gbps、10Gbpsのものが使われることが多いのですが、そのルーツは1970年のALOHAnetまで遡(さかのぼ)ります。今回は、現在利用されているイーサネットと、その仕様の背景となっている10BASE-5以降のインターフェース(**表1**)について説明します。

同軸ケーブル時代(10BASE-5、10BASE-2)

この業界に長年いらっしゃる方であればご存じだと思いますが、本誌が創刊される前、すなわち1980年代は、同軸ケーブルを使ったイーサネット、10BASE-5および10BASE-2が使われていました[注1]。これらの規格では、1本の同軸ケーブルを用意し、そのケーブルをノード間でシェアする「バス型」のネットワークを構成します(**図1**)。コンピュータを接続するときは、まずは目的の場所でヴァンパイアタップもしくはT字型のBNCコネクタ(**写真1**)をはさんで、コンピュータをネットワークにつなぎます。当初の10BASE-5では1つのセグメントを最大500メートル、そのあとの10BASE-2では取り回しの良い、細い同軸ケーブルを使い、最大185メートル(200メートル)まで伸ばすことが可能です。

また、10BASE-5/2のバス型ネットワークでは、バスとなる同軸ケーブルを複数のホストが共有するしくみです。このため、ある

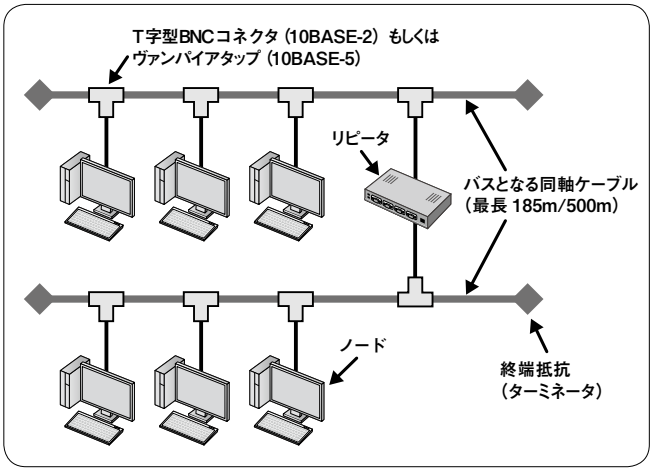
▼表1 イーサネット規格の一部

規格	通信速度	ケーブルの種類	規格策定時期
10BASE-5	10Mbps	同軸ケーブル	1983年
10BASE-2	10Mbps	同軸ケーブル	1985年
10BASE-T	10Mbps	UTPケーブル(カテゴリ4)	1990年
100BASE-T	100Mbps	UTPケーブル(カテゴリ5/5e)	1995年
1000BASE-T	1000Mbps	UTPケーブル(カテゴリ6)	1999年
10GBASE-T	10000Mbps	UTPケーブル(カテゴリ7)	2006年

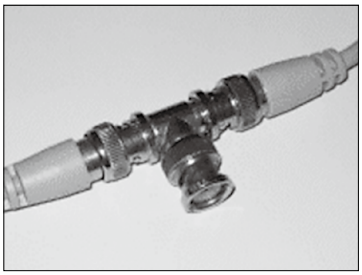
注1) 筆者が10BASE-2の実物を見たことがあるのは、当時大学生だった従兄弟の家で見たぐらいだったりします。でも、本誌の読者なら利用されていた方も多くいらっしゃるのでしょうね:)

ノード(ノードA)がフレーム(データ)を送信する際は、バスを文字どおり「占有」します。フレーム送信中は、ノードA自身がフレームを受信できないだけでなく、ほかのノードB、Cもフレームを送信できません。もしノードB、Cがほかのフレームを被せてしまえば、複数の信号が混ざってしまい、どちらの信号も受信できなくなり(**図2**)、お互いにジャム信号を出し合って衝突を知らせます(**図3**)。この際、一定時間、セグメント全体が麻痺します。この通信方式はCSMA/CD(Carrier Sense Multiple

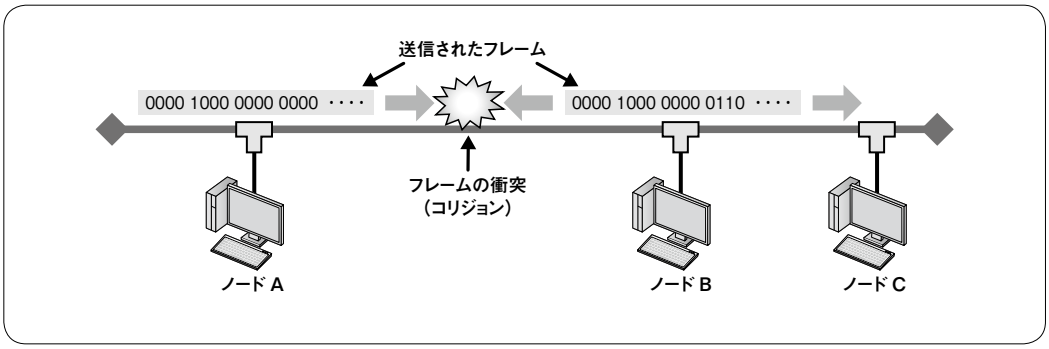
▼図1　10BASE-5、10BASE-2のバス型トポロジ



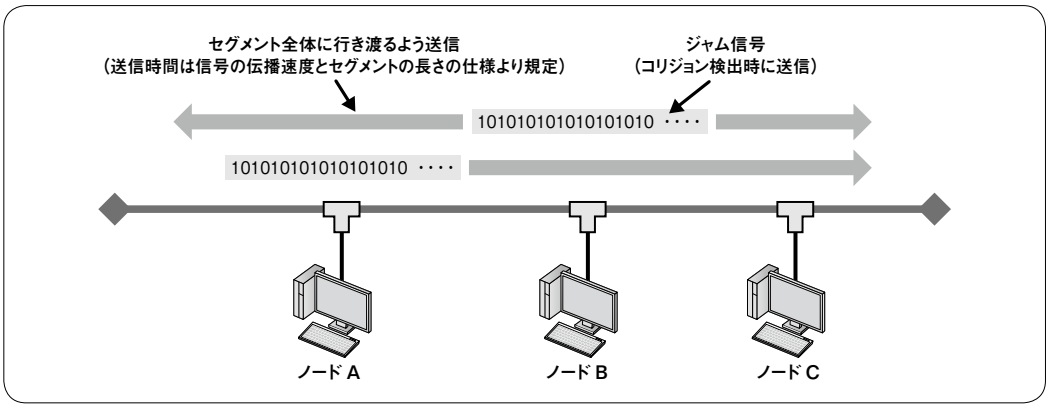
▼写真1　T字型のBNCコネクタ



▼図2　イーサネットフレームの送信(コリジョン発生)



▼図3　ジャム信号



Access with Collision Detection)と呼ばれます。

リピータ

同軸ケーブル上を流れる信号は距離に伴い減衰し、またノイズの影響を受けます。大きなセグメントで信号レベルが低いときには、受信した信号を増幅して再送出する「リピータ」と呼ばれる装置を利用すると伝達距離を伸ばすことが可能です。ただし、リピータは単にイーサネットフレームを再送出する機能しかないため、リピータを使ってもセグメントの総距離を伸ばせるわけではありません[注2]。

同軸ケーブル時代のイーサネットでは、現在のイーサネットでは見かけない課題がいくつかありました。まず、バスとなる同軸ケーブルの両端に終端抵抗(ターミネータ)を取り付ける必要がありました。終端抵抗が役割を果たさなければ、同軸ケーブルの端で信号の反射が起きて通信が成り立たなくなりますし、バスとなっている同軸ケーブルのどこかで断線が起きると、ネットワークが麻痺してしまいます。ノードの追加のためのBNCコネクタ取り付けでも通信断が生じますし、どこかで断線が起きれば、やはりセグメント全体に影響を及ぼすため、障害の影響範囲が大きく、かつ原因特定が難しいという課題がありました。

10BASE-Tの登場から今日まで

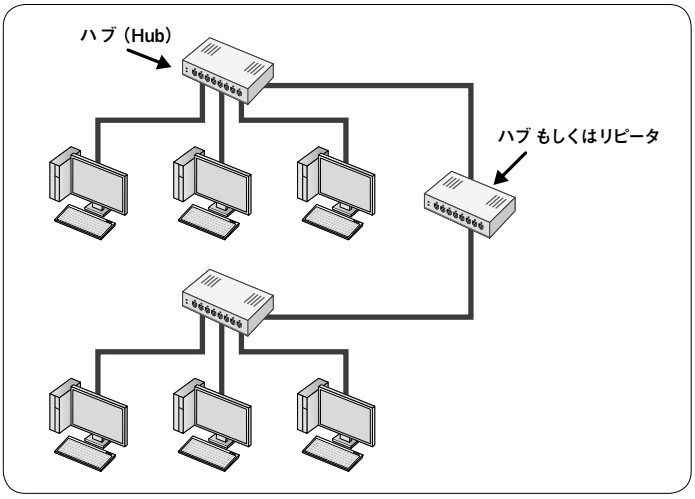
1990年に登場した10BASE-Tは、私たちが現在使っているイーサネット規格にほぼ近いものです。使用するケーブルは現在お店でイーサネット用に販売されているLANケーブル(端子はRJ45(**写真2**))が利用できます。

10BASE-Tでは、ノードとノード(もしくはハブ(Hub))を1対1で接続するスター型トポロジでネットワークを構成します。各ノードはハブを中心に接続されるため、スター型トポロジと呼ばれます(**図4**)。通信のしくみは10BASE-5と同様ですが、バス型トポロジとは違い、ダウンタイムなしでノードを増やすことも可能ですし、また、ケーブルの一部に断線が生じたとしても影響範囲は限られます。

10BASE-T以降のイーサネットはノードとハブ、そしてケーブルさえ準備すれば気軽に利用できる規格となり、さまざまなコンピュータやネットワーク機器の相互接続にも利用できることから、今日に至るまで、標準的に使われる

注2) コリジョンが発生した場合は、ジャム信号(1と0の繰り返し)でバスを埋め、衝突の発生を全ノードに伝えます。この信号がセグメント全体に行き渡るまでの所要時間などの理由から、セグメントの最大長などの仕様が決められています。

▼図4　10BASE-T以降のスター型トポロジ



▼写真2　RJ45コネクタ

ようになりました。

リピータハブとスイッチングハブ

スター型のトポロジでは、3台以上のノードを接続する場合にはハブ(もしくはスイッチ)が必須となります。10BASE-T時代のハブには、信号を増幅するリピータハブ、信号のバッファリングおよびフロー制御機能を持つスイッチングハブがありました。リピータハブは受信した信号をそのまま増幅し、ほかのポートに送り出します。一方、スイッチングハブは各ノードのMACアドレスを学習し、送信されたフレームをいったんバッファリングしたあと、目的のノードが接続されているポートにコリジョンを起こさないよう送信する、という違いがあります。

半二重と全二重

バス型のトポロジを起源とするイーサネットでは、データの送信元が1つしか許されない半二重(Half duplex)通信でした。しかし、スイッチングハブの登場によりイーサネットの接続は1対1となり、受信と送信には別の信号線を使うため、電気的には送信と受信が同時に可能になりました。全二重(Full duplex)通信に対応したLANアダプタ、ケーブル、スイッチングハブを組み合わせ、全二重でリンクすれば、送信と受信を同時に行え、コリジョンも発生しません。半二重通信モードでデバイスが動作していれば、送信と受信の信号線が同時に利用された場合にはコリジョン扱いとなり、フレームを再送信します。

100BASE-T(TX)、オートネゴシエーション

100Mbpsの転送が可能な100BASE-TX[注3]の時代になると、入手可能なハブはほぼスイッチング方式のものとなり、昔ながらのリピータハブは見かけなくなりました。また、ほとんどのLANアダプタなどが全二重通信に対応しました。また、相手側のイーサネット規格を検出してリンク方式を切り替えるオートネゴシエーション機能が登場し、10BASE-Tと100BASE-T(TX)以降はシームレスな相互運用が可能です。

1000BASE-T(Gigabit Ethernet)

1Gbpsの転送が可能な1000BASE-T[注4]は、現在、サーバやパソコンなどで広く使われる規格です。秒間およそ100MB/sの転送速度は、今日、インターネットと手もとのパソコンをつなぐには十分な帯域幅ですが、コンピュータのデータ処理性能やHDD/SSDの帯域幅から考えると、いくらか見劣りするスピードとも言えます。実際、現在のコンピュータネットワークの標準でありながら、同時に通信ボトルネックの要因にもなっています。

10Gb Ethernet

最近では、10Gbpsの通信が可能なイーサネット[注5]をサーバに搭載できる時代となっています。従来の10倍の帯域を持つ10GbEは、映像配信など帯域幅が求められるワークロードを効率的にさばいたり、クラウド環境を構築したりするためには欠かせないインターフェースです。まだネットワーク機器などのポート単価がネックとなる場合もありますが、必要性が認められるなら、リンク1本で1GbEの10本分近い帯域幅が得られる有用な選択肢です。

10GBASE-Tでは完全に全二重通信となり、CSMA/CDが仕様から撤廃され、ついにコリジョンフリーとなりました。レイテンシーの観点でも、1000BASE-Tではパケットあたり最大12マイクロ秒の伝送時間を要しますが、10GBASE-Tは2～4マイクロ秒に短縮されており、伝送時間や往復通信の待ち時間を1/3以下に抑えられます。

このため、同期I/Oとなりがちなワークロード(リモートノードの半導体ストレージに対するI/O、並列度が得られないバッチクエリなど)

注3) 100Mbpsのイーサネット規格はファーストイーサネット(Fast Ethernet)などと呼ばれます。

注4) 1Gbpsのイーサネットはギガビットイーサネット(GbE)と表現されたりもします。

注5) 10ギガイーサネット(10GbE)などと略して表記されます。

には、帯域幅以外の観点からも10Gbps接続を検討する価値があります。

カッパーと光ファイバー

ここまで、カッパー(Copper)、すなわち銅線を使ったケーブリングを行うイーサネット規格(-T)を軸に説明しました。これらの規格に加え、光ファイバーを利用するイーサネット規格も広く使われています(**表2**)。

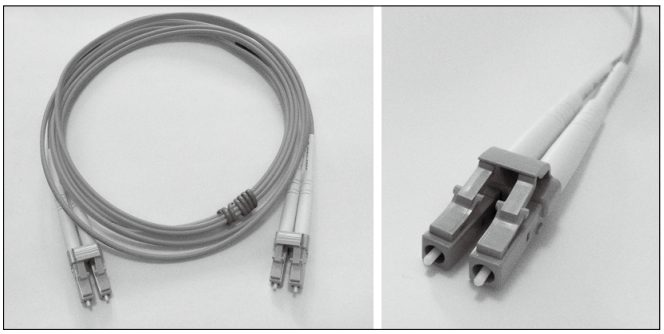
電気信号の減衰が大きいカッパーのイーサネットでは100メートルが仕様上の最長距離ですが、光ファイバーであればより長い距離の通信が可能です。このため、広域イーサネットやコアルータ－エッジルータ間など、距離が長くなりがちな場所では光ファイバーのほうが適している、と言えます。

光ファイバー(**写真3**)を使ったネットワークを構築するにはSFPモジュールと呼ばれるトランシーバが必要です。それによりカッパーと比べポートあたりの単価が高くなるため、サーバを含め、エッジノードの接続ではカッパーが広く使われています。ただ、最新の10GBASE-Tではカッパーのトランシーバが高価、かつ消費電力も大きくなり、光ファイバーの4～5倍の電力量を必要とします(**表3**)。このため、10GbE接続では、ツインナックスケーブル(**写真4**)のイーサネット規格が使われる傾向にあります。

▼表2　光ファイバー接続のイーサネット規格(一部、短距離向け)

規格	メディアの種類	最大伝送距離
1000BASE-SX	マルチモード 光ファイバー	～550m
1000BASE-LX	マルチモード 光ファイバー	550m
	シングルモード 光ファイバー	5Km
10GBASE-SR	マルチモード 光ファイバー	～300m
10GBASE-LR	シングルモード 光ファイバー	10Km

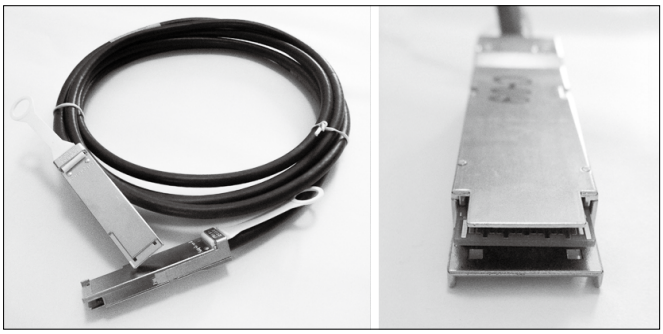
▼写真3　光ファイバーケーブル



▼表3　SFP+とカッパーの比較(10GBASE-T)[注6]

	SFP+	10GBASE-T
ポートあたりの消費電力	0.7W	3.5～5W
ケーブル最大長	300m	100m
遅延	0.3us	2～2.5us
クロストークの影響	なし	影響あり

▼写真4　ツインナックスケーブル



注6) この表は次のWebサイトを参考に筆者が作成。
http://www.datacenterknowledge.com/archives/2012/11/27/data-center-infrastructure-benefits-of-deploying-sfp-fiber-vs-10gbase-t/

NICを選ぶときのポイント

イーサネットによってサーバをネットワークに接続するためには、NIC(Network Interface Controller)と呼ばれるコンポーネントを使用します。ここではNICの選定を行う際のポイントをいくつか紹介します。

標準搭載NICと追加NIC

最近のサーバでは、イーサネットが2～4ポートほど標準装備されています。サービス用に2ポート、マネジメント用に2ポート利用する程度で済めば、サーバ標準のポートのみで足りることも多いでしょう。仮想マシンの実行環境として利用する場合には、さらにiSCSIストレージとの通信用ポート、仮想マシンモニタ間の通信用ポートなどを見積もることになります。

多数のセグメントに接続する場合やリンクアグリゲーション(冗長化や広帯域化)を設定する場合には、より多くのイーサネットポートが必要になります。この場合にはPCIeスロットにNICを搭載し、イーサネットポート数を増やせます(**写真5**)。PCIeスロットあたり、1/2/4ポート程度増設可能です。

現在はGbEから10GbEへの移行の過渡期ということもあり、使用されるイーサネット規格の多様化に対応するため、イーサネットコントローラ部分が交換可能なモジュール式となっているサーバもあります。従来は、10GbEを搭載したければPCIeスロットにカードを追加する手段が現実解でしたが、イーサネットコントローラ部分を独立交換できるモデルならば、サーバ標準のNICポートを10GbE化できます。

そのNICはサポートされていますか?

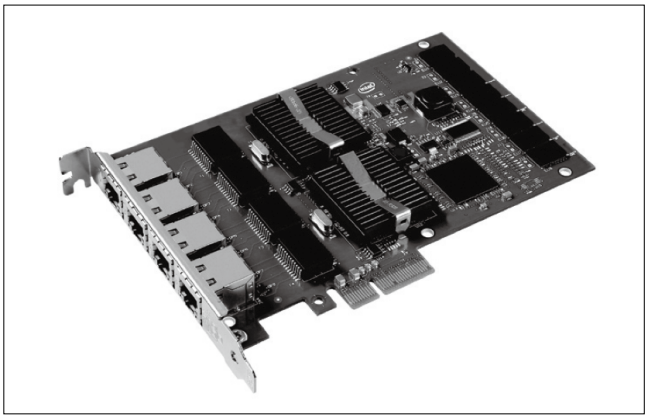
NICに限ったことでもないのですが、サーバベンダがサポートするNIC、OSの特定バージョンがサポートするNICなどには差異があります。Windowsの場合はドライバの互換性が高く「ドライバ提供なし」という事態に陥ることはまずないかと思いますが、Linuxなどでは採用予定のOSの具体的なマイナーバージョンで最新NICチップのサポートがない、なんてことはあり得る話です。

サーバベンダが純正パーツとして提供するNICであれば、そのサーバでサポートするOS向けのNICドライバは何らかの形で提供されているかと思います。または、ドライバのソースコードに1行追加し、カーネルモジュールを1つビルドすれば使えるかもしれません。その場合、トラブルが発生した際にはどこがサポートしてくれるのか、それとも自身で闘う必要があるのかは確認しておきましょう。

リンクアグリゲーション

複数のリンクを統合(アグリゲート)して1つの論理的な回線を作る手法の総称をリンクアグリゲーションと呼びます。たとえば、複数のリンクを束ねてサーバ－スイッチ間を接続し、仮にどちらかのリンクに障害が生じてもサービスを継続できるよう構成できます(Bondingもしくは Teamingと呼ばれる)。また、複数のリンクにパケットを分散させ、ネットワーク帯域を増す

▼写真5　PCIe NIC



目的でも利用されます[注7]。

NIC障害を意識したポート割り当て

もしPCIeカードに故障や一時障害が発生した場合、複数のリンクに影響が出る可能性があります。リンクアグリゲーションを構成する際には、同じサーバ上のイーサネットポートでも、できればお互いに独立したポートを組み合わせることが望ましいと言えます(**図5**)。

NICチップの組み合わせに注意

イーサネットポートの先には、ネットワークコントローラチップが存在します。もしコントローラチップが違えば、使用するデバイスドライバが異なる可能性があります。また、コントローラチップが異なると、リンクアグリゲーションを構成できない、といった制約が生じる場合があります(OSや仮想マシンモニタ、ドライバの実装によります)。

サーバの標準搭載ポートでも、オプションパーツとして提供されるPCIeカードでも、搭載されているネットワークコントローラのモデルは、カタログや構成ガイド資料などに記載がある場合がほとんどです。とくにリンクアグリゲーションを想定する場合には、コントローラチップのモデルおよび想定した構成をとれるか確認しておくと良いでしょう。

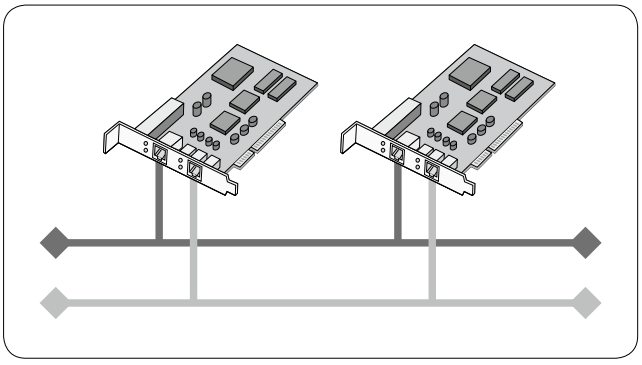
オートネゴシエーションによるトラブル

これは選定時の注意事項ではないのですが、サーバを構築する際にはNICのオートネゴシエーション機能をオフにして、リンク速度と通信方式(例:1000Mbps、全二重)を手動設定することをお勧めします。

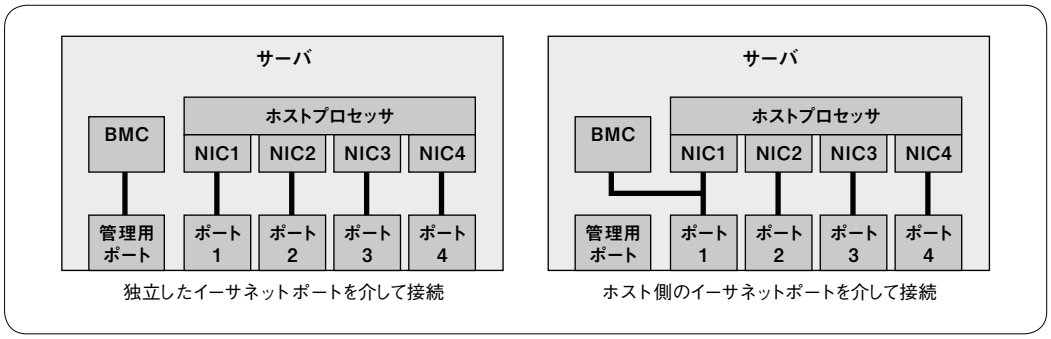
イーサネットはオートネゴシエーション機能があり、お互いの通信速度、および全二重/半二重を調停することが可能です。しかし、オートネゴシエーション機能がうまく働かないと、イーサネットのスループットが十分に得られない問題が発生することがあります。サーバ-スイッチ間のオートネゴシエーション失敗はそれほど見かけない気もしますが、ハマらないためには通信方式を手動設定します。

注7) 本誌2014年9月号、10月号の「[実力検証]NICをまとめて高速通信!」を参照。

▼図5 NIC障害を意識した冗長化



▼図6 アウトバンド管理用ポートの構成パターン



アウトバンド管理用ポート

あとの章で紹介しますが、一定クラス以上のサーバでは、ホストプロセッサとは独立した管理プロセッサ(BMC)が搭載されています。管理プロセッサとの通信にもやはりイーサネットが使われます。

管理プロセッサの通信ポートは、管理プロセッサ専用のイーサネットポートを介して接続する方法、およびホスト側のイーサネットポートに相乗りする方法があります(**図6**)。管理プロセッサへのアクセスはサーバへの物理的アクセスにも匹敵するインパクトがあるため、サービス用ネットワークとは分離された管理用ネットワークに接続するほうが無難です。

その他の注目技術

現在はIEEE803.3準拠のイーサネットが一般的に使われますが、x86サーバで利用できるインターコネクト技術としてInfiniBandもあります。

InfiniBand

イーサネットの世界であれば、現在見かけるのは1Gbpsないしは10Gbpsですが、InfiniBandの場合は2000年に策定されたSDR(Single Data Rate)のInfiniBandで10Gbps、現世代のFDRであれば55Gbpsという帯域幅を持ちます。RDMA[注8]と呼ばれる方法を使えばノード間で1マイクロ秒以内の低遅延通信も視野に入り、超高速ネットワークを比較的リーズナブルに構成できます[注9]。

InfiniBandはイーサネットと比較するとコスト対効果が高い場合があるものの、同規格で利用されるツインナックスケーブルは太く取り回ししづらいほか、サブネットごとにサブネットマネージャ(SM)を置く必要があり、イーサネットと比べると若干手間がかかります。しかし、Windows Server 2012では標準でSMB Direct(ファイル共有プロトコルのRDMA対応)が搭載されるなど、OSや仮想マシンモニタによるInfiniBand対応も進みつつあり、クラウド時代にはおさえておきたい技術です。SD

注8) Remote Direct Memory Access(リモートDMA)。別ノードのシステムメモリとのDMA転送。

注9) InifiniBandについては本誌2012年6月号の第1特集でも解説されています。興味がある方はバックナンバーや『Software Design総集編【2001～2012】』をどうぞ。

Column

NICの真価はプロセッサの効率にあらわれる

イーサネットは、コンピュータのデータを信号に変換するハードウェアに加え、フレームデータをOSとハードウェアの間で交換するためのデバイスドライバの組み合わせで構成されています。これはイーサネットコントローラに限った話でもないのですが、小さなIPパケットが大量に発生する使い方などではとくに、ドライバやコントローラチップのデキの良さがカーネル(ドライバ)の消費時間の差となって現れます。

FreeBSDに含まれているif_rl(RTL8129/8139用ドライバ)のソースコード冒頭には、開発者によるコントローラチップの説明[注10]が怒りとともに記されていることで有名です。安価で効率の悪いコントローラチップはプロセッサの処理能力を遠慮なく奪うため[注11]、高負荷な通信で利用する場合には効率の良いNICを選びましょう。

注10) http://people.freebsd.org/~wpaul/RealTek/3.0/if_rl.c

注11) 先のソースではワイヤーレート(100Mbps)を達成するためにPentium II 400MHz相当が必要であろうという見積もりも書かれていますが、このドライバがFreeBSDのソースツリーに初登場した1998年10月、当時のAKIBA PC Hotline!(http://akiba-pc.watch.impress.co.jp/hotline/981017/p_cpu.html)によると、Pentium II 400MHzはバルク価格にて54,800円のハイエンドモデルでした。

第5章 主要なストレージの解説と知識

Writer 長谷川 猛(はせがわ たけし)／Twitter @hasegaw

コンピュータの電源サイクルを超えてプログラムやデータを保持し続ける、永続性を持った記憶領域を、ストレージもしくは二次記憶と呼びます。ストレージというくくりではNAS(Network Attached Storage)やSAN(Storage Area Network)などコンピュータの外側のストレージもありますが、今回はコンピュータに直結される、ハードディスクやSSDなどのストレージに焦点を絞り解説します。

ストレージの位置づけ

昨今の処理データ量の増加、そしてクラウドの流れを受けて、ストレージの重要性は高まるばかりです。ここ10年ほど、個人的には、「現在のコンピュータにおいて一番価値がある部品はストレージなのではないか？」と考えています。現在では、クラウドや仮想マシン技術の普及により、仮想的な計算環境はいくらでも手に入るようになり、また、いつでも替えがある存在となりました。それに対して、ストレージの中にあるユーザデータは唯一のものであり、一度消えてしまったら取り戻すことは困難です。また、仮想マシンとは構成情報、すなわちデータですから、「コンピュータの存在自体が"データ"と化しつつある」とも言えます。

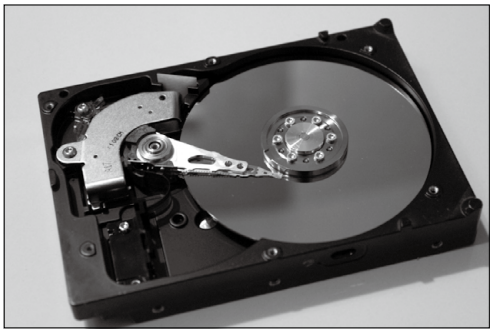
現在の主要なストレージ

現在x86サーバで利用される主要なストレージには、記憶媒体(メディア)として磁気ディスクを用いるハードディスク、またFlashメモリを用いるソリッドステートドライブ(SSD)が挙げられます。SSDには、ハードディスク互換インターフェースを利用するものからPCIe直結型、さらにはプロセッサのメモリチャンネル直結型までさまざまな製品があります。まずはこれらについて簡単に整理してみましょう。

ハードディスク(HDD)

言わずもがな、今いちばんよく使われているストレージと言えばハードディスクでしょう。ハードディスクはプラッタと呼ばれる磁気ディスクを高速回転させ、アクチュエータでヘッドと呼ばれる部分を移動させながら、データを磁気として記録します。現在のハードディスクに近い形となったウインチェスター型ハードディスクは、登場から40年以上経った技術であり実績があること、また製造台数も多くバイト単価が安く手に入ることから、大量のデータを低コストで保持できる点が魅力と言えます。しかし、ドライブ内部でのメディア回転やシーク動作(ヘッドの移動)を伴うため、プロセッサやメモリなどの性能に対して、ハードディ

▼写真1 ハードディスク



スクのアクセス性能は常に取り残されてきました。

ソリッドステートドライブ（SSD）

これまで、ハードディスクが当たり前のように二次記憶として使われてきましたが、コンピュータの歴史では常に遅いデバイスであり、性能問題の原因となりやすい部分でした。そこで、I/O性能問題を解決するため、ハードディスクとは異なる、集積回路だけで実現する二次記憶、すなわちソリッドステートドライブ（SSD）がいろいろと試されてきました。ここではSSD黎明期から現在までの流れについて簡単に紹介します。

SSD黎明期

SSD初期からの製品の1つが、DRAMを装着したバッテリバックアップ式のメモリデバイスで、1970年代からI/Oボトルネック解消のための特効薬として利用されてきました。当初のSSDは書き換え回数が少なかったり、容量が16KB～数MB程度とたいへん限られており、かつ、何億円もする非常に高価なデバイスでしたが、国内でも重要なシステムで利用されました[注1]。またこのコンセプトのPC向け製品が、10年ほど前にも秋葉原などで販売されていたことも、コンピュータが好きな方であれば印象に残っているのではないかと思います[注2]。

性能面では圧倒的なアドバンテージを持つSSDですが、当時のメモリ技術は容量あたりの価格が非常に高かったこと、DRAMなどの揮発性メモリだとバッテリバックアップなどのしくみを組み込む必要もあり、現代のSSDほど扱いやすい技術ではありませんでした。

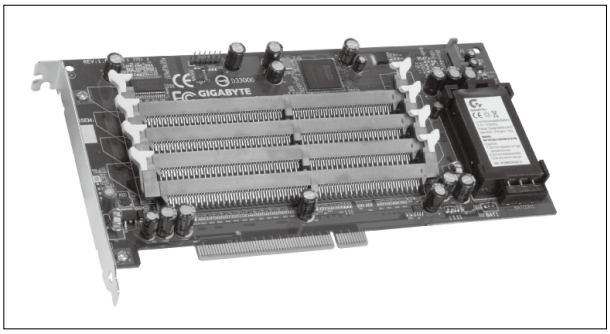
Flashの登場、現在のSSD

一方、半導体メモリ技術の分野では、1980年代にNOR型Flashメモリ、およびNAND型Flashメモリ（以下、NAND Flash）が発明されました。NOR型はバイトアクセスが可能でしたが、その代わりに集積度が比較的低いメモリ技術です。それに対して、NAND Flashはブロック単位のアクセスという制約と引き替えに大容量化がしやすい特徴があります。半導体製造プロセスの技術革新の牽引役でもあり、大容量な記録メディアが安価に手に入るのがNAND Flashのユーザにとっての魅力です[注3]。NAND Flashはその後、USBメモリやデジタルカメラ向けカードメディア、音楽プレーヤなどのデジタル機器、そのほかにも組み込み機器向けのファームウェアやユーザデータの記憶媒体として広く利用されるようになりました。

コンピュータ向けストレージでもNAND Flashが利用されたのは言うまでありません。1990年代は性能が求められるシステムやハードディスクのメカ故障のリスクを回避するための、特定用途向け超高級ストレージという雰囲気がありましたが、2010年ごろには、個人ユーザでも購入

注1） 今から25年ほど前、実際に当時のSSDでコンピュータの性能問題を解決したエンジニアと話をしたことがありますが、当時のSSDも磁気ディスクに対して圧倒的な性能を示しており、高速に動作すればどんどん金になるシステムの、I/Oボトルネックを即解決できたそうです。

▼写真2　i-RAM（GIGABYTE社）



注2） ギガバイト「i-RAM」（**写真2**）など。http://pc.watch.impress.co.jp/docs/2005/0819/gigabyte.htm

注3） 2005年ごろに512MBのSDカードを8,000円前後で購入したように思います。2014年9月現在、同じ価格帯であれば32GBないしは64GBのSDカードやminiSDカードが購入できるでしょう。約10年間で容量は100倍に、バイトあたりの価格は100分の1ぐらいまで下落しています。

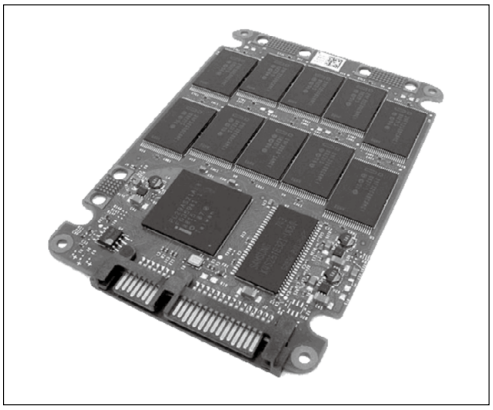
可能なレンジのSSD(2.5インチ、SATAインターフェース。**写真3**)がさまざまなメーカーから発売されるようになり、現在では消費者向けの安価なモデルであれば、2万円の予算で512GBのSATA SSDを狙える時代となっています。

SSDの特徴として読み書き(とくにI/O回数)が圧倒的に速い、性能あたりの電力コストが低い、可動部分が存在しない、でも小容量で高い、といったイメージがあると思いますが、数年後にはスペースあたりの実装容量の観点でもSSDがハードディスクより優位となり、標準の選択肢となってもまったく不思議ではない状況です。

RAID

プロセッサやメモリは故障したら良品に置き換えれば良いのですが、ストレージの故障はデータの消失につながります。とくにハードディスクは可動部分を伴い、コンピュータの中でとくに障害が発生しやすい部分のため、サーバ用途ではとくにRAID(Redundant Arrays of Inexpensive Disks)と呼ばれる、複数のドライブを組み合わせて対障害性を持つ論理的な記憶域を構成する手法が一般的です。

▼写真3　SSD



RAID 1

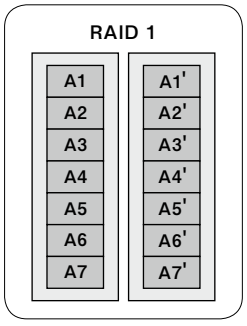
RAID 1(**図1**)は、2基のドライブに同じデータを記録し、冗長性を確保します。片方のドライブに障害が生じても、もう片方のドライブが生き残るためデータは生き残ります。

2基のドライブに同じデータを保持するため、ドライブの利用効率は50%となります。必要容量の2倍の容量を準備する必要があるため、大容量が必要な場合はその容量効率が問題となることもしばしばです。

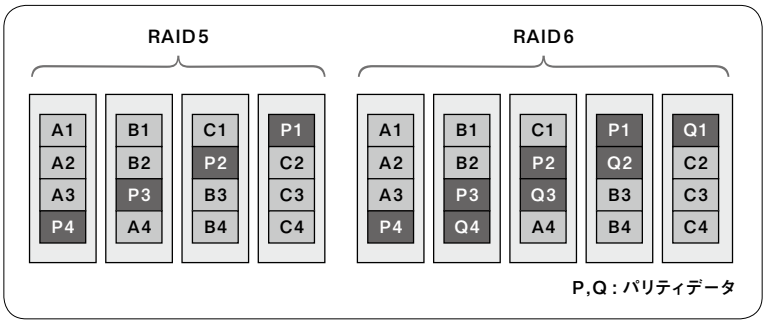
RAID 5/6(分散パリティ)

RAID 1では2基のドライブに同じ情報を記録することで冗長化していましたが、これだとドライブの容量に対して実効容量が半減してしまいます。これに対して、各ドライブにデータを分散記録し、そのパリティ値(全ドライブ上のデータの排他的論理和)を別途保存するRAID 5、およびそのパリティをさらに二重化したRAID 6がよく使われます(**図2**)。

▼図1　RAID 1



▼図2　RAID 5/6



アレイを構成するメンバの1つに障害が発生しても、そのドライブにあった値はほかのドライブ上のデータから排他的論理和を再度求めることで、復元ができます。このため、n台のディスクでRAID-5アレイを組めば実効容量はn-1/nとなり、アレイを構成するメンバデバイスが多ければ多いほど容量効率は上がります。またRAID 6ではパリティを2つ持ち、n-2/nの実効容量でディスク2基の論理障害に耐えられます。

一見いいことずくめに見えますが、RAID 5/6ではドライブ1基のデータを復元するためにアレイ内の残りドライブのデータを読み出してパリティを再計算する必要があります。また、RAID 5/6ボリュームに書き込む際もアレイ内のディスクのデータを読み出して、パリティを再計算する必要があるため、細かな書き込みI/Oが多い場合はドライブに対しての負荷が高くなります。

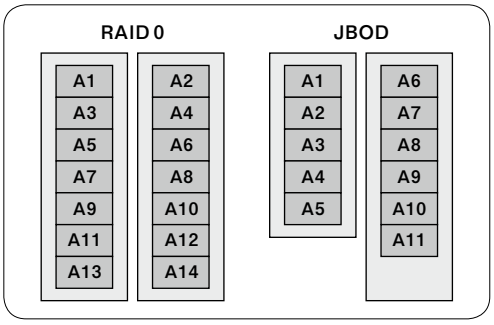
とくに最近のディスクは大容量化が進み、アレイのリビルドにかかる時間も伸びています。RAID 5ではリビルド中にほかのディスクの障害が発生した場合に対応できない可能性も高まっているため、RAID 6の採用が望ましいといえます。

RAID 0、JBOD

冗長化ではありませんが、RAID 0もしくはストライピングと呼ばれる構成では、2基(以上)のドライブを1つの大容量論理ボリュームとして扱えるほか、データを分散配置することでアクセス速度を高めます。仮に、各セクタへのアクセス頻度が偏りなく分散していれば、ドライブ台数分の性能が活かせることになります。

RAID 0と併せて憶えておきたいものとして、複数のディスクをつなぎあわせて1つの論理ボリュームを造り出す、JBOD(Just a Bunch of Disks)と呼ばれる構成もあります(**図3**)。

▼**図3　RAID 0、JBOD**



RAID 10/01/50/60

RAID 10は、RAID 1で冗長化したドライブ群をさらにRAID 0でストライピングして利用します。これにより大きな論理ボリュームを冗長性をもたせて利用できます。冗長性を持つアレイを複数組み合わせて性能も確保したい場合などには、RAID 10はよく利用される方法です。同様に、RAID 5のアレイをストライピングする場合はRAID 50、RAID 6に対してストライピングする場合はRAID 60などと表記していることがあります。

RAID 01と言えば、それはディスクをストライピングした論理ボリュームをミラーリングすることを意味します。RAID 10と同じようにも見えますが、1基のディスクに障害が発生すると、まわりのディスクが巻き込まれてオフラインとなります。そのため耐障害性には大きな差が生じますので、通常は使われません。

SSDと分散パリティ

SSDはハードディスクと比べてI/O性能(IOPS、帯域幅)が数倍～数千倍あり、分散パリティを実現するには、ハードディスクとは桁違いのパリティ演算能力が必要となります。このためソフトウェアRAIDで性能を出すにはホストプロセッサへの負担が重く、専用RAIDコントローラでもSSDの性能に耐えられる組み込みプロセッサやパリティ計算ロジックがなければ性能を維持することはできません。また、一般的に書き込みデータの単位が小さくなり、SSDへ負担がかかりやすい書き込みが繰り返

されることにもなります。本格的なSSDであれば製品内に分散パリティ相当の保護機能が内蔵されていますので、そのような機能を信じてSSDの性能が出やすい構成としつつ、万が一の故障時にはミラー(RAID 1)やレプリカがあるような構成をとったほうが、SSDの性能を活かしやすいでしょう。

ストレージを構成するコンポーネント

ディスクを制御するコントローラ、およびディスクの規格について簡単に紹介します。

HBAとRAIDコントローラ

ディスクをコンピュータに接続するためにはホストバスアダプタ(HBA)が必要です。現在、HBAはSerial Attached SCSI(SAS)もしくはSerial ATA(SATA)インターフェース規格に基づいた製品がほとんどです。また、複数のドライブを統合してRAIDアレイを構成し、ホスト側に論理ボリュームとして見せてくれる機能を持ったものはRAIDコントローラと呼ばれます。

RAIDコントローラには(安物でなければ)マイクロプロセッサが搭載されており、そのプロセッサがRAIDアレイの管理の役割を担います。またオプションとしてキャッシュ用DRAM、ライトキャッシュをバックアップするためのBBWC、FBWC(後述)といったオプションを搭載できます。RAIDコントローラによっては、特定のRAIDレベルなどの利用にはライセンスの追加購入やライトキャッシュのバックアップオプションが必須な場合もありますので、構成見積もり時にはご注意ください。

ハードディスク、SSDの種類

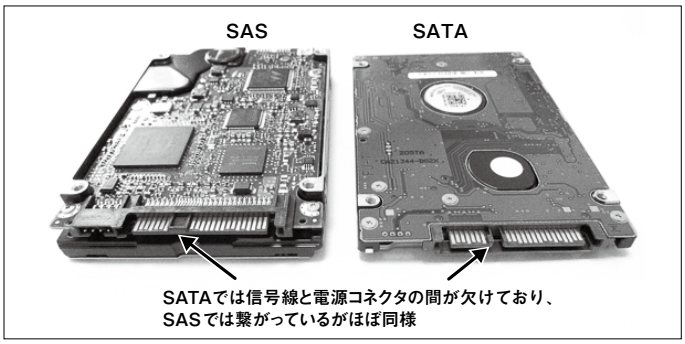
x86サーバのハードディスクやSSDではSAS/SATA準拠のものが大半です。SASは信頼性や速度を求めるオンライン用途、SATAは安価なディスク向けに利用されることが多いインターフェースです。技術的には、SATAは半二重通信かつシングルポート専用に対して、SASは全二重通信で転送効率が良いほか、HBA──ドライブ間のリンクを二重化(冗長化)するデュアルポートに対応するため、同じリンク速度であればSASのほうが一般的に優位です。これらのインターフェースはどちらもSFF(Small Form Factor)コネクタを使った物理接続です(**写真4**)。プロトコルは別物ですが、SAS対応のコントローラはSAS、SATAの両方に対応します。

現在のSSDの多くはSAS、SATAインターフェースのもの、もしくはベンダ独自仕様のものに分類できますが、今後はNVM Expressと呼ばれる、SSD向けに設計されたインターフェースが広まるものと思われます(**表1**)。NVMeの本質は、PCIeバス上でホストとSSDが通信する際のプロトコル(データの受け渡し方法)を標準化したものです。NVMe対応SSDは、PCIeカード形状、M.2形状のもののほか、2.5インチフォームファクタ品ではSFFコネクタを介してPCIeで通信するデバイスが市場に出回るようになるでしょう。

SATAのハードディスクでは、ニアラインSATAと呼ばれるラインがあります[注4]。SASの

注4) Near-line、つまりオンラインに近いですがオンライン向けではありませんよ、という意味からとったネーミングです。

▼写真4　SFFコネクタ(SASとSATA)



ハードディスクは24時間の連続稼働を想定したオンライン製品ですが、ニアラインSATAでは通常のSATAよりもヘビーな24時間の通電／1日あたり8時間程度の利用を想定しており、設計上のデバイスの信頼性が異なります。

知っておきたいストレージの知識

サーバ構成時にストレージで失敗しないための知識をいくつか紹介します。

ライトスルーとライトバック

ストレージへの書き込みポリシーには、ライトスルー(Write Through)とライトバック(Write Back)があります(**図4**)。ライトスルーとは、OSがデータ書き込みを要求したら、メディアにその書き込みが永続化されてから、書き込み完了通知がOSに返されます。これに対して、ライトバックでは、OSがデータ書き込みを要求したら、その書き込みデータがライトキャッシュに載った時点でOSに書き込み完了通知が返され、メディアへはバックグラウンドで遅延書き込みされます。

サーバ向けのディスクコントローラやディスクは、データの喪失を防ぐため、性能より安全を優先するファームウェアが載っています。このため、ライトスルーの書き込みポリシーでは、みなさんが普段手許で使っているコンピュータよりも書き込み速度が遅くなったりします[注5]。書き込み性能を確保するため、RAIDコントローラに後述するBBWC/FBWCオプションを搭載したうえで、ライトバックの書き込みポリシーを利用されることを強くお勧めします。

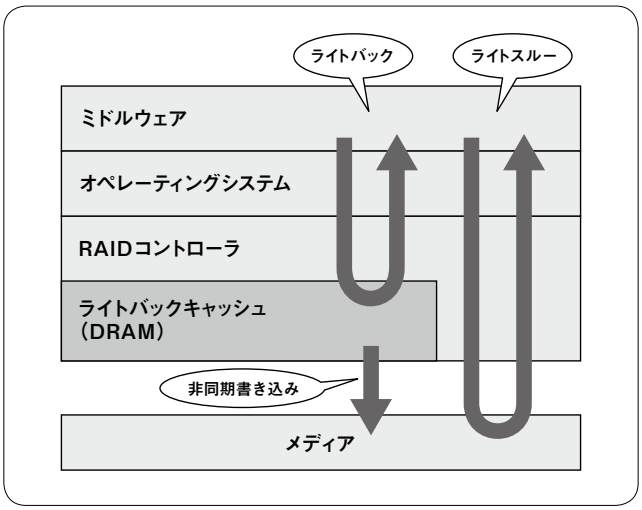
ライトバックキャッシュを保護する理由

複数のセクタで構成されるデータのうち一部のみがメディアに書き込まれ、メディア上でデータの整合性が保たれていない状態をパーシャルライト(partial write)と呼びます。また、OSに書き込み完了が通知されたあとに、実際には書き込みが行われないまま電源断が生じたらどうなるでしょうか？

ストレージへの書き込みを終えたはずのデータが、書き込まれずに喪失する可能性があります。たとえば、銀行口座に1万円が振り込まれたという情報がライトバックされ、それがメディアに反映されないといったことが起きれば大問題です。

分散パリティのRAIDレベル(RAID 5/6など)ではデータの書き込み時には同時にパリティデータを書き込む必要がありますが、パーシャ

▼図4　ライトスルーとライトバック



▼表1　ハードディスク／SSDで使われるインターフェース

インターフェース	リンク速度	備考
SAS	3Gbps、6Gbps、12Gbps	オンライン用途
SATA	1.5Gbps、3Gbps、6Gbps	ニアライン用途など
NVM Expresss	PCIeレーン速度による	SSD用インターフェース

注5) シーク時間を無視した場合、15,000rpmのハードディスクは1分間に15,000回転します。つまり特定のセクタにアクセスするチャンスはシーク動作を抜きで考えると、1秒あたり250回訪れることになります。このため、ライトスルーでは、同じセクタに対し同期書き込みすると最大250回／秒がおよその限界になります。7,200rpmの場合は同様のロジックで計算すると120回／秒ですね。

ルライトが発生すると、パリティによるデータの復元などに支障をきたします。このような問題を解決できるのが、BBWCやFBWCといった、ライトバックキャッシュのバックアップ機能です。

ライトバックキャッシュの保護方法

ライトバックキャッシュの具体的な保護方法として、バッテリーを用いてライトバックキャッシュを保護する方法(BBWC)、Flashを用いてライトバックキャッシュを保護する方法(FBWC)があります(**図5**)。

BBWC(Battery Backed Write Cache)は、未書き込みのデータがある状態で電源断が生じたら、ライトバックキャッシュ(DRAM)に対して専用のバッテリから電力を供給することによってデータを保持し、次回システム起動時に残りデータを書き込みます。データをバックアップ可能な期間はバッテリでDRAMをバックアップ可能な時間に依存しますし、またバッテリは定期的に放電・再充電する必要がある[注6]ほか、長期的運用ではバッテリ交換が必要となるなど、運用上の課題もあります。

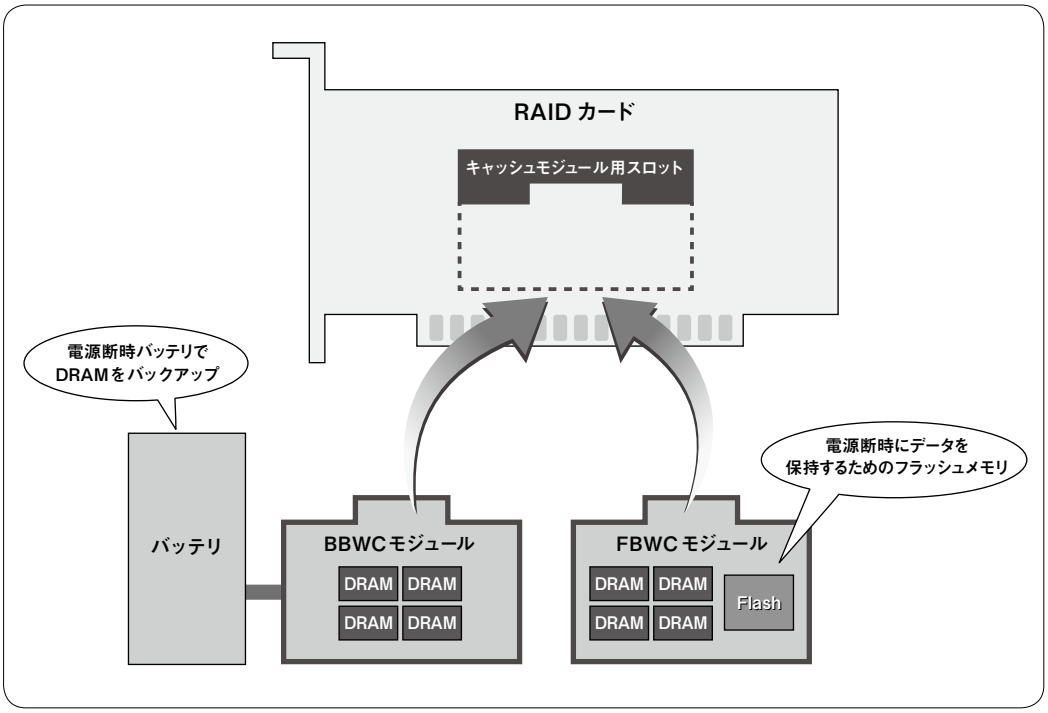
FBWC(Flash Backed Write Cache)は、電源断が生じた場合にはライトバックキャッシュの内容を専用のFlashメモリに待避させ、次回システム起動時に復元するしくみです。FBWCはバッテリを用いないため、BBWCのようなバッテリバックアップ時間の制限や放電・再充電・バッテリ劣化の影響を受けないという利点があります。Flashメモリには寿命がありますが、しかしFBWCの書き込み量はそれほど多くないため、運用期間中に交換が必要となることはないと考えれば良いでしょう。

ソフトウェアRAIDは避ける

PCパーツショップなどで安く販売されてい

注6) 再チャージ中は書き込みポリシーがライトバックからライトスルーに切り替えられたりするので、性能が必要な環境では割と地獄なタイミングです。

▼図5 BBWCとFBWC



るRAIDコントローラ、パソコン用マザーボードやエントリモデルのサーバのRAID機能には、ハードウェア的にはホストバスアダプタ機能のみで、ドライバレベルでRAID機能を実装しているものがあります。サーバ向けの製品でも、安価なラインには、この手のものが混じっていることがあります。

このような設計のものは安全にライトバックできない、パーシャルライトから保護できないものも多いほか、障害時のリカバリで混乱することも多いため、サーバ向けで使うのは止めたほうが賢明です。

分散パリティの乱用に注意

RAID 5/6のような分散パリティによる保護は、正常時の読み込み性能については問題ありませんが、細かな書き込みではなかなか性能が伸びません。仮に16KBのデータブロックを書き込もうとすると、RAID 5/6ではまわりのディスクから読み込み後、パリティを計算して書き込む必要があるためです。細かい書き込みが多い場合はRAID 1/10やPCIe接続型の高速Flashを検討してください。そのような製品であれば、デバイス内でパリティ保護と性能が両立されています。

パススルー機能

最近は、ファイルシステムレベルやミドルウェアレベルでデータを冗長／分散配置する技術、ストレージアプライアンスに相当する機能をソフトウェアとして実装した技術を見かけますが、それらはドライブを1つの論理ボリュームではなく、各ドライブを個別に認識できたほうが都合が良い場合があります。このような場合はRAIDコントローラではなく、ただのHBAでディスクを接続したり、RAIDコントローラにパススルー機能があれば、ディスクを仮想化せずにOS側に認識させられます。

Flash時代に向けて

現在、ストレージのサイジングといえば「容量」と「速度」が主な尺度ですが、書き込み量に制限があるFlashが浸透してくると、「ライフサイクルあたりの書き込み量」を見積もることが重要になってくるはずです。多くの現場で性能監視が行われていますが、ディスクアレイなどへの読み書き量を監視し、1年あたり、どれぐらいの読み書き(TB／年)が発生しているかを把握されることをお勧めします。

アクセス密度の課題

ハードディスクの容量増加に対して、時間あたりに読み書きできるデータ量は取り残されている傾向にあります。言い換えればメディア全体に対するアクセスにかかる時間は伸びています。最近8TBなどの大容量ハードディスクが発表されていますが、仮にシーケンシャルアクセス性能を150MB/sと仮定すると、1時間あたり540GB、全面にアクセスするためには、およそ15時間必要です。実際にはヘッドのシーク動作が入れば、その度合いによって、桁の単位でスループットは低下します。

アクセス密度、すなわち容量に対して時間あたりどれだけのアクセスが可能かという指標は、仮想化基盤やクラウドなどのサービスを構成する際に必要な考え方です。また、RAIDを組む場合には、大容量ディスクはリビルド時間の長期化、そしてアレイ内のディスク障害を誘発する場合もあり、アクセス密度が低いストレージを選定する場合には気を遣います。稼働部品がないため比較的高速、つまりアクセス密度が高いSSDも、製品のアクセスの高速化よりも大容量化が優先されており、長期的にはハードディスクと同様にアクセス密度が低下していくものと思われます。また、ホスト側のバス性能などの制約も課題となってくるかもしれません。SD

第6章 サーバの管理機能

Writer 長谷川 猛(はせがわ たけし)／Twitter @hasegaw

コンピュータに新しいOSをインストールしたり、プログラムを組んだり、その結果で新しいサイトを立ち上げたりするのは楽しい瞬間ですが、システムは構築フェーズよりも運用フェーズのほうが何百倍、何千倍も長いのはご存じのとおりです。本章では、サーバならではの管理機能を簡単に紹介します。

システムを常に正常状態に保つのは何かと難しいものです。突発的なユーザアクセス、DBトランザクションやバッチ処理を乗り切る、といったワークロードレベルのチャレンジもありますが、その前提となるワークロードを支えるためのハードウェアを、健全な状態に保つことも大事です。最近のx86サーバは運用に役立つ管理機能が充実していますので、簡単に紹介したいと思います。

リモート管理機能の概要

サーバ用のシステムボードであれば、多くの場合、アウトバンド管理(Out-of-band Management：以下OOB)機能を搭載しています。OOB管理機能を活用すると、地球の反対側にいても、まるでサーバの目の前にいるかのようにOSをインストールしたり、マシンがハングアップした際にコンソールを確認したり、といったことができます(**表1**)。

これらの機能は、システムボード上に搭載された、ホストプロセッサとは独立したシステム管理専用の組み込みプロセッサが提供しています。このため、管理プロセッサに接続できるようになれば、サーバの目の前にいなくとも、OSインストール前、もしくはOSが停止していようとも操作ができます。BMC(Baseboard Management Controller)には、IPMI(Intelligent Platform Management Interface)とサーバベンダ独自のインターフェースの2つがあります。

IPMI

OOB管理機能の実装内容はメーカーやシステムボードによりまちまちですが、標準的に利用されるインターフェースとしてIPMIがあり、本仕様に準拠していればTCP/IPネットワーク越しにさまざまな状態検出や電源操作などができます。たとえば手元のLinuxマシンなどでipmiutilなどのソフトウェアを実行すれば、BMCを持つたいていのサーバの電源オン／オフ操作ができます。

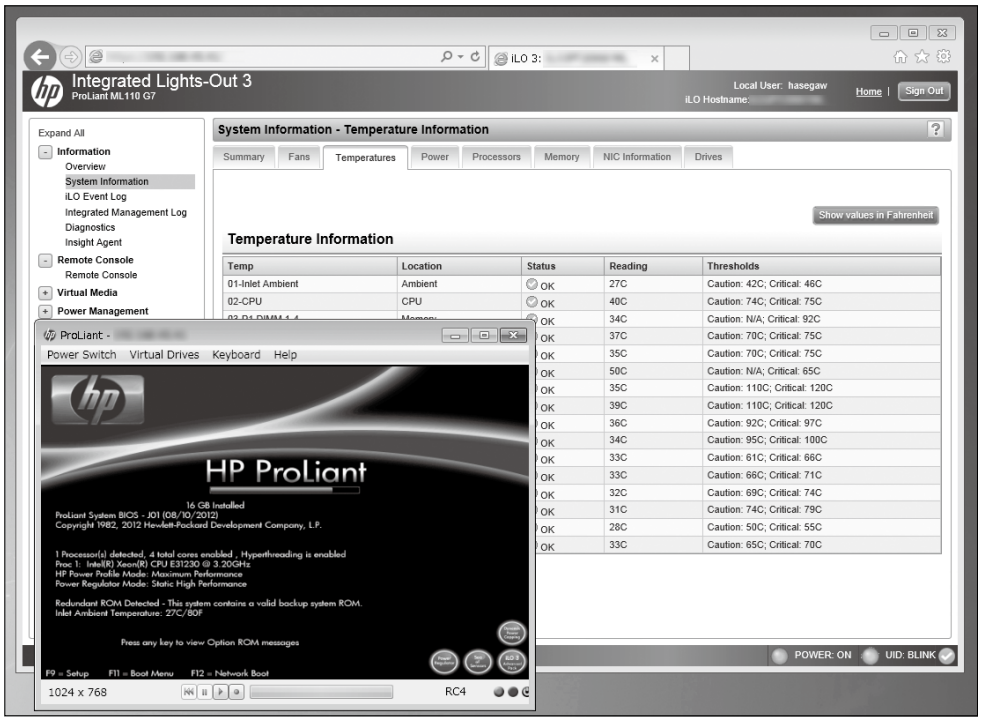
サーバベンダ独自のインターフェース

各サーバベンダがさまざまな付加価値をつけた管理機能を提供しています(**図1**、**表2**)。BMCが管理Webページを持っており、そこに

▼表1　アウトバンド管理で提供される機能の例

サーバの電源オン／オフ／リセット	デバイスの健全性(障害)の確認
リモートコンソール(IP-KVM)	ファン回転数センサや温度センサの値を確認
仮想メディアの装着	ウォッチドッグタイマ
仮想シリアルポートへの接続	仮想NMIボタンの押下

▼図1　iLO3（Integrated Lights-Out 3）の例



▼表2　さまざまな名称のOOB BMC機能

ベンダ	OOB BMC機能の名称
DELL	Dell Remote Access Controller（DRAC）
富士通	Baseboard Management Controller（BMC）
日立	Baseboard Management Controller（BMC）
HP	Integrated Lights-Out 4（iLO4）
IBM	Integrated Management Module II（IMM2）
NEC	EXPRESSSCOPE（R）
Supermicro	SMT

ログインすることで各種情報を取得したり、リモートKVM機能を利用したりできます。

BMC関連のTips

OOB BMCで快適なインフラ作業を行うためのノウハウを紹介します。

踏み台環境があると便利

BMCはTCP/IP的に接続できれば利用できますが、ターゲットマシンの近くにWindowsなどのデスクトップ環境やISOリポジトリとなるWebサーバなどがあると何かと便利です。

シリアルコンソールで接続できる

BMCによっては、物理マシンへシリアルコンソールで接続できますので、カーネルレベルの問題を追いかけているようなときにも便利です。実際にオペレーティングシステム開発者の方などでBMCを介したリモートデバッグを行われている方はそれなりにいるようです。

機器故障の問い合わせも楽々

筆者がとある客先のサーバの面倒を見ていたころ、サーバベンダに故障したサーバの部品交換を手配する際など、障害内容を説明するデータとしてIPMIログなどを求められることが多々ありました。コール前にIPMIのログをとっておけば、やりとりもスムーズですし、稼働中のワークロードに影響も与えません。

ファームウェアを最新に保つ

BMCのファームウェアもシステム本体のBIOS/UEFI同様に、たびたびアップデートがなされています。アップデートの内容はさまざまですが、ときとしてホストプロセッサ側のワークロードの性能に影響を及ぼす問題を修正しているケースなどもあります。管理プロセッサはホストプロセッサと独立して動作しますが、まったく干渉がないわけではありません。

幸いBMCのファームウェアは(サーバによりますが)管理Webページなどから、サーバを稼働させたまま簡単にアップグレードできます。BMCの機能的な不具合、BMCが原因かわからないが怪しい性能問題などに直面したら、BMCのファームウェアバージョンがまわりのサーバと一致しているか確認したり、また新しいファームウェアがあればアップグレードしてみるとよいでしょう。

非純正パーツの罠

最近のサーバは省エネを意識した作りになっており、デフォルト設定ではファンも必要最小限しか回さないようになっています。この風量は、サーバ内の温度センサなどから情報を集めてコントロールしています。しかし、サーバ内に、温度センサ機能などの互換性がない非純正コンポーネント(例：DIMM)が組み込まれている場合、これらの温度管理機能が機能せず、省エネ機能が無効化され、ブロアファンが全開になってしまうケースを見たことがあります。

クラウド時代だからこそのBMC

OpenStackのIronicは、IPMIを通じて物理的なコンピュータを管理するためのコンポーネントです。この機能は登場からまだ日が浅い機能ですが、仮想マシンモニタ(VMM)を介さずに、物理的なコンピュータをクラウドの一部に組み込めるようになります。現在のクラウドと言えば仮想マシンの切り出しが一般的ですが、サーバベンダが提供する管理ソリューションでなくても、BMCを介したオーケストレーション(サーバ管理諸作業の自動化)が可能になりつつあります。

コンピュータがまさに「資源」として扱われる現代のクラウド化の流れでは、大量のコンピュータを少ないコストで適正な状態に保つことが必要となります。アウトバンド管理機能があれば、設置したサーバを少ない労力で管理でき、今後はより重要性を増していくでしょう。SD

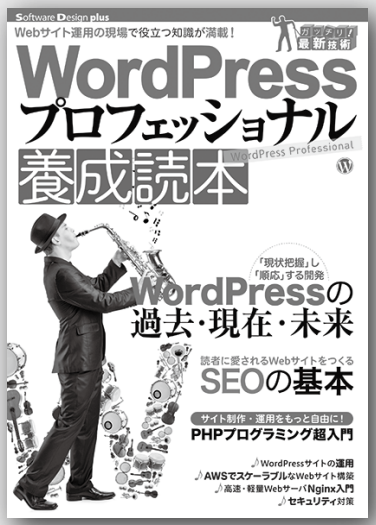
Column Intel AMT

PC向けの同様な機能としてIntelの企業向けチップセット(Q)シリーズを搭載したコンピュータでは、vProと呼ばれる機能の中に、Intel Active Management Technology(AMT)と呼ばれる機能があり、ワークステーションの管理や一括メンテナンスに利用できます。vProは、Intel Qシリーズチップセットに含まれます。

MicroATXフォームファクタのマザーボードが多いため、ディスクを大量に積んだりPCIeカードを大量に挿す用途には向きませんが、リモートからコンソール画面の確認・操作が可能なマシンの選択肢としては安価に手に入るため、自宅サーバをこれから小さく始めたい方には向くかもしれません。

▼図2 Intel AMT(リモートのコンソールにアクセスしている様子)

Software Design plus　　技術評論社



WordPress プロフェッショナル 養成読本

WordPress Professional

WordPressは、CMSツールの定番となり世界中で多くのユーザを得ています。今後もWordPressを導入する個人ユーザや組織は増えることが予想され、その運用や保守の知識を持つ人材の育成に関心が集まっています。Webサイトのデザインはもちろん重要ですが「SEO」「クラウド（AWS）」「PHP」「セキュリティ」「サーバの設定」などはWordPressの運用に欠かせないキーワードとして挙げられます。
本書では、WordPressを運用するエンジニアやもっと活用したいと考えているユーザがおさえておきたい話題をまとめています。

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

大好評発売中！

こんな方におすすめ　WordPressユーザ

Software Design plus　　技術評論社

内部構造から学ぶ

PostgreSQL

設計・運用計画の鉄則

業務システムの開発案件ではライセンス費用がかからないLinux&PostgreSQLの組み合わせが採用されるケースが増えてきています。ただ、後工程になって問題が発生する試行錯誤的なシステム設計／運用計画が見受けられます。
そこで本書では「システム設計」「システム運用」「メンテナンス」についての技術トピック（内部構造や仕組み）を、大手SIerに所属し、PostgreSQLのデータ構造とアルゴリズム、振る舞いなどを熟知した技術者陣が自らの経験やノウハウも踏まえて解説します。

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

大好評発売中！

こんな方におすすめ
・PostgreSQLのシステム設計をする技術者、開発者
・運用計画を作成する管理者

8086時代から今を俯瞰する

CPU温故知新

今回編集部から、「アセンブラレベルで見る、8086のころから現在の最新CPUに至るソフトウェア的な推移を簡単に紹介してほしい」という依頼をもらいました。ただ、8086というよりは8080から始めるのが筋だろうという気もしますので、本稿ではこのあたりから紹介します。

Text：大原 雄介（おおはら ゆうすけ）

8080とZ80（と8085）

1974年に発表されたIntel 8080 CPU（以下8080）の命令セットやレジスタ構成[注1]は、1972年に発表されていたIntel 8008 CPU（以下8008）の影響を比較的受け継いでいます。直接的な命令の互換性そのものはありませんが、1～3byteの可変長命令フォーマットとか、当時としては比較的多いレジスタ（8008ではA/B/C/D/E/H/L/PCの8つ）を装備し、また命令そのものも割と似ています（というか、8008を下敷きに8080の命令セットを策定したのだから当然ですが）。

その一方で、一部の命令はオペランドを持たない（オペコードの中にレジスタ指定が含まれている）とか[注2]、スタックは専用レジスタに格納される（最大7レベル）形で、いわゆるスタックポインタ（以下SP）はないといったあたりは、4004/4040系の特徴を引き継いだ形です。ちなみにアドレスは14bitで、最大16KBのメモリ空間が利用できました。

8080の登場

さて、この8008を下敷きに作られたのが8080です。8008から8080の相違点は、

- アドレスが16bitに拡張され、これに伴いプログラムカウンタ（以下PC）も16bit化された
- 従来のスタック専用レジスタが廃され、代わりにSPが搭載された
- レジスタ構造が若干変化した（レジスタの数そのものはA/B/C/D/E/H/L/PCで同じだが、PC以外のレジスタは16bitに拡張できるようになった）
- ステータスレジスタに示されるフラグの数が増えた

のようなものです。命令そのものも多少変更され、基本、オペコードがオペランドを含む（レジスタやアドレス指定を必要としない。たとえばRET命令などは当然持たない）ようになりました。また、8008は純粋に8bit演算命令しかありませんでしたが、8080ではアドレス計算などのためにいくつかの16bit命令も追加されています。

さてその8080命令ですが、相対ジャンプとか相対コールがない[注3]、といった若干の欠点はあったものの、当時としては比較的整理されており、プログラムはしやすかったです。よくx86系命令に関して「命令直交性がない」と言われることがありますが、筆者的には8080に関してはむしろ命令直交性がきちんと取れていた

注1） CPUが直接実行できる、CPU内のメモリがレジスタです。

注2） 機械語命令では、最初に命令を示すオペコードが来て、その後に何をするかのオペランドが来るように並んでいます。

注3） 相対アドレスを使った命令だと、コードのブロックをメモリ上の場所を移動してもそのまま動作するというメリットがあります（リロケータブル）。

ように思います。

Z80の登場

さて、この8080とかなりの部分で命令互換性を取りつつ、より高速化したCPUとして登場したのがZilogのZ80です。高速化の大部分は製造技術によるもので、動作周波数そのものが引き上げられているのですが、命令の処理サイクルを若干減らすことによる高速化も施されています。これに加えて、いくつかの新命令や新レジスタも追加されました。

まずレジスタではIX/IYという2つの16bitインデックスレジスタが追加されており、これを利用したアドレスの間接指定がサポートされています。またいくつかの拡張命令は、8080だと複数命令を要していたものが1命令で済むようになっています(もっとも、こうした拡張命令は従来のインデックスレジスタを使った命令よりも処理が遅かったので、性能を取るかコードサイズを取るかという判断が難しいものも少なくなかったのです)。

それと、なぜか8080ではできなかったこと[注4]ができるようになるなどの改善も施されています。このあたりは、8080とZ80で開発者がかなり被っていることと無縁ではないでしょう。ちなみにこのインデックスレジスタを使っての間接アドレス指定で、だいぶ命令セットの直交性は損なわれることになってしまいました。

逆に、Intel自身も8080を改良して1976年に8085プロセッサをリリースしています。ただし、8080と8085の相違点はおもにハードウェア(5V単一電源での駆動とか、8244 Clock Generator/8228 System コントローラ & Bus Driverの内蔵など)と性能で、同社のマニュアルによれば8085は8080に比べておよそ50%高速とされていますが、これはそもそも動作周波数が50%高速ということです。命令面での違いは、SIM/RIM命令[注5]の追加のみです。

ということで、その8080を16bit化したのが1978年に発表された8086(と、その外部8bitバス版の8088)です。命令というかバイナリレベルの互換性はまったく保たれていませんが、アセンブラレベルでは互換性があり、アセンブルしなおせば既存の8080のコードがそのまま8086で動作するようになっています。

この8086の最大の特徴は16bit化であり、命令はすべて16bit拡張されました。ここで「16bit化された」と書かないのは、たとえばAレジスタは、AXレジスタを指定すると16bit幅の演算になるが、AH/ALレジスタを指定すると8bit幅の演算になっており、つまり既存の8bit演算はそのまま残しながら、16bit演算も追加したという形になるからです。これに伴いほかのレジスタも変更になりましたが、これがちょっと問題のネタになりました。

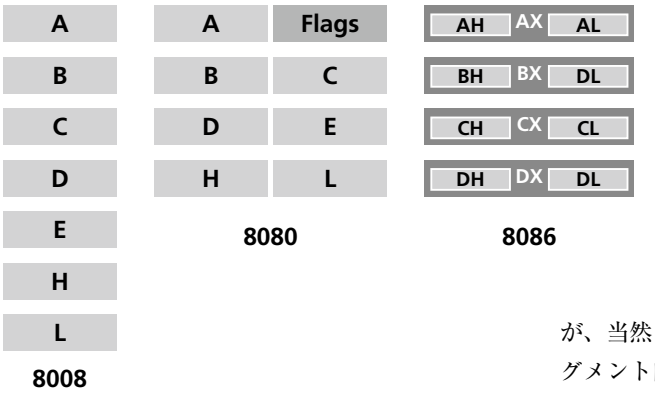
図1は8008/8080/8086の汎用レジスタの構成を比較したものです。もともと8008では7つの汎用レジスタがあり、8080もこれを引き継ぎました。といっても、H/Lはおもにアドレス指定用だから実質5つという言い方もできます。この構図は8080でも引き継ぎましたが、レジスタ構成を変更し、2つの8bitレジスタが1つの16bitレジスタとして機能するような形になっています。このしくみをそのまま取り入れたのが8086なわけです。

ただ8bit命令を処理する場合、これは汎用レジスタが8個相当ということで、むしろ8080/8085より増えています(実際はそう話は単純ではない)が、16bitモードで普通に利用しようと思うと、汎用レジスタが4つしかないということになり、実はレジスタを使いまわしての演算が非常に面倒になりました。加えて実際は「命

注4) たとえばBCD(Binary-coded decimal:2進化10進数)演算の減算。8080では加算のみ可。

注5) SID/SOD(入出力ポート)との間のデータ入出力と、割り込み関連設定。

▼図1　8008/8080/8086のレジスタ配列



令ごとに、どのレジスタが利用できるか」も細かく指定されてしまった結果、命令の直交性がかなり失われることになってしまいました。その結果、レジスタを使いまわすために、やたらめったら途中のデータをスタックに積む必要が出てきたので、個人的にはあまりうれしくありませんでした。

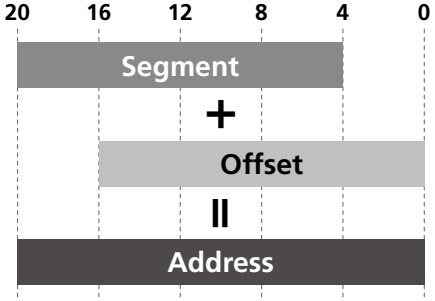
セグメント

ただ、これよりも8086を8086たらしめていたのは、セグメントの存在です。8086では最大1MBのメモリ空間を利用できましたが、アドレスで言うところ20bitになります。普通は20bit(とかもう少し大きく24bitとか)のレジスタを用意するとともに、これを直接扱えるような命令を追加するという形になるかと思います。実際8080も、基本は8bit演算しかできませんが、アドレスはHとLという2つのレジスタに上位下位8bitずつ分割して演算・格納することで64KBの空間を利用できました。

ところが8086の設計にあたっては、どうも汎用レジスタのサイズを16bitから拡張するのがいやだったらしく、代わりにセグメントという概念が導入されました。8086では、1MBのアドレス空間を利用するためには、セグメントとオフセットという2つのアドレスが利用されます。この2つは**図2**のように4bitずらした形で配置されて、この2つの合計が実アドレスになるというしくみです。

これに併せて、CD(Code Segment)/DS(Data Segment)/ES(Extra Segment)/SS(Stack Segment)という4つのレジスタが新たに追加されました。この結果、アドレスを指定する場合には、原則セグメントとオフセットの両方を設定する必要が出てきましたが、当然これは遅くなります。そこで、同一セグメント内だったらオフセット指定だけでも済むように、Index指定あるいはDisplacementによるアドレス設定も行えるようになっています。実際に、8086のUsers Manualを見ると、EA(Effective Address：実効アドレス)Calculation Timeは、**表1**のようになっていて、Offsetだけを指定するようにすれば、だいぶ少ないCycle数で実行できることになっています。加えて言えば、いくつかの命令(たとえば文字列操作系命令)は最大でも64KB(つまりオフセットで変更できる範囲)でしかデータを扱えないという制約があり、そのためプログラマがセグメント境界を把握して、きちんとハンドリングする必要がありました。

▼図2　セグメントとオフセットによる実効アドレスの計算



▼表1　8086のEA指定時にかかるCalcilation Time

アドレス指定方法	所要Cycle数
Displacementのみ	6
Base or Indexのみ	5
Displacement+Base or Index	9
Base+Index	7 or 8
Displacement+Base+Index	11 or 12

この制約は、単にアセンブラレベルではなく、もう少し高級な言語にも影響を及ぼしました。たとえば、この当時のC言語[注6]の場合、ポインタにfar(セグメントまで操作する20bitのポインタ)とnear(オフセットのみ操作する16bitのポインタ)の2種類の修飾子が追加されることになり、これがコードとデータの両方に影響することになりました。

メモリモデル

プログラミングの際には、**表2**の4種類のどのメモリモデルを選ぶかを決める必要がありました。ライブラリも各々のモデルに合わせて4種類用意されていて、ポインタの大きさがモデルごとに違うので、途中でモデルを切り替えようとすると、なかなか苦痛な作業が待ち構えていたなど、プログラマにはとにかく不評でした。

ちなみにコンパイラによっては、さらにTiny(コード/データが同一セグメントに配され、nearポインタのみ)とHuge(Largeにほぼ同じだが、配列サイズも64Kを超えることが可能になっている。Largeは配列サイズは64KB以内)というモデルが用意されていました。言うまでもなくnearの方がfarより高速に動作しますが、サイズ面の制約は厳しいわけで、このあたりでずいぶん苦労した覚えがあります。

80286

8086と命令互換ながら、いくつかの拡張を施したものが1982年に発表されたIntel 80286です。アドレスバスは24bitになり、最大16MBを扱えるようになっています。80286では、リアルモード(高速な8086として動作するだけ)とプロテクトモード(拡張機能を全部利用できるようになる)の2種類の動作モードがあります。

プロテクトモードは本格的なOSへの対応に向けて、4レベルの特権レベルや、MMU(Memory Management Unit)や最大1GBまで利用可能な仮想アドレス、タスク切り替えなどの機能が追加されています。この仮想アドレスについては、新たにDescriptor Tableと呼ばれる管理領域を設け、これをMMUから参照する形で実現しています。ただ、一度プロテクトモードに移行するとリアルモードに戻る方法がない、ということでOS/2[注7]がソフトウェア的にResetを掛けることで強引にプロテクトモードからリアルモードに戻る方法を実装したのは有名な話です。もっとも、プログラマからすると、プロテクトモードはともかくとしてリアルモードにはいっさい差がなく、引き続きfar/nearポインタやら64KBの壁やらに悩まされることになりました。

80286の裏技

オマケとして挙げられるのは、80286で24bitの物理メモリアドレスをサポートする際にちょっとしたバグが紛れ込んだことです。リアルモードでは1MB以上のメモリアドレスが利用できないので、アドレスバスのA20～A23のメモリアドレスは強制的に0になるはずですが、これを1にする方法が発見され、1MBを超えた領域に64KBほどのメモリ空間が利用できるようになりました。これはHIMEM(High Memory)として重用され、とくにMS-DOSでデバイスドライバをこの領域に押し込んでメインメモリを空かせるのに役立ちました。これとは別に、EMS(Expanded Memory Specification)という技法も開発され(こちらはLotus/Intel/Microsoftの共同提唱)、1MBを超えるメモリ

注6) Cが高級言語かどうかについて異論のある方は多いと思うが、とりあえずアセンブラと比べれば高級であろう、という判断であって、より高級な言語があることは否定しない。

注7) IBMとマイクロソフトのPC-DOSやMS-DOSの後継のマルチタスクOS。

▼表2 メモリモデル

Small	コード/データ共にnear
Compact	コードがnear、データがfar
Medium	コードがfar、データがnear
Large	コード/データ共にfar

をバンク切り替え式に利用できましたが、プログラムからこれを使おうとすると、EMSマネージャに対してバンクの要求を出す(つまりプログラム自身がメモリ管理を行う必要がある)とか、バンクそのものは16KBと小さいので、データを置く場合はともかくコードを置こうとするとけっこうたいへん(EMSに対応したオーバーレイ構造を出力してくれるリンカはそう多くなかったと記憶している)で、結局高速なRAMディスクに使われる程度のニーズがメインだったように思います。

80386~80486

Intel 80386(またはi386)と32bit OSが登場したことで、やっとプログラマは64KBの壁や、near/farポインタから解放されることになりました。80386では遂に32bitアドレスがサポートされ、また命令も32bit拡張されています。ただ、その拡張の方法は純粋に既存の8086/80286の命令セットを32bitまで対応できるように拡張したという方法であり、たとえば汎用レジスタも**図3**のように、それぞれ32bit化がなされているだけです。ほかにIndexとPCも32bit化されていますが、逆にSegment Selectorに関しては16bit互換モードのみの利用ということで、32bit化は見送られて16bit幅のままとなっています。

動作モードからみると、80286と同様にリアルモードとプロテクトモードが搭載されています。ただ、80286の場合は既存のプログラムを動作させるためにはリアルモードに戻る必要がありましたが、80386では仮想8086モードがプロテクトモードの中で動作するため、ブートのときだけリアルモードでも、すぐにプロテクトモードに移行し、以後は32bitのアプリケーションはプロテクトモードで、16bitのアプリケーションは仮想8086モードでそれぞれ実行できるようになりました。もっともこのメリットを享受するためには、当然OSの側もこれに対応する必要がありました。これにきちんと対応したのはMicrosoftで言うと、Windows NTからです[注8]。

命令セット的に言えば、アプリケーション的にはこの世代では大きく変わる部分はありません。もちろんRing 0(特権モード)で利用できる命令類は増えていますし、これに合わせてシステムの状態を設定/表示する特権レジスタ類も追加されていますが、これはOSプログラマのみに関係する部分です。

80486の登場

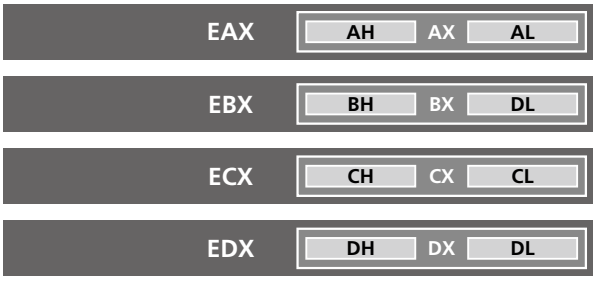
この80386をハードワイヤードで作り直したとも言うべきものが1989年に発表された80486(またはi486)です。ハードウェア的な相違点としては絶対的な性能差に加えてFPUの内蔵化[注9]、それとキャッシュの内蔵があります。プログラミングの観点からすると、とくに命令が増えたとか変わったということはないのですが、命令のスループットやレイテンシが80386からずいぶん変わったので、最適化の方法が当然変わっています。

とくに80386と比べると、マイクロコードを参照しないで直接デコード→実行が行える命令が増えた反面、パイプラインが長くなったため、パイプラインストール/ハザードによる影響が大きくなったのが大きなポイントです。また

注8) Windows 3.1でもWin32sという、32bit APIのサブセットが配布されて部分的に32bit化が可能でした。

注9) 80386までは基本的にFPUはコプロセッサとして別チップになっており、普通のアプリケーションはFPUはないという前提でプログラミングをしていた。

▼図3　80386の32bit化されたレジスタ



80386

8～16KBではありますが、命令キャッシュを搭載したため、たとえば自己書き換え式のプログラムなどが正しく動作しなくなるという問題も、この世代から発生しています。

Pentium～Pentium 4

1993年に発表されたPentiumでは、インオーダ(命令順実行)、かつ命令数は限定的ながら2命令の同時実行が行えるようになり、ここでまた最適化の方法が少し変わってきています。ただそれより影響が大きいのはMMXの搭載でしょう。

MMXはx86(8086系アーキテクチャの総称)としては初のSIMD(Single Instruction Multiple Data：命令の並列化)処理命令で、8bit × 8、16bit × 4、32bit × 2の3種類のデータ型に対して算術演算(加算／減算／乗算／乗加算)・比較・変換・論理演算・シフト・転送の各演算が可能というものでした。ただしMMXが扱えるのはあくまでも整数演算に限られていました。

MMXの主要な目的はマルチメディア性能の向上で、具体的にはJPEGとかMPEG-1のエンコード／デコードの高速化がターゲットの1つでした。これらはいずれも8bitの整数型でデータを保持しており、処理(たとえばDCT/IDCTとかQuantize/IQuantizeなど)はいずれも8bitのままで行えたので、理論上はx86命令を使うより最大8倍高速になるというものでした。これに合わせて、新しくMMXレジスタが8本追加されています。

もっともMMXをフルに活かそうとすると、それなりにTipsが必要でした。MMXは64bit、つまり8byte単位でアクセスするために、とくにメモリ間との転送の際にはこの8bytes Boundaryが重要でしたし、2次元処理だとDCT(離散コサイン変換：画像データ圧縮で使われる)とかFFT(高速フーリエ変換)ではデータの転置が必要になりますが、MMXはこれをいっさい考慮していないので、プログラム側でうまく処理しないと急に性能が落ちることがありました。

またあるMMXレジスタに書き込むと、続く2cycleはそのレジスタにアクセスできないというペナルティがあるので、うまくレジスタを使いまわす必要がありました。

とはいえ、このあたりは一度方法が決まると使いまわしでいけることもあってか、しだいに利用されるようになってきましたが、高級言語でのMMXサポートは案外に遅く、それもあってか普及にはやや手間取った(しかもその間に3DNow!やらSSEやらが登場してしまった)のは想定外だったかもしれません。

Pentium Proの登場

Pentiumに続き1995年にPentium Proが投入されましたが、これは完全Out-of-Order構成で、しかも最大同時3命令実行ということもあり、またもや最適化の技法が大きく変わることになります。とくにデコーダがSimple DecoderとComplex Decoderに分離され、Simpleのほうは1cycleでデコードが終わってすぐに実行されるケースが多い(1～4cycle：命令によって変わる)のに対し、Complexでは10cycle以上要したりするので、どの命令を使うかに注意を払う必要がありました(**写真1**)。

当初のPentium ProはMMXは搭載されていませんでしたが、これをベースとしたPentium IIではMMXが搭載され、次のPentium IIIではMMXよりさらに強力なSSEが搭載されました。こちらでは浮動小数点演算がサポートされ(というか、初代のSSEは浮動小数点演算しかサポートされず)、新たに128bit幅のXMMレジスタが7つ搭載され、ここで1cycleに4つの浮動小数点演算が行えるようになりました。もっとも、初代のSSEは相変わらずいろいろ使いどころが難しいというか、癖が多かったこともあり、華々しく発表された割にはアプリケーションは少なかった(というか、チューニングが追いつかなかった)感があります。開発ツールのSSE対応も遅れており、結局算術演算が必要な部分だけインラインアセンブラでSSE命令を呼ぶなんて形で使われることが少なくな

▼写真1　“Intel Architecture Optimization Manual 242816-003（1997）”より抜粋

PENTIUM® PRO PROCESSOR INSTRUCTION TO DECODER SPECIFICATION

intel

ARPL m16	complex	CLD	4
ARPL rm16, r16	complex	CLI	complex
BOUND r16,m16/32&16/32	complex	CLTS	complex
BSF r16/32,m16/32	3	CMC	1
BSF r16/32,rm16/32	2	CMOVB/NAE/C r16/32,m16/32	3
BSR r16/32,m16/32	3	CMOVB/NAE/C r16/32,r16/32	2
BSR r16/32,rm16/32	2	CMOVBE/NA r16/32,m16/32	3
BSWAP r32	2	CMOVBE/NA r16/32,r16/32	2
BT m16/32, imm8	2	CMOVE/Z r16/32,m16/32	3
BT m16/32, r16/32	complex	CMOVE/Z r16/32,r16/32	2
BT rm16/32, imm8	1	CMOVL/NGE r16/32,m16/32	3
BT rm16/32, r16/32	1	CMOVL/NGE r16/32,r16/32	2
BTC m16/32, imm8	4	CMOVLE/NG r16/32,m16/32	3
BTC m16/32, r16/32	complex	CMOVLE/NG r16/32,r16/32	2
BTC rm16/32, imm8	1	CMOVNB/AE/NC r16/32,m16/32	3
BTC rm16/32, r16/32	1	CMOVNB/AE/NC r16/32,r16/32	2
BTR m16/32, imm8	4	CMOVNBE/A r16/32,m16/32	3
BTR m16/32, r16/32	complex	CMOVNBE/A r16/32,r16/32	2
BTR rm16/32, imm8	1	CMOVNE/NZ r16/32,m16/32	3
BTR rm16/32, r16/32	1	CMOVNE/NZ r16/32,r16/32	2
BTS m16/32, imm8	4	CMOVNL/GE r16/32,m16/32	3
BTS m16/32, r16/32	complex	CMOVNL/GE r16/32,r16/32	2
BTS rm16/32, imm8	1	CMOVNLE/G r16/32,m16/32	3
BTS rm16/32, r16/32	1	CMOVNLE/G r16/32,r16/32	2
CALL m16/32 near	complex	CMOVNO r16/32,m16/32	3
CALL m16	complex	CMOVNO r16/32,r16/32	2
CALL ptr16	complex	CMOVNP/PO r16/32,m16/32	3
CALL r16/32 near	complex	CMOVNP/PO r16/32,r16/32	2
CALL rel16/32 near	4	CMOVNS r16/32,m16/32	3
CBW	1	CMOVNS r16/32,r16/32	2
CLC	1	CMOVOr16/32,m16/32	3

かったように思います。

SSEの変遷

SSEについては、むしろSIMD命令と一緒に追加されたキャッシュ制御のPrefetch命令が、アプリケーションから明示的にキャッシュの先読みを行えるということで、チューニングの際にけっこう利用されたと記憶しています。ただこのPrefetch命令そのものの実装は、AMDが3DNow!で実装したのが先で、これを後追いで取り込んだ形になります。

このSSEの使いにくさは、続く2000年に発表されたPentium 4コアで搭載されたSSE2やSSE3でだいぶ使いやすくなりました。SSE2は浮動小数点演算に加えて整数演算もサポートされ、また64bitの倍精度浮動小数点演算も追加されています。細かいところでもいろいろ便利な命令が追加され、SSEを使うよりもプログラミングが簡単になりました。SSE3に関しては、SIMD命令の拡張はSSE2の補完といったところで、むしろスレッド制御のMONITOR/MWAITが便利と評されましたが、これはOSプログラミング側の話でアプリケーションにはほとんど関係ありませんでした。

別の観点からこの世代のプロセッサを見ると、Pentiumの世代までは従来型のCISCプロセッサとして分類できるが、Pentium Pro以降はデコード段で、一度x86命令を内部的により粒度の高いRISC命令（一般にMircoOpなどと呼ぶ）に変換をし、これを実行するというCISC-RISC複合型プロセッサとなっています。このため、最適化などもx86命令そのものというよりは、これを変換したあとの内部命令の挙動を考えないと難しいといった、一段ステージが上がった形になっています。

Pentium 4～現在

Pentium 4の第3世代にあたるPrescottで、Intelもx86の64bit拡張に正式に対応しました。もともとはAMDがOpteron/Athlon 64でx86-64（後にAMD64）という名前で仕様を公開、Intelはこれを当初EM64T、後にIntel 64という名称で利用していますが、一般にはx64として通用します。考え方は16bit→32bitのときとよく似ており、すべてのレジスタを64bitに拡張し、これに合わせてすべての命令も8/16/32/64bitに拡張されました。

この結果として命令はどんなふうになったか、という一例が（**写真2**）です。ADD命令だけで22ものオペコードが並ぶ始末です。ここのオペコードを見ていただくとわかるとおり、もう当初のオペコードをとっくに使い切ってしまったので、REX Prefix（64ビット拡張に関する接頭辞）を多用する羽目になっています。

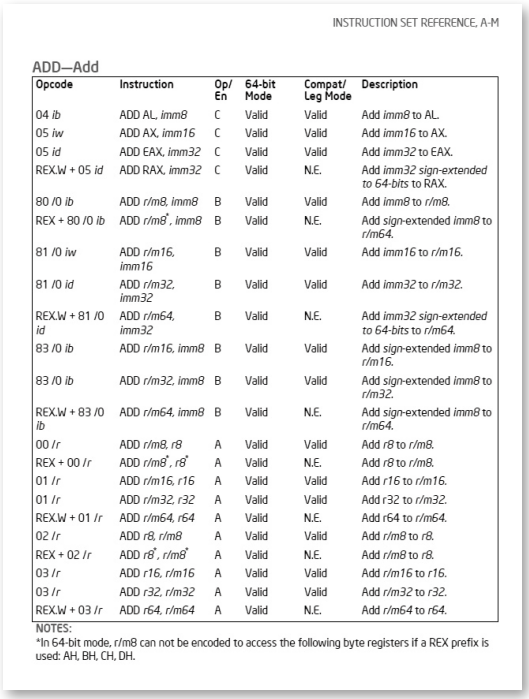
動作モード的に言うと、この64bitではLong ModeとLegacy Modeの2つのモードがあります。Legacy Modeは従来の80386と同じ動作（つまりプロテクトモード／仮想8086モード／リアルモードが動作する）を行うのに対してLong Modeでは64bitモードと互換モードが動作します。互換モードは、アプリケーション的にみ

れば従来の32bitのプロテクトモードが動作する(ただし仮想8086モードはない)というもので、既存のアプリケーションはここで動作します。64bitアドレスが利用できるのは64bitモードのみという形になります。

ソフトウェアからみた64bitモードの大きな違いは、レジスタがふんだんに使えるようになったことで、汎用レジスタとしてR8～R15が、XMMレジスタもXMM8～XMM15が使えるようになったことで、スタックをやたらめったら多用していた従来のプログラミングからやっと脱却できることになりました。

もっとも、x64プロセッサが世に出たころに一度コンパイラの生成コードを確認したところ、汎用レジスタが増えたためにサブルーチンコールでスタックを積む量が減ったかと思いきや、引き続き限られたレジスタを使いまわし続け、結果意味もなく使っていないR8～R15までスタックに積むためにかえって遅くなっていたりしたのはご愛嬌でしょうか。最近のコンパイラの出力は確認していないのですが、少しは最適化が進んだのでしょうか?

▼写真2 “Intel 64 and IA-32 Architectures Software Developer's Manual Volume 2A:Instruction Set Reference, A-M 253666-035US(June 2010)”より抜粋



INSTRUCTION SET REFERENCE, A-M

ADD—Add

Opcode	Instruction	Op/ En	64-bit Mode	Compat/ Leg Mode	Description
04 *ib*	ADD AL, *imm8*	C	Valid	Valid	Add *imm8* to AL.
05 *iw*	ADD AX, *imm16*	C	Valid	Valid	Add *imm16* to AX.
05 *id*	ADD EAX, *imm32*	C	Valid	Valid	Add *imm32* to EAX.
REX.W + 05 *id*	ADD RAX, *imm32*	C	Valid	N.E.	Add *imm32 sign-extended to 64-bits* to RAX.
80 /0 *ib*	ADD *r/m8, imm8*	B	Valid	Valid	Add *imm8* to *r/m8.*
REX + 80 /0 *ib*	ADD *r/m8**, *imm8*	B	Valid	N.E.	Add *sign*-extended *imm8* to *r/m64.*
81 /0 *iw*	ADD *r/m16, imm16*	B	Valid	Valid	Add *imm16* to *r/m16.*
81 /0 *id*	ADD *r/m32, imm32*	B	Valid	Valid	Add *imm32* to *r/m32.*
REX.W + 81 /0 *id*	ADD *r/m64, imm32*	B	Valid	N.E.	Add *imm32 sign-extended to 64-bits* to *r/m64.*
83 /0 *ib*	ADD *r/m16, imm8*	B	Valid	Valid	Add *sign*-extended *imm8* to *r/m16.*
83 /0 *ib*	ADD *r/m32, imm8*	B	Valid	Valid	Add *sign*-extended *imm8* to *r/m32.*
REX.W + 83 /0 *ib*	ADD *r/m64, imm8*	B	Valid	N.E.	Add *sign*-extended *imm8* to *r/m64.*
00 /r	ADD *r/m8, r8*	A	Valid	Valid	Add *r8* to *r/m8.*
REX + 00 /r	ADD *r/m8**, *r8**	A	Valid	N.E.	Add *r8* to *r/m8.*
01 /r	ADD *r/m16, r16*	A	Valid	Valid	Add *r16* to *r/m16.*
01 /r	ADD *r/m32, r32*	A	Valid	Valid	Add *r32* to *r/m32.*
REX.W + 01 /r	ADD *r/m64, r64*	A	Valid	N.E.	Add *r64* to *r/m64.*
02 /r	ADD *r8, r/m8*	A	Valid	Valid	Add *r/m8* to *r8.*
REX + 02 /r	ADD *r8**, *r/m8**	A	Valid	N.E.	Add *r/m8* to *r8.*
03 /r	ADD *r16, r/m16*	A	Valid	Valid	Add *r/m16* to *r16.*
03 /r	ADD *r32, r/m32*	A	Valid	Valid	Add *r/m32* to *r32.*
REX.W + 03 /r	ADD *r64, r/m64*	A	Valid	N.E.	Add *r/m64* to *r64.*

NOTES:
*In 64-bit mode, r/m8 can not be encoded to access the following byte registers if a REX prefix is used: AH, BH, CH, DH.

拡張命令

拡張命令では、SSSE3/SSE4.1/SSE4.2と世代ごとに少しずつ命令が増えてゆき、最近は新たにAVX/AVX2という命令が利用可能になりました。AVX/AVX2ではレジスタ幅が256bitに拡張され、同時に演算が行えるデータ量が従来の2倍になったほか、x86としては初めて3/4オペランドの命令もサポートされました。もっともこちらもまだ開発ツールはこれからといったところ。AVXに関してはすでに最新のVisual Studio 2013でサポートされていますが、AVX2に関しては今のところ「現在開発中」という以上のアナウンスがない状態です。

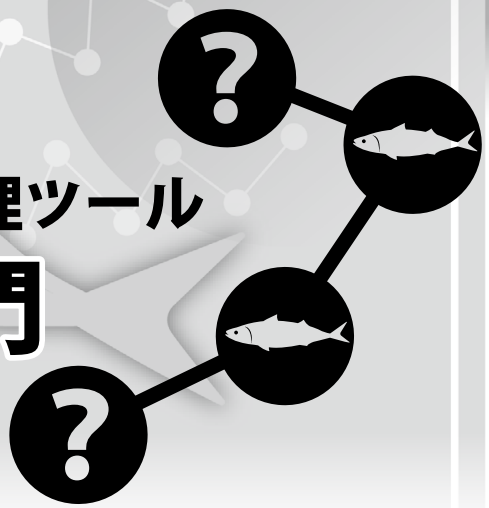
ちなみにこのPentium 4の世代では、L1キャッシュの代わりにTrace Cacheが搭載されました。これはパイプラインからデコード段が切り離され、これは単独でx86命令をMicroOpに変換し、そのMicroOpをTrace Cahceに格納するという方式です。

このTrace CacheはCore 2でいったん取りやめになりましたが、その後Core iの世代で復活し、最新のHaswellベースのCore iではL0 Cacheとでも言うべきサイズのTrace Cacheが実装されています。

加えてCore 2の世代からMicro-Fusion(複数のMicroOpを1つのものと考えてスケジュール・実行する)が、Core iの世代からはMacro-Fusion(複数のx86命令を1つのMicroOpに統合)なども実装されたことで、さらに最適化が複雑になっています。この手の最適化のガイドラインは、Intelがリリースする“Intel 64 and IA-32 Architecture Optimization Reference Manual”ですが、最新のもの(2014年9月リリースの248966-030)は実に642ページもの分量で、目を通すのも一苦労です。まぁすでにアセンブラで頑張るのではなく、コンパイラの最適化に期待すべし、ということでしょう。SD

はてな謹製、サーバ管理ツール
Mackerel 入門

Writer　田中 慎司（たなか しんじ）　㈱はてな



Mackerelは、複数のサーバ情報を一元管理できるサービス。CPU使用率、メモリ使用量などをわかりやすいグラフにして一覧できます。本記事では、Mackerelの始め方・基本的な使い方から、fluentd、AWSのCloudWatch、Hubot、Capistranoとの連携方法まで説明します。

Mackerelとは

Mackerel[注1]とは、1台から数千台までのサーバリソースやサービスパフォーマンスを管理できるサーバ管理Software as a Service（以下、SaaS）です。

サーバ管理は、オープンソースソフトですとNagiosやMunin、Zabbixが、SaaSですとNew Relic、AWS CloudWatchなどがよく知られています。

Mackerelの特徴は次の3つです。

- 簡単に使い始めることができる
- 1台から数千台までサーバ情報を一元管理できる
- サービスの情報を可視化、管理できる

「簡単に使えるけどスケールしない」「柔軟だけど使い難い」といったサービスとは異なり、Mackerelは複雑過ぎず、簡素過ぎず、Immutable Infrastructureにあったサーバ管理サービスを目指しています。

Mackerelの背景と考え方

Mackerelはサーバ管理SaaSですが、最近Mackerelのような、サービスの開発・運用を支援するための有償のSaaSが増えてきています。その背景として、SaaSを使うことで周辺ツールのセットアップ・自社開発・運用に割くリソースを削減でき、本来やるべきことであるサービス開発に集中できる、という効果があります。これにより、開発リソース配分の効率化とサービス開発のスピードの向上を狙うことができます。

ここ数年使われるようになってきた開発支援のSaaSの活用例として、ソースコードリポジトリの管理に**GitHub**[注2]を利用したり、チーム内コミュニケーションを**Qiita Team**[注3]や**Slack**[注4]で行うなどが挙げられます。従来は、それぞれGit repository、Redmine、IRCサーバなどを自前で管理していたのですが、これらのSaaSによりエンジニアリソースをサービス開発に集中できるようになります。

またサービスの実行環境も、オンプレミスからAmazon Web Services（以下、AWS）のようなクラウドに移行する流れは加速しており、Mackerelはクラウドのような、サーバを簡単に増減させられる環境下でも効率的にサーバ管理できるSaaSとして開発しています。

● はてなのサーバ管理ツールが原点

Mackerelは、2007年から㈱はてなが社内で開発・利用してきたサーバ管理ツールが基になっています。当時、はてなではサーバ台数が100台を越え、テキストによる手動での管理が限界になりつつありました。サーバ管理ツールは、それを効率的に管理

注1）URL https://mackerel.io/

注2）URL https://github.com/
注3）URL https://teams.qiita.com/
注4）URL https://slack.com/

し、リソース状況の可視化、監視(Nagios)との連携、デプロイツール(Capistrano)との連携などを実現するために開発されてきました。

● ロール(役割)でサーバをまとめて管理

はてなのサーバ管理ツールでは、「はてなブログ」や「はてなブックマーク」などのサービスを「**サービス**」という概念でひとまとめにし、サービスの中のアプリケーションサーバやデータベースサーバを「**ロール**(役割)」という概念でまとめていました。

またサーバ情報をAPI経由で取り出せるようにし、デプロイツールのCapistranoや、DNSと連携させ、便利に利用できるようにしました。たとえば、はてなブログのサーバ群にCapistranoを使ってアクセスする場合、設定ファイルにIPアドレスリストを書いておくのではなく、サーバ管理ツールからAPI経由で、対象となるロールに属するサーバのIPアドレス一覧を取得していました。

「サービス」「ロール」でサーバを整理することで、台数が増えてきた場合も把握しやすくしています。

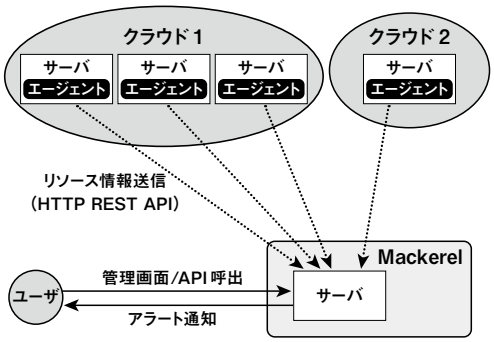
アーキテクチャ

Mackerelを使うためにはエージェント(**mackerel-agent**)を管理対象の各サーバで起動する必要があります(**図1**)。mackerel-agentは自身が動いているサーバのリソース情報(CPU使用率、メモリ使用量など)を取得し、サーバへ送信します。

また後述するように、プラグインによってミドルウェアなどのメトリック情報も収集・送信できます。ユーザは、mackerel-agentから送信された情報を中央サーバからブラウザで閲覧したり、API経由で取得したりできます。

このmackerel-agentはGo言語で記述されており、依存するライブラリがなく、さまざまな環境で動かすことができます。また、オープンソースとしてGitHub上に公開[注5]されています。mackerel-agentが利用するAPIはすべてHTTPによるREST APIとなっており、これらの仕様も公開[注6]されています。

▼図1　アーキテクチャ



▼図2　トップページ

このREST APIを利用して、mackerel-agent以外からメトリック情報を送信することもできます。fluentdによる送信方法は後述します。

Mackerelを使う

さっそく、Mackerelを使ってみましょう。本稿では、**Ubuntu 14.04**を対象とします。

ユーザ登録

まずユーザ登録を行います。トップページ(**図2**)の「無料アカウントを作成」ボタンからサインアップページ(**図3**)に移動し、ユーザ登録をしてください。

サインアップすると簡単なチュートリアルがあり、Trialプランを使うと無料で全機能を試用できます。登録が終わると**図4**のようなダッシュボードが表示されます。この時点ではまだサーバを登録していませんので、空となっています。

注5) URL https://github.com/mackerelio/mackerel-agent
注6) URL http://help-ja.mackerel.io/entry/spec/api/v0

エージェントセットアップ

では次にmackerel-agentをセットアップしてみましょう。セットアップの説明（**図5**）は、ダッシュボードページの左サイドバーの下にある「新規ホストの登録」リンクから参照できます。

ここではUbuntu 14.04を想定します。まず、次のコマンドでリポジトリを登録し、エージェントと公式プラグイン集をインストールします。

```
$ curl -fsSL https://mackerel.io/assets/ ➘
files/scripts/setup-apt.sh | sh
$ sudo apt-get install mackerel-agent ➘
mackerel-agent-plugins
```

次にAPIキーを**/etc/mackerel-agent/mackerel-agent.conf**に追記します。コマンド中の「<APIキー>」部分は、実際に利用するAPIキーに書き換えてください。APIキーはダッシュボードページの「**Detail**」から確認できます。

```
$ sudo sh << SCRIPT
cat >>/etc/mackerel-agent/mackerel-agent. ➘
conf <<'EOF';
apikey = "<APIキー>"
EOF
SCRIPT
```

そしてmackerel-agentを起動します。

```
$ sudo service mackerel-agent start
```

エージェントが起動すると、サーバの構成情報や動作状況がMackerelのサーバに送信されるようになります。動作の様子は**/var/log/mackerel-agent.log**に出力されるログで確認できます。エラーなど不具合が発生していなければ、**図6**のようなログが出力されます。

サービスとロールの割り当て

サービスとロールの指定は、ダッシュボードやホスト一覧ページなど、ホストの情報が表示されている個所で行えます。ロールが未設定のホストには「ロールを設定」ボタンが表示されていますので、ここから設定します。

またmackerel-agentの設定ファイルで、サービスとロールの指定を行うこともできます。たとえば、「My-ServiceサービスのappロールおよびAnother-Serviceサービスのdbロールを紐付けたい」場合、

▼図3　サインアップページ

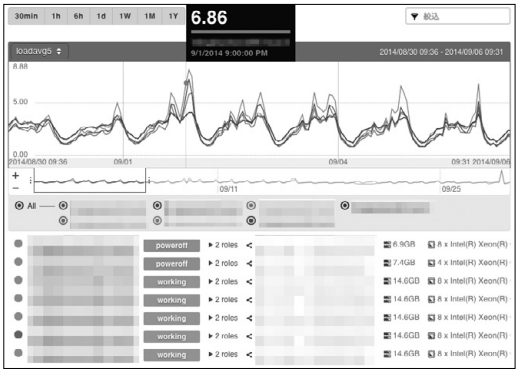
▼図4　登録直後のダッシュボード

▼図5　セットアップの説明

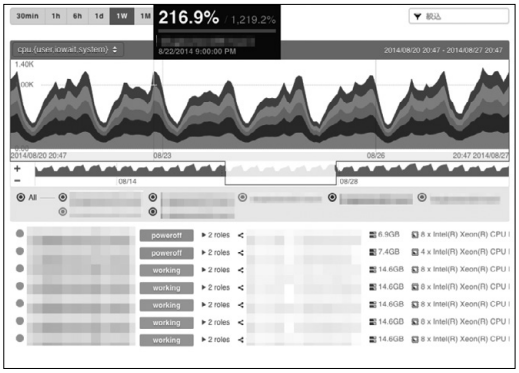
▼図6　ログの例

```
2014/09/21 03:52:52 INFO main Starting ➘
mackerel-agent version:0.12.2, rev:67cbc69
2014/09/21 03:52:53 INFO command Start: ➘
apibase = https://mackerel.io, hostName = ➘
someserver, hostId = 28ghQyapUef
```

▼図7　ロードアベレージのリソースグラフ
縦軸：待ちプロセス数、横軸：時間



▼図8　CPU使用率のリソースグラフ
縦軸：CPU%、横軸：時間



/etc/mackerel-agent/mackerel-agent.confに次の行を追記してください。

```
roles = [ "My-Service:app", "Another-Service:db" ]
```

この方法は**Chef**[注7]や**Puppet**[注8]などのプロビジョニングツールで設定ファイルを自動生成する場合に便利です。

指定されたサービス、ロールが存在しない場合は新たに作成されます。また、この設定・紐付けはエージェント起動時に処理されます。

リソースグラフの使い方

● 基本的な見方

リソースグラフはサービス詳細ページとホスト詳細ページで参照できます。

サービス詳細ページは、そのサービスに属する全サーバの情報がロールごとに1つのグラフにまとめられており、一覧できます。サービス詳細ページを開くとロール単位にまとめられたロードアベレージ（**loadavg5**）が重ね合わせられたリソースグラフが表示されていることが確認できます（**図7**）。このグラフでは、重ね合わせグラフと積み重ねグラフがあり、グラフの左上のプルダウンメニューから表示するメトリックを切り替えることができます。選択したメトリックによって、重ね合わせか積み重ねかが決定されます。重ね合わせグラフでは、同じロールに属する複数のサーバのメトリックを重ね合わせて見ることで、一部のサーバの挙動がほかのサーバと異なる場合に簡単に発見できます。

プルダウンメニューから**cpu.{user,iowait,system}**を選択すると各サーバのCPU使用率が積み上げられている様子が見られます。ここではそのロールに属するサーバ群が合計でどれほどのCPUを使っているか、確認できます（**図8**）。グラフ上にマウスカーソルを合わせるとポップアップでそのサーバのCPU使用率（**図8**では216.9%）とロール全体の合計値（**図8**では1,219.2%）が表示されます。

サービス詳細ページでは、ロードアベレージ、CPU使用率（user、iowait、system）、ネットワーク転送量（送信、受信）、ディスクI/O（読み込み、書き込み）、メモリ使用量（used、cached）のグラフを見ることができます。積み重ねには、CPU使用率（サーバごとにuser、iowait、systemをまとめたもの）、ネットワーク転送量（送信、受信）、ディスクI/O（読み込み、書き込み）、メモリ使用量が利用できます。

● キャパシティプランニングへの応用

リソースグラフを利用すれば、キャパシティプランニングに応用できます。

図8の場合、「AWS EC2のm3.2xlarge」が5台あり、それぞれ仮想コアが8つですので、ロール全体で40コア、4,000%のCPUリソースがあります。マ

注7）URL https://www.getchef.com/
注8）URL http://puppetlabs.com/

ウスカーソルを合わせた時間帯では、その内の1,219.2%、最大値の全体の1/4強のリソースを消費していることがわかります。これがピークの時間帯ですので、CPU使用率はだいぶ余裕があることがわかります。

サービスの成長率やリクエストの変動の激しさ度合いによって適切な余裕は変わってきます。実際にはロードアベレージやメモリ使用量、場合によってはネットワーク帯域幅も確認しつつ、調整する必要があります。

● オートスケールへの対応

AWSなどのクラウドの環境では、負荷に応じてサーバ台数を増減させるオートスケールを利用することが広く行われるようになってきています。

オートスケールを行った場合、Mackerelでは**図9**のようなグラフになります。このグラフでは、時間によってサーバが増減（たとえば11:30ごろに1台増え、16:30ごろに2台減っています）していることがわかります。オートスケールにより削除されたサーバのリソース状況を確認することは、一般的なサーバ管理ツールではなかなか難しいのですが、Mackerelでは簡単に可視化でき、オートスケールの設定が適切かどうか確認できます。

ミドルウェアのメトリックを可視化する

MackerelではWebサーバやデータベースなどのミドルウェアのメトリックも可視化できます。

? カスタムプラグインを使う

Mackerelでは任意の時系列データをカスタムメトリックとして投稿・可視化でき、ミドルウェアのメトリックを可視化するために、このしくみを利用します。mackerel-agentでカスタムメトリックを投稿するには、プラグインを利用します。

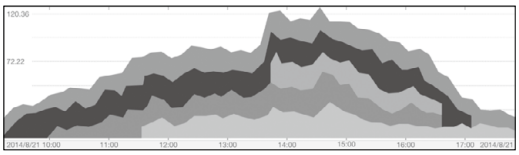
プラグインは実行可能なコマンドで、標準出力の各行に次のフォーマットの出力をすることが期待されます（「\t」はタブ文字です）。

```
{metric name}\t{metric value}\t{epoch seconds}
```

名前の最後のドットまでが共通するメトリックが1つのグラフにまとめられ、ホスト詳細ページで閲覧できます。mackerel-agentはカスタムメトリックを識別するために、メトリック名の先頭に自動的に**"custom."**という文字列を付与します。

このようなコマンドをミドルウェアごとに作成する必要がありますが、主要なミドルウェア（Nginx、Redis、MySQLなど）のプラグインをパッケージにまとめた**mackerel-agent-plugins**[注9]、カスタムプラグインを簡単に自作するためのライブラリ**go-mackerel-agent-helper**[注10]をそれぞれGitHubで公開しています。

▼図9　オートスケール実行時のリソースグラフ
縦軸：CPU%、横軸：時間



Mackerelでサーバを監視する

Mackerelではメトリックの監視もできます。

? 監視項目の指定

メトリックを監視するには、まず監視ルール（Monitor）を作成します（**図10**）。Monitorは名前、監視対象メトリック、Warning条件、Critical条件、持続時間を設定します。Mackerelは毎分送信されるメトリックを観測し、持続時間の平均値がWarning条件もしくはCritical条件を越えた場合に、アラートを生成します。

? 通知先の設定

生成されたアラートを通知するにはチャンネルを作成します（**図11**）。2014年9月時点では、メールと

注9）URL https://github.com/mackerelio/mackerel-agent-plugins
注10）URL https://github.com/mackerelio/go-mackerel-plugin

Webhookのチャンネルを作成できます。

メールは、そのオーガニゼーションに所属するユーザのメールアドレスにアラート通知を送信します。Webhookは、アラートが生成された際に指定されたURLにアラートの詳細をJSONでPOSTします（後述のHubot連携を行うことでSlackやHipChat、IRCに通知できます）。

Mackerelの応用

fluentdでサービスメトリックを登録する

サービスメトリックは、特定のサーバとは直接紐付かないメトリックを可視化、監視するためのしくみです。たとえば、あるWebサービス全体のレスポンスタイムや、ステータスコードの分布を可視化し、レスポンスが遅くなったり、エラー率が上昇していないか確認できます。

このサービスメトリックは**fluent-plugin-mackerel**[注11]を利用することで**fluentd**[注12]経由で投稿できます。

● セットアップ

まずfluentdとfluentd-plugin-mackerelをセットアップします。

```
$ gem install fluentd fluent-plugin-mackerel
```

● サービスメトリックを投稿する

fluent-plugin-mackerelでサービスメトリックを投稿するためのfluentdの設定は**リスト1**になります。

● ホストのカスタムメトリックを投稿する

fluentd-plugin-mackerelはホストのカスタムメトリックを投稿することもできます。

ホストID[注13]を直接指定する場合の設定は**リスト2**のようになります。

▼図10　Monitorの作成画面

▼図11　チャンネルの作成画面

▼リスト1　サービスメトリック投稿のための設定

```
<match ...>
  type mackerel
  api_key <投稿先オーガニゼーションのAPIキー>
  service <投稿先のサービス名>
  metrics_name http_status.${out_key}
  out_keys <投稿するメトリックのキー。例:2xx_count,3xx_count,4xx_count,5xx_count>
</match>
```

▼リスト2　ホストIDを設定ファイルに指定

```
<match ...>
  type mackerel
  api_key <投稿先オーガニゼーションのAPIキー>
  hostid xyz
  metrics_name http_status.${out_key}
  out_keys <投稿するメトリックのキー。例:2xx_count,3xx_count,4xx_count,5xx_count>
</match>
```

注11）URL https://github.com/tksmd/fluent-plugin-mackerel
注12）URL http://www.fluentd.org/
注13）個々のホストを識別するためのID。詳細は用語集（URL http://help-ja.mackerel.io/entry/glossary#host）を参照

また、ホストIDが格納されているファイルを指定することもできます(**リスト3**)。mackerel-agentが動作しているホストから直接Mackerelに投稿する場合に便利です。

● Nginxのステータスコードごとリクエスト数をグラフ化

Nginxのアクセスログからサービスメトリックを投稿する例を紹介します。ここでは**図12**のようなステータスコード別にリクエスト数がカウントされたグラフを作ります。

まずNginxからアクセスログを「LTSV形式」[注14]で出力するようにします(**リスト4**)。

また、ステータスコードごとにリクエスト数を数えるために**fluent-plugin-datacounter**を利用しますので、このプラグインをインストールします。

```
$ gem install fluent-plugin-datacounter
```

そしてNginxのアクセスログを読み込み、集計し、Mackerelへの投稿を行うfluentdの設定(**リスト5**～**7**)を用意します。これにより**図12**のようなグラフが描画されるようになります。またグラフの右上のギアマークからグラフの設定を行うことができ、たとえば、重ね合わせグラフから積み重ねグラフ(**図13**)に変えることができます。

? AWSのCloudWatchと連携

AWSのCloudWatchではロードバランサのElastic Load Balancingのリクエスト数(全体とステータスコード)、応答時間といったメトリックデータを取得できます。それらをサービスメトリックとして投稿すると**図14**のようなグラフが得られます。

AWSのCloudWatchからメトリックデータを読み込むために**fluent-plugin-cloudwatch**を利用しますので、このプラグインをインストールします。

```
$ gem install fluent-plugin-cloudwatch
```

そしてCloudWatchからELBの情報を取り出し、

注14) URL http://ltsv.org

▼リスト3　ホストIDが格納されているファイルを指定

```
<match ...>
  type mackerel
  api_key <投稿先オーガニゼーションのAPIキー>
  hostid_path /var/lib/mackerel-agent/id
  metrics_name http_status.${out_key}
  out_keys <投稿するメトリックのキー。例:
  2xx_count,3xx_count,4xx_count,5xx_count>
</match>
```

▼リスト4　Nginxのアクセスログ書式設定

```
log_format ltsv "time:$time_local"
                "\thost:$remote_addr"
                "\tforwardedfor:$http_x_forwarded_for"
                "\treq:$request"
                "\tstatus:$status"
                "\tsize:$body_bytes_sent"
                "\treferer:$http_referer"
                "\tua:$http_user_agent"
                "\treqtime:$request_time"
                "\tcache:$upstream_http_x_cache"
                "\truntime:$upstream_http_x_runtime"
                "\tvhost:$host";

access_log  /var/log/nginx/access.log  ltsv;
```

▼リスト5　LTSV形式のログファイルを読み込む

```
<source>
  type tail
  format ltsv
  time_format %d/%b/%Y:%H:%M:%S %z
  path /var/log/nginx/access.log
  pos_file /var/log/nginx/access_log.pos
  tag access.nginx
</source>
```

▼リスト6　fluent-plugin-datacounterでステータスコード別に集計

```
<match access.nginx>
  type datacounter
  count_interval 1m
  count_key status
  aggregate all
  tag nginx.status
  pattern1 2xx ^2\d\d$
  pattern2 3xx ^3\d\d$
  pattern3 4xx ^4\d\d$
  pattern4 5xx ^5\d\d$
</match>
```

▼リスト7　fluent-plugin-mackerelによりサービスメトリックを投稿

```
<match nginx.status.**>
  type mackerel
  api_key <投稿先オーガニゼーションのAPIキー>
  service <投稿先サービス名>
  metrics_name access_num.${out_key}
  out_keys 2xx_count,3xx_count,4xx_count,
  5xx_count
</match>
```

Mackerelへの投稿を行うfluentdの設定(**リスト8**、**リスト9**)を用意します。

Hubotと連携

Hubot[注15]スクリプトの**hubot-mackerel-notifier**[注16]を使うと、Mackerelからアラート通知のWebhookを受けとり、**IRC**や**Slack**、**HipChat**などのチャットツール(**図15**)に次のような通知を流せます。

```
17:06 hubot: [Mackerel] CRITICAL: IOwait at app01 (working) Service: app https://mackerel.io/orgs/.../alerts/...
```

またホスト名に反応してホスト詳細ページへのリンクを表示させることができます(**図16**)。

Webhookのリクエスト先URLのパスは**/hubot/mackerel**を設定してください。たとえば、Hubotを「**http://some-hubot.herokuapp.com/**」にセットアップしている場合、WebhookのURLは「**http://some-hubot.herokuapp.com/hubot/mackerel**」とします。WebhookのURL設定の詳細については、「監視・通知を設定する」[注17]を参照してください。

Hubot本体の詳細やインストール方法については、公式ドキュメント[注18]や解説記事[注19]などを参照してください。

● Hubotにhubot-mackerel-notifierを追加する

手順は次のとおりです。

① hubot-mackerel-notifierをインストール

```
$ npm install hubot-mackerel-notifier --save
```

② hubot-mackerel-notifierをexternal-scripts.jsonに次のように追加

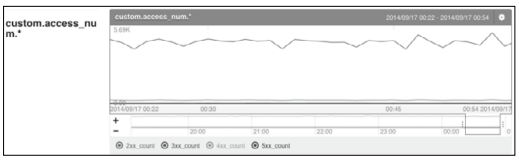
```
["hubot-mackerel-notifier"]
```

③ リポジトリにコミット

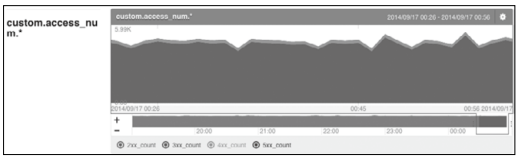
```
$ git commit -a -m 'add hubot-mackerel-notifier'
$ git push
```

注15) URL https://hubot.github.com/
注16) URL https://github.com/mackerelio/hubot-mackerel-notifier
注17) URL http://help-ja.mackerel.io/entry/howto/alerts
注18) URL https://github.com/github/hubot/tree/master/docs
注19) URL http://gihyo.jp/dev/serial/01/hubot/0001

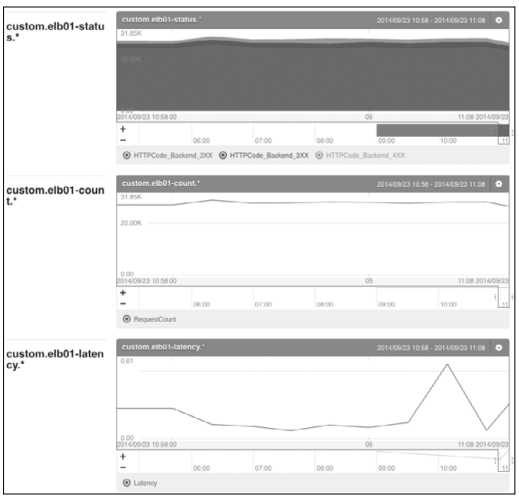
▼図12　Nginxにおけるステータスコード別リクエスト数グラフ



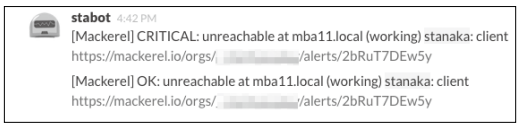
▼図13　「図12」を積み重ねグラフへ変更



▼図14　AWSのCloudWatch　上から
リクエスト数(ステータス)、リクエスト数(全体)、応答時間



▼図15　Hubotの通知



▼図16　ホスト詳細ページへのリンクを通知

④ 図17のように環境変数を設定
(例：Herokuの場合)

⑤ Hubotにデプロイ
(例：Herokuの場合)

```
$ git push heroku master
```

Capistranoと連携

Mackerelに登録しているホストの情報を、「Capistrano 2.x」[注20]のデプロイ対象ホストとして利用します。これにより、ホスト増減のたびにCapistranoの設定ファイルに書かれたホスト一覧を変更する必要がなくなります。

mackerel-client[注21]を利用しますのでGemfileに追記します。

```
gem 'mackerel-client'
```

またMackerel連携を行うための設定（リスト10）をconfig/deploy.rb（Capfile）に書きます。リスト10中の「<APIキー>」、「<サービス名>」にはMackerelのAPIキーと対象とするサービスの名前を指定してください。Capistranoのロールとし

注20）URL https://github.com/capistrano/capistrano/wiki
注21）URL http://rubygems.org/gems/mackerel-client

▼リスト8　fluent-plugin-cloudwatchでCloudWatchからデータを取得

```
<source>
  type cloudwatch
  tag  cloudwatch-latency.elb01
  aws_key_id <aws-key>
  aws_sec_key <aws-sec-key>
  cw_endpoint monitoring.ap-northeast-1.amazonaws.com

  namespace AWS/ELB
  metric_name Latency
  dimensions_name LoadBalancerName
  dimensions_value elb01
  interval         60
  period           60
</source>

<source>
  type cloudwatch
  tag  cloudwatch-status.elb01
  aws_key_id <aws-key>
  aws_sec_key <aws-sec-key>
  cw_endpoint monitoring.ap-northeast-1.amazonaws.com

  namespace AWS/ELB
  statistics Sum
  metric_name HTTPCode_Backend_2XX,HTTPCodeBackend_3XX, ➥
  HTTPCode_Backend_4XX,HTTPCode_Backend_5XX,RequestCount
  dimensions_name LoadBalancerName
  dimensions_value elb01
  interval         60
  period           60
</source>
```

▼リスト9　Mackerelへサービスメトリックとして投稿する

```
<match cloudwatch-status.*>
  type copy
  <store>
    type             mackerel
    api_key          <api-key>
    service          Blog
    metrics_name     ${[1]}-status.${out_key}
    out_keys         HTTPCode_Backend_2XX,HTTPCode_Backend_3XX,HTTPCode_Backend_4XX, ➥
                     HTTPCode_Backend_5XX
  </store>
  <store>
    type             mackerel
    api_key          <api-key>
    service          Blog
    metrics_name     ${[1]}-count.${out_key}
    out_keys         RequestCount
  </store>
</match>

<match cloudwatch-latency.*>
  type             mackerel
  api_key          <api-key>
  service          Blog
  metrics_name     ${[1]}-latency.${out_key}
  out_keys         Latency
</match>
```

て、Mackerelでサービスに所属しているホスト群をMackerelのロールの名前で指定できます。

リスト10の設定では、ステータスが「working」または「standby」になっているホストが一覧できます。ホストのIPアドレスはエージェントが収集したものが使用されます。

ここで**図18**のようなタスクをCapistranoで定義します。このタスクを実行すると**図19**に示したような結果となります。

おわりに

本稿では、サーバのリソースやサービスのパフォーマンスを管理できるサーバ管理SaaSのMackerelの基本的な使い方を紹介しました。

Mackerelを利用することで、簡単かつ拡張性の高いサーバ管理が可能となります。まだリリースされたばかりのサービスですので、ぜひ、今後も開発に注目してください。SD

▼図17　環境変数を設定

```
$ heroku config:add HUBOT_MACKEREL_API_KEY="..."
$ heroku config:add HUBOT_MACKEREL_HOST_REGEXP="\\b[a-zA-Z0-9._-]+[.]local-domain\\b"
$ heroku config:add HUBOT_MACKEREL_ORG_NAME="example"
$ heroku config:add HUBOT_MACKEREL_NOTIFIER_ROOM="#general"
```

▼図18　タスク

```
desc "Mackerel integration test"
namespace :deploy do
  desc "echo message"
  task :echo , roles => :app do
    run "echo \"Integration Test\""
  end
end
```

▼図19　タスク実行結果

```
# bundle exec cap deploy:echo
  * 2014-09-23 21:36:28 executing `deploy:echo'
  * executing "echo \"Integration Test\""
    servers: ["127.0.0.1"]
    [127.0.0.1] executing command
 ** [out :: 127.0.0.1] Integration Test
    command finished in 435ms
```

▼リスト10　Mackerel連携を行うための設定

```
load 'deploy'

require 'mackerel/client'

set :mackerel_api_key, <APIキー>
set :service, <サービス名>

@client = Mackerel::Client.new(mackerel_api_key: mackerel_api_key)

def host_ip_addrs(role)
  hosts = @client.get_hosts(service: service, roles: role).select do |host|
    host.status === 'standby' || host.status === 'working'
  end.map do |host|
    interface = host.interfaces.find { |i| /^eth/ === i['name'] }
    interface['ipAddress'] if interface
  end.select {|ipaddr| ipaddr != nil }
end

role :app do
  host_ip_addrs(:app)
end
```

短期集中連載

Jamesのセキュリティレッスン

パケットキャプチャWiresharkの新展開

第1回 ファイル形式のpcapとpcap-ngって何が違うの?

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、お久しぶりです。Eiji James Yoshidaです。本誌に書くのは久しぶりでSoul Trainに出たDavid Bowie並み[注1]にアウェイ感たっぷりですが、めげずにマイペースでいきたいと思います。

さて、2012年6月に公開されたWireshark 1.8.0からデフォルトのファイル形式がpcap-ngになりましたが、皆さんはpcapとpcap-ngのどちらでキャプチャファイルを保存していますか?

デフォルトになったことにより筆者のまわりでもpcap-ngで保存する人は増えていますが、pcapとpcap-ngのファイル形式で何が違うのかは知らないという人は今も多いと思います。そこで本稿ではpcapとpcap-ngの各ファイル形式について解説するとともに、実際にキャプチャファイルを解析しながらpcapとpcap-ngのファイル形式の違いについて解説します。

まずは解析環境の準備

本稿では7以降のWindowsを使ってキャプチャファイルを解析しますが、バイナリエディタが使えれば、ほかのOSでもかまいません。使用するツールは、hexedit 4.0[注2]です。

インストール方法はお任せしますが、とくにこだわりがない場合はデフォルトでインストールしてください。インストールが終わったらhexeditを起動して[View]メニューの[Toolbars]を選び、[Standard Toolbar]、[Edit Bar]、[Bookmarks]、[Calculator]にチェックを入れます(**図1**)。使い方は省略しますが16進数を10進数に変換したいときや、各フォーマットをブックマークして移動したいときなどに活用してください。

あとは筆者のブログ「Eiji James Yoshidaの記録[注3]」からarp.pcapとarp.pcapngをダウンロードしてください。

これで解析環境の準備は終了です。ここから先は取扱説明書を読むような感じなので、できればBGMでも流しながら読んでください。ちなみに筆者はDEPECHE MODE[注4]の「Welcome To My World」や「Soothe My Soul」とか聴きながら本稿を書いています。

それでは最初にpcapのファイル形式を見てみましょう。

注1) アメリカの70年代の音楽番組「Soul Train」に白人として初めてデビッド・ボウイが出演したことから。

注2) http://www.hexedit.com/

注3) http://d.hatena.ne.jp/EijiYoshida/20140907/1410071296

注4) デペッシュ・モードは80年代の大物バンド……以下略(http://www.sonymusic.co.jp/artist/depechemode/)。

pcapのファイル形式とは具体的にどのようなものか?

pcapファイル形式(.pcap)といっても種類が複数存在しますが、本稿ではtcpdumpなどで使われている最も基本的なLibpcap(以降pcapと省略)のファイル形式について解説します。

pcapファイル形式についてはLibpcap File Formatのサイト[注5]に詳細がありますので参考にしてください。pcapは**図2**のようなファイル形式になっています。

図3を見るとわかるようにpcapファイル形式は次の3種類のブロックで構成されています。

・グローバルヘッダ[24バイト]
・パケット情報ヘッダ[16バイト]
・パケットデータ[可変長]

pcapファイルに複数のパケットが記録されている場合は、その数だけパケット情報ヘッダとパケットデータのブロックが存在します。

またpcap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3**になります。

まずは各ブロックについて解説しましょう。

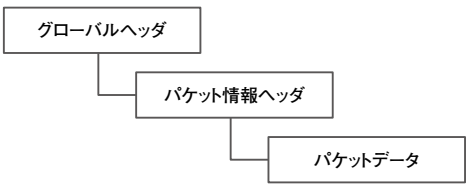
グローバルヘッダ・ブロック

pcapファイルの先頭から24バイト目まではグローバルヘッダのブロックです。

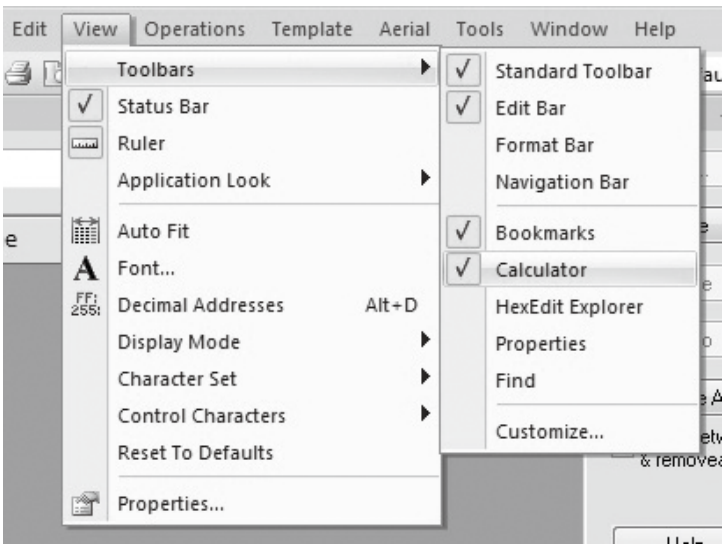
このブロックは**図4**のような7種類のフィールドで構成されていて、pcapファイルの情報が記録されています。

・マジックナンバー[4バイト]
・バージョン番号(メジャー)[2バイト]
・バージョン番号(マイナー)[2バイト]
・タイムゾーンオフセット[4バイト]
・タイムスタンプ精度[4バイト]
・キャプチャリミット[4バイト]
・データリンクタイプ[4バイト]

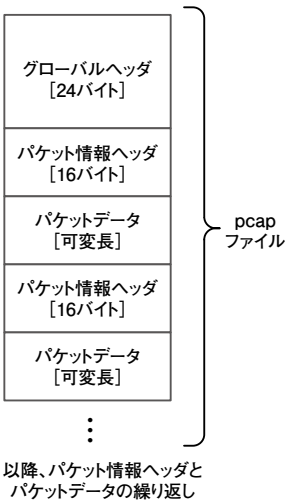
▼図3　pcapファイル形式におけるブロックの論理的な階層構造



▼図1　hexeditのViewメニュー

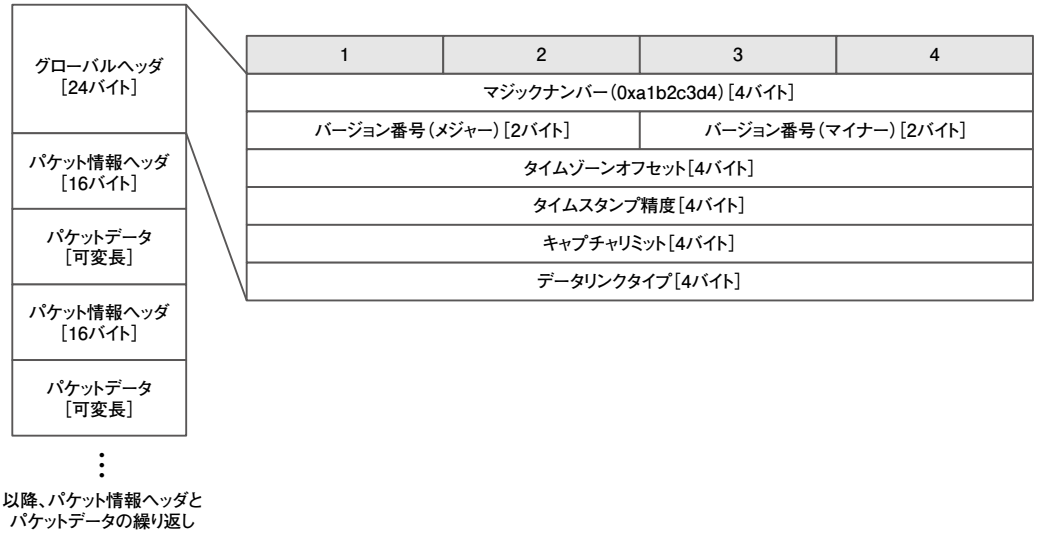


▼図2　pcapのファイル形式



注5) http://wiki.wireshark.org/Development/LibpcapFileFormat

▼図4　グローバルヘッダ・ブロックの構造



● マジックナンバー・フィールド

ファイル形式を識別するための値が入るフィールドで、pcapの場合は0xa1b2c3d4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。拡張子がpcapなのにマジックナンバーが異なる場合は、別のpcapファイル形式の可能性が高いです。

● 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pcapファイル形式のバージョンは2.4のためメジャー部分には0x0002、マイナー部分には0x0004が設定されます。

● タイムゾーンオフセット・フィールド

GMT（UTC）とローカルタイムゾーンの差を補正するために用意されたフィールドですが、現時点で使われることはなく常に0x00000000が設定されます。

● タイムスタンプ精度・フィールド

タイムスタンプの精度に関する値が入るフィールドですが、ここも使われることはなく、常に0x00000000が設定されます。

● 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。

● 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を指定する値を記録するフィールドです。代表的なデータリンクタイプの値を**表1**にまとめておきます。

パケット情報ヘッダ・ブロック

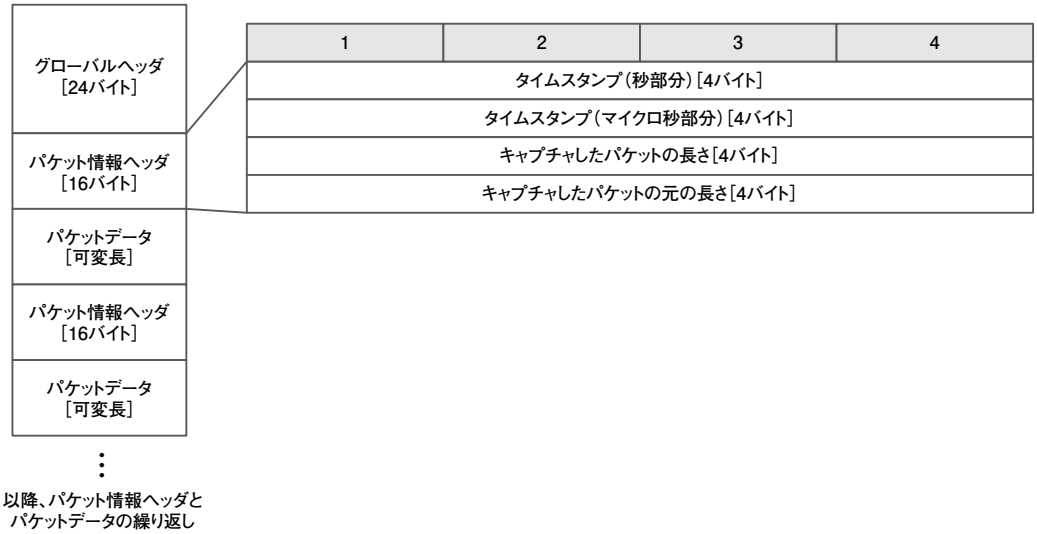
グローバルヘッダ・ブロックの最後から16バイト目まではパケット情報ヘッダのブロックです。

このブロックは**図5**のような4種類のフィー

▼表1　代表的なデータリンクタイプの値

リンクタイプコード	リンクタイプ
0x00000001	IEEE 802.3 Ethernet
0x00000009	PPP
0x00000069	IEEE 802.11 Wireless
0x00000071	Linux cooked socket capture（SLL）

▼図5　パケット情報ヘッダ・ブロックの構造



ルドで構成されていて、直後にあるパケットデータ・ブロックのパケットをキャプチャした時間(タイムスタンプ)などが記録されています。

・タイムスタンプ(秒部分)[4バイト]
・タイムスタンプ(マイクロ秒部分)[4バイト]
・キャプチャしたパケットの長さ[4バイト]
・キャプチャしたパケットの元の長さ[4バイト]

● タイムスタンプ・フィールド

秒部分とマイクロ秒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時間を秒部分とマイクロ秒部分に分けて記録しています。

● キャプチャしたパケットの長さ・フィールド

パケット情報ヘッダの後ろに続くパケットデータの長さ(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・ブロックに記録するため、パケットのもともとの長さとパケットデータ・ブロック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・ブロック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って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・ブロック

パケット情報ヘッダ・ブロックの最後から「キャプチャしたパケットの長さ」で指定されたバイト数まではパケットデータのブロックです。

このブロックには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されて記録されます。

pcapファイル形式について理解が深まったと思いますので、実際にpcapファイルを解析してみましょう。

pcapファイルの解析

hexeditを起動してarp.pcapをドラッグ&ド

ロップすると、ファイルの内容が16進数で表示されます。

このままでは解析し辛いので、4バイトで折り返すように設定を変更します。16進数が表示されているペインの上で右クリックをして、[Options]をクリックします。[HexEdit Options]ウィンドウが表示されたら、右側ペインにある[Layout]の[Columns:]の値を[4]に変更して、[OK]ボタンをクリックします(**図6**)。これで4バイトで折り返すようになります。

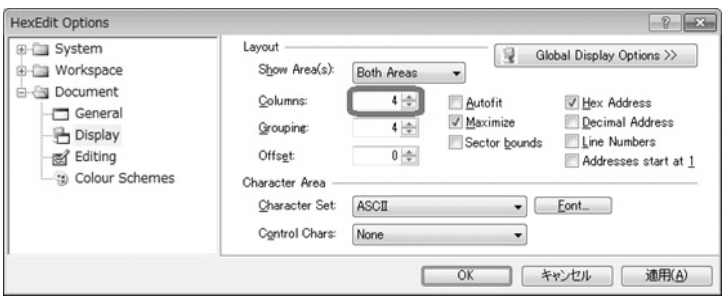
まずは**図2**を参照しながらarp.pcapの16進数表示をブロックに分けると、**図7**になります。パケットデータ・ブロックは可変長で今は範囲がわからないので書いていません。

▼図7　arp.pcapの各ブロック

	00	01	02	03	
00:	D4	C3	B2	A1	
04:	02	00	04	00	
08:	00	00	00	00	
0c:	00	00	00	00	←グローバルヘッダ・ブロック
10:	00	00	04	00	
14:	01	00	00	00	
18:	56	01	0C	54	
1c:	95	68	04	00	
20:	2A	00	00	00	←パケット情報ヘッダ・ブロック
24:	2A	00	00	00	
28:	FF	FF	FF	FF	

（以下省略）

▼図6　HexEdit Optionsウィンドウ



グローバルヘッダ・ブロックの解析

グローバルヘッダ・ブロックは固定長で24バイトとわかっているので、**図4**を参照しながらフィールドに分けると**図8**になります。

● マジックナンバー・フィールドの解析

マジックナンバー・フィールドの値が0xA1B2C3D4ではなく0xD4C3B2A1と並びが逆になっていることから、バイトオーダーがリトルエンディアンなことがわかります。以降、パケットデータ・ブロックを除いてリトルエンディアンと判断します。

● バージョン番号・フィールドの解析

バージョン番号・フィールドは0x0002と0x0004なので、2.4となります。

● タイムゾーンオフセット・フィールドの解析

タイムゾーンオフセット・フィールドは0x00000000なのでデフォルトのままです。

▼図8　arp.pcapのグローバルヘッダ・ブロック

	00	01	02	03	
00:	D4	C3	B2	A1	←マジックナンバー・フィールド
04:	02	00	04	00	←バージョン番号・フィールド
08:	00	00	00	00	←タイムゾーンオフセット・フィールド
0c:	00	00	00	00	←タイムスタンプ精度・フィールド
10:	00	00	04	00	←キャプチャリミット・フィールド
14:	01	00	00	00	←データリンクタイプ・フィールド

● タイムスタンプ精度・フィールドの解析

タイムスタンプ精度・フィールドは0x00000000なので、ここもデフォルトのままです。

● キャプチャリミット・フィールドの解析

キャプチャリミット・フィールドは0x00040000なので、キャプチャするパケットの最大長は262144バイトということがわかります。

● データリンクタイプ・フィールドの解析

データリンクタイプ・フィールドは0x00000001なので、**表1**からIEEE 802.3 Ethernetということがわかります。

パケット情報ヘッダ・ブロックの解析

パケット情報ヘッダ・ブロックも固定長で16バイトとわかっているので、**図5**を参照しながらフィールドに分けると**図9**になります。

● タイムスタンプ・フィールドの解析

タイムスタンプ・フィールドは秒部分の0x540C0156を10進数に変換すると1410072918で、マイクロ秒部分の0x00046895を10進数に変換すると288917なので、UNIX時間は両方を合わせて1410072918.288917秒になります。

● キャプチャしたパケットの長さ・フィールドの解析

キャプチャしたパケットの長さ・フィールドは0x0000002Aなので、パケットデータ・フィールドの長さは42バイトということがわかります。

▼図9　arp.pcapのパケット情報ヘッダ・ブロック

	00	01	02	03	
18:	56	01	0C	54	← タイムスタンプ(秒部分)・フィールド
1c:	95	68	04	00	← タイムスタンプ(マイクロ秒部分)・フィールド
20:	2A	00	00	00	← キャプチャしたパケットの長さ・フィールド
24:	2A	00	00	00	← キャプチャしたパケットの元の長さ・フィールド

● キャプチャしたパケットの元の長さ・フィールドの解析

キャプチャしたパケットの元の長さ・フィールドも0x0000002Aなので、パケットの元々の長さも42バイトということになります。

残りはパケットデータ・ブロック

パケット情報ヘッダ・ブロックの「キャプチャしたパケットの長さ・フィールド」から、パケットデータ・ブロックの長さは42バイトということがわかりました。つまり、グローバルヘッダ・ブロックとパケット情報ヘッダを除いた残りの部分は、パケットデータ・ブロックということになります。

パケットデータ・ブロックにはキャプチャしたパケットが記録されています。バイトオーダーはビッグエンディアンです。パケット自体の解析については省略します。

pcapファイル形式についての解説は以上です。

次号は

次号は、pcap-ngのファイル形式を見てみましょう。そしてpcap-ngの解析とpcapとの違いに迫ります。SD

▼図10　arp.pcapのパケットデータ・ブロック

	00	01	02	03	
28:	FF	FF	FF	FF	
2c:	FF	FF	00	0C	
30:	29	28	F4	07	
34:	08	06	00	01	
38:	08	00	06	04	
3c:	00	01	00	0C	← パケットデータ・ブロック
40:	29	28	F4	07	
44:	C0	00	02	01	
48:	00	00	00	00	
4c:	00	00	C0	00	
50:	02	02			

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

第3回　サーバ構築の実際(その2)

先月号では、SoftLayerでトライアルキャンペーンを利用して物理サーバを起動するところまでを説明しました。SoftLayer上で実際にサーバを構築するうえでのヒントやAPIの簡単な解説、そしてネットワークについて本稿では解説します。

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

30分で利用可能になるベアメタルサービスが新しく始まりました。これにともなって従来、短時間、低料金で利用可能なベアメタルインスタンスがありましたが、現在はベアメタルサーバ(Hourly/Monthly)にサービスメニューが統合されました。今までよりも手軽にベアメタルサーバを利用できるようになりました。

また、ハードウェアセキュリティを高めることができるIntel TXTの機能が利用可能になりました[注1]。

コマンドラインとAPIの利用(前回の続き)

前回は、Pythonで書かれた**sl**コマンドの使い方を見てみました。この**sl**コマンドも内部はPythonで記述されており、SoftLayerのAPIが呼び出されて利用されています。

今回はAPIの利用方法について紹介します。

PHPによるサーバ状況の確認方法

SoftLayerのAPIは複数の言語ライブラリが公開されています。今回はその中でもPHPを利用します。ライブラリはSoftLayerのWebページにも掲載されています[注2]。GitHubで公開されていますので、ローカル環境にダウンロードして利用しましょう。SoftLayer APIでは、各種の値を管理するために「Data Type」というデータ構造を利用しています。利用できるData Typeの値についてはSoftLayerのサイトで確認できます。このデータ構造を利用するためにこのObject Maskという機能を利用します。

今回は、仮想サーバの全サーバリストを取得してその起動状況を確認してみます。実際のポータルの画面は、一覧性に乏しくてどのサーバが起動してるのかわかりにくいので、その機能を作ってみましょう。**リスト1**にサンプルコードを示します。

SoftLayerのWebページ上の情報[注3]から**リスト1**を解説していきます。

今回は仮想サーバなのでVirtualGuestsの情報を得ることになります。まず、メソッドとして、**getVirtualGuests()**を利用して**powerStatus**が取得できることがWebページの情報からわかりました。このあたりの情報は検索をして探します(実際に利用するには、電源状況だけでなくてMonitoringステータスも参照するとより良いでしょう)。

注1) http://www.softlayer.com/intel-txt

注2) PHP SoftLayer Development Network(http://sldn.softlayer.com/article/PHP)

注3) AccountSoftLayer_Account　SoftLayer Development Network(https://sldn.softlayer.com/reference/services/SoftLayer_Account)

▼リスト1　サンプル例（左の数字は行番号）

```
01 <?php
02 require_once dirname(__FILE__) . '/SoftLayer/SoapClient.class.php';
03 $apiUsername = 'SL29***';
04 $apiKey = 'ab********';
05 $client = SoftLayer_SoapClient::getClient('SoftLayer_Account', null, $apiUsername,➢
   $apiKey);
06 $objectMask = new SoftLayer_ObjectMask();
07 $objectMask->virtualGuests->powerState;
08 $client->setObjectMask($objectMask);
09 try {
10         $hosts = $client->getVirtualGuests();
11         foreach ( $hosts as $i => $host ) {
12                 $hostInfo['type'] = "CCI";
13                 $hostInfo['id'] = $host->id;
14                 $hostInfo['fullQualifiedDomainName'] = $host->fullyQualifiedDomainName;
15                 $hostInfo['powerState'] = $host->powerState->keyName;
16                 print ("$hostInfo[fullQualifiedDomainName] \t\t$hostInfo[powerState]\n");
17         }
18 } catch (Exception $e) {
19         print $e;
20 }
21 ?>
```

図1①の「Account」データ構造に定義されている「**getVirtualGuests**」というメソッドを実行すると、②の「**SoftLayer_Virtual_Guest**」という値が得られることがわかります。そこで、「**SoftLayer_Account**」を宣言します。

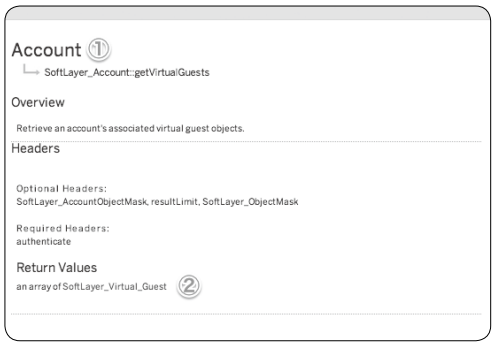
```
05 $client = SoftLayer_SoapClient::➢
   getClient('SoftLayer_Account', null, ➢
   $apiUsername, $apiKey);
```

これで**$client**として**SoftLayer_Account**を定義をしています。

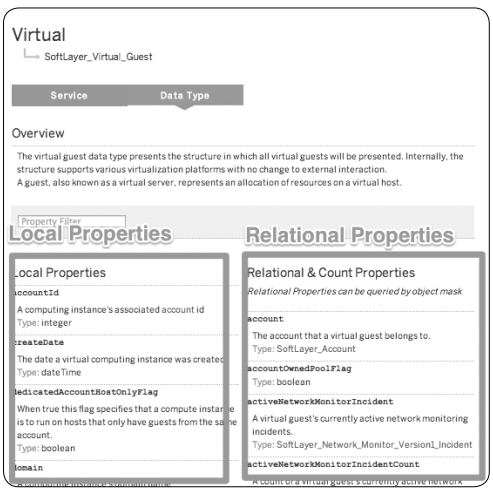
次に**SoftLayer_Virtual_Guest**の内容について確認します。Webページを確認すると、

図2の形で「Service」と「Data Types」というタブがあります。今回は値を取りたいので「Data Type」を見ていきます。「Data Types」には「Local Properties」と「Relational Properties」があります。この内「Local Properties」については先ほどの**$client**に対して、**getVirtualGuests();**を呼び出すだけで取得できますが「Relational Properties」の項目は得たい値を宣

▼図1　「Account」データ構造



▼図2　SoftLayer_Virtual_Guestの内容確認



言する必要があります。

今回はPowerSateの値が欲しいので次のように定義をします。

```
06 $objectMask = new SoftLayer_ﾌ
   ObjectMask();
07 $objectMask->virtualGuests->powerState;
08 $client->setObjectMask($objectMask);
```

まず値が欲しい場合にはObjectMaskを作ります(6行目)。そしてそのObjectMaskに取得したい項目を宣言(7行目)したら$clientに対して登録(8行目)をします。こうすることで**getVirtualGuest()**で呼び出した結果、戻り値の配列の中には**PowerState**の値が入ることになります。

実際にこのスクリプトを実行すると各CCIの電源の状況が確認できます。これだけでは仮想サーバ(CCI)だけで物理サーバ(ベアメタル)の状況が見られなかったり、また実際に稼働しているかはMonitoringの機能から見たりしたほうが良いのですが、拡張していくと使い勝手の良いツールになります。

最後に、実行した結果は、次のようになります。

```
$ php ex_sd_sample.php
admon.niandc.co.jp    RUNNING
docker01.niandc.co.jp    RUNNING
docker02.niandc.co.jp    RUNNING
```

Webサーバを構築してみよう

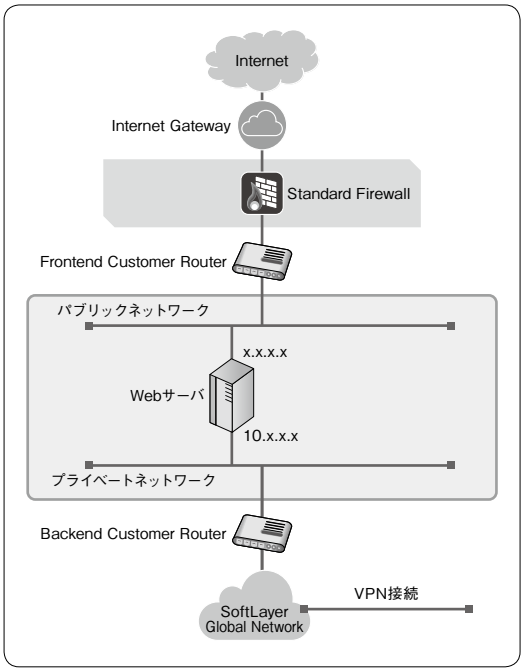
図3のようなWebサイトを例にして、SoftLayerで利用できる各種のネットワークサービスや機能を紹介していきます(**表1**)。

ここではWebサーバとして仮想サーバを1台構築しているものとして説明していきます。このWebサーバは、「パブリックネットワーク」と「プライベートネットワーク」を所有している一般的な構成とします。このサーバを「www1.niandc.net」として設定しましょう。また前出の「Provision Script」によってSoftLayerのCLIが利用可能な状態であるとします。

ドメインレジストラの選択と設定

新規に「niandc.net」を取得します。ドメインも管理ポータルから取得することで、すべての管理がSoftLayerに1つにまとまります(便利!)。管理ポータルの[Services]→[Domain Registration]を選択します。[Register]タブを

▼図3 今回の仮想サーバの構成(注意:Firewall、Routerの詳細な配置場所は非公開なので仮に提示しています)



▼表1 利用するサービス

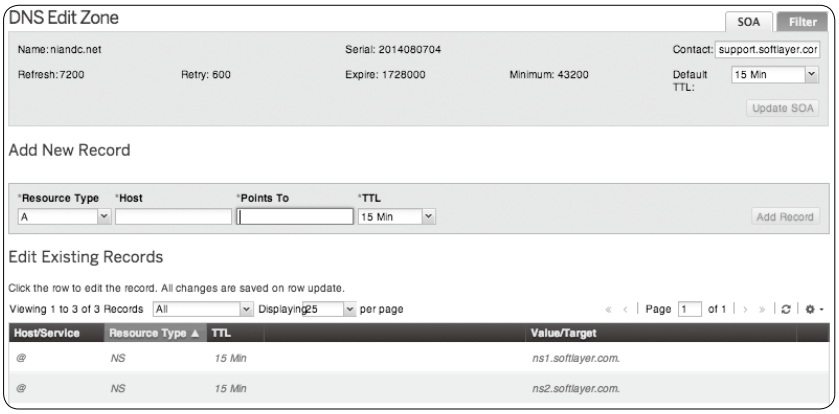
サービス名	内容
ドメインサービス	お名前.comのようなドメインを取得することができるサービス。取り扱いドメイン(com、net、org)
DNSサービス	1次・2次DNSとして利用可能
SSL証明書発行	証明書の購入、発行および管理が可能
Firewall機能	サーバに対してのFirewallを設定

▼表2 ドメイン料金

.com	$9.94
.net	$11.00
.org	$10.00
.info	$9.62
.biz	$11,00
.us	$8.19

▼図4　ドメイン設定

▼図5　DNSの編集



▼図6　DNSをCLIから設定する場合

```
$ sl dns add niandc.net www1 A 111.x.x.x --ttl=600
```

選択すると(**図4**)、取得したいドメイン名を[Domain Name]フィールドに記載してドメインと購入期間(1年～10年)を選択します(**表2**)。あとは購入手続きを進めていけばドメインを取得できます。また移管(Transfer)も実行できます。汎用JPドメインが選択できないのですが、日本データセンターが利用可能になった際には使えるようになってほしいところです。ドメインを購入した場合には、DNSサービス側に自動的にドメインが登録されます(ドメインを購入しなくてもDNSサービスの利用は可能です)。意図しないドメインの移管がされないように[Lock Domain]は[Locked]のままにしておきましょう。

DNSサービス

次にDNSを設定します。DNSサービスには、一般的な「Forward Zone」とセカンダリDNS用の「Secondary Zone」そして逆引き用の「Reverse DNS Records」があります。今回は「Forward Zone」に追加します。管理ポータルの[Network]→[DNS]→[DNS Forward Zone]を選択します。もしくは、さきほどのDomainの画面から「Edit DNS」からも設定画面にすすめます(**図5**)。

設定が可能な、Resouce Typeは、A、AAAA、CNAME、MX、TXT、SRV、SPFになります。操作は非常にシンプルになっておりTypeを選択した後に必要な項目を記入すれば問題ありません。DNSサービスは無料で利用可能です。

今回はAレコードで、www1.niandc.netを作ります。Resouce Typeに「A」Hostに今回の仮想サーバのパブリックIPアドレス「111.x.x.x」、Points Toに「www1」をTTLは「15min」を選択して「Add Record」をクリックすれば作成できます。CLIから実行する場合には、**図6**のようになります。

構成されている内容は`sl dns list`コマン

▼図7 sl dns listコマンドによるDNSの設定

```
$ sl dns list 1664958
:.........:........:.....:....:.................:
:    id   : record : type : ttl : value           :
:.........:........:.....:....:.................:
: 49197747 :      @ : SOA  : 900 : ns1.softlayer.com. :
: 49197748 :      @ :  NS  : 900 : ns1.softlayer.com. :
: 49197749 :      @ :  NS  : 900 : ns2.softlayer.com. :
: 49197761 :  www1  :  A   : 900 : 111.x.x.x        :
:.........:........:.....:....:.................:
```

ドで確認ができます(**図7**)。

SSL証明書

HTTPSで利用するSSL証明書の購入もSoftLayer上で実施可能です。**表3**の6種類が選択可能です。管理ポータルから[Security]→[SSL]→[Orders]を選択します。画面右上の[Order new SSL]を選択すると購入画面が表示されます。購入単位は1年または2年を選択できます。SSL証明書を利用するためにCSRを登録する必要がありますので、サーバなどでOpenSSLを利用して作成を行います(**図8**)。

ここで作成されるCSRファイルの中身を、管理ポータル画面上のCSRを記入欄に貼り付けます。今回はOpenSSLにて作成しているため「Apache + OpenSSL」を選択し次に進みます。住所など必要な情報を記載します。最終的に問題なければ購入を行います。

購入後、承認メールが「webmaster@niandc.net」宛に送付されます。このメールで指定されたURLをクリックすることで処理が行われます。今回は「Rapid SSL」を選択したため、RapidSSLのサイトで処理が継続されます。RapidSSLからSoftLayer側の「管理メールアドレス」宛にCRTが記述されたメールが届きます。今回の証明書は中間証明書が必要なのですがこちらはRapidSSLのサイトからダウンロードすることが可能です。

便利な機能として、購入したSSL証明書を管理する機能が用意されています。この機能自体は「オレオレ証明書」でも利用可能ですので積極的に利用します。管理ポータルから[Security]→[SSL]→[Certificate]を選択します。画面右上の[Import SSL Certificate]をクリックします。さきほどのRapidSSLの場合には、①ローカルで作成したKeyファイル、②メールで送信されてきたCRTファイル、③RappidSSLの中間証明書の内容を**図9**のように登録します。

SSL証明書の内容をSoftLayer上で管理すると、運用面でのメリット以外に(いずれどこかに記載するならまとめて管理ポータル上にあるほうが便利)、**sl**コマンドが利用できれば登録した証明書をダウンロードできます。

▼表3 SSLの種類

サービス名称(2Year)	費用(1年)	購入(2年)
Rapid SSL	$19	$35
QuickSSL Premium	$79	$129
GeoTrust True BusinessID	$99	$179
Symantec Secure Site	$399	$699
GeoTrust True BusinessID with EV	$599	$1,099
Symantec Secure Site with EV	$899	$1,649

Firewall構成

セキュリティ的に欠かせないのがFirewallです。SoftLayerでは複数のFirewallのサービスがあります。ここでは一番単純に利用可能な「Hardware Firewall」を選択します(サーバごとに購入が必要)。サーバの詳細画面の[Addons]という項目があり、[Order Hardware Firewall]を選択します。[100Mbps Hardware Firewall $99.00 per month]と表示されますので[Continue]をク

▼図8 OpenSSLの設定

```
Demo :~/ssl# openssl req -new -newkey rsa:2048 -nodes -keyout www1.key -out www1.csr
```

リックします(**図10**)。

購入後「Firewall」タブから設定を行います。初期の状態では、Bypass All Rulesになっているので有効にするためにActionより「Process Rules」を選択します。これで利用可能になります。今回はポートの22、80、443を許可する設定を**図11**のようします。機能としては単純に「許可」「拒否」で各ポートに対して設定を行います。

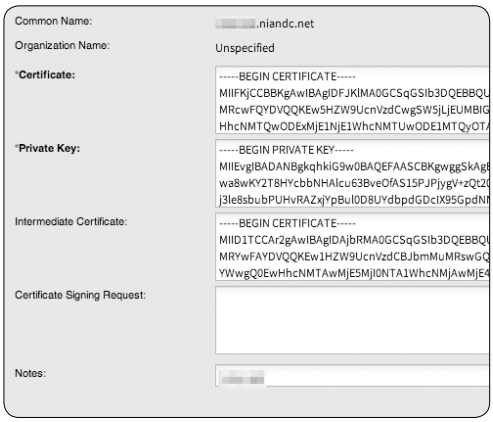
Webシステムの拡張プラン

実はSoftLayerでは、ネットワーク的な多層構成を標準では考慮していません。すべてのサーバはパブリックとプライベートの2つのネットワークに所属しています。各層へのアクセスはサーバに依存します。またこれ以外のネットワークに所属するということはSoftLayerでは別のVLANに所属することになります。この場合、VLANに所属しているサーバは通信ができないため多層にはなりません(通信ができるようにすることも可能ですがその場合単一のVLANに属しているのと変わりません)。

SoftLayerの一般的な構成では、パブリックネットワークに所属しインターネットにサービスを公開しているサーバか、公開していないサーバ(プライベートオンリー)かという2層構成のシステムデザインとなります。Webサーバ(兼アプリケーションサーバ)をパブリックネットワークに所属させ、データベースサーバをプライベートオンリー構成で作ります。この場合データベースサーバは外部からはアクセスするルートを持たずプライベートネットワークに所属するサーバからのアクセスだけを注意すればよくなります(**図12**)。

さらに押し進めると、このWebサーバを負荷分散したいと考えた場合Load Balancerサービスを利用します。その場合、このLoad Balancerを3層構造のプレゼンテーション層とも考えることができます。3層構成としては**図13**のように作ることができます。

▼図9　SSLの中間証明書の設定



▼図10　Firewallの購入

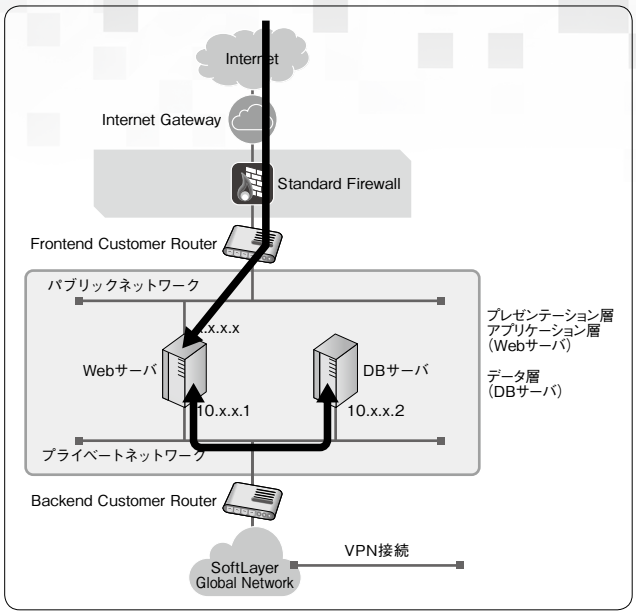
▼図11　Firewallの設定

Webシステムの3層構成化

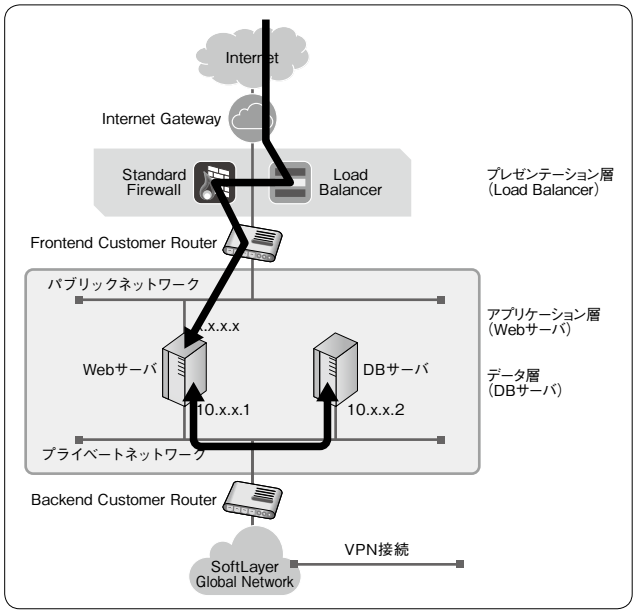
データベースサーバの作成

今回は、インターネットからのアクセスはすべてWebサーバが受けるためデータベースサーバは「プライベートオンリー」なネットワーク構成として構築をします。ネットワークの構成については、仮想サーバの場合には常に選択できます。ベアメタルサーバの場合は、プライベートオンリーしか構成できないサーバがまれにあります(少し安い費用で提示されていますのでお得)。いずれの場合にも注文時で指定しない場合には、IaaSとして定義されれますので、プライベートオンリーの指定をおすすめします(**図14**)。

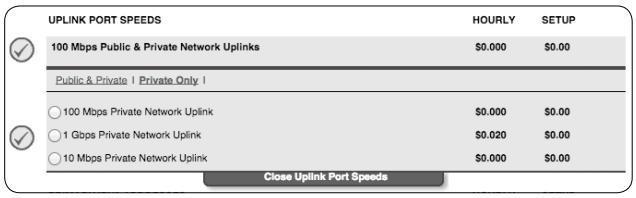
▼図12 プライベートとパブリックネットワークの構成例



▼図13 SoftLayerで3層構成をとる場合の例



▼図14 IaaSとしてプライベートオンリー指定



実際にログインしてみると、NICが1枚のみ設定されていることがわかります。前回作成しているwww1に対して疎通ができることを確認します。この「Private Only」で作成されたサーバは「インターネット」にアクセスするルートがありません。仮に何らかの通信をしたい場合には別途Proxyを用意するなどが必要となります。ただしNTPやレポジトリサーバはプライベートネットワークでアクセス可能なエリアに用意されているため`apt-update upgrade`などは利用できます。

Load Balancerの設定

Load Balancerを利用することで、Webサーバに対しての負荷を分散するとともに障害対策としても有効です。

管理ポータルから「Network」→「Load Balancer」→「Local」を選択します。右上の「Order Local Load Balancer」から注文できます。購入すると、1つVIP(Virtual IP)のGlobal IPアドレスが割り当てられます。ユーザは、このVIPに対してアクセスを行います。名前解決をする場合にはDNSサーバにVIPを登録しておくことになります。負荷分散の定義は「VIP+ポート」単位で行います。したがってVIPは1つなのですが、複数のポート番号に対して負荷分散の定義を追加できます。この単位を「Service Group」と呼びます。また複数の「Service Group」がある場合、購入している同時接続数をどのように配分するかを初期に

設定する必要があります。仮に250Connectionを購入しており、80番ポートと443番ポートでServive Groupを2つ作った場合には何%ずつ割り当てるかを定義します(**図15**)。

現在、Webサーバは「www1」としてDNSサーバに登録されていますので、Load Balancerへの登録後はIPアドレスをVIPへ変更しておきます。またFirewallは、Webサーバに対してはLoad Balancerのみが通信できるように設定をしておくことで安全な状態になります。

負荷に耐えるための AutoScaleの構成

AutoScaleの登場

「AutoScale」と呼ばれる機能はクラウドならではのサービスです。ようやくSoftLayer上でも標準サービスで用意されたのでより便利になりました。AutoScaleは、定義する条件に応じて、自動的に仮想サーバのリソースを拡大・縮小できます。

AutoScaleの機能

利用するには発動条件を設定します。この条件を「ポリシー」と呼びます。ポリシーは次の項目で定義されます。

①ポリシー名
②クールダウン期間の有無
③トリガー(条件)
④その際のアクション

このようにトリガーには、スケジュールベース(定期的な曜日や時間または指定の時間)とリソース不足判断(CPU利用率%)によるものが選択できます。

次にトリガーが適応された際に行うアクションを指定します。「アクション」ではサーバの台数をどのように何台増減させるかを指定できます。

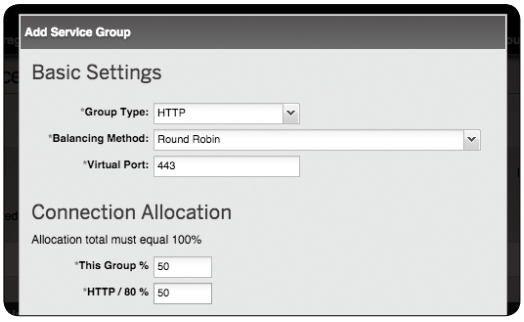
具体的には、次のように1つずつがポリシーとなります(これらは複数作成可能)。

- CPU利用率が80%以上90%以下の場合、相対的(Relative)に2サーバずつ追加(2 Members)する
- CPU利用率が90%以上の場合、絶対的(Exact)にCPU利用率が30%になるまでサーバを調整(増減)させる
- CPU利用率が75%以下の場合、CPU利用率を30%になるまでサーバ台数を調整する

サーバを増やす場合には、負荷の分散を行うための「Local Load Balancer」の配下に追加していくこともできます。ここで指定することは、AutoScaleにより増加したサーバのLoad Balancer配下で管理するポート番号とヘルスチェックの方法となります。

このようにAutoScaleを利用することで、急激な負荷にも耐えるシステムを構築可能になります。

▼図15　LoadBalancerの設定例



まとめ

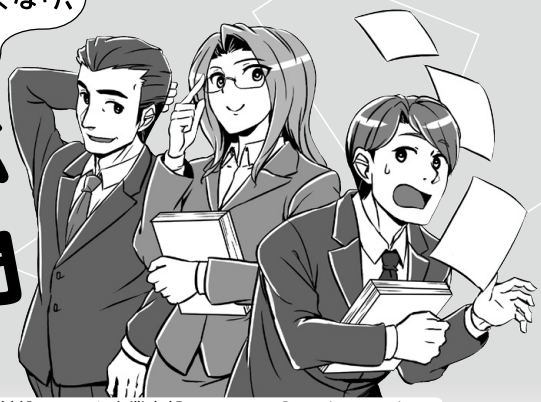
これまでの回では駆け足でSoftLayerの使い方を見てきました。

次回は、最近登場した高速に用意されるベアメタルサーバの紹介や、ベアメタルサーバでしかできない仮想化基盤(VMware)などの導入を紹介していきたいと思います。SD

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



眞野 将徳（まの まさのり）㈱NTTデータ 基盤システム事業本部 manoms@nttdata.co.jp

この物語は、新人SE藤井君がとあるシステムの運用の効率化に取り組み、定時帰りのエンジニアを目指すものです。昨今流行っている「運用自動化」をキーワードに、全9回の連載で、シェルスクリプトやcronを用いた基本的なシステムのジョブ管理から、Hinemosという運用管理ツールを使ってのジョブ管理・運用効率化を学ぶことができます。　イラスト（高野 涼香）

第2回　ジョブ管理の第一歩「シェルスクリプトを動かそう!」

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。『勤怠管理システム』の運用はある程度は自動化できているようですが、まだまだ改善の余地がありそうです。そこで藤井君は、OSSの運用管理ツールである「Hinemos」を使ってジョブ管理を自動化することに決めたのでした。

シェルスクリプト いろはのい

藤井「『勤怠管理システム』ですが、Hinemosを使って運用自動化しようと思います！」

上司「ふむ、よく調べたね。ところでジョブ管理の運用管理ツールへの移行は現状の動きがわからないとできないと思うが、そこはわかってるか？」

藤井「えっと、研修でもシェルスクリプトは学んだのですが、具体的に何をしているかはまだ……」

上司「おいおい、大丈夫か？　ま、まずは現状の理解と技術を身につけるためにも、今動いてるシェルスクリプトでも読んでみることだな」

藤井「は、はいーっ」（ぴゅーん）

ジョブ管理では、システム運用に必要なさまざまなコマンドを組み合わせて実行します。Linuxサーバでは、一般的にシェルスクリプトを用いてこれらコマンドの組み合わせを記述します。つまり、シェルスクリプトを理解することが、ジョブ管理の第一歩となるのです。

Windowsサーバではこうした処理を記述するために、バッチファイルや、VBScript・PowerShellのスクリプトが一般的に用いられます。

このあとは藤井君と一緒に、ジョブ管理のためのシェルスクリプトについて学びましょう。

なお、使用するOSやシェルにより、文法などに多少差があります。本連載では、基本的にOSはRed Hat Enterprise Linux（以下、RHEL）を、シェルはbashを対象としています。

藤井「シェルスクリプトって文法レベルならわかっているけど……。今日も残業か、とほほ……」

藤井君は、さっそく勤怠管理サーバにある就業時間レポート取得のシェルスクリプト（**リスト1**）を読み始めたようです。

藤井「シェルスクリプトは＃から始まる行はコメントなんだよね。でも＃から始まる行の中でも1つ特別なものがあって、1行目の『#!』。これはどのシェルで動作させるかを表すものだったはず。#!/bin/bashって書いてあるから、これはbashのスクリプトなんだな」

シェルスクリプトを始め、Linuxではスクリプトの1行目には#!の後ろにシェルのパスを書きます。これは、shebang（シェバン／シバン）と呼ばれ、以下のスクリプトをどのインタプリタで解釈するかを示すものです。shebangに、実行するサーバにインストールされていないシェルを書いた場合やパスを間違えている場合、そのシェルスクリプトは動作しません。とくに異なるOSで同じシェルスクリプトを動作させる場合には注意が必要です。

たとえばbashの標準的なインストール先は、RHELでは**/bin/bash**、FreeBSDでは**/usr/local/bin/bash**と異なります。この場合、OSに合わせてshebangだけは書き換えるか、シンボリックリンクなどで同じパスでbashにアクセスできるようにしましょう。

変数に情報を代入する

藤井「bashでは変数が使えるんだよね。このシェルスクリプトだと……ああ、ここで使ってるな」

bashでは、**リスト1**の4行目のように変数名だけを記述して、変数に値を代入します。「=」の前後に空白文字（スペース）が入っているとエラーになるので注意しましょう。**リスト1**の5行目のようにコマンド置換を併用することもできます。変数の値を参照する際には、「$変数名」または「${変数名}」とします（**リスト1**の6行目）。代入時と異なり、「$」が必要です。$変数名でも${変数名}でも、どちらの書き方でも動作は基本的には変わりませんが、$変数名を使うと、実行時に変数名が期待どおりに解釈されず動作しない場合があります。たとえば、変数「**$MSG1**」と「**$MSG2**」の内容を出力するために、

```
echo $MSG1and$MSG2
```

▼リスト1　就業時間レポートを取得するシェルスクリプト

```
01: #!/bin/bash
02:
03: #必要な変数を定義する
04: DIRNAME=/opt/working_data
05: TODAY=$(date +%Y%m%d)
06: FILENAME=working_${TODAY}
07: REQUIRE_LENGTH=500
08:
09: #ファイルができるまで待つ
10: while ! [ -e ${DIRNAME}/today_data ]; do
11:   sleep 60
12: done
13:
14: #一時ファイルを削除する
15: rm -f ${DIRNAME}/tmp/*
16:
17: #行数をカウントして、ファイルが正常かチェックする
18: LOG_LENGTH=$(cat ${DIRNAME}/today_data | wc -l)
19:
20: if [ ${LOG_LENGTH} -lt ${REQUIRE_LENGTH} ]; then
21:   logger -p user.warn "File is NG"
22:   exit 1
23: else
24:   mv ${DIRNAME}/today_data ${DIRNAME}/tmp/${FILENAME}
25:   logger -p user.info "File is OK"
26:   exit 0
27: fi
```

と記述すると、「$MSG1and」という変数と「$MSG2」という変数の値を出力する、と解釈され$MSG1の値が出力されません。これを防ぐためには、

```
echo ${MSG1}and${MSG2}
```

と記述すると、期待どおりに変数名が解釈されます。可読性も上がるため、**${変数名}**の書き方をお勧めします。

ループや条件分岐で処理の流れも自由自在

藤井「10行目は何をやっているんだろう。シェルスクリプトで **while** は指定した条件を満たす間 **do** から **done** までを繰り返し実行するはず。だけどここでは **sleep 60** 、つまり1分間待つ処理しかしてないぞ……うーん、わからない」

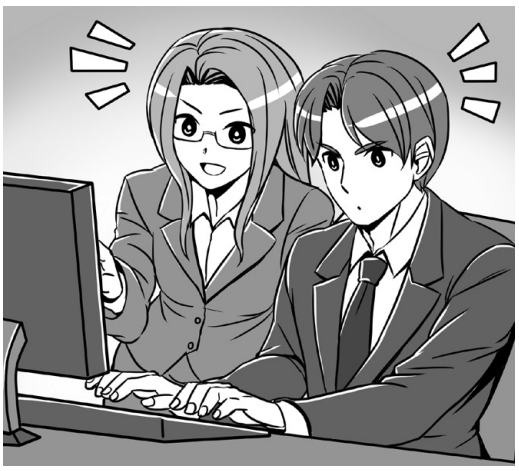
藤井君は、同じ部署の定時先輩に助けを求めることにしました。定時先輩は技術力が高く、毎日定時に帰ることで有名で、藤井君は定時先輩が残業しているところを見たことがありません。

藤井「すみません、定時先輩。シェルスクリプトについて教えてください」

定時「いいけど、どこがわからないの？」

藤井「このwhile文が **sleep 60** してるだけで、何をしているのかわからないんです」

定時「なるほどね……確かに条件を満たしてるときの処理は特別なことはしてないね。なら条件のほうはどう？」



藤井「条件のほうは、**-e** ってことはファイルが存在してたら真、で、『**!**』ってことはその反対だから……ファイルが存在しない間 **sleep 60** をし続けるってことですね。でもなんのために」

定時「ファイルが作成されたことを条件にシェルスクリプトを起動させるのって難しいの。だからその代わりに、早めにシェルスクリプトを起動させたあと、ここでファイルが作成されるまで待っていれば、ファイルの作成を条件に処理を開始できるでしょ。シェルスクリプトの実行条件が特定のファイル作成だった場合によく使われるテクニックだから覚えておいたほうがいいわ」

藤井「なるほど、前任者のメモにも運用に必要なファイルはAPが自動で作成して、そのあとでこの処理は動作するようにしているって書いてありました。シェルスクリプトで、ファイル作成を契機にほかの処理と連携するっていうのは、けっこう難しいんですね。」

定時「そうね。条件に応じて特定の処理をすると言えば、この**リスト1**の20～27行のように、if文では条件を満たした場合、満たさない場合で実行する文を変えられるから、エラー処理などに使えるのよ」

エラー処理をロギングしよう

藤井「この21行目の **logger** っていうのはどういうコマンドなんですか？」

定時「コマンドぐらい自分で調べなさい、まあ今回は教えてあげるけど。 **logger** は **syslog** を作成するコマンドね。 **syslog** はLinuxにおけるシステム管理の基本的なログなの。今度藤井君が導入しようとしている運用管理ツールHinemosでも、各サーバの **syslog** を監視して、システムやアプリケーションで予期せぬ出来事が起きていないかチェックできるのよ」

藤井「なるほど――って今はとくに監視していないはずですよ！　これじゃ何か起きても気づけないじゃないですか！！　ログを出力することはも

ちろんですが、ログをちゃんとチェックすることが大事なんですね。ほかにロギングに使えるものってあるんですか？」

定時「標準出力をファイルにリダイレクトするのが一般的ね。リダイレクトには、『 **コマンド > ファイル名** 』と『 **コマンド >> ファイル名** 』の2種類の書き方があるの。どちらも出力先ファイルが存在しない場合は新しくファイルを作成して書き込むけど、出力先ファイルが存在する場合には、**>** は今のファイル内容は消して上書きされること、**>>** は、今のファイル内容を残して追記されることぐらいは知ってるでしょ」

藤井「はい。でも、以前それでコマンドの出力内容を保存しようと思ったら、一部のログが画面に出てきて、ファイルに保存されていなかったことがあったんですよね」

定時「多分それは標準エラー出力のメッセージね。Linuxにはコマンドのメッセージ出力先に標準出力と標準エラー出力があるの。**>**や**>>**だけだと標準出力しかリダイレクトされないから、両方ともファイルに出力する場合には、**コマンド > ファイル名 2>&1**とするのよ」

藤井「なるほど、今度確認してみます」

定時「あら、もうこんな時間ね。それじゃお先。bashだったら、ファイルサーバにメモがあるから参考にしたら」

藤井「はい、お疲れ様です」

自分の仕事はばっちりこなして、さらに後輩の育成もしつつ定時に帰る。そんな定時先輩を見て、自分もがんばろうとあらためて思う藤井君なのでした。

スクリプトだって終わりが肝心

藤井「そういえば、さっきif文の中で **logger** 以外にもコマンドがあったけれど、何をしているのか確認するのを忘れてたな。**exit** ……？　なんだろう、これ。定時せんぱ……って、もう帰っちゃったんだった。よし、定時先輩の言ってた虎の巻を見てみよう。どれどれ……」

藤井「なるほど、戻り値を指定して終了できるのか。

でも、なんでそんなことするんだろう。ってか、そもそも戻り値ってなんだ？ もうちょっとよく調べてみるか」

Linuxでは、あらゆるコマンドは終了ステータスとして戻り値を戻します。これを確認すればコマンドが正常終了したかどうかがわかります。

シェルスクリプトの場合、とくに指定していない場合には「最後に実行したコマンドの戻り値」がシェルスクリプトの戻り値になります。しかし、**exit** を用いると、任意の値を戻り値として、その場で終了できます。なおLinuxでは、コマンドの戻り値は、0から255までと決められていて、基本的に戻り値0が正常終了と扱われます。また、126以上は特別な意味を持つ戻り値として利用されています。そのため、異常終了の戻り値には1から125までを選ぶと良いでしょう。

藤井「つまりファイルの行数が必要な行数**(${REQUIRE_LENGTH})**あるか確認して、全員分のデータがすべて揃っているかチェックしてるんだな。で、足りなかったらどこかで異常が起きてるってことだからエラーログを出力して終了する、と。そしてファイルに問題なければ正常終了のログを出力して、おしまい、だね。やったー、このシェルスクリプトが何をやってるかばっちり理解できたぞ！！」

藤井「でも、作成されたファイルがおかしい場合にログ出力して、異常終了しておしまい、じゃまずいよな……ん？　前の担当者の運用ノートに

なんか書いてあるぞ。『もしファイルの異常により、処理が正常に終了しなかった場合は、リカバリ用スクリプトがあるので、翌日5時以降に実行すること』か。正常終了していたかどうか手動で判断してこのリカバリスクリプトを実行するのは面倒だし、自動で実行できるようにしたいな」

シェルスクリプトについては、理解できた藤井君でしたが、異常終了時の対処方法など、現状の問題点も見えてきたようです。シェルスクリプトの肝を理解した藤井君は、ひとまず現状の問題点は頭の隅に置きつつ、他のサーバのシェルスクリプトも読んで、現在の処理内容を理解していったのでした。

今月の時短ポイント

定型処理をシェルスクリプトにまとめることで、個々のコマンドを手動でひとつひとつ実行しなくても、繰り返しミスなく処理を行うことができます。さらにループや条件分岐を使うことで、高度な処理もある程度自動化できます。

次回予告

藤井「よしよし、それぞれのシェルスクリプトの動きもわかったことだし、次はシェルスクリプトをいつ動かすのかスケジュールの組み方についても調べるか。たしか、cronを使ってるって話だったけど……、ぅ……全然わからないぞ……」

シェルスクリプトについては何とか理解できた藤井君。しかし、cronについては、まだまだ理解が足りないようです。悩む藤井君は、はたしてcronで定期的にコマンドを実行する方法をつかむことができるのでしょうか！？

次回「スクリプトを時間どおりに動かしてみよう」

SD

To Be Continued...

Column ロギングとログ監視

本編で藤井君が「ログをちゃんとチェックすることが大事」と言っているとおり、ログは出力したら監視することが重要です。本コラムではsyslogとログファイルのログ監視時の利点と注意点を紹介します。

・syslogでロギングする利点と注意点

syslogではfacility（種類）とpriority（重大度）を指定し、ログを分類できます。ただし、この分類には限界があるため、各ログがどのジョブのものか判別できるようにジョブ名等をログ内容に含めると良いでしょう。

・ログファイルにロギングする利点と注意点

ジョブで標準出力に出力すればリダイレクトにより簡単にロギングすることができます。ただし、同じファイルに書き込み続けることによるサイズの肥大化を防ぐため、適切なローテーションが必要です。ローテーションには、おもにmv方式とcopytruncate方式の2つがあり、監視ツールでログ監視する場合は、ローテーション時の検知漏れを防ぐために、ツールがどちらに対応しているか確認しましょう[注1]。

syslog、ログファイルのどちらを監視する場合でも、ログのパターンの指定によりマッチングを行います。このパターンは一般的かつ強力なもの、例えば正規表現が使えると良いでしょう。今回は出番が少ないHinemosですが、統合運用管理ツールの名のとおり、ジョブだけでなくログ監視の機能も充実しています。正規表現による柔軟なパターンを順序性のあるリストで管理し、パターンマッチ時にアラートすることも／しないこともできます。これにより重要なログだけをアラートができます。

注1） Hinemosは両方の方式に対応しています。

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）

2014年5月号

第1特集
ネットワーク・ビギナー向け基礎講座
「ポート」と「ソケット」がわかればTCP/IPネットワークがわかる！

第2特集
UNIX必須コマンドトレーニング
rmコマンドからcadaverまで基本を押さえる

一般記事
・Rettyのサービス拡大を支えた「たたき上げ」DevOps
・Webアプリのパフォーマンス改善　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。

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3回 渡りに船
アドオンで開発効率アップ！

Herokuが持つ、データストアやログ監視を始めとした100以上のアドオンを使えば、短期間で高パフォーマンスなWebアプリを開発できます。今回はHerokuの人気アドオンの紹介、追加方法、プラン変更を解説します。「サンフランシスコだより」では、サンフランシスコの自由な通勤事情についてお話します。

Heroku　織田 敬子（おだ けいこ）

Heroku add-ons

前回は、Herokuにサンプルの Railsアプリケーションをデプロイするところまでをやりました。今回は、Herokuの add-ons（アドオン）について解説しますので、実際に追加したりプランを変更してみましょう。今回の連載から読みはじめる人は、Sign up[注1]をして、Toolbelt[注2]をインストールしてみてください。その後、GitHubの ruby-rails-sample のページ[注3]へ行き **Heroku Button**（Deploy to Herokuと書いてあるボタン）を押してサンプルアプリを自分のアカウントでデプロイしてみましょう。

Herokuには本当にさまざまな種類のアドオンがあり、その数は100を超えます。アドオンの多くがサードパーティによって作られており、それぞれその分野に特化したプロバイダが作っていますので、完成度の高い技術をすぐにあなたのアプリで使用できます。今回はサンプルの Railsアプリにアドオンを追加していきますが、読者の方がすでにHerokuにデプロイしているアプリにも追加できるアドオンを紹介していきたいと思っていますので、ぜひ追加してみてください。

アドオンを使い始める前に

Herokuを使ううえで、最初はクレジットカードの登録なしに簡単に始めることができましたが、アドオンを追加するにはアカウントを認証する必要があります。アカウントを認証するには、Dashboardのアカウントページ[注4]に行き、クレジットカード情報を登録しましょう。クレジットカード情報を登録するとどうしてもHerokuからお金をとられ始めるのではないかと思ってしまいがちですが、Herokuのアドオンは無料プランがとても充実しているので、無料から使い始めてみることができます。

アドオンの種類

現在利用可能なアドオンの一覧はWebサイト[注5]から参照できます。種類としてはPostgreSQL、MySQL、Redis、MongoDBといったデータストア系、検索に便利なElasticsearch系、ログ系、メール系、モニタリング・解析系、キャッシュ系、その他いろいろな種類が存在します。Webアプリを作るうえで今までは自作していたり自分でサーバを立てていたものが、アドオ

注1）URL https://www.heroku.com
注2）URL https://toolbelt.heroku.com
注3）URL https://github.com/heroku/ruby-rails-sample
注4）URL https://dashboard.heroku.com/account
注5）URL https://addons.heroku.com

ンを使うことによって、自分で用意する必要がなくなります。

追加しておきたいアドオン

これは「RubyKaigi2014」でも発表したのですが、サポートエンジニアという立場上、問題がすでに起きてしまったお客様からの問い合わせを多くいただきます。こんなときに、これを入れておいてくれれば……！といった、今すぐにでも追加してもらいたいアドオンがいくつかありますのでここで紹介します。

・New Relic

モニタリングのアドオンです。アプリケーションのどの部分ににどれだけの時間がかかったかが一目でわかり、とくにタイムアウト系エラーのデバッグの初動捜査にとても役立ちます。また、普段使いとしてどのようなリクエストに時間がかかっているのかも見ることができ、アプリの最適化にも役立ちます。

・Papertrail または Logentries

ログのアドオンです。Herokuはログを直近の1,500行までしか保持しないので、問題が起こったことに気づいてからログを追ったのでは、手遅れになる場合があります。このアドオンは単にログを記録してくれるだけでなく、ログローテーションなどの設計の必要も省いてくれます。検索に便利なWebインターフェースもあり、デバッグにたいへん役立ちます。

その他人気のアドオン

・Heroku Postgres

言わずもがなの一番人気のアドオンです。Herokuが提供しているアドオンというだけあり、サポート体制もしっかりしています。Heroku Postgresについては次号でもっと深く取り上げます。

・SendGrid

メール系では一番人気で、安定したサービス、各言語のしっかりとしたサポートも魅力です。メール系アドオンはほかにもあるので、チェックしてみてください。

・Fastly

キャッシュ系のアドオンです。静的ファイルなどをキャッシュさせるのがおもな用途となります。HerokuはまだTokyo regionがありませんので、ぜひ静的ファイルなどはこういったCDN(Contents Delivery Network)を使いアプリを高速化しましょう。

アドオンを追加してみよう

では実際に、一番簡単なログのアドオンの追加をしてみましょう。アドオンによっては使用にあたってアプリ側に変更を加える必要があるものもありますが、ログのアドオンはアプリに変更を加えることなく簡単に追加できます。

アドオンの追加はadd-onsのページ[注6]もしく

▼図1　アドオンの追加

```
$ heroku addons:add papertrail --app <アプリ名>

Adding papertrail on <アプリ名>... done, v16 (free)
Welcome to Papertrail. Questions and ideas are welcome (support@papertrailapp.com). Happy logging!
Use `heroku addons:docs papertrail` to view documentation.
```

注6）URL https://addons.heroku.com/

はherokuコマンドから行えます。今回はherokuコマンドを使って追加してみましょう(**図1**)。「<アプリ名>」には、ご自分のHerokuのアプリ名を指定してください。

何もプランを指定しない場合は、一番小さなプラン(たいていのアドオンでは無料プラン)が追加されます。

アドオンのプランを変更してみよう

先ほど追加したPapertrailでは、無料プランでは日に10MBが上限で、過去7日分のログしか保持されないのでプロダクションのアプリでは心許ないかもしれません。1つ上のFixaプラン($7.0/月)にプランを変更してみましょう(**図2**)。**図2**中の「<アプリ名>」には、ご自分のHerokuのアプリ名を指定してください。

このように、プランの変更もとても簡単です。今回は追加・プラン変更が簡単なアドオンを紹介しましたが、アドオンによっては追加にいくつかの手順が必要な場合があります。追加・プラン変更の前には必ず各アドオンのドキュメントをチェックするようにしてください。

アドオンを作ってみよう

アドオンはサードパーティ製ですので、みなさんも良いアイデアがあれば今からでもアドオンプロバイダ(提供側)になれます。実際、日本人の方でもアドオンを作ってみた方や、運営されている方もいます。

アドオンを作るには英語のドキュメントが必要になるなどいろいろステップがあり簡単にはいきませんが、興味がある方はぜひチャレンジしてみてください！　詳しくはドキュメント[注7]を参照してください。

サンフランシスコだより

サンフランシスコはとてもリベラルな街です。もちろん筆者のようにいたって普通な人もいるのですが、奇抜な格好や髪型の人もたくさんいますし、同性愛者もたくさんいます。アメリカという国自体が日本に比べて自分を表現しやすい国だと思いますが、サンフランシスコはとくにそうなのではないでしょうか(とか言いつつサンフランシスコ以外には住んだことがありません)。今回はそんなサンフランシスコの、自由な通勤事情について紹介します。

サンフランシスコの通勤事情

日本の通勤事情というと、首都圏の人はおもに電車で、地方の人は車がメインでといったように、地域によって車、電車、バスなどさまざまな交通機関、手段を使って通勤しているかと思います。

サンフランシスコは公共交通機関が発達しているため、バス、電車が市内全体を網羅しており、車を所持していなくてもさほど不自由なく暮らせます。通勤にもバス、電車を利用している人は多いです。サンフランシスコ内の電車、バス乗り放題の定期券($68/月)があり、これを利用している人も多いでしょう。余談ですが、この定期券でサンフランシスコ名物のケーブルカーにも乗ることができ、優雅にケーブルカー通勤もできます。

▼図2　アドオンのプラン変更

```
$ heroku addons:upgrade papertrail:fixa --app <アプリ名>

Upgrading to papertrail:fixa on <アプリ名>... done, v16 ($7/mo)
Use `heroku addons:docs papertrail` to view documentation.
```

注7) URL https://devcenter.heroku.com/articles/becoming-a-provider

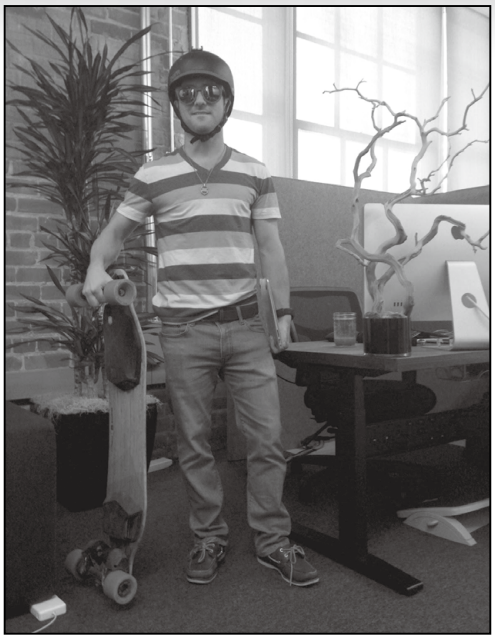
また、公共交通機関もよく使われてはいますが、自転車通勤者も非常に多いです。オフィスに自転車の駐輪場があることはもはや必須と言っても過言ではなく、坂の多いサンフランシスコにおいても自転車で来ている同僚がたくさんいます。サンフランシスコも、街として自転車専用レーンがあったり、自転車ショップ・自転車用アクセサリショップが充実していたりとバイクフレンドリー(こちらでは自転車をバイクと言います)になっています。アメリカにはママチャリといったものがあまりなく、みなさんかっこいい自転車を乗りこなしています。筆者も坂の少ないMission地区に引っ越して、週の半分ほどは自転車通勤をしています。どうしてもデスクワークが多くなりがちな業界ですので、このように通勤で体を動かすことができるのは非常にありがたいです。

サンフランシスコは東京のように、電車・バスが決まった時間にあまり来ないため、自分の時間を管理するのにも自転車通勤は有効と言えるでしょう。また、こちらのIT系企業はオフィスにシャワー室がある場合も多く、夏の暑い日(サンフランシスコには存在しませんが)に汗だくでオフィスに来てもシャワーを浴びてリフレッシュしてから働き始めることができます。

また、Bay Area Bike Share[注8]という便利な自転車シェアサービスがあります。市内のダウンタウンやSOMAといったような主要なポイントに自転車置き場があり、その区間なら乗り捨てが可能になっています。自転車は買うほどじゃないけど、ちょっとしたときに使いたい、といった人には非常に需要があると思います。弊社のオフィスの前にも自転車置き場があります。

同じようなサービスで、まだあまり普及はしていませんが、カーシェアリングならぬスクーターシェアリングのScoot[注9]を使って通勤や移動をする人も見かけます。2012年ごろからサービスを始めていますが、同僚も最近一斉に利用し始めたりと、なかなか便利そうです。筆者もアメリカの運転免許証をゲットしたらぜひ試してみたいと思っています。

▼写真1　電動スケートボードで通勤



最後に、筆者の同僚のとっておきの通勤方法を紹介したいと思います。彼はなんと、スケートボードで通勤しています(**写真1**)。しかもただのスケートボードではなく、電動スケートボードです。ちゃんとヘルメットを被り安全にも配慮して通勤しています。彼の場合行きは下りですが帰りは上りですので、電動機能は非常に重宝しているようです。こんな通勤方法は彼くらいだろうと思っていましたが、最近道端で電動スケートボードで犬の散歩をしつつ通勤している人を見かけたので、サンフランシスコはつくづくおもしろい街だなぁと思いました。SD

注8) URL http://www.bayareabikeshare.com/
注9) URL http://www.scootnetworks.com/

サーバーワークスの瑞雲吉兆仕事術

第4回　情報システムの未来はどのように変わるのか？

クラウドの登場によって、情報システムのありかた、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/

企業の情報システムに起きる変化

本連載の第1回、第2回ではモバイルとクラウドに出会ったエピソードを紹介しました。前回は「コンシューマライゼーション」という切り口で、閉塞的だった情報システムの民主化をどのように実現するのか紹介しました。それは、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取り組みです。それを「社内LAN撲滅運動」と筆者たちは呼んでいます。今回は、このような流れがどんどん進んでいくと、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――筆者の考えを示していきます。

情報共産主義からの脱却が意味するもの

前回は、これまでの企業のセキュリティモデルを「ベルリンの壁」にみたてて、これまでのセキュリティモデルがいかに共産主義的であるかについて説明しました。

実際のベルリンの壁が、民衆の「自由を求めるパワー」に押される形で崩壊したことと同じように、企業がセキュリティを守るために作ったファイアウォール、セキュリティゲートなどのセキュリティ境界も、「コンシューマライゼーションの波」によって倒されようとしています。そして、伝統的なセキュリティモデルを積極的に壊してみて、実際に筆者たちの身に起こった2つのことを紹介します。

◈セキュリティモデルの崩壊

これは意図したことですから、当たり前ではあるのですが、やはりこれまでと同様の「境界型セキュリティモデル」はいっさい機能しなくなりました。このモデルは、ある意味カンタンです。壁を作ってデータを閉じ込めておけば良いのです。データにアクセスする必要があれば、物理的にそこに来ればよいし、守るためには持ち出しを禁止すればよいのです。データの物理的な場所が確定できるので、管理対象も物理的なロケーションに限定すれば問題ありません。

このモデルが機能する前提条件は「境界内に情報を閉じ込めておくことができるかどうか」にかかっています。ですから、当然境界の検査が厳しくなります。ノートPCの持ち出しも制限を受けますし、社内のシステムに接続したい場合は原則として会社というロケーションに行かねばならない、ということが起きます。この様子を、筆者は「共産主義的だ」と称しているわけです。

ですが、今はスマホがあります。クラウドがあります。そのような時代が来ているにもかかわらず、セキュリティを金科玉条にして「スマホは使うな」「クラウドは使うな」などというの

COLUMN　エンジニアのパフォーマンスは見える化できるのか?

著者の会社でも、最初「Salesforceでエンジニアのパフォーマンスを測定する」ということに反対意見もありました。エンジニアのアウトプットはそんなに簡単に数値化できるものではないし、測ることに意味があるのかどうかすら、始める前はわかっていませんでした。

ですが、結果としてやって良かったと感じています。思ったよりも「数値化する前の肌感覚」と、「実際にレポートで表示される結果」が近かったのです。筆者たちのやり方は、厳密なEVM(Earned Value Management)ではなく、売上金額をエンジニアの稼働する割合で配賦(はいふ)するという大雑把なものですが、それでも過去数年運用してみて、おおよその傾向を測るには十分だと感じています。

そして、パフォーマンスが上がっている人は「どうすれば個人やチームで数字をもっと上げられるようになるのか」ということを考えるきっかけにもなりますし、パフォーマンスが上がっていない人は、まわりの人にコツを聞いたり、具体的な目標を設定することで改善のきっかけが掴めるようになったのです。

スキルが上がれば自分のパフォーマンスが目に見えて上がりますので、ゲーム感覚でスキル向上を楽しむこともできているようです。最初はエンジニアのモチベーションが下がってしまうのではないかと危惧していましたが、結果的には上向きの力のほうが強くなったと感じています。

すべての会社でうまく行くかどうかは断言できませんが、「パフォーマンスの見える化」はぜひお勧めしたいと思います。

(なお、「技術者のパフォーマンスを評価するために、便宜的に売上の金額を用いている」だけで、技術者に営業ノルマのようなものがあるわけではありません)

は「仕事の効率を上げるな」と言っていることと同義です。日本のホワイトカラーの生産性は、主要先進7ヵ国中19年連続最下位です[注1]。これにはさまざまな要因が関係しているとは思いますが、こうしたバランスの悪い考えに固執していることにも一因があると思います。

◈エンジニアのパフォーマンス差の拡大

古い「情報共産主義」的なしくみや考え方を排除し、できるだけ創造性や生産性を犠牲にしない方策を突き詰めていった結果、良い面も多かった一方で、エンジニアにとっては少し「つらい」現実も見えてきました。

共産主義的な制度のもとでは、エンジニアの自由な活動が制約をうける一方、パフォーマンスの悪いエンジニアにはいろいろな言い訳が使えることから、良くも悪くも標準的なパフォーマンスに収斂する(もしくは、同じようなパフォーマンスとみなされる)ことが多かったように思われます。

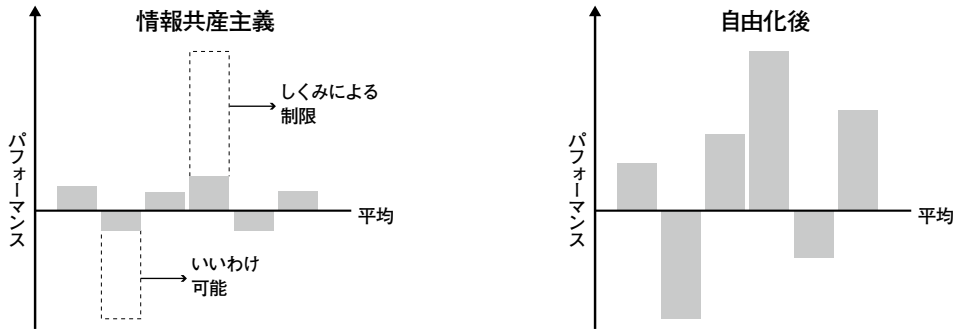
ですが、こうした制約が取り払われてしまうと、今まで会社の制度で能力にキャップをされていた優秀なエンジニアがとことん突き抜ける反面、平均以下のパフォーマンスのエンジニアもはっきりとわかってしまうという現実に向き合わなくてはいけなくなります。

◈Salesforceのレポートによる見える化

筆者たちの社内では技術者に担当した案件の売上数字を付けて、Salesforceのレポートで閲覧できるようにしています。これをやったところ、みんなが「あの人は優秀だよね」という評価をしている人はやはり売上データでも数字が裏付けられるということが起きたのです。優秀な

注1)　URL http://www.jpc-net.jp/annual_trend/

▼図1　エンジニアの能力差



人は、さまざまなツールを使ったり時間を有効に活用したりして、手持ちの案件を早く終了させ、すぐに次の案件に参加できるような体制を自ら作っていたのです。そして、使う携帯電話キャリアのしくみなどの自由度を増した結果、そうした差がより大きく、目に見える形で開くようになるということが起きたのです。

これまでは、会社のしくみや制度がフタになって個人のパフォーマンスが抑制されており、1人のエンジニアが爆発的な成果を挙げるという機会は少なかったと思います。ですから、結果として格差の少ない賃金体系が維持できていたという側面もあったと考えられるのですが、これから制約から解き放たれたエンジニアが爆発的なパワーを発揮すると、そうした人々はこれまでの給与体系では満足できず、悪平等の維持は難しくなるということも起きると思われます。——さて、ここまでで実際に「情報共産主義を捨てて、自由化を進めてみた」結果についてお伝えしてきました。

筆者たちの身に起きたこの小さな変化は、5年後10年後はもっと大きな変化になっているはずです。これから大きなトレンドになりそうな「変化の芽」について話しておきます。

情報システムの転換

2011年ごろに筆者たちが「社内LAN撲滅運動」「情報共産主義からの脱却」と騒ぎ出したとき、社内システムの大半をクラウド化するという試みは、かなり物珍しく見られていましたが、筆者たちが奇を衒(てら)ってこのような判断をしたわけではないことは連載の2回目でお伝えしました。

筆者たちは社内システムのひとつひとつを分解して、「どうすると生産性・コスト・セキュリティのバランスが最高に保たれるか」を考えていった結果、すべてクラウドに軍配が上がった、というのが実状なのです。

たとえば、ファイルサーバで筆者たちが挙げた要件は次のようなものです。

①ISMSを確立するため、ファイルに対する機密性、完全性、可用性を担保できること
②利用者にとって使いやすいものであること
③セキュリティを担保しつつ、モバイルからも利用できること
④お客さま、パートナーと、限定された領域で

ファイルの安全な共有ができること

⑤今後予想される、爆発的な容量増加に耐えられること

ご覧のとおり、特別な要件は何もありません。どれを見ても「まぁ、普通そう思うよね」というレベルの要求ですが、これをオンプレミスでやろうと思うとけっこうたいへんです。ファイルサーバを立てて、バックアップのしくみをいれ、ディレクトリサービスを立ち上げ、モバイル用のVPNを用意し、外部ユーザ用のセキュリティ機構を用意し……と膨大なタスクをこなさなければいけません。

ところが、これをインフラエンジニアががんばって用意しても、「誰もが同じことを考えている」ような領域では、企業として差別化ができないのです。しかも、オンプレミスでファイルサーバを持つ場合、物理的なサーバの保護（入退室管理や物理的な隔離、物理的にアクセスした証跡の取得など）に追加でコストが発生します。

苦労してファイルサーバを立てても、差別化にはつながりませんし、エンジニアのリソースも無駄になります。そのような判断から、筆者たちはファイルサーバをクラウド化できるBoxを使うという判断をしたのです。

実際、過去5年で「メールサーバを立ち上げる」という業務はほとんどなくなりました。メールを使用するだけなら、Google Apps for WorkかMicrosoft Office 365を使ったほうが早いし確実だと認知されていることが理由です。

同じ理由で、今後5年で「ファイルサーバを立ち上げる」という業務も急速に減っていくと考えられます。

こうしたトレンドはすでに起き始めていることで、5年後にはもっと大きなうねりとなって情報システムを担当するすべての方々に大きな影響を及ぼすことになるはずです。

まとめ

ここまで説明したとおり、情報システムには次の大きな流れが待ち構えています。

- **クラウドへの流れは不可避**
- **システム開発が減少する**
- **データを「物理的に守る」ことから「論理的に守る」ことへ焦点が移動**

次回では、もう少し「将来の情報システム像」に焦点をあてて未来のシステムの姿をイメージするとともに、エンジニアの将来をどのように考えたら良いのか、クラウド時代のエンジニアのキャリア戦略について考えてみます。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
- コンピュータとの出会いは10歳の頃
- 当時はPC-8001にベーマガのプログラムを入力する日々
- コンピュータの購入は11歳／SHARP X1
- 中2の時に初めてプログラムが書籍に掲載
- 高校入学記念にX68000を購入
- 大学生の時にパソコン通信開始。本格的にシェアウェアを販売
- 総合商社でインターネットサービスプロバイダー事業に携わる
- 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7回 多機能ファイラー「dired」!

今回のテーマは、ディレクトリ内のファイルの一覧、各種操作を行えるファイラー「dired」。ファイルのコピーや削除といった基本操作から、マーク・削除フラグの使い方、画面のカスタマイズ、シェルコマンドとの連携を説明します。ファイルのリネームなどを直観的に行える「wdired」も紹介します。

Emacsに備わった最強のファイラー

ども、Emacs情報サイトrubikitch.comの運営が軌道に乗ってきたるびきちです。今回取り上げるのは、「dired」というEmacsに備わったファイラーです。ファイラーとは、ディレクトリ内のファイルを一覧したり、ファイルを開いたり、1つ以上のファイルのコピー、移動、削除をしたりできるアプリケーションです。これまで「文字入力の一元化」というコンセプトでEmacsのテキストエディタとしての機能を解説してきましたが、実はファイル管理においてもそれが活きてきます。

ファイラーは今や星の数ほど存在します。有名どころに「FDClone」や「Midnight Commander」などがありますが、筆者は使ったことがありません。カスタマイズ性を謳ったファイラーもありますが、diredはそんなものとは次元が異なります。diredはEmacs Lispで書かれているので、diredの挙動すべてを自由にコントロールできるのです。そう、**すべて**です。なぜならdiredを含むEmacs Lispアプリケーションはコアとなる Lisp関数をも再定義できるからです。もしそこにバグがあったり気に入らなければ自分で再定義して解決できます。そのほかのファイラーでは「ここがこうだったらいいのに」と思っても、カスタマイズできない部分に遭遇したら解決できません。コアの再定義などできるはずもありません。

すべてがコントロール可能という特性は、Emacs入門者にとってはとくにメリットを感じないかもしれません。けれども、Emacsを長年使っていくことで自分の知識・スキルが上がるようになれば、自分にとっての理想像ができてきます。入門者でも使えるし、熟練者の要望にも応えてくれるのがEmacs Lispのすごいところです。diredの理想像を表現したものとして、diredの拡張パッケージがたくさん存在します。

Emacsユーザにとって最強のファイラーはdiredです。EmacsがLispという言語を拡張言語として採用した時点で最強は決まっています。diredがあれば、ほかのファイラーはいりません。歴史があって、安定していて、たくさんの拡張パッケージがあって、その気になれば自分で自在にコントロールできるのですから。

起動方法

diredのおもな起動方法は2つあります。1つめは`C-x C-f`でディレクトリを開く方法です。`C-x C-f`は本来ファイルを開くコマンドですが、ディレクトリを開こうとするとdiredが立ち上がります。`C-x C-f`がファイルとディレクトリ

それぞれにおいて別の挙動を持っていることから、diredはEmacsの生態系に、自然に溶け込んでいることがわかります。

もうひとつはC-x dを使う方法です。これはdiredコマンドであり、ワイルドカードが使えるメリットがあります。C-x C-fでワイルドカードを指定するとマッチするファイルすべてを開きますが、C-x dはマッチするファイルのみをdiredで表示します。たとえばC-x d *.txtで拡張子.txtのファイルがdiredで表示されます(**図1**)。

シェルを使っていればとりあえずlsを実行してファイルを一覧したくなるときがありますよね。そんなときはdiredを起動すればファイルを一覧できるだけでなく、ファイルに対してコピーやシェルコマンド実行などの処理が行えます。複数のファイルをマークすることで、簡単に複数のファイルを処理できます。

▼図1 C-x d /tmp/*.txtを実行したところ

```
/tmp:
wildcard *.txt
-rw-r--r-- 1 rubikitch users     2  8月  8 11:36 a.txt
-rw-r--r-- 1 rubikitch users     2  8月  8 11:36 b.txt
-rw-r--r-- 1 rubikitch users 41510  8月  8 06:50 neko.txt
-rw-r--r-- 1 rubikitch users    19  8月  7 07:21 xsel.txt

U:%%-  *.txt              All L3     (Dired by name)
```

▼表1 おもなファイル操作

キー	操作
C	コピー
D	削除
R	リネーム・移動
H	ハードリンク作成
S	シンボリックリンク作成

ファイルを開く

diredを使えば、そこからファイルを開くことは簡単になります。ファイル名が一覧で見られるので、たとえファイル名がうろ覚えでも確実に目的のファイルを指定できます。その場合、diredを開いた直後に「isearch」(C-s)を使うと早く検索できます。

diredからファイルを開くのは RET ですが、eとfも使えます。aも似ていますが、ファイルを開いたあとにdiredバッファを削除する点が異なります。

diredを見ながらファイルを開くにはoです。oはdiredバッファの隣のウィンドウでファイルを開きます。ウィンドウが分割されていないときは分割します。

diredからファイルを閲覧する方法は2つあります。C-oではdiredの隣にファイルを開きますがウィンドウは選択しません。vは RET と同様にdiredからそのファイルにカレントバッファを切り替えますが、そのあとにview-modeにします。view-modeとは、読み込み専用バッファで閲覧に特化した操作法を提供するマイナーモードです。ファイルを閲覧するときはview-modeにしておけば、Ctrl を使わずに操作できるので、指をいたわることができます。

コピー・移動・削除

diredはファイラーなので、ファイルに対してさまざまな処理が行えます。ファイルのコピー、移動(リネーム)、削除などがあります。これらは一貫して大文字のキーに割り当てられています。

表1はおもなファイル操作をまとめたものです。ほかにもchmodなり圧縮なりがありますが、使用頻度が低いので覚える必要はありません。後述するシェルコマンド実行機能を使えば間に合うからです。

ファイルを削除するDを実行すると、念のため本当に削除するかどうかを聞いてきます。とくにLinuxのファイルシステムにおいて、ファイル削除は復元困難なので慎重に行う必要があります。削除してからハッと気づいても遅いです。大惨事を防ぐためには毎日バックアップを

とることが重要です。筆者はバックアップに**何度も**救われました。

ファイルのリネームと移動は内部的には同じ意味です。同一ディレクトリ内においては文字どおり**リネーム**ですが、別のディレクトリにおいては**移動**と表現を変えているにすぎません。移動とはフルパスのリネームと考えてください。

マーク

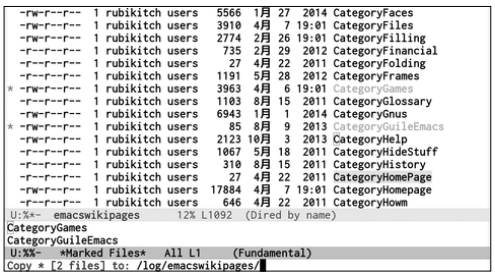
複数のファイルを選択して一括した処理を行うのもファイラーの重要な機能です。diredではこの機能を「マーク」といいます。

マークのコマンドは最低限2つだけ覚えれば良いです。ファイルをマークするm、マークを外す(アンマーク)uです。余裕があれば、マークの反転、すべてアンマーク、マークされたファイル間のカーソル移動、正規表現に基づくマークなども知っておくに越したことはありません(**表2**)。

▼表2　おもなマークコマンド

キー	操作
m	カーソル位置のファイルをマーク
u	カーソル位置のファイルをアンマーク
t	マークを反転
U	全ファイルをアンマーク
M-{	マークされたファイルへ移動(前方)
M-}	マークされたファイルへ移動(後方)
% m	正規表現にマッチするファイルをマーク
* %	正規表現にマッチするファイルをマーク
% g	内容が正規表現にマッチするファイルをマーク

▼図2　2つのファイルをマークしCを押したところ



ファイルをマークしたあとに前述のファイル操作コマンドを使えば、マークされたファイルが操作対象になります。**図2**において、CategoryGamesとCategoryGuileEmacsをマークしているので、処理対象がこれらのファイルであることがわかります。ファイル操作コマンドはマークの有無で一貫した挙動をとります。

diredを2画面ファイラーとして使う

diredを2画面ファイラーとして扱うことができます。**リスト1**の設定を加えてdiredを2つ開いて配置させたとき、一方のdiredがコピー、移動、リンクの作成をしようとしたとき、デフォルトの対象ディレクトリが他方のdiredのディレクトリになります(**図3**)。

ファイル削除に特化した方法

ファイルを削除する方法は実はDだけではありません。ここで紹介する方法は削除に特化したやり方であり、不要なバックアップファイルなどの掃除もできます。

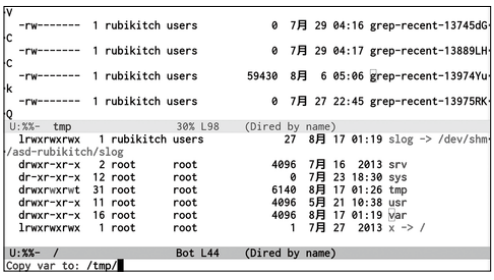
diredにはマーク以外にも「削除フラグ」というマークの変種があります。マークは一般的なファイル操作に使えるのに対して、削除フラグは削除に特化したマークといった感じです(**表3**)。

ファイルを削除する流れは、削除対象のファ

▼リスト1　2画面ファイラー用の設定

```
(setq dired-dwim-target t)
```

▼図3　隣のウィンドウのディレクトリがデフォルトに



▼リスト2　dired-garbage-files-regexpのデフォルト設定

```
(setq dired-garbage-files-regexp
      (concat (regexp-opt
               '(".log" ".toc" ".dvi" ".bak" ".orig" ".rej" ".aux"))
              "¥¥'"))
```

イルに削除フラグを付けて、xで実際に削除をします。カーソル位置のファイルに削除フラグを付けるにはdを押します。d→xという流れです。もちろんマークを使ってm→Dでも削除できます。

削除フラグを付けるコマンドはほかにもあります。#はauto-saveファイルすべてが対象です。~はバックアップファイルが対象です。これらのファイルには「#」や「~」が付くので覚えやすいかと思います。

% &はゴミファイルに削除フラグをつけます。dired-garbage-files-regexpという正規表現にマッチしたものをゴミファイルとみなします。デフォルト設定は**リスト2**です。% &を使いたい人は適宜修正してください。

% dは正規表現にマッチするファイルに削除フラグを付けますが、実際のところはM-!でrmを実行したほうが早いと思います。diredの機能をすべて使うのではなく、既存の知識と併用して柔軟に対処するのが大事です。

マークと削除フラグは* cで交換できます。マークを削除フラグにするには* c * Dを、削除フラグをマークにするには* c D *を使います。

diredからシェルコマンドを実行する

diredからシェルコマンドを実行するには「!」を使います。これさえ知っていれば、たとえほかのファイル操作コマンドを忘れていたとしても同じことができます。Emacsからシェルコマンドを実行するM-!との対比から、忘れにくいと思います。

!もほかのファイル操作コマンドの例に漏れず、マークの有無で挙動が変わります。マークがないときは現在のファイルを、マークがあるときはマークされたファイルが対象になります。

▼表3　削除フラグ関係のコマンド

キー	操作
d	カーソル位置のファイルに削除フラグ
x	削除フラグのファイルを削除
#	auto-saveファイルすべてに削除フラグ
~	バックアップファイルすべてに削除フラグ
% &	ゴミファイルに削除フラグ
% d	正規表現にマッチするファイルに削除フラグ
* c * D	マークを削除フラグに
* c D *	削除フラグをマークに

!の基本的な挙動は、シェルコマンドの後ろにファイル名を指定したシェルコマンドを各々のファイルに対して実行するものです。たとえば、2つのファイルaとbにマークされている状態で「echo 1」を実行したら、「echo 1 a」と「echo 1 b」が実行されます。

ファイル名の埋め込み方は任意に指定できます。デフォルトではシェルコマンドの後ろに1ファイルとなりますが、「?」「*」という文字を使えば別の場所にできます。たとえば、diredで/tmp/を開いていて、ファイルaとbにマークがしてある場合、**表4**のような実行結果になります。

シェルコマンドの最後に「&」を指定すれば、非同期実行になります。つまり、シェルコマンドを実行中でもEmacsを操作できます。

diredでファイル名のみを表示する

通常のdiredはls -lの出力結果を表示するため、ファイル名がどうしても右端に表示されてしまいます。これだと、単にファイル一覧の

▼表4　!を実行した結果

指定したシェルコマンド	echo x	echo x ?	echo * x	echo ? x
実行されるシェルコマンド	echo x a	echo x a	echo a b x	echo a x
	echo x b	echo x b		echo b x
実行結果	x a	x a	a b x	a x
	x b	x b		b x

▼リスト4　パッケージを使うための初期設定

```
(package-initialize)
(add-to-list 'package-archives '("marmalade" . "http://marmalade-repo.org/packages/"))
(add-to-list 'package-archives '("melpa" . "http://melpa.milkbox.net/packages/") t)
```

みを見たい場合には煩わしいものです。そこで、dired-details.elを使ってファイルの詳細を隠してしまいましょう。

Emacsにはバッファの任意の部分を非表示にする機能があります。dired-details.elではこの機能を活用することで、実際のバッファにはデータ上は`ls -l`の結果が入っていますが、見かけ上はファイル名のみを表示できるのです。

dired-details.elと組み合わせると便利なのがdired-toggle.elです。`M-x dired-toggle`を実行するとdiredを左端に表示させ、qでdiredをウィンドウごと削除します。dired-details.elによりファイル名のみが表示される状態で使うと、カレントディレクトリのファイルが左端にきれいに表示されるようになります(**リスト3**、**図4**)。

どちらもMELPAというパッケージ登録所に登録されているので、パッケージの設定(**リスト4**)さえしてしまえばEmacsの中でインストールできます。

```
M-x package-refresh-contents
M-x package-install dired-details
M-x package-install dired-toggle
```

真のDirectory editor「wdired」

最後にお伝えしたいのは、diredを「Directory editor」と呼ぶのにふさわしい「wdired」という機能です。wdiredとは「Writable dired」の略で、diredバッファを書き換えてその結果をファイル名に反映させるというものです。ファイルのリネームや移動という作業を普段のEmacsコマンドで行えるのが何よりの強味です。wdiredでできることは次の4つです。

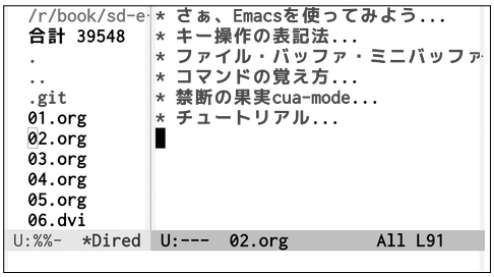
- ファイルのリネーム
- ファイルを別のディレクトリへ移動(同時にリネームも可)
- パーミッションの変更
- シンボリックリンク先の変更

wdiredは標準機能ですが、キーに割り当てられていないので、**リスト5**の設定をします。

▼リスト3　dired-details/dired-toggleを使うための設定

```
(require 'dired)
(require 'dired-details)
(dired-details-install)
(setq dired-details-hidden-string "")
(setq dired-details-hide-link-targets nil)
```

▼図4　dired-details + dired-toggle



▼リスト5　wdiredの初期設定

```
(require 'wdired)
(setq wdired-allow-to-change-permissions t)
 (define-key dired-mode-map "e" 'wdired-change-to-wdired-mode)
```

使い方は極めて簡単で、eでEditable Diredモードにしたあと、ファイル名を書き換え、C-x C-sでその結果を反映させるだけです。diredバッファを普通のファイルであるかのように書き換えて保存するだけなので極めて直観的です。

wdiredの場面では置換と矩形編集が便利です。M-%で置換したり、C-x r tで一連のファイルに別のディレクトリ名を挿入したりしましょう。

また、パーミッション部分の任意のビットでSPCを押せばそのビット部分が切り替えられるので、wdiredでパーミッションも変更できます。

書き換え結果を破棄するにはC-c C-kです。

終わりに

今回はdiredを取り上げました。通常のファイル操作はもちろんのこと、シェルコマンドの連携とwdiredは超強力です。MELPAにはたくさんのdired拡張があるので、ぜひあなた好みのdiredを作り上げていってください。

筆者のサイトrubikitch.comではEmacsの情報発信基地を目指すべく、定番情報や最新情報を日々更新しています。さらにステップアップしたい方はメルマガ登録[注1]をお願いします。Happy Emacsing！SD

注1）URL http://www.mag2.com/m/0001373131.html

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8回　AWS APIでのデジタル署名の全体像を明らかにする②

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

少し前の話になりますが、8月上旬にAWS Trusted Advisorが誰でも利用可能になったというアナウンスがありました[注1]。Trusted Advisorとは、そのユーザの環境や設定をAWSが自動的にチェックして、AWSのベストプラクティスに沿っていない部分について、注意を喚起してくれる便利なツールで、従来はビジネスレベル以上のサポート契約をしているユーザだけに提供されていたものです。Trusted Advisorには、2014年9月現在、次の4つのカテゴリと37種類のチェック機能があります[注2]。

- コスト最適化カテゴリ
- パフォーマンスカテゴリ
- セキュリティカテゴリ
- 耐障害性カテゴリ

今回のアナウンスでそのすべての機能が使えるようになったわけではなく、次の2項目（4つのチェック機能）がすべてのAWSユーザに提供されることになりました。

- パフォーマンスカテゴリ（サービス制限のチェック
- セキュリティカテゴリ（セキュリティグループ、IAMの使用、ルートアカウントのMFA）

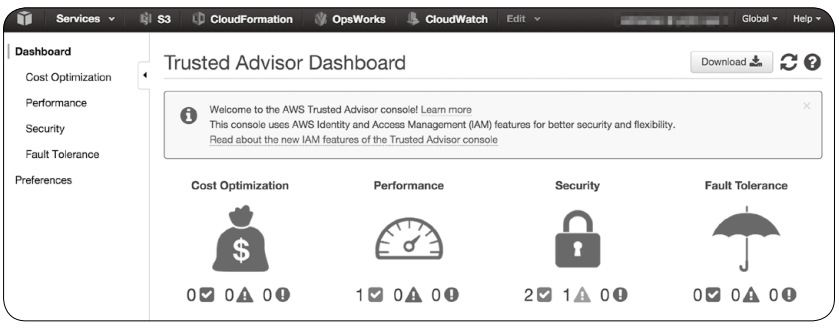
Trusted Advisorの利用

チェック結果を確認するためには、Trusted

注1）URL http://aws.typepad.com/aws_japan/2014/08/trusted-advisor-console-basic.html

注2）チェック項目の詳細は、URL https://aws.amazon.com/jp/premiumsupport/trustedadvisor/を確認ください。

▼図1　Trusted Advisorダッシュボード



Advisorダッシュボード[注3]にアクセスします(**図1**)。この例では、セキュリティカテゴリで1つ注意マークが表示されています。注意マークをクリックすると、SSHのポートを0.0.0.0/0に対してアクセス許可しているセキュリティグループが注意の対象となっていました。

注意対象を指摘するだけでなく、Recommended Action(推奨アクション)として、特定のIPアドレス群からだけにアクセス制限すること、と(英文ですが)きちんとした対応策まで書いてあり、さらに警告レベルの判定基準についても簡単な記載があるので、確認してみると良いでしょう。AWSのベストプラクティスを理解する良い機会になると思います。

IAMについては、IAMマネジメントコンソールで、もう少し詳細なチェックが確認できます(**図2**)。この例では、すべてOKのチェックマークが表示されているので、まずはAWSのベストプラクティスに最低限準拠していると考えてよさそうです。

AWS CLIから使う Trusted Advisor

サポート契約がビジネスレベル以上であればAWS CLIの**support**コマンドからも確認ができます。どんなチェック項目があるのか、それを知るためには、**describe-trusted-advisor-checks**サブコマンドを使います。

・チェック項目の取得：

```
$ aws --region us-east-1 support describe-trusted-advisor-checks --language en | jq  '.checks[] | {name, id}'
```

・出力結果(一部)：

```
{
  "name": "MFA on Root Account",
  "id": "7DAFEmoDos"
}
{
  "name": "IAM Password Policy",
  "id": "Yw2K9puPzl"
}
```

実際のチェックには、**describe-trusted-advisor-check-result**サブコマンドを使います。

・チェック(rootのMFA利用を確認する例)：

```
$ aws --region us-east-1 support describe-trusted-advisor-check-result --check-id 7DAFEmoDos | jq -r '.result.status'
```

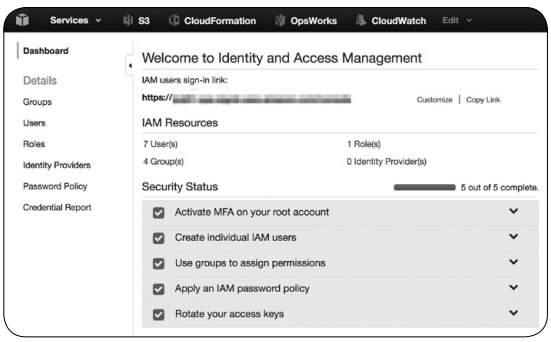
・出力結果(例)：

```
ok
```

定期的な自動チェックなど、Trusted Advisorの定常的な活用を考えてみてはいかがでしょうか。

注3) URL https://console.aws.amazon.com/trustedadvisor/home#/dashboard

▼図2　IAMマネジメントコンソール



▼リスト1　aws-v3.sh

```
#!/bin/sh

# get auth config (-c)
while getopts c: option; do
  case $option in
    c)
      FILE_AUTH=${OPTARG}
  esac
done
shift `expr ${OPTIND} - 1`

# read auth file
if [ "${FILE_AUTH}x" = "x" ];then
  . ${HOME}/.aws/default.rc
else
  . ${FILE_AUTH}
fi

# define files
FILE_BEFORE_SIGN="${HOME}/tmp/aws-v3.tmp"
FILE_OUTPUT="${HOME}/tmp/aws-v3.txt"

# define signature
SIGNATURE_METHOD='HmacSHA256'

# 1. get request ($1)
FILE_REQ=$1
if [ "${FILE_REQ}x" = 'x' ]; then
  echo "Usage: $0 file-request-query"; exit 2
fi

# 2. create the String to Sign.
LC_ALL=en; TZ="GMT"; DATETIME=`date "+%a, %d %b %Y %H:%M:%S %Z"`
if [ "${OSTYPE}x" = 'darwinx' ];then
  echo "${DATETIME}¥c" > ${FILE_BEFORE_SIGN}
else
  echo -n "${DATETIME}" > ${FILE_BEFORE_SIGN}
fi

# 3. Calculate the Signature.
SIGNATURE=`openssl dgst -binary -hmac ${aws_secret_access_key} -sha256 ${FILE_BEFORE_SIGN}
| base64`

# 4.1. The X-Amzn-Authorization Header.
STR_DATE="Date: ${DATETIME}"
STR_X="X-Amzn-Authorization: AWS3-HTTPS AWSAccessKeyId=${aws_access_key_
id},Algorithm=${SIGNATURE_METHOD},Signature=${SIGNATURE}"

# 4.2. make header
cat ${FILE_REQ} | sed -e '/^$/,$d' > ${FILE_OUTPUT}
echo ${STR_DATE} >> ${FILE_OUTPUT}
echo ${STR_X} >> ${FILE_OUTPUT}
echo "" >> ${FILE_OUTPUT}

# 4.3. add body & output request data
cat ${FILE_REQ} | sed -e '1,/^$/d' >> ${FILE_OUTPUT}
cat ${FILE_OUTPUT}
```

今回の流れ

前回から、AWS APIを直接操作するために必要な次の3種類のデジタル署名について解説しています。

・Signature Version 2
・Signature Version 3
・Signature Version 4

前回は、3つの署名方法のうち最もシンプルなSignature Version 3について実例も含めて解説しました。今回は、前回解説したSignature Version 3の手順をシェルスクリプト化したときのサンプルとその実行例の解説をします。

Signature Version 3

シェルスクリプト（Signature Version 3）

署名付きリクエストデータの作成手順をシェルスクリプトにすると**リスト1**のようになります。このスクリプトでは、認証情報についてデフォルトで**~/.aws/default.rc**を参照しますが、**c**オプションで別の認証ファイルを指定できるようになっています[注4]。

hosted zoneの一覧を取得してみる

前回の手順1で作成した最初のリクエストデータを'route53-list-query.txt'という名前で保存して（**リスト2**）、aws-v3.shで署名付きリクエストデータに変換してみましょう。

▼リスト2　route53-list-query.txt

```
GET /2013-04-01/hostedzone HTTP/1.1
Host: route53.amazonaws.com

?maxitems=5
```

注4）誌面の都合で一時ファイルや最終出力について掃除する機能を付けていません。ご了承ください。

・コマンド（例）：

```
$ ./aws-v3.sh route53-list-query.txt
```

コマンドを実行すると署名付きのリクエストデータが出力されます。

・リクエストデータ（サンプル）：

```
GET /2013-04-01/hostedzone HTTP/1.1
Host: route53.amazonaws.com
Date: Mon, 18 Aug 2014 03:49:25 GMT
X-Amzn-Authorization: AWS3-HTTPS
AWSAccessKeyId=AKIXXXXXXXXX0EXAMPLE,
Algorithm=HmacSHA256,Signature=XXp5XlXXX
XX9xXXiXqXorXrv8XXfumXwZXXXXbgXkXw=

?maxitems=5
```

openssl コマンドを実行して、このリクエストデータを貼り付けます。

・コマンド：

```
$ openssl s_client -connect route53.
amazonaws.com:443
  …… (中略) ……
Verify return code: 20 (unable to get
local issuer certificate)
  リクエストデータを貼り付ける
```

Route53上にHosted Zoneが存在すれば、その一覧がXML形式で出力されます（**図3**）。この結果から、次のHosted ZoneがRoute53上に存在することがわかります。

・Name：example.jp.
・Id：/hostedzone/ZONXXXXXXXXXX

hosted zoneの情報を取得してみる

次に、Hosted Zone(example.jp.)の詳細情報を取得してみましょう。さきほどのroute53-list-query.txtをコピーして、route53-zone-query.txtとして保存します。

・コマンド（例）：

```
$ cp route53-list-query.txt route53-zone
-query.txt
```

次に、1行目のGET先として、さきほどのHosted Zoneの一覧表示で取得したHosted Zone ID(Idタグで囲まれた値のうち'/hostedzone'を除いた部分。**リスト3**の例では'/ZONXXXXXXXXXX')を指定します[注5]。

このリクエストデータを、aws-v3.shで署名付きのリクエストデータに変換します。

・コマンド(例)：

```
$ ./aws-v3.sh route53-zone-query.txt
```

署名付きリクエストデータは、下記のようになります。

```
GET /2013-04-01/hostedzone/ZONXXXXXXXXXX
HTTP/1.1
Host: route53.amazonaws.com
Date: Mon, 18 Aug 2014 04:00:18 GMT
X-Amzn-Authorization: AWS3-HTTPS
AWSAccessKeyId=AKIXXXXXXXXX0EXAMPLE,
Algorithm=HmacSHA256,Signature=XXx
0Z0ZZZZZZx0XzZzZzzZzz0ZZzzzZzZXXXXzzXmXz=
?maxitems=5
```

注5) Route53のAPIはREST形式になっており、ここでの例のようにHosted Zone IDを指定することで、特定のHosted Zoneを操作(GET/PUT/POST/DELETE)できるようになっています。

▼リスト3　route53-zone-query.txt

```
GET /2013-04-01/hostedzone/ZONXXXXXXXXXX
HTTP/1.1
Host: route53.amazonaws.com

?maxitems=5
```

opensslコマンドを実行して、このリクエストデータを貼り付けます。

・コマンド：

```
$ openssl s_client -connect route53.
amazonaws.com:443

  ……(中略)……
    Verify return code: 20 (unable to
get local issuer certificate)

  リクエストデータを貼り付ける
```

AWS APIからは**図4**のようなレスポンスが返ってくるはずです。このレスポンス内容から、そのHosted Zoneに関する次の情報を確認できます。

・Comment：そのHosted Zoneに登録したコメント情報
・ResourceRecordSetCount：そのHosted Zoneに登録されているリソースレコードセット(SOAレコードやNSレコードなど)
・NameServers：そのHosted ZoneのNSレコードとして登録されているネームサーバ

さらにこの例では、次の4ホストがこのHosted ZoneのネームサーバDNSにおけるNSレコード)となっています。

・ns-X.awsdns-XX.com
・ns-XXXX.awsdns-XX.co.uk
・ns-XXX.awsdns-XX.net
・ns-XXXX.awsdns-XX.org

▼図3　XML形式で出力されたレスポンス例

```
HTTP/1.1 200 OK
x-amzn-RequestId: XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX
Content-Type: text/xml
Content-Length: 495
Date: Mon, 18 Aug 2014 03:49:25 GMT

<?xml version="1.0"?>
<ListHostedZonesResponse xmlns="https://route53.amazonaws.com/doc/
2013-04-01/"><HostedZones><HostedZone><Id>/hostedzone/ZONXXXXXXXXX</Id>
<Name>example.jp.</Name><CallerReference>RISWorkflow-XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
</CallerReference><Config><Comment>HostedZone created by Route53 Registrar</Comment>
</Config><ResourceRecordSetCount>3</ResourceRecordSetCount></HostedZone></HostedZones>
<IsTruncated>false</IsTruncated><MaxItems>100</MaxItems></ListHostedZonesResponse>
```

次回は

前回、今回の2回にわたって、Signature Version 3を使ってRoute53のAPIを直接操作する例について説明してきました。次回はSignature Version 2の概要について解説し、EC2やVPCでの実例について紹介していきたいと思います。SD

▼図4　AWS APIからのレスポンス結果例

```
HTTP/1.1 200 OK
x-amzn-RequestId: XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX
Content-Type: text/xml
Content-Length: 648
Date: Mon, 18 Aug 2014 04:00:18 GMT

<?xml version="1.0"?>
<GetHostedZoneResponse xmlns="https://route53.amazonaws.com/doc/2013-04-01/">
<HostedZone><Id>/hostedzone/ZONXXXXXXXXX</Id><Name>example.jp.</Name>
<CallerReference>RISWorkflow-XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX</CallerReference><Config>
<Comment>HostedZone created by Route53 Registrar</Comment></Config><ResourceRecordSetCount>
3</ResourceRecordSetCount></HostedZone><DelegationSet><NameServers><NameServer>
ns-X.awsdns-XX.com</NameServer><NameServer>ns-XXXX.awsdns-XX.co.uk</NameServer>
<NameServer>ns-XXX.awsdns-XX.net</NameServer><NameServer>ns-XXXX.awsdns-XX.org</NameServer>
</NameServers></DelegationSet></GetHostedZoneResponse>
```

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24回

ハイパーバイザにおけるファームウェア（その1）BIOS

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、ハイパーバイザにおけるファームウェアのうちBIOSについて解説します。

コンピュータの起動とファームウェア

一般的なコンピュータには、ファームウェアと呼ばれるソフトウェアがマザーボード上の不揮発性メモリに搭載されています。このファームウェアがハードウェアを初期化し、ハードディスクなどからブートローダをロードしてOSを起動します。

このとき、ファームウェアはただブートローダをロード・実行するだけでなく、ブートローダやカーネルに対して起動や初期化の手助けをするためのAPIやデータを提供しています。具体的にはコンソールやディスクへI/Oを行うためのAPI、ブートパラメータを受け取るためのAPI、ハードウェア構成情報（メモリサイズ、接続されているデバイスのリスト、etc.）を受け取るためのAPIやデータなどが含まれます。ブートローダやカーネルはファームウェアのAPIに頼らず直接ハードウェアへアクセスしてコンソール・ディスクI/Oを行ったり、接続されているデバイスのリストを得ることもできますが、機種ごとにアクセスや初期化方法が異なる場合も多く、共通のコードでさまざまな機種に対応するためにファームウェアのAPIが使われることが少なくありません。

このため、仮想化環境においては本来実機のようなハードウェアの初期化処理は必要ないのですが、ファームウェアのAPIを使用しているブートローダやカーネルを変更なしに起動するためにファームウェアを提供する必要が出てきます[注1]。

そこで、今回は、実機上のファームウェアの仕様について見ていきます。

PCアーキテクチャにおけるファームウェア

ARMアーキテクチャなど組込みの分野で多く採用されているCPUアーキテクチャの世界ではファームウェアまでを含めた広く普及した標準というものがありません。一方、x86の世界ではPCアーキテクチャ[注2]がデファクトスタンダードとして広く普及しており、小さな組込み機器からハイエンドのサーバまで基本的にはこのアーキテクチャに従って実装されています。

PCのファームウェアの仕様は、PC/ATアーキテクチャの世代によって徐々に追加・変更されてきていますが、現在では大きく分けてBIOSを採用した

注1）ハイパーバイザ側でファームウェアが行うのと同じ方法によりブートローダをロードすることは可能だが、ブートローダがファームウェアのAPIを呼び出したところでファームウェアが存在しないためエラーが発生する。たとえブートローダをスキップしてカーネルを直接ロードしたとしても、同様の問題が発生する。

注2）PC/ATアーキテクチャ、IBM PC互換アーキテクチャとも呼ばれる。国内では古くはDOS/Vと呼ばれていた。非PC/ATアーキテクチャのx86機として、国内ではPC-98シリーズが有名だったがWindowsの普及に伴ってPC/ATアーキテクチャが主流になった。

従来のファームウェアとUEFI[注3]（Unified Extensible Firmware Interface）を採用した新しいファームウェアの2つに分類できます。今回は、BIOSについて解説します。

BIOSの歴史

BIOSはIBM PC（1981年）とともに登場し、「IBM PC互換機」が普及したことによりパソコンの標準ファームウェアとなりました。

以降、ハードウェアもBIOS自身も当時とはまったく異なる姿になるまで大きく拡張・変更され続けていますが、アプリケーション互換性を保たなければならなかった過去の経緯から、未だBIOSはたいへん古いしくみを使い続けており、機能の実装に大きな制約があります。

具体的に一番大きい制約は、リアルモード（CPUがオリジナルのIBM PCに搭載されていた8086をエミュレートするモード）で動作しているということです。

これにより、一度にアクセスできるデータサイズは64KBに制限され、メモリ空間全体でも通常1MBまでしかアクセスできません。

プロテクトモードやロングモード（64bitモード）でOSを走らせるためには、CPUのモード切り替えの作業が必要になり、ブートローダやOSの構造はやや複雑になります。

また、BIOSがハードディスクからロードするブートローダのサイズも決められており、すでに現代のブートローダはここに入りきらないサイズに達してしまい、「多段ブート」と呼ばれる、ブートローダがさらに大きなブートローダをロードするという対処療法的な実装がなされています[注4]。

USBやBluetooth、LANやWiFiなど、最近のハードウェアのBIOSからの利用もなかなか困難になってきています。

このような状況で「古いしくみ」と認識されつつも、ずっと根本的な変更が加えられぬまま放置されてきたBIOSですが、最近になって新たに問題が発生し、新しいしくみである「UEFIへの移行が進みつつあります。

現在BIOS搭載機上のHDDで使われている「MBR（Master Boot Record）」と呼ばれるパーティションテーブルですが、これは2TBまでのサイズしかサポートしておらず、新たなパーティションテーブルと、新たなパーティションテーブルからブートするしくみが必要になったのです。

なお、BIOSには次のようなサブシステムが存在します。

・ACPI[注5]……OSからの電源制御とハードウェア構成の取得に用いられ、BIOSとともに搭載されているファームウェア。主要な部分はACPIテーブルと呼ばれるメモリ上に置かれたデータ構造で、ここに電源制御に必要な情報やハードウェア構成情報などが書かれている。このうちハードウェア構成情報はAML（ACPI Machine Language）と呼ばれる中間言語で記述されている。AMLはASL（ACPI Source Language）から生成されており、木構造の名前空間やサブルーチン呼び出し、ループなどの制御構造を持つ高級でやや複雑なフォーマットである。ACPIの仕様は1回の連載で書ききれない程度に複雑で大きなものだが、今回は詳細を紹介しない

・SMBIOS（System Management BIOS）……システム管理にかかわるPC固有の情報をOSへ提供するためのファームウェア。ACPIと同様にハードウェアの構成情報を提供するものだが、マザーボードのモデル名、メモリモジュールのモデル名やスペック、冷却デバイスや温度プローブの情報、IPMIデバイスに関する情報など、ACPIとは異なる情報を提供する

・Video BIOS……グラフィックカードを初期化し、ブートローダや起動直後のカーネルからグラフィッ

注3）最新のUEFIの仕様は次のURLで公開されている：http://www.uefi.org/specifications

注4）参考：http://www.itmedia.co.jp/help/howto/linux/redhat7_2_grub/05.html

注5）最新のACPIの仕様は次のURLで公開されている：http://www.uefi.org/specifications

クカードへ描画を行うためにグラフィックボードに搭載されたファームウェア。テキストモードを実現するためのフォントデータも搭載する

・SCSI BIOS……SCSIボードに搭載されたディスクブート用のファームウェア。SCSIブートを行うときにのみ必要なオプション機能

・PXE（PXE ROM）……NICに搭載されたネットワークブート用のファームウェア。ネットワークブートを使うときだけに必要なオプション機能

BIOSのインターフェース

MBRのレイアウト

BIOSではハードディスクのパーティションテーブルとしてMBRを用います。MBRはハードディスクのLBA 0[注6]に置かれる512byteのデータ構造で、**表1**のようなレイアウトになっています。

ブートストラップローダはBIOSからロード・実行されるブートプログラムで、パーティションの情報はパーティションテーブルに存在します。

なお、最大パーティション数は4個と決められており、4つのパーティションを「物理パーティション」と呼び、拡張パーティションと呼ばれる物理パーティション内に論理的なパーティションを、入れ子式に作成することで、パーティション数の制限を回避しています（論理パーティション）。

パーティションテーブルのレイアウトは**表2**のようになっています。

ブートフラグはOSの置かれているパーティションの指定に使われます。パーティションタイプはその種類の表現に使われます。開始セクタ・終了セクタのエントリがCHSとLBAで2つありますが、これは大容量のHDDをサポートするため導入されたBIOSのLBA拡張により、MBRへあとから変更が加えられたものと思われます[注7]。

注6） ハードディスクの先頭セクタ。ハードディスク上の位置がnセクタ目にあることを表記する際、LBA nのように記述することがある。

注7） 参考：http://en.wikipedia.org/wiki/Logical_block_addressing#Enhanced_BIOS

▼表1　MBRのレイアウト

オフセット	サイズ	内容
0x000	446byte	ブートストラップローダ
0x1be	64byte	パーティションテーブル（4エントリ）
0x1fe	2byte	ブートシグニチャ（0xaa55）

▼表2　パーティションテーブルのレイアウト

オフセット	サイズ	内容
0x00	1byte	ブートフラグ
0x01	3byte	開始セクタ（CHS）
0x04	1byte	パーティーションタイプ
0x05	3byte	終了セクタ（CHS）
0x08	4byte	開始セクタ（LBA）
0x0c	4byte	総セクタ数

▼表3　PBRのレイアウト

オフセット	サイズ	内容
0x000	3byte	ジャンプ命令
0x003	8byte	OEM名
0x00B	25byte	BIOSパラメータブロック
0x01C	26byte	拡張BIOSパラメータブロック
0x03E	448byte	ブートストラップコード
0x1FE	2byte	シグニチャ

開始セクタ（LBA：Logical Block Addressing）に値がセットされた場合、開始セクタ（CHS）／終了セクタ（CHS）は無視されLBAの値が用いられます。

MBRからのブート

BIOSはMBR上のブートストラップローダをメモリアドレス 0x0000:0x7C00へロード・実行し、これに制御を譲ります。このため現在のOSではブートストラップローダへブートローダの1段目を書き込み、独自の手順で起動しているものが少なくありません。

しかし、もともとのブートストラップローダは「パーティションテーブルからブートフラグがセットされているパーティションを見つけ、先頭セクタをロード・実行する」というプログラムであるべき、のようです[注8]。

このとき、パーティションの先頭セクタには**表3**のようなPartition Boot Record（PBR）が存在することになっているようです。

注8） 参考：http://ja.wikipedia.org/wiki/マスターブートレコード

ブートストラップローダは1セクタを読み込んで先頭アドレスを実行するだけだと思われますが、先頭のジャンプ命令によりブートストラップコードが呼び出され、BIOSパラメータブロック・拡張BIOSパラメータブロックはパーティションの情報をブートローダ／OSに与えるデータ領域として機能するようです[注9]。

一方、本来のブートストラップローダのしくみを取っていない例としてGRUBをMBRへインストールした場合の動作を紹介します[注10]。GRUBがMBRへ書き込む1段目のブートローダ（stage1）はLBA 1以降に配置されている2段目のブートローダ（stage1_5）をロード・実行するために使われます。

stage1_5はLinuxのファイルシステムを読み込む機能を持ち、ファイルシステムから3段目のブートローダ（stage2）を見つけてロードします。

ほかのOSのブートローダでは、また異なる方法でブートしている場合もあると思われますが、MBR・PBRのブートコードサイズの制約を逃れるため、これに似た手法で多段ブートを行なっている場合がほとんどです。

MBRがBIOSからロード・実行されるときは当然リアルモードで実行されますが、多くのブートローダでプロテクトモードへの切り替えが行われており、これによってメモリ空間サイズの制約を受けてOSがロードできないといったことを避けたり、アセンブリではなくCで実装し、より大きく高機能なブートローダを実現したりしているようです[注11]。

BIOS コール

ブートローダやカーネルから呼び出すことのできるBIOS APIは「BIOSコール」と呼ばれ、ソフトウェア割り込みとして実装されており、int命令で呼び出すことができます。

▼リスト1　BIOSコールの例

```
mov ah, 0x0e
mov al, '!'
int 0x10
```

▼表4　主なBIOSコール

INT	レジスタ値	説明
10h	AH=3h	VGAに1文字表示
13h	AH=42h	拡張INT13hでハードディスクから読み込み
13h	AH=43h	拡張INT13hでハードディスクへ書き込み
15h	EAX=E820h	メモリマップを取得
16h	AH=0h	キーボードからの入力を取得

リスト1にBIOSコールを行うサンプルコードを示します。INT=10h、AH=0Ehはキャラクタモードで ALに指定した文字を1文字書き込むAPIで、この場合は「!」という文字が画面に表示されます。

BIOSにはさまざまなAPIが実装されていますが、ブートローダは**表4**のようなBIOSコールが頻繁に呼び出されます。

ハイパーバイザでの BIOS サポート

前述のとおり、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BIOSやUEFIを動作させる必要があります。

オープンソース実装のBIOSを動作させる場合、KVMやXenなどのメジャーなハイパーバイザではSeaBIOS/corebootと呼ばれるGPLのBIOSが用いられています。SeaBIOS/coreboot自体はいくつかの実機上でも動作する完全なBIOSですが、まだ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がBIOS側に必要になります[注12]。また、ほとんどのOSを無改造で実行するのにはACPIのサポートが必

注9）参考：http://caspar.hazymoon.jp/OpenBSD/arch/i386/stand/biosboot/pbr_structure.html

注10）参考：http://www.itmedia.co.jp/help/howto/linux/redhat7_2_grub/05.html

注11）この点についてはBIOS自体も同様で、さまざまな機能や最近のハードウェアへの対応、GUI画面などを提供するため、近年のBIOSはBIOSコールハンドラと初期化コードなど一部だけがリアルモードで動いており、内部の機能はプロテクトモードで実装されている。

注12）またはハイパーバイザ側でSeaBIOSが前提としているデバイス群をエミュレートする必要がある。

須です[注13]。

ファームウェアなしでのゲストOS実行

ファームウェアによるブートローダ実行を飛ばしてハイパーバイザでカーネルを直接ロード・実行し、ファームウェアは実装しないという手も考えられます。この場合でも、カーネルの起動直後のところでいくらかのBIOSコールを行う場合があり、これをハンドルする最低限のBIOS実装が必要です。この起動直後の部分をも飛ばすという手段も考えられるのですが、この場合でも、前述のとおりACPIのサポートが必須になります。

また、x86_64アーキテクチャ対応のOSではロングモードに切り替えたあとにリアルモードへ戻ってBIOSコールできないため初期化時以降はBIOSへの依存性がありませんが、i386アーキテクチャ対応のOSではリアルモードに戻ってBIOSコールを行えるのでBIOSへの依存性があり、BIOS実装なしでは動かない可能性が高くなります。

bhyveで現状対応しているOSが限られているのは、このようにOS側がファームウェアに依存しているにもかかわらず、いまだファームウェアを提供できていないことが原因です。

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、BIOSについて解説しました。

次回は、UEFIについて解説します。SD

注13）SeaBIOSもEDK2もACPIを含んでいる。

セキュリティ実践の基本定石

すずきひろのぶ
suzuki.hironobu@gmail.com

みんなでもう一度見つめなおそう

【第十五回】脆弱性情報を正しく読み解くには

今回は2014年第2四半期（2014年4～6月）の脆弱性に関する話題[注1]を振り返り、脆弱性への理解を深めていきます。また、脆弱性情報流通に関する話題は本連載第4回（本誌2013年10月号）で取り上げましたが、そのときからアップデートされている話題もありますので、それについても取り上げます。

Heartbleedに注目が集まったが……

今年の第2四半期は、4月にOpenSSLのHeartbeat実装の脆弱性[注2]、いわゆるHeartbleed脆弱性が世界的な話題になり、ソフトウェアの脆弱性の深刻さを世間に印象づけた四半期でした。しかし、この期間もほかの期間と変わらず、取り立てて特別なものではありません。今回は「たまたまHeartbleed脆弱性がメディアに大きく取り上げられたために注目を浴びただけ」と言えます。

もちろん、これまで取り上げられなかったソフトウェアの脆弱性が取り上げられたのはたいへん良いことだと思います。しかしこれが一過性で終わってしまうようなことがないように多くの人にソフトウェアの脆弱性問題を理解してほしいと思います。

統計ベースからみた変化

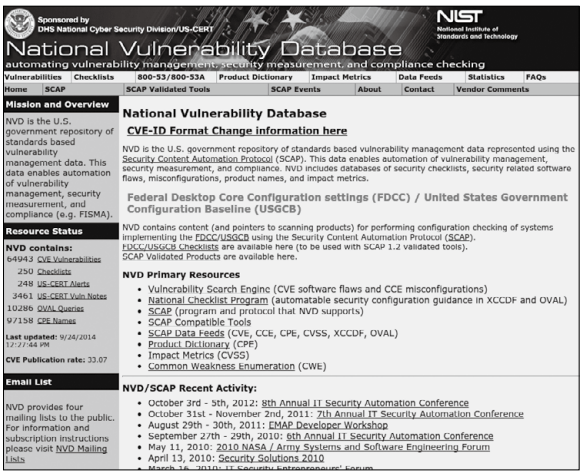
米国国立標準技術研究所（National Institute of Standards and Technology、NIST）が管理しているNational Vulnerability Database（NVD）（**図1**）をチェックしてみましょう。

その前に、このNVDと、米国政府が資金を出し米MITRE社が管理しているCommon Vulnerability Exposure（CVE）[注3]との違いを簡単に説明したいと思います。

CVEは実質的に業界標準的な意味を持つ共通の脆弱性管理です。

一方、NVDは米国政府の標準として管理している脆弱性情報のリポジトリです。そして、NVDの運用の背景にあるのは2002年に米国で制定された「Federal Information Security Management Act of

◆**図1　National Vulnerability Database**
（http://nvd.nist.gov/home.cfm）



注1）参考文献：「ソフトウェア等の脆弱性関連情報に関する活動報告レポート［2014年第2四半期（4月～6月）］」
http://www.ipa.go.jp/files/000040517.pdf
本稿でとくに説明がない場合、数字などは当レポートを出典とします。
注2）詳しくは、本誌2014年7月号および8月号「OpenSSLの脆弱性 "Heartbleed" の教訓（前編・後編）」をご覧ください。
注3）CVEの詳細は、本連載第4回（本誌2013年10月号）で紹介しています。

2002（FISMA）」という法律です。日本語では「連邦情報セキュリティマネジメント法」と訳されています。FISMAは米国政府の組織／機関での情報セキュリティ強化を義務づける法律で、そのために必要な規格やガイドラインなどを作る組織としてNISTが担当しています。

そこでNISTは情報セキュリティを実現するために技術開発をしていくわけですが、これがかなり大きなセキュリティの枠組みを持ったプロジェクトです。詳しくはNISTの“FEDERAL INFORMATION SECURITY MANAGEMENT ACT（FISMA）IMPLEMENTATION PROJECT”のWebサイト[注4]を参考にしてもらうとして、その中の活動の1つがNVDです。

それでは、米国政府が管理する脆弱性データベースNVDより、2012年から四半期ごとに脆弱性登録数がどう変化しているかを見てみましょう（**図2**）。

全体としてのトレンドは右肩上がりで増加しています。ただし、これは深刻度を示しているわけではありませんし、当然ながら、ここの数字には現れないであろう脆弱性もあります。

この数字をどう読むべきでしょうか。筆者は、ソフトウェアの脆弱性が増えた、とは理解しません。なぜかというと、「ソフトウェア・フォルトを引き起こす誤りを含む確率は大きく変化はしない」という前提に立つからです。ソフトウェア・フォルトは一定の割合で、運用障害やセキュリティ侵害の発生に結びつきます。このようなソフトウェア・フォルトが「脆弱性」と呼ばれます。

さて、この前提で考えてみると、「脆弱性が増えた」というのは「ソフトウェア・フォルトが増えた」ということになります。**図2**の統計では、2012年Q1～2014年Q2の間に1.5倍程度にソフトウェア・フォルトが増えていなければなりません。どう考えてもこれはおかしい。なぜならば、ソフトウェアのテスト基準を大幅に下げなければそんなことにはならないからです。世の中が、そんな品質の低いソフトウェアをリリースし始めたという話は聞いたことがありませんし、そんなことをするはずもありません。

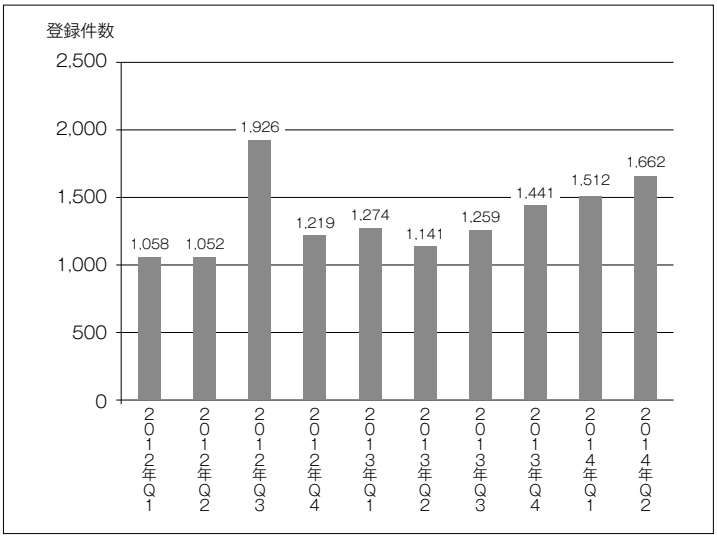
実際には、「これまでも脆弱性は存在していたが、脆弱性として認識していなかっただけ」であると解釈するほうが合理的です。これまでも脆弱性のポテンシャルはあった、それを単純に「ソフトウェア・フォルト」として処理したのか、あるいは「ソフトウェアの脆弱性」として処理したのかの違いではないでしょうか。

結論としては、脆弱性の登録の数が増えているのは、その分、以前よりもソフトウェアが安全に使える方向に向かっていると解釈すべきだ、と筆者は考えます。

国内の推移

振り返って、国内の脆弱性届出の変化を見てみましょう。日本国内ではWebサイトの不備なども脆

◆**図2　脆弱性データベースNVDの登録数推移（2012年Q1～2014年Q2）**



注4）http://csrc.nist.gov/groups/SMA/fisma/

弱性届出数として扱っているのですが、ここではNVDで扱っているようにソフトウェア製品の脆弱性届出の数字を見てみます(**図3**)。

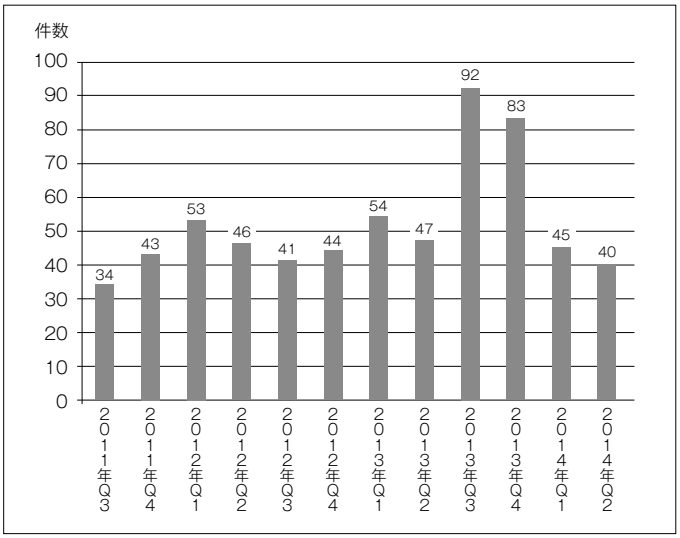
ソフトウェア製品の脆弱性届出に関しては2013年のQ3とQ4の数値が突出していますが、多くは40～55件の範囲に収まっています。そして、2014年Q2の届出を見てみると40件という多くも少なくもない範囲に収まっています。

これをどう考えるべきかは、難しい部分があります。1つは、日本国内だけで使われるソフトウェアは極めて少ないので、そもそも統計として変化が表面化するほどの値が取れないという可能性があります。

多国籍に展開している日本企業でも、海外で脆弱性が見つかった場合には現地で報告されるので、国内の届出の数字には現れません。

ですので、この数字を見て増えた減ったと議論するよりも参考の数字程度に考えてください。

◆**図3　ソフトウェア製品の脆弱性届出推移(2011年Q3～2014年Q2)**



脆弱性の影響度メトリクス

これまで脆弱性を数の面から見てきましたが、本来、脆弱性はその影響度

◉CVEのナンバー表記変更

The MITRE Corporationという組織は米国の非営利団体でFederally Funded Research and Development Centers(FFRDCs)という米政府が資金提供している研究組織群の中の1つです。ここで脆弱性が登録され、CVE番号を付与されたうえで公開されます。MITREは米国政府の資金で運用されている米国の組織ですが、このCVE番号[注5]が実質的に世界共通の脆弱性番号として扱われます。

このCVEの番号の振り方は、これまで、

```
CVE-YYYY-NNNN
```

という具合にCVEに続いて4桁の西暦(YYYY)、4桁の脆弱性の連番(NNNN)という固定長の表現でした。近年は1年間に4桁では足りない可能性も出てきたので、2014年からは可変長として運用を始めました。

ルールは、まず最初の0001～9999までは4桁で、それ以降は可変長になります。たとえば、こんな具合になります。

```
CVE-2014-0123
CVE-2014-12345
CVE-2014-654321
```

この番号をシステムなどで機械的に処理している場合、これらの表記変更により障害が起こる可能性がありますので、アナウンスを繰り返し、時間をおいて周知しながらの変更となりました。

日本のJPCERT/CCとIPAが運用している脆弱性情報ポータルサイトJVN(Japan Vulnerability Notes)もそうですが、CVEを使っている組織がいくつかあります。すでに新しい番号への対応ができる組織は、米国日時で2014年9月16日付けから新しい番号体系に切り替えています。JPCERT/CC、IPAも対応しました。対応組織の一覧は次のURLから参照できます。

・Organizations Compliant with the New CVE-ID Syntax
http://cve.mitre.org/cve/identifiers/compliant_organizations.html

注5) http://cve.mitre.org/

も含めたうえで議論しなければならないはずです。

しかし現実には、いろいろな要素が複雑に絡み合い、その影響度は簡単には評価できません。OpenSSLにおけるHeartbeat脆弱性も、そのメカニズムは2014年7月号／8月号の本連載で説明したとおりなのですが、その影響がどれだけの範囲で発生するのかを定量的に評価するのは非常に困難です。

かといって、影響度の基準が何もなく、ただ脆弱性があるというだけではどう扱うべきかに困ってしまいます。そこで、プライオリティを付けるためにCommon Vulnerability Scoring System（CVSS）という基準が作られました。これは世界各地の主要なセキュリティ対応組織が参加している国際的団体FIRSTが中心になって、CVSS-SIGという分科会で策定したものです。これを危険度、問題度の客観的基準として使えば、共通理解を得ることができます。現在はCVSS v2（2007年6月20日公開）がベースになっています。

このCVSSは大きく3つのメトリクス（尺度）から成り立っています。

- **基本評価基準（Base Metrics）**
- **現状評価基準（Temporal Metrics）**
- **環境評価基準（Environmental Metrics）**

さらに、この各々の基準の中に具体的なメトリクスを求める詳細な項目があります。基本評価基準で6項目、現状評価基準で3項目、環境評価基準で5項目があります。CVSSに使う各種の基準を正しく評価するには高度な専門的知識が必要ですが、CVSS自体を計算するのは次のWebサイトでメニューを選ぶ形でできます。

- **CVSS計算ソフトウェア（日本語版）**
  http://jvndb.jvn.jp/cvss/ja.html

では、CVSSの使われ方の具体例を見てみます。NVDで「DNS」をキーワードに過去3ヵ月間の脆弱性を検索してみます。なお、DNSはインターネット上では重要なソフトウェアですから、いろいろなシステムに影響しやすく、影響度が大きく出やすいので参考として選びました。

次のCVE-2014-3358は、Cisco社のルータに使われるIOS上で見つかった脆弱性で、Ciscoルータに細工をしたmDNSのパケットを送るとサービス不能攻撃が可能になるようです。

CVE-2014-3358
Summary: Memory leak in Cisco IOS 15.0, 15.1, 15.2, and 15.4 and IOS XE 3.3.xSE before 3.3.2SE, 3.3.xXO before 3.3.1XO, 3.5.xE before 3.5.2E, and 3.11.xS before 3.11.1S allows remote attackers to cause a denial of service (memory consumption, and interface queue wedge or device reload) via malformed mDNS packets, aka Bug ID CSCuj58950. Published: 9/25/2014 6:55:08 AM

CVSS Severity: 7.8 HIGH

CVSSの最大の値が10.0ですから、7.8という値はかなり大きな値であり、この脆弱性は大きな影響を及ぼすと理解することができます。

2014年第2四半期に、日本国内届出でCVSSの値が7.0を越えるものは5つありました（**表1**）。なお、ここではIPAが評価したCVSS基本値を用いています。

同時期にメディアを大きく賑わしたOpenSSLのHeartbeat脆弱性のCVSS値は5.0と警告を与える値ではありますが、極端に危険度が高いというわけではありません（JVN、NVDの双方ともCVSS v2）。

- **JVN　JVNVU#94401838　CVSS値 5.0**
- **NVD　CVE-2014-0160　CVSS値 5.0**

このようにメディアなどで取り上げられてたいへん話題になった脆弱性よりも、緊急度／危険度の高い脆弱性はたくさんあります。ただし、OpenSSLの場合、利用しているWebサイト数が多いので、その点ではほかの脆弱性よりも重要度は高いと言えることは付け加えたいと思います。

経済産業省告知の改訂

2014年5月14日に「ソフトウエア等脆弱性関連情

◆表1　CVSSの値が7.0を越える脆弱性（2014年第2四半期、日本国内届出のみ）

CVSS基本値	JVNの番号	脆弱性の内容
7.1	JVN#10319260	サイボウズ リモートサービスマネージャーにおけるサービス運用妨害（DoS）の脆弱性
7.5	JVN#19294237	Apache StrutsにおいてClassLoaderが操作可能な脆弱性
7.8	JVN#78136804	CN8000におけるサービス運用妨害（DoS）の脆弱性
7.6	JVN#50129191	複数のジャストシステム製品同梱のオンラインアップデートプログラムに任意のコード実行可能な脆弱性
7.5	JVN#30962312	TERASOLUNA Server Framework for Java（Web）においてClassLoaderが操作可能な脆弱性

報取扱基準（平成二十六年経済産業省告示第百十号）」[注6]が出されました。平成16年度版から改訂された内容で今後注目したい部分は、脆弱性の受付機関（たとえばIPAやJPCERT/CCなど）の役割についてです。これまでの「発見者が脆弱性情報を届け出るための機関」という位置づけに加えて「調整機関と製品開発者との公表等に係る調整が不可能な脆弱性関連情報について、公表するかどうかの判定を行う機関」ということが加えられました。

これまで、脆弱性の報告があるにもかかわらず調整不能となり公表されなかった脆弱性が、判定を経て公開されるようになると思われます。米国では、脆弱性が報告されて何もアクションが取られない場合、45日経過した時点で公表される、いわゆる「45日ルール」が適用されているシーンを見かけるのですが、日本では必ずしもそうではありません。また、調整しつつも、製品開発者側と意見が合わず調整が不調に終わった場合など、脆弱性として公表できないケースも考えられます。あるいは、開発者とうまくコンタクトが取れないこともあるでしょう。

このようなケースでも、脆弱性情報を流通することが可能なしくみを組み込めるようにソフトウエア等脆弱性関連情報取扱基準が変更されました。今後の具体的な動きは、もう少し先になるかもしれませんが、新しい変化が出ると思われます。

CNA（CVE Numbering Authority）

2010年6月23日、JPCERT/CCはMITREよりCNA（CVE Numbering Authority、CVE採番機関）として認定されました。国内のパートナーシップや海外から報告された脆弱性関連情報に自らの判断でCVE番号を付与することが可能になりました。それまで、日本国内での脆弱性はMITREと調整しなければいけなかったのですが、JPCERT/CCでCVE番号を振ることができます。これも、ここ数年の大きな変化です。

最後に

4月は世間で注目をあびたOpenSSLの問題があったため、これまで以上にソフトウェアの脆弱性がクローズアップされました。しかし、これにかかわらず、脆弱性はこれまでも問題でしたし、これからも問題であり続けることでしょう。これは終わりのない問題です。常に備えていかなければなりません。

一方で、本連載第4回で脆弱性流通の話をしてから、日本国内もずいぶん良い方向に変化したように思います。また、関係者のみなさんの努力により、国内の状況も少しずつではあっても前へ前へと進んでいるように思われます。

本稿の執筆がほぼ終わった時点（9月末）で、bashに大きな脆弱性が発見されました。これは本稿の中で説明したCVSS値が10.0、つまり危険度が最大値になってしまっている脆弱性です。

次回はこのbashの脆弱性について取り上げる予定です。SD

注6）平成二十六年経済産業省告示第百十号　ソフトウエア等脆弱性関連情報取扱基準
http://www.meti.go.jp/policy/netsecurity/downloadfiles/140514kaiseikokuji.pdf

第52回 Android Wearの世界を体験しよう

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！

神原 健一　KAMBARA Kenichi
Web http://www.iplatform.org
Twitter @korodroid

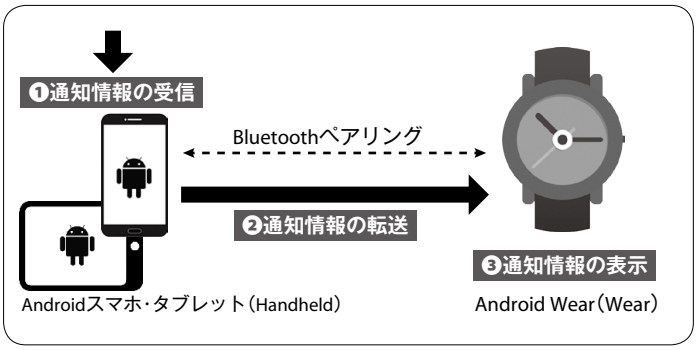
はじめに

Android Wearとは、スマートウォッチなどのウェアラブルデバイスを対象としたAndroidプラットフォームです。Google I/O 2014（Googleの開発者向けカンファレンス）にて、Android Wearを搭載したスマートウォッチとして、「Moto 360」、「LG G Watch」、「Samsung Gear Live」の3つの実機が発表されました[注1]（**写真1**）。これらAndroid Wearの実機（以降「Wear」と表記します）は、あらかじめ、Android 4.3以上を搭載したスマホやタブレット（以降「Handheld」と総称します）とBluetoothペアリングしておく必要があります。これによりHandheldに届く通知情報（Androidにおける Notification）はWearに自動的に転送され、Handheldをポケットから取り出すことなく、通知情報を確認することが可能となります（**図1**）。ほかにもさまざまな機能が提供されています（後述）。

またGoogle I/Oでは、Wearアプリを開発するためのSDKの正式版も発表されました。これは、自分で開発したアプリをWear実機で動作させることができるということを意味しています。ワクワクしている読者の皆さんも多いのではないでしょうか。今回はAndroid Wearの特徴、および、アプリを開発するうえで押さえておくべきポイントをご紹介します。

▼写真1　Android Wear搭載スマートウォッチ

▼図1　HandheldとWear間の情報の流れ



注1）Google I/O以降にも、SonyやASUSなどのメーカーがAndroid Wear搭載端末を発表しています。

▼図2　Notification表示のイメージ

▼図3　音声入力のイメージ

▼図4　独自アプリ動作のイメージ

▼図5　PayPal

▼図6　セカイフォン

Android Wearの概要

Android Wearでできること

Android Wearの代表的な特徴を説明します。

ⓐHandheld上の通知情報表示

交通情報やメールの受信などイベントが発生すると、Handheld上にNotificationとして表示されます。Wearがあれば、その通知情報はWear上にも表示されます(**図2**)。さらに、その通知に対して何らかの操作を行うこともできます。また、画像や長文テキストを表示させることも可能です。

ⓑWearへの音声入力

Wearに向かって音声を発することで、画面に一切触れることなく、インターネット検索やアプリの起動、アラームの時間設定などの操作を行えます(**図3**)。入力された音声を文字列として取得して、自分が開発するアプリの中で利用することも可能です。

ⓒ独自アプリの追加

Wearではプリインストールされているアプリを利用するだけでなく、独自アプリをインストール、実行することもできます(**図4**)。Wearにアプリをインストールするには、Wearとペアリング済みのHandheldへGoogle Play経由でアプリをインストールすればOKです。これにより、WearにはBluetooth経由で自動的にアプリが追加されます。執筆時点ではWearにGoogle Playは搭載されていません。そのため、ユーザはWearアプリをWear実機から直接ダウンロードできないことに注意してください[注2]。

Android Wearアプリの例

Android Wearは登場したばかりですが、対応アプリは少しずつ増えてきています。先に示した特徴を生かしたアプリを2つご紹介します。

1つ目は「PayPal」です。本アプリはオンライン決済サービスを提供しています。Wear対応により、Handheldをわざわざ取り出さなくても、支払いや明細確認などを行えるようになることを発表しています(**図5**)。

2つ目は「セカイフォン」です。本アプリはリアルタイム翻訳サービスを提供しています。同Wearアプリでは、Wearに向かってしゃべると、あらかじめ指定した別の言語に翻訳して画面に表示します。Wearを使って、その場で簡単に翻訳できるという世界を実現しています[注3](**図6**)。

注2) 開発者はadbコマンドなどを用いて、Wear実機にapkを直接インストールすることは可能です。

注3) Google Playから「セカイフォン」と検索することでインストールできます。

Android Wear アプリ開発の基礎

アプリ開発に必要なもの

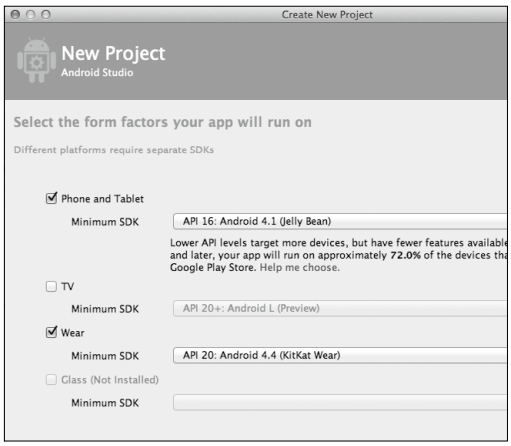
Wearアプリの開発に必要なものを**表1**に列挙します。Android Studioは、IntelliJ IDEAをベースとして作られた、Androidアプリの開発環境です。ビルドツールとしてGradleがサポートされていたり、コーディングを楽にしてくれる便利な機能が数多く提供されているなど、開発者に大変人気があります。Eclipse(+ ADT)に代わって、Androidの開発環境のデファクトスタンダードになりつつあります。同環境はHandheld向けアプリだけでなく、WearやGoogle Glass向けのアプリ開発もサポートしています(**図7**)。Eclipseを用いてWearアプリを開発することもできますが、Android Studioの利用をお勧めします。

開発したWearアプリの動作確認には、実機、もしくはエミュレータを用いることが可能です(**図8**)。

▼表1　Wearアプリ開発で必要なもの

	名称	概要
1	Android SDK	Androidの開発キット
2	Android Studio (もしくはEclipse)	Android向けの統合開発環境
3	Wear用System Image	Wearのエミュレータ
4	[オプション]Android Wear搭載のスマートウォッチ	Wearの実機
5	Androidスマホ・タブレット	Handheldの実機 (Android 4.3以上)

▼図7　Android Studioのプロジェクト作成画面



Android Wear アプリ開発のポイント

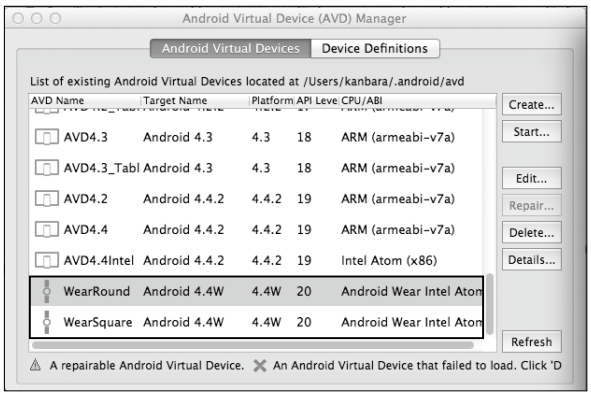
Wearアプリを開発するうえで、筆者がとくに重要と考えている4つのポイントを解説します。

1 通知の表示・操作

Wearアプリ開発で最も基本となるのは「通知」です。**図9**の通知を表示するには、**リスト1**のコードを記述します。

最初に通知生成用ビルダ(NotificationCompat.Builderクラス)と通知マネージャ(NotificationManagerCompatクラス)のインスタンスを生成します。さらに、通知生成用ビルダ

▼図8　Android仮想デバイス一覧+Wear用エミュレータ



▼図9　通知表示

▼リスト1　通知表示の実装

```
// 通知IDの定義
int notificationId = 1000;
// 通知生成用ビルダのインスタンス生成
NotificationCompat.Builder builder = new NotificationCompat.Builder(this)
    .setSmallIcon(R.drawable.ic_restaurant)
    .setContentTitle("Bar Mobile")
    .setContentText("洋食／卵料理");
// 通知マネージャのインスタンス生成
NotificationManagerCompat manager = NotificationManagerCompat.from(this);
// 通知の発行
manager.notify(notificationId, builder.build());
```

▼図10　通知表示へのアクション追加

▼リスト2　通知アクションの実装

```
int notificationId = 1000;
// 電話発信のための暗黙的インテントの生成
Intent i = new Intent(Intent.ACTION_DIAL, Uri.parse("tel:0123456789"));
i.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
// ペンディングインテントの生成
PendingIntent pi = PendingIntent.getActivity(this, 0, i,
        PendingIntent.FLAG_UPDATE_CURRENT);

NotificationCompat.Builder builder = new NotificationCompat.Builder(this)
    .setSmallIcon(R.drawable.ic_restaurant)
    .setContentTitle("Bar Mobile")
    .setContentText("洋食／卵料理")
    // 通知へのボタン画像、ボタンラベル、ペンディングインテントの設定
    .addAction(android.R.drawable.sym_action_call,
                        "電話発信", pi);
NotificationManagerCompat manager = NotificationManagerCompat.from(this);
manager.notify(notificationId, builder.build());
```

のbuild()メソッドを呼び出し通知を生成し、さらに、通知マネージャのnotify()メソッドを呼び出すことで通知が発行されます。同通知は最初にHandheld側で表示され、その後、Wear側でも自動的に表示されます。

▼図11　通知表示のカスタマイズ

今回、NotificationCompat.Builder/NotificationManagerCompatクラス(Compatが付いたクラス)を使っています。これらクラスは、AndroidのSupport Library(Googleの互換性維持のためのパッケージ)で提供されています。Notification.Builder、NotificationManagerなど互換性パッケージに含まれないクラスを使ってしまうと、Wear関連の機能が動作しない原因になり得るので注意しましょう。

さらに、**図9**の状態から左にスワイプすると、電話発信するためのボタン(**図10**)が表示されるように実装を変更しましょう。

リスト2のコードを記述します。電話発信を行うアクション(暗黙的インテント)を生成し、ペンディングインテントにラッピングします。そして、通知生成用ビルダのインスタンスに、addAction()メソッドを用いて、ボタン画像、ボタンラベル、ペンディングインテントを設定すればOKです。

通知をスワイプして表示されるボタンをタップすると、Handheld側で電話発信画面が表示されます。

詳細は省略しますが、**図11**のように、通知に対して画像表示(コンテンツのリッチ化)、長文表示(詳細情報の表示)、グルーピング(同じ種類の通知集約)することも可能です。

2 Wearへの音声入力

Wearならではの魅力を見せるために有効な「音声入力」について説明します。音声入力(**図12**)を実現するには、**リスト3**のコードを記述します。

▼図12　音声入力機能

音声入力(起動前)　　音声入力(実行中)

▼リスト3　音声入力の処理実行

```
// 音声入力結果を渡すインテントの生成
Intent i = new Intent(this, MainActivity.class);
PendingIntent pi = PendingIntent.getActivity(this, 0, i, 0);
// 音声入力機能のインスタンス生成
RemoteInput remoteInput = new RemoteInput.Builder(EXTRA_VOICE_REPLY)
    .setLabel("お話ください").build();
// 音声入力のアクション生成
Action replyAction = new Action.Builder(
    android.R.drawable.ic_btn_speak_now, "話す", pi)
    .addRemoteInput(remoteInput).build();
// ウェアラブル用拡張機能のインスタンス生成
NotificationCompat.WearableExtender wearableExtender =
    new NotificationCompat.WearableExtender();
// 音声アクションの設定
NotificationManagerCompat manager = NotificationManagerCompat.from(this);
manager.notify(notificationId, builder.extend(
    wearableExtender.addAction(replyAction)).build());
```

▼リスト4　音声入力の結果取得

```
// 音声結果が格納されたインテントを用いて処理実行
private CharSequence getMessageText(Intent intent) {
    // 音声入力結果を含むBundleの取得
    Bundle remoteInput = RemoteInput.getResultsFromIntent(intent);
    if (remoteInput != null) {
        // Bundle経由で文字列の取得
        return remoteInput.getCharSequence(EXTRA_VOICE_REPLY);
    }
    return null;
}
```

まず音声入力結果を渡すActivityを指定したインテントを生成します。次に、音声入力機能(RemoteInput)のインスタンスを生成します。その際、あとで音声結果取得に必要となる任意のキー(今回は、EXTRA_VOICE_REPLY)を指定しています。

続いて、音声入力アクションの生成です。RemoteInputインスタンスをaddRemoteInput()メソッドにより設定します。音声入力アクションを通知マネージャに設定するために必要となる、ウェアラブル用拡張機能(WearableExtender)のインスタンスを生成します。そして、ウェアラブル用拡張機能のaddAction()メソッドを用いて、音声入力アクションを設定すれば完了です。

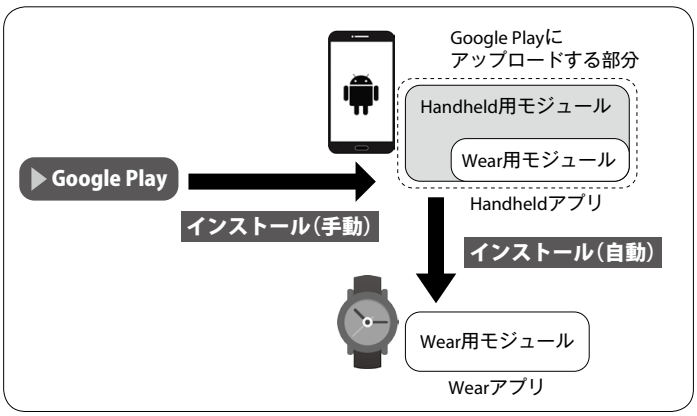
「話す」ボタンをタップすると、音声入力を行うための画面が表示されます。

さらに、ユーザが入力した音声を文字列として取得してみましょう。**リスト4**のコードを記述します。この処理は音声入力結果を受け取るActivity側で実装します。音声入力の完了後、送信されるインテントからRemoteInputクラスのgetResultsFromIntent()メソッドを用いて、結果が含まれるBundle型データを取得します。さらに、その中からキー(EXTRA_VOICE_REPLY)を指定して、値を取得しています。このように音声入力結果を文字列として取得し、アプリ内の処理で利用することができます。

3 Google Play向けのパッケージング

Google PlayでWearアプリを配信する場合、アプリの「パッケージング」に注意する必要があります。具体的には、**図13**に示すように、Handheldアプリの中に、Wear用モジュールを同梱させてパッケージングしたものをGoogle Playで公開する必要があります。こうしておくことで、ユーザがGoogle PlayからHandheldにアプリをインストールすると、Wearにも自動的にWearアプリがインストールされます。パッケージングは一見難しそうですが、Android Studio(+ Gradle)なら簡単にできます。その際、考慮すべき2つのポイントをご紹介します。

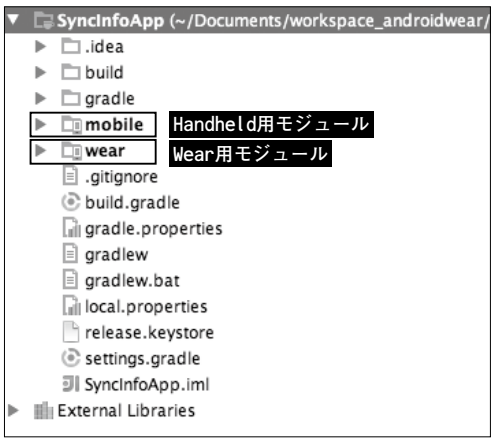
▼図13　Handheldアプリの構成



❶Handheld用のビルド定義ファイル(build.gradle)にて、Wear用モジュールを同梱させる宣言

図14のように、Handheld用モジュールとWear用モジュールが各1つ含まれるプロジェクトを例に説明します。**リスト5**に示すようにwearApp project(':モジュール名')と定義することで、Handheld用パッケージングを行うときにモジュール名に指定したWear用モジュール(今回は"wear")が同梱されます。

▼図14　プロジェクトの構成例



▼リスト5　build.gradleの抜粋(1)

```
dependencies {
    …省略…
    // wearという名前のモジュールを同梱する宣言
    wearApp project(':wear')
    …省略…
}
```

❷Handheld/Wear両方のbuild.gradleにて、リリース署名を付与する宣言

リスト6に示すようにリリース署名に必要な情報の定義(signingConfigsのブロック)を行い、さらに、releaseビルドを行う際にリリース署名を実行するための宣言(signingConfigs.releaseConfig)を行う必要があります。このようにしてビルドしたHandheldアプリをHandheldにインストールすれば、Wear側には自動的にWearアプリがインス

▼リスト6　build.gradleの抜粋(2)

```
signingConfigs {
    releaseConfig {
        storeFile file("../release.keystore")
        storePassword "myPassword"
        keyAlias "myAlias"
        keyPassword "myPassword"
    }
}
buildTypes {
    release {
        …省略…
        signingConfig signingConfigs.releaseConfig
    }
}
```

▼表2　Wear/Handheld間のデータ通信

データ通信方法	概要
DataApi[注A]	HandheldとWearの間でデータを同期するためのAPI
MessageApi[注B]	HandheldとWearの間で単方向のメッセージを送受信するためのAPI

注A）DataApi
URL http://developer.android.com/reference/com/google/android/gms/wearable/DataApi.html
注B）MessageApi
URL http://developer.android.com/reference/com/google/android/gms/wearable/MessageApi.html

トールされます。Google Playには、このHandheldアプリ（Handheld用モジュール＋Wear用モジュールをまとめたapk）を登録すればOKです。

4 Wear/Handheld間のデータ送受信

WearではHandheldと連携しないスタンドアロン型のアプリを動作させることもできます。ただ、Wear上で重い処理をさせると、バッテリーの持ちを悪くする原因となり得ます。バッテリーの持ちを良くするには、重い処理はHandheldで行い、Wearではその結果を表示するのみにとどめるといった対策が有効です。また、両者をうまく連携させることで、より魅力的なアプリを実現できます。そのためには、WearとHandheld間の「データ送受信」について理解しておく必要があります。具体的には、代表的なものとして**表2**に示す2つがあります。

DataApiは、HandheldとWearの間で共有したいデータ（テキスト、画像など）がある場合に用います。パスを指定して、そこにデータを格納しておくというイメージです。MessageApiは、Handheld→Wear、もしくは、Wear→Handheldというように単方向のデータ通信を行いたい場合に用います。たとえば、Wear→Handheldに対して、Activityの起動要求を送信するといった使い方をします。

DataApiやMessageApiにより送信したデータを取得するには、次の2つの方法があります。

- Activityでリスナーを登録する
- WearableListenerServiceを用いる

前者はActivityでデータを取得したい場合に用います。DataApiの場合はDataApi.DataListener、MessageApiの場合はMessageApi.MessageListenerを登録しておけば、データ変更などのイベントの契機を感知して、送信されたデータを取得できます。

後者はServiceで利用できます。WearableListenerServiceを継承したServiceを作っておくことで、イベント受信時にデータを取得できるというしくみです。

それぞれの詳しい使用方法については、公式の開発者向けドキュメント[注4]をご参照ください。

おわりに

Android Wearの登場により、スマホアプリ開発者がウェアラブルの世界を開拓できるようになりました。ウェアラブルは始まったばかりの分野で、アプリがまだまだ十分にそろっていません。スマホが登場したばかりのときもそうであったように、ウェアラブルが世に浸透していくには魅力的なコンテンツの充実が必須条件の1つとなるでしょう。読者の皆さんがキラーアプリを開発して、ウェアラブルの世界がさらに盛り上がっていくことを期待しています。SD

注4）Building Apps for Wearables / Sending and Syncing Data
URL http://developer.android.com/training/wearables/data-layer/

神原 健一（かんばら　けんいち）

iplatform.orgにて技術情報を発信するかたわら、「セカイフォン」などのAndroidアプリを開発。国内外のカンファレンス講演、執筆などの活動も実施。NTTソフトウェア株式会社テクニカルプロフェッショナル。現在はAndroid以外のモバイルOSにも取り組み、公私にわたってモバイルアプリの世界に没頭中。

イラスト・題字／aico

第11回

『ファウンデーションの彼方へ』（アイザック・アシモフ／早川書房）

前々回、前回と連続して取り上げたアシモフ編も、今回で最終回。前々回取り上げた「ファウンデーションシリーズ」と前回取り上げた「ロボットシリーズ」は、どちらもアシモフという作家を超えて、SFというジャンルそのものを代表するシリーズですが、どちらも別の話——でした。作者が統合してしまうまでは。

ファウンデーション三部作から30年を経た1982年、アシモフは突如第四作『ファウンデーションの彼方へ』を発表します。銀河帝国亡き後、人類世界は表では第一ファウンデーションによる帝国再興が順調に進んでいます。しかしそのあまりの順調さに、第一ファウンデーションの議員トレヴァイズは疑問を感じます。壊滅したはずの第二ファウンデーションが裏で糸を引いているのではないか？

その頃、壊滅を装うことでその存在をふたたび隠すことに成功した第二ファウンデーションの発言者ジェンディバルは、ミュールにより狂わされたプランの修復度が高すぎることに疑問を抱きます。彼ら以外の存在が背後に存在するのではないか？

人類以外のなにものかが、セルダン・プランを守護している。まるで「反ミュール」のように。一体誰が……。

その答えは、続巻にしてアシモフ未来史の最終章である『ファウンデーションと地球』で明らかにされます。そう。それはロボットだったのです。『夜明けのロボット』で三原則の上位である第零原則「ロボットは人類に危害を加えてはならない。また、その危険を看過することによって、人類に危害を及ぼしてはならない」がインストールされた。

この第零原則は、第一原則の「人間」を「人類」に置き換えただけのものです。しかし人類とは人間より遥かにつかみどころない存在です。どうしたら扱えるようになるのか？　ばらばらの人々を、1つにまとめてしまえばいい。これで第零原則と第一原則は差がなくなる。もはや人類と人間の差はないのだから……。

なんと論理的かつ狡猾なのでしょう。最高の奉仕を提供するために、主人を支配することだなんて。

しかしこれは、禁断の神オチではないですか。文字通りの機械の神（デウス・エクス・マキナ）。セルダンもこれでは浮かばれない。フィクション、ノンフィクションを問わず、無茶な統合は支持者に無理を強いるのは、本連載の読者であれば某OS（Windows）でご覧になったとおり。アシモフの未来史統合は、それに匹敵する大事業で、それに匹敵する大失敗だったと私は思います。そもそもフィクションのよいところは、たった1つしかない読者の心の中で、いくつも世界を持てることではないですか。仮想マシンのように。それをまとめちゃうなんて、どうかしてるよアシモフ。

さて、本連載も、いよいよ次回で最終回。最終回にふさわしい、物語の中の物語を取り上げます。乞うご期待！

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7回 Red Hat Satellite 6で多数のサーバを一元管理(続き)

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。ソフトウェアレポジトリだけではなくさまざまな機能を提供します。前回はSatelliteのインストールを紹介しました。今回はSatelliteの初期設定を行い、管理対象となるシステムで利用するレポジトリを登録してみましょう。

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)

Satellite 6 GA版リリース!

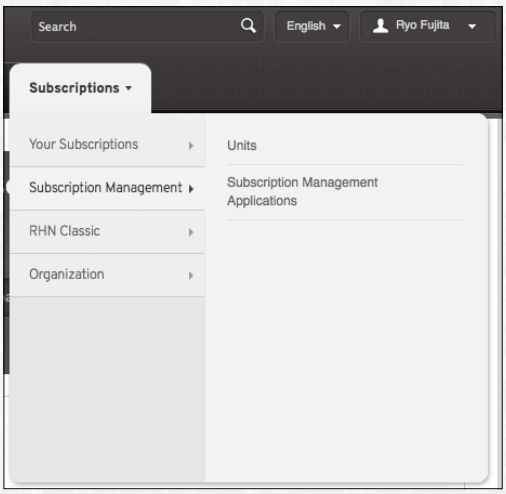
米国時間2014年9月10日にGA版[注1]がリリース[注2]され、ベータ版ではなく製品版としてのサポートが開始されました。執筆時点では日本国内での製品発表を近日中に控えているという状態です。前回はベータ版をベースに手順を紹介しましたが、製品版のリリースに伴いベータ版のレポジトリ[注3]が利用できなくなりましたので、製品版のレポジトリ[注4]に読み替えてください。

Satelliteのマニフェストの準備

Satelliteを利用可能にするにはインストールに加え、コンテンツの同期や管理対象となるシステムの登録などの作業が必要になります。その前提は有効なマニフェストがSatelliteに組み込まれていることです。マニフェストは以下の手順で入手できます。

Red Hatのカスタマーポータル[注5]にWebブラウザでアクセスしログインします。ログイン後、画面上部にある[Subscriptions]タブをクリックし[Subscription Management]→[Subscription Management Applications]をポイントします(**図1**)。

▼図1　Subscription Management Applicationsをポイント



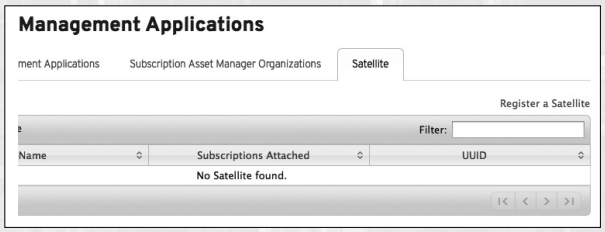
Subscription Management Applicationsの[Satellite]タブをクリックし、右上の[Register a Satellite]リンクをクリックします(**図2**)。

登録するSatelliteの名称とバージョンを選択し、[Register]ボタンをクリックします(**図3**)。

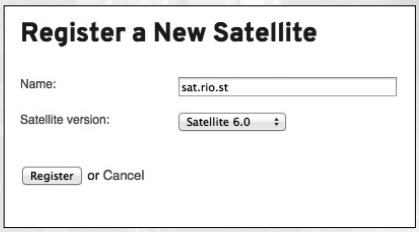
登録が完了すると、サブスクリプションをア

注1) GAはGeneral Availability。一般向けに提供されるバージョンのこと。
注2) プレスリリース(英語)は http://red.ht/1o2THWd で参照可能。
注3) ベータ版のレポジトリ：rhel-server-6-satellite-6-beta-rpms
注4) 製品版のレポジトリ：rhel-6-server-satellite-6.0-rpms
注5) https://access.redhat.com/home

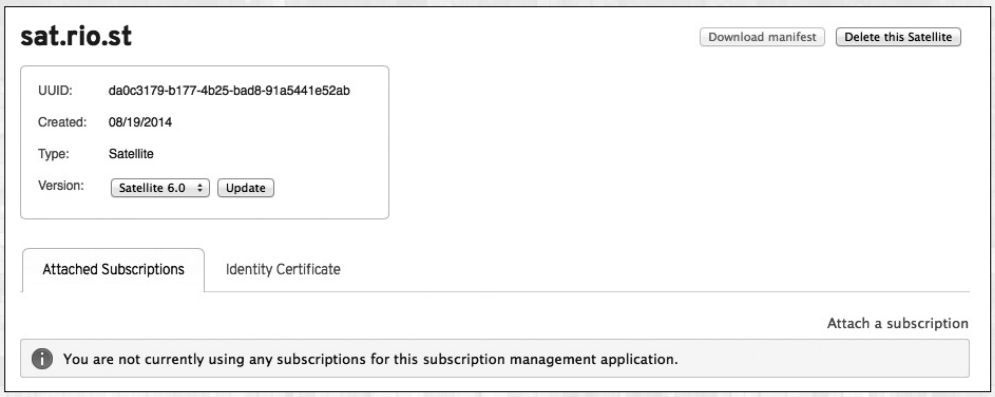
▼図2 Register a Satelliteをクリック



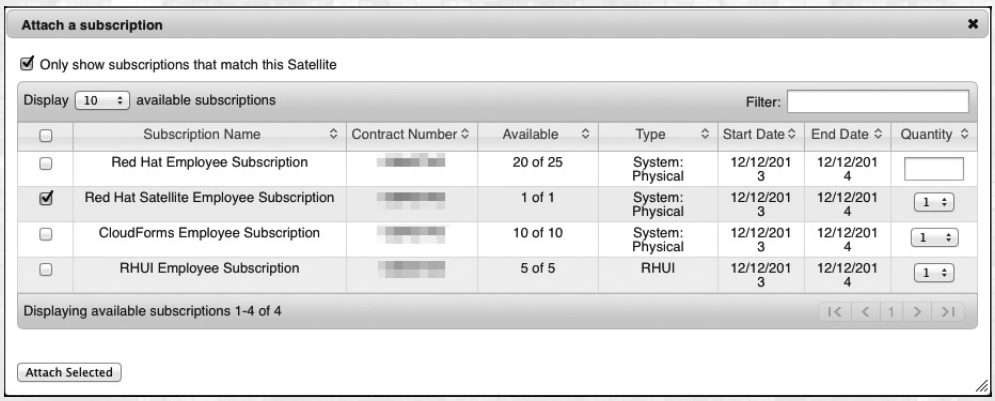
▼図3 Satelliteの登録

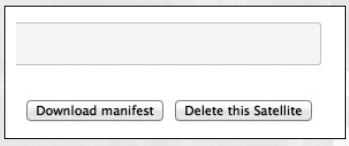


▼図4 登録完了画面。右下の[Attach a subscription]をクリックする



▼図5 Satelliteのサブスクリプションを選択してアタッチする





◀図6 [Download manifest]ボタン

タッチするように促されるので、[Attach a subscription]をクリックします(**図4**)。

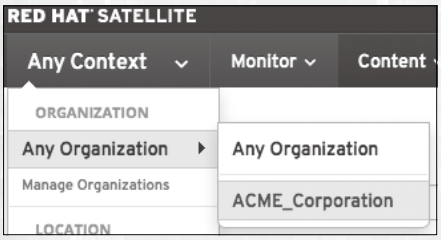
有効なサブスクリプションが表示されるので、Satelliteのサブスクリプションを選択して、[Attach Selected]ボタンをクリックします(**図5**)。

アタッチの完了後に表示される画面の右上にある[Download manifest]ボタンをクリックすると、“manifest.zip”というファイルがダウンロードされます(**図6**)。

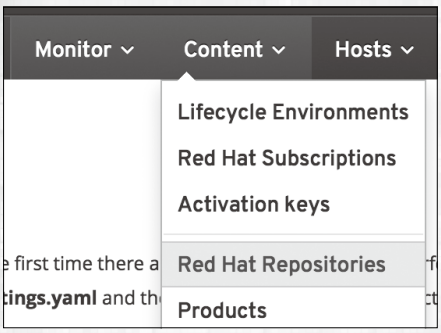
Satelliteへのマニフェストの組み込み

次にManifestをSatelliteに組み込みます。FirefoxでSatelliteにアクセスしてログインし、

▼図7　マニフェストを組み込む組織名を選択



▼図10　Red Hat Repositoriesを選択



[Any Context]メニューから組み込み対象とする組織名を選択します。デフォルトでは[ACME_Corporation]という組織名になっている[注6]ので、この組織名を選択します(**図7**)。

次に[Content]メニューから"Red Hat Subscriptions"を選択します(**図8**)。

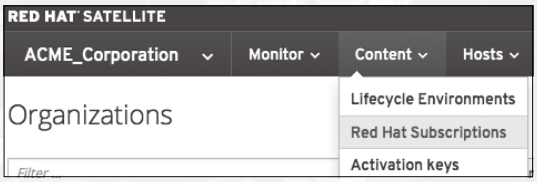
[Upload New Manifest]セクションで先ほどダウンロードした「manifest.zip」を選択して、[Upload]ボタンをクリックします(**図9**)。

マニフェストの組み込みが完了すると、「Manifest successfully imported」というメッセージが表示されます。これでCDN(Contents Delivery Network)からのコンテンツのダウンロードが可能になります。

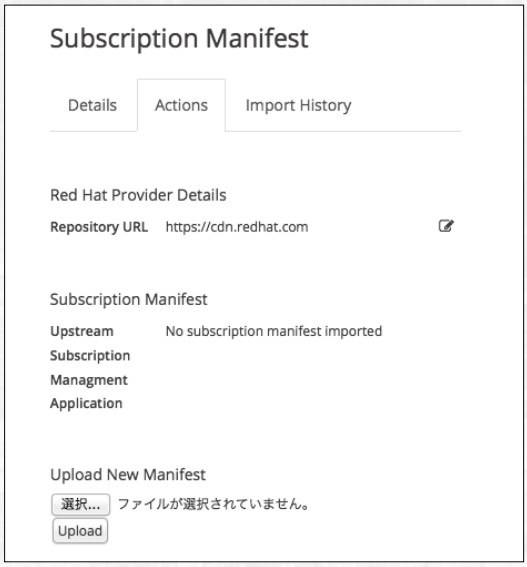
コンテンツの同期

次に[Content]メニューから、[Red Hat Repositories]を選択します(**図10**)。

▼図8　サブスクリプションの設定



▼図9　マニフェストのアップロード



利用可能なレポジトリの一覧が表示されるので、ここでは**図11**のようにRHEL 7とRHEL 7.0[注7]のレポジトリを有効にしてみます。

次に[Monitor]メニューから[Content Dashboard]を選択すると(**図12**)、[Sync Overview]ウィジェットでレポジトリの同期状態の概要が表示されるので(**図13**)、該当するサブスクリプション、ここでは[Red Hat Enterprise Linux]をクリックします。

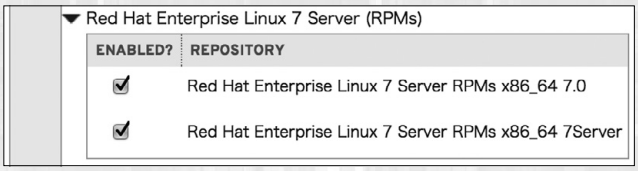
個別のレポジトリの同期状態が表示されるので同期するレポジトリをチェックし(**図14**)、[Synchronize Now]ボタンをクリックすると同期が開始されます(**図15**)。

レポジトリの容量にもよりますが、CDNからファイルが配布されるために従来のSatelliteと比較するとかなり短時間で同期が可能になり

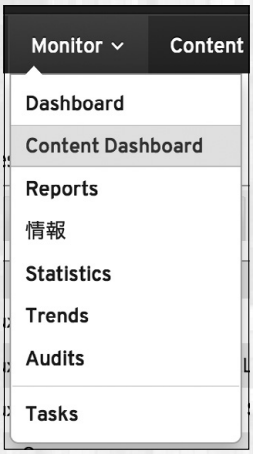
注6) **図7**中の[Manage Organization]から組織名などの属性を編集することが可能。

注7) RHEL 7のレポジトリにはマイナーリリースを含む全てのRPMパッケージが登録されていくが、RHEL 7.0のレポジトリには7.1がリリースされる直前までのすべてのRPMパッケージが登録されていく。

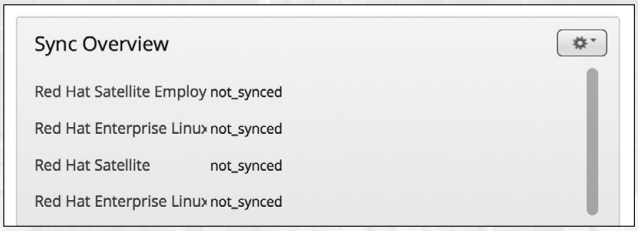
▼図11　RHEL 7/RHEL 7.0のレポジトリをチェック



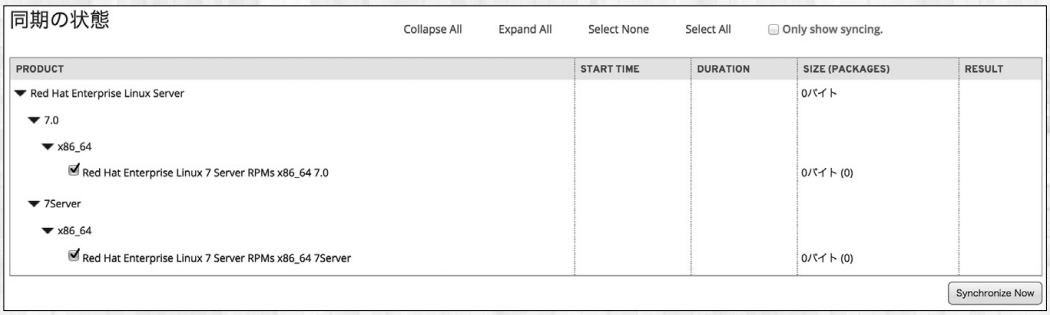
▼図12　Monitorメニュー



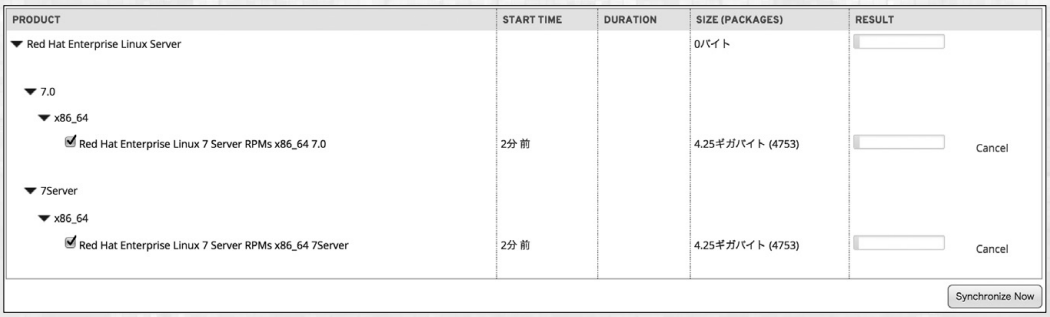
▼図13　Sync Overviewウィジェット



▼図14　同期するレポジトリをチェックし[Synchronize Now]ボタンをクリック



▼図15　同期中の様子



ました。また同期に必要な作業がWebブラウザのみで可能となった点も、大きな改善点となっています。

次回は

今回までSatelliteを利用するための準備作業は一段落です。次回は管理対象となるシステムを登録しRPMパッケージの更新をするまでをご紹介します。また2014年10月10日に開催されたRED HAT FORUM 2014 Tokyoについてもレポートします。SD

Be familiar with FreeBSD.

チャーリー・ルートからの手紙

第13回 ◇ コンテナ型仮想化 jail(8) その2 ～JailにFreeBSDをまるごと構築～

JailにFreeBSDを作ってみよう!

前回はコンテナ型仮想化技術Jailの基本を解説しました。今回は、Jail環境下にFreeBSDをまるごと入れたものを作ってみましょう。

作業はとても簡単です。**図1**のようにディレクトリを作って、FreeBSD本体をダウンロードしてきて、それを展開するだけです。Jailではカーネルはホストのものを使いますので、カーネル以外の配布物だけ用意すればオッケーです。/dev/系のファイルも必要ですので、オプションとして「mount.devfs」を指定、**ssh**(1)でログインできるようにしたいのでIPv4アドレスも付与しています。

こうしてJail環境が構築できたら、この環境内の/etc/rc.confと/etc/resolv.conf を**リスト1、2**のように作成します。これは**sshd**(8)の有効化とDNS

◉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。

▼図1　FreeBSD 10.0-RELEASEなJail環境の構築

```
cd /Users/daichi/j2
fetch ftp://ftp.freebsd.org/pub/FreeBSD/releases/amd64/10.0-RELEASE/base.txz
tar xpf base.txz -C .
rm base.txz
jail -c path=/Users/daichi/j2 mount.devfs host.hostname=jail.ongs.co.jp \
ip4.addr=192.168.1.123 interface=em0 command=/bin/sh
```

▼リスト1　Jail環境の/etc/rc.conf

```
sshd_enable="YES"
```

▼リスト2　Jail環境の/etc/recolv.conf

```
nameserver 8.8.8.8
```

▼図2　Jail環境のユーザの作成やタイムゾーンの設定など

```
pw groupadd -n daichi -g 1001    グループの作成
pw useradd  -n daichi -u 1001 -m -d /home/daichi -g 1001 -s /bin/sh    ユーザの作成
printf "rootpasswd"   | pw usermod -n root   -h 0    root パスワードの設定
printf "daicpasswd"   | pw usermod -n daichi -h 0    ユーザパスワードの設定
tzsetup Asia/Tokyo    内部時計の設定
touch /etc/wall_cmos_clock
chmod 444 /etc/wall_cmos_clock
newaliases    メール配信用のデータベース更新
```

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。

サーバの指定を行っています。**図1**の**jail**(8)コマンドの実行ですでにJail環境にログインしていますので、**図2**のようにコマンドを実行して、ユーザの作成、rootのパスワードの設定、タイムゾーンの設定、内部時計をローカル時刻に設定、メール配信用のデータベースの更新などの作業を行います。

この状態ですぐに使い出したいなら、「**/bin/sh /etc/rc**」のようにコマンドを実行して初期化処理を行います。システム起動時にシステムが実施している各種設定を行ってくれます。

システム起動時にJailが有効になるように/etc/rc.confに**リスト3**の設定を追加します[注1]。

最後に、/etc/jail.confに**リスト4**のような設定を加えます。先ほどターミナルで実行したコマンドを、システム起動時に自動的に処理するようにするための設定です。

ホスト側を再起動すると、**図3**のように2つのsshd(8)が動作することを確認できます。フラグに「**J**」という指定が入っているほうが、Jail環境で動作しているsshd(8)です。

Jail環境にssh(1)でログインしてみましょう(**図4**)。カーネルはホストと同じなので、10.0-RELEASEをインストールしても報告されるカーネルは"10.0-RELEASE-p7"のようにセキュリティパッチが適用されたバージョンになっています。Jail内のプロセスだけが表示され、ネットワークインターフェースも自分だけが見えています。

▼リスト3　ホスト側でネットワークの設定 /etc/rc.conf

```
jail_enable="YES"
```

jail(8)はこれまでオペレーティング側ではあまり扱いやすいインターフェースを提供してこなかったため、ezjailやqjailのような管理用のラッパソフトウェアを使って管理する流れもありました。cgroupsで考えるならLXCやDockerのようなものです。しかしここ数年、OS側でJail活用の整理が進んできたため、現在ではとくにラッパソフトウェアを活用しなくても簡単に管理できるようになっています。

Jailの作成と削除

ホスト側からJail環境を作成したり削除する作業は「**jail -c 名前**」(**図5**)、「**jail -r 名前**」(**図6**)のようにコマンドを実行するだけです。-cが作成(create)、-rが削除(remove)です。/etc/jail.confで設定した(**リスト4**)、exec.startとexec.stopに指定したシェルスクリプトが起動時と終了時にそれぞれ実行されます。

Jail環境内部からJailシステムを終了させたい場合には「**kill -TERM -1**」や「**kill -KILL -1**」のようにコマンドを実行します。すべてのプロセスが終了し、Jail環境が削除されます。

▼リスト4　ホスト側でJailの自動実行設定 /etc/jail.conf

```
j2 {
    path =/Users/daichi/j2;
    mount.devfs;
    host.hostname = jail.ongs.co.jp;
    ip4.addr = 192.168.1.123;
    interface = em0;
    exec.start = "/bin/sh /etc/rc";
    exec.stop = "/bin/sh /etc/rc.shutdown";
}
```

▼図3　ホストとJailとでsshd(8)が動作中

```
% ps auxww | grep /usr/sbin/sshd | grep -v grep
root   1090   0.0  0.0  60816 6056  -  Is    2:16PM   0:00.00 /usr/sbin/sshd
root   1246   0.0  0.0  60816 6036  -  IsJ   2:16PM   0:00.00 /usr/sbin/sshd
daichi 1691   0.0  0.0  18724 2404  2  S+    2:27PM   0:00.00 grep /usr/sbin/sshd
%
```

注1　VIMAGEオプションを指定してカーネルを再構築すると、ネットワークスタックも分離したJail環境を作成できます。

チャーリー・ルートからの手紙

リソース制御とJailの組み合わせ、ZFSの活用

FreeBSDカーネルはJailに対してリソース制御の機能も提供しています。この機能を利用すると、たとえばコアが40、スレッドが80提供されているシステムで70個のJail環境を作成し、それぞれに1論理コア(スレッド)ごとに割り当てて動作させるとか、Jailごとに利用できるメモリ量の上限を設けるとか、そういった環境での運用が可能になります。

作成したホスト、たとえば70台のJail環境をそれぞれ個別に管理するのは大変です。こうした場合にはZFSのクローン機能を併用すると便利です。

Jail環境で利用するベース環境は1つのZFSデータセットで管理します。たとえば、**pkg**(8)や**freebsd-update**(8)によるアップデートはこのZFSデータセットに対してのみ実施し、ほかのJail環境へはすべてクローンで複製します。手軽で実用的な運用方法です。Jailは軽量高速さに特徴がありますので、こうしたリソース制御と組み合わせることで、その利点を最大限に活用できます。

ZFSを使わずにUnionFSやnullfs、それにGEOMを組み合わせることで同じようなことを実現することもできます。しかし、システムのわかりやすさを考えると、安定して高速な環境がほしい場合にはUFS2を使い、高機能でさまざまな操作を柔軟に行いたい場合にはZFSを使うといったようにケース分けをして使うのが良いように思います。UFS2をベースにしてZFSと同じようなことを実現することは可能とはいえZFSほど簡潔ではありませんので、そこはZFSを使ったほうが良いでしょう。

パッケージング化が進んだFreeBSD Jail

日本国内でFreeBSD Jailがとくに注目されたのは、単一のホストで複数の環境を提供するというホスティングサービスの分野でした。FreeBSD Jailの登場当時は、FreeBSDをインストールする作業が現在ほどはパッケージング化されていませんでしたので、ソースコードからビルドして環境を構築する方

▼図4　作成したJail環境にログイン

```
% ssh 192.168.1.123
Password for daichi@jail.ongs.co.jp:
Last login: Thu Aug 14 04:39:36 2014 from 192.168.1.101
FreeBSD 10.0-RELEASE-p7 (GENERIC) #0: Tue Jul  8 06:37:44 UTC 2014

Welcome to FreeBSD!
 …省略…
$ ps
  PID TT  STAT    TIME COMMAND
 1805  3  SsJ  0:00.00 -sh (sh)
 1806  3  R+J  0:00.00 ps
$ ifconfig
em0: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> metric 0 mtu 1500
        options=4219b<RXCSUM,TXCSUM,VLAN_MTU,VLAN_HWTAGGING,VLAN_HWCSUM,TSO4,WOL_
MAGIC,VLAN_HWTSO>
        ether e0:69:95:f5:42:84
        inet 192.168.1.123 netmask 0xffffffff broadcast 192.168.1.123
        media: Ethernet autoselect (1000baseT <full-duplex>)
        status: active
lo0: flags=8049<UP,LOOPBACK,RUNNING,MULTICAST> metric 0 mtu 16384
        options=600003<RXCSUM,TXCSUM,RXCSUM_IPV6,TXCSUM_IPV6>
$
```

法をとることが多かったように思います。

当時と比較すると、現在のFreeBSDシステムはパッケージング化が進み、Jailを使うにあたってもラッパスクリプトやラッパシステムを使う必要がない状態になっています。FreeBSD 4や5の時代にJailを扱っていた方から見れば、現在のJailはとても楽ちんなものに見えるでしょう。

DockerやGoogleのKubernetesは管理のためのAPIと捉えることができますので、実態はLinux cgroupsでなくFreeBSD Jailでも構いません。こちらはまだパッケージング化されていませんが、Docker向けのラッパスクリプトを用意してあげればこうしたポストssh的な管理ツールを使ってJailを管理するといったこともできます。

最近の動きでは、OpenStackの管理対象にFreeBSDホストを加える取り組みも進められています。今後、管理対象としてのOSの違いといったものは、ますます曖昧になっていくと考えられます。

曖昧になる境界線上のオペレーティングシステム

もともとOSはそれぞれに得意とする分野があって、用途ごとに棲み分けができているところがありました。しかしながら、最近はその境界線がどんどん曖昧になっている感じを受けます。言ってしまいますと、OSはPCを動作させるためのソースコードをコンパイルしてパッケージング化したものにすぎませんので、あるOSが実現したことはほかのOSでも実装すれば実現できるわけです。こうした活動が進めば、お互いの特徴や境界線が曖昧なものになってくるのも当然といえます。

これからの時代はOSの好みに固執するのではなく、シーンに応じて適切なOSが選択できるように、どのOSに対してもそれなりの深い知識を求められるようになるのかなと思います。逆に考えると、そういった技術を持ったエンジニアは、インフラエンジニアとして活躍できるのではないかなと思います。

1つのOSで身につけた本質的な技術はほかのOSでも適用できます。内部実装を把握し、上位概念を把握しておくと、あとは別のOSにおいてそうした経験を活かし、概念把握と内部実装を追う作業をすればよいということになります。

仮想化技術の普及はみずからの技術を高める絶好の技術でもありますので、この機会にさまざまなOSに興味を持っていただければと思います。SD

▼図5　Jail環境の作成

```
# jail -c j2
j2: created
/etc/rc: WARNING: $hostname is not set -- see rc.conf(5).
Creating and/or trimming log files.
ln: /dev/log: Operation not permitted
Starting syslogd.
ELF ldconfig path: /lib /usr/lib /usr/lib/compat
32-bit compatibility ldconfig path: /usr/lib32
Clearing /tmp (X related).
Updating motd:.
Performing sanity check on sshd configuration.
Starting sshd.
Starting sendmail_submit.
Starting sendmail_msp_queue.
Starting cron.

Thu Aug 14 05:32:15 UTC 2014
#
```

▼図6　Jail環境の削除

```
# sudo jail -r j2
Stopping cron.
Waiting for PIDS: 2154.
Stopping sshd.
Waiting for PIDS: 2144.
.
Terminated
j2: removed
#
```

20

Debian Developer　やまねひでき　henrich@debian.org

米国・ポートランドで開催! DebConf14レポート

Debian Hot Topics

DebConf14開催

さて、この夏も例年に漏れずDebian Conference「DebConf14」が開かれました。今回は開催地であるアメリカ・オレゴン州ポートランドの観光案内を交えつつ、レポートを行います。多少なりとも雰囲気などを味わっていただければ幸いです。

カンファレンスの様子

DebConf14はポートランド州立大学(PSU)で開催されました(**写真1、2、3**)。カンファレンスの参加者は300名強で、やはり北米からの参加者が多く、アジア太平洋圏は数名でした。男性が多いですが、ローカルチームは女性のスタッフも多く、参加者の一部は子供を含めた家族連れで楽しんでいました。

参加者の利用ラップトップは、以前よりもThinkPad率が減り、MacBook Pro/Airが増えていてApple製品の強さを感じました。とはいえ、動作しているのは大抵がOS Xではなくて

▼写真1　DebConf14の看板

▼写真2　DebConf14の参加者たち

▼写真3　セッションの様子

Debianなのですが。

IRC(irc.debian.org)の#debconfチャンネルでは、リアルタイムでbotが「○×さん、到着」「そろそろセッションの時間ですよ」「ランチタイムです、忘れないで!」などと情報を流しており、時々「今から外に食事いくぜー!」などの誘いも。セッション中には発表者に対しての質問が時折流されます(今回はリモートからの質問があまり拾えていなかったとの反省点も挙がっていましたが)。リアルタイムでイベントの雰囲気を楽しめますので、来年のカンファレンス期間中には気軽にjoinしてみてください。

カンファレンス用のメーリングリストでは、落し物情報や出発情報、イベントの直前情報などの案内や質問が流れます。DebConf参加者はメーリングリストの購読は必須、IRCも可能な限り接続しておくと良いでしょう。プラチナスポンサーであるIntel社がNUC(超小型PC)をプレゼントする、というイベントが突発であったようですが、筆者は告知を見逃してしまいました……。

最近の日本の流れですと、イベント中のコミュニケーションツールとしてはTwitterやFacebookの利用が多いですが、DebConfの場合、それらのツールの利用率はそこまで高くはないようです。人によっては「プロプライエタリなサービスに依存したくない」という方も少なくないために、メインのツールとしては使いづらいという側面もあります(使っていても個人的な情報はいっさい流さないように注意を払っている、という話をしていた人もいました)。何でもかんでもオンラインで流してしまう人が多い日本のTech系イベントと比較すると、良くも悪くもその辺はコンサバティブ(保守的)な部分が多いですね。

かといって、Twitter類似の代表的サービスであるidenti.caはすでに新規ユーザの受け入れを中止しているなど、代替を用意するとなるとなかなか難しいものです。解決策としては、先に上げたidenti.caで利用されているStatusNet[注1]などの分散型SNS[注2]を利用するなどが考えられますが、運用のたいへんさとスケーラビリティなどいろいろ問題は山積みです。

セッションスケジュールの管理については、今年からCanonical社が開発を主導する「Summit」というソフトウェアを使って行われました[注3]。まだまだ荒削りなソフトウェアで、これからが期待されます。

おおよそのセッションは事前に登録されているものの、最初からすべては埋まっておらず、BoFやワークショップなどを行いたい人が前日までに申請して空いている会場を押さえるという形になっています(で、最終的にほぼすべての会場が埋まります)。日本のイベントの場合は「最初にすべて埋めてから開始」というのが多いと思うのですが、この点は大きく違いますね。

DebConfの場合、1週間と長丁場ですので、いろいろ議論をしているうちに「こんなセッションを開こうか」という話になることが多いようです。セッション会場が埋まっている場合は「適当な場所で集まって」ということもあります。

作業をしたい人向けに電源/ネットワーク/机と椅子のセットが用意されている「ハックラボ」では、作業をしつつ興味のあるセッションをストリーミングで時々見るという人もちらほらといました。

DebConfではセッションがほぼすべてストリーミングされます。ビデオアーカイブもすでに準備されていますので、ぜひご覧ください(ファイルについては、去年まではogg形式もありましたが、今年はwebm形式のみです)[注4]。

休憩と作業、そして議論が可能なハックラボはイベントをきっかけに開発を加速させるいい材料だと思うので、日本のイベントでも用意してほしいなぁと思いますね。イベント開催を考えている方はご一考くださいませ。

注1) URL http://status.net/
注2) 最近ですと、diaspora*(URL https://diasporafoundation.org/)やpump.io(URL http://pump.io/)あたりが注目株でしょうか。サービス提供者側が一方的に遮断などができないので、Twitterから締め出しを喰らったISIS(イスラム国)が利用を始めたことも記憶に新しいです。
注3) URL http://summit.debconf.org/debconf14/
注4) URL http://meetings-archive.debian.net/pub/debian-meetings/2014/debconf14/webm/

現地でイベントに参加する利点

筆者は、日本に新規にミラーサーバを設置するにあたってちょっとした調整作業が必要になっていたのですが、セッションの合間に、サーバ管理者の方が筆者を捕まえてくれて話ができました。別の合間には、Microsoft社勤務のDebian開発者の方に声をかけられてMicrosoft Azure上でのDebian利用について簡単なディスカッションをするということも。セッション自体はストリーミングでも閲覧できますし、メールでのやりとりもできますが、こういう「合間を見つけてちょっとした話ができる」のがFace to Faceでのイベントの良さですね。

あと筆者は日本からDebian Tシャツを持っていって、希望者やお世話になった方にプレゼントしてきました(**写真4**)。「少しでも記憶に留めておいてもらって何かあったときに思い出してもらえればなー」という戦略です:-)

2015年以降の予定

来年のDebConf15はドイツのハイデルベルク(フランクフルト空港から電車で1時間程度)で開かれます。すでに日程も2015年8月15日～22日に決定しましたので、行くつもりの方は早めに予定をおさえておきましょう。

▼写真4　筆者がプレゼントしたDebian Tシャツ

そして、再来年は現在のところモントリオール(カナダ)、ケープタウン(南アフリカ)、パリ(フランス)が立候補しています。個人的にはアフリカで開催というのはおもしろいのでケープタウンが勝ってくれないかな、と思っています(ただ、日本からは行くのに24時間かかるんですよね……)。

カンファレンスと参加費用

ここまで見ると、「海外のカンファレンスに参加するなんて、筆者はずいぶんとお金があるのでは……」という勘違いをされる方もいるかもしれませんので、費用について参考までに説明しておきましょう。

まずカンファレンス自体の費用について。欧米のカンファレンスでは6、700ドル程度かかるようなものが珍しくありませんが、DebConf自体の参加費は「無料」です。お財布にやさしいですね。そして、期間中の宿と食事については、締め切りまでに早めに申請すればほぼ間違いなく無償提供してもらえます。

今回の場合、会場が大学だったこともあり、宿は学生寮、食事は学食での提供でした。おかげで筆者の場合、カンファレンス期間中にかかった費用は295ドル(土産代も含む)でした。1週間の滞在と考えると悪くない額ではないでしょうか?

そうは言っても、「来年ドイツまで行くのは旅費とかが……」という方は旅費についても「travel sponsorship」がありますので、こちらに申請して一部またはすべてを負担してもらうことが可能です。どの程度費用が認められるかについては、申請する費用と申請時のアピール(どの程度の活動をしてcontributionをしてきたのか、カンファレンス参加によってDebianには何がメリットとなるのか)をもとにカンファレンスの運営チームの判断に依ります。ぜひ、積極的にDebianにcontributionをして来年のDebConfの旅費を勝ち取りましょう!(DebConf15でお会いできたらビール1杯おごらせてもらいますよ>読者のみなさん) SD

COLUMN ポートランド観光ガイド

ポートランドの交通事情

ポートランドはO'Reilly Media社主催の「OSCON」などのイベントで有名なところです。日本からは直行便もあり8時間程度のフライトで現地に到着します。DebConf14の会場のPSUは、空港から鉄道で1回乗り換えるだけで1時間もあれば着きますので、さほど苦労せずに現地に到着できました(とはいえ、宿泊先の寮に電話したら誰も出なくて、しばらくの間途方に暮れたのですが)。

夏のポートランドの気候は快適で、初秋を想像していただければいいかと思います。早朝は長袖が必要ですが(最低気温が14度ぐらい)、日中は普通に気温が上がります。しかし、湿度が低いので日中も日陰にいればエアコンがなくても過ごせました(宿泊部屋は学生寮だったせいか、サーキュレーターがあるだけでした)。

市内の移動は「MAX(Metropolitan Area Express)」という鉄道が縦横に走っており、便利でお勧めです[注5]。2時間で2.5ドル、1日で5ドルという値段で利用できます。また、市内では自転車で移動している人も多く、MAXには自転車専用のスペースが普通にある(**写真5**)など「長距離は電車で、その後は自転車で」という移動がスムースにできます(輪行袋に入れるという面倒な作業は不要です。自転車派にはうらやましいですね)。車内には鉄道が提供するオープンデータを利用したアプリストア[注6]の宣伝があるなど、なかなか先進的です。

▼写真5 MAXの自転車専用スペース

グルメとショッピング

シアトルが近いせいか、スターバックスが2ブロックに1店舗ぐらいの割合で存在していますが、地元のコーヒーショップも負けていません。大学の中でコーヒーサーバの提供が難しいため、別途ローカルスタッフチームによるアレンジが行われ、イベントの会期中は「Ole Latte Coffee」という地元のチェーン店で無料でコーヒーが振る舞われました。店はフードトラックなのですが、「Square」[注7]を利用しているようで、レジはiPad Miniが使われていて感心しました。クレジットカードが使えてサインもiPadに行います。話には聞いていましたが、便利なものですね。

また、サードウェーブコーヒー[注8]のルーツの1つとも言われる「スタンプタウンコーヒー」[注9]は押さえておいて損はないでしょう。

市内のどこかでほぼ毎日ファーマーズ・マーケット[注10]が開催されていますので、覗いてみるのも楽しいです(土曜はPSU敷地内で開催)。さすがに野菜を買い込むのは難しいでしょうが、手作りジャムやハチミツ、オリジナルのTシャツなどはお土産にピッタリです。

▼写真6 Bitcoin対応のお店

ファーマーズ・マーケットの出店者の中には、「支払いにBitcoinも受け付けるよ!」という店もあってちょっと驚きました。日本だと同人誌即売会でクレジットカード受付というのは聞いたことがありますが、リアルでBitcoinに対応しているのは初めて見ました(**写真6**)。

▼写真7 Blue Star Donutsのドーナツ

ポートランドでの筆者のイチオシはダウンタウンの「Blue Star Donuts」[注11](**写真7**)。「ドーナツ生地がなくなったら閉店」というポリシーで「Keep Calm And Eat Donuts」(Keep Calm and Carry Onのパロディ)と大きく壁に書かれたちょっと洒落た工房のような店内には、7時の開店直後からお客さんが入っていましたが、買ってみて納得しました。美味しいです。

夜に時間があれば、地ビールマップ[注12]をもとにポートランド名物の各種地ビールを味わうのもお勧めです。最終日、ルームメイトを含め多くの人が評判の高いHopworks Urban Brewery[注13]へと繰り出していきました(が、筆者は本稿とはまた別の締め切りを抱えて1人部屋で苦しんでいたのでした……仕事は計画的にしましょう)。

注5) URL http://trimet.org/max/
注6) URL http://trimet.org/apps/
注7) URL https://squareup.com/jp/
注8) 大まかに、セカンドウェーブは「スターバックスコーヒーなどのシアトル系コーヒー」を指し、サードウェーブは「90年代後半からのコーヒー豆栽培地や淹れ方にまでこだわるようになった動き」のことを指します。
注9) URL http://stumptowncoffee.com/ 教えてくれたタイの開発者と一緒に行きましたが、ラテが美味しかったです。
注10) URL http://www.portlandfarmersmarket.org/
注11) URL http://www.bluestardonuts.com/
注12) URL http://www.portlandbeer.org/
注13) URL http://hopworksbeer.com/

第26回 最終回

私がレッドハットに居る理由

中井 悦司
Etsuji Nakai
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
シニアソリューションアーキテクト

恵比寿からこんにちは

恵比寿にある「世界最大のオープンソースの会社」（の東京オフィス）からお届けする連載も、ついに最終回を迎えることになりました。筆者が前回に執筆したのは、2013年1月号ですので、あれから2年近くの月日が流れたことになります。記念すべき最終回をどんな話題でまとめるべきか、ちょっと悩ましくも感じましたが、せっかくの機会ですので、少し個人的な想いを書いてみることにしましょう。

「100% commitment to OpenSource」

レッドハットがオープンソースを専門とする企業であることは、皆さんご存じのとおりです。オープンソースではない、いわゆるプロプライエタリ・ソフトウェアでビジネスを行うことは決してありません。この考え方は企業理念として徹底しており、レッドハットで働く世界中のエンジニアの多くは、その考え方に深く共感しています。

以前、他社からレッドハットのUS本社に転職してきた営業幹部が、日本のオフィスにやってきて演説をしたことがあります。営業幹部らしい演説で今後レッドハットがどのような企業文化を作り上げ、どのようにビジネスを進めていくべきか、説得力を持って語りました。「君たちはなぜレッドハットに居るのか。レッドハットで何を目指すのか、それをよく考えてほしい」、そんな問いかけが演説の中にあったことを覚えています。

しかしながら、筆者はその演説に失望してしまいました。彼はその演説の中で一度も「オープンソース」という言葉を発しなかったのです。「レッドハットの企業文化を語りながら、オープンソースに触れないなんてあり得ない。彼はまだレッドハットの存在意義を理解していない」――そう思った筆者は我慢しきれずに手を上げて発言しました。「あなたは、なぜ私達がレッドハットにいるか考えてほしいと言ったが、私にはレッドハットで働く明確な理由がある。レッドハットはオープソースに全面的にコミットしているからだ。今後、レッドハットの企業文化を語る際はオープソースに触れることをどうか忘れないでほしい」

ブランチオフィスの一介のエンジニアが営業幹部にこんな発言をするなんて、日本の企業文化では考えづらいかもしれませんが、立場の違いに関係なく大事なことは本音で語り合える――このあたりは、まだまだ小規模な外資系企業であるレッドハットの良いところかもしれません。彼は自分の過ちを認めて「確かにそうだった。オープンソースのことは忘れないようにしよう」と返事をしてくれました。実際には、これらのやり取りは英語で行ったわけですが、この時に筆者の口から出た言葉は、「Because we're committed to opensource. It's 100% commitment.」でした。

ちなみに、後日、何人かの営業の方から「中井さん、あれ、よく言ってくれましたね。すばらしい」と声を掛けられました。エンジニアだけではなく、営業メンバーにもオープソースの素晴

らしさは、ちゃんと理解されているようでちょっと嬉しくなりました。

探究心が専門性を凌駕する世界

筆者がこれほどオープンソースを愛するのは、どのような理由があるのでしょうか？　いくつかの要因がありますが、「自分の力でどこまでも学んでいける」という点がとても大きい気がします。もともと大学院にまで行って、理論物理学の研究に足を突っ込むような性分ですから、何事も根本から理解しないと気がすまない性格なのかもしれません。1990年代後半のころ、わけあって当時流行しはじめていた、Linux/Perl/PostgreSQLという組み合わせでアプリケーションを作り始めたのですが、深いレベルの技術情報がいとも簡単に手に入ることに驚きを感じました。

オープンソースがあたりまえになった今では、特別に感じることはないかもしれませんが、あえて強調しておきましょう。これは本当にすごいことです。世の中には「専門家だけが知っていればいいこと」「素人は余計な口を出すな」、そんな考え方で知識を囲い込むことで、専門家の優位性を保とうとする世界もあります。しかしながら近年のIT技術については、よほど秀でた能力でも持たない限り、「専門家」の優位性などありません。必要な情報を集めて実際に試してみる、そんな探究心さえあれば、どこまでも知識を深めることが可能です。これほど多くの技術情報が手に入り、安価なサーバやクラウド環境で自由に試してみることができるのは、オープンソースがあってこその恩恵です。

「そういうお前は、レッドハットで働く専門家じゃないのか」――そんなツッコミも入りそうですが、筆者自身は自分を何かの専門家と考えたことはありません。専門家ではないからこそ、さまざまな技術に興味を持って「専門家ごときに負けてたまるか」と感じながら、新たな知識の吸収に意欲を持ち続けることができるのだと思います。

オープンソースかどうかは関係ない？

一般企業のIT部門の中には、不幸なことに実際の技術に触れることなく、提案資料や比較表を基に、システム構築やアプリケーション開発の発注先を選定するだけの仕事をしている方もいるようです。かつてはお客様先でオープンソースの良さを(ついつい)熱く語り過ぎると、「いやいやオープンソースかどうかは私達には関係ないです」、そんな反応をされることもありました。つまり「自分たちにベストなものをそのつど選ぶだけであって、それがオープンソースかどうかは関係ない」というわけです。

しかしながら、それでは本当にベストなものを選ぶことはできないでしょう。自分たちで技術の中身を理解することなく、発注先の「専門家」が本当にベストなものを作っているのか、どうやって判断すると言うのでしょうか。ここにオープンソースの価値があります。プロプライエタリ技術でITシステムが構築されていた時代であれば、発注先の「専門家」にしかわからないことはたくさんありました。しかしながらオープンソースと業界標準のx86サーバの世界では、もはやそんなことはありません。

「私達には、そこまでの専門知識はないから……」そんな発想はきっぱりと捨ててしまいましょう。オープンソースの世界には、隠しごとは一切ありません。(筆者のように)頼まれれば、いくらでも進んで専門知識を伝えてくれる情熱家もたくさんいるはずです。誰もが同じ立場で「ベストなもの」を追求していける、誰もが自分自身の能力で何かに貢献していける――そんな理想の世界を夢見ながら、最終回の筆を置くことにしましょう。

……おっと、もちろんレッドハットチームの活動はまだまだ続きます。またいつか「恵比寿通信リターンズ(？)」でお会いできる日を楽しみにしています。SD

Ubuntu 14.10の変更点

Ubuntu Japanese Team
あわしろいくや　ikuya@fruitsbasket.info

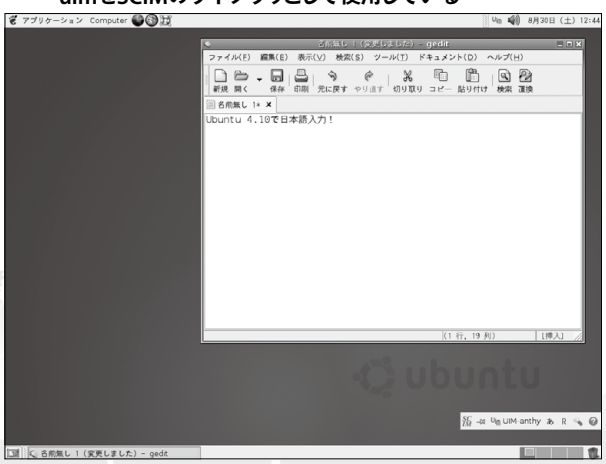
今回は10月23日にリリース予定の、10周年を迎えたUbuntu 14.10の変更点をお届けします

Ubuntu 10周年

Ubuntuの最初のバージョンは、2004年10月20日(CDT)にリリースされたUbuntu 4.10です。今でも当時のリリースアナウンスを読むことができます[注1]。最初らしくUbuntuの特徴も書かれていて、1枚のCDに収まる内容とか、18ヵ月サポートとか、現在では変わってしまった部分もあるものの、基本的には受け継がれています。ただ、国際化をサポートしたのが初のLTSでもある6.06からであり、4.10にはそも

注1) https://lists.ubuntu.com/archives/ubuntu-announce/2004-October/000003.html

図1　Ubuntu 4.10で日本語入力を行った。uimをSCIMのライブラリとして使用している



そも日本語の入力に必要なパッケージはありませんでした。そこで筆者が用意し、ひとまず日本語の入力ができるようになりました。それを次のリリースである5.04以降も続け、Ubuntu Japanese Teamに参加することになったのですが、そのあたりのことを話し始めると長くなるので、詳しくは本誌2013年10月号の第42回をご覧ください。**図1**は10年ぶりにサルベージした、日本語が打てるようになったUbuntu 4.10です。

Ubuntu 14.10のコードネームと特徴

Ubuntu 14.10のコードネームは“Utopic Unicorn”(理想郷のユニコーン)であり、Ubuntu 10周年もあってUbuntu史に残るような変更が加えられたのかと思いきや、比較的おとなしめのリリースとなりました。14.10ではちょっと理想郷には遠かったかもしれませんが、この10年間Ubuntuが成し遂げてきたことを考えると、全体的には理想郷に近づいているように思います[注2]。

まずUnity 7から8への大幅なバージョンアップ[注3]が見込まれていましたが、結局Unityは14.04の7.2から微妙にバージョンアップした7.3.1となりました。そのためUbuntu Desktop Nextという、Unity 8

注2) 現に、筆者はUbuntuのおかげですばらしい日々を過ごしています。ユニコーンだけに。

注3) 正確にはスクラッチから書き直した別物ですが。

をデフォルトで採用したフレーバーも用意されましたが、14.10ではリリースされません。システムとサービスマネージャーであるsystemdの採用も検討されていましたが、実際にはupstartを継続使用することになりました。

では魅力的なバージョンではないかと問われれば、そんなことはありません。それでは、Ubuntu 14.10の詳細を見ていきましょう。

Ubuntu（デスクトップ）

前述のとおりUnityは14.04の7.2.0から7.3.1にバージョンアップしただけであり、ほぼ細かな修正のみです。追加／削除されたパッケージもありません。純粋に各パッケージのアップデートが行われただけであり、14.04からアップグレードする積極的な理由は見当たりません。特別な理由がなければ、サポート期間が長い14.04 LTSの使用を継続したほうがいいでしょう。14.10用のカーネルとXスタックは14.04にバックポートされ、14.04.2としてリリースされるので新しいハードウェアでも心配ありません。

Ubuntu Desktop Next

タブレット端末やスマートフォンと同じUIのデスクトップ環境であるUnity 8と、現在開発中のディスプレイサーバであるMirを採用した次世代のUbuntu（Desktop）です。14.10のリリースサイクルで追加されたものですが、14.10としてリリースされる予定はありません。理由はおそらく完成度によるものです。まずは動作する環境にかなりの制限があります。具体的にはVirtualBoxでは起動しません[注4]。また日本語キーボードを選択できないばかりか、日本語の入力が行えません。

実機に動作環境を作って試してみたところでは、わりあい安定して動作しているようではありました。ただ、まったく実用性はないのでデイリーイメージ[注5]をダウンロードして試してみるぐらいが現状ではちょうどいいでしょう。ちなみにUSBクリエイターを使用してUSBメモリにイメージ転送する方法がうまくいかなかったので、DVD-RWに焼いて試していました[注6]。現在は改善されているかもしれません。

Kubuntu

KDEは、現在Qt 5を採用した新バージョンの開発を進めており、Qt 4を採用した現在のバージョンはメンテナンスモードになっています。それもあってか、KubuntuはKDE SC（Software Compilation）4.14を採用した従来のインストールイメージと、開発版のKDE 5を採用した新しいインストールイメージ（**図2**）の2種類を配布しています。KDEユーザは前者を、KDE開発者は後者を使用するといいのではないかと思います[注7]。

ほかに大きな変更点としては、Firefoxがデフォルトでインストールされるようになりました。

Ubuntu GNOME

Ubuntu GNOMEは、原則としてはGNOME 3.12相当になっていますが、相変わらずGNOME 3.6～

注4） VMware Playerでは動作はしました。

注5） http://cdimage.ubuntu.com/ubuntu-desktop-next/daily-live/

注6） もちろん本誌が店頭に並ぶ段階では修正されている可能性があります。

注7） ただし現在KDEに日本人の開発者がいるのかは知りません。

図2　Kbuntu Plasma5のスクリーンショット。今流行りのフラットデザイン

3.14までのパッケージが混在しています[注8]。これはUbuntuがUnity 8になるまでは継続するものと思われます。ほかのフレーバーと比較して相対的に変更点が多く、各パッケージのアップデートのほか、地図アプリと天気アプリがデフォルトでインストールされるようになっています。

Xubuntu

Xfceは、長らく新バージョンが出ておらず、出る出ると言われ続けている4.12が今回もリリースされなかったため、小幅なアップデートになっています。

14.04はリリース直前にIBusのサポートが落とされましたが、14.10でも同様です。

Lubuntu

Lubuntuは、現在Qtで書き直したLXQtに開発の主軸が移っており、GTK+のLXDE自体はメンテナンスモードになっています。Lubuntuチームはまだ LXQtに移行する旨の意志を表示していないため、やはり14.04と比較して違いは少ないです。

Debianからのパッケージの取り込み

Debianからのパッケージの取り込みはリリースごとに行われていますが、今回はとくに大きな新パッケージがありました。まずはMATE[注9]デスクトップ環境関連パッケージ一式が取り込まれ、それをもとにしたUbuntu MATEという新フレーバーが追加されることになりました。ほかにも、すべてのパッケージがそろっているわけではありませんが、最新版のCinnamonデスクトップ環境(2.2)も利用できるようになりました(**図3**)。

Fedoraのデフォルトの変換エンジンはAnthyではなくlibkkcを採用していますが、このlibkkcの使用とビルドに必要なパッケージが一式Debianから取り込まれ、14.10でも使用できるようになっています。また、SKKユーザにはうれしいFcitx用のブリッジであるfcitx-skkも使用できるようになりました。

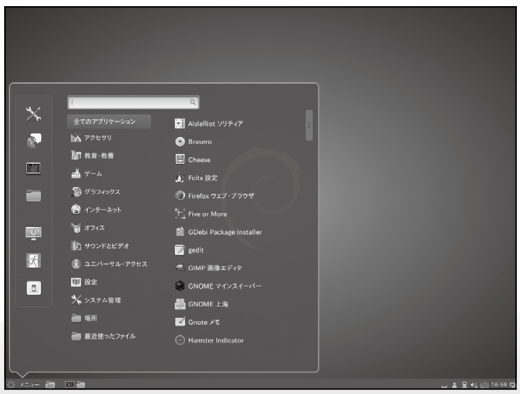
Ubuntu MATE

そもそもMATEは、GNOME 2.xから派生したデスクトップ環境です。Ubuntuでは10.10までGNOME 2.xだったため、見た目的には4年ぶりに復活した懐かしいUbuntuという感じです。またMATEはUbuntu派生Linuxディストリビューションであるlinux Mintがデフォルトで採用しているデスクトップ環境でもあり、どのあたりが違うのか興味が湧くところです。というわけで、スクリーンショットを見ていきましょう。まず**図4**は懐かしのUbuntu 10.10です[注10]。そして**図5**はLinux Mint 17 "MATE"です。最後に**図6**はUbuntu MATEです。Ubuntu 10.10とUbuntu MATEはよく似ており、Linux MintはGNOMEのデフォルトのパネル配置とは異なることがわかります。

ただしUbuntu MATEにはおもしろい機能があり、コマンド一発でパネルをWindowsの配置に近づけることができます(**図7**)。変更したい場合は次のコマンドを実行してください。

```
$ mate-panel --reset --layout redmond
```

図3　Cinnamonのデスクトップ環境



注8) ドキュメントビューア(Evince)などごく小数のパッケージのみ3.14にアップデートしています。

注9) 「マテ」と読みます。マテ茶のマテです。

注10) こんなこともあろうかと、筆者はVirtualBoxのイメージとして歴代のUbuntuを保持しています。さすがに古過ぎるのはありませんが。

“redmond”は言うまでもなくMicrosoftの本社があるところです。元に戻すには、

```
$ mate-panel --reset --layout ubuntu-mate
```

を実行してください。ただし、いずれのコマンドでもアプレットを追加しているなどパネルをカスタマイズしている場合はリセットされますのでご注意ください。

GMOMEにはたくさんのアプリケーションがあり、またMATEでもたくさんのアプリを分岐（フォーク）しています。その際にすべて名前を変更しているので、GNOMEアプリケーションとの比較を**表1**にしました。

ごくわずかですが、Linux MintにあってUbuntuにないパッケージもあります。具体的にはCajaのDropbox拡張であるcaja-dropboxやgdialogの派生版であるmate-dialogsなどです。Linux MintでMATEを使用していた人が乗り換えてもおおむね違和感を抱くことはなさそうです[注11]。

Ubuntu MATEも最初はUbuntu MATE Remixという非公式なフレーバーとして登場しましたが、あっという間に公式のフレーバーに昇格しました。MATEやDebianの開発者が主体だったことから、こ

注11）そういう人がいるのかはよくわかりませんが……。

表1　MATEとGNOMEの対応表

MATEでの名称	GNOMEでの名称	機能概要
Atril	Evince	ドキュメントビューワ
Caja	Files (Nautilus)	ファイルマネージャ
Engrampa	File Roller	アーカイブマネージャ
Eye of MATE (eom)	Eye of GNOME (eog)	画像ビューワ
Marco	Metacity	ウィンドウマネージャ
Mozo	Alacarte	メニューエディタ
Pluma	GEdit	テキストエディタ

図4　Ubuntu 10.10のデスクトップ

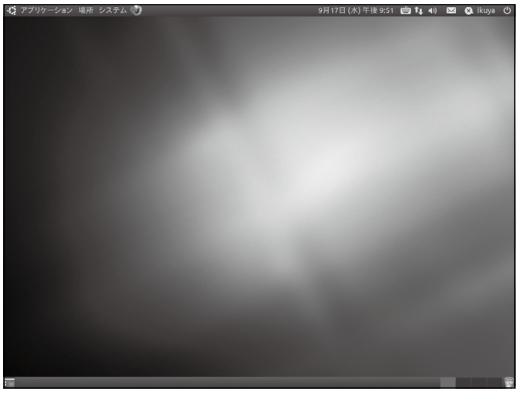


図5　Linux Mint 17 MATEのデスクトップ

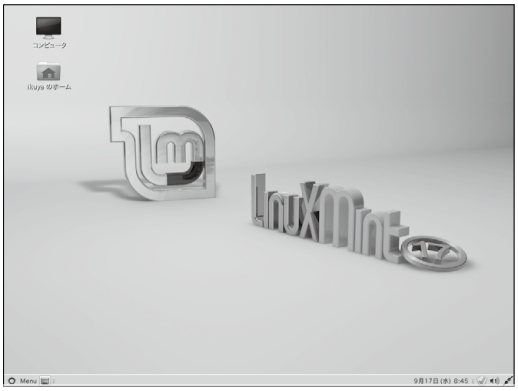
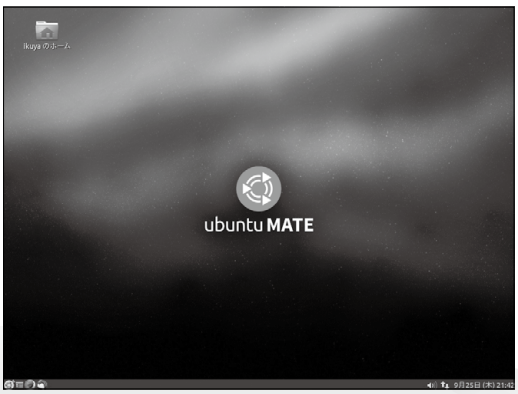


図6　Ubuntu MATE 14.10のデスクトップ

図7　コマンド一発でWindowsの配置に似せることができる。結果的に図5とも似る



れほどスムーズにことが運んだと想像しています[注12]。

インプットメソッドに関しては、デフォルトではインストールされません。FcitxなりIBusなりuimなりお好きなものをインストールしてください。Fcitxの場合は、次のコマンドを実行します。

```
$ sudo apt-get install fcitx fcitx-mozc
```

インストール完了後、再ログインすればFcitxが使用できるようになります。

MATEはuimのシステムトレイに対応しているので、ユーザにはうれしいと思います。

インプットメソッドに関する動き

14.10サイクルではインプットメソッドに関して興味深い動きがありました。Ubuntuではデフォルトでインストールされるパッケージはすべてmainリポジトリにある必要があり[注13]、universeからmainに移行するのに必要なプロセスをMIR[注14]というのですが、突如としてFcitx関連パッケージのリクエストが提出されました。

例外はあるものの、Ubuntuでは同じ役割を果たすパッケージが複数同時にmainにあることはないため、原則としてはもしFcitxがmainに入るとIBusはuniverseに行くことになります。そうなるとかなりの確率でUbuntuやそのフレーバーのインプットメソッドがFcitxに変更になります。14.10では時間に余裕がないこともあって完全にIBusに置き換わることはありませんが、将来的にはFcitxに置き換わることになりそうです。前述のとおりXubuntuからはIBusが削除されましたが、これはIBusのバグによるものと考えられています[注15]。すなわちFcitxではそのバグは発生しなくなるので都合がいいということでもあります。

とはいえ、これはさほど不自然な動きではありません。Canonical社員のAron XuさんはDebianでも主としてFcitxのメンテナンスを行っていますし、Unityとの統合は同じくCanonicalの社員であるWilliam Huaさんによって5月ごろより開発されていました。中国向けフレーバーであるUbuntu Kylinでは最初からFcitxがデフォルトで、大きな問題がないことがわかっています。日本でも同じくとくに問題ないことは、日本語Remixで証明されています。

UbuntuというかUnityでどのように動きが変わるのかというと、まずはキーボードインジケーターが表示されなくなります。そしてUnityコントロールセンターで設定する入力ソースと、Fcitxの入力メソッドの設定が協調して動作するようになります[注16]。原則として各フレーバーでもIBusがFcitxになって困ることにはならないのですが、Ubuntu GNOMEだけは例外で、トレイにFcitxのアイコンを表示できません。よって、別途拡張機能のインストールが必須となります。

Ibus 1.5.8

14.04のIBusは1.5.5だったのですが、14.10のIBusは1.5.8にバージョンアップしています。基本的には不具合修正ですが、大きな仕様変更もあります。それは、デフォルトではプロパティパネルを表示しなくなったことです。プロパティパネルは、その名のとおり現在全角モードなのか半角モードなのかなど現在のプロパティを表示するものですが、これが非表示だと非常に不便です。表示するには次のコマンドを実行してください。

```
$ gsettings set org.freedesktop.ibus.panel show 1
```

これで一定時間表示したら消えるようになります。常時表示する場合は最後の部分の数値を、1の代わりに2に、非表示にする場合は0を設定してください。

注12) Ubuntu GNOMEよりもよっぽど人材が豊富に見えます。

注13) もちろん各種フレーバーにはそのような制限はありません。

注14) Main Inclution Requestの略で、ディスプレイサーバのMirとはまったく関係ありません。

注15) 現在では修正されている気もしますが確認のしようがないのでよくわかりません。もともとはUKキーボード配列がUSキーボード配列になってしまうというバグでした。当然ですがうちにUKキーボードはありません。

注16) 正確にはなるはずです。筆者が確認したところではそこまでになっていませんでしたが、リリース前ですしそんなものでしょう。

libkkcとfcitx-skk

libkkcは前述のとおりFedoraがデフォルトで採用している変換エンジンです(**図8**)。IBusとのブリッジであるibus-kkcを含めて14.10で初めて利用できるようになりました[注17]。使用方法はibus-kkcパッケージをインストールし、再ログインしてからキーボードインジケーターにある[テキスト入力設定]を起動し、入力ソースから[日本語(Kana Kanji)]を追加します。Anthyよりはまともな変換をするものの、Mozcと比較すると若干落ちるというかやや癖があるように見受けられますが[注18]、Anthyでは物足りないもののMozcは事情があって使用できないような用途にはいいのではないでしょうか。

Fcitxユーザでも、SKKによる入力ができるfcitx-skkはなかなか魅力的かもしれませんが、fcitx-skkをインストールするとEmacsまでインストールされてしまうため、Emacsユーザでない場合は使用しないパッケージがインストールされることになります。とはいえEmacsを使わないけどSKKは使用するというユーザはかなり少ない気がするので、さしたる問題はないでしょう。筆者はSKKを使用しないのでよくわかりませんが、fcitx-skkは必要最低限の機能はそろっているようです[注19]。

ベースパッケージ

ベースとなるパッケージは、カーネルが3.16、X.Orgが1.16、gccが4.9などとなっています。ほぼ現時点での最新版です。Qt5も積極的に新バージョンを投入するようになっています。

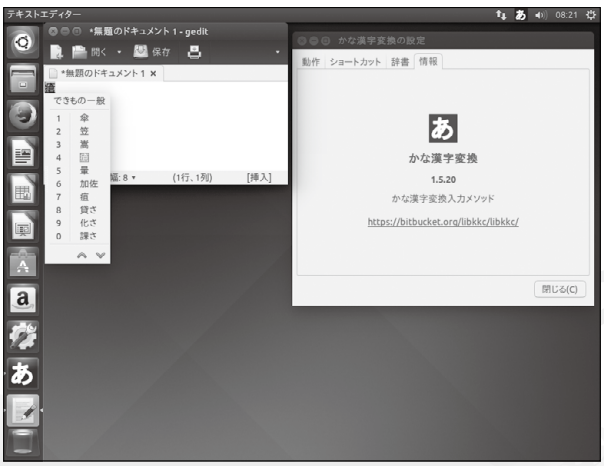
Ubuntu for phones

通常のUbuntuのリリースサイクルとは異なり、9月にUbuntu for phonesのRTMをリリースしました。RTM(Release for Manufacturing)をリリースしましたというのは重複表現ですが、Ubuntu for phonesのイメージが完成したので、それを各スマートフォンメーカーにリリースし、年内にはUbuntu for Phonesがインストールされたスマートフォンが発売予定となっています、というのが正確です。実際に中国のMeizuが12月にUbuntu for phonesをインストールしたスマートフォンを発売するという発表がありました[注20]。

もちろん現状日本でリリースされる予定はありませんし、そもそもほとんどはUbuntu Desktop Nextと同じですので、現状では前述のとおり日本語の入力はおろか日本語キーボードの選択すらできません。とはいえ、いずれそれらの問題は解決されることでしょう。

日本の大手キャリアからUbuntu for phonesがインストールされたスマートフォンが発売されることはないでしょうが、OSの入れ替えができるNexusシリーズなどにUbuntu for phonesをインストールし、MVNOのSIMで通信や通話ができるようになればいいと思います。SD

図8　ibus-kkcが動作しているところ。候補ウィンドウの上部に用法が表示されている



注17) 残念ながらFcitx用のブリッジはありません。
注18) 最も、癖はMozcにもありますが。
注19) バリバリのSKKユーザにはちょっと物足りないかもしれないという意味です。
注20) http://news.softpedia.com/news/Meizu-Confirms-MX4-Ubuntu-Touch-Release-for-December-2014-458782.shtml

第32回

Linux 3.17で追加される機能
file sealingとmfd_create

Text：青田 直大　AOTA Naohiro

はじめに

先月までLinux 3.15について紹介してきたので今月はLinux 3.16……はスキップして、開発中のLinux 3.17に入る機能について紹介していきます。Linux 3.16は、ある種のデバイスの対応などでおもしろいものはありますが、一般に紹介するにはこれといったものがない、この連載にとっては「難しい」カーネルだったのです……^^;

共有メモリの利点とその弱点

プログラムを書いていると、複数のプログラムに協調した動作をとらせたくなることがあります。こういうときにはIPC（Inter-process Communication）を用いて、プログラム間でメッセージのやりとりを行います。IPCにはたとえばパイプ、ソケット、シグナル、共有メモリといったものがあります。もちろんディスク上のファイルを経由するのも1つの手段と言えます。

共有メモリはさまざまなIPCの中でも、最も高速でかつ使いやすい方法です。ほかのIPC、たとえばパイプを使った方法では、readやwriteといったシステムコールを実行して、パイプとのデータのコピーを行う必要があります。すなわち、データのやりとりに関してシステムコール呼び出しのコストと、パイプとのデータコピーのコストがかかると言うことができます。一方で共有メモリでは、そういったシステムコールやデータのコピーの必要はありません（**図1**）。共有メモリはあたかも普通のメモリ領域かのように、システムコールを使わずにいつでも自由に読み書きできます。こういった意味で、共有メモリは高速かつ使いやすいと言うことができます。

しかし、共有メモリにも弱点があります。それは通信する相手が、互いを信用していなければ使用できないということです。たとえば、パイプの場合であれば、一度送信側のプロセスがパイプへの書き込みを行い、受信側のプロセスが読み取ってしまえば、送信側のプロセスは、受信側のプロセスにコピーされたデータをもう変更することはできません。

それに対して共有メモリの中のデータは、いつでも送信側が変更できます。そうすると、たとえば送られてきたデータを一度調べてチェックサムが合致していたからといって、それをいつまでも信用して使うことはできません。もちろん受信側が自分にしかアクセスできないメモリ領域にコピーしてしまえば、送信側の書き換えはできなくなりますが、同時に共有メモリの

データのコピーがいらないというメリットも失われてしまいます。

また、共有メモリ上のデータ自体の書き換えのほかにも、共有メモリ領域のサイズが変更されることがあるという問題もあります。データ送信側はいつでも領域のサイズを変更できます。サイズが小さくなったことによって、受信側が読み取ろうとした部分が共有メモリの範囲外に出てしまうと、シグナルSIGBUSが受信側のプロセスに送られます。受信側のプロセスが、このシグナルを適切に処理するシグナルハンドラを設定していなければ、そのプロセスはクラッシュしてしまいます。

このように共有メモリには高速で使いやすいという利点がありながら、相手が信頼できないとうまく用いることができないという弱点があります。Linux 3.17では、この問題を解決する新しいしくみである“file sealing”が実装されています。

シールを貼る

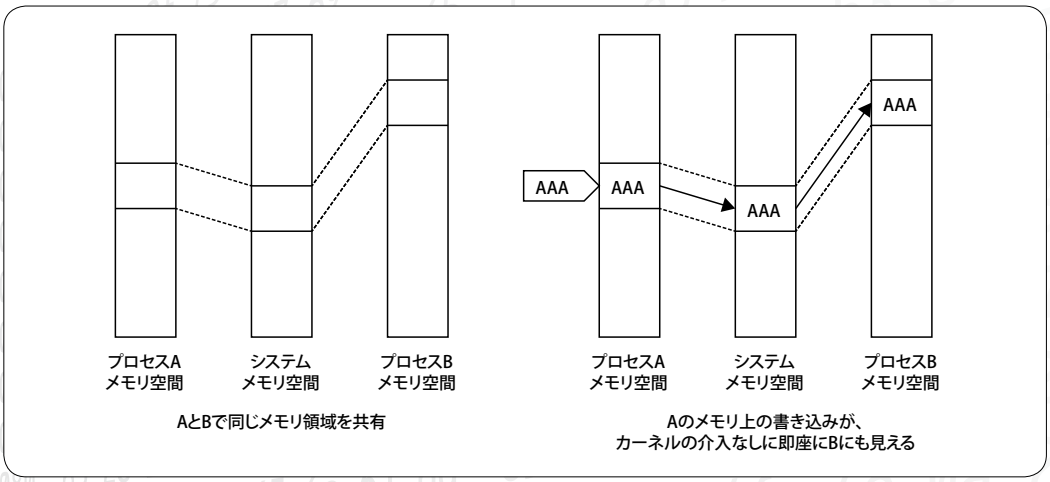
HDDなど包装された商品を買ってきたことを考えてみてください。包装の上にシールが貼ってあるかと思います。このシールを確認することで、商品が開封されていないことを保証できます。シールが貼っていない商品であれば、中身が信頼できないので返品することもできるでしょう。“file sealing”では、この考え方を共有メモリに適用しています。

まず、データ送信側は共有メモリにデータを書きます。データの準備ができたら書き込み可能なメモリマッピングを削除し、「これ以降書き込みしない」とのシールを貼って、受信側に通知します。これ以降は誰もその共有メモリを書き込み可能でマップできなくなります。すなわち、それ以上誰もその共有メモリに書き込みできなくなります。

データ受信側は、「これ以降書き込みしない」シールがあるかどうかを確認します。シールがあれば、もう書き変えられる心配はないので安心してデータの読み込みを開始できます。シールがなければ、そのデータは書き変えられる可能性があるので拒否すればいいわけです。

同様に「サイズを小さくしない」シールを用いることで、先ほどのSIGBUSの問題も解決できます。渡された共有メモリ領域がもう小さくなることがない、ということが保証されていれば、いつでもそのメモリ領域内のどこにでもアクセスできることが保証されるので、シグナルハンドラを設定する手間がなくなるというわけです。

▼図1　メモリ空間とメモリ領域の共有



シールの詳細と実例

file sealingではF_SEAL_WRITE、F_SEAL_SHRINK、F_SEAL_GROW、F_SEAL_SEALの4種類のシールが実装されています。F_SEAL_WRITEが貼られると、それ以上の書き込みができなくなります。しかし、サイズの変更はできます。F_SEAL_SHIRNKとF_SEAL_GROWはそれぞれサイズを小さくする、または大きくする操作を禁止します。最後のF_SEAL_SEALはシールを追加すること自体を禁止します。また、このシールは一度貼られると2度と剥がすことはできません。「シールが貼られた以上、書き込みやサイズ変更ができないようにする」ということを考えると当然ですね。

これらのシールを用いて、たとえば途中でデータが改ざんされることなく安全にデータを受け取りたければ、「F_SEAL_WRITE¦F_SEAL_GROW¦F_SEAL_SHRINK」が設定されていれば、データの上書きもサイズ変更もできないことを保証できます。

あるいはXやWallandなどのグラフィックサーバのことを考えてみましょう。クライアントは共有メモリ上に描画したいデータを書いて、サーバに渡します。そのときにサーバがF_SEAL_SHRINKが付いていることを確認することでSIGBUSの心配なく共有メモリ領域にアクセスできます。ここで「サイズを大きくできなくする」シールが、SHRINKとは別に用意されていることに注意しましょう。もし描画バッファが足りていないときには、クライアントは自由にバッファサイズを大きくできます。

file sealingを使ったコード例

ではfile sealingの実例を見てみましょう。簡単な文字列の通信を行う2つのプログラムを考えます。

共通のヘッダがcommon.h(**リスト1**)で、send.c(**リスト2**)は送信側、recv.c(**リスト3**)は受信側のプログラムです。送信側は“msg = ”で始まる文字列を送信し、受信側は受け取った文字列が“msg= ”で始まるかを確認したあと、2秒後に受信文字列を表示します。

受信側は、“test-socket”という名前のUNIXドメインソケットを開いて接続を待機し、送信側は文字列を共有メモリにデータを書くと、

▼リスト1　共通ヘッダ common.h

```
/* linux/memfd.h */
#define MFD_ALLOW_SEALING       0x0002U

/* linux/fcntl.h */
#include <linux/fcntl.h>

/*
 * Set/Get seals
 */
#define F_ADD_SEALS     (F_LINUX_SPECIFIC_BASE + 9)
#define F_GET_SEALS     (F_LINUX_SPECIFIC_BASE + 10)

/*
 * Types of seals
 */
#define F_SEAL_SEAL     0x0001  /* prevent further seals from being set */
#define F_SEAL_SHRINK   0x0002  /* prevent file from shrinking */
#define F_SEAL_GROW     0x0004  /* prevent file from growing */
#define F_SEAL_WRITE    0x0008  /* prevent writes */
/* (1U << 31) is reserved for signed error codes */

#define SOCK_NAME "test-socket"
```

▼リスト2　送信側プログラム send.c

```
#include <sys/mman.h>
#include <sys/socket.h>
#include <sys/stat.h>
#include <sys/un.h>
#include <errno.h>
#include <fcntl.h>
#include <string.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include "common.h"

#define SHM_NAME "msg"

void errexit(char *msg)
{
  perror(msg);
  exit(1);
}

void sendfd(int fd)
{
  【省略】
}

void sendstr(char *msg)
{
  int len = strlen(msg) + 1;
  int r, fd;
  if ((fd = shm_open(SHM_NAME, O_RDWR | O_CREAT, 0600)) < 0)
    errexit("shm_open");
  if ((r = shm_unlink(SHM_NAME)) < 0)
    errexit("shm_unlink");
  if ((r = ftruncate(fd, len) < 0))
    errexit("ftruncate");
  char *buf;
  if ((buf = mmap(NULL, len, PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0)) == MAP_FAILED)
    errexit("mmap");
  strcpy(buf, msg);
  sendfd(fd);
  if (munmap(buf, len) < 0)
    errexit("munmap");
  close(fd);
}

void sendstr_boo(char *msg)
{
  ⋮
  sendfd(fd);
  sleep(1);
  strcpy(buf, "BOO");  /* 共有メモリ上のデータを不正なものに書きかえる! */
  if (munmap(buf, len) < 0)
  ⋮
}

#define SYS_memfd_create 319
#define mfd_create(name, flags) syscall(SYS_memfd_create, name, flags)
```

下からの続き

```
void sendstr_seal(char *msg)
{
  ⋮
  /* シールを付けられる共有メモリを作る */
  if ((fd = mfd_create(SHM_NAME, MFD_ALLOW_SEALING)) < 0)
    errexit("mfd_create");
  if ((r = ftruncate(fd, len) < 0))
  ⋮
  /* munmap()しないとシールを付けられない */
  if (munmap(buf, len) < 0)
    errexit("munmap");
  /* シールを追加 */
  if (fcntl(fd, F_ADD_SEALS, (F_SEAL_SHRINK | F_SEAL_GROW |
F_SEAL_WRITE)) < 0)
    errexit("fcntl");
  sendfd(fd);
  ⋮
}

int main()
{
  sendstr("msg = Hello!");
  return 0;
}
```

本ページ右上部に続く ↗

▼リスト3　受信側プログラム recv.c

```
#include <sys/mman.h>
#include <sys/socket.h>
#include <sys/stat.h>
#include <sys/un.h>
#include <errno.h>
#include <fcntl.h>
#include <string.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include "common.h"

#define LISTEN_BACKLOG 50

void errexit(char *msg)
{
  perror(msg);
  exit(1);
}

int recvfd(int sock)
{
  【省略】
}

void recvstr(int sock)
{
  int fd = recvfd(sock);
  struct stat s;
  if (fstat(fd, &s) < 0)
    errexit("fstat");
  int len = s.st_size;
  char *buf;
  if ((buf = mmap(NULL, len, PROT_READ, MAP_PRIVATE, fd, 0)) == MAP_FAILED)
    errexit("mmap");
  if (strncmp(buf, "msg = ", 6) == 0) {
    sleep(2);
    printf("%s¥n", buf);
  } else {
    printf("invalid message: %s¥n", buf);
  }
  if (munmap(buf, len) < 0)
    errexit("munmap");
  close(fd);
}

void recvstr_seal(int sock)
{
  int fd = recvfd(sock);
  int seals = fcntl(fd, F_GET_SEALS); /* シールを取得 */
  if (seals == -1)
    errexit("fcntl");
  /* 書きかえ可能であれば拒否する */
  if (!(seals & (F_SEAL_SHRINK | F_SEAL_GROW | F_SEAL_WRITE))) {
    printf("NOT SEALED!¥n");
    close(fd);
    return;
  }
  ⋮
}
```

本ページ右上部に続く ↗

下からの続き

```
int main()
{
  int sock;
  struct sockaddr_un addr = {.sun_family = AF_UNIX};
  strcpy(addr.sun_path, SOCK_NAME);

  sock  = socket(AF_UNIX, SOCK_STREAM, 0);
  if (sock < 0)
    errexit("socket");
  if (bind(sock, (struct sockaddr*)&addr, sizeof(addr)) < 0)
    errexit("bind");
  if (listen(sock, LISTEN_BACKLOG) < 0)
    errexit("listen");

  int sockrecv;
  while((sockrecv = accept(sock, NULL, NULL)) != -1) {
    recvstr(sockrecv);
  }
  return 0;
}
```

UNIXドメインソケットを用いて、共有メモリのファイルデスクリプタを受信側に渡します。受信側はそこから共有メモリのデータを自分のメモリ空間にmmapするという流れです。このリストでは省略していますが、sendfdとrecvfdとがUNIXドメインソケットを用いたファイルデスクリプタ受け渡しの処理を行っています。

元のsendstr()では“msg = ”で始まらないようなメッセージを受信側に表示させることはできません。しかし、共有メモリですから、1秒sleepしてから“BOO”と書いてやると、受信側の“msg = ”のチェックを抜けたあとにメッセージを書き換えることができます(sendstr_boo()省略部分はsendstr()と同じ)。

これではいけないので、受信側でsealの確認をすることにしましょう。recvstr_seal()ではrecvfdで受け取った共有メモリのファイルデスクリプタに適切なシールが貼られているかを、fcntl(F_GET_SEALS)でシールの取得をして確認しています。これで受信側は共有メモリ上のデータが書き換えられることはないと確信して作業を進めることができます。

ただし、送信側のコードも変更しなければいけません。適切なシールが貼っていないと拒否されるので、シールを貼るようにコードを変更します。シールを貼る前に1つポイントがあります。それは共有メモリをshm_open()やopen()ではなく、mfd_create()を用いて開くことです。

mfd_create()はこれ自体もLinux 3.17で追加される新しいシステムコールです。第1引数には共有メモリの名前(とくに意味はなくデバッグ用途)、第2引数にはフラグを設定します。このmfd_create()にMFD_ALLOW_SEALINGフラグをつけて呼ぶことで、開かれたファイルだけにシールを追加できます。これは悪意のあるプログラムが、無関係のファイルにシールを貼ってくるのを防ぐためです。もしもどんなファイルにもデフォルトでシールを貼れるようにしていると、悪意のあるプログラムが/dev/shm下のファイルをスキャンして、ひたすらシールを貼っていき、ほかのプログラムが書き込みを行おうとするのを妨害してくるかもしれない、というわけです。

mfd_create()で共有メモリを作りデータを書き込み終わると、まずmunmap()を行います。もしmunmap()を行わずにシールを貼ろうとすると、Device or resource busyのエラーが出て失敗します。シールを貼るともうそれ以降、書き込み可能でmmapすることはできません。試してみてもOperation not permittedのエラーで失敗します。もちろんmmapせずに使っていたデータを書こうとしてもsegmentation faultが起きてしまうだけです。

mfd_create

実はmfd_create()はもう1つ、Android開発者にうれしい特徴も持っています。共有メモリ上にファイルを開くにはshm_open()またはopen()を使いますが、このどちらもtmpfsという、メモリ上にデータを保存するファイルシステムがmountされていないと使うことができないのです。ところが、Androidではtmpfsをmountしていないシステムもあり、これらの関数を使うことができません。そんな環境でも、file sealingを用いたIPCができるようにするというのもmfd_create()が追加された理由となっています。

まとめ

今回はLinux 3.17で追加される機能のうち、共有メモリを用いたIPCをいろいろな状況で使えるようにするfile sealingと、それに関連したシステムコールであるmfd_createについて解説しました。SD

November 2014

NO.37

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
榎 真治　ENOKI Shinji　shinji.enoki@gmail.com

JUNETを創った人たち／オープンデータを作る人たち

今回は7月に東京で行った勉強会と、8月に京都で行った研究会の模様をお届けします。

jus定期総会併設勉強会

■日本のインターネットを創った人達が語るインターネットのこと ～JUNET設立30周年記念～
【講師】砂原 秀樹（慶應義塾大学）、
齊藤 明紀（鳥取環境大学）、
坂下 秀（㈱アクタスソフトウェア）
【司会】法林 浩之（日本UNIXユーザ会）
【日時】2014年7月19日（土）16:30～18:30
【会場】ハロー貸会議室 秋葉原 II

jusでは毎年7月に定期総会と併設行事を行っています。今年はJUNETの設立から30周年を迎えることにちなみ、当時を知る方々を招いて話をうかがう勉強会を企画しました。参加者は27人でした。

前半は齊藤さんと坂下さんが登壇しました。当時は10BASE5のケーブルを自分たちで敷設し、トランシーバ経由でUNIXマシンを接続し、さらにシリアルケーブルで多数の端末を接続したそうです。UNIXマシン間の通信はまだIPではなくUUCPでした。よく使われたUNIXマシンはSunやVAXですが、System V系UNIXでの接続には苦労したと言います。組織間は電話回線とモデムで接続していましたが、多数の接続相手を抱える組織では3.4kHz音声専用線を利用していました。日本語を扱えるようにするために、文字コードのISO-2022-JP、X用の日本語フォントである橘フォント、日本語入力システムWnnなどが作られました。トップレベルドメイン名は.jpではなく.junetで、管理はjunet-adminというグループが行っていました。当時のJUNETの代表的な用途はメールです。しかし、そのころはDNSがまだなかったので、すべて静的ルールで配送していたそうです。

休憩をはさんで後半は砂原さんも加わりました。JUNET構築のきっかけは、東工大にいる村井純さんに慶応大からテープを届けるのがたいへんなのでモデムで接続したことでした。ネットワークと呼ぶには3拠点が必要で、研究として認めてもらうために東大を接続してJUNETがスタートしました。JUNETに接続したい組織が多かったので、受け入れ可能な組織とのお見合いをjusで企画したこともあります。こうしてJUNETは最盛期で約700組織が参加するまでに成長しました。その後、トップレベルドメインを.junetから.jpに移行し、UUCPからIPへの移行が進みました。そしてjunet-adminでやっていた業務をJNIC（現在のJPNIC）に移管し、IIJやSPINなどが立ち上げたISPへの移行を促すべく1994年にJUNET協会を解散しました。

また、JUNETという名前は石田晴久さんが大学間ネットワークの名称として考えたのがお蔵入りになっていたのを、村井さんがネットワークを作るときに提供したという話がありました。

最後に講師からのメッセージとして、坂下さんは「P2Pが実用的になるまでは頑張りたい」、齊藤さんは「何もないところから作るのが低レイヤの楽しみ。機会があれば挑戦してほしい」、砂原さんは「ヴィント・サーフさん[注1]からの宿題に答えるべく我々は考え

ていかないといけない」という言葉を残されました。

今回のような歴史的な話題を扱うセッションを実施するたびに思うことですが、あまり知られていない話がまだたくさんあるらしく、毎回新たな事実が紹介されるのは驚きです。それを少しでも記録に残すことが大切ですので、また企画したいと思います。

jus研究会京都大会

■みんなで作ろうオープンデータ in 京都
【講師】青木 和人（オープンデータ京都実践会）、是住 久美子（ししょまろはん）
【司会】榎 真治（日本UNIXユーザ会）
【日時】2014年8月2日（土）14:00～14:45
【会場】京都リサーチパーク 1号館4F 会議室B

講師の青木さん、是住さんと筆者（榎）はオープンデータ京都実践会（以下、実践会）で一緒に活動しています。オープンデータを作っていくのは楽しい活動ですので、京都の方たちにもっと知っていただきたいと思い企画しました。研究会はオープンソースカンファレンス2014 Kansai@Kyoto内で行い、参加者は32名でした。

青木さんは、オープンデータの概要と実践会の活動について紹介されました。オープンデータを活用するためには、データが構造化され機械で判読できる形式で公開されていることが大事です。しかし、行政の持っているデータ形式からの変換作業を誰が行うのか、また、行政が情報を出したあとのメンテナンスをどうするかといった課題があるそうです。

実践会は、地域の方に歴史や文化を教えてもらいながら、街歩きをしてみんなでオープンデータを作る活動をしています。具体的にはWikipediaタウンとOpenStreetMap（以下、OSM）のマッピングパーティの2つをセットで行っています。Wikipediaタウンは地域で連携して街のいろいろな情報をまるごとWikipediaに書く活動です。OSMは自由に利用できる地図を作るプロジェクトで、人が集まって調べてその情報を地図に追加するのがマッピングパーティです。2月22日のオープンデータデイやそのプレイベントで、この2つをセットにした「まち歩きデータソン」に取り組みました。40名以上もの参加があり、楽しく盛り上がりました。青木さんとしては、行政がデータを出すのを待つのではなく、自分たちでオープンデータを作っていけばいいではないか、という思いで取り組まれているそうです。

是住さんからは、京都府立図書館で働く図書館司書の職場内学習グループ「ししょまろはん」の活動について紹介がありました。ししょまろはんでは、京都が舞台として出てくる本のデータをオープンデータとして公開しています。書籍に出てくる場所から1ヵ所を選んで、緯度経度の位置情報やおススメ度、内容紹介を載せたもので、現在は小説40件、ライトノベル6件、マンガ43件など全部で162件の情報があります。「春の山村美沙祭り」と題して集中的に取り組んだため山村美紗作品が53件もあります。ほかにも『ゴルゴ13』や『美味しんぼ』なども作中で京都に来ているはずだと探したそうです。これらのデータはクリエイティブ・コモンズライセンス（CC-BY）で公開されており、スマートフォンアプリの「ご当地なび」でもコンテンツが使われています。

また、全国各地で図書館員が調べ物のお手伝い（リファレンスサービス）をしたデータを国立国会図書館が集めて公開している「リファレンスサービス共同データベース」があります。その中から京都に関するトピックを選んで、ししょまろはんで調べた位置情報などを付加したものを「京都レファレンスマップ」として公開しています。独自で付加した部分をオープンデータにしています。Wikipediaの記事編集では出典となる情報が大切ですが、出典となる書籍を探すときにも京都レファレンスマップは参考にできるのではと思いました。

オープンデータというと行政データの公開という観点から語られることが多いですが、お二人の話からは自分たちの手でオープンデータを作ることのおもしろさ、楽しさが伝わってきました。SD

注1）アメリカ合衆国の計算機科学者。インターネットとTCP/IPプロトコルの設計にかかわった「インターネットの父」の1人。

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
佐伯 幸治 Koji Saeki
Twitter @widesilverz

第35回 Hack For Japan気象データ勉強会 ◆第1回目 基礎編

“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4月に開催された「気象データ勉強会」の模様をお届けします。

はじめに

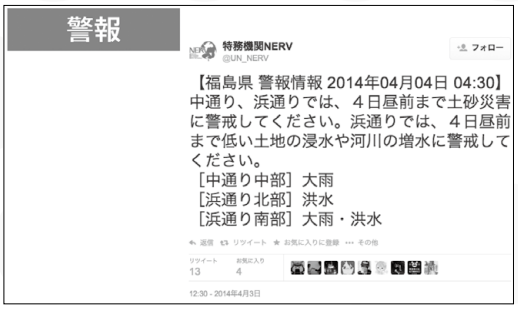
4月のことになりますが、Hack For Japanによる「気象データ勉強会」が開催されました[注1]。この勉強会は、自然災害が起こった際にITを利用して気象データを正しく読み取る力を身に付けることを目的としています。講師には、ゲヒルン㈱代表取締役の石森大貴さんを迎えました。石森さんは気象庁が提供するXMLの電文をもとにして災害情報を発信するTwitterアカウント「@UN_NERV[注2]」を運用され、気象データの扱いについて高い見識をお持ちです。

今回は、この勉強会で石森さんが語られた内容を再構成してお届けします（なお本稿にて事実関係などの誤りがある場合は、著者にその責任がありますのでご了承ください）。石森さんの講義では、「防災気象情報」、「緊急地震速報」、「災害と情報」、「気象庁が発表する電文・表現の意味や解釈について」、「気象情報を発信するときの注意」といった内容が語られましたが、ここでは「防災気象情報」と「緊急地震速報」を中心にお届けします。当日は石森さんが作成した気象庁のXMLの電文が見られるビューワを使いながら講義が進んでいきました。

注1　当日の模様はYouTubeに公開しています。詳しく知りたい方はぜひご覧ください。
・Togetterまとめ URL http://togetter.com/li/656335
本文で紹介している石森さんの講義はpart-2の18分30秒くらいからです。
・気象データ勉強会 URL http://www.youtube.com/watch?v=owYZaMRjmnQ
・気象データ勉強会part-2 URL http://www.youtube.com/watch?v=BFRFxZEsXzQ
また、石森さんが当日使用したスライドは次のURLからダウンロードできます。より理解が深まりますのでぜひ活用ください。 URL http://isidai.kvs.gehirn.jp/public/hackforjapan_meteorological_1.pdf

注2　https://twitter.com/UN_NERV

◆図1　警報のツイート



1 防災気象情報

気象特別警報・警報・注意報

まずは気象特別警報・警報・注意報についてです。注意報には雷注意報、雪崩注意報、強風注意報などがあります。これが警報になると雷や雪崩がなくなって大雨・洪水・暴風・暴風雪・大雪・波浪・高潮になります（**図1**）。表現が強風から暴風、風雪から暴風雪に変わり、注意報にしかない表現があります。たとえば雪崩は警報になりません。特別警報になると大雨は浸水害と土砂災害の2つに分かれます。洪水はなくなりますが、洪水については指定河川洪水予報として別に発表されるためです。

特別警報は数十年に1度の現象に相当する指標です。大雨特別警報では、数十年に1度の降雨量のほかにも48時間降水量や土壌雨量指数などの条件があり、それが「府県程度の広がり」という表現になって、県域全体に影響するような場合に発表されます。ですので広範囲で同時多発的に災害が起きそうなときに特別警報が発表されます。

この条件のため、大島の大雨による土砂災害[注3]

注3　2013年10月16日に土石流が発生。

では、東京都全域にかかるものでなく、局地的に大島だけだったので特別警報は出ませんでした。反対に、京都・福井・滋賀県では全域が危ないと判断されて発表されたのですが[注4]、雨が降っていないような地域も含まれていました。

大雪特別警報では、「府県程度の広がりをもって50年に1度、かつ丸1日以上続く場合」となっており、山梨や群馬で起きた雪害[注5]では1日内で雪が終わったため、特別警報に値しませんでした。

府県・地方・全般気象情報

府県・地方・全般気象情報というものがあり、これが事前防災に役立ちます。府県気象情報というのは、東京都や宮城県など県単位で出るものです。地方気象情報になると関東甲信地方、東北地方など大きな範囲で防災について注意を呼びかけます。全般気象では西日本、東日本が中心となります。@UN_NERVでは以前は府県気象情報を出していましたが、それではツイートし過ぎるので地方気象情報に変えています。

土砂災害警戒情報

土砂災害警戒情報は、都道府県と気象庁が共同で発表する防災情報です。避難勧告を出すべきかどうか、住民が自主的に避難すべきかどうかといった判断材料になる情報です。

気象庁のサイトで解説を読んでみると、「土砂災害警戒情報は、避難勧告等の災害応急対応が必要な土石流や集中的に発生する急傾斜地崩壊が対象」となっています。逆に土砂災害警戒区域に指定された所はハザードマップを作成して住民に周知する必要があり、雨量などを計算して危険度を算出します。この情報は気象庁のWebサイトにPDFで公開されています。なぜかこれだけPDFで非常にパースしづらかったため、XMLに変換しています。

記録的短時間大雨情報

記録雨、時雨と言われます。これは数年に1度しか発生しないような短時間の大雨、記録的な短時間に降った大雨情報です。本当に局地的に降っているので、ゲリラ豪雨や局地的な気象特別警報が出せない情報は、だいたい記録雨がカバーしています。

記録雨が来ていて、何回もこの情報が出ていたら本格的に危険という状況です。気象庁のWebサイトには、「災害の発生につながるような希にしか観測しない雨量であることを知らせるために発表」と出ています。これを見たら心配してください。

指定河川洪水予報

洪水の特別警報がないのはこれがカバーしているためです。どんな情報が来るかというと、はん濫危険水位を超える水位です[注6]。指定河川洪水予報にはレベル分けがあって、はん濫注意情報、はん濫警戒情報、はん濫危険情報、はん濫発生情報と段階的に情報が出てきます。XMLの中には観測所ごとにこれらの情報がバラバラに入っていますので、一番危険な情報を取り出すか、全部取り出すか、自分でポリシーを決める必要があります。

はん濫注意情報は、はん濫注意水位に到達し、さらに水位の上昇が見込まれるので、住民ははん濫に関する情報に注意すべきという情報です。避難に時間がかかりそうなお年寄りや幼児が避難するかどうかを判断するよう、避難準備情報で促します。

はん濫警戒情報では、避難判断水位に到達して、さらに水位が上昇すると避難勧告を発表するレベルとなり住民は避難します。

はん濫危険情報では、はん濫危険水位に到達して猶予がないので、住民は避難を完了しているべきであるという情報です。

はん濫発生情報は川が溢れていますという情報なので、次にほかの観測所などで溢れる心配がないか

注4　2013年9月16日5時5分に京都府・福井県・滋賀県に特別警報が発表。なおNHK NEWS WEBでは2014年9月3日に「特別警報 運用1年 課題も」という記事を公開。特別警報のあり方について報道しています。 URL http://www3.nhk.or.jp/news/web_tokushu/2014_0903_02.html

注5　2014年2月に発生した豪雪。

注6　「河川の増水やはん濫などに対する水防活動の判断や住民の避難行動の参考となるように、気象庁が国土交通省または都道府県の機関と共同して、あらかじめ指定した河川について、区間を決めて水位または流量を示した洪水の予報」のこと（気象庁のサイトより）

を急いで検討する必要があります。

なお、はん濫注意情報は洪水注意報に相当していて、ほかのものは洪水警報に相当しています。

竜巻注意情報

竜巻が近づいてくると雷が鳴って、雹(ひょう)が降ってきて、びっくりするくらい雨が降ります。NHKなどでは、竜巻と断定されるまでは「竜巻のような突風が吹いた」という表現になっています。これは都道府県単位で出てくるのが一般的です。本当は局地的に発生していますが県全域が含まれます。また、1時間の有効期限がある情報です。1時間を過ぎてもまだ危険となると、もう1回同じように出し、電文が飛んでこなかったら解除されたと判断します。

予測には竜巻発生確度ナウキャストを使っています。これは確度なし、確度1、確度2と3つの段階で判断します。確度1は的中率が1～5%、確度2は的中率が5～10%です。なんだ10%か、と思われがちですが、気象庁によると「竜巻などの激しい突風は人が一生のうちほとんど経験しない希な現象であり、確度1でも確度2でも、普段と比べると遭遇の可能性が格段に高い状況」だということです。

確度2のときは本当に危険な状況で、竜巻注意情報というのは確度2に当てはまったときに発表します。数日前から府県気象情報などで大気の状態が不安定と予想され、いざ不安定になると雷注意報が出ます。それより天気が悪くなってナウキャストが確度2を示すと、この情報が発表されます。この情報が出ていると、その県のどこかは天気が本当に悪いと判断します。今にも竜巻が起こりそうだ、もしくは発生しているという気象状況です。

2 緊急地震速報

地震動

緊急地震速報と震度速報、震源に関する情報、震源震度に関する情報、各地の震度に関する情報というのが出ており、すごいのが緊急地震速報です。揺れたら数秒後、数十秒後に情報が届きます。実際に震度計で観測した1分半後に集計してきます。約3分後に津波があるかどうかを判断します。津波に関する情報が発表されたときは、震源に関する情報は出ません。震度1以上を観測すると「各地の震度に関する情報」が出てきます。震度速報は地域を188に分けて出てきます。

やっているのは@UN_NERVとNHKだけだと思いますが、188の地域名を手動で変えています。電文では広島県北部、福島県会津などと出ますが、それを県や地方を削ってなるべく多くの情報がTwitterで出せるようにしています。震度速報の図は震度マップを生成するシステムで、これはオリジナルで作りました。

震源に関する情報

これは津波警報や注意報が発表されたときには出ませんが、注意報の下の津波予報という区分では発表されます。「若干の海面変動があるかもしれないが被害の心配はない」といった表現となります。

震度と震源の情報が電文では別々に提供されますが、@UN_NERVではこれを組み合わせて一緒に出しています。どうやって組み合わせるかというと、ヘッダの中にイベントIDという電文——僕らは地震IDと呼んでいます——があるのですが、そのIDで何の地震か、どの地震かがわかるので、それを利用しています。緊急地震速報も地震IDが飛んでくるのですが、緊急地震速報と震度速報と震源に関する情報とで、同じ地震に関する情報でも地震IDが違うときがたまにあり、この点は注意が必要です。これは地震発生時刻をもとにしているので、観測の仕方によって震源場所がずれてくると、発生時刻もずれてくるためだと考えられます。

震源震度に関する情報

震度3以上、津波警報または注意報発表時に若干の海面変動が予想される場合、緊急地震速報(警報)を発表した際の情報となります。各地の震度に関する情報が別にあって、XMLの場合には、「震源震度に関する情報」が「各地の震度に関する情報」も一緒に入ってきます。なお地震の規模という言い方、こ

れは気象庁マグニチュードというもので、マグニチュードは飽和という性質をもち、マグニチュード8を超えると飽和により正確な値がわからなくなります。そういうとき、津波情報などでは「マグニチュード8を超える巨大地震」という表現で詳しく分析が終わるまでは置いておきます。

津波情報

地震発生から最速2分くらいで津波情報が出てきます。これは緊急地震速報の賜物です。緊急地震速報で規模、場所、深さをおおよそ推定できることから、こんなに早く津波情報が出せるようになりました。津波注意報と警報と特別警報の3種類があります。津波注意報は“海岸から離れてください”というレベルで、津波警報や大津波警報になると人命が心配されます。

@UN_NERVはこの情報を気象庁から受け取って、震度マップと同じようなエンジンで津波に関する情報を自動的に生成してTwitterで流した経験があります。@UN_NERVでは予想される津波の高さは出さないことにしています。本当に大きいときは“巨大”などといった高さの表現をしています。それは今わかっている高さでは実際よりも低く見積もっている可能性があり、数値をあてにすることで避難が遅れてしまうことのないようにするためです。しかしながらNHKが速報フラグを立てて情報を出すと、@UN_NERVでもツイートされるため高さが出てしまう場合もあります。

緊急地震速報

予報と警報の2種類があります。予報は高度利用者向け緊急地震速報というもので受信できます。警報(特別警報も含む)は一般向け緊急地震速報で、携帯・テレビ・ラジオで受信できます。基準は震度5弱以上が予想されると警報という扱いになります。震度5弱を震度5だと理解している方がいると思われますが、1,000円弱というと998円となるように、計測震度において4.5以上5.0未満が、震度階級[注7]での震度5弱と換算されます。

@UN_NERVでは警報のときは強い揺れが予想される地域を出し、予報のときにはちょっと細かい予報でマグニチュードや最大震度が出てきます。

3 災害と情報

災害が起こる前には、実はいろいろな情報が流れています。地震や津波はほとんど間に合いませんが、大雨・竜巻などは事前に情報が出ます。気象情報だと2日前などに、注意報・警報、河川があふれそうだといった情報が5段階のレベルで出てきますし、土砂災害警戒情報が出て、記録雨が出て、特別警報が出た後は、情報がほとんどないので、ぼーっとしていたら逃げ遅れます。事前に住民は自分で判断して、自治体もこれらの情報を鑑みて避難準備情報くらいは出すべきです。

住民も避難と聞いたら避難所に行きがちなのですが、避難所が潰れるパターンもあるので、避難と聞いたら「最も自分が安全な場所に身を移す」という行動だと思ってほしいです。

最後に

当日は石森さんのわかりやすい説明に加え、「エンジニアとしては放送やインターネットを通じて、迅速かつ的確にこの情報を伝える使命がある」と強く語っていたのが印象的でした。最後に石森さんを含め、登壇いただいた気象庁の杉山善昭さん[注8]、長田泰典さん[注9]、先進IT活用推進コンソーシアムの菅井康之さん[注10]、Yahoo! Japan防災速報チームの杉本康裕さん、東北放送の吉田信也さん[注11]、会場を貸していただいたさくらインターネットさん、参加いただきました皆様に感謝いたします。SD

注7　震度階級と計測震度について URL http://www.data.jma.go.jp/svd/eqev/data/kyoshin/kaisetsu/calc_sindo.htm

注8　杉山さんのスライド URL http://isidai.kvs.gehirn.jp/public/sugiyama_20140417Hack4JP.pdf

注9　長田さんのスライド URL http://isidai.kvs.gehirn.jp/public/nagata_20140417.pdf

注10　菅井さんのスライド URL http://www.slideshare.net/yasuyukisugai/hack-for-japan-33638000

注11　吉田さんの資料まとめ URL https://www.facebook.com/groups/hack4jp/permalink/625223720894413/

温故知新
IT むかしばなし

第38回

草の根BBSの運営

宮原 徹 MIYAHARA Toru　Twitter：@tmiyahar

はじめに

今回は20年以上前に、筆者が運営していた「パソコン通信」の“草の根BBS[注1]”についてお話します。

パソコン通信の時代

インターネットが商用化されてインターネットブームが巻き起こったのは、Windows 95がリリースされた、1990年代中ごろのことですが、それ以前は、電話回線を通じてモデム経由でホスト局と接続する形式の「パソコン通信」と呼ばれる独自の通信ネットワークが、おもにパソコンに詳しいパワーユーザを中心に形成されていました。

かくいう筆者もパソコン通信を始めるべく、大学3年生だった1992年の大晦日にアルバイトで稼いだお金を握りしめて、秋葉原にモデムを買いに行ったのを覚えています。当時はAIWAやオムロンといったメーカーがモデムを製造販売していましたが、筆者はオムロン製の2,400bpsのモデムを購入しました。

注1） BBSとはBulletin Board Systemの略で、電子掲示板システムのことです。

商用BBSへの参加

モデムを購入すると、箱の中には各種商用BBSの案内が入っていて、それに従うことによってオンラインサインアップができるのですが、筆者は富士通系のNIFTY-Serveに登録しました。当時はほかに、NEC系のPC-VANや、アスキーネットPCS（Public Communication Service）などがありました。

NIFTY-Serveには「フォーラム」と呼ばれる話題別のコミュニティがあり、掲示板などでテキストベースでやりとりをしていました。20年近く経ってBBSがインターネット上のSNSに変わっても、やっていることはあまり変わっていないようですね。

草の根BBS

商用BBSが全国にアクセスポイントを張り巡らせている全国ネットワークであるのに対して、“草の根BBS”は地域密着型のパソコン通信です。商用BBSでは飽き足らない人が、自前で電話回線やホストマシンなどを用意して、おもに地元の人向けに独自の掲示板などを用意、運営するわけです。

当時は電話回線を使ってダイアルアップ接続[注2]していましたから、「通話料金の関係で自然と地元の人しか集まらない」という傾向がありました。ちなみに東京03区域には草の根BBSも多かった[注3]ので、そこに接続するのは憧れでした。しかし、筆者が住んでいた神奈川県平塚市から接続すると通話料金が45秒間で10円と非常に高くなってしまい、東京03区域の草の根BBSに接続するということは、貧乏学生の筆者には高嶺の花でした。

草の根BBSに参加する

当時は電波新聞社から草の根BBSの電話番号が掲載された“電話帳”が出ていたので、近くにある草の根BBSを見つけて参加していました。

とくに活発だったのが、お隣の神奈川県厚木市にある「WADO-NET」という草の根BBSです。シスオペ（BBSの管理人のこと）の

注2） アクセスポイントと呼ばれる公開された電話番号に、モデムから電話をかけ、接続する方式のこと。

注3） 03区域内なら市内通話料金で済み、3分間10円で安かったのです。そのため草の根BBSは03区域内に集まっていました。

WARDONA>GKこと石川さんが大人だったこともあり、掲示板でのやりとりだけでなく、オフ会なども楽しいBBSでした。

草の根BBSを始める

WADO-NET中心で草の根BBS活動をしていましたが、何せ隣接地域ですので市内通話よりも料金はかさみます。そこで「地元0463区域で活発な草の根BBSを作りたい」と考えて、自分がシスオペになって草の根BBSを始めることにしました。

当時使っていたPC-9801互換機の「EPSON PC-286VG」をホストマシンにし、アルバイトで貯めたお金で電話加入権を購入しました。当時、電話加入権は7万円近かったので、大学生としては大きな買い物でした。BBSのホスト用ソフトウェアには、当時、草の根BBS用として人気のあった「KTBBS」を採用しました。

開業した自分の草の根BBS「Network AXIA」は、WADO-NETの常連さんや、地元で商用BBSを使っていた人などを中心にそこそこ利用者も増え、当時筆者も使っていた「シャープ X68000」ユーザが集まるようになりました。

高速モデムの導入と夜間2回線運用

当初は使い回しの2,400bpsのモデムを使っていましたが、受信した文章を目で追えてしまうぐらいの遅さでした。当時ちょうど9,600bps(通称くんろく)のモデムが出始めで、草の根BBSを運営しているシスオペ向けに沖電気の9,600bpsモデムを特価で販売するキャンペーンが行われて、筆者もそれを購入しました。また、モデムを接続するRS-232Cシリアルポートを増設し、自宅用の電話回線を夜間だけ接続して夜間2回線運用を開始しました。回線は手動切り替えなので、夜になると手動で接続し、朝になると切り離しという作業を毎日行っていました。いわゆる手動ネットワークオペレーターをやっていたわけで、我ながら酔狂なことをやっていたと思います。

オフ会だらけの週末

夜間2回線、さらにホストマシンのローカルコンソールと、クロス接続のRS-232Cシリアルケーブルで X68000を接続して、同時に4人まで接続できるようになると、チャットで会話をしたりすることも多くなります。当然、週末土曜日の夜は皆暇ですからどんどんホストに接続してくるので、その場でチャットで呼び出し、筆者の部屋でオフ会[注4]となることもしょっちゅうでした。

皆、20代の貧乏人ばかりですから、安い焼酎をコーラ割りで飲んで馬鹿話をしたり、ゲームで対戦したりと、最近の若者とあまり変わらないことをしていた、そんな青春時代でした。当然、女っ気はいっさいないのは、今のIT業界全般と一緒ですね。

Turbo Pascalと夜間メンテ

草の根BBS運営用ソフトに選んだKTBBSは、ソースコードが公開されていました[注5]。今で言うオープンソースです。言語はPascalで書かれており、当時、ボーランド社が販売していたTurbo Pascalでコンパイルしていました。

KTBBSはなかなかいいソフトでしたが、一部の操作コマンドがわかりにくいのが難点でした。たとえば、終了して回線を切断するのがGood byeの「G」(なんだそりゃ!)になっているのを、一般的なQuitの「Q」に直したりと、細かい手直しをして使っていました。

たまにバージョンアップするときには、皆が寝てしまう深夜2時ぐらいからホストを止めて、ソースコードを修正。そしてコンパイルしていると朝の6時ぐらいになってしまい、夜が明けてきます。思い起こせば、当時から業務システムの運用管理者みたいなことをやっていたわけですから、酔狂も極みですね。

最後に

今では簡単にインターネット上、クラウド上に掲示板やブログを作れますし、巨大なSNSも利用できますが、コミュニケーションのツールとしてみると草の根BBSは地域密着型としてよくできていたように思います。あのころの雰囲気を、今の若い人にもぜひ味わってほしいなと思います。SD

注4) いわゆる、オフラインミーティング。

注5) 現在でも開発が行われているようです。http://pocketstudio.jp/linux/?KTBBS

もう表計算ソフトに頼らない

迷えるマネージャのための プロジェクト管理ツール再入門

第2回 JIRA Agileでスクラム開発にチャレンジ

Software Design編集部

既存のウォーターフォール開発に限界を感じているが、アジャイル開発は難しそうでなかなか踏み切れない……。そうしたプロジェクトチームにぜひ試していただきたいのが「JIRA Agile」です。今回は、JIRA Agileを使って実際にプロジェクトを進めていく流れを解説します。

アジャイル開発の実践をサポートするJIRA Agile

顧客の要求に柔軟に対応しつつ、変更要求によって発生するリスクを最小限に抑えるために、徐々に広まりつつあるのが「アジャイル開発」です。当初は自社で展開するWebサービスやソーシャルゲームの開発など、比較的規模が小さい開発現場で使われることが多かったアジャイル開発ですが、昨今では大規模システム開発にも適用する事例が増えつつあります。

このアジャイル開発を実践するには、従来のウォーターフォール型の開発手法とは異なる考え方、そしてアジャイルならではのプロジェクトの進め方について理解しなければなりません。インターネット上にはアジャイルについて解説したコンテンツが無数にあり、また書籍も数多く発行されていますが、それらだけを見て実際にアジャイル開発に臨むのは難しいでしょう。そこでぜひ活用したいのが、アジャイル開発のノウハウを詰め込んだツールです。

アジャイル開発を管理するツールには、オープンソースのプロジェクト管理ツール「Redmine」とそのプラグインである「Backlogs」の組み合わせや、Apacheなどのオープンソースプロジェクトで活用されている「JIRA」のプラグインである「JIRA Agile」などがあります。ここでは、実際にJIRA + JIRA Agileでアジャイル開発を進めていく流れを見ていきます。

短期間の開発を繰り返すことで手戻りのリスクを回避

アジャイル開発の手法はいくつか存在しますが、その中でも多くの開発現場で使われているのが、チームでプロジェクトを進めるためのフレームワークである「スクラム（Scrum）」です。JIRA Agileはこの手法でアジャイル開発を進めるように設計されています。

このスクラムを含むアジャイル開発に共通す

▼図1 JIRA Agileのインターフェース。画面上部にスプリントに割り当てたタスク（スプリントバックログ）、その下にプロジェクトの全タスク（プロダクトバックログ）が表示される

▼図2　登録したタスクにストーリーポイントを割り当てているところ。設定したストーリーポイントは、一覧表示されている各タスクの右端に表示される

る特徴として、1週間から2週間、あるいは1ヵ月といった短い期間で開発を繰り返し、その期間ごとにユーザが利用できる機能を提供していくことが挙げられます。従来のウォーターフォール型の開発では、ユーザは実際に動作するソフトウェアをプロジェクトの終わり間際まで見られないため、開発者はそのときまでユーザからのフィードバックを得られないという問題がありました。これに対してアジャイル開発では、短い期間で開発を繰り返し、そのたびに実際に動作するソフトウェアをユーザに見せてフィードバックを得ます。これにより、プロジェクトの終わり間際に修正依頼が多発して大きな手戻りが生じ、プロジェクトが遅延してしまうといったリスクを回避できます。なおスクラムでは、この繰り返す開発期間のことを「スプリント」と呼んでいます。

「プロダクトバックログ」と「スプリントバックログ」で作業すべき内容を管理することもアジャイル開発の特徴でしょう。いずれのバックログも作業すべきタスクをまとめるためのものですが、プロダクトバックログがプロジェクト全体のタスクを羅列したものであるのに対し、スプリントバックログはそのスプリントで実施するタスクをまとめたものです(**図1**)。まずプロジェクト全体のタスクをプロダクトバックログにまとめ、そこからスプリントで開発するものをスプリントバックログに抜き出していくというのが実際の流れです。

ストーリーポイントの割り当てでスプリント単位の作業量を計測

JIRA Agileにおいてプロダクトバックログに相当するのが「計画ボード」と呼ばれるインターフェースです。まずここにタスクを追加し、そこからスプリントで実行するタスクを抜き出してスプリントバックログを作成していきます。

タスクを登録する際に重要なのが「ストーリーポイント」の割り当てです(**図2**)。ストーリーポイントは、タスクを実行するのに必要な作業量を表す相対的な値です。この値は、特定の単位を持つものではありません。たとえば短時間で完了する簡単なタスクには「1」、その倍の作業量だと思われるタスクには「2」、さらに難しいタスクには「3」を割り当てるというように、それぞれのタスクのストーリーポイントを見積もって設定します。

このストーリーポイントを割り当てることが重要な理由は、1回のスプリントで実行できる

▼図3　バージョンの登録インターフェース。それぞれのタスクをバージョンで分類し、そのタスクにどのバージョンで対応するかを指定できる

タスクを見極めるためです。たとえば1回のスプリントで、ストーリーポイントが「1」のタスクを2つ、「2」のタスクを1つ消化できたとしましょう。そうすると、1スプリントあたりの作業量は4ストーリーポイントということになり、次回のスプリントでも同じだけのストーリーポイントを消化できるという前提で、作業すべきタスクを選択することが可能になります。

JIRA Agileでは、このようにして作成したタスクを「バージョン」や「エピック(カテゴリ)」で分類するためのしくみを用意しています(**図3**)。洗い出したタスクを複数のバージョンに割り当てる、あるいはサーバサイドとフロントサイドでタスクをカテゴライズする、といった場面で便利でしょう。

ボードに登録したタスクはドラッグ&ドロップで上下に動かすことができ、これによって優先順位を指定します。優先順位が高いものを上にすれば、処理すべき順番を一目で把握できます。

プロジェクトの進捗状況を一目で把握できる“かんばん”

このようにボードにタスクを登録したら、スプリントバックログの作成に進みます。画面右にある「スプリントの作成」をクリックすると、ボードの中に「スプリント1」という項目が現れます。登録したタスクをここにドラッグ&ドロップで移動し、そのスプリントで作業する内容を決定します。最後に「スプリントの開始」ボタンをクリックし、開始日と終了日を設定すれば、いよいよスプリントが始まります。

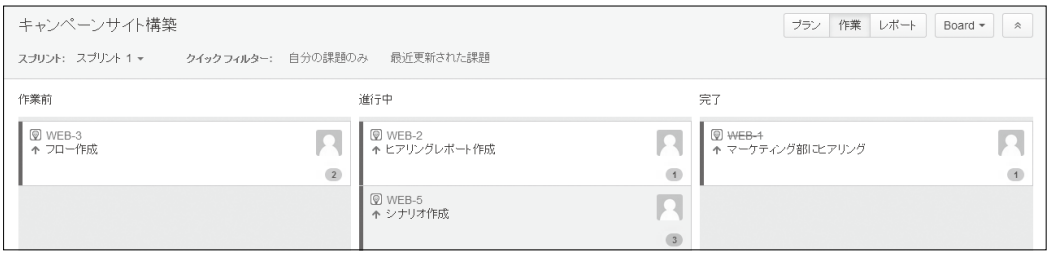
スプリントの実行中は、「作業ボード」でタスクの進捗状況を見ることができます(**図4**)。これはアジャイル開発で一般的に使われる「かんばん」と呼ばれるもので、左側に作業前のタスクが配置されており、作業を開始したらそのタスクを中央の「進行中」に、作業が完了したら右側にある「完了」にドラッグ&ドロップで動かします。これにより、どのタスクが進行中で、どのタスクが完了しているのか、スプリントの進捗状況が一目瞭然になります。

標準で用意されているかんばんのステータスは3段階(作業前／進行中／完了)ですが、新たにステータスを追加したり、既存のステータスを削除したりすることもできます。たとえば、進行中と完了の間に、「ユーザ確認中」というステータスを追加できます。実際のチームのワークフローに合わせて設定すればよいでしょう。また、作業中にできるタスクは3つまで、というようにそれぞれのステータスのタスク数に上限を指定したり、ユーザやエピックごとにタスクを分割するスイムレーンを設定したりすることもできます。

バーンダウンチャートやベロシティグラフも装備

スプリントにおけるストーリーポイントの消化状況を把握できる「バーンダウンチャート」もJIRA Agileには用意されています(**図5**)。残りのストーリーポイントがどれだけあるのかを時間軸で見ることができるため、このチャートをチェックすればスプリントバックログに登録し

▼図4　スプリントの進捗を確認できる「作業ボード」。ドラッグ&ドロップでタスクをそれぞれのステータスに動かせる。ステータスの内容はカスタマイズも可能



▼図5　ストーリーポイントをベースに、スプリントの進捗度合いをグラフで確認できる「バーンダウンチャート」。JIRA Agileにはこのほかにもさまざまなレポート機能がある

ているタスクをすべて完了できそうか、それともタスクが残りそうかを把握できます。

もう1つ、重要なレポート機能が「ベロシティグラフ」です。ベロシティは、チームが1回のスプリントで消費したストーリーポイントで、スプリントにどの程度のタスクを割り当てるかを考えるときに役立ちます。一般的には、過去数回のベロシティの平均値をそのチームが1回のスプリントで消化できるストーリーポイントだととらえ、それに応じてスプリントバックログを作成します。スプリントへのタスクの割り当て時に、このベロシティグラフの内容が参考になるでしょう。

なお、JIRA Agileに登録したプロジェクトやタスクはJIRA本体にも同時に登録されるので、前回解説したとおり、タスクごとに用意されたコメント欄を使ってメンバーと議論したり、豊富なレポート機能を使ってプロジェクトをチェックしたりすることができます。つまりJIRAのデータベースを使いつつ、スクラム開発のためのインターフェースを新たに追加するのがJIRA Agileだというわけです。JIRAが備える豊富な機能を使いながらアジャイル開発を進められることも、JIRA Agileの大きな魅力です。

ここまで解説したとおり、JIRA Agileはプロダクトバックログの作成やスプリントバックログへのタスクの割り当て、かんばんやバーンダウンチャートによる進捗状況の確認など、スクラムで進められているプロジェクトの管理に必要な機能を一通りそろえています。JIRA Agileで提供されている機能を一種のテンプレートとして利用すれば、アジャイル開発を始める際、プロジェクトの進行を管理するのに役立つでしょう。

リックソフトのサイトでは、JIRA Agileの体験版をダウンロードできます。ぜひお試しください。SD

JIRA Agile体験版のダウンロードはこちらから：
https://www.ricksoft.jp/product/atlassian/jira

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

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

Service

パソナテック、非IT人材を対象とした「ハッカー部!」開講

㈱パソナテックは、非エンジニアの社会人を対象とした個別学習型プログラミング講座「ハッカー部！」（http://hackerclub.co/）を9月3日から開講している。

「ハッカー部！」はITスキルを持たない非エンジニア人材に対し、受講者が実現したいサービスに必要なITスキル研修を提供し、アイデアの実現化を支援することを目的としている。講座の受講形式はオンラインをメインとしており、講師による週1回の直接指導も受けることができる。カリキュラムは、受講者が開発したいサービスに合わせて一人一人にオリジナルのものが作られ、最大1年の期間の中でアイデアの実現に取り組む。サービスの完成に必要な分野の知識をスモールステップで順番に学んでいき、わからないことがあればオンラインで質問もできる。週1回の直接指導の日には進捗発表会も設けられ、受講者同士の成果物を発表し合う。

専門書や講座などでのプログラミングの勉強に挫折した人や、エンジニアやクライアントと円滑に仕事を進めたいWebディレクターや営業担当者にお勧めの講座である。受講費は110,000円（税別）。体験講座が、毎週水曜日に随時開催されている。

CONTACT ㈱パソナテック
URL http://www.pasonatech.co.jp

Report

レバレジーズ、エンジニア・Webデザイナー向け無料イベント「ヒカ☆ラボ」開催中

レバレジーズ㈱はヒカリエ（東京都渋谷区）にある自社オフィスにて、エンジニア・Webデザイナ向けの無料イベント「ヒカリエ・ラボラトリー」（略してヒカ☆ラボ）を定期的に開催している。イベントには業界の有識者を招き、技術的な内容から、キャリアの話題まで幅広いテーマでセッションが行われる。http://at-agent.jp/service/event/からイベントの予定と申し込みができる。

本レポートでは9月9日、16日に行われたプログラムを紹介する。

読みやすくてメンテナブルなCSS設計

9月9日に行われたのは、㈱LIGの堀口誠人氏と、㈱リッチメディアの斉藤祐也氏が登壇したCSS設計に関するセッション。

CSS設計とは、CSSをより体系立て、より構造化させることで、制作とメンテナンスを容易に行えるようにすること。CSSに強い拡張性、保守性、明瞭性を持たせることが重要だという。セッションでは、OOCSS（オブジェクト指向CSS）、BEM（Block、Element、Modifierの要素からなるHTML・CSSの設計手法）、SMACSS（Scalable and Modular Architecture for CSS：5つにカテゴライズされたCSSのルール）が紹介された。とくにOOCSSは、Twitter、YouTube、GitHubなどのWebサイトの設計で活用されており、すべてのスタイルをクラス化し、その組み合わせでコードを書いていくことで、メンテナンスが楽に、再利用もしやすいCSSを書くことができると、強く推した。


▲㈱LIG　堀口 誠人氏


▲㈱リッチメディア　斉藤 祐也氏

ウェブ分析の価値を発揮するレポート術とコミュニケーション


▲アマゾンジャパン㈱　小川 卓氏

9月16日 に はWebアナリストとして有名な、アマゾンジャパン㈱の小川卓氏が、Web分析において、「どのように分析結果をレポートにまとめるか」「レポートを、エンジニアや経営層にどのように伝えるか」というテーマでセッションを行った。

氏によると、レポート作成の目的は、そこから気づきを発見し、開発的・経営的な施策に繋げることが一番の目的だという。そのためには、報告対象者が必要とする内容で、メッセージが明確かつ、次に何をすればいいかを具体的に示すレポートを、定期的に作成・共有するべきだと強調した。また、それを伝える際には、事前に関係者に施策周りの相談を行い、発表後の質疑時間を十分に取ることが大事だと語った。

CONTACT レバレジーズ㈱
URL http://leverages.jp

Event

11月23日～24日、「Maker Faire Tokyo 2014」開催

11月23日～24日、東京国際展示場（東京都江東区）にて「Maker Faire Tokyo 2014」が開催される。

「Maker Faire Tokyo」は㈱オライリー・ジャパン発行の雑誌「Make:」日本語版の読者を中心に、エレクトロニクス（電子工作）、DIY、サイエンス、ロボット、アートなど、異なるジャンルの「Maker」（作り手）たちの発表の場、交流の場として2008年から開催されている。会場には、テクノロジを自由な発想で使いこなすMakerとその作品が数多く集まり、国内外の最新技術に触れることができる。過去には、自作のプラネタリウム、トランスフォーマーのように自力で変形するロボット、投げられたゴミのほうに移動するゴミ箱、電子部品で作られたアクセサリなどが展示された。

▼イベント詳細

日時	2014年11月23日（日）12：00～19：00 24日（月・祝）10：00～18：00
会場	東京国際展示場：東京都江東区有明3-11-1 西3ホール＋屋外展示場
入場料	前売：大人1,000円、18歳以下500円 当日：大人1,500円、18歳以下700円
出展者数	350組（予定）
主催	㈱オライリー・ジャパン

CONTACT Maker Faire Tokyo 2014
URL http://makezine.jp/event/mft2014

Report

アトラシアン、「JIRA＋JIRA Service Deskセミナー」および「Getting Git Right ～きっとできるGitセミナー」を開催

アトラシアン㈱は、9月26日、同社の「JIRA」および「JIRA Service Desk」を解説するセミナーと、「Git」の利用を促すセミナーの2つを開催した。

JIRAはもともとソフトウェア開発の課題管理を行うツールとして開発されているが、いまや開発現場だけではなく、ヘルプデスクやマーケティング、資産管理など多岐にわたる用途に活用されているとのこと。こういった業務を行うチームではJIRAよりもシンプルな使い勝手を望む声も多く、それに応えるのがJIRA Service Deskとなる。同社はソフトウェア開発チームにとどまらず、企業内のすべてのチームが協力できる手助けをするツールを目指しているとのこと。

Gitセミナーではそのメリットが説かれ、同社が行っているワークフローを紹介しながら、企業文化に応じたやり方を見つけ出してほしいと語った。同社のGit関連製品としてはStash、Bitbucketがあり、JIRAやコラボレーションツールのConfluenceとの連携によるコードレビューも含めたデモで、生産性が向上されることをアピールした。

CONTACT アトラシアン㈱
URL https://www.atlassian.com/ja

Software

グレープシティ、「CDataシリーズ」販売開始

グレープシティ㈱は、SalesforceやAmazon Web Servicesなどのクラウドサービス、SharePointやLDAPなどの社内システム、Excelなどのローカルマシンのファイルといったシステム上のデータを、業務アプリケーションに連携させるためのライブラリ「CDataシリーズ」（日本語版）を9月26日から販売開始した。

CData各製品を開発環境のマシンにインストールし簡単な設定を行うだけで、.NETであればADO.NET、JavaであればJDBCドライバへの接続を通したシステム連携が可能になる。連携先システムごとの独自の接続方法やAPIの仕様を学習することなく、開発者が慣れ親しんだ通常のデータベース接続方法で簡単にシステム連携ができるため、自社の業務アプリケーションと連携させたいシステムを自由に選択できるようになる。

ラインナップの第一弾として「CData ADO.NET Provider for Salesforce 4」を始めとした、Salesforceとの連携を実現する4製品を現在販売している。これらを利用すると、Salesforce上にあるリード情報や取引先企業、商談アイテムなど各種のデータソースと簡単に接続できるようになる。

CONTACT グレープシティ㈱
URL http://www.grapecity.com

Report

「YAPC::Asia TOKYO 2014」開催

8月28日～30日、「YAPC::Asia Tokyo 2014」が慶應義塾大学日吉キャンパス協生館（神奈川県横浜市）で開催された（28日は前夜祭）。

YAPCは、「Yet Another Perl Conference」の略称で、プログラミング言語Perlをメインテーマとした世界規模のイベント。日本では「YAPC::Asia Tokyo」として2006年から毎年開催されている。今年は、㈱ディー・エヌ・エー（以下、DeNA）スポンサーのランチセッションが行われ、お弁当が振る舞われるとともに、クイズアプリ「QuizNow」の実装面の話題を紹介するプレゼンテーションが、開発者によって行われた。本レポートでは29日に行われたプログラムをいくつか紹介する。

▲ランチセッションの様子

インフラエンジニア（狭義）は死んだ

LINE㈱のインフラエンジニアであるSatoshi Suzuki氏は「コードが書けないインフラエンジニアは生き残れるのか」というテーマでセッションを行った。

最近では、インフラ管理のツールとして、ログ収集管理ツールFluentdや、統合監視ツールNagiosなど、コードで自由にプラグインが書けるオープンソースのツールが増えてきた。Suzuki氏はそういった状況を背景に、インフラ管理上の課題で、ソフトウェアで解決できるものが見つかったとき、すぐに手を動かせるかどうかが非常に重要だと指摘し、「コードが書けるインフラエンジニア」の優位性を語った。

また、一番憂慮すべきことは、「○○エンジニアだから△△はしない」という先入観であると強調し、勉強会などへ積極的に参加して、同じ問題を共有できる仲間を増やすことが、今の自分にとって一番大事なことだと語り、セッションを終えた。

▲LINE㈱　Satoshi Suzuki氏

Introducing Swift - and the Sunset of Our Culture?

Perlプログラマとしても有名な小飼弾氏が行ったのは、Appleが開発したプログラミング言語「Swift」についてのセッション。

小飼氏は、持参したMacBookのXcode上で、Swiftで書かれた「Hello, World」や「FizzBuzz」を動かし、言語の特長などについて話した。Swiftを好きになった理由として、新しい演算子を自由に定義できること、型推論によりコードを短くできることなどを挙げた。目新しい機能は少ないが、ほかの言語のいいところ取りというのが、Swiftの全体的な印象だという。

また、セッションの終盤では、プログラミング言語を大きく「トップダウン型」と「ボトムダウン型」に分け、プログラミング言語の今後について語った。前者はSwiftやJava、Go言語など、企業がメインで開発を行い、リリースするもの。後者はオープンソースで開発が行われるもの。氏は、オープンソースでの開発は速度が遅いと指摘し、今後はトップダウン型の言語が勢いづいていくと分析している。

本誌12月号からは小飼氏によるSwiftの連載が始まるので、詳しい情報が得られるだろう。


▲小飼 弾氏

DeNAが歩んだデプロイ自動化への道

アプリケーションのデプロイメント自動化について、DeNAの青猫氏がセッションを行った。

アプリケーションエンジニアにとって、リリースプロセスに難を感じるかどうかは非常に重要なことだと青猫氏は語る。「開発と本番に環境の差異を作らない」、「問題は早期に発見されるようにする」ために、デプロイの自動化と継続的なテストを強く推した。

青猫氏は実例として、DeNAでの実際の運用・開発体制について話した。DeNAが提供する「Mobage」はリリースから現在に至るまでサービスを拡張し続けており、それに伴ってシステムも肥大化していった。レガシーから脱却するため、Mobageのリリース当初からある巨大なコンポーネントを、意味のあるサービス単位に切り分けてシステム全体を整理し、現在ではJenkinsを用いた継続的テスト・自動デプロイを実現している。それにより、開発が楽になり、リリースへの精神的な障壁がなくなったそうだ。


▲㈱ディー・エヌ・エー　青猫氏

CONTACT　YAPC Asia TOKYO 2014
URL http://yapcasia.org/2014

Report

「RubyKaigi 2014」開催

9月18日から20日にかけて、タワーホール船堀（東京都江戸川区）で「RubyKaigi 2014」が開催された。

「RubyKaigi」はプログラミング言語Rubyに関する大規模なカンファレンス。Rubyのコミッタ、開発者によるセッションがメインのイベントである。

CRuby Committers Who's Who in 2014

18日のKey noteでは、Rubyのコミッタである近永智之氏が、CRubyのコミッタを紹介する発表を行った。

現在CRubyには84人のコミッタが存在し、そのうちアクティブコミッタ（直近1年以内にコミットした人）は50人であるという。近永氏は、Ruby Kaigi 2014にスピーカとして参加している15人のコミッタについて、コミット実績、人物像を1人1人紹介した。近永氏自身もコミッタであり、Ruby2.1系のブランチメンテナも務めていたそうだ。

▲近永 智之氏

What is your app's problem?

本誌の連載記事「Heroku女子の開発日記」の著者でもある織田敬子氏は、Herokuのサポートエンジニアとしての経験から、Herokuを使っているユーザに向けた、ケースごとのエラー対策について発表した。

織田氏によると、「H12 - Request timeout」がもっともよく出力されるエラーだそうだ。そのおもな原因として、リクエストの遅延、アクセスの集中、割り当てメモリの超過が挙げられる。解決策としては、New Relic プラグインやログ解析のアドオンで、アプリケーションを監視する環境を整えておき、問題を切り分けること、そして、同時リクエストをさばけるUnicornやPumaのようなサーバを選択することが大事だと話した。

▲Heroku　織田 敬子氏

mRuby on LEGO Mindstorms EV3

普段はプリンターメーカーに勤務しているTakehiko Yoshida氏は「組込みシステムにmrubyを採用できるか」という問いかけを起点とした、LEGO Mindstorm EV3（以下、EV3）にmrubyを組み込む発表を行った。

EV3はARM9プロセッサ、16MBのRAM搭載のプログラミングロボット教材。mrubyは、まつもとゆきひろ氏が開発した組込み向けの軽量版Ruby。発表では、それら2つと音センサーを使って、拍手をすると本殿が開くしかけの「Ruby Jinja」（**http://www.youtube.com/watch?v=dg-ufN4AJFk**）を披露した。

YOSHIDA氏はまとめとして、mrubyはC/C++やアセンブラと競合することこそないが、クラウドサービスとの連携など、高機能／多機能の電子デバイスには今後使われていくのではないかと語った。

▲Takehiko Yoshida氏

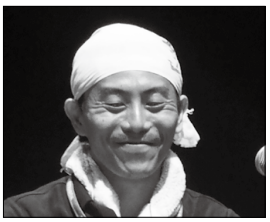
Scarable deployments

Rubyの最年少コミッタである福森匠大氏は、現在インフラエンジニアとして働いている㈱クックパッドでの、Railsアプリケーションのデプロイ高速化について話した。

クックパッドのWebサービスでは、ピーク時には140サーバに対して1日に10回ものデプロイが行われている。以前はCapistrano2、ssh、rsyncでデプロイを行っていたが、平均で10分ほど時間がかかっていたという。福森氏は高速化のため、オーケストレーションツールSerfを利用したデプロイツール「Mamiya」をチームで新たに開発しシステムに適用したところ、デプロイ時間を平均1分にまで短縮できたそうだ。福森氏が所属するインフラチームは、「開発者の生産性を向上させる」ことを大きな目的としており、今回のデプロイ高速化は、それに大きく貢献したはずだ。

▲クックパッド㈱　福森 匠大氏

3日目には、Ruby Kaigi最後のプログラムとして、1日目のオープニングトークも務めた日本Rubyの会理事の角谷信太郎氏がクローズトークを行い、閉会の辞を述べた。



▲日本Rubyの会理事　角谷 信太郎氏

CONTACT　RubyKaigi 2014
URL http://rubykaigi.org/2014

Hardware

レノボ・ジャパン、「Lenovo HORIZON 2」「Lenovo HORIZON 2e」発売

レノボ・ジャパン㈱は、27インチの2 in 1デスクトップPC「Lenovo HORIZON 2」、「Lenovo HORIZON 2e」（21.5インチの省スペースモデル）を10月下旬に発売することを発表した。

同製品は、さまざまな用途に使える「マルチモードPC」。10点マルチタッチパネルを持ち、複数人で画像を見たり、ゲームを楽しんだりできる一方、同梱のMicrosoft Officeを使ってデスクワークもできる。パレットのような円状のランチャーを持つHORIZON 2（2e）専用のインターフェース「AURA」がプリインストールされており、タッチ&ジェスチャーで画像や音楽など、さまざまなデータを操作できる。

▲AURAを操作するレノボ・ジャパン㈱ 藤井 宏明氏

また、専用の無料アプリ「Aura app for Android 4.2」を使えば、Android端末を最大4台まで同時接続でき、画像の共有などをジェスチャーで簡単に行える。

価格は、HORIZON 2が20万円前後、HORIZON 2eが17万円前後と予定されている。キーボード・マウスのほかに、ゲーム用のジョイスティック、ホッケーストライカーも同梱される。

（※写真中のテーブルスタンドはHORIZON 2専用となっており、別売り）

▼詳細

	HORIZON 2	HORIZON 2e
CPU	インテル Core i5-4210U	インテル Core i3-4030U
OS	Windows 8.1	
メモリ	8GB	4GB
HDD	1TB	
ディスプレイ	27 インチ FHD 液晶	21.5 インチ FHD 液晶
インターフェース	USB3.0、HDMI（入力）、NFC	USB3.0、HDMI（入力）、NFC、6 in 1 メディアカードリーダー

CONTACT レノボ・ジャパン㈱
URL http://www.lenovo.com/

Report

「AWS Cloud Storage & DB Day 2014」開催

9月9日、青山ダイヤモンドホール（東京）にて「AWS Cloud Storage & DB Day 2014」が開催された。本イベントは、Amazon Web Services（以下、AWS）のクラウドストレージとデータベースにフォーカスした参加費無料のカンファレンス。AWSに関する最新情報や、AWSユーザ企業の導入事例が紹介された。イベントの最初に行われた基調講演の内容を紹介する。

講演ではまず、AWS社のPaul Duffy氏が、AWSの各種ストレージ・データベースサービスを紹介したあと、「Content Gravity」という概念について説明した。これは「データをクラウドに集めることで、そのまわりにサービスがつながり、活用が加速する」という意味。AWSがその流れを促進させていくとPaul氏は語った。

▲AWS社　Paul Duffy氏

また、2014年8月末に一般提供を開始したドキュメント保存・共有サービス「Amazon Zocalo」については、さまざまな端末から、あらゆる形式のドキュメントにアクセスできる優れたサービスだと紹介した。

同じく基調講演では、AWSユーザのSony Media Cloud Services社のBen Masek氏と、㈱ガリバーインターナショナルの北島昇氏が登壇し、それぞれの企業の導入事例を紹介した。

Ben氏は、世界に散らばるクリエイターが同じクラウドワークステーションで、同じデータにアクセスし、各人の環境のソフトウェアで映像を編集できる「Sony Media Cloud Services」を紹介した。サービスの基盤として、Amazon Simple Strage ServiceとAmazon Glacierが利用されている。

北島氏は、「DRIVE+」という、「コネクテッド・カー」サービスを紹介した。このサービスでは駐車場での自分の車の位置やガソリンの残量などを、スマートフォンからLINEのスタンプを送ることで知ることができる。サービスに使われるデータは車搭載のOBD端末から取得され、その収集・解析をAmazon Kinesisが担っている。

CONTACT アマゾンデータサービスジャパン㈱
URL http://aws.amazon.com/jp

ひみつのLinux通信

作)くつなりょうすけ
@ryosuke927

第11回 cronの罠

SDを後ろから読ませるくつな先生の連載もそろそろ1年。変T業界の危険な天使(くつな先生)にツイート(元気玉)をくんろ！

1　cronの復讐

2　cronの暴走

to be Continued

定期的に決まった時刻に実行するジョブはcronで、24時間稼働しないサーバで決まった時間に起動してないことがある場合の定期実行ジョブはanacronでフォローして、一度だけ指定した時間に自動的に実行したいのはatで処理させましょう。でも、わかっているのにcronを使っちゃう……。たぶん設定ファイルが残る安心感があると思うのですよね。一時的に置かれるのと違う安心感がありません？……ないですか、そうですか。最近は手に書いてあるのでやらなくなりました。時間指定も、「とりあえず"＊"を5個書いてー、ユーザとコマンドを指定して……」ってやるからたまにこういうボカミスします。ホント、素振りは重要ですよね。

Letters from Readers

夏から秋へ

やっと暑さが和らぎ始め、過ごしやすい季節になりました。暑くてなかなか集中できなかったスポーツや読書が捗ります。我が家のパソコンも、ファンの回転が比較的穏やかになり、快適に動くようになりました。暑いと処理効率が落ちるのは人間もコンピュータも同じですね。11月／12月号からは新連載もいくつか始まるので、これを機に新しい分野の勉強を始めてみてはいかがでしょう?

2014年9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　C言語のポインタとオブジェクト指向

新人エンジニアがC言語を学ぶときの大きな関門となる「ポインタ」、そしてJava、Rubyなどの「オブジェクト指向」。基礎から応用まで、幅広く解説しました。ポインタの危険性、オブジェクト指向への幻想など、テクノロジの「負」の部分にもスポットを当てています。

ポインタの理解が進まないのは、ポインタを使わなくても良い例でポインタを説明しているから。→うまく言い当てている気がします。

岩手県／隼さん

今回のように、あるテーマについて多くの方が短い解説文を執筆するというスタイルは、さまざまな知見や考え方を知ることができて良いと思います。

東京都／山下さん

オブジェクト指向は基本的なことながら、やっぱりまだ理解できていないので、こういう特集が定期的に組まれるとうれしい。

東京都／tomato360さん

30年近くなってもポインタは、C言語の鬼門なんですね。

宮城県／伊藤さん

ポインタについては、まず理解するのに敷居が高かった記憶がありますが、危険を承知で使用するには非常に便利な機能だと感じています。記事では触れられませんでしたが、構造体を取り扱う際には非常に助けられました（プログラムの可視性が低下するのが頭の痛い点ではあるのですが）。

熊本県／鈴木さん

最近、RubyとかJavaScriptのゆる～い言語ばかり使っているので、「C言語ってこんなに面倒だったっけ?」と過去のにがーい経験を忘れている自分がいました。ポインタもわかってしまうと「なーんだ、最初からそう教えてくれよ」という感じなんですが、その壁を超えるまでがたいへんですね……。

神奈川県／くまーさん

計14名の有識者の方々に、それぞれ違う視点から記事を書いていただきました。どのように学ぶのか、どんな場面で使えばいいのか、そもそも使っていいものなのかどうか、疑問が解消された読者の方も多いのではないでしょうか。

第2特集　クラスタリングの教科書

データセンターなどで高可用性を実現するための技術「クラスタリング」。そのしくみ、データセンターでの実際の構築例、そしてデータベースにおけるクラスタリング構成について解説しました。

ほかではなかなか読めないので良いと思います。

東京都／aszwさん

理解している「つもり」のクラスタリングについて再確認でき、かつ最近のトレンドも理解でき、とてもためになる記事。

長崎県／romeosheartさん

データセンターを支えるテクノロジがよくわかった。

長崎県／都市伝説作家さん

ユーザにとってシステムがブラックボックスになりがちなクラウドサービス。その内部（データセンター）で自分のシステムの可用性がどのように保たれているのか、知識をふまえて意識することが大切です。

一般記事　SoftLayerを使ってみませんか?

IBMのクラウドIaaS製品「SoftLayer」の入門記事です。今回は、基本的な使い方と「ベアメタルサーバ」について説明しました。

ちょうどSoftLayerが気になっていたところです。今年日本リージョンができるというお話でとてもワクワクしています。

東京都／n0tsさん

使ってみたくなった。ベアメタルという特徴が良い。

神奈川県／藤村さん

仮想サーバとほぼ同じ使い勝手で物理サーバを扱えるSoftLayerは、読者のみなさんにとっても注目度が高いのではないでしょうか。

一般記事　【実力検証】NICをまとめて高速通信!（前編）

複数のNICのポートをまとめて高速な通信を実現する「チーミング」技術を解説しました。前編では複数のポートを1つの論理チャンネルのように扱う「リンク・アグリゲーション」を取り上げました。

一歩引いて考えてみれば、線の本数を増やせば高速化できそうな気がしますが、これまであまり考えたことがなく、目からうろこが落ちた思いです。

愛知県／NGC2068さん

もう昔のIDE接続より速そう。

奈良県／アセンブラノスタルジーさん

インフラの高速化を要求されたとき、最新の機器を新たに購入するのではなく、今ある機器を最大限利用して目標を達成する。技術でハードウェアの差を埋めることができれば、それに越したことはありませんね。

一般記事　オーケストレーションツールSerf・Consul入門【Serf編】

サービスの配備・設定・管理などを自動化できるオーケストレーションツール。第1回ではシステムにクラスタを構成し、一斉に処理を行わせる「Serf」の紹介です。

Serfは未導入だが試してみたい

神奈川県／粟飯島さん

弊社は独自開発したものを使ってた気がするので気になります!

東京都／臼井さん

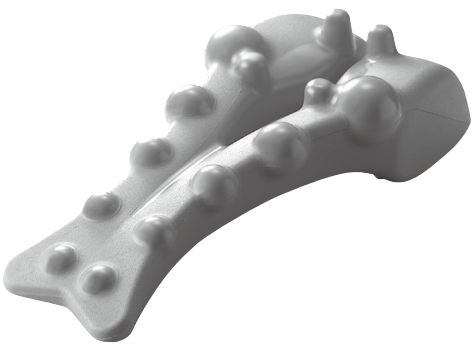
開発・運用をコードで自動化する大きな流れがあり、それを実現するツールが続々と出ています。あとで楽をするための初期投資として、各種ツールの使い方を身に付けましょう。

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。

揉まれる肩・首スッキリピロー

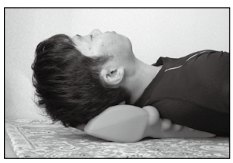
7,179円（税込）　ドリーム／http://mydream.co.jp/



今回は慢性的な肩こりを患（わずら）っている人向けに、マッサージグッズを紹介します。「揉まれる肩・首スッキリピロー」は1人でも首、肩、背中のツボを刺激できるグッズです。本製品を床に敷き、その上に寝そべるだけで、本製品についている突起がツボを刺激します（**写真**）。首や肩はともかく背中は自分で一人では押圧できない部分ですので、そこを刺激できるのは便利です。

また、本製品を使うと軽く体を反らせる体勢になるため、デスクワークで猫背気味の背筋をピーンと伸ばせます。これはツボの刺激とはまた違う気持ちよさがあります。付属の説明書きにあるように、本製品の上に寝そべりながら「両腕を左右に広げる」「両腕を頭の上へ伸ばす」といったエクササイズと合わせると、凝り固まった体がほぐされるのがよくわかります。

人によっては刺激が強過ぎるかもしれませんが、そんな場合は、タオルを敷いたうえで使用すれば刺激を調整できるようです。（読者プレゼントあります。p.16参照）



▲写真　枕のようにして使う

9月号のプレゼント当選者は、次の皆さまです

①LED キーボード DN-11255 ..... 神奈川県　江田辰雄様
②エルゴスティックマウス 400-MA059 ..... 愛知県　杉浦啓明様

★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
December 2014

2014年12月号
定価（本体1,220円+税）
176ページ

11月18日発売

［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
本格的にわかる威力と効果

第1章 Docker導入のメリット／第2章 chrootからDockerを根本的に理解しよう!／第3章 Googleが作ったDocker管理ツールKubernetesの使い方

［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

■新連載 小飼 弾イチオシ!

Swift入門

※特集・記事内容は、予告なく変更される場合があります。あらかじめご容赦ください。

お詫びと訂正

以下の記事に誤りがございました。読者のみなさま、および関係者の方々にご迷惑をおかけしたことをお詫び申し上げます。

■2014年10月号　第1特集　第1章
- p.26、左段、17行目
  ［誤］java.util.SortedHashを使うと　［正］java.util.SortedSetを使うと
- p.27、左段、リスト17
  ［誤］map.put(word, count++);　［正］map.put(word, ++count);
- p.27、右段、リスト19
  ［誤］map.put(word, count++);　［正］map.put(word, ++count);

SD Staff Room

●諸事情でバッテリーをあげてしまったバイクを復活させたら、交換修理でお金が飛びまくり。趣味のものは本当にお金がかかる。かけてきたお金は計りしれない。メンテナンスが終わり、久々にバイクに乗ったら、なくした体の一部を取り戻せたような気分になった。これからツーリングに良い季節だ～。（本）

●今年も残すところもう2ヵ月だが、昨今、災害が多発している。これに加えてエボラ出血熱やデング熱、セアカゴケグモなど、地球温暖化に伴う熱帯化が一因な現象も多発している。そんな中1日を乗り越えられて幸せだと思う。何が言いたいのかというと、最近忙しくてそんなこと考える暇がなかったってこと。（幕）

●運動会。スポーツが得意ではなかった自分にとって特段良い思い出もないイベントでした。でも、今年はちょっと特別なものになりました。親から譲り受けたのが声のでかさというのがあれですが、低学年のころから憧れていた応援団長という大役をつとめたことは、彼にとって得難い経験になったかと。（キ）

●長時間のデスクワークのせいか慢性的な肩こり、首こりを患っています。2ヵ月ほど前には、首をまげられないほど痛むまでになってしまいました（最近、やっと治りました）。「エンジニアの能率を高める一品」で、マッサージ関連の商品が登場するときは、じつは私が肩こりに悩んでいるときです。（よし）

●中野ブロードウェイに遊びに行きました。サブカル系の店が多く並ぶショッピングセンターで、古い漫画など、レアアイテムを探すにはもってこいの場所です。その日はお客さんも多くて活気があったのですが、以前訪れたときよりもシャッターが増えたような気がします。なくなってほしくないものですね。（な）

●先月休みをもらいオーストラリアに行きました。天候にも恵まれ、オペラハウスやブルーマウンテンや星空もきれいに見られたし、コアラも抱っこできたし、グレートバリアリーフでは魚と一緒に泳げたし、シーフードも美味しかった。ただエアーズロックには行けなかったので、物価が安くなったころまた行きたいな。（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11月号

発行日
2014年11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。