special feature 1 Nginxことはじめ
special feature 2 ネットワーク基礎の基礎

Software Design

2014 7

2014年7月18日発行
毎月1回18日発行
通巻351号
（発刊285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

そろそろ

[多機能]
[高速処理]
[高負荷対策]

Nginx移行

special feature 1

を考えているあなたへ

あなたに
現場で培われた
ノウハウを
つたえましょう!

special feature 2
知っているようで知らない
DHCPサーバの教科書
当たり前が落とし穴! 知って得する基礎の基礎

特別企画
Rettyのサービス拡大を支えた
“たたき上げ”DevOps
リアルタイム監視ツール
「OpenTSDB」（後編）

短期集中特別企画
ITビジネスの足下を揺るがす大きなバグ
OpenSSLの脆弱性“Heartbleed”の教訓（前編）

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

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

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

第67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。

Docker

コンテナ型仮想化ツール「Docker」

オープンソースのコンテナ管理ソフトウェア「Docker」が注目を集めています。Dockerはアプリケーションをその実行環境ごと“コンテナ”にパッケージングすることで、デプロイや実行といったサーバ管理作業を簡略化することができるツールです。Amazon Web ServicesやGoogle Compute Engine、OpenShiftといった主要なクラウドサービスで採用されるなど、Docker人気の波は徐々に業界全体に広がっています。

Dockerでは、ミドルウェアやライブラリ、プログラミング言語の実行環境、フレームワーク、そしてアプリケーション本体をまとめてパッケージ化して管理します。このパッケージを「コンテナ」と呼びます。作成したコンテナは、別のサーバに移動やコピーをしてもそのまま動作させることができます。

Dockerのようなしくみは“コンテナ型の仮想化”と呼ばれます。仮想化というとVMwareやVirtual Boxなどによる完全仮想化が思い浮かびますが、コンテナ型の仮想化のしくみはそれとは大きく異なります。完全仮想化は、ハードウェアをエミュレートすることでホストOS上にまったく異なる独立したOS空間を構築します。それに対してコンテナの場合は、複数のコンテナが1つのホストOSおよびコンテナ管理ソフトウェアの上に同居し、リソースやライブラリをコンテナ間で共有します。

コンテナをアプリケーション側から見た場合、プロセスやネットワーク、リソースはそれぞれのコンテナで独立しており、管理者権限でのプログラムの実行も可能なため、独立した仮想OSのように見えます。一方でホスト側から見れば、各コンテナはそれぞれ1つのプロセスとして動作しており、エミュレーションも行っていません。このしくみによって、コンテナは隔離されたアプリケーション実行環境を実現しつつも、完全仮想化に比べて少ないディスクスペースや少ないメモリによって動作させることができます。

Docker が解決する問題

Dockerが注目を集める背景には、クラウドサービスの普及によって以前よりも手軽に多数のサーバ環境を扱えるようになったことが挙げられます。複数のサーバを利用する場合、環境構築の手間を最小限に抑えることが極めて重要です。また、アプリケーションをクラウド上の環境にデプロイする際に、手元のテスト環境との違いをどう吸収するのかも大きな課題になります。テスト環境では問題なく動作していたものが、クラウド上にデプロイしてみたら動かなかったという経験は誰でもしていることでしょう。

Dockerであればこれらの問題を回避することが可能です。アプリケーションの実行環境をまとめてコンテナ化するため、ホストOS側の設定に対する依存性を最小限に抑えながら、極めて高いポータビリティを実現することができるからです。

Dockerは、本誌2014年6月号で取り上げた「Infrastructure as Code」とも密接なかかわりがあります。Dockerではコンテナの構成を「Dockerfile」というテキストファイルに記述できます。Dockerfileには、OSのスクラッチイメージからアプリケーションが動作するまでの一連の作業や設定がすべて記載されます。これはアプリケーションの実行環境そのものをコードとして管理できるということを意味します。

システムの管理者にとっては、Dockerはインフラ管理を簡素化し、常に最新のニーズに対応した環境を構築していくための格好のツールです。一方で開発者にとっては、ライブラリの依存関係やデプロイの問題などを意識せず、アプリケーション実行環境としてのコンテナの構成のみ意識すればいいというメリットがあります。

本稿執筆時点では、DockerはVer. 0.11でRelease Candidate（正式版候補）の段階にあり、正式版のリリースも間近に迫っています。SD

docker
https://www.docker.io/

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

Contents

［第1特集］

そろそろ

Nginx移行を考えているあなたへ

［ 多機能 ］
［ 高速処理 ］
［高負荷対策］

017

第1章 ApacheからNginxへ移行するメリットとは 伊勢 幸一 018

第2章 各Webサーバの比較に見る、Nginxを導入する理由 佐野 裕 022

第3章 移行前のチェックポイントの洗い出し 長野 雅広 027

第4章 Nginxのインストールとコンフィグ設定の基本 橘 慎太郎 035

第5章 Nginx引っ越し本番! 田籠 聡 043

第6章 移行後に気を付けておくべきこと 田籠 聡 052

第7章 クラウドでのNginxの使い方 大久保 智之 055

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 Jul. 2014

2

［第2特集］

知っているようで知らない

DHCPサーバの教科書

063

Part1 その役割を確認! DHCPをご存じですか? 中井 悦司 064

Part2 じっくり押さえる基礎の基礎 DHCPの舞台裏 中井 悦司 069

Part3 動作原理を実機で確認してみませんか! DHCPサーバの構築・運用 中井 悦司 074

Part4 クラウド&スマホを例として考える 「今どき」のIPアドレス管理 中井 悦司 081

Practical Column1 DHCPとDNSの連携 鶴長 鎮一 086

Practical Column2 Amazon VPCのDHCPオプションで設定を変更するには 鶴長 鎮一 091

［一般記事］

ITビジネスの足下を揺るがす大きなバグ OpenSSLの脆弱性“Heartbleed”の教訓【前編】 すずきひろのぶ 094

Web標準技術で行うWebアプリのパフォーマンス改善[3] ブラウザでパフォーマンスを計測する 川田 寛 101

リアルタイム／分析機能／スケーラブルが武器 複雑化するサーバ環境の監視を変える「OpenTSDB」【後編】 sap 110

Rettyのサービス拡大を支えた“たたき上げ”DevOps[3] やっぱり楽しい! トレンドに乗ったインフラ改善 梅田 昌太 118

［巻頭Editorial PR］

Hosting Department[99] H-1

［アラカルト］

ITエンジニア必須の最新用語解説[67] Docker 杉山 貴章 ED-1

「LinuxCon Japan 2014」開催、Linus Torvalds氏が語るコミュニティのあり方 015

読者プレゼントのお知らせ 016

SD BOOK FORUM 109

SD NEWS & PRODUCTS 178

Letters From Readers 182

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）

Contents

Software Design Jul. 2014

3

Column

digital gadget[187]	360度、周囲を取り囲むパノラマ映像	安藤 幸央	001
結城浩の再発見の発想法[14]	Virtual	結城 浩	004
enchant ～創造力を刺激する魔法～[15]	不協和音	清水 亮	006
秋葉原発! はんだづけカフェなう[45]	オシロスコープとロジックアナライザ	坪井 義浩	010
Hack For Japan～エンジニアだからこそできる復興への一歩[31]	Hack For Japan 3.11 ～3年のクロスオーバー振り返り(後編)	鎌田 篤慎	170
温故知新 ITむかしばなし[35]	プリンタ	北山 貴広	176
SDでSF[7]	『ディアスポラ』	小飼 弾	180
ひみつのLinux通信[7]	コマンドヒストリに時を刻め	くつなりょうすけ	181

Development

るびきち流 Emacs超入門[3]	反復練習に勝るものなし ――打鍵すべし! 設定を書くべし!	るびきち	126
シェルスクリプトではじめるAWS入門[4]	AWS利用環境の構築(中編) 請求関連の設定	波田野 裕一	132
Androidエンジニアからの招待状[48]	アプリの脆弱性を気にしてますか? 学習・点検ツール「AnCoLe」	谷口 岳	138

OS/Network

RHELを極める・使いこなすヒント .SPECS[3]	開発フローを考慮したサーバの安定運用	藤田 稜	144
Be familiar with FreeBSD ～チャーリー・ルートからの手紙[9]	仮想ディスクのサイズ調整が便利になったgrowfs(8)	後藤 大地	148
Debian Hot Topics[16]	効率よくリポジトリミラーを構築する方法	やまねひでき	152
レッドハット恵比寿通信[22]	Enjoy Open Source!!	橋本 賢弥	156
Ubuntu Monthly Report[51]	Ubuntu 14.04 LTS日本語Remixを作ってみる	あわしろいくや	158
Linuxカーネル観光ガイド[28]	Linux 3.15の新機能 ～dm-eraとFile private POSIX lock～	青田 直大	162
Monthly News from jus[33]	技術は変わり、インフラは不変になる	法林 浩之	168

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	> Uwe Krejci /gettyimages
Illustration イラスト	> フクモトミホ、髙野 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

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。

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

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

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。

[改訂新版]Linuxエンジニア養成読本
養成読本編集部 編
定価 1,980円+税　ISBN 978-4-7741-6377-2

Webアプリエンジニア養成読本
和田 裕介、石田 絢一(uzulla)、すがわら まさのり、斎藤 祐一郎 著
定価 1,880円+税　ISBN 978-4-7741-6367-3

Androidライブラリ実践活用
菊田 剛 著
定価 2,480円+税　ISBN 978-4-7741-6128-0

おいしいClojure入門
ニコラ・モドリック、安部 重成 著
定価 2,780円+税　ISBN 978-4-7741-5991-1

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

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

独習Linux専科
中井 悦司 著
定価 2,980円+税　ISBN 978-4-7741-5937-9

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

Raspberry Pi[実用]入門
Japanese Raspberry Pi Users Group 著
定価 2,380円+税　ISBN 978-4-7741-5855-6

Linuxシステム[実践]入門
沓名 亮典 著
定価 2,880円+税　ISBN 978-4-7741-5813-6

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

小銅弾のコードなエッセイ
小飼 弾 著
定価 2,080円+税　ISBN 978-4-7741-5664-4

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

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

サウンドプログラミング入門
青木 直史 著
定価 2,980円+税　ISBN 978-4-7741-5522-7

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

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

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

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

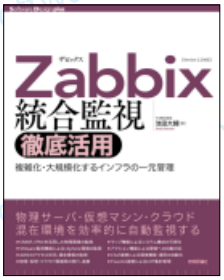
養成読本編集部 編
B5判・128ページ
定価 1,980円(本体)+税
ISBN 978-4-7741-6429-8

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

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

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

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

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、㈱ロンウイット 監修
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

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

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

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

言語のセオリーを
徹底解説

パーフェクトシリーズ

ISBN978-4-7741-6516-5

パーフェクトRuby on Rails 最新刊!

すがわらまさのり、前島真一、近藤宇智朗、橋立友宏　著／B5変形判／432ページ　定価（本体2,880円+税）

Ruby2.0への対応を見据え、基本からビュー、モデル、コントローラの開発、テスト、REST化まで実践的なWebアプリケーション開発手法を徹底解説。

ISBN978-4-7741-5879-2

パーフェクトRuby

Rubyサポーターズ　著／B5変形判／640ページ　定価（本体3,200円+税）

Rubyのセオリーを徹底解説。基本からgemパッケージの作成方法やWebアプリケーション開発まで最新の技術を完全網羅。

ISBN978-4-7741-4437-5

パーフェクトPHP

小川雄大、柄沢聡太郎、橋口誠　著／B5変形判／592ページ　定価（本体3,600円+税）

PHPのセオリーを徹底解説。基本からWebアプリケーション開発、セキュリティまで完全網羅。

ISBN978-4-7741-5539-5

パーフェクトPython

Pythonサポーターズ　著／B5変形判／464ページ　定価（本体3,200円+税）

Pythonのセオリーを徹底解説。基本からネットワーク・デスクトッププログラミングまで最新の技術を完全網羅。

ISBN978-4-7741-4813-7

パーフェクトJavaScript

井上誠一郎、土江拓郎、浜辺将太　著／B5変形判／544ページ　定価（本体3,200円+税）

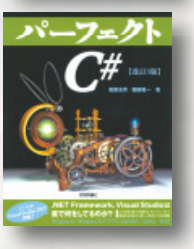
JavaScriptのセオリーを徹底解説。基本からクライアントサイド、サーバサイドまで最新の技術を完全網羅。

ISBN978-4-7741-3990-6

パーフェクトJava

井上誠一郎、永井雅人、松山智大　著／B5変形判／640ページ　定価（本体3,600円+税）

なぜJavaではこう書くのかを徹底解明。言語の基本からサーブレット、データベース、GUIの実践まで完全網羅。



ISBN978-4-7741-5680-4

[改訂3版] パーフェクトC

斎藤友男、醍醐竜一　著／B5変形判／608ページ　定価（本体3,600円+税）

.NET Framework、Visual Studioは裏で何をしてるのか？C#の言語仕様から実践的なプログラミングまで、普遍的な技術から新しい技術まで完全網羅。

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

BOOKS INFOMATION

ISBN978-4-7741-6441-0
A5判・144ページ
定価（本体1,480円+税）

10倍売れるWebコピーライティング

コンバージョン率平均4.92%を稼ぐ
ランディングページのつくり方

●バズ部　著

本書は、単なるコピーライティングの書ではありません。あなたのビジネスに、あらゆる好循環を生み出すための書籍です。インターネット上でモノを売るためのエッセンスを凝縮し、誰にでも実践できるような形にまで、徹底的に落とし込んだものです。そのエッセンスのすべてを、余すところなくあなたにお伝えしています。第2章以降、あなたにも実践していただくためのワークシートがついています。
紙とペンを取り出し、今、売りたい商品を思い浮かべながら実践してみて下さい。

ISBN978-4-7741-6356-7
四六判／232ページ
定価（本体1,580円+税）

なぜ、仕事が予定どおりに終わらないのか？

「時間ない病」の特効薬！タスクシュート時間術

●佐々木正悟　著　●大橋悦夫　監修

締切まで余裕があると思っていたら、いつの間にか時間がなかった。締切ギリギリにならないと、やる気が出ない。集中していたのに話しかけられて、集中力が一瞬で途切れた。めんどうな仕事だから、ついつい先送りに・・・

「こんな仕事、もっと早く片づくと思ったのに！」という、永遠の悩みを解決するにはどうすればいいか？

ライフハックの第一人者大橋悦夫氏が生み出した「タスクシュート」の考え方と実践ノウハウを記した、時間術の決定版！

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

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

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

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

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 187

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

360度、周囲を取り囲むパノラマ映像

パノラマ機材の浸透

皆さんは、パノラマ（PANORAMA）の語源をご存じでしょうか？　PANORAMAは、ギリシア語で「PAN（すべて）」と「HORAMA（眺め）」を組み合わせて作られた言葉だそうです。画家であるBarker Edinburgh氏がパノラマ背景を絵画で描いたことに由来します。

パノラマという言葉が初めて使われたのは1792年にロンドンで開催された大規模な展示会で、左右に広がった風景絵画を円形の建物の高い位置から実際の風景を見下ろすがごとく、没入感のある体験が楽しまれました。それ以降、現代までさまざまな機材と表現でパノラマ映像が記録されてきました。

人間の視野は限られており、カメレオンのように左右別々の方向を見ることは困難です。片方の眼につき上方に60度、下方に75度、鼻側に60度、耳側に100度という広い視野を持ちますが、後方に大きな死角があります。

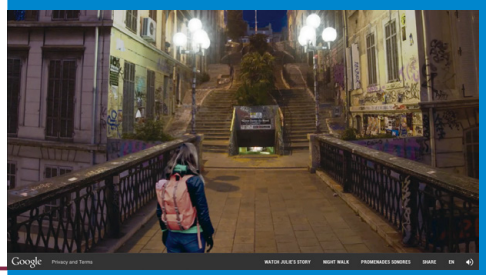
人間よりネコのほうが視野は狭く、さらに大型の肉食動物は前方に集中するため、両眼視野は60度程度と、人と比べるととても狭くなっています。一方、草食動物である馬は、後方視野も広く、単眼で145度程度の視野が確保されています。競馬のときは前方に集中するよう、ブリンカー（遮眼帯）と呼ばれる視野を制限するカップ状のカバーを装着します。鳥類は20～30度の視野しかありませんが、頭の回転範囲が広く、頻繁に動かすことで狭い視野をカバーしています。

パノラマ映像の流行は約20年前にさかのぼります。

1994年にリリースされた「QuickTime VR」がCD-ROMマルチメディアコンテンツなどに活用され、手軽に楽しむことができるようになりました。さらに最近では、普通のコンパクトデジカメやスマートフォンに搭載のカメラでもシャッターを切りながら横に動かすことによって、パノラマ写真が平易に撮影できるようになりました。

加えて、SNSのGoogle+に連続写真を投稿すると、自動的に写真をつなぎ合わせてパノラマ写真にする機能といった画像処理技術の進化やパノラマ専用カメラの登場により、パノラマ写真／パノラマ映像ブームが再燃しています。

さて、パノラマ写真の魅力はどこにあるのでしょうか？　「撮影したときの感覚が蘇（よみがえ）る。没入感があり、その場の雰囲気を思い出すことができる。行っ



» Barker Edinburghのパノラマ絵画（Wikipediaより）

« Google Street Viewを最大限活用した音声ガイドつき観光案内「Google Night Walk」

※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。

たことがないところでも、パノラマ写真からその雰囲気を感じ取ることができる。演出によって撮影済みの視点が決まっているものではなく、自由に見たいところに着目して見られる」などといったことでしょうか。

パノラマサービスと撮影機材

パノラマ写真が一気に一般的になったのは、Google Street Viewによる、世界各地の実用的かつ美しいパノラマ写真の大量の蓄積による効果も大きいでしょう。Googleではさらに、Google Night Walk（https://nightwalk.withgoogle.com/）という音声付き観光案内や、地図と連動したパノラマ写真共有サービスのGoogle Photo Sphereを開始しています。

パノラマ写真が楽しまれるようになったのは、パノラマ撮影専用カメラ、リコー THETAの登場が大きく貢献しています。スペックはISO100～1600、シャッタースピードは1/8000～1/7.5秒、内蔵メモリは約4GBで、約1200枚の撮影（内蔵バッテリーでは約200枚の撮影が可能）。正距円筒図法で補正されたJPEGで、最大3584×1792ピクセル。解像度はまだまだ向上してほしいとの要望が多いですが、子供でも手軽に撮影でき、楽しいシャッター音とともに、1回で全周の撮影が完了します。適切な自動補正が行われ、手軽にネット上に公開、共有できます。撮影機材単独ではなく、スマートフォン用ビューアアプリ、公開Webサイト、Webへの埋め込み、SNS共有など、パノラマ撮影した後の周辺環境も広がっていることが人気の理由でしょう。

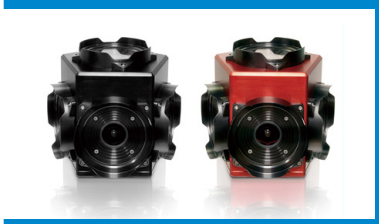
また流行には左右されず、脈々とパノラマ写真を撮り続けているプロ級の方々もいて、各種機材や撮影ノウハウも蓄積されているようです。

コンピュータで制御する専用の雲台と一般的なデジタル一眼カメラを使って200億画素級、超高解像度のパノラマ写真を撮影できるGigapixels（http://360gigapixels.com/）。これはMicrosoftの技術を応用し、スムーズに拡大縮小できるGigapixels ArtZoomサイト（http://gigapixelartzoom.com/）での見せ方が逸品です。

プロ用パノラマ機材としては、SPHERON、LadyBug4などGoogle Street Viewの撮影にも使われたハイエンド製品のほかにも、LOMO-SPINNER、Horizon Perfektといったトイカメラのようなパノラマカメラ、GoProカメラを複数台装着して利用するパノラマカメラなど、パノラマ業界はさまざまな機材にあふれています。

パノラマ映像のこれから

パノラマ映像に関する機材やノウハウが蓄積され、利用者が広がった後は、コンテンツの充実が重要な要素になってきます。走行中のF1レースをレーサー視点でパノラマ視聴できるiOSアプリ「F1 W05 360（http://www.mercedesamgf1.com/en/car/f1-w05-360-video/）、ソリューションを提供する「Making View」（http://makingview.no/?page_id=23）、インテリア向けの「yellowBird」（http://www.yellowbirdsdonthavewingsbuttheyflytomakeyouexperiencea3dreality.com/）など、360度の臨場感、視野が覆われる感



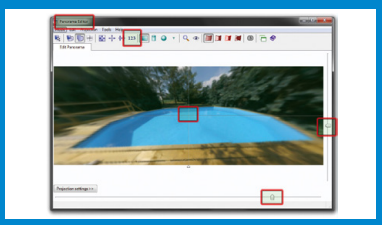
[LadyBug]
5:5Mピクセル超高解像度パノラマカメラ
http://ww2.ptgrey.com/Products/Ladybug5/

[Horizon Perfekt]
ゼンマイ仕掛けのパノラマカメラ
http://microsites.lomography.jp/horizonsjp/

[SpheronVR]
LEDライトつき自動雲台パノラマカメラ
https://www.spheron.com/

[360 Panorama]
簡単パノラマ撮影アプリ
http://occipital.com/360/app

[Cycloramic]
iPhoneが振動して自分で回転するパノラマカメラアプリ
http://www.cycloramic.com/

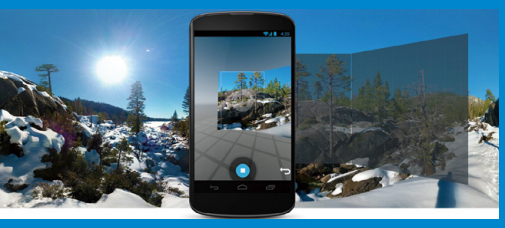


[krpano]
パノラマビューア環境、HTML5やFlash対応
http://krpano.com/

覚、リアルタイム性、没入感、自身による操作感など、普通の写真や動画にない独特の感覚が得られます。

映画の画面サイズがどんどん横方向に拡大してきた歴史からすると、360度、見たいところを見る映画やライブ中継などもこれから実現していくかもしれません。360度の映像の最大の特徴は、あらかじめ決められた演出と視点ではなく、見る人が見たいところを見られるという映像作りにあります。逆に考えると、見てほしいところを見てもらえないという状況もあるということです。

妖怪のように頭の後ろに目がないかぎり、真後ろの風景は見られないのですが、人間は正面の注視しているものよりも、周辺視野によって空間や状況を把握しています。360度撮影のアート作品や、360度＋深さ（距離）方向のデータ取得が可能な安価なレーザースキャナ「RPLIDAR」なども登場し、空間を扱うテクノロジの可能性はますます広がってきています。将来は、映画も演劇もライブも、パノラマ映像で楽しむ世界が待っているかもしれません。SD



[Google Photo Sphere]
Android端末のカメラで撮れる360度写真
https://www.google.co.uk/maps/about/contribute/photosphere/

[Giroptic 360cam]
フルHD、防水仕様、ストリーミングも可能な360度カメラ
http://www.360.tv/

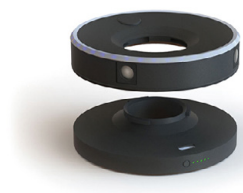
GADGET 1

CENTR Camera

http://www.centrcam.com/

手のひらサイズの360度ビデオ撮影

CENTR Cameraは360度、最大60fpsのビデオ撮影が可能な手のひらサイズのアクションカム。HD解像度のカメラを4台搭載し、6900×1080または4600×720ピクセルで撮影、ソフトウェアによる補正でパノラマ動画が生成されます。手のひらの上に載せたり、中央の穴に指を差し込んだり、GoProのマウントも利用できるそう。撮影コントロールや設定用のiPhoneアプリも予定されています。重量は約250g。2015年発売。予定価格は399ドル。



GADGET 2

BubbleScope

http://bubblepix.com/

スマホケース一体型パノラマ撮影機材

iPhone5/5sのカメラで360度パノラマ写真を撮影する機材。スマートフォンのカメラアプリでもパノラマ撮影可能ですが、BubbleScopeはシャッター一発で360度の写真、または動画が撮影可能。専用のiPhone、Androidビューアアプリあり。BubbleScopeを購入したユーザは専用のパノラマ素材共有サイトを利用することができます。また、BubblePodという360度回転して撮影するスマホ台の販売も予定されています。

GADGET 3

Bublcam

http://bublcam.com/

360度ビデオ配信カメラ

Bublcamは360度撮影し、スマートフォンやパソコンにWi-Fi経由で映像配信可能なカメラです。1080p-15fps、または720p-30fpsで撮影可能。USB経由、microSD経由でも映像が得られます。重量280g、半径4cm、ソフトボール球ぐらいの大きさです。Google Driveやdropbox、Younityでのファイル共有も考えられています。579ドルで2014年6月発売予定。

GADGET 4

360Fly

http://www.360fly.com

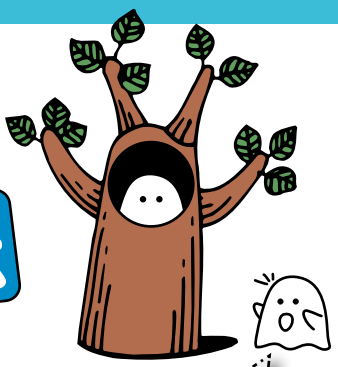
設置容易な360度アクションカム

360Flyはサイズの小さな360度撮影アクションカムです。アクションカムとしてはスタンダードなGoProのように、ヘルメットなどに装着して撮影することができます。1500×1500ピクセル解像度、30fpsの動画が記録できます。撮影範囲は360度パノラマが上下240度、構造上、下方の撮影は不得意ですが、テニスボールほどの大きさで重さは120g、水深5mまでの防水性もあります。iOSとAndroid用にビューアアプリも提供されています。

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

14

結城浩の再発見の発想法



Virtual

Virtual——バーチャル

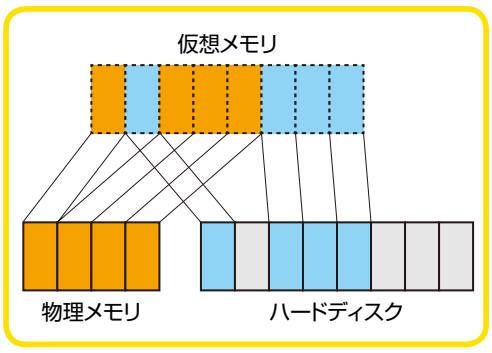
バーチャルとは

バーチャル(virtual)とは、**実際には存在しないのに、現実的な何かをハードウェアやソフトウェアの力を使ってあたかも存在するかのように見せること**の総称です。日本語で言えば「仮想的」です。バーチャルの対義語はリアル(real)で、これは「現実的」ですね。場合によっては対義語がフィジカル(physical)になる場合もあります。こちらは「物理的」です。

バーチャルなものあれこれ

仮想メモリ(virtual memory)は、バーチャルなものの典型例です(**図1**)。仮想メモリは、物理的に存在するメモリよりも大量のメモリがあたかも存在するかのように見せるしくみで、ハードウェアとOSによって実現されています。物理的に存在するよりも多くのメモリが必要になった場合、現在、使われていないメモリの内容をハードディスクなどに一時的に保存し、仮想的に空きメモリを作り出すのです。仮想メモリを使えば、プログラムの側では物理的にメモリが少ないことを気にせず動けます。

▼図1　仮想メモリ



仮想マシン(virtual machine)は、ソフトウェアを使って別のコンピュータがあたかも存在するかのように見せたものです。これはたいへんポピュラーな技術で商品もたくさん出ていますね。1つのコンピュータ上で別のコンピュータを動かせるわけですから、さまざまなバージョンのOSを切り替えられて便利です。

また、物理的なコンピュータが1台でも、仮想マシンを複数台動かして、あたかも複数台のコンピュータが存在するかのように見せることもできます。これはとくにVPS(Virtual Private Server)のように、レンタルサーバのサービスとしても広く用いられています。

前で述べた「仮想マシン」とは意味が変わりますが、そもそも物理的には存在しないコンピュータをソフトウェアとして実現したものもあります(Java VMやLLVMなど)。仮想マシンを使って、物理的なコンピュータと理論的なコンピュータを分離させると、多様なアーキテクチャ上で実行できるプログラムの開発が容易になります。プログラムにとっては、命令コードが実行できさえすればいいからです。

ボーカロイドの初音ミクもバーチャル歌姫と

呼ばれていますね。実際の歌手はいないけれど、ソフトウェアの力であたかもそこに歌手が存在するかのように見せているのです。

バーチャル化とホスト

何かをバーチャル化(仮想化)するには、それを支えるハードウェアやソフトウェアが必要です。ここではそれを**ホスト**と呼びましょう。

仮想メモリでは、ハードウェアやOSがホストです。プログラムが存在しないアドレスにアクセスしたのを受け、アドレスの変換やハードディスクの読み書きを行います。

仮想マシンでは、その仮想マシンを実際に動作させているリアルなコンピュータがホストの役目を果たします。

バーチャル歌姫では、歌を司(つかさど)るソフトウェアが初音ミクの魅力を支えるホストです。

いずれの場合でも、ホストの性能がバーチャル化したものの性能を決めることになります。

リアルとバーチャルの境界線

現代ではリアルとバーチャルの境界線はあいまいになってきています。たとえば、コンピュータの画面にはたくさんのボタンやスイッチが表示されています。あれは画面上の画像であって、リアルなボタンやスイッチがあるわけではありません。あたかも存在するように見え、クリックすると押せるように見え、押すことで何かが起こるように見えるだけです。

でも、存在していて、押すことができて、押せば何かが起きるというのなら、それはもうリアルといってもいいのかもしれません。

文章や音楽、映像などはデータ化されますから、ネットワークを通じて容易に送れます。これまでは物理的な「もの」を瞬間的に送ることは困難でした。

しかし、現代では3Dプリンタがあります。リアルに物質を転送しているわけではないのですが、いわばバーチャルに物質を転送できる時代はもうそこまで来ています。3Dスキャナで取り込んだデータを送信して3Dプリンタで出力すればいいからです。

日常生活とバーチャル化

日常生活でもバーチャル化は行われています。たとえば会社の**役職**などは人間をバーチャル化した存在といえます。社長は、会社の代表者としての役割を担う仮想的な存在です。この場合には実際の人間をホストにして、社長というインターフェースを外に見せていると考えることができます。

社長というバーチャルな存在とリアルな個人を分けることには交換可能性というメリットがあります。たとえば、リアルな個人が病気になっても、別の個人を社長の役割に据えて会社を続けられるでしょう。

また、役職というバーチャルなインターフェースを置くと「兼務」が可能になります。もしも会社での役割がリアルな個人に結びついてしまっていたら、兼務はできません。これはちょうど、1つの物理的なサーバの上で仮想サーバを複数立てるのに似ています。

会社ではよく**担当**を立てます。内容に応じて広報担当やサポート担当などが設置されますね。「プレス関係なら広報担当に、トラブルが起きた場合にはサポート担当にコンタクトを取ってください」という具合です。このような担当も、人間をバーチャル化した存在と言えるでしょう。

背後でどんな人がホストとして仕事しているかはさておき、外部に対してはあたかも1人の人がいるかのように見える。それが担当です。もしかしたら、ホストは複数の人間かもしれません。複数の人間がホストになることで、物理的な個人では発揮できない性能を担当が持つこともできるでしょう。

あなたの周りを見回して、バーチャル化されたものはありますか。そこでのホスト役はいったい何でしょう。バーチャル化することが困難なものはありますか。困難な理由は何でしょう。

ぜひ考えてみてください。SD

enchant ～創造力を刺激する魔法～

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

第15回 不協和音

ついていけません

「清水さんのやり方にはついていけません」

enchantMOONの最初のバージョンの開発を担当した近藤誠はそう言って僕のもとを去りました。

学生時代から数えて5年近く続いた近藤と僕の関係性も、ここで終わりでした。18歳の頃からアルバイトの身分で僕の側近として常に側で働き、ついに大学院を卒業して、正社員として入社してから、わずか1年での退職。

このことは、全社に衝撃を与えました。enchantMOONの出荷がようやく落ち着いたものの、開発陣は休む間もなく毎月のアップデートに追われているころのことです。

僕が寄せる期待と、近藤の努力がうまく噛み合っていないということは開発の途中から感じてはいました。しかし、彼なら必ず乗り越えられると思った壁は、高過ぎ、出口が見えず、私生活が犠牲になっていく日々に、ついに近藤は耐えられなくなってしまったようでした。

もちろん彼には彼の言い分があるのでしょう。そのことについて、とやかく言う権利は僕にはありません。ただひとつ、間違いなく言えるのは、私は私自身のマネジメントの失策で、重要なスタッフを失ったということでした。

しかし出来上がったコードは混迷を極めており、もはや他人が修正するのは不可能に思えるほど複雑化していました。僕自身、コードの最適化のためにベンチマークをとったり、アルゴリズムを最適化したりして近藤に提案を繰り返していましたが、その多くは時間的問題を理由に却下され、ほとんど活かされることはありませんでした。

まったくのゼロから誰も見たこともないアーキテクチャを創りだす、ということは相当な難易度を要求します。僕と長年にわたって数々のプロトタイプや製品開発を行ってきた近藤誠でなければ、enchantMOONの最初の形を作ることはできなかったでしょう。

しかし、ひとたび形が決まってしまえば、あとはベテランが根本から設計しなおす番です。何を作るべきか、どのような形になるのか、明確化されたとき、初めて再設計が可能になります。それは毎月のように行っているアップデートのペースを大幅に落とすことを意味し、同時に収入のあてがないことに開発リソースを費やすことになります。

僕はそれを承知のうえで断行することにしました。どれだけの犠牲を払ってでも、いまここでそれをしておかなければ、このプロジェクトに未来はないと考えたからです。

責任

「君は社長として、どう責任をとるつもりなんだ」

CESから帰った僕を待ち受けていたのは、針のむしろのような取締役会でした。それもそのはず、予定より遥かに多くの注文をいただき、大量に生産し、あえて戦略的な低価格で販売した結果、enchantMOONの事業は極めて利幅の薄いものになっていたからです。加えて度重なる出荷の延期に痺れを切らしたお客様たちが次々とキャンセルを入れていました。そしてそういうことのすべてが、取締役やオブザーバとして参加している投資家を苛立たせているのでした。

「メインの開発者が、辞めたそうじゃないか。部下の管理も満足にできていないのではないのかね？」

「これからどうするつもりなんだ？　まだ続けるのか？　どうやって？」

うるさがたのベンチャーキャピタリストが問い質します。メガネの奥の瞳は冷たい光を放っていました。

「会社の預金残高のすべてを投じてでも、バージョンアップさせる必要があります」

僕は答えました。

「馬鹿げてる。金をドブに捨てるようなものです。そんなことのために投資したのではない」

投資家はかぶりを振りました。

「ソーシャルゲームのためにまだお金を使いますか？」

苦境の原因のもうひとつは、ソーシャルゲーム事業の不振でした。予想よりもフィーチャーフォンのゲームの売上衰退が早く、苦戦を強いられることになっていました。いくつかの受託案件は成功していたものの、その成功はとても自社のソーシャルゲームの失敗を埋め合わせることができるほどのものではありませんでした。

「ソーシャルゲームの世界は、もう完全に変わったんです。あなた方が投資判断をした、3年前とは状況がまったく違う。いまやゲームの世界で闘って行くのに必要な資金は、十億単位です。そんな資金をもはや我々は持っていません。我々が持つ唯一無二の価値に投資しなければ、それこそ投資を受け入れた意味がありません」

「それがenchantMOONだと？　どこがです。ネットの評判を見てください。ボロクソじゃないですか」

「まったく新しいハードウェアに、まったく新しいOSです。そのうえまったく新しいUIです。誰にでもすぐに受け入れられるわけじゃない。これはある程度、予想されたことです」

「ふざけるな！　予想していただと!?　なぜ回避するよう努力しない」

僕の返答に、彼は不快そうに鼻を鳴らしました。

「歴史上のOSを見てください。Windows 1.0を、あなたは使ったことがありますか？　Mac OSの最初のバージョンは？　Newtonはどうです？　iPhoneだって、最初のOSではWebブラウザで2ページほど見ると落ちるなんてことも珍しくなかった。

enchantMOONはまだ、産まれたばかりのものなんです。いまここで投資をやめれば、UEIはその存在意義を失うことになるでしょう。ソーシャルゲームはいまやみんな作っていて、熾烈な競争を闘う必要がある。膨大な資金を投じても、生き残れるかどうかわからない。レッドオーシャンです。しかしenchantMOONは、誰も真似できない、我々だけの技術の固まりです。我々の強みなんです。それを伸ばさずして、今後どんな成長戦略が描けますか？」

会議室は沈黙しました。

「とにかく結果を出せ。さもなくば社長としての責任はとっていただこう」

拒絶

会議室を出た僕を待っていたのは、また別の問題でした。enchant.js、そしてenchantMOONの事業を統括するARCの部長を務める増田哲

郎が、暗い顔で「ちょっといいですか」と僕を捕まえます。

「清水さんともう一緒に働くことはできません、そう伝えてほしいと言われました」

増田は現場からの苦情を代弁しに来たのでした。不満を持っているのは近藤だけではなかったのです。

「僕はどうすればいい？」

「どうにもならないですね。一緒に働くのが嫌だということなので」

「じゃあ放っておけということか？」

「そうなります。それとも……」

増田は一瞬、鋭い目つきで僕を見、そして目を伏せました。最悪の事態を想定しているのだ、と僕は直感しました。

「いまのチームは世界最高のチームのひとつだ。世界のどこに、OSからプログラミング環境まで含めて、これだけの短時間で開発できるチームがあるというんだ。目標とするenchantMOONの実現には一人として欠かせない。それはわかるだろう。……わかったよ。去るべきは僕のほうだな。今後一切、僕は現場に干渉しない」

こうして僕は開発現場を追放されました。

一人の人間がソフトウェアとハードウェアの両方を同時に作ろうとすることの難しさを思い知ることになったのです。結局、どちらかに注力すれば、どちらかが疎(おろそ)かになる。両方同時にやろうとするには、まだ僕は未熟すぎたのです。

フライング・ダッチマン

それから、僕はまず自らの執務室を「フライング・ダッチマン（さまよえるオランダ人）」と改名しました。海賊旗を掲げ、幽霊船の名前を名乗ることにしたのです。僕はenchantMOONのチームから切り離された亡霊となって、社外で活動したり、いろいろなクライアントの案件をこなすことを主な業務としました。現場に直接干渉せず、社外とのやりとりに集中したのです。

社外とやりとりをしていくなかで、一歩引いて外からenchantMOONという製品を見直してみると、確かに巷で言われるように足りない部分、できていない部分は数多くあるものの、いくつかはまばゆく光る要素があることに改めて気付かされました。

そこで僕は新卒の技術者、福本将悟にいくつか試作を繰り返させることにします。物理学専攻で、高い数学的能力を持つ福本は僕のアイデアを次々に実装していきました。また、福本自身も自分の目線で同じ問題に取り組み、彼自身のアイデアを織り交ぜることで仮説をひとつずつ検証し、自信を深めていきました。

いわばenchantMOONの外側から、enchantMOONのあるべき姿を捉えなおすという作業を行ったのです。これはさまざまな貴重な知見を与えてくれました。スウェーデンでの出張講義や、成蹊大学での授業もその1つです。プログラミングを文科系の学生に教える、というテーマもそうですが、それ以上にenchantMOONをごく普通の大学生が使ってみたらどのような結果が得られるか、ということが非常に興味深いテーマでした。

そうして得られた知見を総合した結果、僕は全面的なアーキテクチャの見直しが不可欠であるという結論に達しました。そのためには、まず開発者たちにもその必要性を理解してもらう必要があります。

ですが開発現場への干渉はしない約束です。彼らが自分自身で何が必要なのか気付くまで辛抱強く待つ必要がありました。アップデートがかかるたびに、ああ違う、そうじゃない、と何度も僕は思いました。しかしそのときは、ひたすら、現場を信用し、彼らが気付いてくれるまで待つのが正しい方法だと僕は信じていました。

その間も資金は流出し、時間はどんどん浪費されていきました。果たしてどこまでこの状態を続けることができるか、僕にはわかりませんでした。

迷い

そのとき、僕は迷っていました。自分はいま、本当は何をすべきなのかと。

取締役会からのプレッシャーは、いつものことです。それよりも現場をコントロールできないのに結果に対して責任を持つといういびつな構造をどうすべきかが問題でした。部下を信じて任せる、といえば聞こえはいいですが、結局、何もしていないのと同じです。

現場に嫌われ、一方で取締役会からは結果を求められる。僕はどうすればいいのか、自分で自分がわからなくなってしまいました。そうした迷いは、日に日に強まっていきます。

「部下を信頼して任せるべきか、それとも自分がやりたいようにやるべきか悩んでるって？まったく、何を言ってるんだか」

その日は久しぶりに会った親友の橋本と、新橋のバーで飲んでいました。

「経営者は誰も信頼しない。信用することがあるだけだ」

橋本は数年前、自ら社長を勤めていた会社を清算し、サラリーマンに戻っていました。お互いの会社の黎明期から励ましあい、なんでも相談できる唯一の人間でした。

「信用と信頼は違う。だろ？」

「どういう意味で？」

「信頼は、信じて頼ること。言ってみれば、相手に丸投げして任せきること。信頼していた奴に裏切られたら、こっちはおしまい。ゲームオーバー。けど、信用は、いったんは信じて用いること。ただし、信用していた奴に裏切られても、こっちはそれは織り込み済みで、いくらでもリカバリーできる。それが信用だと。

社員や取引先を信頼してはならない。それではいざというときにどうにもならない。信用して、裏切られることも予想して、いざというときはリカバリーする。それしかないだろ」

橋本はそう言って笑いました。

部下を信頼して任せる、と言えば聞こえはいいですが、それは責任の放棄ともとれます。いったんは信用して使い、だめならば自分でなんとかする。それが経営者としての責任のとり方なのではないか。現場が干渉を嫌ったとしても、彼らが責任をとるわけではありません。結果を求められ、責任をとるのは経営者たる私です。

部下にどれだけ恨まれ、憎まれようとも、僕には自分が正しいと信じることをする義務と責任があります。現場の望むようなやり方で仕事をさせてやりたいのは山々でしたが、放任主義では限界があることはもはや明白でした。すでに現場はアップデートのスケジュールに追われ、大局的なものの見方ができなくなっています。即物的な対応やアップデートに奔走し、本質的にやらなければならないことから目を背けているのです。

ここまでの局面に至っても、なお僕が臆病になっている理由は、近藤誠との確執が尾を引いていたからでした。近藤とはともにさまざまなプロジェクトをやりました。成功した経験も、失敗した経験も、どちらも僕にとって財産です。

しかし、どれだけ愛し、大切に育て上げたとしても、最後の最後で相手に嫌われれば、いとも容易く見捨てられてしまう。部下に切り捨てられるということを、僕は本能的に恐れていたのです。これ以上、手塩にかけた部下を失いたくないという想いが、僕自身を愚鈍なリーダーに仕立て上げていたのです。

だがそれを続けていては、結局、意義ある結果を産むことはできません。僕には答えはわかっていました。何をすれば良いか、何を優先するべきか、ということです。それを強硬すれば、また僕は大切な部下を失うことになるかもしれません。

しかし、現実に僕はその覚悟を迫られているのでした。SD

はんだづけカフェなう

オシロスコープとロジックアナライザ

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/

オシロスコープとは

オシロスコープ[注1]は、電気信号を観測するための装置です。画面の横軸が時間、縦軸が電圧になっていて、時間とともに変化する電圧をグラフにして見ることができます。電圧を測るといえばテスターを連想しますが、テスターではパルス（短時間に急な変化をする信号）を測ることは難しいです。たとえば、前回記事の最後で少しだけ紹介したPWM（パルス幅変調）を使ったとき、意図した信号が出ているかを確認するためにオシロスコープを使って信号を見ます。

昔のオシロスコープは、ブラウン管が表示装置になっていました。測定した結果をブラウン管の内側に塗られている蛍光体を光らせることで表示していたのです。蛍光体の光はすぐには消えませんが、固定して表示しておくことはできませんでした。昔のオシロスコープで記録をするには、画面写真を撮ったりしていました。

近年、主流になっているのは、デジタル・ストレージ・オシロスコープ（DSO）と呼ばれるものです。DSOは、入力された信号をADC（アナログデジタル変換器）を使ってデジタル信号にして用いるタイプのオシロスコープです。ブラウン管に表示するタイプのものもありましたが、今ではほとんどが液晶に表示するタイプになっています。デジタル信号にすることによって、測定したデータをメモリやストレージに保存しておくことができますし、パソコンにデータを転送して処理することもできます。今回はこのDSOを中心に紹介したいと思います。

注1）　オシロと略して呼ばれたりします。

▼写真1　DSO Quad

オシロスコープの性能

オシロスコープと言っても、1万円ちょっとのものから、1千万円以上するものまで幅広い製品があります。もちろん性能が大きく異なるために価格も異なります。ここでは、筆者の手元にある3種類のオシロスコープを例に、その性能の違いについて説明しましょう。筆者の手元にあるのは、2万円ちょっとで買えるSeeed StudioのDSO Quad[注2]（**写真1**）、15万円程度で買えるアジレント・テクノロジーのDSOX 2002A[注3]（**写真2左**）、同社の130万円程度の機種のDSOX4034A[注4]（**写真2右**）です。

注2）　http://www.switch-science.com/catalog/1163/
注3）　http://www.home.agilent.com/ja/pd-1944597-pn-DSOX2002A/
注4）　http://www.home.agilent.com/ja/pc-2198434/

▼写真2　DSOX2000シリーズ（左）、DSOX4000シリーズ（右）

・帯域幅

オシロスコープの最もわかりやすく、重要な性能の違いは帯域幅です。つまり、オシロスコープで測定ができる信号の周波数（Hz）です。たとえば、DSO Quadでは8MHz、DSOX2002Aでは70MHz、DSOX4034Aでは350MHzとカタログに記されています。試しに50MHzの信号を3台でそれぞれ計測してみました（図1～3）。帯域の大きな機種になればなるほど、より信号を正確にとらえていることがわかると思います。

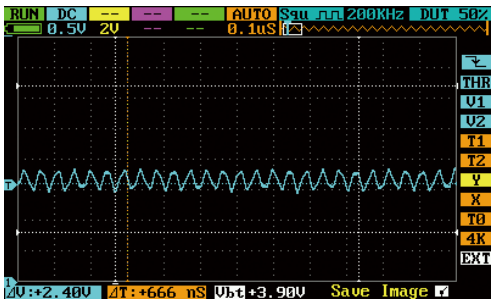
帯域幅70MHzのオシロスコープだからといって、50MHzの信号を正確に測定できるわけでもないことが一目瞭然です。一般的に、測定対象の信号の3～5倍以上の帯域幅のオシロスコープを使用することが推奨されています。つまり、50MHzの信号を計測しようと思えば、150～250MHz以上の帯域幅を持つオシロスコープを使用することが望ましいということになります。

筆者がアジレントのオシロスコープを所有する理由の1つが、この帯域幅に関係します。アジレントのオシロスコープは、あとからライセンスを購入することで、帯域幅を広げることができます。たとえば、筆者の所有している2002Aであれば、帯域幅70MHzから100MHzにアップグレードしたり、帯域幅100MHzから200MHzにアップグレードすることができます。

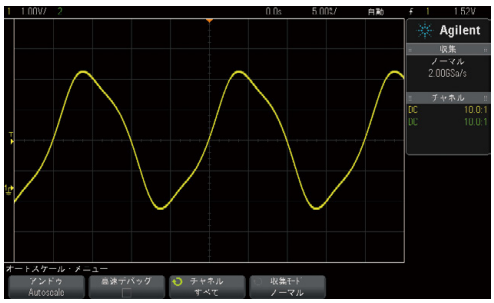
・チャンネル数

チャンネルは、オシロスコープの入力のことです。最も一般的なものは2チャンネルまたは4チャンネル、つまり“2つの信号を同時に入力できる”“4つの信号を同時に入力できる”、などということです。複数の入力を扱うとき、オシ

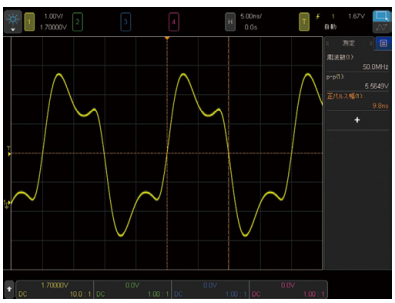
▼図1　DSO Quadによる計測（50MHz）



▼図2　DSOX2002Aによる計測（50MHz）



▼図3　DSOX4034Aによる計測（50MHz）



ロスコープの画面には各チャンネルの信号が色違いで表示されて、同時に見比べることができるようになっています。時間軸(X軸)は各チャンネル共通で、電圧(Y軸)の原点(0V)の場所やスケールを変えて、グラフが重なり合わないようにして表示できます(**図4**)。DSO Quadは4チャンネルの入力がありますが、このシリーズにはDSO Nanoという2チャンネル入力のタイプの製品もあります。

プローブ

オシロスコープの性能を決定する、重要な要素はプローブです(**写真3**)。プローブは、オシロスコープに信号を入力するための要ですから、プローブの品質が測定結果に与える影響はとても大きくなります。プローブにも、オシロスコープと同様に帯域幅があります。オシロスコープを購入すると、たいていは本体の帯域幅に十分な帯域幅を備えたプローブが添付されてくるはずです。ですので、購入時にプローブの帯域幅を気にする必要はあまりありません。

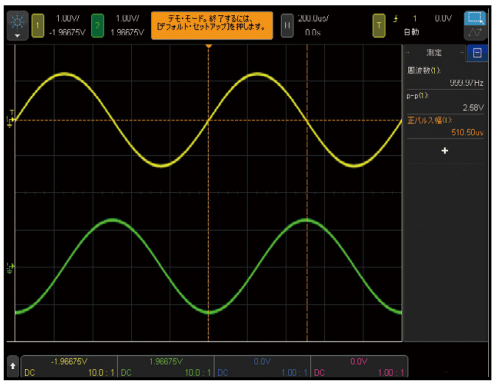
写真3の左上と左下のプローブはDSO Quadに付属、右上はDSOX2002Aに付属のもの、右下はDSOX4034Aに付属のプローブです。それぞれ、本体の性能に応じたプローブが本体の付属品になっています。しかし、本体に付属してくるプローブの品質はメーカーなどによって大きく異なります。筆者自身、3万円程度のオシロスコープと付属のプローブを使って見えなかった信号が、アジレントのプローブを使ってみたところ見えるようになって、原因が追及できたという経験があります。安価なオシロスコープで帯域幅が十分であっても、プローブの性能が原因で見ることができない信号もあります。

また、正確な計測を行うには、計測対象にプローブをうまく当てるコツがいくつかあります。たとえば、先ほどの**写真3**ではプローブのGNDに接続する線が、長い黒い線先についたクリップになっていました。オシロスコープのプローブには、このような線が付属しているのが一般的です。しかし、この線を使った接続では、計測対象の信号によっては正確に信号をとらえることができなくなってしまうことがあります。GNDの線が長過ぎるために生じる問題です。こういった問題を避けるために、**写真4**のように短いGNDの線を用意して計測を行ったりします。

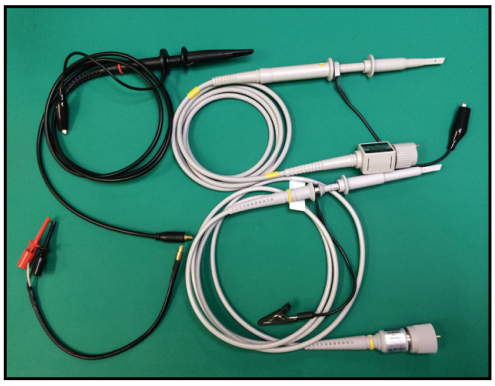
オシロの選定

筆者は当初、DSO Nanoを使っていましたが、もう少し良いオシロスコープが欲しくなりました。そこで、1ランク上の価格帯のものを探すことにしました。その結果、アジレントのDSOX2002Aを選択したのですが、決め手は同価格帯のオシロスコープと比較して画面のサイズが大きいことでした。オシロスコープの有名メーカーとしては、ほかにテクトロニクス、テクシオ・テクノロジー、岩通計測、テレダイ

▼図4　2チャンネル入力をした例



▼写真3　オシロスコープのプローブ



ン・レクロイなどがあります。購入時点では、これらのメーカーから出ていた低価格帯のオシロスコープでは、アジレントの製品が最も画面が大きなモデルでした。

先ほど帯域幅のアップグレードの話をしましたが、アジレントのオシロスコープではチャンネル数のアップグレードは行うことができません。この点をわかっていながらも、つい価格を理由に2チャンネルのモデルを選んでしまいました。その後、やることの範囲が広がり、なぜ4チャンネルのモデルを購入しなかったのかと激しく後悔しました。

ロジックアナライザ

同様によく使われる計測器である、ロジックアナライザを紹介します。オシロスコープは時間とともに電圧が変化する様子をアナログで計測することのできる装置でしたが、ロジックアナライザは「0」または「1」といったデジタルで計測する装置です。オシロスコープでは、「0」や「1」といったデジタル信号となるよりも低いレイヤの問題を見つけることができます。一方で、ロジックアナライザは、高速に通信されているデジタル信号のデータの中身を確認することができます。ロジックアナライザには、この連載でも紹介してきたI²CやSPI、非同期シリアル(UART)といったプロトコルを解析する機能が付いていたり、追加できるようになっています。ロジックアナライザ、というのは長いのでロジアナと略して呼ばれることが一般的です。

筆者が気に入っている低価格帯のロジックアナライザに、SaleaeのLogic16[注5]という製品があります。Logic16は299ドル、約3万円で売られている、かなり安価な製品です。

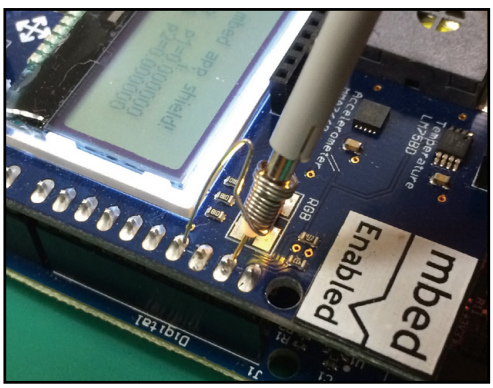
オシロスコープにもUSB接続をして、操作や表示をパソコンで行う製品があるのですが、たいていの低価格帯のロジックアナライザはパソコンで操作と表示を行うように作られています。中でも、Saleaeの製品を使うためのアプリケーションは、Windowsに加えて、OS XとUbuntu用が提供されています。筆者は日頃はOS Xを使っているため、そのまま手軽に使えるSaleaeの製品を使い始めました。

写真5は、I²Cで通信している内容を観測するためにLogic16を接続してみているところです。I²CはSDAとSCLの2本の線を使って通信を行いますので、この2本の線とGND(0V)をロジックアナライザに接続しています。計測を行うと、Logicのクライアントソフトには**図5**のようにI²Cの通信内容が表示されます。I²Cのプロトコル解析機能が標準搭載されているLogicでは、画面中にWriteやRead、ACKやNAKといった具合に注釈が付いたうえで通信内容が数値に戻されて表示されます。

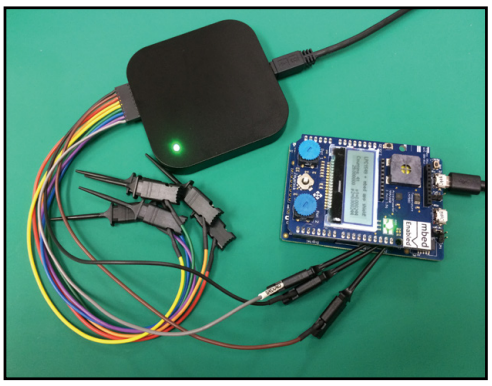
オシロスコープと同様に、ロジックアナライザにも取り込みができる速度やチャンネルと

注5) https://www.saleae.com/logic16

▼写真4　短いGND接続線



▼写真5　Saleae Logic16を使用している例



いった性能のパラメータがあります。Saleaeの Logic16の場合、16チャンネルあるのですが、2チャンネル使用時に100MHzの信号を、4チャンネル使用時には50MHz、8チャンネル使用時に25MHzといった具合に、使用するチャンネル数でサンプルすることのできる信号の速度が変わります。筆者が通常扱っているプロトコルであれば、2～3チャンネルあれば十分なのですが、Logic8というより低価格のモデルよりもLogic16のほうがサンプルできる速度が早いため、Logic16を導入しました。

オシロの解析オプション

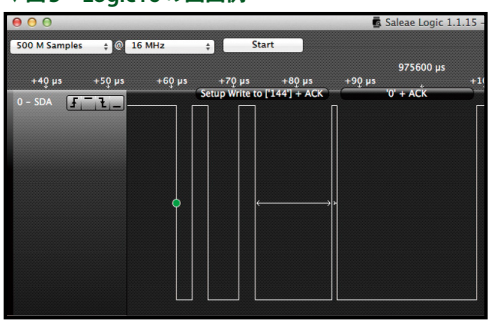
繰り返しになってしまいますが、オシロスコープは0や1ではなく、基本的には信号をアナログ的に見る装置です。ですので、先ほどのI²Cの信号をオシロスコープで観測すると、図6のように信号が見えます。もちろん、オシロがとらえた信号の幅や信号の数を目視で数えて、流れている信号のデータを割り出すことも可能は可能です。実際にそうしている方もいらっしゃるのですが、こらえ性のない筆者には考えられない作業です。

最近のオシロスコープにはこの信号を解析する機能を追加できる機種があります。図7は、解析オプションを追加したオシロスコープでI²Cの信号を解析してみた際の画面です。このようにロジックアナライザのようなプロトコル解析をオシロスコープで行うこともできますが、やはり専用機のほうが使い勝手は良いと筆者は考えています。

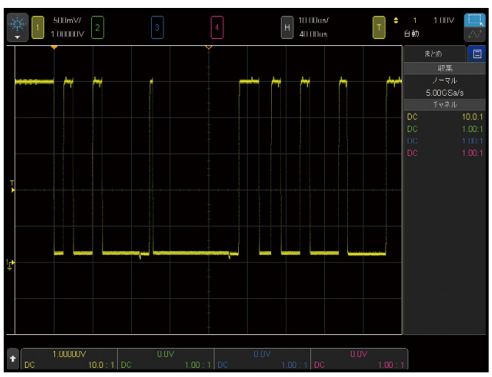
まとめ

今回はオシロスコープとロジックアナライザの世界を簡単に紹介しました。開発しているハードウェアやソフトウェアが意図どおり動かないとき、あるいは意図とおり動いているかを確認するとき、デバッグツールが問題点を見つける助けになります。知り合いの電子部品店の方と話していたところ、安価なオシロスコープのホビーユーザへの売れ行きは非常に良いそうです。数年前と異なり、10万円を切る価格帯のオシロスコープが数多く出てきています。また、ロジックアナライザも低価格で実用になる製品が出てきています。もう少しマイコンでいろいろなことをして遊んでみたい方には、オシロスコープやロジックアナライザを使ってみることをお勧めします。SD

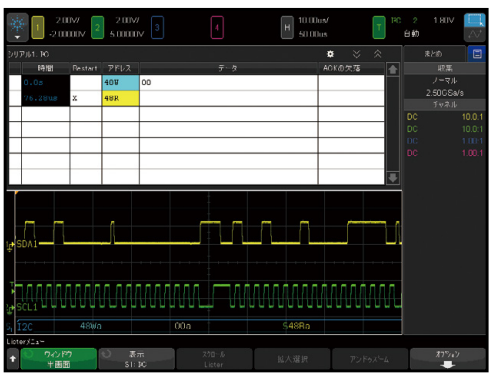
▼図5　Logic16の画面例



▼図6　I²Cの信号をオシロスコープで観測



▼図7　I²Cの信号をオシロスコープで解析



Report

「LinuxCon Japan 2014」開催、Linus Torvalds氏が語るコミュニティのあり方

▲Linus Torvalds氏

5月20日～22日、椿山荘（東京都文京区）において、「LinuxCon Japan 2014」が開催された。本イベントはThe Linux Foundationが主催するもので、毎年、この時期に実施されている。Linuxのほか、同団体がかかわるオープンソースソフトウェア（OSS）についての技術的なセッションや展示が多数設けられている。

Linuxの創始者Linus Torvalds氏も来日し、21日のキーセッションに登壇した。キーセッションは、Intel社のDirk Hohndel氏が司会兼インタビュアー役となり、会場の参加者から質問を募り、それにLinus氏が答えるという、ざっくばらんな雰囲気のもとで行われた。質問は多岐にわたったが、やはりカーネル開発の進め方、コミュニティのあり方について多くの質問が挙がった。

メンテナーの役割

最近のカーネルコミュニティでは、メンテナーの役割に就いている人たちは、マネージャーとしての仕事に追われ、あまり自らパッチを送ることはないという。そんな状況をみて、開発者の1人から「新しく若い開発者を入れるよりも、ベテランのメンテナーたちが自らコードを書き、パッチを送れるようにしたほうがいいのではないか」そんな問題提起がなされた。

それに対し、Linus氏は「今、私が欲しいのは、新しいコードを書く人ではない。パッチを集めて、レビューをして、パッチに対してコメントをしてくれる人だ」と述べた。コミュニティには3,700名もの開発者がいる。その人たちがLinus氏1人にパッチを送っても、プロジェクトは回らない。末端の開発者とLinus氏の中間に位置するメンテナーこそ、プロジェクトのスケーラビリティを確保するうえでは重要、というわけだ。

彼自身は、新しいコードが送られてくるよりも、コードをサブメンテナーからプルして、新たなメンテナーとして間に入ってくれる人が現れたときこそ、わくわくするという。「コミュニティのプロセスがうまく回っている」と実感するそうだ。

リーダーは優しくあるべき？

Linus氏はコミュニティ活動において、気に入らない行動を目にすると、その行動をとった人を口汚く罵ることがある。そんな暴君としての一面に対して、Dirk氏から質問がとんだ。「Linuxの指導者として、もっとみんなに優しくてもいいのでは？」。

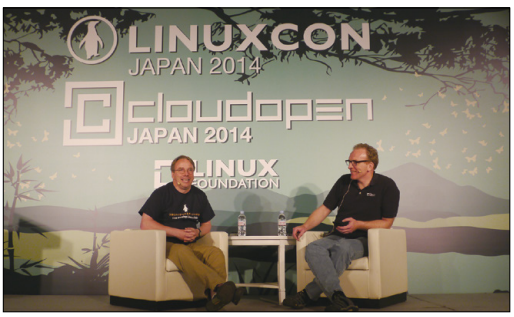
「私たち（コミュニティの幹部）は、十分優しいでしょう。私以外は」。彼はそう冗談交じりに答えながらも、自分が他人に厳しいこともちゃんと自覚している様子。ただ、こうも主張する「コミュニティにはいろんな人がいる。Greg（主要メンテナーのGreg Kroah-Hartman氏）のように優しい人もいれば、私のように皮肉っぽい議論好きな人もいる。人の集まりだから、全員が全員とうまくやれるわけではない。だが、それでOKなんだ」。

Linus氏は確かに乱暴な一面もあり、それを非難されることも多々ある。しかし、本人はそれもLinuxのカリスマとしての、自分の役割と認識しているようだ。それを象徴する言葉として、次のような一言を述べた。「長い間、私はコミュニティでみんなとやりとりをしてきた。だから、みんなは私がどういう人間かを知っている。私が変なことはしないと知っている。私を信頼してくれているのだと思う。だから、私は長い間この座にいるし、乱暴にNoと言うこともできる。中には、それを受け入れられない人もいるかもしれない。しかし、誰かがそういう形でNoと言えることは、コミュニティにとって大事なことだと思う」。

Linus氏の話を間近で聞いていると、批判的なコメントも多く、近寄りがたい印象を受けることもある。だが、同時にコミュニティの長として人々の声を聞こうとする姿勢も垣間みられる。本セッションの最初の彼の一言にもそれが表れている。「私は、人前で話すのは嫌いなんだ。でも、人と話をするのは好き。みんながいったい何を考えているのか知りたいんだ」。

海外イベントの空気を体験できる

同イベントは、海外からの参加者も多く、セッションは基本的に英語で行われる。だが、同時通訳のレシーバも借りられ、登壇者への質問も日本語でできる（通訳してもらえる）。国内にいながら、国際的なOSS活動の空気を肌で感じられる貴重なイベントなので、まだ参加したことのない人は、ぜひ一度、出向いてみて欲しい。



▲Linus Torvalds氏（左）とDirk Hohndel氏（右）

CONTACT The Linux Foundation
URL http://www.linuxfoundation.jp/

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 7 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 2名

iPad Air 対応 QODE Thin Type キーボード

iPad Air 対応キーボードです。iPad Air をスナップセキュアマグネットで固定できます。iPad 本体とは Bluetooth により接続されます。最薄部が 4mm、製品重量が 340g と非常に軽量で、持ち運びに便利な商品です。カバーを開けると自動で電源オンになり、閉じればスリープ状態になるオートウェイク機能付き。色はホワイト（型番：F5L155qeWHT）とグレー（型番：F5L155qeGRY）の 2 色です。各色 1 名様ずつプレゼントします。

提供元 ベルキン **URL** http://www.belkin.com/jp/

02 1名

mouse fit

バイブレータ機能が付いた USB 接続の光学マウス。側面にあるボタンを約 5 秒押し続けるとバイブレーションモードになり、短く一度押すごとに 6 種類の振動パターンを切り替えられます。弧を描く流線型のフォルムが肩や腕・首にフィットし、疲れたあなたを癒します。

提供元 アートファクトリー **URL** http://www.art-factory.ne.jp/

03 1名

ネットワーク HDD ケース MV-NAS25U3

本製品の SATA コネクタに 2.5 インチ HDD/SSD を取り付け、ネットワークに接続することで、ネットワーク HDD（NAS）として使えます（本製品に HDD/SSD は付属しません）。付属の USB ケーブルで PC に直接接続して外付けドライブとしても使用できます。

提供元 アユート **URL** http://www.aiuto-jp.co.jp/

04 2名

パケットキャプチャ入門 第 3 版

竹下 恵 著／
B5 判、448 ページ／
ISBN = 978-4-89797-950-2

オープンソースの LAN アナライザソフト「Wireshark」を使って、パケットを取得する方法を解説しながら、ネットワークのしくみを解き明かします。Wireshark の新バージョン「1.1X」系に対応。

提供元 リックテレコム **URL** http://www.ric.co.jp/telecom/

05 2名

小さな会社の LAN 構築・運用ガイド

橋本 和則 著／
B5 変形判、264 ページ／
ISBN = 978-4-7981-3228-0

社内 LAN 構築・運用手法を徹底解説。Windows 8.1/8/7/Vista を使った、簡単・低コストな社内 LAN の構築をガイドします。Windows ベースで社内 LAN を構築した企業の成功事例も紹介しています。

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

06 2名

Chef 活用ガイド

澤登 亨彦、樋口 大輔 著、クリエーションライン㈱ 監修／
B5 変形判、672 ページ／
ISBN = 978-4-04-891985-2

サーバ管理ツール「Chef」の解説書。公式ガイドブックをもとに、Chef によるシステム運用のノウハウ、Chef の仕様の詳解、用法、用例を加えて再構成しました。

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

07 1名

開発効率を UP する Git 逆引き入門

松下 雅和、船ケ山 慶、平木 聡、土橋 林太郎、三上 丈晴 著／
A5 判、224 ページ／
ISBN = 978-4-86354-146-7

サイバーエージェントで開発に携わっている著者陣が、Git の使い方を速習できるように逆引きという形でわかりやすく解説します。これから Git を使いたいと考えている方にお勧めの 1 冊です。

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

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

最近、利用者が増えてきているNginxは、速くて軽いWebサーバとして注目されています。Apacheから乗り換えを考えている方もいらっしゃるのではないでしょうか。Nginxは高速で多機能で、高負荷にも耐えられると、いいとこどりのように思われがちですが、いざ導入するとなると問題が起こる可能性もあり、そう簡単にはいかない場合も多いようです。

本特集では、ApacheとNginxを比較し、実際に移行するにはどのようなことを考えておかなければならないかを紹介します。また移行後のトラブルシューティングやクラウドでの利用についても解説しています。

第1章 ApacheからNginxへ移行するメリットとは ―― 伊勢 幸一 18

第2章 各Webサーバの比較に見る、Nginxを導入する理由 ―― 佐野 裕 22

第3章 移行前のチェックポイントの洗い出し ―― 長野 雅広 27

第4章 Nginxのインストールとコンフィグ設定の基本 ―― 橘 慎太郎 35

第5章 Nginx引っ越し本番！ ―― 田籠 聡 43

第6章 移行後に気を付けておくべきこと ―― 田籠 聡 52

第7章 クラウドでのNginxの使い方 ―― 大久保 智之 55

第1章 ApacheからNginxへ移行するメリットとは

株式会社データホテル　伊勢 幸一（いせこういち）

本章では、従来のApache環境からNginxへの移行を行う前に、Webサーバの基礎からそれぞれの技術的な特徴について、わかりやすく解説を行います。

Webサーバってなあに?

Webサーバとは

インターネット上におけるさまざまな文書、画像、動画、アプリケーションサービスなどを提供するサーバシステムを「Webサーバ」といいます。もともとはHTTPという通信プロトコルにより、HTML形式の情報リソースを配信するサーバとして「HTTPサーバ」と呼ばれていました。しかし、ほとんどのHTTPサーバはFTPやSMTPなどHTTP以外のプロトコルも処理し、また、プレーンテキストやPDF、XMLといったHTML以外の多様なデータも配信するので、現在ではそれらをひっくるめてWebサーバと呼ぶのが一般的のようです。

情報を配信するWebサーバに対し、その情報を受け取って利用する側をWebクライアントといいます。もっとも代表的なクライアントアプリは、PCやスマートフォンなどにインストールされているWebブラウザでしょう。しかし、汎用的なWebブラウザでなくともLINEやGoogle Mapsのような専用スマートフォンアプリも、Webサーバと通信して何らかの情報をやりとりしている限り、これらもWebクライアントの範疇に入ります。誤解を恐れずにインターネット利用者の立場でざっくり言うと、インターネットはこれらWebサーバとWebクライアント間のデータ通信によって成り立っています。

Webサーバの歴史

Webサーバは、1990年代前半にインターネット上に出現しました。Netcraft社のサーバシェア調査では1996年4月以後ずっと世界第1位の地位を守って来たのは1995年にリリースされた「Apache HTTP Server（以降、Apache）」です。同じころMicrosoft社がWindows NTのオプションとしてIIS（Internet Information Server）[注1]というWebサーバをリリースし、その直後からIISがApacheに次ぐ世界第2位のWebサーバになりました。以降、ApacheとIISはWebサーバシェアの1位と2位を独占し、現在に至ります。

ApacheとIIS以外のWebサーバでは、2007年から第3位にランキングしたGoogle（Googleサービスの基幹で運用されているWebサーバ）、iPlanet（現Oracle iPlanet Web Server、旧称Sun Java System Web Server）、Zeus（現Riverbed Technology）、lighttpdなどがシェアランキングに登場しています。そして本特集の主役である「Nginx」が2009年のレポートからランキング入りし、翌年の2010年12月にはGoogleを追い抜いて第3位となりました。

注1）Windows 2000から標準ツールとなりInternet Information Serviceと改名されました。

Webサーバの役割

Webサーバはインターネット上にさまざまな情報を発信すると述べましたが、その処理内容によっていくつかの役割に分けることができます。

①データ配信

もっとも古典的なWebサーバの役割であり、HTMLテキストファイルや画像、音声データなどの静的ファイルをディスクなどのストレージから読み出し、HTTPプロトコルによってWebクライアントに返信する役割です。先に述べたようにWebサーバによってはFTPなどのプロトコルをサポートし、HTTP以外のプロトコルで配信する実装もあります。

②アプリケーション実行

Webクライアントから送られてくるFORM入力を解釈し、必要な処理を実行して動的にHTMLドキュメントや画像を生成する役割です。もともとWebサーバとは別にCGI(Common Gateway Interface)プログラムという独立した外部プログラムを起動することによって行っていましたが、現在ではメモリの節約、処理負荷の軽減や高速化を図るため、Webサーバプログラム内にPHPやPerlなどのインタプリタモジュールを内蔵し、Webサーバが直接アプリケーションを実行する形式が多くとられています。

③プロキシ処理

家庭内LANや組織内LANと、インターネット間の通信を中継する役割です。利用者から見てインターネット上の情報を閲覧する際に中継することをフォワードプロキシ、インターネット上への配信を中継することをリバースプロキシといいます。

プロキシサーバは単にHTTPセッションを中継するだけではなく、一度中継したデータをキャッシュし、同じデータに対するリクエストには、リクエストを中継するのではなくそのキャッシュを返すという役割も担います。また、リバースプロキシとして機能する場合、リクエストによって適当なサーバを選択し、処理を分散するというロードバランサの役割も兼ねます。

詳しくは、本誌2014年3月号の第2特集「プロキシサーバの教科書」を参照してください。

なんてったってApache！

1996年から現在に至るまでWebサーバシェア第1位を維持しているApacheは、1995年4月にその前身であるNCSA HTTPD1.3のパッチバージョンとしてベータ版がパブリックリリースされました。同じ年の12月にバージョン1.0がリリースされました。もともとApache Groupと呼ばれるボランティアによってメンテナンスされてきましたが、1999年にApache Groupのメンバ達がApache Software Foundationを設立し、以後この組織がApacheのメンテナンス、リリース、ライセンシングを司ることになります。2000年末には2.0のアルファバージョンがアナウンスされ、その後2.1、2.2、2.3と続き、本稿執筆時点の最新バージョンは2.4.9です。

1.0から1.2までにさまざまな機能追加や改善が施されてきましたが、1.x系は1.3を以(も)ってファイナルバージョンとなっています。1.3ではUnix以外にもMicrosoft社のWindowsもプラットフォームとしてサポートしたことが大きなアップデートです。そして1.3から2.0へのバージョンアップにおいて大きく追加／改善された機能は、マルチスレッド化とマルチプラットフォーム化です。しかし、LAMP(Linux + Apache + MySQL + Perl/PHP)構成という単語があるようにApacheが最も多く利用されているプラットフォームはLinuxであり、ApacheとLinuxのコンビネーションは今なおWebサーバプラットフォームの主流となっています。

Nginxとは

Nginxはロシアの Rambler という検索サイトのニーズをもとに、同じロシアの Igor Sysoev

氏によって2002年から開発が始まり、2004年に最初のバージョンが公開されました。その後、Igor氏は2011年にNGINX INCを設立し、Nginxのメンテナンスと開発、商用サービスなどを展開しています。Nginxが急速にシェアを伸ばしてきた理由には、ほかのWebサーバと比べて軽く、並行性に優れ、高速であるからとされています。

Nginxはなぜ速くて軽いの?

Apacheとの比較

書籍やネット上において、NginxとApacheのアーキテクチャや機能、性能比などについてさまざまな比較評価がなされています。性能比についてはハードウェアや設定パラメータ、ベンチマークプログラムによってさまざまな値が得られますが、デフォルトの設定であればNginxの方がApacheに比べ約1.5倍～2倍ほど処理性能が高いようです。一般的に言われているNginxとApacheとの違いを**表1**に示します。

これは感覚的な比較であり、それぞれハードウェアやパラメータ設定、もしくは外部プログラム、プラグインなどの利用によってどちらとも言えない項目もあります。ただし、定性的な違いとして挙げられるのはタスク処理と並列化、I/O処理方式です。メモリ消費量、同時処理数、処理速度の違いはこのアーキテクチャの違いによる影響が大きく、一般的にそれは「非同期処理とイベントドリブン採用」と表現されますが、それはいったいどのような方式なのかを初心者向けに説明した資料はそれほど多くはないでしょう。具体的なNginxの方式やアルゴリズムは第2章以降で詳しく説明していきます。

▼表1　ApacheとNginxとの比較

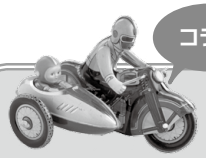
	Apache	Nginx
開発言語	C, C++	C
データ形式	動的ドキュメントに向いている	静的ドキュメントに向いている
タスク処理	マルチプロセス	シングルプロセス
並列化	マルチプロセス	イベントドリブン
I/O処理	同期	非同期
消費メモリ量	大きい	小さい
同時処理数	やや多い	非常に多い
処理速度	まぁまぁ	速い

移行、導入のメリットとは

Nginxは単一のプロセススレッドでイベントドリブンによるノンブロッキング処理(次ページコラム参照)をすることで非常に高速な処理を可能にしています。ノンブロッキング処理によりプログラムの制御がイベントハンドラに戻ったとしても、実際のデータ読み書きはOS(カーネル)内のシステムコールプログラムとハードウェアとの間で実行しているため、その処理が長過ぎると結局システムコールキューにリクエストが溜まり過ぎ、パフォーマンスが低下する恐れがあります。

したがってNginxは非常に小さいデータを大量に配信するサーバや、ハードディスクへの読み書きが発生しないメモリ内キャッシュサーバ、リバースプロキシサーバ、フロントエンドロードバランサなどの役割に適しています。逆に非常に複雑なCGI処理や動画データなどの配信、データベース処理などを実行するには不向きなプログラムです。

つまりWebサーバシステムのすべてをNginxに置き換えるとシステムの性能が向上するという単純な話ではなく、従来Apacheで行っていた前述のような処理に限定して差し替えるといった検討と工夫が必要でしょう。ただし、冒頭で述べたようにここ数年のサーバシェアの伸び具合は驚くべき状況であり、システムの有効な部分に導入するメリットは多くのエンジニアに認められています。本特集を参考にしてぜひNginxを活用してください。SD



ノンブロッキングと非同期

➡ブロッキングI/Oとノンブロッキング I/O

オペレーティングシステムが提供するサービスを呼び出す関数をシステムコールといい、システムコールには「ブロッキング」と「ノンブロッキング」があります。ブロッキングとはプログラムがシステムコールを呼び出してから結果が返されるまで次の処理に移れないコールのことです。ファイルの読み書きを行う通常のread、writeというI/Oシステムコールはブロッキングコールです。

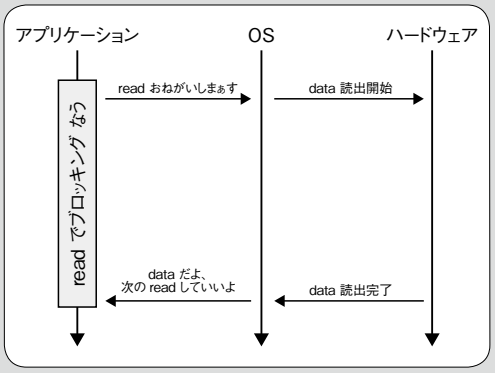
逆にノンブロッキングコールとは、呼び出した直後に制御がプログラムに戻され、システムコールの終了を待たずに次の処理に移れるコールです。対象ファイル記述子にfcntl()システムコールでO_NONBLOCKフラグがセットされている場合の読み書きはノンブロッキングになります。ただし、プログラムはsignalやselect、pollなどを使ってシステムコールの終了を捕捉する必要があります。

➡同期I/Oと非同期I/O

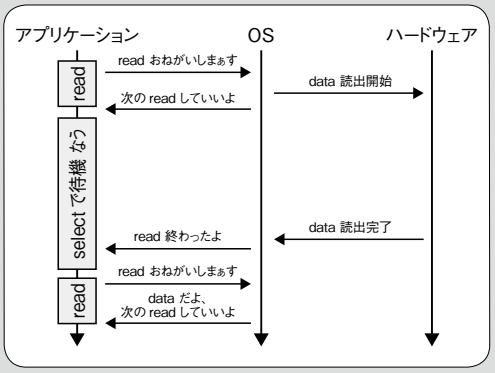
同期、非同期とはread、writeといったI/Oを伴うシステムコールにおいて、システムコールの実行が可能な状態を待ってリクエストするか、直前のシステムコールの終了を待ってリクエストするかの違いになります。

実際にはブロッキングの有無、そして同期非同期の組み合わせによってI/O方式が実装されるため、全部で4種類のI/Oパターンが考えられます。次にreadシステムコールを例としてその4種類の処理チャートを**図1〜4**に示します。 これらのチャートからわかるようにノンブロッキング方式では、アプリケーション側でシステムコールの終了を待つ必要がなく、リクエストを発行した直後から別の処理が実行でき、効率の良い同時並行処理が可能になります。

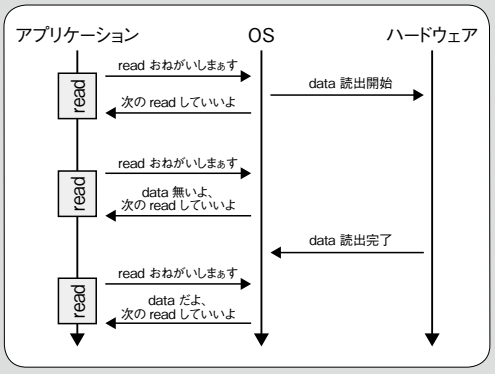
▼図1　ブロッキング同期I/O



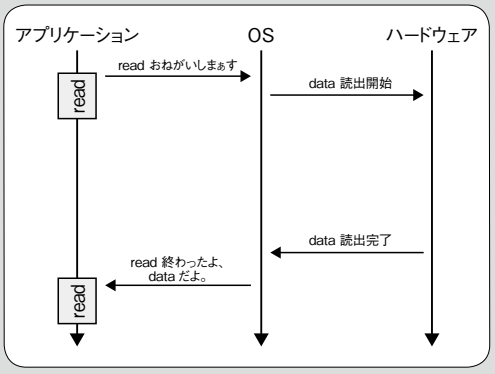
▼図2　ブロッキング非同期I/O



▼図3　ノンブロッキング同期I/O



▼図4　ノンブロッキング非同期I/O



第2章 各Webサーバの比較に見る、Nginxを導入する理由

LINE株式会社　佐野 裕(さの ゆたか)

Webサーバソフトウェアに期待されることとして、安定性、高速性、設定の容易性といったことが挙げられるかと思いますが、Nginxはいずれの要素も十分満たすソフトウェアです。そこで本章ではそれらの要素をふまえたうえでNginxの導入理由を考えてみたいと思います。

はじめに

Nginxはロシア人であるIgor Sysoev氏によって作られたWebサーバソフトウェアです。Webサーバ機能以外にもリバースプロキシやキャッシュサーバなどの機能も搭載されています。最近流行しているNginxですので比較的新しいソフトウェアかと思われる方も多いと思いますが、実際は2004年に公開された10年以上の歴史を持つオープンソースです。

オープンソースはまわりの人が使っていると普及しだすという傾向があるようで、Nginxについても「よくわからないけれども皆使っているので使ってみようか」という流れで利用者が増えてきているように思います。

Nginxの特徴

まずはNginxの特徴について見ていきましょう。

C10K対応

NginxはC10K(クライアント1万)の同時接続処理に対応していることを売り文句としています。従来のWebサーバではせいぜい数千クライアントまでの処理しか対応できなかったのに対して、Nginxでは適切に設定すれば1万クライアントの処理が行えます。1万クライアントの処理が行えると謳っていても、「動作が不安定な中でなんとか1万クライアントの処理ができる」ということでは意味がありませんが、Nginxは多くの大規模サイトにて、クライアント数が非常に多い環境の中で用いられてきているため、安定性や高速性については十分実証されていると言えます。

メモリ使用量が少ない

Nginxはイベント駆動方式のアーキテクチャです。Apacheなどでも採用されているプロセスベース方式のアーキテクチャでは、同時接続数が増えれば増えるほど大量の物理メモリが消費されますが、イベント駆動方式ではプロセスベース方式と比べてプロセス数が少ないために物理メモリ使用量が少なくて済みます。

豊富な対応OS

Nginxは、Apacheほど幅広くはないですが、一般的に用いられるさまざまなOSに対応しています。Nginxの公式サイト(http://nginx.org/)を見ると、Linux、FreeBSD、Solaris、Windows、Mac OS Xなどの上で動作することが明記されています。公式サイトのダウンロードページに行くと、ソースコードはもちろんのこと、Ret Hat Enterprise Linux、CentOS、Debian、Ubuntuについてはパッケージでのインストール方法に

ついての記載があります。

モジュールベース

Nginxには複数の機能がモジュール単位で開発されていて、Nginxのコンパイル時に必要なモジュールを組み込むことができます。一般的にWebサーバで必要とされるようなモジュールは一通りそろっています。組み込めるモジュールは、http://wiki.nginx.org/ModulesJa から参照できます。

Webサーバ以外の機能

NginxにはWebサーバ機能以外にも、ロードバランス、キャッシュなどの機能があります。

Nginxの長所

Webサーバソフトウェアを導入する際、安定性、高速性、そして設定の容易性は重要な選定要素となる場合が多いです。Nginxではこれらをどう見れば良いでしょうか。

安定性

動作の安定性を考えるうえで、多くのWebサイトで十分な稼働実績があるというのは安定性を示す1つのわかりやすい要素です。この意味ではNginxは稼働実績が十分と言えるでしょう。

もうちょっと踏み込んでみると、ソフトウェアがどのようなアーキテクチャ(構造)をしていて、そのアーキテクチャに致命的な欠陥がないか検証する方法や、いくつかのテストシナリオを作って動作テストしてみる方法などがあります。安定性を100%証明することは難しいことですが、しくみを理解し実際に試してみるというのは良い方法です。Nginxのアーキテクチャについては後述していますので参考にしてください。

高速性

Nginxはハードウェアリソースを効率的に使うように設計されています。具体的には次のような工夫がされています。

ワーカープロセスモデルの採用

マルチプロセス構成とは、各ワーカープロセスがクライアントからのリクエストを受け付け、レスポンスを返す構成のことです。

マルチプロセス環境においてはプロセスごとにメモリが確保され、かつ各CPUコアがそれぞれのプロセスを処理します。理屈上Nginxではワーカープロセスの数をCPUコア数以下にするとCPUが並列に稼働します。

シングルスレッドの採用

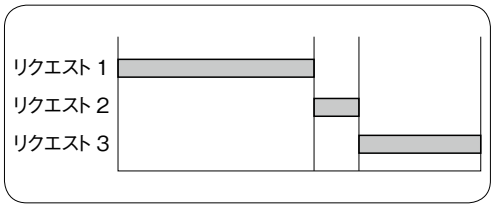
複数のワーカープロセスがシングルスレッドで稼働することにより、コンテキストスイッチと呼ばれるCPUの切り替え処理が発生しないようにしています。

通常マルチスレッド環境では、一定時間が過ぎるとCPUの処理対象が切り替わるコンテキストスイッチが発生するため処理が遅くなります。しかしシングルスレッド構成であれば切り替えが発生する対象がそもそも存在しないため、Nginxに起因するコンテキストスイッチが発生しません。

非同期処理

多くのWebサーバソフトウェアにおいては同期処理構成が採用されています(**図1**)。同期処理構成とは、複数のリスエストがあったときに1つ1つのリクエストを順番に処理していきます。1つのリクエストに応答時間がかかるようであれば、その後のリクエストも引きずられて応答速度が遅くなります。

▼図1　同期の例



それに対して非同期処理構成では同期を取らないため、リクエストが来たらほかのリクエストの処理状況にかかわらず直ちにリクエストを処理します(**図2**)。応答に時間がかかるリクエストがほかにあったとしても引きずられることがありません。

Nginxの設定の容易性

Nginxの設定ファイルは初期状態で31行(空行含む)しかありません(**リスト1**)。

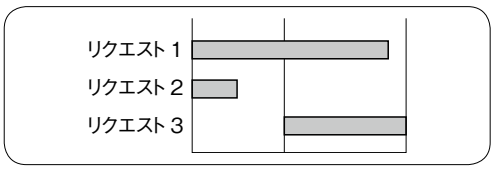
静的コンテンツ配信を行いたい場合は、設定をnginx.conf内もしくは別のファイルに記述してnginx.confの中で読み込むようにします。たとえば**リスト2**をhttpディレクティブ内[注1]に書き込んでからNginxを起動させるだけでWebサーバとして稼動します。この例は非常にシンプルですが、もちろんもっと細かい設定を行っていくこともできます。

注1) 第4章で解説しています。

Nginxのアーキテクチャ

Nginxを深く理解するために、Webサーバで用いられるクライアント処理方法の比較をしながら、Nginxで採用されているイベント駆動方式と呼ばれるクライアント処理方法について紹介します。

▼図2 非同期の例



▼リスト1 nginx.confの初期状態

```
user  nginx;
worker_processes  1;

error_log  /var/log/nginx/error.log warn;
pid        /var/run/nginx.pid;


events {
    worker_connections  1024;
}


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

    log_format  main  '$remote_addr - $remote_user [$time_local] "$request" '
                      '$status $body_bytes_sent "$http_referer" '
                      '"$http_user_agent" "$http_x_forwarded_for"';

    access_log  /var/log/nginx/access.log  main;

    sendfile        on;
    #tcp_nopush     on;

    keepalive_timeout  65;

    #gzip  on;

    include /etc/nginx/conf.d/*.conf;
}
```

▼リスト2 静的コンテンツ配信の設定

```
http {
    ⋮
    ⋮
    server {
        location / { root /var/www/html; }
        location ~ /¥. { deny all; }
    }
}
```

プロセスベース方式

プロセスベース方式ではプロセスがリクエストを処理します(**図3**)。すなわち同時接続が1,000であれば1,000個のプロセスが必要となります。この方式は実装が最も簡単ですが、その代わりプロセスを生成することはメモリを喰うため、大量の物理メモリが必要な方法と言えます。たとえば1プロセスあたり100MBのメモリを用いる環境では、同時接続が100程度であっても100MB × 100 = 10GBもの物理メモリが必要となります。Apacheのpreforkはプロセスベース方式です。

スレッド駆動方式

スレッド駆動方式ではスレッドがリクエストを処理します(**図4**)。スレッドはプロセスの中で作られます。1プロセスの中で複数スレッドを動作させることができます。スレッドはプロセス内で共有メモリ領域を使う方式であるため、メモリ容量を節約しながら同時接続数を増やすことができます。Apacheのworkerはスレッドベース方式です[注2]。

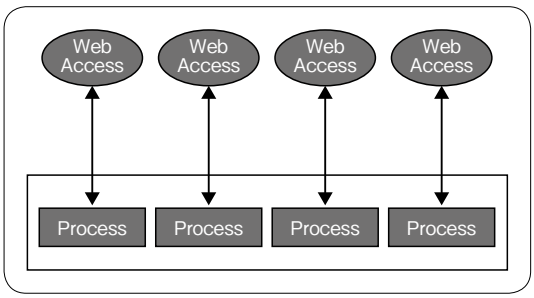
イベント駆動方式（マスタプロセスがクライアント対応）

マスタプロセスがクライアント対応するイベント駆動方式は、マスタプロセスと呼ばれる1つの親プロセスがクライアント側のWebアクセスを受け付け、実際のイベント処理はワーカープロセスと呼ばれる複数の子プロセスで処理を行う方式です(**図5**)。

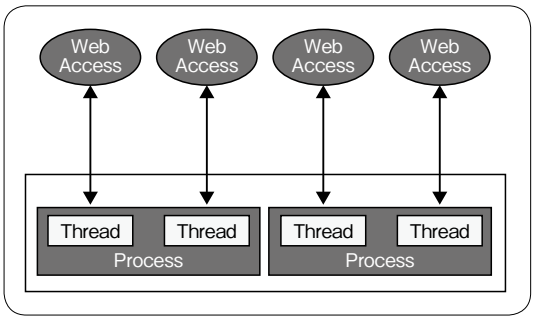
イベント駆動方式（ワーカープロセスがクライアント対応）

ワーカープロセスがクライアント対応するイベント駆動方式は、各ワーカープロセスでイベントを受け取り、イベントを処理する方式です(**図6**)。これはNginxで採用されている方式です。

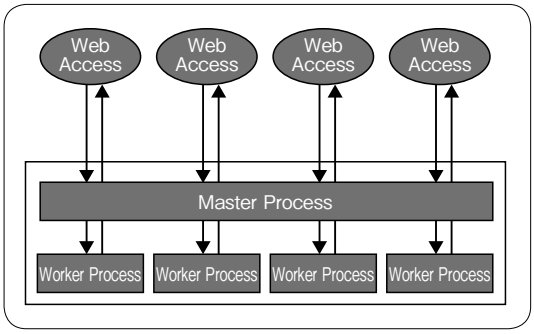
▼図3　プロセスベース方式



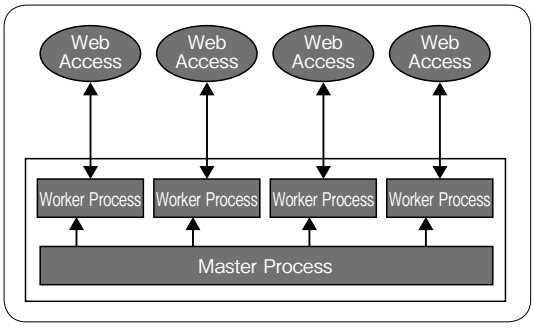
▼図4　スレッド駆動方式



▼図5　イベント駆動方式（マスタプロセスがクライアント対応）



▼図6　イベント駆動方式（ワーカープロセスがクライアント対応）



注2）正確にはマルチプロセス＋マルチスレッドのハイブリッド型です。

各Webサーバの比較

表1は各Webサーバの比較となります。この表の中のメリット部分にだけ着目すると、イベント駆動方式(ワーカープロセスがクライアント対応)において「最も省メモリ、CPUコアをフルに使いこなすとかなり高速」となっているので一番良いのではないかと思われるかもしれません。

しかし、先ほど、Webサーバソフトウェアを導入する際、安定性、高速性、そして設定の容易性は重要な選定要素であると記しました。デメリット部分にも着目すると「実装が非常に難しい」と書いてあります。これは言い換えると開発の難易度が上がるためにバグが混在しやすいだとか、何か問題が起きたときに問題の特定が難しいということを意味します。その点、プロセスベース方式は「シンプルで実装が簡単」と書いてあるように、何か問題が起きても問題の特定が比較的容易といえます。

Nginxはオープンソースですので、何か問題が起きたときに自分たちでソースコードを分析して問題部分の特定を行うことができます。そのレベルまで自分たちで行おうと考えているのであれば実装の難易度まで考慮して、自分達が理解できるレベルの実装であるか、ソースコードを実際に見てみて判断してみるのも良いかもしれません。

終わりに

アクセス数があまり多くない一般的なサイトにおいては、実はどのWebサーバソフトウェアを使っても体感上ほとんど差がないと言えます。しかしとくに大規模アクセスをさばくような環境においてNginxは絶大な効果があります。

筆者の持論として、「自分が慣れているものが一番うまく使いこなせる」というものがあります。どんなに良いものであっても使いこなせなければ宝の持ち腐れになりますし、新しいものに慣れるまでどうしても時間がかかります。ですから、普段十分使いこなしているWebサーバソフトウェアがあるのであれば、それを使い続けたほうが合理的という場面は多いと思われます。

しかしNginxは、今後間違いなく主流になっていくと思われるため、「今のうちにNginxに慣れておこう」という考え方もあれば、「今、とくに困ってないからそのうちでいいや」という考え方もあるかと思います。でも、せっかくの機会ですので、今回Nginxを試してみませんか？ SD

▼表1　各Webサーバの比較

方式	処理主体	メリット	デメリット	対応例
プロセスベース方式	プロセス	シンプルで実装が簡単	大量の物理メモリが必要、遅い	Apache (Prefork)
スレッド駆動方式	プロセスとスレッドのハイブリッド	省メモリ、設定次第でそこそこ高速	イベント駆動方式より性能限界が先に来る	Apache (Worker)
イベント駆動方式(マスタプロセスがクライアント対応)	マスタプロセスがクライアント受付し、ワーカープロセスが応答	マスタプロセス部分の実装が比較的簡単	マスタプロセスがボトルネックになる	lighttpd
イベント駆動方式(ワーカープロセスがクライアント対応)	ワーカープロセスがクライアント受付と応答	最も省メモリ、CPUコアをフルに使いこなすとかなり高速	実装が非常に難しい	Nginx

第3章 移行前のチェックポイントの洗い出し

LINE株式会社　長野 雅広（ながの まさひろ）

前章まででApacheからNginxへ移行する理由、Nginxの特徴が整理できました。本章では、実際にApacheからNginxへと移行を考えるにあたって確認すべきことがらをまとめてみます。なお、ここで紹介するApache HTTP Server（以下Apache）は2.2系のバージョン、OSはCentOSを想定しています。

ApacheからNginxへ移行する目的を整理しよう

Nginxは多くの場合、Apacheより高速に動作し、メモリ使用量やCPUの使用率などOS／ハードウェアにかかる負荷も小さいというメリットがあります。これだけを聞くと、「よし、こんなにメリットがあるならすぐにでもNginxに移行しよう！」と思うかもしれません。

しかし、ApacheとNginxは互換性があるソフトウェアではありませんので、「インストールして移行完了！」というわけにはなりません。すでに稼動しているApacheがある場合、その設定を確認して、Nginxでも同じように動作するようシステムを構成し、設定ファイルを書かなければなりません。また、Apacheの豊富な機能やモジュール群を、Nginxはすべてサポートしているわけではありませんので、利用している機能によってはApacheを活用したり、Apache、Nginx以外の別の手段を用意する必要が出てきます。

このようにApacheからNginxへの移行は簡単にはいきません。無理矢理移行しようとしても設定のミスや機能の不足などで正しくサービスを提供することができなくなる恐れがあります。ApacheよりNginxのほうが速そうだから、負荷が小さそうだからという理由だけで移行を決めるのは危険です。多くの場合、Apacheでも十分に安全に、安定したサービスの提供ができます。筆者の管理するサーバでもApacheが動作しているものが数多くあります。

筆者の経験から、ApacheからNginxへ移行するメリットがあると考えるのは次の2つのパターンです。

①大規模なWebサービスで、大量のクライアントからの接続を高速に捌（さば）きたい
②限られたリソースでサービスを動作させる必要があり、なるべく負荷の小さいWebサーバにしたい

①も②もNginxの数多くの接続を扱えたり、OS／ハードウェアにかかる負荷が小さいというメリットを最大限活かせます。このような場合ではApacheの設定を確認し、Nginxの設定を作成するコストよりも移行するメリットが大きくなるでしょう。もし、あなたの管理するサーバでApacheが動作していて、Nginxにしてみようかなと思ったら、売り文句に踊らされるのではなく、移行するメリットや目的を整理してみることをお勧めします。

NginxとApacheを平行運用する方法

ApacheからNginxへすべての機能を移行できない場合、サーバ内でNginxとApacheの両者を動作させ、平行運用することが考えられます。移行する際に考慮したいチェックポイントを紹介する前に、平行運用の方法を簡単にまとめます。

TCPポートを変更しての運用

最初の方法が、NginxとApacheでbind(2)[注1]するTCPポートを変更することです(**図1**)。一般ユーザからのアクセスはTCPポートを80番から変えるのは難しいですが、限られた人だけがアクセスする管理画面やツールは比較的自由にTCPポートを変更できるはずです。たとえばNginxをTCPポート80番で動作させ、Apacheを8080番で動かし、両者に直接指定してアクセスをします。

IPアドレスを変更しての運用

TCPポートを変更できない場合、サーバに複数のIPアドレスを持たせ、bind(2)するIPアドレスを変更する方法が考えられます(**図2**)。Apacheが必要なサイトとそうでないサイトに分けて、それぞれ別のIPアドレスに名前解決されるDNSホスト名を付加します。

Nginxをリバースプロキシとした運用

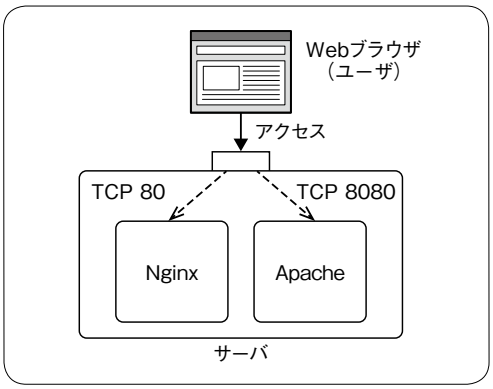
システム構成は「TCPポートを変更して運用」に似ていますが、直接Apacheにアクセスするのではなく、Nginxへリクエストを行い、必要に応じてNginxからApacheへリクエストがプロキシされる方式です(**図3**)。ユーザからのアクセスを受け、大量のコネクションを扱ったり、単純な静的ファイルはNginxで行い、複雑なアプリケーション処理はApacheで行うという両者の特徴が活かせる構成です。

移行する際に考慮したい設定

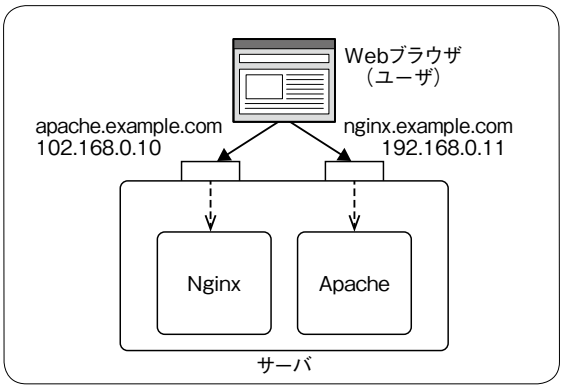
さて、ここからはApacheからNginxへ移行するために、どんなことをチェックしたらいいのかを紹介します。ここで紹介するチェックポイントのほとんどはApacheに機能があり、Nginxにその機能がないパターンとなります。Nginxでサポートされていない機能を使っている場合、何かしらの対策を行わないとApacheからNginxへの移行はできません。

注1) 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。

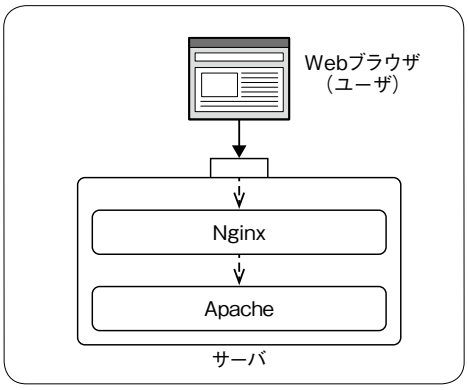
▼図1　TCPポートを変更した運用



▼図2　IPアドレスを変更した運用



▼図3　Nginxをリバースプロキシとした運用



Nginxでサポートされていない機能を使っていないか確認するには、Apacheの設定ファイルに目を通す方法以外にはありません[注2]。

フォワードプロキシとしての利用

フォワードプロキシという言葉はあまり聞き慣れないかもしれません。一般的には「プロキシサーバ」と言われることが多いでしょう。プロキシサーバには2種類のサーバがあり、**図4**の①のようにブラウザから「どこどこのページを持ってきて」とリクエストを受け取るのがフォワードプロキシ、**図4**の②のようにブラウザからリクエストを受け取り、実際にリクエストを処理するサーバに代理で送るのがリバースプロキシと呼ばれます。Apacheに**リスト1**のような設定があれば、フォワードプロキシが動作しています。

Nginxではどうかいうと、プロキシ機能はありますが、フォワードプロキシの機能はありません。Nginxのプロキシ機能はリバースプロキシとしての動作に特化しています。

もし、ApacheからNginxに移行を考えていて、対象のサーバでフォワードプロキシを行っている場合、NginxとApacheを異なるTCPポートで動作させ、継続してApacheでフォワードプロキシを提供する必要があります。

.htaccessによる設定変更

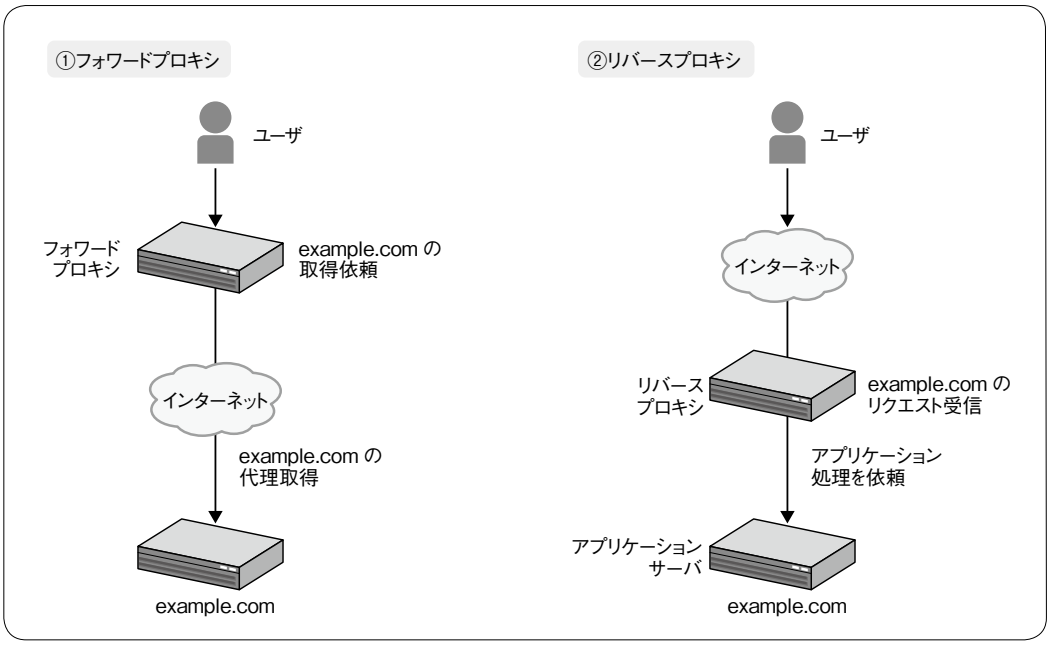
Apacheでは、配信するコンテンツを設置するディレクトリに「.htaccess」というファイルを設置することでWebサーバの動作の一部を変更できます。「.htaccess」を使用するためにはApacheの設定で、AllowOverrideディレクティブの設定が必要となりますが、一度「.htaccess」を許可すれば、root権限を持ったWebサーバの管理者でなくても、コンテンツの管理者が「.htaccess」

▼リスト1　フォワードプロキシの設定例

```
<VirtualHost *:8080>
  ServerName proxy
  ProxyRequests On
  <Proxy *>
    Order deny,allow
    Allow from 127.0.0.1 192.168.0.0/24
  </Proxy>
</VirtualHost>
```

注2）CentOSの場合、Apacheの設定ファイルは/etc/httpd/conf と /etc/httpd/conf.d/以下にあります。

▼図4　プロキシサーバの種類



を設置したディレクトリ以下に限り、Webサーバの動作を変更できます。

リスト2はAllowOverrideであるディレクトリ以下に「.htaccess」の設置を許可している例で、**リスト3**は、「.htaccess」でディレクトリへのアクセスがあった場合にファイル一覧ページの表示を有効にするサンプルです。

サーバ内に「.htaccess」が多くあり、さまざまな設定が行われていると、Nginxへの移行はやっかいです。Nginxでは「.htaccess」のような設定変更の手段は提供していません。すべての設定をNginxの設定ファイル内に記さなければなりません。もしApacheの設定にAllowOverrideがあり、「.htaccess」ファイルを無効にするNone以外が設定されている場合、公開ディレクトリ以下に「.htaccess」ファイルがないか調査する必要があります。findコマンドでは次のようにすることで「.htaccess」ファイルを探すことができます。

```
find /path/to/public_html -name '.htaccess'
```

「.htaccess」ファイルが見つかった場合は、その記述内容をNginxの設定に移行できるか確認する必要があります。もし、root権限を持ったサーバ管理者以外が「.htaccess」を日常的に変更しており、都度サーバ管理者に変更を依頼して対応するのが難しい場合は、その部分だけApacheを継続して利用するなどの対応が考えられます。

動的サイトの構築

CGI

CGI(Common Gateway Interface)は、クライアントからのリクエストに応じてサーバ側でプログラムを起動し、その結果をクライアントに返す動的サイト構築のためのしくみです。リクエストごとにプログラムの読み込みと起動が必要となるので、サーバにかかる負荷が大きくなってしまいがちですが、古くからあり、簡単に動的サイトを作ることができるので今も利用されています。

CGIを使っている場合、Apacheでは拡張子とハンドラ(処理方法)を結び付ける設定、

```
AddHandler cgi-script .cgi
```

と公開ディレクトリに、

```
Options ExecCGI
```

という設定があります。前述の「.htaccess」で有効にされている場合もあります。

動的サイトの構築のためにCGIはしばしば利用されていますが、Nginxではサポートされていません。CGIをデーモンプログラムとして動作させ、専用のプロトコルで通信を行うFastCGIやSCGIをサポートしているのみで、プログラムを都度起動するCGIはサポートされていません。もちろんCGIプログラムをデーモン化し、起動コストをなくしたほうがサーバにかかる負荷が削減でき、レスポンスも高速になりますが、CGIプログラムを常駐させるための設定が必要となり、また既存のプログラムがそのまま動作するとも限りません。

ApacheからNginxに移行する際に、対象サーバにてCGIが動作していた場合、Apacheを利用し続ける必要があるかもしれません。

▼リスト2　.htaccessを有効化する例

```
<Directory /path/to/public_html>
  AllowOverride All
</Directory>
```

▼リスト3　.htaccessでファイルの一覧の表示を有効にする

```
Options +Indexes
```

SSI

SSI(Server Side Include)も、またCGIと並んで古くからある動的サイトを構築するための技術です。CGIがプログラムを起動してレスポンス全体を出力するのに対して、SSIはHTMLの中に専用のタグを埋め込んで、Webサーバにて置換を行います。SSIは、

```
<!--#コマンド オプション -->
```

のようなフォーマットで記述され、いくつかのコマンドが用意されています。たとえば、**リスト4**のようなHTMLがあった場合、「<!--」から「-->」までの間をWebサーバが処理し、header.htmlファイルの中身と差し替えます。

SSIのコマンドの中にはApacheでサポートされて、Nginxでサポートされていないコマンドがあります。その1つが外部コマンドを起動して、その出力に置き換えるexecコマンドです。以下は現在の時間を出力するSSIです。

```
<!--#exec cmd="/bin/date" -->
```

execのほかにはfsizeやflastmodといったコマンドがNginxではサポートされていないようです。

SSIは古くからある技術で最近ではあまり使われることがありませんが、もしNginxに移行する中で動作しないSSIが出てきた場合、SSIを処理するためにApacheを残すか、もしくはサーバサイドではなくクライアント側のJavaScriptで代替できないかコンテンツの管理者と相談するのが良いでしょう

mod_php、mod_ruby、mod_perl、mod_wsgi

CGIがWebサーバから外部プログラムを起動するのに対して、mod_php、mod_ruby、mod_perl、mod_wsgiはプログラムを動かすためのインタプリタをWebサーバに組み込んで、Webサーバ内でプログラムを動作させることで高速化を狙ったApacheのモジュールです。それぞれPHP、Ruby、Perl、Pythonを動かすためのモジュールとなっています。

Nginxではこうしたモジュールは、一部を除いてサポートされていません。Apacheでこうしたモジュールを使っていた場合、Nginxに移行することはできないので、Apacheを残すか、別の手段を考慮する必要があるでしょう[注3]。

▼リスト4　SSIの例

```
<html>
<body>
<!--#include file="header.html" -->
</body>
</html>
```

認証・認可

Webページの一部に、パスワードをやアクセス元のIPアドレスを使って閲覧の制限をかけることはよく行われています。Apacheでいくつかの認証／認可モジュールが提供されています。NginxでもWebページの閲覧を制限する機能が提供されていますが、Apacheほど多様な機能はサポートされていません。

リスト5はApacheで特定のディレクトリに対して、アクセスの制限をかける例になります。ApacheでBasic認証を使っている場合このような設定がなされていると思われます。**リスト5**の設定でWebページに対してアクセスが許可がされるのは、192.168.0.0/24およびローカルホストからのアクセスと、Basic認証で認証されたユーザとなります。許可するユーザはAuthUserFileで指定した「.htpasswd」ファイルにて管理されています。「.htpasswd」はhtpasswdコマンドで生成します。

```
$ htpasswd -c /path/to/htpasswd sduser
New password: パスワード入力
Re-type new password: パスワード入力
Adding password for user sduser
```

注3） PHPについては第4章でPHP-FPMを使う方法が紹介されています。

▼リスト5　ApacheでのBasic認証の設定例

```
<Directory /path/to/secrets>
  Order Allow,Deny
  Allow from 192.168.0.0/24 127.0.0.1
  Deny from All
  AuthType Basic
  AuthName "Secret Page"
  AuthUserFile /etc/to/.htpasswd
  Require valid-user
  Satisfy Any
</Directory>
```

-cオプションは、ファイルを生成するオプションで付けるのは初回だけです。「.htpasswd」ファイルを覗くと**リスト6**のように、

```
ユーザID:暗号化・ハッシュ化されたパスワード
```

というフォーマットでユーザ名とパスワードが記録されています。

Basic認証はNginxでもサポートされています。Nginxで同じ認証の設定を行ったのが**リスト7**です。NginxではApacheのhtpasswdコマンドで作られたユーザ管理ファイルをサポートしているので、ApacheからNginxに移行する際に、変換の必要なくそのまま利用できます。

Digest認証

前述したBasic認証では、パスワードがBase64でエンコードのみ行われた状態でクライアント／サーバ間でやりとりされるので、通信の盗聴や改ざんの恐れがあります。そこで考案されたのがDigest認証です。Digest認証ではユーザ名／パスワードをランダムな文字列とともにハッシュ化して送信するので、通信内容の盗聴があったとしても元のパスワードを知ることは難しく、改ざんもされにくいという特徴があります。

ApacheはDigest認証をサポートしています。ApacheでDigestを使っている場合、**リスト8**のような設定がなされています。AuthUserFileファイルで指定される「.digest_pw」ファイルはBasic認証で使うファイルとは異なります。Digest認証のユーザ管理ファイルはhtdigestコマンドを使って作られます。

Nginxは標準ではDigest認証をサポートしていませんが、サードパーティのnginx-http-auth-digestモジュール[注4]を組み込むことでDigest認証が実現できます。**リスト9**はNginxにDigest認証を設定した例となります。auth_digest_user_fileで指定されるユーザ管理ファイルはApacheのhtdigestコマンドで生成されたファイルをサポートしているようです。

後述しますが、Nginxの標準にないサードパーティモジュールを組み込む場合、Nginx自体の再ビルドが必要となるので運用が難しくなりがちです。ApacheでDigest認証を使っている場合、Nginxにサードパーティのモジュールを組み込んでNginxをビルドするほかには、認証が必要な部分のみApacheを使うか、Digest認証を止め、HTTPSとBasic認証を使用することなどが考えられます。

ホスト名での認証

リスト5のApacheの設定では、アクセス元のIPアドレスを使った閲覧制限が行われていますが、このApacheではIPアドレスの部分にドメ

注4）https://github.com/samizdatco/Nginx-http-auth-digest

▼リスト6　.htpasswdファイルのサンプル

```
sduser:$apr1$YWF92N3w$k/GF5PK54qh1NkFAiceGN/
gihyo:$apr1$1aQwjW8k$1IyIZ8404yoaMOijUEOq5.
kazeburo:$apr1$ADIPQRap$jpdEhmnfM9jhkDM88oJ9T.
```

▼リスト7　Nginxでの Basic認証設定例

```
location /sercret_html {
    root /path/to/secret_html
    allow 192.168.0.0/24;
    allow 127.0.0.1;
    deny all;
    auth_basic           "Secret Page";
    auth_basic_user_file /etc/to/.htpasswd;
    satisfy any;
}
```

▼リスト8　Apacheでの Digest認証設定例

```
<Location /secret_diges/>
  AuthType Digest
  AuthName "Secret Pages"
  AuthDigestDomain /secret_diges/
  AuthUserFile /etc/to/.digest_pw
  Require valid-user
</Location>
```

▼リスト9　Nginxでの Digest認証設定例

```
auth_digest_user_file /etc/to/.digest_pw
location /private{
  auth_digest 'Secret Pages'
}
```

イン名を書くことができます。

```
Allow from gihyo.jp
```

この設定があった場合、ApacheはHostname Lookupsの設定にかかわらずアクセス元のIPアドレスからホスト名の逆引きを行い、ホスト名が「*.gihyo.jp」もしくは「gihyo.jp」であった場合にアクセスを許可します。

Nginxではこの機能をサポートしていないので、IPアドレスでのアクセス制限を利用するか、別の手段を検討する必要があります。

外部のDBを使った認証

Basic認証、Digest認証にて、ファイルを使ったユーザ管理を紹介しましたが、Apacheはユーザ管理の方法としてファイル以外に、LDAP、DBMファイル、RDBMSなどの外部DBをサポートしています。これらの機能を使っている場合、Apacheの設定ファイルに、

```
AuthBasicProvider ldap
```

のように書かれています。利用している外部DBによって ldap の部分は dbm や dbd となっています。

外部DBを使用すると、たとえばApacheからMySQLサーバに接続して任意のSQLを発行してアクセス許可を行えるなど、ほかのシステムと連携した柔軟なユーザ管理が実現できます。しかし、Nginxではこのような認証のしくみは用意されていません。ファイルを使ったユーザ管理のみがサポートされています。

Apacheで外部のDBを使った認証を行っている場合、それをNginxのみで実現することはできません。認証が必要な部分だけApacheを利用するか、Nginxに含まれるngx_http_auth_request_module[注5]モジュールを使って、認証を外部のサーバに問い合わせる方法が考えられます。

注5）http://nginx.org/en/docs/http/ngx_http_auth_request_module.html

mod_rewriteによる黒魔術

Apacheでmod_rewriteを多数活用している場合は、ApacheからNginxへ移行した際にNginxの設定が複雑になることがあるので注意が必要です。mod_rewriteを使っている場合、設定ファイルに「RewriteEngine」「RewriteCond」「RewriteRule」などが出てきます。

mod_rewriteはApacheの黒魔術とも言われ、強力なカスタマイズ機能を持っています。**リスト10**はmod_rewriteの比較的簡単な例です。リクエストメソッドが「GET」または「HEAD」で、X-Forwared-HTTPSヘッダが大文字小文字無視の「on」ではない場合、httpsのサイトにリダイレクトするという設定になります。

同じ設定をNginxで実現したのが**リスト11**です。Nginxにはifというディレクティブがあり、条件によって処理の変更ができます。このifディレクティブは一般的なプログラミング言語のように見えますが、複数の条件を書くことができず、またifブロックをネストすることもできません。ですので、リクエストメソッドとリクエストヘッダの2つの条件が満たされたことを確認するために$redirecthttpsという変数を用意し、リクエストメソッドがGETまたはHEAD

▼リスト10　mod_rewriteでの条件付きhttpsへのリダイレクトの設定例

```
RewriteEngine On
RewriteCond %{REQUEST_METHOD} ^(GET|HEAD)$
RewriteCond %{HTTP:X-Forwarded-HTTPS} !^on$ [NC]
RewriteRule ^(.+)$ https://gihyo.jp/$1 [R]
```

▼リスト11　Nginxでの条件付きhttpsへのリダイレクトの設定例

```
if ($request_method ~ '^(GET|HEAD)$' ) {
  set $redirecthttps "tr";
}
if ( $http_x_forwarded_https !~* 'on' ) {
  set $redirecthttps "${redirecthttps}ue";
}
if ( $redirecthttps = "true" ) {
  rewrite /(.+)$ https://gihyo.p/$1 redirect;
}
```

だったときに"tr"を格納し、X-Forwared-HTTPSヘッダがonだった場合に、"ue"を後ろに連結し、最後に$redirecthttpsが"true"だったらhttpsにリダイレクトを行います。

このようにmod_rewriteでは簡潔に書くことができる設定が、Nginxでは遠回りをしたかのような書き方をしなくてはなりません。Apacheでmod_rewriteを使っている場合、Nginxの設定を作成していく前にテストケースを作り、テストドリブンで移植作業を行うのをお勧めします。

動的なモジュール追加

ApacheもNginxも標準にない機能は、サードパーティのモジュールを組み込むことで追加できますが、両者のサードパーティモジュールの組み込み方は異なります。そのため運用の方法なども変わってきます。Apacheは本体に対して動的にモジュールが組み込めるようになっています。拙作のmod_copy_headerモジュール[注6]を組み込む場合、

```
$ sudo apxs -c -i mod_copyheader.c
```

としてモジュールをビルド／インストールします。Apacheの設定ファイルに**リスト12**の設定が自動的に追加されていることを確認して、Apacheを再起動すると、追加したモジュールが使えるようになります。

Nginxはこのようなモジュールの動的読み込み機能をサポートしていないので、コンパイル時にモジュールを組み込まなければなりません。**図5**はNginxのproxy処理を拡張するモジュールの組み込み例です。configureの実行時に--add-moduleオプションで組み込むモジュールを指定します。

Nginxを独自にビルドして運用を行っている場合、再ビルドは比較的容易ですが、rpmなどのパッケージを使っている場合、再ビルドのためにはパッケージのソースコードをダウンロードし、パッケージのビルド方法を示したスペックファイルにパッチをあて、独自にビルドをしなくてはなりません。独自にビルドを行ってしまうと、パッケージ配布元からのアップデートパッケージの適用やサポートが受けられなくなってしまう可能性があります。

Nginxへサードパーティモジュールを組み込む場合は、バージョンアップやセキュリティの問題が出たときに、アップデートやパッチ適用の運用をどのように行うかのポリシーをあらかじめ決めておく必要があるでしょう。

まとめ

本章では、ApacheからNginxへの移行前チェックポイントの洗い出しとして、ApacheとNginxとで互換性のない機能を紹介し、ApacheとNginxの平行運用を含め、どのように対応したら良いかを書いてきました。ApacheとNginxの特徴を把握し、最適なシステム構成を考える際の資料となれば幸いです。SD

注6）https://github.com/kazeburo/mod_copy_header

▼リスト12　Apacheでのモジュールの読み込み設定

```
LoadModule copyheader_module   modules/mod_copyheader.so
```

▼図5　Nginxのビルド例

```
tar zxf ngx_http_upstream_consistent_hash.tar.gz
tar zxf nginx-1.6.0.tar.gz
cd nginx-1.6.0
$ ./configure --with-http_stub_status_module ¥
    --add-module="../ngx_http_upstream_consistent_hash"
$ make
```

第4章 Nginxのインストールとコンフィグ設定の基本

株式会社データホテル　橘 慎太郎(たちばな しんたろう)

本章では、実際にNginxのインストール方法と、コンフィグの記述例を交えた設定方法を紹介します。Nginxのインストールから Webサーバの構築、リバースプロキシサーバの構築の順に解説していきます。Webサーバ構築の節では、php-fpmを利用したPHP環境の構築方法も紹介します。

Nginxのインストール

本節では、Nginxのインストール方法を紹介します。Nginxのインストールにはいくつかの方法がありますが、今回は公式リポジトリ注1を利用したインストール方法を紹介します。

インストールに必要なもの

インストールに必要なものは次のとおりです。

- CentOS 5/6系がインストールされているサーバ
- サーバにアクセスするためのPC
- SSHを利用するための端末ソフトウェア(「TeraTerm」「Putty」「Poderosa」など)
- Webブラウザ

また、viやEmacsなどのエディタも使用します。今回はOSとしてCentOS 6.5を利用しました。

注1) Linuxの各ディストリビューションやソフトウェアの提供元などが、そのディストリビューションで利用できるソフトウェアを取得するための場所として、バージョン管理を行い、一般に公開しているサーバです。各ディストリビューションごとに標準で登録されているリポジトリは安定して利用できる反面、最新版が利用できないこともあるため、最新版を利用したい場合はほかのリポジトリを探すか、自分でソースファイルを取得してインストールする必要があります。

インストールの流れ

さっそくNginxをインストールをしてみましょう。手順は次のとおりです。

(1)Nginx公式リポジトリの追加
(2)Nginxをインストール
(3)インストール後の動作チェック

Nginx公式リポジトリの追加

最初にSSHなどでサーバにログインし、rootに昇格します。

```
$ su -
Password: 管理者パスワードを入力
```

次にNginx公式リポジトリを追加するために、「/etc/yum.repo.d/」配下に、viで「nginx.repo」を作成します。ファイルの中には次の内容を記述します。

```
[nginx]
name=nginx repo
baseurl=http://nginx.org/packages/ ➡
centos/$releasever/$basearch/
gpgcheck=0
enabled=1
```

Nginxをインストール

CentOSのアップデートを行ったあとにNginxをインストールします(**図1**)。執筆段階での

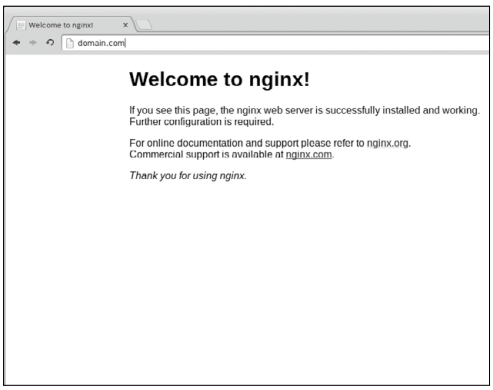
Nginxのバージョンは「1.6.0-1.el6」です。

インストール後の動作チェック

Nginxが正常に動作しているか確認をします。まず、念のためにpsコマンドでNginxプロセスが立ち上がっているかを確認します。**図2**の下線部分のように「master process」と「worker process」が表示されていればOKです。

次に、Webブラウザでサーバのドメインかアドレスでアクセスして、Nginxのテストページが表示されることを確認します(**図3**)。

▼図3　Nginxテストページ



▼図1　Nginxのインストール

```
# yum update      ←CentOSのアップデート
# yum -y install nginx      ←Nginxのインストール
 ...(略)...
Dependencies Resolved

================================================================================
 Package     Arch                 Version          Repository       Size
================================================================================
Installing:
 nginx       x86_64       1.6.0-1.el6.ngx             nginx      335 k

Transaction Summary
================================================================================
Install     1 Package(s)
 ...(略)...
Installed:
  nginx.x86_64 0:1.6.0-1.el6.ngx

Complete!
```

▼図2　Nginxプロセスの確認

```
# ps aux|grep nginx
root     10975  0.0  0.0 107464   944 pts/1    S+   01:09   0:00 grep nginx
root     13089  0.0  0.0  45068  1320 ?        Ss   Apr27   0:00 nginx: master process /usr/ ↵
sbin/nginx -c /etc/nginx/nginx.conf
nginx    13090  0.0  0.0  45496  2056 ?        S    Apr27   0:00 nginx: worker process
```

Webサーバとして Nginxを設定する

Nginxの起動が確認できたら、次にNginxの設定を行います。はじめに今回の設定例を紹介し、後半で簡単に代表的なディレクティブ、コンテキストを解説します。また、最後におまけとしてNginxでPHPを動作させるための設定を紹介します。

Nginxの設定ファイル「nginx.conf」

Nginxの核となる設定は「nginx.conf」内に記述されています。以降で行う設定のほとんどはnginx.conf内に記述することでNginxに反映されます。nginx.confは「/etc/nginx/」配下に設置されています。

nginx.confの内容は**リスト1**のとおりです(各設定の簡単な説明をコメントで追記しています)。

設定作業

何はともあれ、設定をしてみましょう。設定の流れは次のとおりです。今回はバーチャルホスト[注2]を使ってWebサーバを構築してみます。

①sites-availableディレクトリを作成して、配下にvirtual.confを作成する
②nginx.confを編集して、virtual.confファイルを読み込ませる
③設定ファイルが正しく記述されていることを確認し、Nginxを再起動して設定を有効にする
④ドキュメントルートにHTMLファイルを設置する
⑤正常に稼働しているかを確認する

注2）バーチャルホストとは、1つのWebサーバ上に複数のドメインで公開するためのしくみです。

①sites-availableディレクトリとvirtual.confを作成

バーチャルホストを使用するときには、バーチャルホスト用の設定ファイルの置き場所としてsites-availableディレクトリを作成します。sites-availableディレクトリを作成して、その中に設定ファイルvirtual.confを作成、編集します。virtual.confの内容は**リスト2**のとおりです。

```
# cd /etc/nginx/
# mkdir sites-available
↓ファイルを作成（リスト2参照）
# vi sites-available/virtual.conf
↓HTMLファイルを置くディレクトリを作成
# mkdir /var/log/nginx/virtual/
```

②nginx.confを編集

①で作成したvirtual.confをNginxに読み込ませるために、httpコンテキストにincludeディレクティブを追記します。**リスト3**のようにnginx.confを編集します。

▼リスト1　nginx.confの全体像

```
#workerプロセスを実行するユーザを指定
user  nginx;                                          ❶
#workerプロセスを起動する数を指定
worker_processes  1;                                  ❷

#エラーログを残すファイルのパスを指定
error_log  /var/log/nginx/error.log warn;
#プロセスIDを保存するファイルのパスを指定
pid        /var/run/nginx.pid;

events {                                              ❸
    #1つのworkerプロセスあたりの同時接続数を指定
    worker_connections  1024;
}

http {                                                ❹
    #mime.typesファイルの読み込む
    include       /etc/nginx/mime.types;
     #MIMEタイプの指定
    default_type  application/octet-stream;

    #アクセスログのフォーマットを指定
    log_format  main  '$remote_addr - $remote_user [$time_local] "$request" '
                      '$status $body_bytes_sent "$http_referer" '
                      '"$http_user_agent" "$http_x_forwarded_for"';

    #アクセスログを残すファイルのパスを指定
    access_log  /var/log/nginx/access.log  main;

    #sendfile APIを使うかを指定
    sendfile        on;
    #tcp_nopush     on;

    #キープアライブ機能をタイムアウトする時間を設定
    keepalive_timeout  65;

    #gzip  on;

    #/etc/nginx/conf.d/配下のconf拡張子のファイルをすべて読み込む
    include /etc/nginx/conf.d/*.conf;
}
```

▼リスト2　virtual.confに記述する内容

```
server {
    #サーバIPアドレスで80番ポートで公開する宣言
    listen 1.2.3.4:80;
    #公開するWebサーバのドメインを指定
    server_name domain.com www.domain.com;
    #アクセスログを残すファイルのパスを指定
    access_log /var/log/nginx/virtual/access_log;
    #エラーログを残すファイルのパスを指定
    error_log /var/log/nginx/virtual/error_log;

    location / {
        #HTMLファイルを置くドキュメントルートパスを設定
        root /usr/share/nginx/virtual/;
        #サイトのトップページにするファイル名を設定
        index index.html;
    }
}
```

▼リスト3　nginx.confの編集内容

```
user nginx;
worker_processes 1;
    ...(略)...
http {
    ...(略)...
    #include conf.d/*.conf
    ↑先頭に#を付けてコメントアウト
    # /etc/nginx/sites-available/配下の
    # .confで終わるファイルをすべて読み込む
    include sites-available/*.conf  ←追記
}
```

▼リスト4　HTMLファイルの例

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8" />
<title>Nginxテストページ</title>
</head>
<body>
テストページです。
</body>
</html>
```

▼図4　設定の確認とNginxの再起動

```
# /etc/init.d/nginx configtest
nginx: the configuration file /etc/nginx/nginx.conf syntax is ok
nginx: configuration file /etc/nginx/nginx.conf test is successful
↑この2行が表示されていたらOK
# /etc/init.d/nginx restart  ←Nginxを再起動
```

③設定ファイルの確認／Nginxを再起動

「/etc/init.d/nginx configtest」を実行すると、設定ファイルが正しく記述されているかを確認できます(**図4**)。エラーがないことが確認できたら、Nginxを再起動して設定を有効にします。

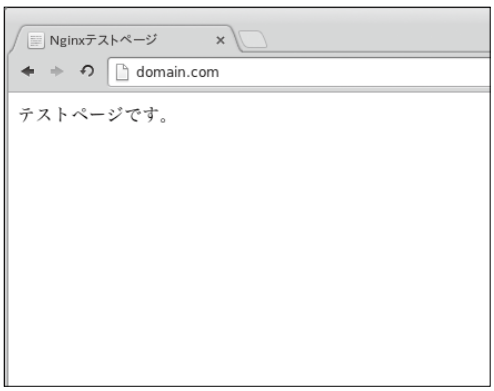
④HTMLファイルを設置

virtual.confに記述したドキュメントルート「/usr/share/nginx/virtual/」に、**リスト4**のようなHTMLファイルを設置します。

⑤正常に稼働しているかを確認

リスト2のserver_nameディレクティブに設定したドメインにブラウザでアクセスして、**図5**のように表示されれば、Webサーバとして正常に稼働しています。また、/var/log/nginx/virtual/access_logを参照して、正常にアクセスログが記録されてい

▼図5　正常稼働している場合に表示される画面



▼図6　access_logの確認

```
# cat /var/log/nginx/virtual/access_log
123.23.34.45 - - [27/Apr/2014:03:45:22 +0900] "GET / HTTP/1.1" 200 156 "-" "Mozilla/5.0 (X11; ➡
Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/34.0.1847.116 Safari/537.36"
 - - [27/Apr/2014:03:45:22 +0900] "GET /favicon.ico HTTP/1.1" 404 570 "-" "Mozilla/5.0 (X11; ➡
Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/34.0.1847.116 Safari/537.36"
```

るかを確認します(図6)。

各ディレクティブ、コンテキストの紹介

「ディレクティブ」とは、Nginxになんらかの指示を出すために記述するもので、

```
[ディレクティブ名] [設定値]
```

のように記述します。ディレクティブ名と設定値の間はスペースやタブで区切ります。たとえば、**リスト1**の❶や❷はディレクティブにあたり、❶の「userディレクティブ」には「nginx」、❷の「worker_processesディレクティブ」には「1」という値が割り当てられています。

また、**リスト1**の❸、❹のように特定のディレクティブには**{}**の中に値を設定するものがあります。この**{}**の中に記述するものを「コンテキスト」といいます。❸、❹をそれぞれ「eventsコンテキスト」「httpコンテキスト」と呼びます。コンテキストにはディレクティブを記述しますが、それぞれのコンテキストによって記述できるディレクティブが決められています。例外として、**リスト1**の❶や❷はmainコンテキストというコンテキストに記述されており、**{}**で囲わずに記述します。

mainコンテキスト

mainコンテキストはプロセスの管理などの設定を記述します。前述のとおり、mainコンテキストだけは**{}**で囲わずに記述します。mainコンテキストに記述する代表的なディレクティブは次のとおりです。

・userディレクティブ

Nginxはmasterプロセスとworkerプロセスによって管理されていて、アクセスが来た際の処理はworkerプロセスが行います。userディレクティブはこのworkerプロセスを実行するユーザ、グループを指定できます。

・worker_processesディレクティブ

workerプロセスは複数起動できます。worker_processesディレクティブはworkerプロセスを起動する数を指定できます。CPUのコアの個数を目安に設定されることが多いです。

・error_logディレクティブ

エラーログを残すパスを指定します。**リスト1**ではパスのあとに「warn」と指定されていますが、これはログを残すエラーのレベル指定になります。error_logディレクティブはmainコンテキスト以外にも、後述するhttpコンテキストやserverコンテキストなどにも設定できます。

・pidディレクティブ

Nginxのpidファイルのパスを指定できます。

eventsコンテキスト

eventsコンテキストは接続の処理に関する設定を記述します。基本的には次のworker_connectionsディレクティブの設定で十分です。

・worker_connectionsディレクティブ

1つのworkerプロセスあたりの同時接続数を指定できます。同時接続数を増やすことで、Webサーバに同時にアクセスできる数を増やせますが、サーバの負荷が増えます。また、少なくしすぎてもサイトにアクセスできる人の数が少なくなってしまうため、512や1024を基準に調整していくと良いでしょう。

httpコンテキスト

httpコンテキストはWebサーバ、Proxyサーバに関する設定を記述します。

serverコンテキスト

serverコンテキストはバーチャルホストに関する設定を記述します。serverコンテキストはhttpコンテキスト内に各バーチャルホストごとに記述します。

・access_logディレクティブ

アクセスログを残すパスを指定します。serverディレクティブごとに記述することで、バーチャルホストごとにアクセスログを残せます。serverコンテキストに記述しない場合は、httpコンテキストに指定しているパスに残ります。

includeディレクティブ

includeディレクティブは外部ファイルを読み込ませるときに使用します。外部ファイルのパスを指定して読み込みをさせますが、

```
include /etc/nginx/conf.d/*.conf
```

のように表記することで、/etc/nginx/conf.d配下のconf拡張子のすべてのファイルを読み込ませることができます。includeディレクティブは設定ファイルの任意の場所に設置できます。

外部ファイル有効利用のススメ

Webサーバなどの設定はhttpコンテキストに記述することで有効になりますが、運用していくにつれてhttpコンテキスト内が煩雑になりがちです。それを防ぐために、**リスト1**のhttpコンテキストのように、極力、各設定ごとに外部ファイルに記述して、includeディレクティブで読み込ませるようにすることをお勧めします。よくある外部ファイルの分け方として、次のようなものがあります。

・バーチャルホストごとにファイルを分ける
・Webサーバの設定やProxyサーバの設定ごとにファイルを分ける

NginxでPHPを動かす

最後に、NginxでPHPを動かす設定を紹介します。Apacheの場合はPHPモジュールを組み込むことで利用できますが、Nginxにはそのようなモジュールはないため、ほかの手段を利用する必要があります。今回はFastCGI形式でPHPを動かす「php-fpm」を利用します。

まずは、PHP関連パッケージとphp-fpmをインストールします。

```
# yum -y install php-fpm php-devel php-pear php-mbstring php-mysql php-pdo
```

次に、バーチャルホスト用の設定ファイルとして作成した、virtual.confに必要な設定を記述します(**リスト5**)。記述する内容のテンプレートは、「/etc/nginx/conf.d/default.conf」に記述されています。

▼リスト5　virtual.confの設定内容

```
server {
    ...(略)...
    location / {
    ...(略)...
    }

    # phpファイルへのアクセスが来た場合の動作の設定
    location ~ ¥.php$ {

        #.phpにアクセスが来た場合は、ローカルの9000番ポートへ渡す
        fastcgi_pass   127.0.0.1:9000;

        #トップページとしてNginxに認識させるファイル名を設定
        fastcgi_index  index.php;

        #バーチャルホストのドキュメントルートで設定したディレクトリパスのあとに$fastcgi_script_nameを付ける
        fastcgi_param SCRIPT_FILENAME /usr/share/nginx/virtual/$fastcgi_script_name;

        #fastcgi関連の設定一覧を読み込む。詳細は/etc/nginx/fastcgi_paramsを参照
        include fastcgi_params;
        #include fastcgi_params以下に記述した場合は、fastcgi_paramsの内容が優先されてしまうため注意
      }
}
```

次の手順でNginxの設定を有効化し、php-fpmを起動します。

```
↓記述ミスがないかチェック
# /etc/init.d/nginx configtest
↓Nginxを再起動
# /etc/init.d/nginx restart
↓php-fpmを起動
# service php-fpm restart
```

ドキュメントルート(/usr/share/nginx/virtual/)に、次のようなphpinfo関数の書かれたファイル「info.php」を設置することで、動作を確認できます。

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

設置後、ブラウザでinfo.phpにアクセスしてPHPの設定一覧が表示されれば、正常に動作しています。

おまけのおまけとして、PHPでセッションを利用する場合に正常に動作しないことがあります。これは、セッションのファイルを「/var/lib/php/session」内に保存するはずが、workerプロセスがsessionディレクトリに保存する権限を持っていないことに起因します(筆者はこれに2時間ほど悩まされました……)。もしセッションがうまくいかない場合は、nginx.confで設定したworkerプロセスの実行ユーザが参照できるように調整を行ってください。

リバースプロキシサーバとしてNginxを利用する

本節では、Nginxをリバースプロキシサーバとして利用する方法を紹介します。今回は(もはや少し古くなりつつある構成かもしれませんが)、バックエンドにApacheを動かしてみます。

設定作業

設定の手順は次のとおりです。

①初期設定、Webサーバの設定を無効にし、Nginxを停止する
②Apacheのインストールと必要最小限の設定をする
③nginx.confとリバースプロキシ用の設定ファイルproxy.confを作成、編集する
④動作を確認する

さっそく設定を行っていきましょう。まずはNginxを停止します。

```
# /etc/init.d/nginx stop
```

次にApacheのインストールと設定を行います。リバースプロキシを確認するための最低限の設定のみ行います。

```
# yum -y install httpd
# /etc/init.d/httpd start
↓ファイルを作成(リスト6参照)
# vi /var/www/html/index.html
```

index.html(**リスト6**)が作成できたら、ブラウザでアクセスして「Apacheテストです。」ページが表示されることを確認します。

その後、**リスト7**のとおりにhttpd.confの編集を行い、Apacheを再起動し設定を有効化します。これにより、Apacheへはサーバ側のローカルからしかアクセスできなくなります。ブラウザからもアクセスできません。

```
# /etc/init.d/httpd restart
```

今度はnginx.confとリバースプロキシ用の設

▼リスト6　index.htmlの例

```
<!DOCTYPE html>
<html>
<head>
<meta charset=UTF-8 />
    <title>Apacheテストです。/title>
</head>
<body>
    Apacheテストです。
</body>
</html>
```

▼リスト7　httpd.confの編集内容

```
...(略)...
#Listen 12.34.56.78:80
#Listen 80   ←先頭に#を付ける
Listen 127.0.0.1:8080   ←追記
...(略)...
```

定ファイルproxy.confを作成、編集します。nginx.confのWebサーバの設定を無効にするとともに、キャッシュ化のための設定を記述します(**リスト8**)。また、/etc/nginx/conf.d/proxy.confを作成して、リバースプロキシ用の設定を記述します(**リスト9**)。さらに、デフォルトで記述されている/etc/nginx/conf.d/default.confの設定を無効にするため、default.conf.bkにリネームします。**リスト8**の①の設定でconf拡張子のファイルのみを読み込むように設定してあるため、bk拡張子にリネームすることでそのファイルは読み込まれなくなります。

```
# mv conf.d/default.conf conf.d/
default.conf.bk
```

以上で設定は完了です。ブラウザでアクセスして「Apacheテストです。」ページが表示されれば、リバースプロキシとして正常に動作しています。またアクセスした際にキャッシュが生成されます。正常にキャッシュされていることを確認するために、/var/cache/nginx/cache/を確認します。index.htmlの内容が含まれたキャッシュファイルがあれば、OKです。

最後に

今回いくつかの設定例を紹介しましたが、みなさんが今後運用される際には、ここでは紹介していないディレクティブを扱うことになるかと思います。しかし、基本的な記述方法は今回紹介したことをベースに設定が行えると思いますので、参考にしていただければ幸いです。SD

▼リスト8　nginx.confの編集

```
http {
     ...(略)...
    #キャッシュを残すディレクトリのパスを指定
    proxy_cache_path /var/cache/nginx/cache/ levels=1:2 keys_zone=cache_zone:40m
inactive=7d max_size=100m;
    #一時ファイルを残すディレクトリのパスを指定
    proxy_temp_path /var/cache/nginx/temp/;

    include /etc/nginx/conf.d/*.conf;   ←①
    #include /etc/nginx/sites-available/*.conf;   ←コメントアウト
}
```

▼リスト9　/etc/nginx/conf.d/proxy.confの作成

```
server {
    #80番ポートにアクセスが来た際に動作する
    listen 80;
    location / {
        #80番ポートにアクセスが来た場合は、ローカルの8080番ポートに渡すようにする
        proxy_pass http://127.0.0.1:8080;
        #受け渡す際のhttpのバージョンを指定する
        proxy_http_version 1.1;
        #受け渡す際のヘッダ情報を指定
        proxy_set_header Host $host:$server_port;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded_Proto http;

        #キャッシュを残すゾーンの名前を定義
        proxy_cache cache_zone;
        #Webページを正常なステータスを返した際のキャッシュを残す時間を指定
             proxy_cache_valid 200 302 20m;
        #Webページがエラーのステータスを返した際のキャッシュを残す時間を指定
        proxy_cache_valid  404 20m;
    }
}
```

第5章 Nginx引っ越し本番!

LINE株式会社　田籠 聡（たごもり さとし）

本章では、インストール・設定したNginxへの移行を行うにあたり、どういった点に気を付ければ良いか、具体的にどのような手順で作業すれば安全な移行ができるかを解説します。すべての確認をこのとおりに実施するのはかなり冗長ですが、一度ていねいにやっておけばだいたいの勘所がわかるようになります。

単体での動作確認

Nginxの設定を行い起動したら、サービスに投入する前に、まず設定した内容が意図どおりに動作するかどうかを確認します。些細なミスで必要なコンテンツの配信ができない状況だった、ということはよく起きるので、面倒なようでもきちんと事前に確認は行いましょう。

動作確認に有効な設定・ツール

サービス投入前ですので、DNS（Domain Name System）の設定などを行うわけにはいきません。手元のPCからサービス投入後のドメイン名でアクセスできるように/etc/hostsを利用します[注1]。

```
#/etc/hosts
203.0.113.16  web.nginx.example.com  app.nginx.example.com
```

これにより、web.nginx.example.comを指定したとき、ブラウザは203.0.113.16に接続します。

また最近のブラウザであれば通信内容を詳細に表示してくれる機能がついていることが多く、これらを有効にして動作確認を行うと、問題があったときに原因の調査がスムーズに進むでしょう。Internet Explorerは[F12]押下で「F12 開発者ツール」が有効になります。Firefoxは「開発ツール」、Chromeなら「デベロッパーツール」という名前で備わっています。Safariは「開発」メニューを設定から有効にすると「Webインスペクタ」が使えるようになります。

またHTTP通信を行うコマンドを用いての確認も有効でしょう。curlコマンド[注2]であればMac OS Xや多くのLinuxディストリビューションで最初から利用できるはずです。またWindowsでも導入は可能なようです。

サービス投入前のサーバにcurlコマンドで接続する場合であれば/etc/hostsを使わず、コマンドラインからHostヘッダを指定することでHTTPリクエストを送れます。-vオプションを指定しておくとリクエストとレスポンスがHTTPヘッダとともに表示され、詳細な通信内容の確認に役立ちます。

```
curl -v -H "Host: web.nginx.example.com" http://203.0.113.16/contents/article1
```

特定のHTTPリクエストヘッダが指定されていた場合に動作が変わるケースの確認などでは、HTTPリクエストの内容を好きなように指定できるcurlコマンドは非常に有用です。慣れておくと助けになる場面は多いでしょう。

注1）LinuxおよびMac OS Xの場合は/etc/hostsを使いますが、Windowsの場合でもhosts設定はできます。パスはWindowsのバージョンによって異なりますので、個別にご確認ください。

注2）http://curl.haxx.se/

設定の確認

サーバの動作を確認するときは設定セクションごとにそれぞれ確認します。おおまかにはserverセクションごと、およびその中のlocationセクションごとに確認します。

serverごとの確認

1つのNginxで複数のserverを設定する場合、動作確認は必ずそれぞれのserverごとに実施します。server_nameで指定したサーバ名でのアクセスはすべてのパターンを試しておくと良いでしょう。とくにserver_nameにワイルドカードや正規表現を使用した場合、マッチする順序は設定ファイル上での記述順序とは異なり、**表1**のルールが適用されます。

こういったルールを混在させて設定を記述する場合[注3]、設定時の意図とは異なったルールが適用されてしまうことがあります。リクエストしたホスト名に対応したコンテンツが正常に配信されているかどうか注意深く確認しましょう。

注3) 筆者はそういった設定記述方法は可能な限り使わないようにしています。しかし経験上、実際の要件を考えるとどうしようもないケースも存在します。

locationごとの確認

server設定セクションの中で複数のlocation設定を記述すると思います。これもできればすべてのlocationについて動作を確認できれば良いでしょう。locationの評価の優先順位もApache HTTP Server(以降、Apache)のLocationとは異なり、記述方法ごとに優先順位があります。まず、locationの記述については**表2**と**リスト1**および**表3**を見てください。

locationは指定したブロックがどのパスにマッチするかのルールを記述しますが、実際にどのルールが適用されるかは比較的複雑です。具体的には次の優先順序に基づいて判定されます。

①完全一致チェック
②最長一致チェック(優先)
③正規表現マッチのチェック(記述順:~と~*では優先順位の差はなし)
④最長一致チェック

最長一致チェックについては記述順序による優先順位付けはなく、リクエストされたURIに対して最も長く一致したものが選ばれます。また修飾子なしの最長一致チェックよりも正規表

▼表1 server_nameの指定ルール

順序	パターン	例
1	完全一致名	app.nginx.le.com
2	アスタリスクで始まるワイルドカード名	*.nginx.example.com
3	アスタリスクで終わるワイルドカード名	app.nginx.*
4	正規表現名	~^app¥d*¥.nginx¥.example¥.com$

▼表2 locationの記述方法

パターン	説明
修飾子なし	文字列の最長一致
=修飾子	文字列の完全一致
^~修飾子	文字列の最長一致(優先)
~修飾子	正規表現マッチ
~*修飾子	正規表現マッチ(大文字小文字区別なし)

▼リスト1 location記述パターン例

```
location /path/content {
 #(A)最長一致チェック
}
location = /path/content2 {
 #(B)完全一致チェック
}
location ^~ /path/content3 {
 #(C)最長一致チェック(優先)
}
location ~ /path/content[0-9] {
 #(D)正規表現マッチのチェック
}
location ~* /path/content[0-9] {
 #(E)正規表現マッチのチェック
 #   (大文字小文字区別なし)
}
```

▼表3 パスがマッチするlocationセクション

パス例	セクション
/path/content2	B
/path/content9	D
/path/CONTENT4	E
/path/content30	C
/path/content	A
/path/content_x	A

現マッチのほうが優先されます。一方、正規表現マッチについては記述順でチェックされ、最初にマッチしたものが使われます。

なお、最長一致もしくは正規表現によるlocationは、その中にさらにネストしたlocationを記述することもできます(**リスト2**)。複雑な設定のWebサーバを設定する場合、わかりやすさを優先してこのネストを使用するのも良いでしょう。

Webサーバとして Nginxを使う

Webサーバとして静的コンテンツを配信する場合に問題となるケースは多くありませんが、Apacheで意識せずに行っていた設定に依存する動作などが変わってしまう可能性はあります。ソフトウェアを変更すると細かい部分でそういった挙動が異なるケースがあり、注意が必要です。

▼リスト2　ネストしたlocationの例

```
server {
 ...(略)...
  location = / {
    # トップページについてのみの設定
    root /var/www/html;
    index index.html;
  }
  location /pub {
    location ¥.(gif|png|jpg)$ {
      # 画像ファイルはこのサーバから直接配信
      root /var/www/content;
    }
    location /pub {
      # 画像ファイル以外はreverse proxyする
      proxy_pass http://127.0.0.1:5000;
    }
  }
  location /css {
    location ¥.css$ {
      # CSSファイルはそのまま配信
      root /var/www/css;
    }
    location /css {
      # それ以外のリクエストはすべて拒否する
      deny all;
    }
  }
  location / {
    # それ以外はすべて/var/www/htmlから返す
    root /var/www/html;
  }
}
```

Content-Typeに注意

HTML、JSON、XMLやJavaScript、CSSなど、Webページの一部としてテキストファイルを配信するケースは非常に多く、これらが正しいHTTPレスポンスとともに返されていないと表示上の文字化けの原因になったり、外部ページから正常にデータを読み込めなくなったりします。また画像や動画などのメディア系ファイル、Office系ソフトウェアの文書ファイルなど、データの配信そのものが主要な目的となる場合についても注意しておきたいところです。

とくに問題になりやすいのがContent-Typeヘッダです。Apacheからの移行であればmime.typesファイルに書いてあった内容によって解決され、意識しないままにセットされていたということも少なくないでしょう。また運用を続ける内に、何者かによってmime.typesを編集、あるいはAddTypeディレクティブを設定され、こっそりうまくいくよう変更されている、などということもあるかもしれません。そのサーバの主要な目的に応じて、HTMLなり画像／動画なり、主要なファイルについてはContent-Typeヘッダが変わっていないかどうか、移行前と移行後で確認しておいたほうが良いでしょう。テキストファイルの場合はcharset指定についても実際のエンコーディングと異なっていないかを確認しておく必要があります。

想定どおりのものが出力されていなかった場合は変更する必要があります。Nginxにもmime.typesファイルはありますが、そのファイルを直接変更するよりは、自分でサーバ設定の一部として明示的に変更を加えたほうが良いでしょう(**リスト3**)。mime.typesの編集は、そのサーバで必要な設定は何だったのかが埋もれてしまう傾向があります。

またファイルのアップローダなど、特定のディレクトリ以下はすべてバイナリファイルとして扱いたい場合などもあるかもしれません。そのときはmime.typesの指定を無効にしdefault_typeのみを指定します(**リスト4**)。

コンテンツのgzip圧縮

サイズの大きなHTML、JavaScriptやJSON、あるいはXMLなどを配信するとき、トラフィックを軽減するために転送時の圧縮を有効にするケースがあります。CPUに負荷をかけるので、すべてのサーバで有効にすれば良いというものではありませんが、とくにインターネットとの接続回線が低速である場合や通信量ごとの料金が負担となっている場合には検討すると良いでしょう。また移転元で有効であった場合、Nginx移転時に無効にしてしまうと、急激なトラフィックの増大などに驚くことになるかもしれません。

サーバ全体で有効にする場合は単純にserverセクションでgzip on;と記述します(**リスト5**)。またデフォルトではtext/htmlのコンテンツのみが対象であるため、そのほかに必要なmime typeを指定する必要があります。

▼リスト3　.pubファイルをtext/plainとして配信したい例

```
server {
 ...(略)...
  include mime.types;
  types {
    text/plain  pub;
  }
  default_type  application/octet-stream;
}
```

▼リスト4　/download以下はすべてバイナリファイルとして扱う場合

```
server {
 ...(略)...
  location /download/ {
    types { }
    default_type application/octet-stream;
  }
}
```

▼リスト5　gzip圧縮の有効化

```
server {
 ...(略)...
  gzip on;
  gzip_types text/plain text/css text/xml application/
javascript;
  gzip_min_length 1000;  # 1000bytes以下のファイルは圧縮しない
}
```

確認はcurlコマンドを使って簡単に行えます(**図1**)。リクエスト送信時にはgzip転送を有効にするようAccept-Encoding: gzipヘッダを付けます[注4]。有効／無効はContent-Typeごとに異なるため、サーバでリクエストの多いコンテンツについては個別に確認するようにしましょう。

リバースプロキシサーバとしてNginxを使う

Nginxをリバースプロキシサーバとして運用する場合、クライアントとの間のリクエスト／レスポンスのほかに、アプリケーションサーバとの間のリクエスト／レスポンスについても注意する必要があります(**図2**)。

とくにアプリケーションサーバはクライアントからの通信を直接受けないという前提で起動していることが多く、Nginxがクライアントからの不正なリクエストをそのまま素通しにしてしまうと問題になるケースもあるため、できれば事前にチェックしましょう。

アプリケーションサーバへのリクエスト

リバースプロキシが間に入る場合、アプリケーションサーバからは、HTTPクライアントはリバースプロキシサーバとなるため、追加の情報なしには本来のクライアントが誰なのかをアプリケーションサーバに伝えることができなくなります。

Apacheのmod_proxyはこれを防ぐため、次のHTTPリクエストヘッダを自動的に付与してアプリケーションサーバにプロキシしています。

- X-Forwarded-For：クライアントのIPアドレス
- X-Forwarded-Host：クライアントがHostヘッダで渡す、オリジナルのホスト名
- X-Forwarded-Server：プロキシサーバのホスト名

注4) なお単に転送時の圧縮を有効にするだけならcurl --compressedオプションで有効にできます。ここではgzipを明示するため-Hを用いました。

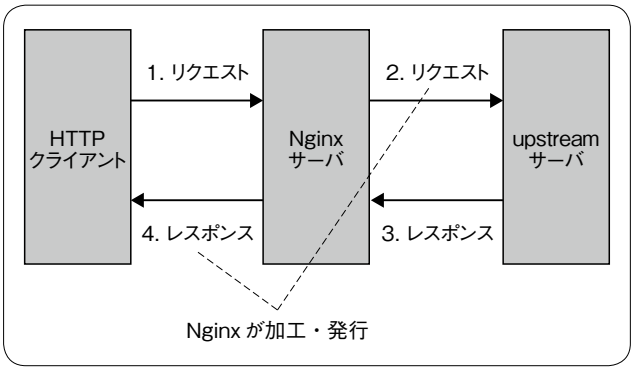
これらヘッダを付与するためには、Nginxでは**リスト6**の設定を加えます。httpセクションに加えることもできますがserverセクションで個別に設定したほうが、サーバごとの動作設定が明確になって良いでしょう。これらの設定を加えることで、多くのアプリケーションサーバはアクセスログのクライアントIPアドレスを正しく表示できるようになるはずです。またアプリケーションサーバがリクエストのHostヘッダを必要とする場合は、これもセットするようにします。逆にX-Forwarded-Forなどがクライアントからのリクエストヘッダにセットされており、リバースプロキシサーバがそれを素通しにしてしまうと、アプリケーションサーバ上のログにユーザが自己申告してきたIPアドレスが記録されることになります。これは非常にリスクが大きいため、X-Forwarded-Forのセットは忘れないようにしましょう[注5]。

注5）多段リバースプロキシを構成する場合には$remote_addrではなく$proxy_add_x_forwarded_forをセットするようにしましょう。

クライアントへのレスポンス

アプリケーションサーバはデバッグ目的などでサーバ自身の状況をレスポンスヘッダにセットするような場合があります。こうした情報は

▼図2　リバースプロキシの扱うリクエストとレスポンス



▼リスト6　リバースプロキシサーバにおけるヘッダの追加

```
server {
 ...(略)...

  proxy_set_header X-Forwarded-For    $remote_addr;
  proxy_set_header X-Forwarded-Host   $host;
  proxy_set_header X-Forwarded-Server $host;

  proxy_set_header Host               $host;
}
```

▼図1　gzip圧縮が有効かどうかの確認

```
$ curl -v -s -H "Accept-Encoding: gzip" -H "Host: web.nginx.example.com" http://203.0.113.16/
> /dev/null
* About to connect() to 203.0.113.16 port 80 (#0)
*   Trying 203.0.113.16 ...
* connected
* Connected to 203.0.113.16 (203.0.113.16) port 80 (#0)
> GET / HTTP/1.1
> User-Agent: curl/7.24.0 (x86_64-apple-darwin12.0) libcurl/7.24.0 OpenSSL/0.9.8y zlib/1.2.5
> Host: web.nginx.example.com:80
> Accept: */*
> Accept-Encoding: gzip        →gzip転送を有効にするようリクエスト
>
< HTTP/1.1 200 OK
< Server: nginx/1.6.0
< Date: Sun, 18 May 2014 05:18:14 GMT
< Content-Type: text/html
< Last-Modified: Sun, 18 May 2014 04:47:47 GMT
< Transfer-Encoding: chunked
< Connection: keep-alive
< Content-Encoding: gzip        →gzip転送が有効であることを示すレスポンスヘッダ
<
{ [data not shown]
* Connection #0 to host 203.0.113.16 left intact
* Closing connection #0
```

アプリケーションの開発時には有用ですが、顧客に向けてサービスを稼働するときにはクライアントに向けて送信する必要はありません。このような無用なヘッダがわかっていれば、リバースプロキシサーバで落としてしまう(**リスト7**)ほうが良いケースもあります。

Nginxのproxy_http_moduleはデフォルトでいくつかのヘッダを削除します[注6]。それ以上に削除したいものがあればこのように指定をしますが、何でも削除すればいいというものでもありません。無闇に行うのではなく、確実に不要だというものについてのみ行うようにしましょう。

ダウンタイムのない切り替え

動作確認が完了したら、現行サーバとの切り替えを行います。切り替え手順を誤るとサービスが稼働していない時間ができてしまい、クライアントにとって不都合な状態になります。このような事態を避けるため、いくつかの手順を守る必要があります。構成によって気を付ける点が異なるため、それぞれのパターンについて確認していきましょう。

単独サーバの移行

1サービス用にサーバが1台しかなく、そのサーバをApacheからNginxに切り替える場合を考えてみます。

DNS変更による切り替え

ユーザがドメイン名を指定したときにどのサーバにリクエストを送るかを決めるのがDNSレコードです。ApacheからNginxへの切り替えを考える場合、それぞれ別個にサーバを準備し、Nginx側の準備ができた時点でDNSレコードの登録内容を変更して、それがユーザの手元に反映されるのを待つのが確実です(**図3**)。

このとき、切り替え時にトラブルが発生して切り戻すことを考えると、次のような手順で切り替えるのが良い手順でしょう。

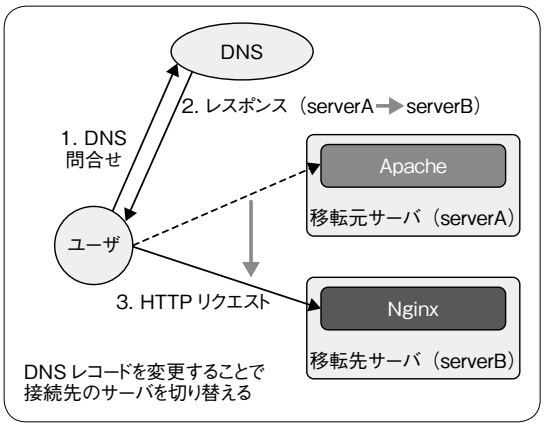
①該当のDNSレコードのTTL(Time to Live)を短く変更する(切り替えの2〜3日前)
②切り替え先Nginxの動作確認
③DNSレコード変更による切り替え実施
④切り替え元サイトへのリクエストが十分少なくなるまで様子を見る
⑤問題なければTTLをもとに戻す

DNSレコードのTTLを短くしておくのは非常に重要です。DNSレコードのTTLは86,400秒(1日)となっていることも多く、そのままだと切り替え実施時に変更したDNSレコードはDNSキャッシュサーバなどで1日のあいだ保持されてしまい、万が一切り替えに失敗して切り戻したいときにDNSレコードを再度変更しても、変更後の情報を取得しにきてくれない可能性があるからです。

▼リスト7　リバースプロキシサーバで不要なヘッダを削除する

```
server {
 ...(略)...
  proxy_hide_header  X-Cache;
  proxy_hide_header  X-Cache-Lookup;
  proxy_hide_header  Warning;
  proxy_hide_header  Via;
}
```

▼図3　DNS切り替えによる変更



注6) http://nginx.org/en/docs/http/ngx_http_proxy_module.html#proxy_hide_header

DNSレコードのTTLが86,400秒の設定になっている場合は、移転作業の1日以上前にTTLを短くする作業を行う必要があります。短か過ぎても問題があるので、60秒(1分)から600秒(10分)程度の数値を指定するのが良いでしょう。あまり短過ぎるとDNSサーバへの負荷が増大する可能性があります。一方長くすると切り替え／切り戻しが有効になるまでの所要時間が伸びることになります。最終的にはご利用中のDNSサーバの状況も考えて決めてください。

切り替えを実施したら、切り替え元(Apache)と切り替え先(Nginx)の両方でログなどで継続的に状況を確認します。理想的にはApache側にまったくリクエストが来なくなった時点で切り替え完了となります。

しかし現実的にはそうはならず、大部分のリクエストが切り替え先のNginxに送られるようになった時点で切り替え完了と割り切る必要があります。Operaなどの一部のブラウザや各所のDNSキャッシュサーバなど、DNSレコードで指定されているTTLを無視してキャッシュを保持するソフトウェアやサーバがどうしても存在するためです[注7]。

切り替え完了と判断した時点で、切り替え元のApacheをシャットダウンし、DNSレコードのTTLをもとに戻します。

同一サーバ上でのソフトウェア変更

使用するサーバは同じで起動するソフトウェアをApacheからNginxに切り替える場合、ダウンタイムが完全にゼロというのは難しいでしょう。Apacheを終了してからでないとNginxは起動できません。できるだけユーザからのアクセスが少ない曜日・時間帯を選び、「えいや」で切り替えることとなります。できることなら切り替え先を別のサーバに用意し、DNS変更による切り替えを行うほうが望ましいでしょう(**図4**)。

注7) たとえばOperaは再起動しない限りDNSキャッシュを保持し続けるため、Operaを起動しっぱなしにしているユーザがいれば切り替え前のサーバへのアクセスはずっと来ることになります。

サーバの手配ができないなどのどうしようもない場合は、覚悟を決め、できるだけ失敗しないように切り替えを実行します。

```
$ nginx -t
 →必ず設定が正しく行われているかなどを確認する
$ service httpd stop; service nginx start
```

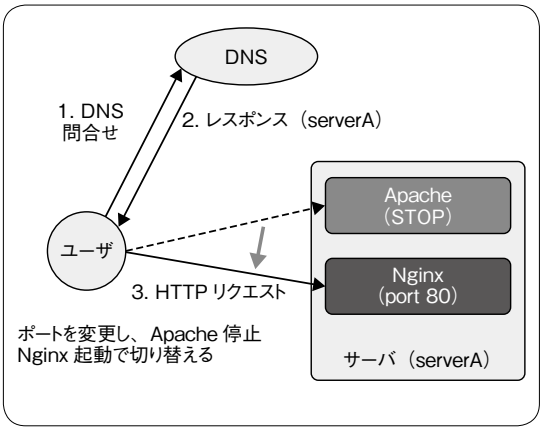
Nginxの設定ファイルに文法エラーなどがあると、Apacheが停止したあとNginxが起動しないという状況となり、停止時間が大幅に伸びることになりますから絶対に避けなければなりません。事前に必ずnginx -tオプション[注8]を使用してチェックを実施しましょう。

またlistenディレクティブを複数用いることで、あらかじめ起動して動作確認を行ったNginxをApacheとの切り替えに用いる、という手順もとれます。

①Apachは80番ポートで起動した状態のまま
②Nginxを80番ポート以外のポートで起動し動作確認する(**リスト8**)
③動作確認が完了したら80番ポートを使用するよう設定ファイルを書き換える(**リスト9**)
④Apacheを停止
⑤Nginxの設定をリロードして80番ポートも使用する

注8) Apacheにおけるapachectl configtestにあたる機能です。

▼図4　Apache/Nginx切り替え



```
$ service nginx reload
 →あるいはnginx -s reload
```

このようにすれば、いざ切り替えようとしたNginxが起動しない、というリスクは多少なりとも下げられるはずです。

複数サーバ構成での移行

DNSラウンドロビン構成の場合

DNSラウンドロビンを用いて複数サーバによるサービス提供構成を作っている場合、移転の方法は基本的には単独サーバにおけるDNS変更での切り替えと同じです。あらかじめTTLを短くし、切り替え対象のDNSレコードを変更して、切り替え完了と判断できたらTTLを戻します(**図5**～**図7**)。より詳細に見ると、次の方法があります。

・全レコードを一度に切り替え
設定されているDNSレコードをすべて同時に書き換える

・レコードの追加と削除による切り替え
まず移転先サーバを参照するレコードを追加。問題なければ移転元サーバを参照するレコードを削除する

レコードの追加と削除による切り替えを行うほうが、切り替え元の確実に動作するDNSレコードに手を入れる必要がなく、確実な移行が可能でしょう。切り戻しも、追加したレコードの削除のみで実施できるのでより単純です。

全レコードの書き換えを行うほうが作業としては単純ですが、DNSレコードの変更はどちらにせよ即座に全ユーザに対して切り替えを実行できません。DNSラウンドロビン構成をとっている以上は切り替え前後のサーバが並行して動

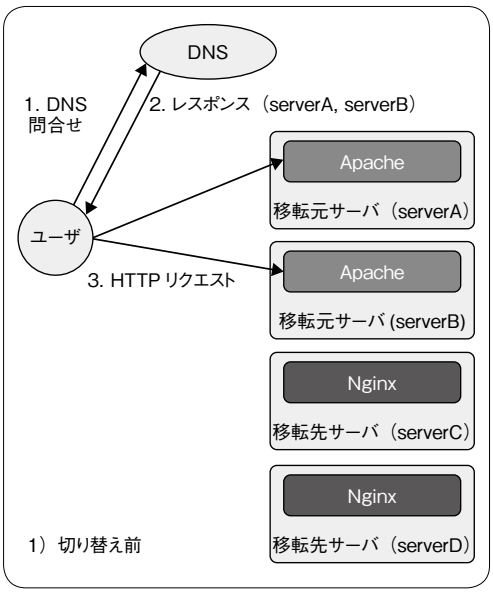
▼リスト8　Nginxにおける80番ポート以外での起動

```
server {
  # listen 80;
  listen 8080;
 ...(略)...
}
```

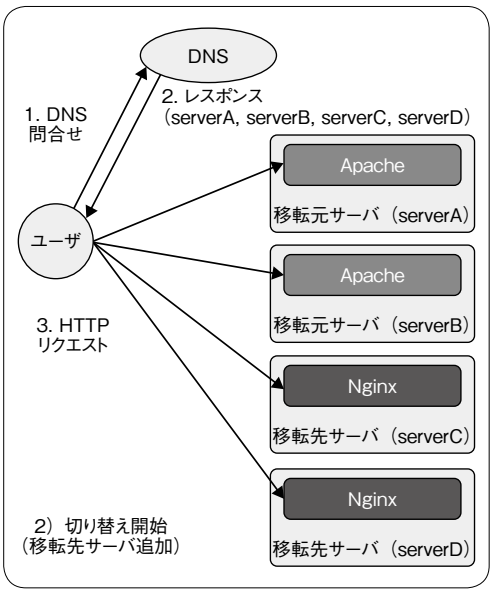
▼リスト9　Nginxで80番ポートを有効にする設定

```
server {
  listen 80;
  listen 8080;
 ...(略)...
}
```

▼図5　DNSラウンドロビン構成での切り替え作業（切り替え前）



▼図6　DNSラウンドロビン構成での切り替え作業（切り替え開始）



作することは避けられないため、レコードの追加と削除で実施するほうが確実でしょう。

ロードバランサ構成の場合

サービス提供にロードバランサを用いている場合、切り替え作業はこのロードバランサの設定変更によって行うこととなります。専用ハードウェアとしてのロードバランサは比較的高価ですが、クラウドサービス各社がサービスの一部としてロードバランサ機能を提供していることも多く、最近であればロードバランサを用いた複数サーバ構成を採ることのほうが多いかもしれません。最も利用例が多いと思われるAmazon Elastic Load Balancing(Amazon ELB)については第7章でも解説されています。

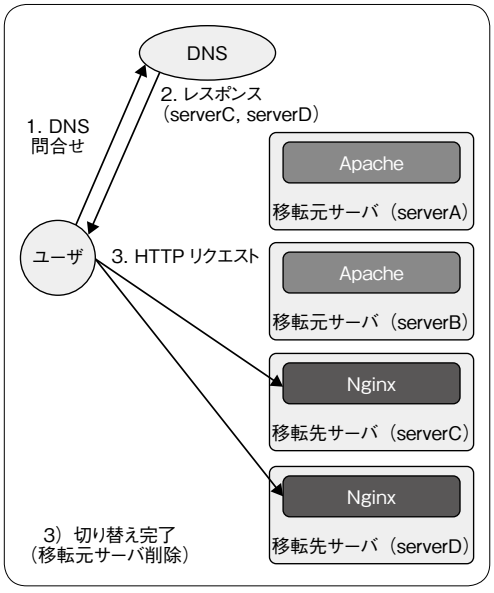
ロードバランサによる切り替えは、設定変更を行うと即座に、確実に反映されます。DNS変更などと比べると手順も少なく確実でしょう。

ロードバランサを用いている場合も、DNSラウンドロビンのときと同様、一度に変更する方法と、段階的に切り替える方法があります。

しかしDNSラウンドロビン構成とは異なり、ロードバランサによる切り替えであれば設定は即座に、全体的に反映されるはずです。段階的に切り替えることについては利点があまりないため、ロードバランス先のリストを一度に移転元から移転先にすべて書き換えてしまうほうが単純で良いと思います(**図8**)。切り戻しについても同じく設定変更で即座に反映させられるはずですので、あまり怖れることはありません。

設定変更を実施したら、移転元サーバ側にはHTTPリクエストがまったく届かなくなっていることを確認しましょう。移転元サーバがロードバランス先のリストに残ってしまっていたりするとトラブルのもとになります。

▼図7　DNSラウンドロビン構成での切り替え作業(切り替え後)

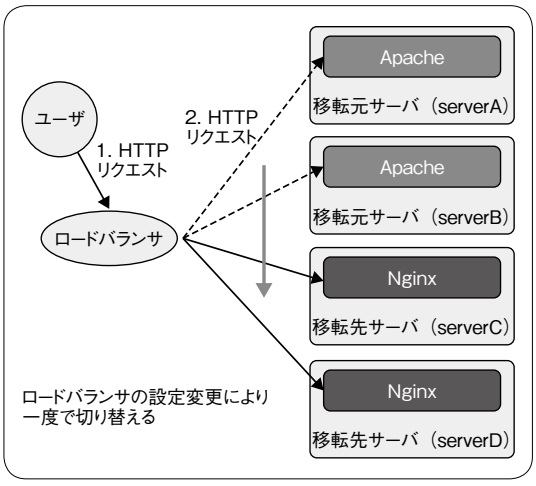


まとめ

動作確認や実際の移転の方法は、サーバをどういう構成にするかによって異なります。こうすればどのような場合でも大丈夫、という万能の方法は残念ながらありません。

しかし常に大事なのは、作業を行うときには入念に確認すること、何をやったかをきちんと記録すること、いつでも元の状態に戻せるようにすること、です。とくに確認も終わっていないのに移転元の環境をうっかり壊してしまうと取り返しがつきませんから、そこには細心の注意を払いましょう。SD

▼図8　ロードバランサ構成での切り替え作業



第6章 移行後に気を付けておくべきこと

LINE株式会社　田籠 聡(たごもり さとし)

動作確認も切り替えも完了しました、という状況においても、やはりさまざまなことが起きます。トラブルとは言えない些細なことであっても運用上の負荷になりますし、サービス提供に直結しないことであれば動いていないものがあってもなかなか気づけない、ということもあります。ここではそういったトピックをいくつか取り上げ、長期の運用における注意点の例を示します。

運用手順の変更

運用手順については、そう変わったところはありません。Apache HTTP Server(以降、Apache)ではapachectlを使用していた操作が

```
$ nginx -s OPTION
```

を用いるようになることくらいでしょうか。ただしいくつか異なる部分もあります。

サーバ管理

設定を反映させる

yumなどのパッケージシステム経由でインストールした場合は設定ファイルを編集したあと

```
$ service nginx reload
```

で反映させることができます。Nginxのコマンドであれば

```
$ nginx -s reload
```

です[注1]。これは実際にリクエストを処理しているプロセスの再起動を伴い、ほとんどあらゆる設定変更はこの操作で反映させられるはずです。

Nginxを再起動

OSレベルでのプロセスに対する設定を変更した場合などは再起動が必要です。

```
$ service nginx restart
```

とするか、もしくは

```
$ nginx -s stop
```

のあとにNginxを起動します。

ログローテーション

Nginxはログローテーション時にログファイルを開き直すための専用の操作があります。「nginx -s reopen」で実行できます[注2]ので、手でログファイルのリネームを行った後などには実行するようにしましょう。

またサーバ移行時に忘れやすいのがログローテーション設定の変更・追加です。Nginx公式のrpmには/var/log/nginx/*.logをローテートするための/etc/logrotate.d/nginx設定ファイルが含まれていますが、ここ以外のパスにログファイルを置くようにした場合などに変更を忘れることも多いので注意しましょう。

監視・ログの扱いの変化

中長期の動作状況の把握に、リソースとログのモニタリングは欠かせません。できればモニタリングのしくみを完備してからサービスの切

注1) Nginxプロセスへの SIGHUP送出と同じ。

注2) Nginxプロセスへの SIGUSR1送出と同じ。

り替えを実施したいところですが、なかなかそうもいかず、見逃してしまいがちです。気づいた時点でしっかり設定するようにしましょう。

動作状況のモニタリング

Nginxサーバの動作状況はシステムリソースのモニタリングとNginx自身が出力するステータスのモニタリングの両方を行いましょう。どちらかだけではトラブルが起きていることはわかってもその原因がまったく想像できない、といった状況になってしまいます。

またこれらの数値はすべてグラフにしておくと、いつ何が起きたのか、長期的な傾向はどうなっているのか、それぞれの数値は相互にどう関係しているのか、など多くの情報が得られるようになります。Ganglia[注3]やZabbix[注4]、Cloud Forecast[注5]などのツールが有効です。

リソースのモニタリング

サーバリソースのモニタリングを行い次の数値を時系列で記録しておくのは非常に重要です。

- ネットワークトラフィック(inbound/outbound、単位：Mbps)
- CPU使用率(user/system/iowait/...、単位：%)
- メモリ使用率(used/buffer/cached/avail/swap、単位：%)
- ロードアベレージ

Nginxはイベント駆動アーキテクチャを採用しているため、通常プロセスのCPU使用率は非常に低く抑えられます。またApacheをprefork mpmで起動しているときに比べてメモリの使用量も非常に低くなるでしょう。CPU使用率／メモリ使用率が不自然に大きくなる場合、NginxでCPU／メモリに負担のかかる処理を行わせ過ぎている可能性があります。普段の負荷の上下はネットワークトラフィックとロードアベレージの変動を見ていれば傾向がわかるはずです。

注3) http://ganglia.sourceforge.net/
注4) http://www.zabbix.com/jp/
注5) https://github.com/kazeburo/cloudforecast

http_stub_status module

Nginx自身の動作状況を見るにはstub_statusモジュール[注6]が非常に便利です。Nginx公式rpmパッケージでは有効な状態でビルドされていますので、設定を追加すれば使用可能です(**図1**、**リスト1**)[注7]。

機能的にはApacheのmod_statusとほぼ同じです。またstub_statusを有効にするパスは、弊社環境ではサービスのパスとかぶることがないよう、常に先頭にアンダースコア(_)を複数付けたものを設定しています。

これらの出力を定期的に取得しグラフ化しておくと、Nginxに対してリクエストが多い時間帯や長期の傾向が非常にわかりやすくなります。CPU使用率やロードアベレージと同じ画面で参照できるようにしておくと良いでしょう。

ログ書式の変更点

Webサーバにおいてアクセスログを出力するのは非常に重要です。Nginxでは自由にログ書

注6) http://wiki.nginx.org/HttpStubStatusModule
注7) 手元のバイナリで有効になっているかどうかは「nginx -V」で確認できます。有効になっていなければ--with-http_stab_statusオプションを有効にしてNginxを再ビルドする必要があります。

▼図1　stub_statusの出力例

```
$ curl -s http://localhost/___nginx_status
Active connections: 1223
server accepts handled requests
 4152033745 4152033745 9873080103
Reading: 0 Writing: 70 Waiting: 1153
```

▼リスト1　stub_statusを有効にするための設定

```
server {
 ...(略)...
  location /___nginx_status {
    stub_status on;
    # オフィスやLAN内からのアクセスのみ許可すること
    # allow 10.0.0.0/8;
    allow 127.0.0.1;
    deny all;
  }
}
```

式が設定できますが、Apacheの書式と同じように設定しても細かいところでどうしても違いが出ます。ログを扱っているプログラムなどがある場合は、その動作や設定もチェックしておくと良いでしょう。

ApacheとNginxにおけるログ書式

Apacheにおいて最も広く使用されている書式はCombinedと呼ばれる書式ではないかと思います[注8]が、筆者はこれに加えてリクエストを処理するのにかかった時間(%D)を必ずログに出力するようにしています(**図2**、**リスト2**)。

これとほぼ同じログ書式をNginxでも指定できますが、どうしても異なる部分が出てきます(**図3**、**リスト3**)。

Apacheの%Dは処理にかかった時間をマイクロ秒で出力しますが、Nginxの$request_timeはリクエスト処理時間を秒単位で、かつミリ秒まで出力します。処理に1ミリ秒もかからなかった場合は"0.000"と表示されます。アクセスログを回収して処理時間の傾向などの算出を行っている場合、この差を吸収する必要があるので注意しましょう。

またこのほかにもNginxでログ出力に使用できる情報は数多くあります。ドキュメント[注9]には一度目を通しておくと良いでしょう。

まとめ

この章では移行後に気を付けるべきこととして、Nginxのオペレーションにおける基本的な操作や設定などについてまとめました。この内容はごく基本的なことがほとんどですが、一方でこの内容をきちんと抑えて実施しておくことで、多くのトラブルの原因究明に必要な情報を手に入れられる、というものでもあります。

何かトラブルが起きたときに最も大事なことは、平常時から情報収集を継続的に行っていること、それらの情報を丹念に見ることです。また普段の動作状況を見て傾向を把握することは、さまざまなトラブルを未然に防ぐためにも、無理や無駄のない規模の拡張を行うためにも欠かせません。

実施しておくべきことはApacheでもNginxでも(あるいはほかのWebサーバソフトウェアでも)変わりはありませんが、だからこそ、使用するソフトウェアの変更という良い機会に復習しておきたいものです。SD

注8) "Combined Log Format" http://httpd.apache.org/docs/2.2/ja/logs.html

注9) http://nginx.org/en/docs/http/ngx_http_core_module.html#variables

▼図2　Apacheアクセスログ出力例

```
203.0.113.17 - - [18/May/2014:21:00:02 +0900] "GET /path/to/content HTTP/1.1" 200 30145 ↵
"referer" "user-agent" 1829050
```

▼リスト2　Apache LogFormat例

```
LogFormat "%h %l %u %t ¥"%r¥" %>s %b ¥"%{Referer}i¥" ¥"%{User-Agent}i¥" %D" accesslog
```

▼図3　Nginxアクセスログ出力例

```
203.0.113.17 - - [18/May/2014:21:00:02 +0900] "GET /path/to/content HTTP/1.1" 200 30145 ↵
"referer" "user-agent" 1.829
```

▼リスト3　Nginx log_format例

```
log_format accesslog  '$remote_addr - $remote_user [$time_local] "$request" '
                      '$status $body_bytes_sent "$http_referer" '
                      '"$http_user_agent" $request_time';
```

第7章 クラウドでのNginxの使い方

株式会社データホテル　大久保 智之（おおくぼ ともゆき）

本章では、パブリッククラウド環境におけるNginxの利用例を挙げます。AWS（Amazon Web Services）の環境を利用して、WordPress環境を構築してみます。クラウドでNginxを利用する際のイメージをつかんでください。

AWSとは

AWSとはAmazonが提供しているクラウドサービスです。クラウドサービスが持つ特徴の1つは「使いたいときに」「使いたいサービス」を選んで使えることでしょう。また、AWSではAmazon EC2やAmazon RDS、ELBをはじめ、さまざまなサービスが用意されており、その充実ぶりは各社クラウドサービスの中でも随一です。本章ではNginxと同時に次のサービスを利用することにします。

・ELB（Elastic Load Balancing）
・Amazon RDS（Relational Database Service）
・Amazon EC2（Elastic Compute Cloud）
・Amazon VPC（Virtual Private Cloud）

Amazon EC2でNginxを動かそう

まずはAWSの代表的なサービス「Amazon EC2」（以降、EC2）を使って、EC2上で動くインスタンスにNginxをインストールし、動作させるまでを一例として紹介します。

鍵ペアの作成

のちほど起動させるインスタンスにリモートログインするために、事前に鍵ペアを作成しておきます。

（1）EC2 Dashboardの左メニューから「Key Pairs」を選択します
（2）「Create Key Pair」ボタンを押下します。「Key pair name」を入力後、「Create」ボタンを押下すると鍵ペアが作成されます

鍵作成と同時に、秘密鍵がローカルPCにダウンロードされます。秘密鍵はなくさないようにしてください。

Amazon EC2インスタンスの選択と起動

それではEC2でインスタンスを立ち上げてみましょう。マネジメントコンソールのウィザードに従い、インスタンスを作成していきます。

最初にマネジメントコンソールから、「EC2」を選択します。

（1）EC2 Dashboardの中央メニュー「Create Instance」から「Launch Instance」ボタンを押下します
（2）「Step 1: Choose an Amazon Machine Image(AMI)」では、起動するインスタンスを選択します。ここでは左メニューの「AWS Marketplace」を選択します
（3）「AWS Marketplace」でCentOS 6のAMIを検索します。検索した結果表示されるイ

ンスタンスの中から今回は「CentOS 6 (x86_64)-with Updates(ami-31e86030)」を選択します

(4) 「Step 2: Choose an Instance Type」では、インスタンスを選択します。今回は「t1.micro」を選択します。「Next:Configure Instance Details」ボタンを押下します

(5) 「Step 3: Configure Instance Details」では次の値を選択し、「Next: Add Storage」ボタンを押下します
Number of instances：1
Network：Launch into EC2-Classic
Subnet：No preference
Availability Zone：No preference
IAM role：None
Shutdown behaivior：Stop
Enable termination protection：「Protect against accidental termination」にチェック
Monitoring：チェックなし

(6) 「Step 4: Add Storage」では次の値を選択し、「Next: Tag Instance」ボタンを押下します
Type：EBS
Device：/dev/sda
Size(GiB)：8
Volume Type：Standard
Delete on Termination：チェック

(7) 「Step 5: Tag Instance」では次の値を選択し、「Next: Configure Security Group」ボタンを押下します
Key：nginx
Value：nginx1

(8) 「Step 6: Configure Security Group」では、今回の環境用に新しいSecurity Groupを作成します。「Review and Launch」ボタンを押下します
Assign a secirty group：「Create a new security group」を選択
Security group name：SD-nginx
Description：Security Group for nginx

セキュリティグループの設定例は**表1**のとおりです。

(9) 「Step 7: Review Instance Launch」では、選択結果に問題なければ「Launch」ボタンを押下します(**図1**)

(10) 「Select an existing key pair or create a new key pair」とポップアップ表示されます。先ほど用意したkey pairを選択し、「I acknowledge that I have access to the selected private key file (your key pair), and that without this file, I won't be able to log into my instance.」のチェックボックスにチェックを入れて、「Launch Instances」ボタンを押下します。
「Lanuch Instances」ボタンを押下後、しばらくすると、インスタンスが起動してきます

▼表1　EC2のセキュリティグループ

Inbound

Type	Protocol	Port Range	Source	備考
SSH	TCP	22	自身の接続元IPアドレス	リモートログインできるIPを制限する
HTTP	TCP	80	自身の接続元IPアドレス	—

Outbound

Type	Protocol	Port Range	Source	備考
All traffic	All	All	0.0.0.0/0	—

インスタンスへのログイン

インスタンス作成時に選択したkey pairの秘密鍵を使って、SSHでrootログインをします。接続先はインスタンスの「Description」から確認できます。

起動したインスタンスのPublic DNS/Public IPは可変です。固定にしたい場合は、Elastic IPを利用してください。

CentOSのセットアップ

今回選択したAMIはiptablesが有効になっています。Nginxの動作に必要なため、HTTP(80)を許可する必要があります。iptablesで次のルールを許可するか、iptablesを無効にしてください。

```
-A INPUT -m state --state NEW -m tcp -p tcp --dport 80 -j ACCEPT
```

また、SELinuxが有効になっています。これを無効にする場合は、「/etc/selinux/config」ファイルを編集後、インスタンスを再起動してください。

起動させたインスタンスは、初期状態ではSwap領域を持ちませんので、必要に応じてSwapファイルを作成し、これをマウントしてください。

以降で必要になるRPMパッケージを事前にインストールしておきます。

```
# yum install wget mysql
```

Nginxのセットアップ

本題のNginxをセットアップします。NginxのインストールはNginx公式サイトのRPMパッケージを用います。

(1) Nginx公式サイトからリポジトリ用のRPMパッケージをインストールします

```
# yum install http://nginx.org/packages/centos/6/noarch/RPMS/nginx-release-centos-6-0.el6.ngx.noarch.rpm
```

(2) Nginxをインストールします。RPMパッケージをインストールした時点で自動起動の設定が有効になっています

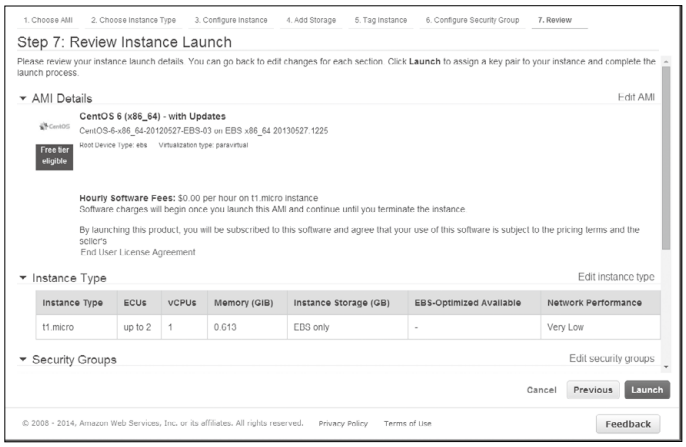
```
# yum install nginx
```

(3) Nginxを起動します

```
# service nginx start
```

(4) ブラウザでPublic DNSもしくはPublic IPで起動したインスタンスへアクセスします。Nginxのトップページ(**第4章の図3**)が表示されればOKです

▼図1　インスタンスの確認画面



PHP環境のセットアップとNginx/php-fpmの連携

ここではPHPの環境を用意し、Nginxと連携させます。

(1) PHPの環境をyumでインストールします

```
# yum install php php-fpm php-mbstring php-mysql
```

(2) php-fpmの自動起動設定を有効にします

```
# chkconfig php-fpm on
```

(3) php-fpmの設定ファイルを編集します

```
# vi /etc/php-fpm.d/www.conf
user = nginx
group = nginx
```

(4) php-fpmを起動します

```
# service php-fpm start
```

(5) default.confをphp-fpmと連携できるように修正します

```
# vi /etc/nginx/conf.d/default.conf
※編集内容はリスト1を参照
```

(6) Nginxを再起動します

```
# service nginx restart
```

(7) セッション保持用ディレクトリの権限を変えておきます

```
# chown root:nginx /var/lib/php/session
```

(8) 動作確認をします。第3章の「NginxでPHPを動かす」で示したようなPHPファイル(info.php)をコンテンツとして用意し、ブラウザでアクセスしたときにPHPの設定一覧画面(図2)が表示されればOKです

AMIの作成

ここで、これまで使っていたインスタンスからAMIを作成します。対象のインスタンスを選択し、右クリック→「Create Image」でAMIを作成します。こうしておくことで、いつでもAMIから今と同じ状態のインスタンスを新たに起動できます。

▼図2　PHPの設定一覧画面

▼リスト1　default.conf

```
server {
    listen       80;
    server_name  "";
    root   /usr/share/nginx/html;
    index  index.html index.htm index.php;

    access_log  /var/log/host.access.log  main;

    location ~* ¥.php$ {
        fastcgi_pass   127.0.0.1:9000;
        fastcgi_index  index.php;
        fastcgi_param  SCRIPT_FILENAME  $document_root$fastcgi_script_name;
        include        fastcgi_params;
    }
}
```

NginxとAWSを組み合わせてWordPressを動かそう

それでは次にAWSで提供されているサービスとNginxを組み合わせてみましょう。EC2上で作成した環境を基に、さらにWordPressをインストールします。最終的な環境は**図3**のようになります。

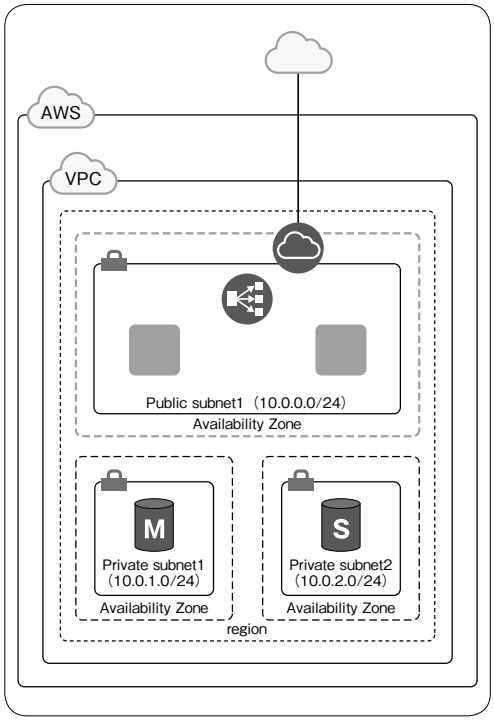
Amazon VPCの設定

まずは「Amazon VPC」でVPC環境を作ります。マネジメントコンソールから、「VPC」を選択します。

(1)「Start VPC Wizard」ボタンを押下します
(2)「Step 1: Select a VPC Configuration」では、「VPC with a Single Public Subnet」を選び、「Select」ボタンを押下します
(3)「Step 2: VPC with a Single Public Subnet」では次の値を選択し「Create VPC」ボタンを押下します

IP CIDR block：10.0.0.0/16
VPC name：SD-nginx-vpc
Public subnet：10.0.0.0/24
Availability Zone：us-east-1a
Subnet name：Public subnet1
Enable DNS hostnames：Yesを選択

▼図3　AWSで作るWordPress環境



これで新規にVPCが作成されます。

続いて追加でサブネットを作成しておきます。VPC Dashboardの左メニュー「Subnets」を選択します。「Create Subnet」から次2つのサブネットを作成します。

- **Subnet 1**
  Name tag：Private subnet1
  VPC：先ほど作成したVPC
  Availability Zone：us-east-1b
  CIDOR Block：10.0.1.0/24
- **Subnet 2**
  Name tag：Private subnet2
  VPC：先ほど作成したVPC
  Availability Zone：us-east-1c
  CIDOR Block：10.0.2.0/24

また、VPC用のセキュリティグループを作成しておきます。セキュリティグループの設定例は**表2**のとおりです。

RDSのセットアップ

次はRDSをセットアップします。先にDB Subnet Groupを作成しておきます。Availability Zoneは、先ほど作成した「Private subnet1」「Private subnet2」を選択します。

(1) RDS Dashboardの中央メニュー「Resources」から「Launch a DB Instance」ボタンを押下します
(2)「Step 1: Engine Selection」からmysqlを選択します
(3)「Step 2: Production ?」から、「Yes, use

Multi-AZ Deployment and Provisioned and Provisioned IOPS Storage as defaults while creating this instance」を選択し、「Next Step」ボタンを押下します

(4) 「Step 3: DB Instance Details」で各種パラメータを選択します。今回は次のようにします

DB Engine：mysql
Licent Model：general-public-license
DB Engin Version：5.6.13
DB Instance Class：db.t1.micro
Multi-AZ Deployment：Yes
Auto Minor Version Upgrade：No
Allocated Storage：5GB
Use Provisioned IOPS：チェックなし
DB Instance Identifer：sd-wordpress
Master Username：wordpress
Master Password：任意で指定

(5) 「Step 4: Additional Config」で各種パラメータを選択します。今回は次のようにします

Database Name：wordpress
Database Port：3306
Choose a VPC：先ほど作成したVPC
DB Subnet Group：先ほど作成したDB Subnet Group
Publicly Accessible：No
Availability Zone：No Preference
Option Group：default.mysql5-6
Parameter Group：default.mysql5.6
VPC Security Group(s)：先ほど作成したVPCのセキュリティグループ

(6) 「Step 5: Management Options」で各種パラメータを選択します。今回は一時的な環境ですのでバックアップを取らない設定にします

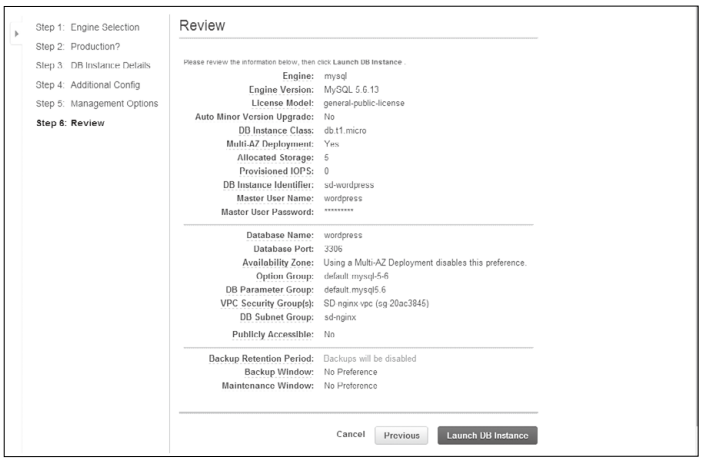
Enable Automatic Backups：No
Maintenance Window：No Preference

(7) 「Step 6: Review」の結果、問題なければ「Launch DB Instance」ボタンを押下します(**図4**)。

VPCのインスタンスを起動

先ほど作成したAMIを使って新規にインスタンスを起動します。EC2 Dashboardから「AMIs」を選択し、起動するAMIを選択して「Launch」ボタンを押下します。以降の作業はEC2インスタ

▼図4　RDSの確認画面



▼表2　VPCのセキュリティグループ

Inbound

Type	Protocol	Port Range	Source	備考
SSH	TCP	22	自身の接続元IPアドレス	—
HTTP	TCP	80	0.0.0.0/0	—
MYSQL	TCP	3306	10.0.0.0/16	RDSへの接続用

Outbound

Type	Protocol	Port Range	Source	備考
All traffic	All	All	0.0.0.0/0	—

ンスの起動とほぼ同じですが、起動先をVPCにする点が異なります。VPCのSubnetはPublic subnet1を指定します。また、セキュリティグループはVPC用に作成してあるものを選択します。

起動時にPublic IPかElastic IPを付与し、インターネットからリモートログインできるようにしましょう。

WordPressのインストール

今度はWordPressをセットアップしましょう。あらかじめWordPress用のデータベースをRDSへ作成しておいてください。

(1) 最新の日本語版WordPressをダウンロードします(執筆時点の最新版は3.9.1)。WordPressはその時点での最新版を使用するようにしてください

```
# wget http://ja.wordpress.org/ ➪
wordpress-3.9.1-ja.tar.gz
```

(2) ダウンロードしたWordPressを展開します

```
# tar -xzC wordpress-3.9.1-ja.tar.gz / ➪
usr/share/nginx/html
```

(3) 権限を変更します

```
# chown -R nginx:nginx /usr/share/ ➪
nginx/html
```

(4) wp-config.phpを用意します

```
# cp /usr/share/nginx/html/wp-config- ➪
sample.php /usr/share/nginx/html/wp- ➪
config.php
# vi /usr/share/nginx/html/wp-config.php
  define('DB_NAME', 'wordpress');
  define('DB_USER', 'wordpress');
  define('DB_PASSWORD', '先ほど設定した ➪
パスワード');
  define('DB_HOST', 'RDSのホスト名');
```

(5) wp-config.phpのパーミッションを変更します

```
# chmod 400 /usr/share/nginx/html/wp- ➪
config.php
```

(6) ブラウザから「http://xxx.amazonaws.com/wp-admin/」(xxxはインスタンスごとに異なる)にアクセスして、WordPressのインストールに必要な情報をWebから入力します(図5)。

WordPressの初期ログイン画面が表示されれば、インストールは成功しています。ここであらためてこのインスタンスのAMIを作成しておくと良いでしょう。

ELBのセットアップ

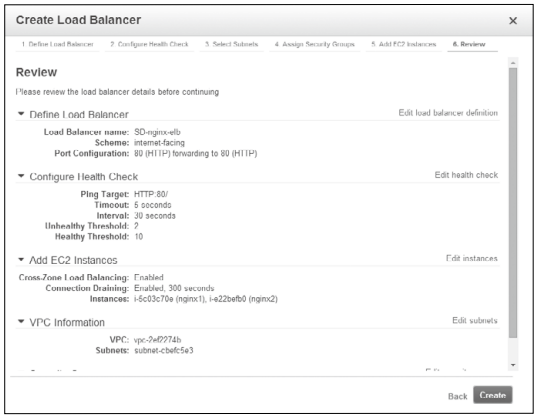
最後にELBの設定を行います。インスタンスは2台以上起動させ、WordPressがRDSにアクセス可能な状態を作ってください。

(1) マネジメントコンソールから「EC2」を選択します
(2) EC2 Dashboardの左メニューで「Load Balancers」を選択します
(3) 「Create Load Balancer」をボタンを押下します

▼図5 WordPressの情報入力画面

(4)「1.Define Load Balancer」では次の値を選択します
Load Balancer name：SD-nginx-elb
Create LB Inside：作成したVPC
(5)「2.Configure Health Check」では次の値を選択します
Ping Protocol：HTTP
Ping Port：80
Ping Path：/
Response Timeout：5
Health Check Interval：30
Unhealthy Threshold：2
Healthy Threshold：10
(6)「3.Select Subnets」では「Public subnet1」を選択します
(7)「4.Assign Security Groups」ではVPCのセキュリティグループを選択します
(8)「5.Add EC2 Instances」では先ほど作成したインスタンスをメンバーとして追加します
(9)「6.Review」で問題なければ「Create」ボタンを押下します。Load Balancerが作成されます(**図6**)

▼図6　ELBの確認画面



▼図7　WordPressのサンプルページ

ブラウザでELBのDNS nameに対してアクセスし、WordPressの画面が表示されればOKです(**図7**)。

まとめ

今回の構成は、最終的にWordPressが稼動するインスタンスを複数台構成にしました。が、インスタンス間のコンテンツ同期は考慮されていないので工夫が必要です。具体的には、「rsyncを使った同期をする」「NFSサーバ用のインスタンスを追加で用意する」「s3fsを使って、Amazon S3を外部ストレージとして使う」など、コンテンツの同期／共有をもう一工夫する必要があります。

また、インターネット環境にサーバを公開する際には、セキュリティ面での注意も払っていただければと思います。具体的には、「最新のWordPressを利用する」「WordPress管理画面に対して、アクセス制御や、SSLの利用を検討する」、「各種コンテンツ、設定ファイルの権限を適切に設定する」などが挙げられます。

最後に

本章ではAWS上でのNginx使用例を挙げてみました。クラウドには便利なサービスが多数用意されており、これらをうまく使うことで、クラウドならではの環境が構築できます。

クラウドは便利ですが、使い終わったら不必要なサービスは利用を停止しましょう。くれぐれもクラウド破産しないようにご注意を！SD

第2特集

知っているようで知らない「DHCPサーバの教科書」

押さえておきたい基礎の基礎

「DHCPは縁の下の力持ち」とよく言われます。wifiノートPCも、スマホも、タブレットもTCP/IPでつながらなければ何もできません。ただの箱(というか板)になります。TCP/IPでさまざまなIT機器がつながることで、インターネットのいろいろなサービスを享受できます。その大事な「インターネットのつながるしくみ」を支えている技術の1つが「DHCP(Dynamic Host Configuration Protocol)」です。ネットワークに接続するときにIPアドレスを割り当てるしくみです。この技術のおかげでどんなにネットワーク構築が楽になったことか……。読者の多くの皆さんは知らないかもしれませんが、瞬く間にDHCPが普及していきました。さて基礎の基礎を新しい観点で復習する本特集は、DHCPのしくみを解説したあと、実際に手を動かしてコマンドを入力して確認したり、クラウド環境でどのようにDHCPが使われているのか、ユーザはどのように使うのかといったところまで解説します。古くても新しいDHCPを学んでください。

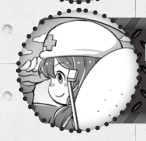
CONTENTS

Writer：中井 悦司


Part 1 その役割を確認 DHCPをご存じですか？ 64


Part 2 じっくり押さえる基礎の基礎 DHCPの舞台裏 69


Part 3 動作原理を実機で確認してみませんか！ DHCPサーバの構築・運用 74


Part 4 クラウド＆スマホを例として考える 「今どき」のIPアドレス管理 81

Writer：鶴長 鎮一


Practical Column 1 DHCPとDNSの連携 86


Practical Column 2 Amazon VPCのDHCPオプションで設定を変更するには 91

イラスト：高野涼香

Part 1 その役割を再確認! DHCPをご存じですか?

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

DHCPは縁の下の力持ち。ネットワークに接続されるさまざまなアプライアンスにIPアドレスを供給し、管理の礎となります。さらには、ネットワークのレイアウトを決めるものでもあります。ネットワークの根源を決め、ユーザに継続的かつ安定したサービスを提供するためになくてはならないDHCPサーバのしくみをしっかり学んでおきましょう!

DHCPに見るネットワーク管理の変遷

本誌読者のみなさんであれば、もちろん(!)、DHCP(Dynamic Host Configuration Protocol)という言葉を聞いたことはあるでしょう。ノートPCをオフィスや家庭のネットワークに接続すると、自動でIPアドレスを割り当ててくれる「あの」機能です。ひと昔前、オフィスのPCと言えば、共有のデスクトップPCが各部署に数台ずつ割り当てられている時代もありました。そのような環境であれば、オフィスのIT管理者がそれぞれのPCに決め打ちでIPアドレスを割り当てて、PC上でIPアドレスを個別設定することもできます。――「IPアドレス管理台帳」というExcelシートを見ながら、PCのネットワーク設定にいそしむ管理者の姿が目に浮かびます……。

一方、最近のように、各自が自由にPCをネットワークに接続する環境では、管理者が個別にIPアドレスを割り当てるのは、現実的ではありません。DHCPサーバを用意して、IPアドレスの割り当てを自動化することが必要です。これで、いちいちIT管理者にお願いしてIPアドレスを用意してもらう必要はなくなります。現代の環境では、一般の利用者にとってのDHCPサーバは、もはや空気のような存在かもしれません。

とはいえ、DHCPサーバを構築・設定するIT管理者には、当然ながら、DHCPをはじめとするネットワークの正しい理解が求められます。誰かが勝手に、検証用のDHCPサーバをオフィスネットワークに接続してしまい、本来のDHCPサーバが機能しなくなるなど、DHCPならではのトラブルにも対応する必要があります。また、自宅のブロードバンドルータやモバイルルータでもDHCPが使われており、IT管理者ならずとも、DHCPの理解が求められる時代になったとも言えるでしょう。

さらに、DHCPの使い道は、IPアドレスの自動割り当てにはとどまりません。Part3で説明するように、データセンタの数百台、数千台のサーバに対して、「固定IPアドレス」を設定するためにDHCPを利用することも可能です。KickStartに代表される、サーバの自動インストール機能にもDHCPが関わります。

本特集では、DHCPを基礎から学びながら、ネットワークのしくみをより深く理解することを目指します。伝統的なIPアドレス割り当て用途から始まり、サーバ仮想化、そして、クラウド環境でのDHCPなど、「今どき」のDHCPのあり方についても解説を進めます。このPart1では、IPネットワークの基礎とDHCPの概要を振り返り、次のPart2では、DHCPの技術的なしくみを徹底的に解説します。Part3でDHCPサーバの構築方法を学び、最後のPart4では、クラウド&スマホにおけるIPアドレス管理を説明します。

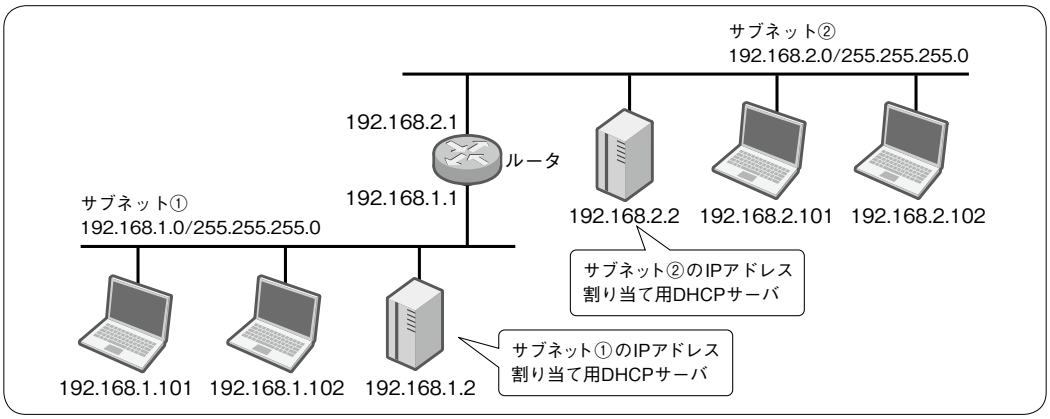
IPネットワークのしくみとDHCPの役割

冒頭で触れたように、DHCPの基本的な役割は「IPアドレスの割り当て」です。それでは、DHCPが具体的にどのような情報を提供するかご存じでしょうか? 「192.168.1.101」のようなIPアドレスだけをPCに割り当てても、ネットワーク通信はできません。一般にDHCPサーバは、IPアドレス、サブネットマスク、デフォルトゲートウェイの「基本3点セット」、そして、DNS(Domain Name System)サーバのIPアドレス、ドメイン名などの追加情報を提供します。

基本3点セットの詳細は、本誌2014年6月号の第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」(Part4)を見ていただくことにして、ここではポイントを簡単に説明します。まず、IPネットワークは、**図1**のように、複数のサブネットがルータで相互接続された構造になります。それぞれのサブネットには、「ネットワークアドレス」と「サブネットマスク」が割り当てられており、これらから、そのサブネットで利用できるIPアドレスの範囲が決まります。

たとえば、「ネットワークアドレス/サブネットマスク」が「192.168.1.0/255.255.255.0」のサ

▼図1　IPネットワークの基本構成



Column

どっこい生きてる固定IPアドレス

多くのオフィスでは、DHCPの利用は「当たり前」になっていますが、今でもDHCPを使用せずに固定IPアドレスをPCに割り当てるオフィスもあります。特定の業務に従事する従業員に対して、その業務専用のPCを割り当てる場合、とくに機密情報を扱う業務などの場合にその傾向があるようです。

これは、PCとIPアドレスを1対1で紐(ひも)づけることにより、各端末から接続できるネットワークを制限したり、何か問題が起きた時にIPアドレスから端末を特定するなどの意図があるものと思われます。ただ、このような現場でも、時代とともに管理対象の端末が増えていき、手作業でのIPアドレスの割り当てが限界で困っているという話を聞くこともあります。

もちろん、現在のネットワーク技術を駆使すれば、DHCPサーバを導入したうえで、端末ごとのアクセス制限を実現することもできます。しかしながら、そのほかの管理システムが端末のIPアドレスが固定されている前提で作られていると、DHCPへの切り替えも簡単にはいきません。

一般にITシステムの設計では、将来の利用環境の変化や技術動向を見据えることが大切だと言われます。「IPアドレスの割り当て」といった、一見すると単純な事柄にも深い洞察が求められるのが、サーバ、ネットワークなど、インフラ技術の醍醐味であり、エンジニアの真の腕の見せどころとも言えるでしょう。

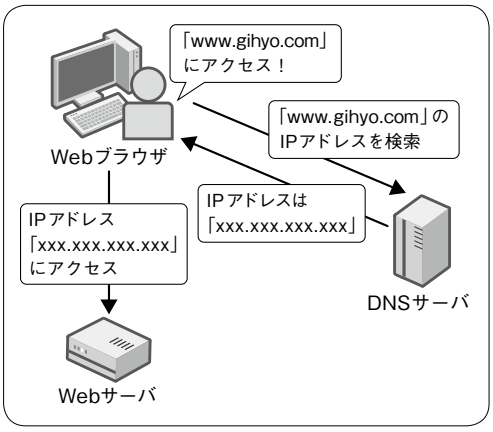
ブネットでは、「192.168.1.1」～「192.168.1.254」が機器に割り当て可能なIPアドレスになります。

また、**図1**からわかるように、サブネット①のPCがサブネット②にパケットを送信する際は、いったんルータのIPアドレス「192.168.1.1」にパケットを送ったうえで、その先の配送処理をルータに依頼する必要があります。このように、ほかのサブネットにパケットを送信する際に使用するルータのIPアドレスが「デフォルトゲートウェイ」になります。

PCがパケットを送信する際は、自身のIPアドレスとサブネットマスクをもとにして、送信先のIPアドレスが自身と同じサブネットのものかを判別します。同じサブネットであれば、相手の機器に直接にパケットを送信して、異なるサブネットであれば、ルータのIPアドレスに向けてパケットを送信します。以上から、「IPアドレス、サブネットマスク、デフォルトゲートウェイ」の3点セットをPCに設定することで、外部のサブネットと通信できるようになることがわかります。

DNSサーバのIPアドレスは、いわゆる「名前解決」に必要な情報です。名前解決のしくみは**図2**のようになります。PCのWebブラウザで「www.gihyo.jp」のようなFQDN(Fully Qualified Domain Name:完全修飾ドメイン名)を指定すると、所定のDNSサーバから対応するIPアドレスを取得したうえで、そのIPアドレスのサーバにアクセスします。

▼図2　DNSサーバによる名前解決



最後のドメイン名は、DHCPでIPアドレスを割り当ててもらうPC自身が属するドメインの名前になります。PC自身のドメイン名が「example.com」の場合、「www」のようにドメイン名を省略してアクセスしようとすると、自身のドメイン名を付加した「www.example.com」のIPアドレスをDNSサーバに問い合わせます。自分と同じドメイン名を持つサーバに対しては、ドメイン名を省略した「短縮名」でアクセスできるようにするしくみになります。

少し説明が長くなりましたが、これらネットワーク通信に必要な設定情報をPCに通知するのがDHCPサーバの役割です。DHCPがない環境では、これらの情報をすべて手動でPCに設定していく必要がありました。一方、DHCPの環境では、PC上で稼働する「DHCPクライアント」がDHCPサーバと通信して、これらの情報を自動で取得・設定してくれることになります。

――え？　「DHCPクライアントがどうしてDHCPサーバと通信できるのかわからない」ですって？　確かに、DHCPクライアントは、ネットワークを設定する前のPCにおいて、DHCPサーバから必要な情報を取得する必要があります。そもそも、DHCPクライアントは、どのようにして、DHCPサーバを発見できるのか不思議な気もします。このあたりの詳細は、Part2で解説することにしましょう。

自宅にもあるDHCPサーバ

話が混みいってきましたので、少し話題を変えて、身近にあるDHCPサーバを見てみます。最も身近なものと言えば、みなさんの自宅にもある、「自宅ネットワーク」のDHCPサーバでしょう。古い話で恐縮ですが、**図3**は、15年ほど前の筆者の自宅ネットワーク環境です。当時は、電話回線を使ってインターネットに接続する「ADSL」を使用していました。NTTから借りたADSLモデムを自宅の電話線に接続して、その

先にルータ用のLinuxマシンを接続しています。

一般的には、ADSLモデムの先には、ルータ用Linuxではなく、普通のWindows PCを接続して、「PPPoE(PPP over Ethernet)クライアント」という専用ツールを起動します。PPPoEクライアントが電話回線の先にあるプロバイダのネットワークからIPアドレスを取得することで、インターネットに接続できるようになります。ただし、この方法では、1台のPCしかインターネットに接続できません。そこで、**図3**のように、NIC(Network Interface Card)が2つあるLinuxマシンを接続して、ルータ代わりに使用していました。

ルータ用LinuxでPPPoEクライアントを起動すると、ADSLモデムに接続したNICにIPアドレスが割り当てられます。これは、プロバイダが用意したグローバルIPアドレスで、インターネットと直接に通信できるIPアドレスです。もう一方のNICには、プライベートIPアドレス(192.168.1.1)を手動で割り当てて、スイッチングHUBに接続します。

これで、自分が使用するPCをスイッチングHUBに好きなだけ接続することが可能になります。ルータ用LinuxをDHCPサーバとして構成しておき、プライベートIPアドレスをPCに割り当てるようにしてあります。**図1**では、ルータとは別にDHCPサーバが用意されていましたが、ここでは、ルータとDHCPサーバが1台のLinuxマシンに同居した形になります。ルータ用LinuxのプライベートIPアドレス(192.168.1.1)が各PCのデフォルトゲートウェイになります。

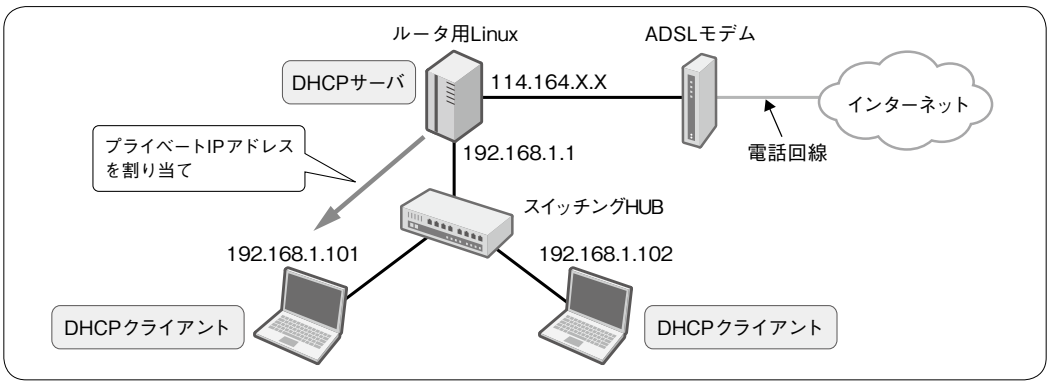
この環境でPCがインターネットに接続する場合は、ルータ用LinuxがプライベートIPアドレスとグローバルIPアドレスの変換処理を行います。一般に「IPマスカレード」と呼ばれる方式で、すべてのPCがルータ用Linuxのグローバル IPアドレス(114.164.X.X)を代表アドレスとして共有する形でインターネットに出ていきます。

ちなみに、現在の筆者の自宅ネットワークは、**図4**のような構成になっています。ずいぶんと複雑になった気もしますが、本質的には**図3**と変わりはありません。今は、NTTの光回線を使用していますので、NTTから借りた「ひかり電話ルータ」がインターネットとの接続口になります。このルータには、DHCPサーバやIPマスカレードなど、先のルータ用Linuxと同等の機能が入っていおり、この先にスイッチングHUBを介して、自分のPCを接続します。ファイルサーバとプリンタは、IPアドレスが変化すると困るので、DHCPを使用せずに固定的にIPアドレスを設定しています。

モバイルルータとテザリングの違い

最近では、自宅でも、モバイルルータでインターネットにアクセスする方も多いかもしれま

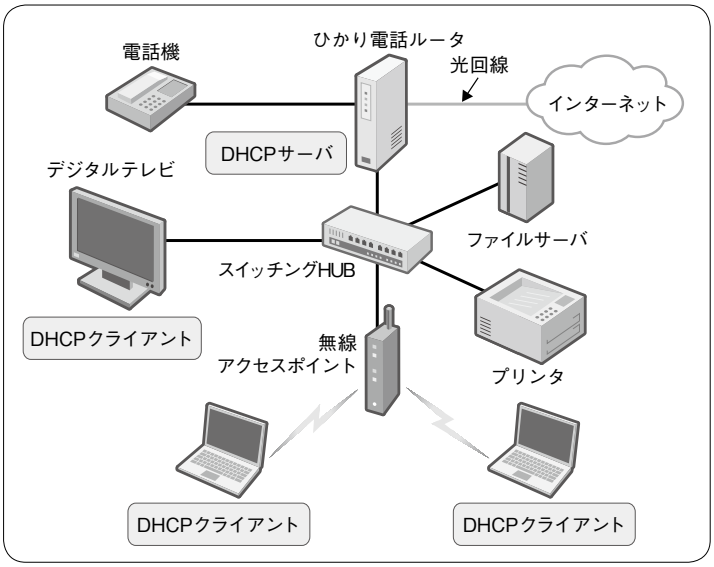
▼図3　筆者の自宅ネットワーク構成(15年前)



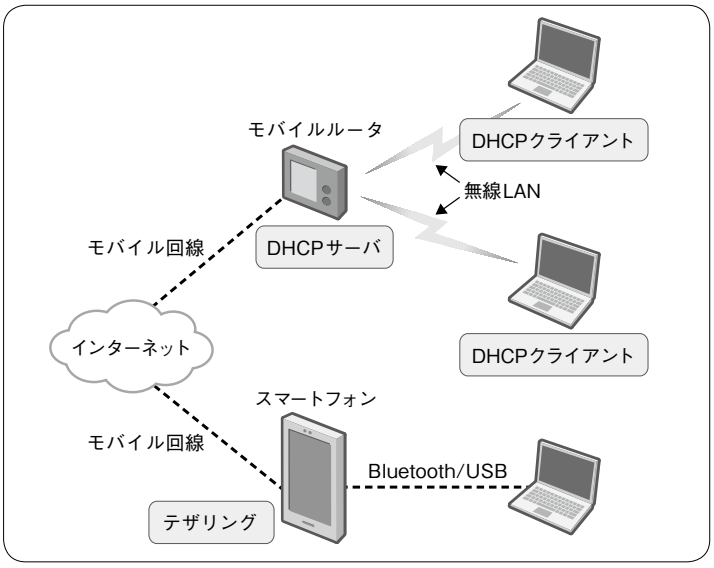
せん。モバイルルータは、LTEや3Gなどのモバイル回線でインターネットに接続するわけですが、そのほかのしくみは、先のルータ用Linuxと大きな違いはありません。モバイルルータの内部にDHCPサーバやIPマスカレードの機能が内蔵されており、**図5**(上)のように、無線LANを通じて複数のPCにプライベートIPアドレスを割り当てます。

一方、スマートフォンを利用した「テザリング」では、しくみが異なる場合があります。スマートフォンの機種によっては、モバイルルータとして利用できるものもあり、その場合は、**図5**(上)のモバイルルータをスマートフォンに置き換えた形になります。一方、**図5**(下)のように、BluetoothやUSBケーブルでスマートフォンを接続する場合、スマートフォンとPCの間は独自規格での通信が行われます。PCには専用のドライバアプリケーションが必要です。スマートフォン自体がPCに付属のモバイル通信カードのように振る舞うと考えるとわかりやすいでしょう。

▼図4　筆者の自宅ネットワーク構成(2014年)



▼図5　モバイルルータとテザリングの違い



まとめ

Part1では、DHCPの基礎となるIPネットワークの知識を整理したうえで、身近なDHCPサーバの例を紹介しました。本文の例のほかには、Wi-Fiのアクセスポイント(ホットスポット)を利用する場合なども、DHCPによって、PCにIPアドレスが割り当てられます。アクセスポイント用の機器の中で、DHCPサーバが動いているわけです。もはや、DHCPがない生活は考えられない時代と言えるでしょう。

次のPart2では、DHCPクライアントがどのようにDHCPサーバと通信を行うのか、その技術的なしくみを解説します。SD

Part 2 じっくり押さえる基礎の基礎 DHCPの舞台裏

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

本章では、より具体的にDHCPのしくみに迫ります。まずはDHCPのプロトコルの中身を見てみましょう。IPアドレスがどのようにクライアントに渡されるのか順を追って説明していきます。その精巧なしくみを知ると、IPネットワークのしくみをより深く知りたくなるでしょう。

DHCPのプロトコルを徹底解説

ここからは、DHCPの技術的な解説へと進みます。Part2では、DHCPクライアントとDHCPサーバがやりとりするメッセージの内容、すなわち、「DHCPのプロトコル」を説明します。Part1でも触れたように、DHCPクライアントは、ネットワーク設定がなされる前の状態で、DHCPサーバとの通信を開始する必要があります。このあたりのしくみもここで解明しておきます。

DHCPでIPアドレスの割り当てを行うにあたり、DHCPクライアントとDHCPサーバは、UDP（User Datagram Protocol）のパケットで、**表1**の4つのメッセージをやりとりします。DHCPサーバとDHCPクライアントは、それぞれ、UDPポート「67」と「68」でパケットを受信します。PCをネットワークに接続すると、PCで稼働しているDHCPクライアントは、DHCP-DISCOVERを最初に送信して、DHCPサーバにIPアドレスの割り当てを要求します。このDHCP-DISCOVERから始まる処理の流れを前半と後半に分けて、詳細に説明します。

▼表1 DHCPの主要メッセージ

メッセージ名	送信元	説明
DHCP-DISCOVER	クライアント	DHCPサーバを探してIPアドレスの割り当てを要求
DHCP-OFFER	サーバ	割り当て候補のIPアドレスを提示
DHCP-REQUEST	クライアント	候補として提示されたIPアドレスの使用を要求 もしくは、IPアドレスの有効期限の延長を要求
DHCP-ACK	サーバ	クライアントの要求を受け入れ

IPアドレス割り当ての流れ（前半）

まず、前半の流れを図示すると、**図1**のようになります。Part1の**図1**において、サブネット①のPCが、IPアドレス「192.168.1.2」のDHCPサーバとやりとりしていると考えてください。一般にUDPで通信するためのパケットには、ヘッダ部分に「送信元／宛先MACアドレス」「送信元／宛先IPアドレス」「送信元／宛先ポート番号」の情報が付随しています。これらの情報を明記してありますので、注意深く見てください。DHCPクライアントとDHCPサーバのMACアドレスは、例として、それぞれ、「0000.0000.1111」および「0000.0000.2222」としてあります。

DHCP-DISCOVERのヘッダ情報を見ると、なかなか面白い内容であることがわかります。まず、送信元であるDHCPクライアントは、まだIPアドレスを持っていませんので、送信元IPアドレスには便宜上「0.0.0.0」が入っています。そして、宛先MACアドレスと宛先IPアドレスには、「FFFF.FFFF.FFFF」と

「255.255.255.255」がセットされています。これらはともに、「ブロードキャストアドレス」と呼ばれます。

一般に、同じサブネットの機器は、スイッチングHUBなどの「L2スイッチ」で接続されていますが、L2スイッチの仕様として、宛先MACアドレスがブロードキャストアドレスのパケットはすべてのポートから同じパケットを送出して、サブネット内のすべての機器にパケットを送り届けるようになっています。さらに、L2スイッチからパケットを受け取った機器は、宛先IPアドレスが自身のアドレスかを調べて、自分宛のパケットだけを受け取ります。

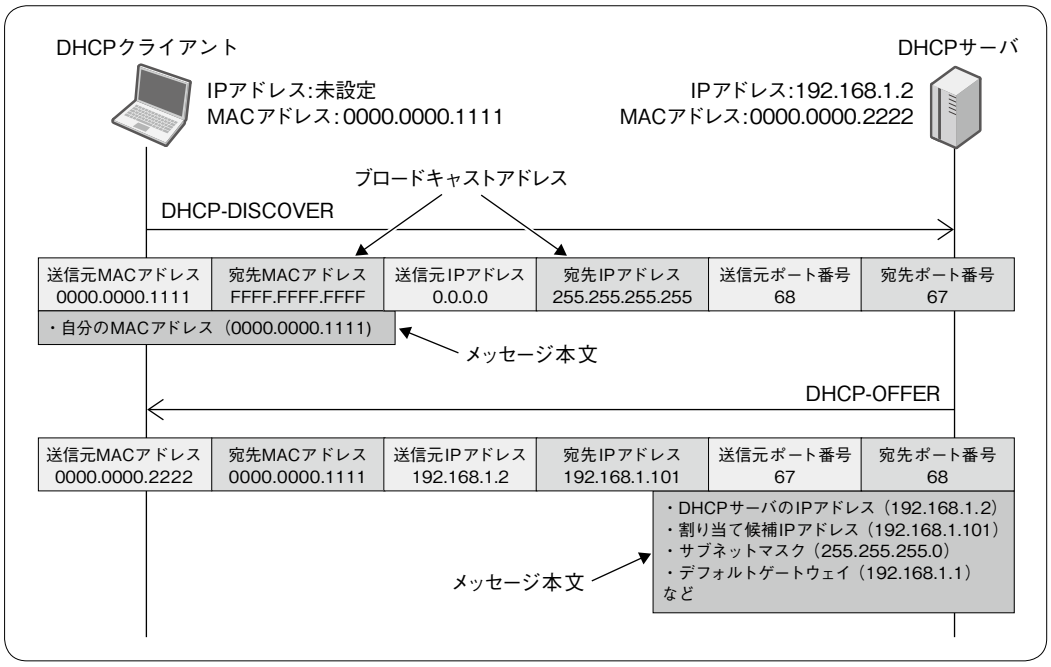
しかしながら、今回は、宛先IPアドレスもブロードキャストアドレスになっており、これは、実際のIPアドレスとは関係なくすべての機器が受信する約束になっています。結果として、DHCP-DISCOVERのパケットは、同じサブネットのすべての機器が受信します。ただし、宛先ポート番号「67」でUDPパケットを待ち受けているのは、DHCPサーバだけですので、実際にこのパケットを受け取って処理するのは、DHCPサーバだけになります。

DHCPクライアントは、DHCPサーバのIPアドレスを知らないわけですが、ブロードキャストアドレスを使用して、DHCPサーバにリクエストを届けることに成功しました。DHCP-DISCOVERのメッセージ本文には、誰からの要求かがわかるように、DHCPクライアントのMACアドレスの情報が埋め込まれています。

続いて、DHCPサーバは、自身の「IPアドレス管理台帳」から未使用のIPアドレスを探しだして、その1つを「割り当て候補IPアドレス」としてDHCPクライアントに返送します。これが、DHCP-OFFERになります。ここでは、例として「192.168.1.101」を返送するものとします。DHCP-OFFERのメッセージ本文の中には、サブネットマスクやデフォルトゲートウェイの情報、そして、DHCPサーバ自身のIPアドレスなども含まれます。

図1を見ると、DHCP-OFFERの宛先MAC

▼図1　DHCPによるIPアドレス割り当ての流れ（前半）



アドレスには、DHCP-DISCOVERに含まれていた、DHCPクライアントのMACアドレスが指定されています。したがって、このパケットは、L2スイッチによって、DHCPクライアントだけに届けられます。このようなパケットは、先ほどの「ブロードキャスト」に対して、「ユニキャスト」と呼ばれます。この時、宛先IPアドレスには、DHCPサーバが候補として選んだIPアドレスがセットされていますが、DHCPクライアントは、まだIPアドレスを持っていませんので、宛先IPアドレスは無視してこのパケットを受け取ります。

なお、IP電話機など、DHCPクライアントとして動作する機器によっては、IPアドレスが未設定の状態ではユニキャストのパケットを受け取れないものもあります。そのようなDHCPクライアントは、DHCP-REQUESTを送信する際に、返答パケットもブロードキャストで送信するようにDHCPサーバに要求を出します。DHCPにおけるブロードキャストとユニキャストの使い分けは、RFC-2131[1]に記載があります。

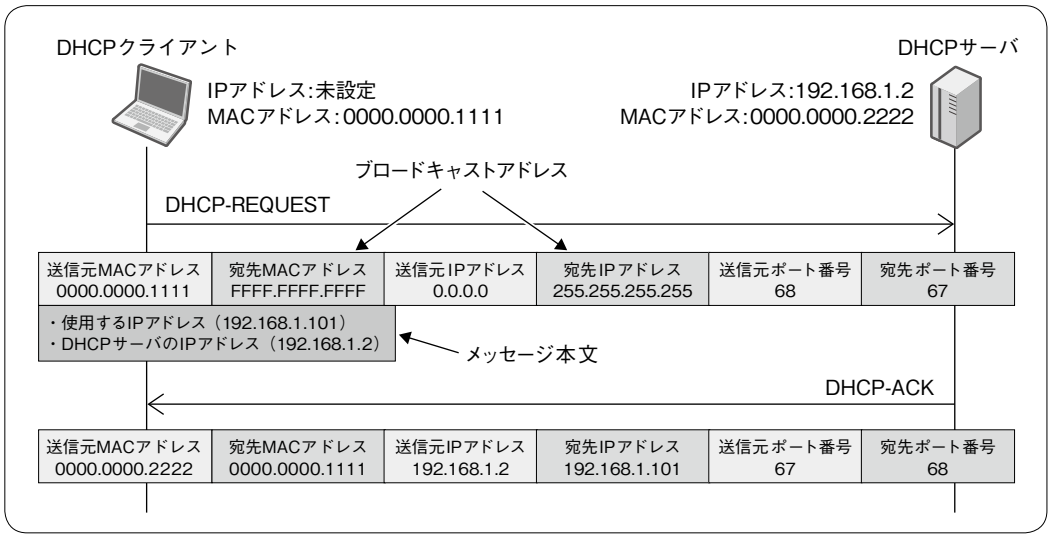
これで、DHCPクライアントは、自分が使用できるIPアドレスの候補を手に入れました。ただし、このIPアドレスを使用する前に、あらためて、DHCPサーバにこのIPアドレスの使用を宣言します。ここからが、後半の流れになります。

IPアドレス割り当ての流れ（後半）

後半の流れを図示すると、**図2**のようになります。DHCPクライアントは、再度、ブロードキャストアドレスを使用して、DHCP-REQUESTをDHCPサーバに送信します。DHCP-REQUESTのメッセージ本文には、候補として受け取ったIPアドレスとDHCPサーバのIPアドレスが含まれており、「所定のDHCPサーバに対して、このIPアドレスの使用を宣言する」という意味になります。これを受け取ったDHCPサーバは、受信確認を示すDHCP-ACKを先と同じユニキャストパケットで返送します。

これで、DHCPのやりとりは完了です。DHCPクライアントは、受け取ったIPアドレスを自身に設定するとともに、DHCPサーバは、自身の「IPアドレス管理台帳」に対して、このIPアドレスを「使用中」として記録します。どのクライアン

▼図2　DHCPによるIPアドレス割り当ての流れ（後半）



トが使用しているかわかるように、クライアントのMACアドレスも併せて記録しておきます。

ところで、この後半のやりとりは、一見すると余計にも思えます。前半のやりとりだけで、IPアドレスの割り当てを完了してはいけないのでしょうか？　実は、後半のやりとりには、2つの役割があります。

1つは、このサブネットに複数のDHCPサーバがあった場合への対処です。最初のDHCP-DISCOVERは、ブロードキャストで送信されるので、すべてのDHCPサーバがこれを受け取って、DHCP-OFFERを返送します。この場合、DHCPクライアントは、最初に受け取ったDHCP-OFFERのIPアドレスを採用して、DHCP-REQUESTを送信します。DHCP-REQUESTのメッセージ本文には、(実際にIPアドレスを採用した)DHCPサーバのIPアドレスが含まれていますので、該当のDHCPサーバのみが「IPアドレス管理台帳」の更新を行います。そのほかのDHCPサーバは、候補として提示したIPアドレスは採用されなかったものとして、未使用状態のままにします。つまり、実際には使用されていないIPアドレスについて、誤って、使用状態と記録することが防止されます。

もう1つは、IPアドレスの再割り当て処理に関係します。DHCPサーバがクライアントに割り当てたIPアドレスには、割り当ての有効期限があります。有効期限を過ぎたIPアドレスについては、DHCPサーバは、「IPアドレス管理台帳」

Column
DHCPサーバはサブネットごとに用意が必要？

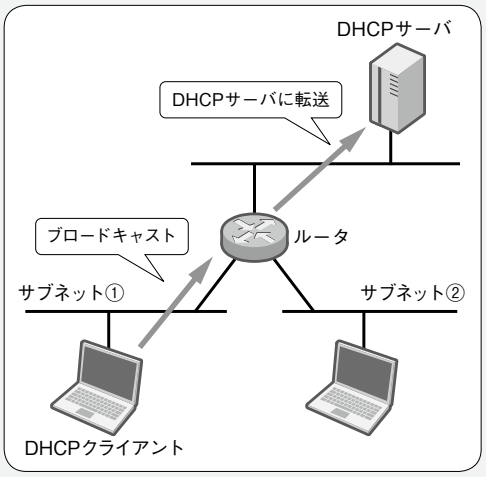
Part1の**図1**を見ると、サブネット①とサブネット②のそれぞれにDHCPサーバが用意されています。これはなぜでしょうか？　本文で説明したように、DHCPクライアントは、「ブロードキャストアドレス」を使用して、DHCPサーバにメッセージを送ります。ブロードキャストアドレス宛のパケットはルータにも到達しますが、ルータは、このパケットは他のサブネットには転送せず、そこで破棄します。つまり、DHCPクライアントとDHCPサーバのやりとりは、ブロードキャストパケットが到達する、1つのサブネット内に限定されることになります。そのため、サブネットごとにDHCPサーバを用意する必要があるわけです。

仮に、ルータがブロードキャストパケットをほかのサブネットに転送したら、何が起きるでしょうか？

次のサブネットの先にあるルータは、またその先のサブネットにパケットを転送します。つまり、1つのブロードキャストパケットがインターネット中に広がるという、非常に「はた迷惑」な状況になってしまいます。

とはいえ、複数部署にまたがるオフィスLANのように、多数のサブネットがある環境では、サブネットごとにDHCPサーバを用意して管理するのは面倒です。そこで、業務用のルータ(L3スイッチ)機器には、「DHCPリレー」という機能が用意されています(**図3**)。これは、DHCPサーバ宛のブロードキャストパケットを受信したルータは、事前に設定された特定のDHCPサーバに向けて、そのパケットを転送するという機能です。DHCPサーバからの返答は、再度、元のサブネットに送り返します。この機能は、それぞれの機器で独自に実装されているため、製品ごとに設定方法や細かな動作のしくみが異なるという課題もあります。

▼図3　DHCPリレーのしくみ



の記録を未使用状態に戻します。クライアントが停止した後、そのIPアドレスの記録が使用中のままで残らないためのしくみです。

一方、有効期限を超えて、割り当てられたIPアドレスの使用を続けたい場合もあります。そのような場合、DHCPクライアントは、有効期限が切れる前に、再度**図2**のDHCP-REQUESTを送信して、DHCPサーバからDHCP-ACKを受信します。これにより、DHCPサーバは、該当のIPアドレスを再割り当てしたものと認識して、有効期限を延長します。DHCPクライアントは、有効期限が切れるのを待って、再度、DHCP-DISCOVERからやり直す必要はありません。

そのほかのメッセージ

DHCPの基本的なやり取りは以上になりますが、参考として、DHCPで規定されているその他のメッセージ(**表2**)を説明しておきます。まず、DHCP-RELEASEは、クライアントが割り当てられたIPアドレスの使用を停止する際に、そのIPアドレスの解放を明示的にDHCPサーバに通知します。ただし、このメッセージの送信は、義務付けられているわけではありません。PCをシャットダウンするなどした場合、多くのDHCPクライアントは、DHCP-RELEASEを送信することなく、黙ってIPアドレスの使用を停止します。そして、再度PCを起動すると、DHCPクライアントは、DHCP-REQUESTを送信して有効期限の延長を行ったあとに、以前と同じIPアドレスの使用を継続します。

PCの停止中に割り当ての有効期限が切れた場合は、DHCPサーバ側で該当のIPアドレスは未使用状態に戻されます。仮に、PCがDHCP-REQUESTを送信した場合、DHCPサーバは、DHCP-NAKを返して、有効期限の延長を拒否します。そのため、クライアントは再度DHCP-DISCOVERからやりなおして、新たなIPアドレスを取得します。ただし、多くのDHCPサーバは、同じクライアントには、できるだけ同じIPアドレスを割り当てるように動作します。そのため、以前に使用していたIPアドレスがまだ未使用の場合は、ほとんどの場合、再度同じIPアドレスが割り当てられます。

また、DHCPサーバは、IPアドレスの設定情報以外に、さまざまな追加情報を提供することができます。すでにIPアドレスを取得したクライアントがこれらの追加情報を要求する際は、DHCP-INFORMを使用します。DHCPで提供可能な情報は、RFC-2132[2]で定義されています。

▼表2　DHCPのそのほかのメッセージ

メッセージ名	送信元	説明
DHCP-RELEASE	クライアント	IPアドレスの解放を通知
DHCP-NAK	サーバ	クライアントの要求を拒否
DHCP-INFORM	クライアント	IPアドレス以外の設定情報を要求

まとめ

Part2では、DHCPクライアントとDHCPサーバのやりとりを詳細に解説しました。ブロードキャストとユニキャストの使い分けなど、意外と複雑なしくみに感じられたかもしれません。当たり前のように使っているDHCPですが、そのしくみを理解するには、IPネットワークの深い知識が必要となります。この機会にもう一度、ネットワークの基礎を見なしておくとよいでしょう。

次のPart3では、実際にDHCPサーバを構築・運用する方法を説明します。さらに、DHCPのメッセージのやりとりの様子を実機で確認してみましょう。SD

●参考文献
[1]「Dynamic Host Configuration Protocol」
http://www.ietf.org/rfc/rfc2131.txt
[2]「DHCP Options and BOOTP Vendor Extensions」
http://www.ietf.org/rfc/rfc2132.txt

Part 3

動作原理を実機で確認してみませんか! DHCPサーバの構築・運用

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

本Partではいよいよ DHCP サーバを自分で構築してみます。難しいことは少なく、作りあげる予定のネットワークに必要な項目を挙げていき、実際に手を動かして作ってみます。設定ファイルも掲載されているサンプルを参考に自分で書いて、まずはやってみてください。体で覚えたほうが理解がいっそう深まります。

KVM仮想マシンで実験環境を構築

Part3では、Red Hat Enterprise Linux(RHEL) 6.5を利用してDHCPサーバを構築します。ただし、構築手順を説明するだけでは面白くありませんので、DHCPクライアントとのメッセージのやりとりなど、Part2で学んだ内容を実際に確認しながら、DHCPの理解をさらに深めることを目指します。とくにここでは、物理サーバ1台で試せるように、Linux KVMの仮想マシン環境を利用して、DHCPクライアントとDHCPサーバのそれぞれを仮想マシンとして用意します。自分専用の環境で、思う存分にDHCPのしくみを研究していきましょう。

具体的な構成は、**図1**のとおりです。RHEL6でLinux KVMの環境を構築すると、デフォルトで「default」という名称の仮想ネットワークが用意されます。ここでは、DHCPの実験を行うために、「private」という仮想ネットワークを追加しています。DHCPクライアントとDHCPサーバの仮想マシンは、それぞれ、「client01」と「dhcp01」という名称で作成して、「default」と「private」の両方の仮想ネットワークに接続します。ホストLinuxから仮想マシンにログインする際は、「default」側のネットワークを使用します。

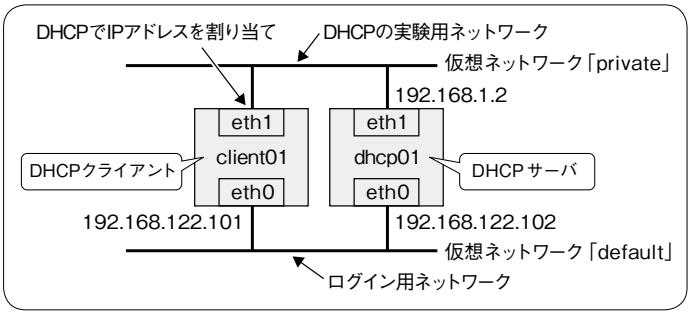
それでは、**図1**の環境の構築手順を簡単に説明します。はじめに、物理サーバにRHEL6を導入して、Linux KVMの環境を用意します。この部分の手順はWeb記事[1]などを参考にしてください。続いて、仮想ネットワーク「private」を追加します。**リスト1**のファイル「private.xml」をカレントディレクトリに作成して、次のコマンドを実行します。

```
# virsh net-define private.xml
# virsh net-autostart private
# virsh net-start private
```

このあとは通常の手順で仮想マシン「client01」と「dhcp01」を作成して、ゲストOSとしてRHEL6.5を「基本サーバー」の構成で導入します。この段階では仮想NICは「eth0」のみを作成して、仮想ネットワーク「default」に接続しておきます。「eth0」には**図1**に記載のIPアドレスを割り当てておきます。

最後に、それぞれの仮想マシンに仮想NIC「eth1」を追加して、

▼図1 KVM仮想マシンによる実験環境



仮想ネットワーク「private」に接続します。ホストLinuxのデスクトップで「仮想マシンマネージャー」を起動して、仮想マシンの構成画面を表示したら、左下の「ハードウェアを追加」を押します。追加するデバイスとして「Network」を選択したら、**図2**のように「ホストデバイス」と「デバイスモデル」を選択して、最後に「完了」を押します。これを「client01」と「dhcp01」のそれぞれで行います。

DHCPサーバを構築

それでは、仮想マシン「dhcp01」にログインして、DHCPサーバとして構成します。

▼リスト1　private.xml

```
<network>
  <name>private</name>
  <bridge name='virbr1'/>
</network>
```

▼図2　仮想NIC「eth1」の追加

▼図3　dhcp01での/etc/sysconfig/iptablesの追加行

```
（前半は省略）
-A INPUT -m state --state NEW -m tcp -p tcp --dport 22 -j ACCEPT
-A INPUT -m state --state NEW -m udp -p udp --dport 67 -j ACCEPT
-A INPUT -j REJECT --reject-with icmp-host-prohibited
-A FORWARD -j REJECT --reject-with icmp-host-prohibited
COMMIT
```

この行を追加

まず、仮想NIC「eth1」には、固定IPアドレス「192.168.1.2」を割り当てます。設定ファイル「/etc/sysconfig/network-scripts/ifcfg-eth1」を**リスト2**の内容で作成して、次のコマンドでインターフェースを有効化します。

```
# ifup eth1
```

続いて、iptablesで、DHCPクライアントからのメッセージの受信を許可します。設定ファイル「/etc/sysconfig/iptables」に**図3**の行を追加して、次のコマンドで設定変更を反映します。

```
# service iptables restart
```

Part2で説明したように、DHCPサーバはUDPポート「67」でメッセージを受信しますので、このポート宛のパケットの受信を許可しています。

最後に、DHCPサーバの本体となるソフトウェアを導入します。ここでは、RHEL6に同梱の「ISC DHCP」を使用します。次のコマンドで、「dhcp」のRPMパッケージと依存パッケージをまとめて導入します。

```
# yum -y install dhcp
```

これで、DHCPサーバとしての基本的な準備ができました。このあとは、設定ファイル「/etc/dhcp/dhcpd.conf」に使用する環境に合わせた設定を追加して、dhcpdサービスを起動します。ここでは、比較的シンプルな例として**リスト3**の内容を使用します。各行には説明用の行番号を入れてありますが、実際には、行番号は入力する必要はありません。設定ファイルを用意したら、次のコマンドでdhcpdサービスを起動します。

```
# chkconfig dhcpd on
# service dhcpd start
```

このとき、システムログ「/var/log/messages」には、**図4**のようなメッセージが記録されます。これは、「eth1」からのリクエストを受け付けて、「eth0」からのリクエストは無視することを示しています。**図1**にあるように、DHCPクライアントからのリクエストを受け付けるのは「eth1」ですので、意図どおりの動作になっています。

それでは、**リスト3**の内容を説明しながら、このような動作になる理由を説明します。まず、1行目の「ddns-update-style」は、DDNS(Dynamic DNS:動的DNS)と連携する際の設定ですが、DDNSを使用しない場合は、「お約束」として、「none」を指定します。

2行目の「default-lease-time」は、IPアドレス割り当ての有効期限(秒)です。ここでは、8時間に設定しています。3行目の「max-lease-time」は、割り当ての最大期間です。DHCPクライアントは、有効期限が切れる前に、DHCP-REQUESTを再送して有効期限の延長を行うことができますが、これ以上は延長を認めないという上限になります。ここでは、24時間に設定しています。

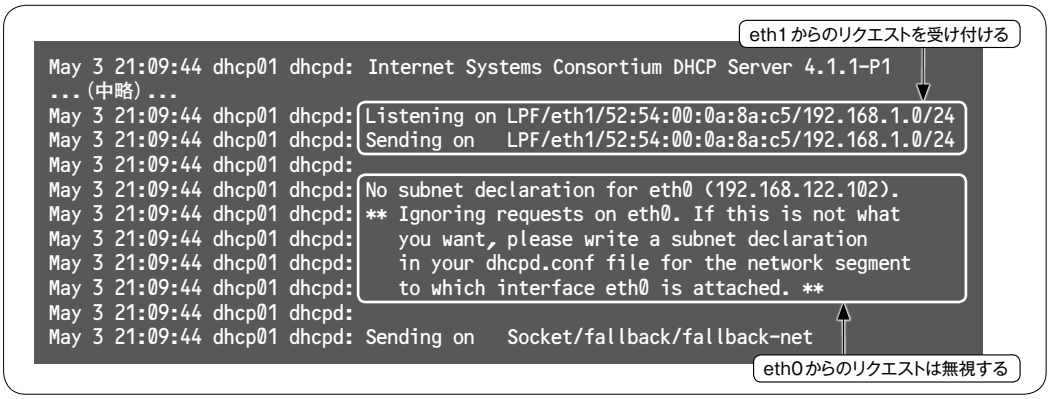
続いて、5行目～12行目のブロックは、ネットワークアドレスが「192.168.1.0/255.255.255.0」のサブネットからのリクエストに対する応答の設定です。今回の構成では、dhcp01は、「eth1」に対して、このサブネットのIPアドレスが振られていますので、「eth1」で受信したリクエストに対しては、この設定が適用されます。一方、「eth0」のIPアドレスが所属するサブネット「192.168.122.0/255.255.255.0」に対する設定は存在しませんので、「eth0」で受信したリクエストは無視されることになります。これが、**図4**で表示されたメッセージの理由になります。

Part2でも説明したように、一般には、サブネットごとにDHCPサーバが必要になりますが、今回のdhcp01のように、複数のサブネットに接続したサーバを利用すると、それぞれのサブネットに対してDHCPサーバとして機能させることもできます。その場合は**リスト3**の5行目～12行目に相当するブロックをサブネットごとに追加します。

6行目の「authoritative」は、DHCPクライアントがDHCP-REQUESTを用いて、「eth1」とは異なるサブネットのIPアドレスの使用をリクエストしてきた際の動作を指定します[注1]。この設定の場合は、DHCP-NAKを返して、DHCP-DISCOVERからIPアドレスの取得をやり直すように要求します。一方、「not authoritative」を指定すると、DHCP-NAKを返さずに、黙ってリクエストを無視するようになります。通常は、

注1) ほかのサブネットに接続していたPCを持ってきて、このサブネットに再接続した場合などが考えられます。

▼図4　DHCPサーバ起動時の/var/log/messagesの出力



「authoritative」を指定するのがよいでしょう。

7行目の「range」は、このサブネットに配布するIPアドレスの範囲を指定します。この範囲外のIPアドレスは、固定IPアドレスとして使用できます。8行目～11行目は、それぞれ、「サブネットマスク」「デフォルトゲートウェイ」「DNSサーバ」「ドメイン名」の情報になります。今回の**図1**の構成では、デフォルトゲートウェイやDNSサーバの設定は意味を持ちませんが、例として値を入れてあります。

DHCPクライアントで動作確認

続いて、DHCPクライアント「client01」を用いて、DHCPサーバの動作確認を行います。ここでは、DHCPのメッセージのやりとりをパケットレベルで確認するために、DHCPサーバで次のコマンドを実行しておきます。

```
# tcpdump -nlSvi eth1 dst port 67 or dst port 68
```

これは、tcpdumpコマンドで、DHCPの処理に伴うパケットの内容を画面表示するものです。Ctrl＋Cで停止するまで表示を続けるので、このままにしておきます。

次にclient01にログインして、仮想NIC「eth1」のIPアドレスをDHCPで設定するようにします。設定ファイル「/etc/sysconfig/network-scripts/ifcfg-eth1」を**リスト4**の内容で作成して、次のコマンドでインターフェースを有効化します。

```
# ifup eth1
```

このとき、先のtcpdumpには、Part2で解説した4つのメッセージのやりとりが記録されます。少し長くなりますが、重要なポイントなので、詳しく見ておきましょう。まず、**図5**は、前半の2つのメッセージです。DHCPクライアントから、ブロードキャストアドレス宛にDHCP-DISCOVERが送信されて、DHCPサーバからは、DHCP-OFFERにより、割り当て候補IPアドレス「192.168.1.101」がユニキャストで返信されていることがわかります。また、DHCP-OFFERには、**リスト3**で設定したサブネットマスクやデフォルトゲートウェイなどの追加情報も含まれています。

続いて、**図6**は、後半の2つのメッセージです。DHCPクライアントからは、再度、ブロードキャストアドレス宛にDHCP-REQUESTが送信されます。このとき、DHCP-OFFERで提示されたIPアドレスについて、その使用を要求しています。そのあと、DHCPサーバからは、ユニキャストでDHCP－ACKが返信されます。先ほどの追加情報は、DHCP-ACKにも含まれています。client01でifconfigコマンドを実行すると、「eth1」にはIPアドレス「192.168.1.101」が設定されていることがわかります。

割り当て中のIPアドレスの情報は、DHCPサーバ側では、テキストファイル「/var/lib/dhcpd/dhcpd.leases」に記録されています。また、DHCPクライアント側では、テキストファイル「/var/lib/dhclient/dhclient-eth1.leases」に、「eth1」に割り当て中のIPアドレスの情報が記録されます。それぞれの中身を見ると、割り当

▼リスト2　dhcp01のifcfg-eth1

```
DEVICE=eth1
BOOTPROTO=static
IPADDR=192.168.1.2
NETMASK=255.255.255.0
NM_CONTROLLED=no
ONBOOT=yes
```

▼リスト3　/etc/dhcp/dhcpd.conf

```
 1 ddns-update-style  none;
 2 default-lease-time 28800;
 3 max-lease-time     86400;
 4
 5 subnet 192.168.1.0 netmask 255.255.255.0 {
 6         authoritative;
 7         range 192.168.1.101 192.168.1.199;
 8         option subnet-mask          255.255.255.0;
 9         option routers              192.168.1.254;
10         option domain-name-servers 192.168.122.1;
11         option domain-name          "example.com";
12 }
```

て日時や有効期限が切れる日時なども記録されています。

ここで、先ほどのtcpdumpの出力を継続しながら、DHCPクライアントでインターフェースの無効化／有効化を行ってみます。client01で、次のコマンドを実行して下さい。

```
# ifdown eth1
# ifup eth1
```

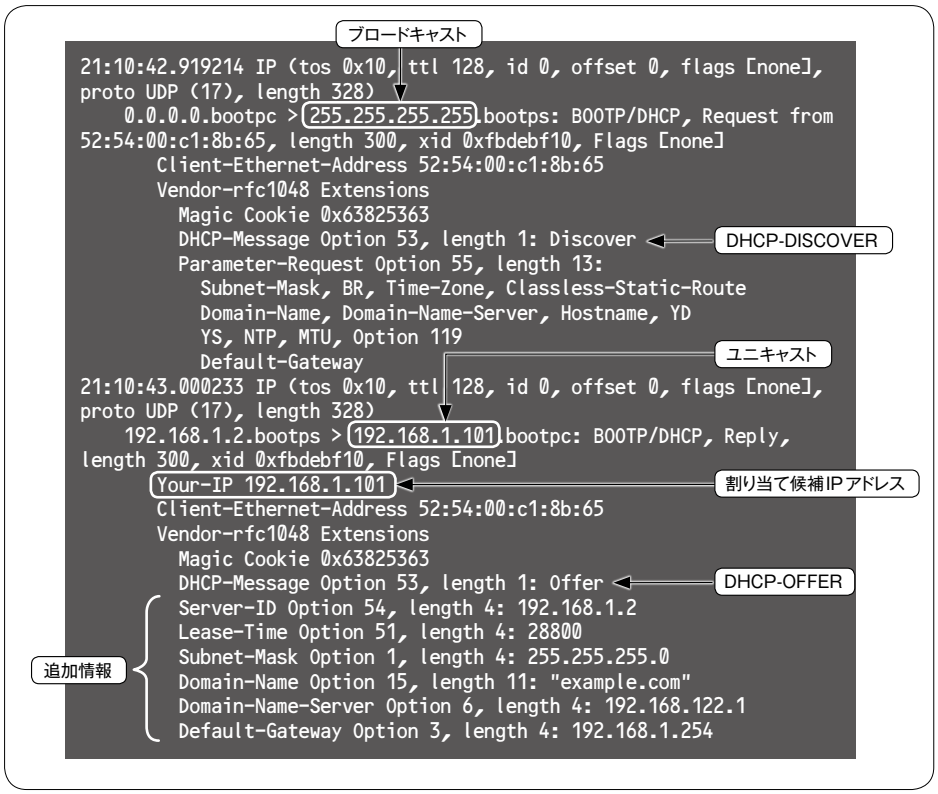
まず、ifdownコマンドで無効化した際は、DHCPのメッセージは発生しません。DHCP-RELEASEでIPアドレスを解放するほうが「お行儀」は良いのですが、Part2で説明したように、これは必須というわけではありません。そして、ifupコマンドで有効化すると、今度は**図6**と同様の後半部分のやりとりだけが発生します。DHCPクライアントは、先の「dhclient-eth1.leases」を見て、これまで使用していたIPアドレスを確認したうえで、再度、同じIPアドレスの使用をDHCP-REQUESTで要求している形になります。もう一度、DHCP-DISCOVERから始まるやりとりを確認したい場合は、ifdownコマンドで無効化したあとに、「dhclient-eth1.leases」を削除してください。

あるいは、コマンドで、明示的にDHCP-RELEASEを発行することもできます。「eth1」にIPアドレスが割り当てられた状態で、次のコマンドを実行すると、DHCPサーバにDHCP-RELEASEが送信されて、「eth1」のIPアドレスがなくなります。

```
# dhclient -r eth1 -lf /var/lib/dhclient/dhclient-eth1.leases
```

このあと、ifdown/ifupコマンドでインタフェースを有効化しなおすと、DHCP-DISCOVERから始まるIPアドレスの再取得が行われます。

▼図5　DHCPメッセージのやり取り（前半）



また、**リスト4**に「DHCPRELEASE=yes」の指定を追加すると、インターフェースを無効化する際に、DHCP-RELEASEが発行されるようになります。

DHCPで固定IPアドレスを設定

最後に、DHCPサーバの少し面白い使い方を紹介します。DHCPによって、クライアントごとに固定のIPアドレスを割り当てる方法です。これは、DHCPサーバの設定ファイルにおいて、クライアントのMACアドレスに対して、固定のIPアドレスを割り当てることで行います。

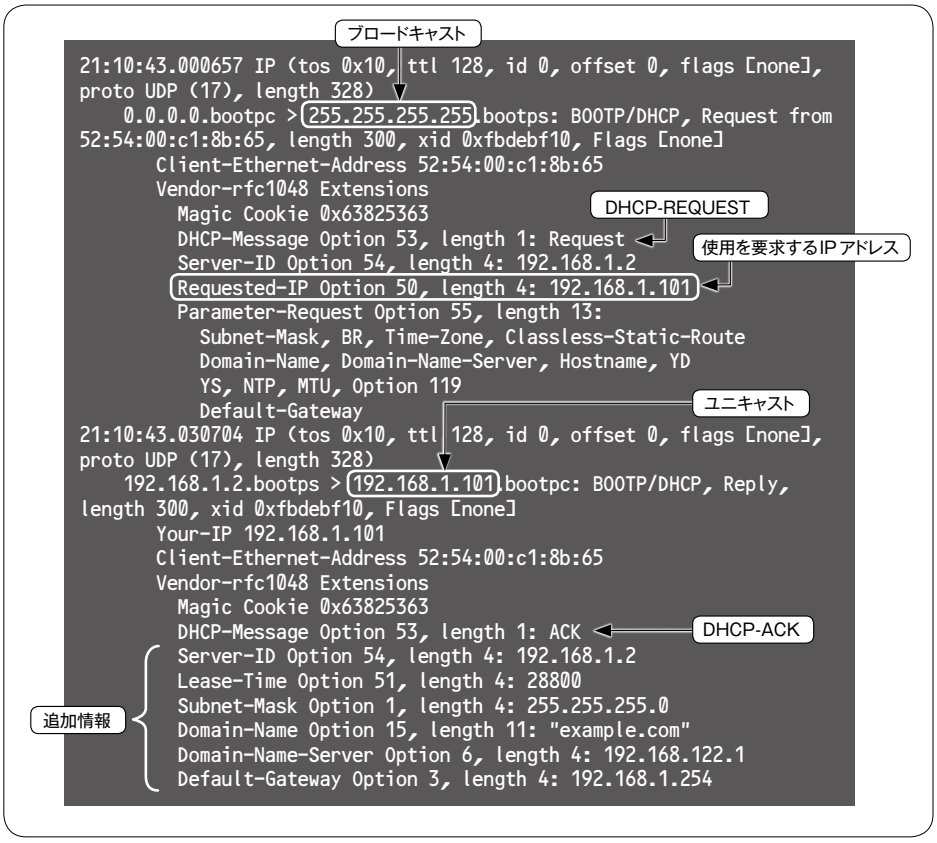
この際、DHCPクライアントには特別な設定は不要です。IPアドレス割り当ての要求を受けたDHCPサーバの側で、要求元のNICのMACアドレスを確認して、必ず、指定のIPアドレスを割り当て候補として提示します。結果として、このクライアントには、指定のIPアドレスが必ず割り当てられます。

この機能は、データセンタで大量のサーバをまとめて管理する際に活用できます。たとえば、新規に100台のサーバを追加して、それぞれに固定のIPアドレスを割り当てる場合を考えます。事前に各サーバのNICのMACアドレスを調べて、DHCPサーバにMACアドレスごとの固定IPアドレスの設定を追加しておきます。新規に追加したサーバは、すべて、**リスト4**のよう

▼リスト4　client01のifcfg-eth1

```
DEVICE=eth1
BOOTPROTO=dhcp
NM_CONTROLLED=no
ONBOOT=yes
```

▼図6　DHCPメッセージのやり取り（後半）



な設定でDHCPによるIPアドレスの割り当てを行うと、期待どおりの固定IPアドレスが設定されます。サーバごとに個別の設定を行う必要がなくなります[注2]。

DHCPサーバ側の具体的な設定例は、**リスト5**です。12行目～16行目のブロックをクライアントごとに指定します。13行目がクライアントのNICのMACアドレスで、14行目が対応する固定IPアドレスの指定です。15行目は、このクライアントに設定するホスト名になります。クライアントがRHEL6の場合、クライアント側でのホスト名を「localhost.localdomain」に設定しておくと、DHCPでIPアドレスを取得する際に、ここで指定したホスト名が適用されます。

なお、DHCPの設定を変更した際は、次のコマンドで設定変更を反映します。

```
# service dhcpd restart
```

まとめ

Part3では、RHEL6.5を使用して実際にDHCPサーバを構築したうえで、tcpdumpコマンドを利用して、DHCPのメッセージの詳細を確認しました。メッセージのやりとりを実際に確認することで、Part2の内容が、より深く理解できたと思います。これらメッセージのやりとりは、DHCPサーバのシステムログ「/var/log/messages」にも簡易的な内容が記録されています。動作確認の際は、そちらも確認するようにしてください。

Part4では、クラウド環境など、「今どき」なDHCPの使い方を紹介します。SD

注2）サーバごとのMACアドレスを調べるほうがたいへんと思うかもしれませんが、100台のサーバをまとめて発注する場合、MACアドレスのリストは、発注先のサーバメーカーが作成してくれます。

●参考文献
[1]「KVMで始めるプライベート・クラウドへの第一歩（第2回 KVMシステム導入手順）」
http://gihyo.jp/admin/serial/01/ibm_kvm/0002

▼リスト5　/etc/dhcp/dhcpd.conf

```
 1 ddns-update-style  none;
 2 default-lease-time 28800;
 3 max-lease-time     86400;
 4
 5 subnet 192.168.1.0 netmask 255.255.255.0 {
 6         authoritative;
 7         range 192.168.1.101 192.168.1.199;
 8         option routers             192.168.1.254;
 9         option domain-name-servers 192.168.122.1;
10         option domain-name         "example.com";
11
12         host client01 {
13                 hardware ethernet 52:54:00:C1:8B:65;
14                 fixed-address     192.168.1.99;
15                 option host-name  "client01";
16         }
17 }
```

Part 4 クラウド&スマホを例として考える「今どき」のIPアドレス管理

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

これまでのPartでDHCPサーバがどんな機能を持ち、ネットワークシステムを支えてきたのか理解するヒントをたくさん得られたと思います。本Partでは、まとめとして仮想環境・クラウド環境での使われ方、そしてスマートフォンにおけるIPアドレスの割り当てを解説します。

サーバ構築自動化にも役立つDHCP

最後のPart4では、変わり種のDHCPを紹介しながら、現代的なIPアドレス管理について解説を進めます。たとえば、Part3では、多数のサーバに固定IPアドレスを設定するためにもDHCPが使えることを説明しました。このほかには、KickStartでOSのインストールを自動化する際にもDHCPが活躍します。

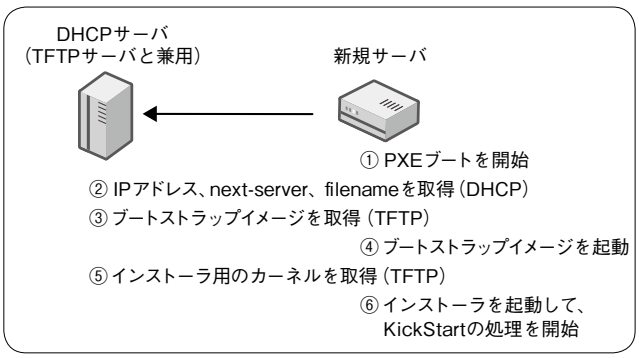
KickStartは、RHELやFedoraのインストーラである「Anaconda」が提供する機能です。GUI(Graphical User Interface)のインストール画面で設定項目を入力する代わりに、テキストファイル(KickStartファイル)に設定項目を記載しておくことで、インストール処理を自動化できます。さらに、PXEブートと組み合わせると、新しく購入したサーバをネットワークに接続して、電源を入れるだけで、OSのインストールを開始できます。

PXEブートは、サーバに搭載されたNIC(ネットワークカード)自身が持つ機能で**図1**のような流れでインストーラと連携します。サーバを「ネットワークブート」のモードで起動すると、DHCPサーバからIPアドレスを取得すると同時に、「next-server」と「filename」の追加情報を取り出します。DHCPサーバ側では、設定ファイル(Part3の**リスト3**における5行目～12行目のブロック)に**リスト1**の項目を追加しておきます。

そして、「next-server」で指定されたサーバから、「filename」で指定されたプログラム(ブートストラップイメージ)をTFTPプロトコルでダウンロードして起動します。このプログラムは、インストールするOSの選択メニューを表示して、さらに、選択したOSのインストール処理をKickStartで実施するという流れになります。

図2は、Part3のtcpdumpコマンドで、PXEブートを実施した際のDHCPパケットの内容を確認したものです。最後のDHCP-ACKだけを記載していますが、この「next-server」と「filename」で指定した項目が返送されていることがわかります。RHEL6でのKickStart環境の構築方法については、『クラウド時代のサーバー構築・運用の基礎(電子書籍)』[1]が参考になります。

▼図1　PXEブートによるインストーラ起動の流れ



サーバ仮想化環境のDHCP

続いて、サーバ仮想化環境におけるDHCPを説明します。Part3では、実験用にLinux KVMの仮想マシンをDHCPサーバとして構成しましたが、Linux KVMの仮想ネットワーク上には、独自のDHCP機能も提供されています。デフォルトで用意される仮想ネットワーク「default」に接続した仮想マシンは、DHCPでIPアドレスを取得することができます。このDHCPサーバ機能は、ホストLinuxで稼働する「dnsmasq」によって提供されます。

dnsmasqは、簡易的なDHCPサーバ機能を提供するコマンドで、起動時のオプションで各種設定を行います。Part3で構築したホストLinuxで次のコマンドを実行すると、実際の起動オプションが確認できます。

```
# ps -ef | grep dnsmasq
```

出力結果を整形して見やすくしたものを**図3**に示します。「--listen-address」のIPアドレスを持つインターフェースで受信したリクエストに対して、「--dhcp-range」で指定した範囲のIPアドレスを割り当てます。

仮想ネットワーク「default」の実体は、**図4**で示すように、ホストLinux上の仮想ブリッジ「virbr0」で、「--listen-address」に指定している「192.168.122.1」というIPアドレスは、「virbr0」に割り当てられたIPアドレスです。これにより、dnsmasqは、仮想マシン上のDHCPクライアントからのリクエストを仮想ブリッジ経由で受け取る形になります。仮想マシンマネージャーの仮想ネットワーク管理画面(**図5**)でも、同様の情報が確認できます。

dnsmasqは、またDNSサーバとしても機能します。正確には、外部の

▼リスト1　PXEブート用の追加設定

```
next-server 192.168.1.10;
filename    "pxelinux.0";
```

▼図2　PXEブートにおけるDHCP-ACKの詳細

```
08:25:47.974537 IP (tos 0x10, ttl 128, id 0, offset 0, flags [none],
proto UDP (17), length 328)
    192.168.1.2.bootps > 192.168.1.102.bootpc: BOOTP/DHCP, Reply,
length 300, xid 0xef9c9b, Flags [none]
      Your-IP 192.168.1.102
      Server-IP 192.168.1.10          <- next-serverの情報
      Client-Ethernet-Address 52:54:00:ef:9c:9b
      file "pxelinux.0"               <- filenameの情報
      Vendor-rfc1048 Extensions
        Magic Cookie 0x63825363
        DHCP-Message Option 53, length 1: ACK
        Server-ID Option 54, length 4: 192.168.1.2
        Lease-Time Option 51, length 4: 28800
        Subnet-Mask Option 1, length 4: 255.255.255.0
        Default-Gateway Option 3, length 4: 192.168.1.254
        Domain-Name-Server Option 6, length 4: 192.168.122.1
        Domain-Name Option 15, length 11: "example.com"
```

▼図3　仮想ネットワーク「default」のdnsmasq起動オプション

```
/usr/sbin/dnsmasq --strict-order
  --pid-file=/var/run/libvirt/network/default.pid
  --conf-file=
  --except-interface lo
  --bind-interfaces                   <- リクエストを受けるインターフェースと割り当てるIPアドレスの範囲
  --listen-address 192.168.122.1
  --dhcp-range 192.168.122.2,192.168.122.254
  --dhcp-leasefile=/var/lib/libvirt/dnsmasq/default.leases
  --dhcp-lease-max=253
  --dhcp-no-override
  --dhcp-hostsfile=/var/lib/libvirt/dnsmasq/default.hostsfile
  --addn-hosts=/var/lib/libvirt/dnsmasq/default.addnhosts
```

DNSサーバに対するキャッシュサーバとして機能するもので、クライアントから名前解決のリクエストを受け取ると、外部のDNSサーバにリクエストを転送して、得られた結果をクライアントに返します。このとき、得られた結果をメモリ上に記録しておき、次回から、同じリクエストに対しては、dnsmasq自身が結果を返します。

クラウド基盤のDHCP

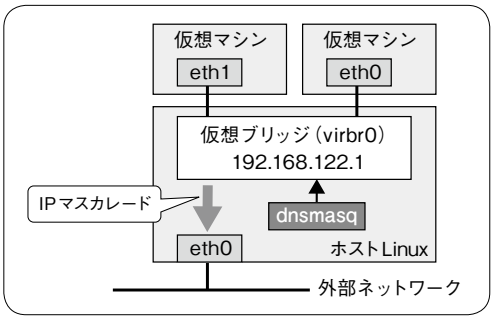
dnsmasqは、クラウド基盤でも利用されています。ここでは、Amazon EC2などと同じ、IaaS(Infrastructure as a Service)タイプのクラウド基盤を構築するソフトウェアの「OpenStack」を例に説明します。

OpenStackでは、典型的には、**図6**のような仮想ネットワークを提供します。OpenStackは、マルチテナントのクラウド環境を用意できて、テナントごとに専用の仮想ルータを割り当てます。各テナントのユーザは、自身の仮想ルータの背後に、自由に仮想スイッチを追加していきます。ちょうど、テナントごとの「家庭内LAN」が用意されており、それぞれがブロードバンドルータを経由してインターネットに接続するような状況です。

それぞれの仮想スイッチは、1つのサブネットに対応しており、クラウド上で仮想マシンを起動して仮想スイッチに接続すると、DHCPによって該当サブネットのIPアドレスが割り当てられます。OpenStackを構成するサーバ群の中で仮想ネットワークを構成する「ネットワークノード」の上では、仮想スイッチごとに、対応するdnsmasqのプロセスが起動して、DHCPサーバとしての役割を果たします。

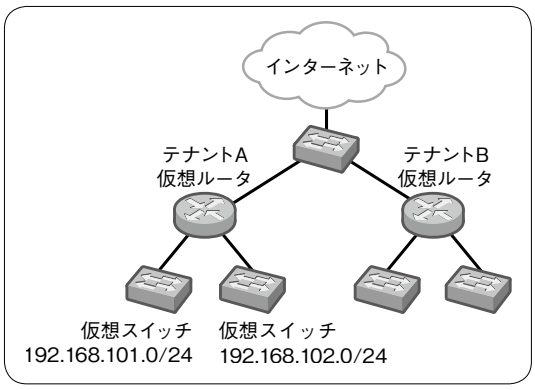
ただし、仮想マシンを再起動してもIPアドレスが変わらないように、MACアドレスと固定IPアドレスを紐づける方法が用いられています。OpenStackの仮想ネットワーク管理機能は、「Neutron」と呼ばれるコンポーネントが提供しており、仮想マシンを起動するタイミングで、Neutronが新しいIPアドレスを用意します。そして、仮想マシンが持つ仮想NICのMACアドレスに対して、このIPアドレスを固定的に割り当てるようにdnsmasqを設定します。MACア

▼図4　仮想ブリッジとdnsmasqの関係



▼図5　仮想マシンマネージャーの仮想ネットワーク管理画面

▼図6　OpenStackが提供する仮想ネットワーク環境



ドレスとIPアドレスの紐づけは、**図3**のオプション「--dhcp-hostsfile」で指定したファイルに記載します。筆者の手元のOpenStackの環境では、**図7**のような内容になっていました。

これにより、仮想マシンのゲストOSでは、一律にDHCPでIPアドレスを設定するようにしておけば、クラウドの管理機能の側で自由にIPアドレスを指定できるようになるわけです。パブリッククラウドサービスであるAmazon EC2では、内部で使用しているソフトウェアの詳細は公開されていませんが、おそらくは同じような方法でIPアドレスの管理を行っているものと想像されます。

ちなみに、この手法は、Part3で紹介した、大量の物理サーバの固定IPアドレスをDHCPで管理する方法と、本質的には同じことです。クラウドの中では、古くからあるさまざまな基礎技術が応用されていることがよくわかります。

スマホのIPアドレスとCGN/LSNの世界

Linux KVMの仮想ネットワーク「default」(**図4**)やOpenStackの仮想ネットワーク(**図6**)では、DHCP(dnsmasq)でIPアドレスを割り当てますが、通常そのIPアドレスは「プライベートIPアドレス」に限定されます。これは、オフィスLANや家庭内LANなど、閉じたネットワーク環境で自由に利用できるIPアドレスの範囲で、「192.168.X.X」などが有名な例です。自宅のLANと隣の家のLANで、偶然に同じIPアドレスを使用していても、プライベートIPアドレスであれば問題ありません[注1]。

一方、インターネットに接続する際は「グローバルIPアドレス」を使用する必要があります。それぞれのグローバルIPアドレスは、インターネットの中でただ一つの機器だけが使用できます。プライベートIPアドレスのネットワークからインターネットに接続する際は、何らかの方法で、グローバルIPアドレスに変換してから接続する必要があります。

図4の場合は、仮想ブリッジから外部ネットワークに出て行く際に、送信元IPアドレスをホストLinuxの物理NICに割り当てられたグローバルIPアドレスに変換します。これは、ホストLinuxのiptablesの機能で行っており、「IPマスカレード」と呼ばれます。

図6のOpenStackの環境でも、IPマスカレードが利用できます。ここでは、各テナントの仮想ルータがIPマスカレードの機能を提供しており、仮想ルータに割り当てられたグローバルIPアドレスを共有して利用する形になります。

このように、1つのグローバルIPアドレスを複数の機器で共有することは、IPアドレスの枯渇対策にもなります。IPアドレス枯渇の問題は、読者の皆さんもご存じだと思います。IPv4の場合、グローバルIPアドレスの総数には限りがありますので、インターネットに接続する機器が爆発的に増えた結果、利用可能なグローバルIPアドレスが不足するという問題です。

とくに最近は、スマートフォンにもIPアドレスが振られるようになったため、IPアドレスの枯渇に拍車がかかったとも言われます。そこで、スマートフォンを提供する通信事業者では、スマートフォンにもプライベートIPアドレスを割り当てる方式を採用するようになりました。スマートフォンからインターネットに接続する際は、通信事業者のネットワーク内でIPマスカレードの処理を行ってから、インターネットに出て行くというしくみ

注1) 「192.168.0.1は私が使っているIPアドレスだから勝手に使わないでください！」という苦情が届いたという有名なネタがあります。

▼図7　dnsmasqでのMACアドレスと固定IPアドレス対応リスト

```
fa:16:3e:6d:35:db,host-192-168-101-2.openstacklocal,192.168.101.2
fa:16:3e:03:da:73,host-192-168-101-1.openstacklocal,192.168.101.1
fa:16:3e:c3:0e:41,host-192-168-101-3.openstacklocal,192.168.101.3
```

です。1つの通信事業者が管理するスマートフォンのIPネットワーク全体が、巨大な「家庭内LAN」になると考えても良いでしょう。

もちろん、膨大な数のスマートフォンからのアクセスをたった1つのグローバルIPアドレスを共有して行うわけではありません。各通信事業者は、複数のグローバルIPアドレスを分配して利用するようなしかけを用意しています。具体的な方法は通信事業者によって異なりますが、たとえば、NTTドコモの「spモード」では、送信元のポート番号に応じて、変換先のグローバルIPアドレスを選択する方法を採用しているとの技術文書が公開されています[2]。

通信事業者やISP(Internet Service Provider)が使用する、このような大規模なIPマスカレードのしくみは、「CGN(Carrier Grade NAT)」や「LSN(Large Scale NAT)」とも呼ばれます。

まとめ

Part4では、サーバ仮想化、クラウド基盤、そして、スマートフォンなどさまざまな環境でのIPアドレス管理を紹介しました。なお、最後のスマートフォンの例では、スマートフォンに対するIPアドレスの割り当ては、DHCPではなく、各通信事業者の独自方式が採用されています。

身の回りのさまざまな機器にIPアドレスが割り当てられて、「モノのインターネット(Internet of Things)」とも呼ばれる現代、IPアドレスの割り当て管理は古くて新しい問題と言えるでしょう。IPアドレス管理の基礎として、この機会にもう一度、DHCPのしくみを根本から理解しておきましょう。SD

●参考文献
[1]『クラウド時代のサーバー構築・運用の基礎(電子書籍)』中井悦司(日経BP)
http://tatsu-zine.com/books/kuraudojidai
[2]「2010年スマートフォン新サービス・機能／スマートフォン向けサービス提供基盤」
https://www.nttdocomo.co.jp/corporate/technology/rd/technical_journal/bn/vol18_3/038.html

Column

IPv6はアドレス自動設定が大前提

IPv4のアドレス枯渇の話題が本文にありますが、IPアドレスの枯渇対策と言えば、IPv6にも触れないわけにはいきません。IPv6は、1つのIPアドレスの長さが非常に長いため、IPv4のように手動でIPアドレスを設定するのは現実的ではありません。何らかのIPアドレス自動設定機能が必須になります。

そのため、IPv6では、DHCPサーバがない環境でもIPアドレスの自動割り当てが行われるようになっています。IPv6対応のOSを搭載したPCでは、NICをネットワークスイッチに接続すると、「リンクローカルアドレス」と呼ばれるアドレスが自動的にセットされます。これは、NICのMACアドレスなどを元に生成することで、必ずユニークなIPアドレスになるようになっています。これで、サブネット内部での相互通信が可能になります。

ただし、ルータを超えてほかのサブネットと通信するには、デフォルトゲートウェイなどの追加情報が必要になります。そこで、PCは「全ルータマルチキャスト」と呼ばれる特別なアドレスに対して、「RS(Router Solicitation)」メッセージを送信します。このメッセージは、同じサブネットのルータが受け取って、ルータからは、「RA(Router Advertisement)」メッセージが送信されます。RAの中には、このサブネットのプレフィックス(IPv4のネットワークアドレスに相当するもの)やデフォルトゲートウェイの情報が含まれており、これを元にして、PCはほかのサブネットと通信可能なアドレスを再設定します。

このようにして、IPv6対応のルータさえあれば、クライアントPCは、アドレスを自動設定してネットワーク接続を開始することが可能になります。このようなIPアドレスの設定を「ステートレス設定」と呼びます。

ただし、ステートレス設定では、MACアドレスに応じた固定IPアドレスの割り当てなどはできません。サーバ環境では、やはり、DHCPサーバが必要となります。IPv6対応のDHCPサーバを使用したIPアドレスの設定は、「ステートフル設定」と呼ばれます。

Practical Column 1

DHCPとDNSの連携

Writer 鶴長 鎮一(つるなが しんいち) shinichi.tsurunaga@gmail.com

はじめに

DHCPは面倒なネットワーク設定が自動化されとても便利な反面、サーバのアクセスログをIPアドレスで追跡しようとしても、どのクライアントにどのIPを割り当てているかを追跡するのが面倒なため、正確にクライアントを特定するのは困難です。そこでDynamic DNSでDHCPサーバとDNSサーバを連携させ、クライアントに割り振ったIPアドレスとホスト名をDNSに登録し、ホスト名でアクセスログを記録できるようにします。

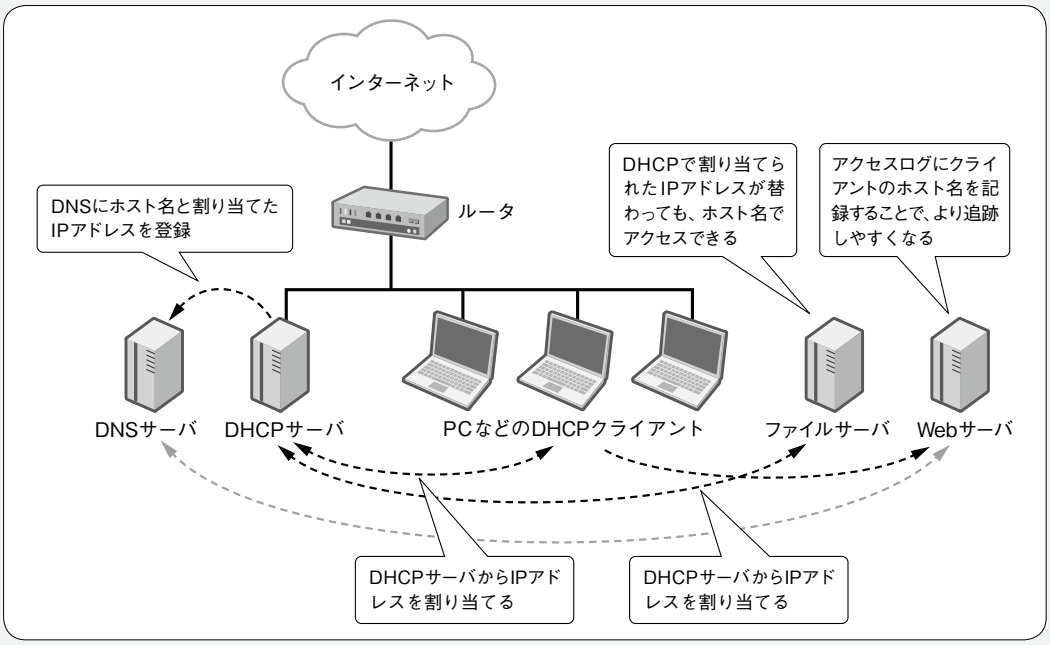
また最近はクラウド環境の普及で、サーバの作成・削除を頻繁に行います。そのため固定IPアドレスが通例だったサーバでもDHCPで設定することが珍しくありません。DHCPサーバとDNSサーバとの連携で、IPアドレスが替わっても同じホスト名でアクセスし続けることが可能になります(**図1**)。

インストールと設定

インストールの注意

DHCPサーバやDNSサーバにCentOS 6.5を使用して解説します。ほかのLinuxディストリビューションではパッケージのインストール方法

▼図1 Dynamic DNSでDHCPサーバとDNSサーバを連携(ローカルネットワーク)



や使用するコマンドなどが異なりますが、設定方法は共通です。インストールは「端末」や「ターミナル」を起動して行いますが、作業には管理者権限が必要になります。CentOSでは「su -」を実行しrootユーザに切り替えます。また、Linuxのファイアウォールや SELinuxが有効だと、外部からのアクセスに失敗したり、システムファイルへのアクセスが制限され正常に動作しません。次の手順で一時的に無効化します。

```
# service iptables stop
# setenforce 0
```

なお、再起動後は元に戻ります。設定やインストールが完了して長時間運用する際には、用途に合ったセキュリティ設定を施してください。

サーバ／ネットワーク構成

図2のようなサーバ／ネットワーク構成例で解説します。DNSサーバは「example.jp」ドメインの正引きと、「192.168.10.0/24」の逆引きができるようゾーンサーバとして設定します。アクセスはローカルネット内のみとし、外部からのアクセスは受け付けないようにします。また、通常のホスト名解決に使えるようキャッシュサーバとしても機能するようにします。

DNSサーバのインストールと設定

DNSサーバをインストールします。「example.jp」と「192.168.10.0/24」のゾーンサーバとして機能するようにします。次のようにyumコマンドでDNSサーバのパッケージ一式をオンラインインストールします。

```
# yum -y install bind
```

続いて次のファイルを修正、または新規に作成します。

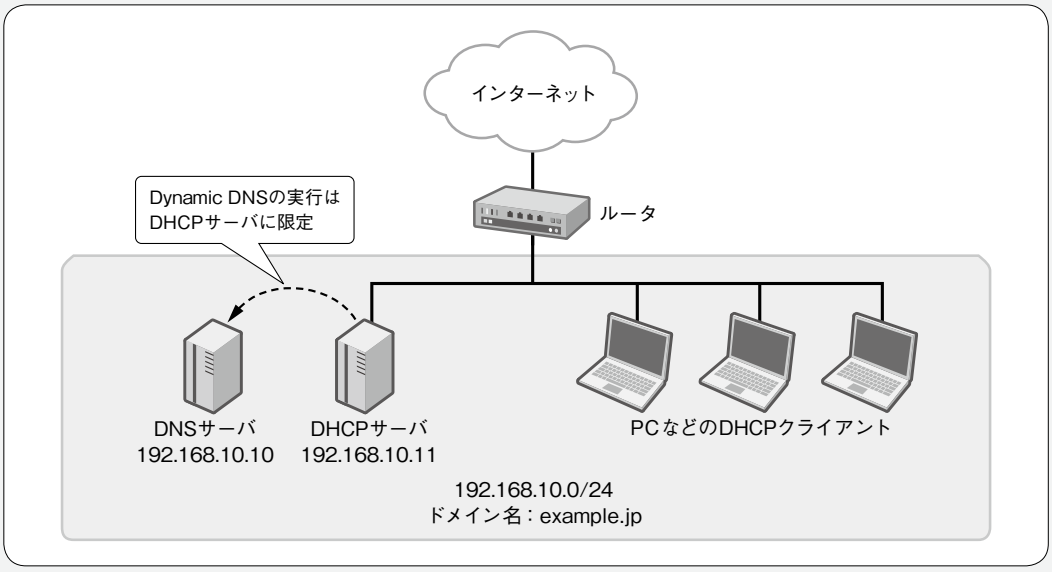
- /etc/named.conf →**リスト1**
- /var/named/example.zone →**リスト2**
- /var/named/example.rev →**リスト3**

インストールと設定は以上です。

図3の手順でDNSサービスを起動します。続いて動作の確認を次のように行います。

1 ゾーンサーバとして動作しているか、DHCPサーバや各クライアントから確認（Linuxの場合digコマンドを使用）

▼図2　サーバ／ネットワーク構成



2 Dynamic DNSの動作を確認(DHCPサーバ上でnsupdateコマンドを使用)

紙幅の都合で申し訳ありませんが、実行結果についてはサポートサイト注1に置きますので、そちらをあわせてご確認ください。

なお、nsupdateは更新対象のドメインを管理しているDNSを見つけるのに、NSレコードではなくSOAレコード中のDNS名が使用されます。リクエストを受信したDNSサーバは即座にメモリにロードされているゾーン情報を更新します。ローカルディスク上のゾーンファイル

注1) http://gihyo.jp/magazine/SD/archive/2014/201407/support

▼図3 DNSサービスの起動／自動起動の設定

```
・DNSサービスの起動
# service named start

・設定ファイルやゾーンファイルを修正した際の再読み込み
# service named reload

・サーバ再起動後もDNSサーバを自動起動するようにする場合
# chkconfig named on
```

▼リスト1 /etc/named.conf

```
acl  "internalnet "{      ←アクセス制御
    192.168.10.0/24;      ←プライベートネットワーク内に限定
    127.0.0.1;            ←localhostからの問い合わせを許可
};

options {
    listen-on port 53 { 127.0.0.1; 192.168.10.10;}; ←DNSサーバ自身のIPアドレスを追加
    directory "/var/named";                         ←ゾーンファイルの保存先
    allow-query { localhost; 192.168.10.11; };      ←DHCPサーバのIPアドレスを追加
    recursion yes;                                  ←DNS再帰問い合わせ(通常のDNS問い合わせ)を有効化
    …省略…
};

zone "example.jp" {           ←「example.jp」ドメインの正引きを設定
    type master;
    file "example.zone";      ←ゾーンファイル名を指定
    allow-update{
        192.168.10.11;        ←DHCPサーバのIPアドレスを追加
    };
};

zone "10.168.192.in-addr.arpa" { ←「192.168.10.0/24」の逆引きを設定
    type master;
    file "example.rev";          ←ゾーンファイル名を指定
    allow-update{
        192.168.10.11;           ←DHCPサーバのIPアドレスを追加
    };
};
```

▼リスト2 /var/named/example.zone

```
$ORIGIN .
$TTL 86400 ; 1 day
example.jp  IN SOA dns.example.jp. root.➡
example.jp. (
    2014051826 ; serial
    3600       ; refresh (1 hour)
    900        ; retry (15 minutes)
    604800     ; expire (1 week)
    86400      ; minimum (1 day)
    )
   NS dns.example.jp.
$ORIGIN example.jp.
dns   A 192.168.10.10
```

▼リスト3 /var/named/example.rev

```
$ORIGIN .
$TTL 86400 ; 1 day
10.168.192.in-addr.arpa IN SOA dns.example.➡
jp. root.example.jp. (
    2014105014 ; serial
    3600       ; refresh (1 hour)
    900        ; retry (15 minutes)
    604800     ; expire (1 week)
    3600       ; minimum (1 hour)
    )
   NS dns.example.jp.
$ORIGIN 10.168.192.in-addr.arpa.
10   PTR dns.example.jp.
```

(example.zone)へは一定間隔で反映され、それまではジャーナルファイル(jnl)に一時保存されます。

DHCPサーバの設定

DHCPサーバの設定ファイル「/etc/dhcp/dhcpd.conf」に**リスト4**の内容を追加し、次のようにDHCPサービスを再起動します。

```
# service dhcpd restart
```

クライアントの設定

ホスト名にはDHCPクライアントに設定されているものが使用されます。Windows、Mac OS X、Linux、スマートフォンなど、OSによらずクライアント側の設定は不要ですが、Linuxでネットワークマネージャを使わずに手動で設定していると、追加設定が必要になる場合があります。CentOSやRed Hatの場合、「/etc/sysconfig/network-scripts/ifcfg-eth0」ファイルに次の1行があるのを確認します。

```
DHCP_HOSTNAME=ホスト名
```

動作確認

設定は以上です。通常の利用でDNSサーバへの登録が開始されます。このあと動作確認として次のことを行います。

▼リスト4 「/etc/dhcp/dhcpd.conf」に追加

```
ddns-update-style interim;  ←「interim」を指定

↓正引きexample.jpのゾーンサーバを指定します
zone example.jp. {
    primary 192.168.10.10;
}

↓逆引き192.168.10.0/24(10.168.192.in-addr.arpa)の
  ゾーンサーバを指定します
zone 10.168.192.in-addr.arpa. {
    primary 192.168.10.10;
}
```

・digコマンドでホスト名の正引きやIPアドレスの逆引きができるかどうか試す
・各サーバの「/var/log/messages」にエラーが出力されていないかを確認

こちらも紙幅の都合で申し訳ありませんが、サポートサイトに実行結果などを置きます。あわせてご確認ください。

ゾーンサーバの指定に誤りがあったり、ゾーンサーバのAレコードが見つからないと「timed out」がログに出力されます。dhcpd.confファイル中の「zone」設定が間違っていると「refused」や「not a zone」が出力されます。

その他の設定

jnlファイルの注意

Dynamic DNSでリクエストを送信すると、更新内容はいったんjnlファイルに蓄えられ、ディスク上のゾーンファイルには即座に反映されません。namedプロセスが障害で停止してもjnlファイルがあれば復旧できます。ただしゾーンファイルを直接編集してしまうとjnlファイルとの整合性が取れなくなり、**リスト5**のようなエラーをログに出力します。復旧するには更新分のjnlファイルを削除し、DNSサービスを再起動します。

セキュリティ対策

Dynamic DNSで更新できるようになると、外部から攻撃される危険性が増加します。攻撃される危険性を減らすには設定を追加します。

リスト1の設定では、Dynamic DNSによる更新をDHCPサーバのみに限定していますが、ほかにも更新できる情報を制限する方法があります。

▼リスト5 jnlファイルとの整合性が取れなくなった場合のエラー

```
May 18 21:24:47 localhost named[9284]:
zone example.jp/IN: journal rollforward
failed: journal out of sync with zone
```

それには「allow-update」の代わりに、「update-policy」を設定します。たとえば、Aレコードだけ更新を許可する場合には、**リスト6**のように「named.conf」を設定します。update-policyには**リスト7**のような指定が可能です。ほかにもDynamic DNSの実行にTSIG認証を導入するなどします[注2]。

ホスト名

DNSに登録されるホスト名には、クライアントで設定されているものが使用されます。クライアント側で自由に設定できるため、同じホスト名が存在した場合に備える必要があります。たとえば同じホスト名を持つクライアントAとBからリクエストがあった場合、クライアントAのゾーン情報が、クライアントBによって上書きされたり削除されたりする危険性があります。

そのためDHCPサーバは登録するレコードに対してハッシュ値を生成し、ゾーン情報にTXTレコードとして埋め込みます。以降の更新にはハッシュ値が必要になるため、ハッシュ値が一致しないリクエストは無視されます。

ホスト名をサーバ側で指定することもできます。クライアントごとにホスト名を指定したり、MACアドレスをホスト名の一部に使用することができます(**リスト8**)。MACアドレスを使用するため、クライアントがホスト名を詐称する危険性がなくなり、アクセスログを追跡する際に役立ちます。

クライアントからホスト名でなくFQDN(フルドメイン付きホスト名)が渡される場合もあります。その場合想定外のドメインが付随している危険性があります。DHCPサーバ側で**リスト9**のような設定を追加することで、FQDNに対処することができます。SD

注2) http://www.atmarkit.co.jp/ait/articles/0307/08/news001_3.htmlを参考。

▼リスト6　「allow-update」の代わりに「update-policy」を使って設定したnamed.confの例

```
key "example.jp" {
    …省略…
};

zone "example.jp" {
    type master;
    file "example.zone";
    update-policy{
        grant example.jp wildcard
        *.example.jp. A;
    };
};
```

▼リスト7　update-policyに設定できる内容

```
・許可を実施する場合
grant keyの名前 nameタイプ name (type)

・不許可を実施する場合
deny keyの名前 nameタイプ name (type)

※nameタイプには次のものを指定
   name      :更新を行うFQDNホスト名がnameと同じである場合
   subdomain :更新を行うFQDNホスト名がnameをドメインの一部に持つとき
   wildcard  :更新を行うFQDNホスト名がnameのワイルドカード指定にマッチするような場合
   self      :更新を行うFQDNホスト名がkeyの値と同じである場合
```

▼リスト8　ホスト名をDHCPサーバで指定する方法(「dhcpd.conf」に追加)

```
・クライアントごとにホスト名を指定する場合(クライアントの指定にはMACアドレスを使用します)
host bbb { hardware ethernet MACアドレス; ddns-hostname "クライアントのホスト名"; }

・クライアントのMACアドレスをもとにホスト名を自動的に割り当てる場合
ddns-hostname = binary-to-ascii (16, 8, "-", substring (hardware, 1, 6));
```

▼リスト9　FQDNへの対応(dhcpd.conf追加)

```
・FQDNからホスト名のみを取り出しドメイン名はDHCPサーバで指定しているものを使用する場合
ignore client-updates;

・FQDNからホスト名とドメイン名を拾い出し、そのドメインに対するゾーンサーバを見つけてリクエストを送信
(クライアントがホスト名しか渡されない場合には正引きのAレコードは登録されず、逆引きのPTRだけ登録される)
allow client-updates;
```

Practical Column 2 Amazon VPCのDHCPオプションで設定を変更するには

Writer 鶴長 鎮一(つるなが しんいち) shinichi.tsurunaga@gmail.com

はじめに

クラウドコンピューティングサービスを使ってサーバ環境を構築することが多くなっています。数あるクラウドサービスの中でも、多くのユーザを有しているのがAmazonの「Amazon Web Services(AWS)」です。AWSの仮想ネットワークは「Amazon VPC(Virtual Private Cloud)」機能によって提供されており、Amazon VPC(以下、VPC)では仮想サーバが使用するIPアドレスをDHCPによって付与します。仮想ネットワークを使いこなすには、VPCのDHCPオプションについて理解しておく必要があります。

Amazon VPCのIPアドレス

すでにAWS上にサーバ環境を構築していたり、何度か検討したりした読者も少なくないと思います。AWSで仮想マシンを起動すると、自動的に仮想ネットワーク環境が与えられます。これは昨年から無料で提供されるようになったAmazon VPC機能によるものです。

VPCを使用すると、仮想マシンなどのリソースが各アカウント専用の仮想ネットワーク内に配置できます。ほかの仮想ネットワークからは切り離されているため、排他的なプライベートネットワーク環境として利用できます。

VPCでは、使用するIPアドレスの範囲、IPアドレスの付与、サブネットの作成、ルーティング、DNS、NAT、インターネットゲートウェイといったものを自由に選択したり導入したりできます。VPCをポリシーごとに複数個作成したり、VPC間で通信できるようにしたりも自在です。IPアドレスにはプライベートアドレスの/16から/28までのCIDR[注1]を利用できるため、スケーラビリティも十分です。

この広大なプライベートIPアドレス空間からIPアドレスを払い出すのにDHCPサーバを使用します。オンプレミスでサーバを構築する場合、一般的に固定IPアドレスを割り当てますが、AWSではDHCPを利用します。AWSではインストールや設定が完了した仮想マシンをAMI(Amazon Machine Image)として保存し、同一環境の仮想マシンを簡単に複製できます。その際固定IPアドレスを割り当てていると、AMIを複製した際にIPアドレスが重複する危険性が発生するためです。DHCPを使えばIPアドレスの重複を避けることができます。またAWSでは、仮想マシンの作成・削除を高頻度で行うことになるため、DHCPのほうが管理面で優れています。

家庭内LANのDHCPでは、PCを起動するたびに付与されるIPアドレスが変更になりますが、VPCでは同じプライベートアドレスが再起動後も付与されます。VPCでは仮想マシンのElastic Network Interfaces (ENI)に対してIPアドレスを払い出すため、ENIが変更にならない限りMACアドレスやIPアドレスが変更になることはありません。

各仮想マシンのENIはAWSマネジメントコンソールの「EC2 Management Console」を開き、左側のナビゲーションペインから「NETWORK

注1) ネットワーク長を1ビット単位で選択できるIPアドレスの割り当て方式。サイダーと発音。

& SECURITY」−「Network Interfaces」と選択することで確認できます(**図1**)。

Amazon VPCのDHCPオプション

実際にVPCのDHCPサーバからクライアントに付与されたIPアドレスの情報を確認しましょう。仮想マシンタイプが「Amazon Linux」の場合、「/var/lib/dhclient/dhclient-eth0.leases」ファイルで確認できます。**リスト1**のように多くの情報が付与されていますが、設定変更できるものは限られます。

VPCのDHCPサーバからクライアントに付与されるDHCPオプションとして、以下のものが設定可能です。

■domain-name-servers

DNSサーバのIPアドレス。デフォルトは「AmazonProvidedDNS」が指定されており、Amazon DNSサーバが使われます。Amazon DNSサーバのIPアドレスはVPCのネットワークアドレスに「2」を加えたものになります(ネットワークアドレスが10.0.0.0/16なら、DNSサーバは10.0.0.2)。

■domain-name

ドメイン名を指定。デフォルトはリージョン固有のドメイン名(TOKYOリージョンなら「ap-northeast-1.compute.internal」)が指定されます。

■ntp-servers

NTPサーバ(時刻調整)のIPアドレスを指定。デフォルトは指定なし。

■netbios-name-servers

NetBIOS Name Service(WINS)サーバのIPアドレスを指定。デフォルトは指定なし。

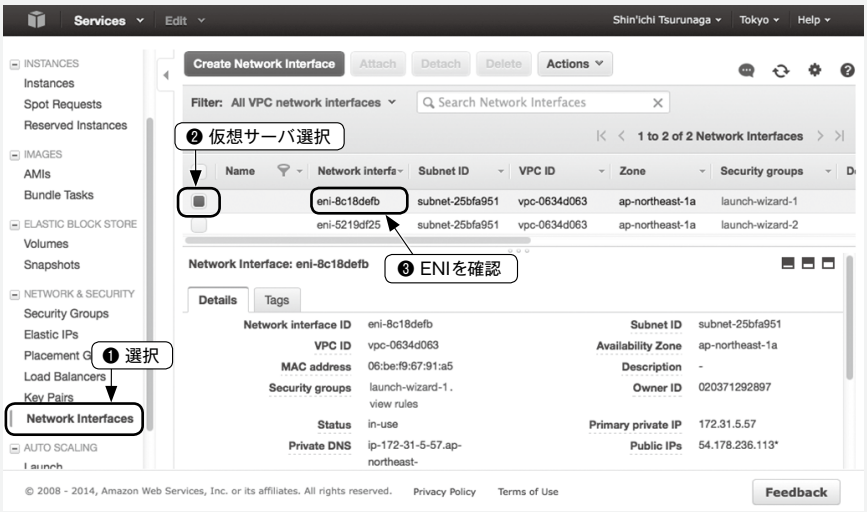
■netbios-node-type

NetBIOSノードタイプを指定。デフォルトは指定なし。

▼リスト1　VPCのDHCPサーバからクライアントに付与されたIPアドレスの情報

```
lease {
  interface "eth0";
  fixed-address 172.31.5.57;
  option subnet-mask 255.255.240.0;
  option routers 172.31.0.1;
  option dhcp-lease-time 3600;
  option dhcp-message-type 5;
  option domain-name-servers 172.31.0.2;
  option dhcp-server-identifier 172.31.0.1;
  option interface-mtu 1500;
  option broadcast-address 172.31.15.255;
  option host-name "ip-172-31-5-57";
  option domain-name "ap-northeast-1.compute.ア
internal";
  renew 1 2014/05/19 20:06:41;
  rebind 1 2014/05/19 20:32:41;
  expire 1 2014/05/19 20:40:11;
}
```

▼図1　仮想マシンのENIを確認



DHCPオプションセットの利用

DHCPオプションセット

VPCでDHCPオプションを指定するには、「DHCPオプションセット」を作成し、各VPCにDHCPオプションセットを関連付ける必要があります。一度オプションセットを作成すると後から変更することはできません。別のオプションセットを新規に作成し、VPCへの関連付けをやりなおします。

現在使用しているVPCに関連付けられているオプションセットを確認するには、AWSマネジメントコンソールの「VPC Management Console」を開き、左側のナビゲーションペインから「DHCP Option Sets」を選択します(**図2**)。

VPCに関連付けることができるDHCPオプションセットは1つだけです。VPCを削除すると、関連付けられているオプションセットも同時に削除されます。

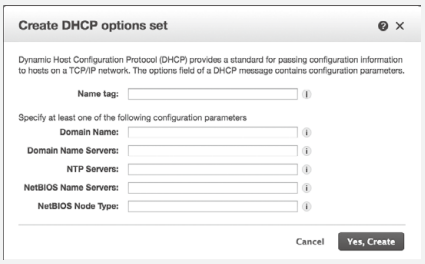
DHCPオプションセットの作成

新たにDHCPオプションセットを作成するには、「VPC Management Console」を開き、左側のナビゲーションペインから「DHCP Options Sets」を選択し、右ペインの「Create DHCP options set」ボタンをクリックします。**図3**のダイアログボックスが表示されたら、使用するオプション値を入力し「Yes, Create」ボタンをクリックします。作成が完了するとオプションセット一覧に作成したものが表示されるため、ID(dopt-○○○○○)を控えます。

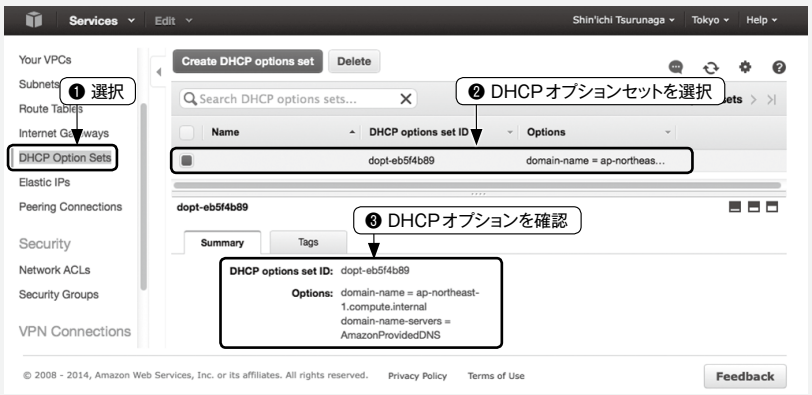
VPCで使用するDHCPオプションセットを変更する

作成したDHCPオプションセットを有効にするには、VPCへの関連付けを行う必要があります。「VPC Management Console」を開き、左側のナビゲーションペインから「Your VPCs」を選択。右ペインでVPCを選択した後、「Edit」ボタンをクリックして「DHCP options set:」項目に先ほど作成したDHCPオプションセットのIDを選択します。「Save」ボタンをクリックすれば変更が反映され、クライアントには次のDHCP更新のタイミングで新しいDHCPオプションが付与されます。即座に反映させるには仮想マシン側でネットワーク情報を再取得するか、再起動します。なおDHCPオプションを使用しないようにすることもできます。SD

▼図3　DHCPオプションセット作成のダイアログボックス



▼図2　VPCに関連付けられているDHCPオプションセットを確認



セキュリティ実践の基本定石【特別編】

ITビジネスの足下を揺るがす大きなバグ

OpenSSLの脆弱性“Heartbleed”の教訓 前編

「Heartbleed」として知られることとなったOpenSSLのHeartbeat Buffer Overreadの脆弱性はたいへん話題になり、あちらこちらで多くの情報が流れました。本稿では前後編の2回に分けて、歴史的背景からソースコードレベルまで、この問題の本質的な原因を深く分析してみます。

すずきひろのぶ
suzuki.hironobu@gmail.com

はじめに

OpenSSLのHeartbeat Buffer Overreadの話題は、単純に技術的に脆弱性があったというだけではなく、それを取り巻く社会状況的にもほかの脆弱性とは違った部分も多いのが特徴です。このテーマを理解するためには、技術的な話題だけではなく、脆弱性情報といった枠組みの理解も必要になってきますので、それも含めて考えていきましょう。前編では、問題の概要と、ここまでの歴史的経緯を説明したいと思います。

Heartbeat Buffer Overreadが騒がれた理由

まずは、Heartbeat Buffer Overreadがこのように注目された理由について整理してみましょう。

詳しくは後ほど説明しますが、今回の脆弱性はOpenSSL 1.0.1から加わったHeartbeatという拡張機能に起因します。これは接続を維持するために一定間隔でリクエストを送る機能です。ここにバグがありました。攻撃側がHeartbeatのリクエストのデータを変更して送信するだけで、サーバ側のOpenSSLのメモリ内のデータが返ってきてしまいます。1回のリクエストにつき、最大64KBを入手することが可能です。

どんな内容が外部に漏れるかは、そのときのメモリの状態に左右されます。つまり、何が漏れるかは誰も事前にはわかりません。しかし、そのメモリ空間にはOpenSSLの秘密鍵の情報や通信中の共通鍵暗号の鍵など極めて重要な、外部に漏れるとサーバにとって致命的とも言える情報が入っている可能性もあります[注1]。またクライアント（サーバを使うユーザ）側からみれば、ログイン認証したあとの（Cookieなどに残されている）セッションIDを取られてしまい、それを用いて詐称したクライアントがサーバに接続し、ログインした状態にするという可能性も否定できません。しかも、このHeartbeatのリクエストが行われたかどうかは、外部ログとして残りません。

そのためインパクトは大きく、セキュリティ界の大御所Bruce Schneier氏は「これは“カタストロフ”という言葉がふさわしい。影響度を1～10までで計るとすると、これは11だ」とまで言っています[注2]。

注1）そのメモリ状態を入手でき、そのデータから暗号解読を試みるには相応の技術力が必要ですが、その件に関してはここでは横において話を進めます。

注2）Bruce Schneierのブログ https://www.schneier.com/blog/archives/2014/04/heartbleed.html

この問題は、Googleのセキュリティグループと、フィンランドのCodenomicon社の両方が同時期に見つけていました。そして何よりも今回の話題をヒートアップさせたのは、Codenomicon社が脆弱性情報公開とともにheartbleed.comというサイトを立ち上げ、「Heartbleed Bug」というキャッチーなキーワードと印象的なロゴを掲げたことでした（**図1**）。

以前よりOpenSSL開発に深くかかわっているGoogleセキュリティチームは、これまでのバグと同様にOpenSSLの開発チームに直接コンタクトを取りました。その一方、Codenomicon社はフィンランドのNational CSIRTを経由して脆弱性流通の枠組みで対処しようとしました。同時期に問題に気づいた両社が別々に脆弱性対応に動くという偶然があったことも、ちょっとした話題になりました[注3]。

Heartbeat Buffer Overreadの第一報

このHeartbeat Buffer Overreadは、どのように世界中に知れ渡ったのでしょう。

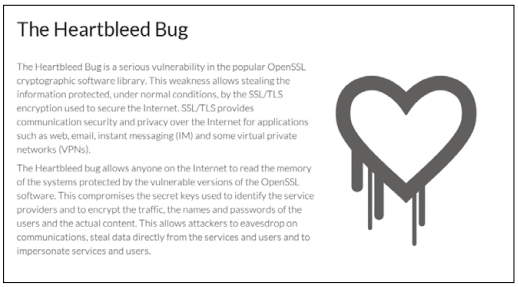
米国日時で2014年4月7日、コンピュータやネットワーク装置などで使われているソフトウェアの脆弱性情報を一元的に採番／管理するしくみである「Common Vulnerabilities and Exposures（CVE）」と、米国国内で脆弱性情報をデータベース化し一般に告知するしくみである「National Vulnerability Database（NVD）」において、CVE-2014-0160という番号が付いた脆弱性情報が公開されました（**図2**）。

これは、「OpenSSLにTLSのHeartbeat拡張を取り扱うコードを実装した際に誤りがあり、OpenSSLで使っているメモリ上にある情報が、ネットワークで接続している側に漏れる。このコードはバージョン1.0.1で入り、1.0.1gまで含まれていた」という内容です。

ただ、同じ4月7日、この発表に先駆けてOpenSSLチームは改修済みのソースコードを公開しアナウンスしています。CVEの発表時には、この脆弱性に対処する方法も明らかにされていたわけです。

また、NVDに連動してCERT/CCが、“Vulnerability Note VU#720951 OpenSSL TLS heartbeat extension read overflow discloses sensitive information”という情報を公開しています[注4]。

▼**図1　Heartbleed Bug**



http://heartbleed.com/

▼**図2　National Vulnerability Databaseでの第一報**

http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2014-0160

一般的な脆弱性流通のしくみ

ここで、一般的な脆弱性情報が公開されるしくみを説明しておきましょう。まず本稿では、脆弱性のことを「第三者がソフトウェアの瑕疵（かし）をセキュリティ侵害の目的で意図的に利用できるもの」と定義します。ポイントを挙げると次のようになります。

①ソフトウェアにバグがある
②そのバグを第三者が意図的に再現できる
③それによってセキュリティ上の問題が発生する

注3）OpenSSLの対応ではGoogleのセキュリティチームのみに謝辞を送っているかたちになっています。 **https://www.openssl.org/news/secadv_20140407.txt**
毎日新聞も興味深い切り口で、この一連の流れを記事にしています。「OpenSSL：事前調整間に合わず 欠陥公表の舞台裏」**http://mainichi.jp/feature/news/20140513mog00m040001000c.html**

注4）**https://www.kb.cert.org/vuls/id/720951**

今回の場合は、「OpenSSLのHeartbeat拡張部分にバグがあり、外部からの接続で細工をしたパケットを送ると、秘密である情報が漏洩する」という問題が発生します。

脆弱性を見つけるフェーズ、ソフトウェア開発者に通知するフェーズ、修正したソフトウェアをユーザに届けるフェーズなどいろいろな場面でステークホルダー（利害関係者）が複雑に絡み合います。「脆弱性はベンダや開発者の責任範囲」と考えている方も多いかもしれませんが、脆弱性情報の取り扱いを考えていくと、最終的には中立的な第三者が統一的に脆弱性情報を一元化して取りまとめ、そしてハンドリングする方法が最も効率的ではないかというところに落ち着きます。

一般に、CVEや同様の脆弱性情報を扱う組織から脆弱性情報が公開されるのは、「その脆弱性の対応が終了し、ユーザに利用してもらえる環境になっているもの」、あるいは最悪、「回避方法を示すことが可能なもの」のみです。脆弱性の対処ができていない状態での公開というものは原則ありません。

今回のHeartbeat拡張の問題の場合、開発元のOpenSSL Projectから、OpenSSL 1.0.1gにアップデートするか、手持ちのコードで**-DOPENSSL_NO_HEARTBEATS**というオプションを付けてリコンパイルし利用すること、というアナウンスが出ました。

CVEとNVD

CVEとはMITRE Corporation[注5]が管理している脆弱性の情報のことで、同社がCVE番号を振ったうえで一般に公開しています。このCVE番号が世界共通の脆弱性の統一番号と言えるものです。この情報はWebサイト[注6]で公開されています。

CVEは統一番号を管理するしくみですが、より具体的な情報はUS-CERTとNIST（National Institute of Standards and Technology：米国立標準技術研究所）のNational Vulnerability Database（NVD）において管理されています。この両組織が、米国内に脆弱性情報を告知し、データベース化し、ユーザなど（この場合はエンドユーザというより、技術者／専門家向けです）に必要な情報を公開しています[注7]。

国内での脆弱性流通

日本でCVEにあたるものがJVNです。

> JVNは、“Japan Vulnerability Notes”の略です。日本で使用されているソフトウェアなどの脆弱性関連情報とその対策情報を提供し、情報セキュリティ対策に資することを目的とする脆弱性対策情報ポータルサイトです。
>
> JPCERTコーディネーションセンターと独立行政法人情報処理推進機構（IPA）が共同で運営しています。
>
> 出典：**https://jvn.jp/nav/jvn.html**

また、NVDやCERT/CCのVulnerability Noteにあたるのが、JPCERTコーディネーションセンターから発行される注意喚起などです。

Heartbeat Buffer Overreadに関して、日本国内では2014年4月8日に「JPCERT-AT-2014-0013 OpenSSLの脆弱性に関する注意喚起」として第一報が公開されています。

JVNサイト：

- JVNVU#94401838
- OpenSSLのHeartbeat拡張に情報漏洩の脆弱性
- https://jvn.jp/vu/JVNVU94401838/index.html

JPCERT/CCサイト：

- JPCERT-AT-2014-0013

注5）MITRE Corporationは、米国の非営利団体「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。直接的には、米国国土安全保障省（U.S Department of Homeland Security）のサイバーセキュリティ＆コミュニケーション室（Office of Cybersecurity and Communications）から資金を得ています。もっとあからさまに言えば、脆弱性のハンドリングは、米国の国家安全保障の枠組みの中で運用されています。

注6）**http://cve.mitre.org/**

注7）US-CERTサイト　**http://www.us-cert.gov/**
National Vulnerability Databaseサイト **http://nvd.nist.gov/home.cfm**
脆弱性情報を告知する役目はUS-CERTとNISTが同時に作業をしているので、NVD自体はNISTのサイトになっています。

- OpenSSLの脆弱性に関する注意喚起
- https://www.jpcert.or.jp/at/2014/at140013.html

時差の関係で1日遅れのように見えますが、実際には同じ日です。これは世界レベルで脆弱性情報のコーディネーションがなされているからです。各国のナショナルサートと呼ばれる有力なCERTチームやステークホルダーが一斉に脆弱性報告を出すような国際コーディネーションを行っています。日本ではJPCERTコーディネーションセンターが、国際連携を行っています（**図3**）。

◆ ◆ ◆

今回のCVE-2014-0160はメディアが大きく取り上げていますが、OpenSSL-1.0.1gのソースコードの変更履歴CHANGESを確認すると、これまで63個のCVE番号が振られていました。OpenSSLが脆弱性を修正するというのは特別なものではなく、日常的なルーティンワークとなっています。

Heartbeatとは何か

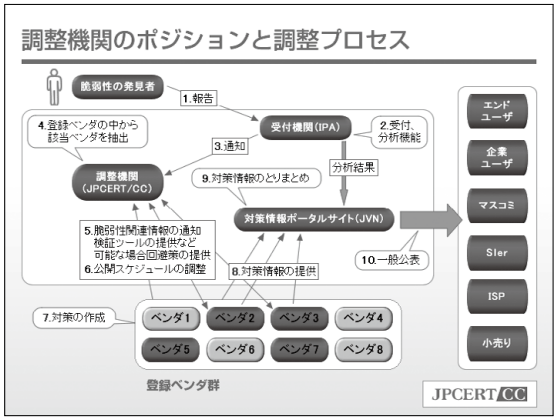
いよいよ、脆弱性の原因について考えていきます。今回のバグの原因となったHeartbeatを理解するために、まずはTLSについて説明します。

TLS/SSLとは

Webサイトの安全性といえば、TLS/SSLというキーワードが思い浮かぶと思います。TLS/SSLはHTTPサーバとWebブラウザの通信を暗号技術によって保護し、安全性を保つために使います。

SSL（Secure Socket Layer）はNetscape Communications社（以下、Netscape社）が作った独自の暗号通信のためのプロトコルで、必ずしもHTTPプロトコル専用に作られたわけではありません。SSLはHTTPプロトコルを変更することなく、TCP/IPの通信に一皮かぶせる汎用のプロトコルとして設計されました。そのためSSLはファイル転送プロトコルのFTPでも、メール転送のSMTPでも、もちろんWWWのプロトコルであるHTTPでも、任意のアプリケーションのプロトコルに対し暗号技術を組み込むことができるようになっています（**図4**）。（各社独自の）VPNソフトウェアのセキュリティレイヤとしてSSLを使う、いわゆるSSL-VPNなどの製品も数多く出回っています。

▼**図3　調整機関のポジションと調整プロセス**

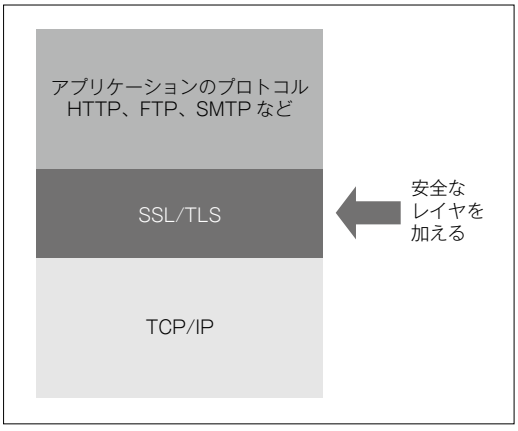


出典：https://www.jpcert.or.jp/vh/index.html

SSL 1.0はNetscape社の内部的な仕様として存在していましたが、一般には公開されていません。1995年にSSL 2.0、そして1996年にSSL 3.0が公開されます。SSLは特定企業の独自仕様にしか過ぎませんでした。その一方で、SSL 3.0はデファクトスタンダードとして多くのWebブラウザに使われ、広まってしまっていました。そのため、SSL 3.0の仕様はRFC6101で「歴史的な記録」として参照のために残されています。

TLS（Transport Layer Security）は、SSLをベースとしてIETFのWGグループで作られた規格で

▼**図4　TLS/SSLのレイヤ関係**



TLS 1.0は1999年にRFC2246で発行されました。改訂されたTLS 1.1はRFC4346（2006）、TLS 1.2はRFC5246（2008）です。

今回の話題であるHeartbeatはTLSの拡張機能ですので、厳密にはSSLは関係しません。文中でTLS/SSLと表現していてもHeartbeatに関してはTLSの範囲だけ、と理解してください。

TLS/SSLはTCP/IPのセキュリティレイヤですが、UDPプロトコルのためのTLSに相当するセキュリティレイヤDTLS（Datagram Transport Layer Security：RFC6347）もあります。そしてDTLSに対応するSSLの規格はありません。そして、Heartbeat拡張はこのDTLSのために提案されました。

Heartbeat拡張

筆者はIETF TLSメーリングリストを購読しているのですが、そのアーカイブをチェックすると、Heartbeat拡張はもともとはDTLSのために2009年7月に提案されていました[注8]。

UDP（Datagram）は、TCPとは違いパケットの到着を保証していません。また、TCPのように通信路が生きているかどうか、一定時間ごとにパケットを送って確認するKeep-Aliveの機能も当然ありません。そこでDTLSのレイヤにこの機能を入れたかったように見えます。

そこで、HeartbeatRequestを送ると、その内容をコピーして、HeartbeatResponceとして送り返すという単純なしくみを組み入れます。2009年8月になるとドラフトのバージョンが00から01に上がり、タイトルが“Transport Layer Security and Datagram Transport Layer Security Heartbeat”という具合に「TLSとDTLSの～」と変化します。2010年2月にバージョン02に改訂されます。

2010年3月に米国カリフォルニア州アナハイムで開催されたIETF 77が開催されます。TLSワーキンググループの3月24日の午後のセッションで提案者からHeartbeatの説明が行われます。それを経て今度は正式にHeartbeatワーキンググループが立ち上がり、本格的にRFC化するための議論が行われるようになります。最初のドラフトであるバージョン00は2010年6月18日付で提案されています。このように議論はオープンに行われており、またIETFサイトでHeartbeatのドラフトの変遷履歴をすべてチェックできます[注9]。

ドラフトバージョン00と01

Heartbeatのドラフトバージョン00と01で**リスト1**のような変更が行われます。

draft 00を読んでいくと具体的なペイロードサイズは何も書いていません。それを具体化して定義したのがdraft 01にあたります。この構造体が今回のOpenSSLのHeartbeat Buffer Overreadの問題を引き起こします。

ドラフトを00から04までアップデートし、長い議論を経て2012年2月8日にRFC6520が発行されました。最初の提案が2009年7月ですから、足掛け4年ほどかかっています。IETFのワーキンググループとしては、とくに長くも短くもなく、まあ、それぐらいの時間はかかるでしょう、といったところです。

IETFの議論は良くも悪くも長くかかります。その代わり衆人監視で行われます。もし問題があり、その問題のアピールに説得力を持たせることができるならば、その意見は反映されるという公平な場であると言えます。

Heartbeatの規格は密室で決められたわけでもなく、拙速に決められたわけでもなく、意見があれば誰でも発言でき、それにあくまでもTLSの拡張としての規格ですから、この機能がなくともTLSは機能するという、ごくごく普通に作り上げられた追加の規格の1つと言えます。

OpenSSLとは何か

1995年にEric A.Young氏とTim Hudson氏という2人のオーストラリア人がNetscape社が作った

注8）Datagram Transport Layer Security Heartbeat Extension
http://tools.ietf.org/html/draft-seggelmann-tls-dtls-heartbeat-00

注9）IETFサイト Heartbeatの変遷履歴 http://tools.ietf.org/wg/tls/draft-ietf-tls-dtls-heartbeat/

▼リスト1　バージョン00と01における変更点

```
 draft 00 の HeartbeatMessage 構造体
struct {
  HeartbeatMessageType type;
  opaque payload<0..2^14-5>;
  opaque padding<0..2^14-5>;
} HeartbeatMessage;

 draft 01 の HeartbeatMessage 構造体
struct {
  HeartbeatMessageType type;
  uint16 payload_length;
  opaque payload[HeartbeatMessage.payload_length];
  opaque padding[padding_length];
} HeartbeatMessage;
```

SSLを実装し、「SSLeay」という名前を付けてインターネットで公開しました。SSLeayの「eay」はEric A.Youngから来ています。ライセンスは今でいうオープンソースにあたるものでした。当時のオーストラリアは米国と比べてはるかに暗号輸出についての規制がゆるい国で、RC4やRSAは米国では特許として保護されていたのですが、オーストラリアでは自由に使えたようです。リリースしたあと、1996年にEric A.Young氏とTim Hudson氏はSSLeayをサポートする会社Cryptsoft社を作ります。

転機は1998年8月に訪れます。2人はRSA Security Australiaに転職します。そしてRSA社の製品と競合するSSLeayを開発し続けることができない立場になります。そこでSSLeayのデベロッパたちが「OpenSSL Project」[注10]を立ち上げ、SSLeayを引き取り、名称をSSLeayから「OpenSSL」へと変更します。そしてライセンスを整備し、Apacheライセンスに近いものになります。

現在のOpenSSL Projectは、コアメンバー3名、ボランティア15名前後という規模で開発しています。そのメンバーのほとんどはヨーロッパ在住です。またOpenSSL Software Foundation,Inc.(OSF)[注11]という組織を持っていて、そこには専任のデベロッパがいます。OSFは複数の企業のサポートサービスを行い、その中には米国連邦政府や米国国防総省も含まれています。

OpenSSLはオープンソースという形で公開されていますが、フルセットをサポートしているバージョンと、OSF社がサポートしFIPS 140-2認証を取得しているバージョンがあります。

前者はTLS 1.2をフルサポートし参照実証としての役割を与えています。後者はFIPS 140-2認証が必要な米国連邦政府で使うシステム向けに使われており、ソースコードは公開されていますが、OSF社のサポートにより特定のプロセッサ向けに最適化するなどされています。

OpenSSL FIPS 140-2認証取得版

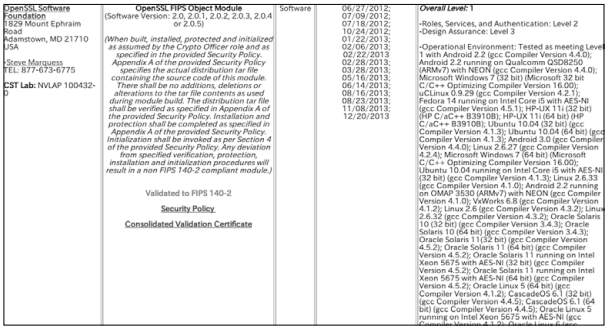
FIPSというのは米国連邦政府の情報処理標準規格です。FIPS 140（1982）は暗号モジュールについての規格になります。まだ1980年代は暗号装置（ハードウェア）のことしか要件定義されていませんでしたが、ソフトウェアの時代に移り、FIPS 140-1（1994）ではソフトウェアが要件定義に入ります。現在の最新版はFIPS 140-2（2001）です。

FIPS 140-2（FIPS 140-1）で定義する暗号モジュールの要件を満たしているかどうかの認証はCRYPTOGRAPHIC MODULE VALIDATION PROGRAM（CMVP）で行われます[注12]。この認証を受けると、その暗号モジュールは米国連邦政府で使える（納品できる）ようになります。日本でも同様の暗号モジュール認証のプログラムJCMVPがあります[注13]。

OpenSSLのサブセットであるOpenSSL FIPS Object Module V2（**図5**）はFIPS 140-2の認証（認証番号#1747）を取得しています。ただし、このFISP 140-2認証を取った形で利用するためには、使う側もいろいろと制約があります。OpenSSLサイトのFIPS関連のドキュメントに詳しく書いてあります[注14]。

注10）https://www.openssl.org
注11）http://opensslfoundation.com/
注12）NISTのWebサイトにあるCRYPTOGRAPHIC MODULE VALIDATION PROGRAM（CMVP）のページ http://csrc.nist.gov/groups/STM/cmvp/index.html
注13）IPAのWebサイトにあるJCMVPの説明 http://www.ipa.go.jp/security/jcmvp/index.html
注14）https://www.openssl.org/docs/fips/

▼図5　NISTサイトにあるOpenSSL FIPS Object Module V2



OpenSSL Software Foundation 1829 Mount Ephraim Road Adamstown, MD 21710 USA Steve Marquess TEL: 877-673-6775 **CST Lab:** NVLAP 100432-0	**OpenSSL FIPS Object Module** (Software Version: 2.0, 2.0.1, 2.0.2, 2.0.3, 2.0.4 or 2.0.5) *(When built, installed, protected and initialized as assumed by the Crypto Officer role and as specified in the provided Security Policy. Appendix A of the provided Security Policy specifies the actual distribution tar file containing the source code of this module. There shall be no additions, deletions or alterations to the tar file contents as used during module build. The distribution tar file shall be verified as specified in Appendix A of the provided Security Policy. Installation and protection shall be completed as specified in Appendix A of the provided Security Policy. Initialization shall be invoked as per Section 4 of the provided Security Policy. Any deviation from specified verification, protection, installation and initialization procedures will result in a non FIPS 140-2 compliant module.)* Validated to FIPS 140-2 Security Policy Consolidated Validation Certificate	Software	06/27/2012; 07/09/2012; 07/18/2012; 10/24/2012; 01/22/2013; 02/06/2013; 02/22/2013 02/28/2013; 03/28/2013; 05/16/2013; 06/14/2013; 08/16/2013; 08/23/2013; 11/08/2013; 12/20/2013	*Overall Level:* 1 -Roles, Services, and Authentication: Level 2 -Design Assurance: Level 3 -Operational Environment: Tested as meeting Level 1 with Android 2.2 (gcc Compiler Version 4.4.0); Android 2.2 running on Qualcomm QSD8250 (ARMv7) with NEON (gcc Compiler Version 4.4.0); Microsoft Windows 7 (32 bit) (Microsoft 32 bit C/C++ Optimizing Compiler Version 16.00); uClinux 0.9.29 (gcc Compiler Version 4.2.1); Fedora 14 running on Intel Core i5 with AES-NI (gcc Compiler Version 4.5.1); HP-UX 11i (32 bit) (HP C/aC++ B3910B); HP-UX 11i (64 bit) (HP C/aC++ B3910B); Ubuntu 10.04 (32 bit) (gcc Compiler Version 4.1.3); Ubuntu 10.04 (64 bit) (gcc Compiler Version 4.1.3); Android 3.0 (gcc Compiler Version 4.4.0); Linux 2.6.27 (gcc Compiler Version 4.2.4); Microsoft Windows 7 (64 bit) (Microsoft C/C++ Optimizing Compiler Version 16.00); Ubuntu 10.04 running on Intel Core i5 with AES-NI (32 bit) (gcc Compiler Version 4.1.3); Linux 2.6.33 (gcc Compiler Version 4.1.0); Android 2.2 running on OMAP 3530 (ARMv7) with NEON (gcc Compiler Version 4.1.0); VxWorks 6.8 (gcc Compiler Version 4.1.2); Linux 2.6 (gcc Compiler Version 4.3.2); Linux 2.6.32 (gcc Compiler Version 4.3.2); Oracle Solaris 10 (32 bit) (gcc Compiler Version 3.4.3); Oracle Solaris 10 (64 bit) (gcc Compiler Version 3.4.3); Oracle Solaris 11 (32 bit) (gcc Compiler Version 4.5.2); Oracle Solaris 11 (64 bit) (gcc Compiler Version 4.5.2); Oracle Solaris 11 running on Intel Xeon 5675 with AES-NI (32 bit) (gcc Compiler Version 4.5.2); Oracle Solaris 11 running on Intel Xeon 5675 with AES-NI (64 bit) (gcc Compiler Version 4.5.2); Oracle Linux 5 (64 bit) (gcc Compiler Version 4.1.2); CascadeOS 6.1 (32 bit) (gcc Compiler Version 4.4.5); CascadeOS 6.1 (64 bit) (gcc Compiler Version 4.4.5); Oracle Linux 5 running on Intel Xeon 5675 with AES-NI (gcc

http://csrc.nist.gov/groups/STM/cmvp/documents/140-1/140val-all.htm

このようにOpenSSLをオープンソースで開発している背景には、サポートする企業もあり、かつ、Googleのセキュリティチームのような厚いバックアップ体制が敷かれているという状況があります。

OpenSSLとHeartbeatのコード

TLS 1.2の参照実装としてのOpenSSLのバージョンは、かなりコードのサイズが大きなものになっています。いろいろな環境で動作できるようにツール群に含んだ形のソースコードツリーになっているためです。参照実装ですので、規格提案をした人たちからのコード、あるいは規格を満たすためにすでにほかで使われていたコード、寄贈されたコードなどもマージされます。たとえば、楕円曲線暗号のライブラリは2005年にSun Microsystems Laboratoriesから寄贈されてOpenSSLの中に入りました。

ソースコードのテストが完全に終了し、品質の高い最適なコードだと思われていても、ソフトウェアにはバグがつきものです。これはオープンなコミュニティで書かれたコードであれ、商用のプロプライエタリな環境で書かれたコードであれ、多かれ少なかれ同じリスクを持っています。とくにソースコードが大きくなればなるほど、機能が複雑になればなるほど、相互的な見通しは悪くなり、バグが入りやすくなる傾向があるのは、どのようなシステムでも同じです。

さて、問題のHeartbeatのコードですが、これは2012年3月14日にリリースされたOpenSSL 1.0.1から加わりました。Heartbeat拡張は、あくまでもTLS/DTLSの拡張ですから、必須の機能ではありません。そして、Heartbeatの機能はTCPであるTLSでは（TCPがHeartbeatの機能を持っているので）必要ありません。

しかしながら、OpenSSLのコンパイルのコンフィギュレーション（設定）は、デフォルトではとにかく入るだけの機能を入れてコンパイルします。つまり拡張で使う予定もないHeartbeat拡張のコードがコンパイルされます。この拡張を入れたくない場合には、明示的に`-DOPENSSL_NO_HEARTBEATS`のオプションを付けてコンパイルする必要があります。ですが、これはTLSやOpenSSLの深い知識がないと難しいと思います。

Heartbeatのバグは、書き込む先のバッファサイズを確認せずに、データの書き込みを行うという極めて単純で、かつ、非常にありふれた間違いをしていました。本質的にはバッファオーバーフローと同じです。しかも、このHeartbeat拡張はDTLSで使うために用意したという、めったに使う機会のない機能です。たとえば、実際に使っていてプログラムがハングアップしたというのなら、問題に気づくと思います。ですが、今回のHeartbeatのBuffer Overreadであれば、（mallocの用意したメモリ空間のような）通常の使用範囲ではハングアップもしません。ですから2年間も気づかれずにいたのも納得です。

見つけてしまえば単純な話なのですが、それが簡単にできないのがソフトウェア開発の難しいところです。とくに現在のOpenSSLのような、コードを付け足し付け足しサイズの大きくなったものはなおさらです。

次回予告

今回は、問題の原因が作りこまれた背景を知るために、まずは歴史や規格策定の流れについてお話しました。次回は、ソースコードレベルでの説明に入りたいと思います。SD

「Web標準技術」で行う Webアプリのパフォーマンス改善

短期集中連載

第3回（最終回）

ブラウザでパフォーマンスを計測する

Writer 川田 寛（かわだ ひろし）
NTTコムウェア㈱ 技術SE部
Web技術者コミュニティ「html5jエンタープライズ部」部長
URL furoshiki.hatenadiary.jp Twitter @kawada_hiroshi

この短期連載もついに最終回です。今回は、Web標準でWebアプリケーション（以下、Webアプリ）のパフォーマンスをモニタリングする手法を紹介します。本稿で取り上げる「Real User Monitoring」は、Webアプリ開発者の協力なしには実現できません。インフラエンジニアだけでなく、アプリ開発エンジニアにも必須の技術です。

Real User Monitoring

Webアプリのパフォーマンスを改善するための機能が、W3Cの正式な仕様として採用される例が増えてきています。その背景には、HTML5の登場でフロントエンドのパフォーマンスが問題視されているという状況があります。Webが表現力を獲得し、jQueryなどのOSSのJavaScriptライブラリが充実した昨今、HTML/CSS/JavaScriptのコンテンツも当たり前になり、Webページを表示する際の処理量も増えています。

ただ、従来のパフォーマンス監視と言えば、NagiosやZabbix、Hinemosなどを活用してネットワーク／サーバ側を監視するものが中心です。これを「Server-side Monitoring」と呼びます。このような方法だけでパフォーマンスを監視していると、「個々のサーバは十分なスループットを確保できているのに、利用者側のブラウザでWebページを表示すると、すごく重い」という状況に陥（おちい）ることがあります。サーバが高速でも、Webページの表示が高速になるという保証はありません。スループットは応答性能ではないのです。利用者側から見たパフォーマンスを可視化するためには、利用者が使っているブラウザからパフォーマンスを計測しなくてはいけません。

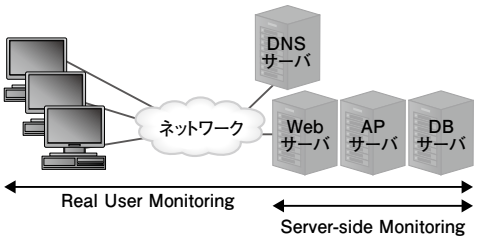
この状況を鑑みて、W3CではWeb Performance Working Groupが発足され、「Real User Monitoring（RUM）」を実現する機能が作られました。RUMとは「利用者が実際に使っているリアルな環境でパフォーマンスを計測しよう」という考え方です（**図1**）。

RUMを実現する新たなWeb標準

最近は、さまざまな企業がRUMを実現するためのサービスを提供しており、利用者環境の視点からパフォーマンス計測を行うことが容易になりました。とくに筆者は、RUMというムーブメントを語るうえで、Yahoo!発のOSS製品「boomerang」[注1]は外すことのできない重要な存

注1）http://yahoo.github.io/boomerang/doc/ja/

▼図1　Real User Monitoringが対象とする監視範囲



在に思えます。多くのサービスは利用者の環境をあくまで擬似的に再現させようという方向性なのに対し、boomerangは、実際に利用者が使っている環境から生のパフォーマンス情報を得ようという方向性です。

boomerangは、JavaScript上からパフォーマンス情報を取得しログサーバへの送信を支援する、小さくて軽量なJavaScriptライブラリです。ただ、boomerangはその機能の一部において、Web標準のAPI「Navigation Timing」をシンプルにラップしているだけですので、少し情報を拾いたいだけなら、そこまで頑張って導入する必要性は感じられなかったりします。

Navigation Timingとは、ブラウザでWebページを表示する際に、ブラウザ内の処理の要所要所で時刻を記録する機能です。今後は、多様なRUMのライブラリが出現することが予想されますが、内部的にはboomerangと同様、このNavigation Timingを利用することになるでしょう。いろんなライブラリを触ってみるのも良いですが、まずはベースとなっているNavigation Timingを押さえておきましょう。

Navigation TimingはHTML5でも比較的早期から議論が開始されたため、Internet Explorer（以下、IE）では9からサポートされ、FirefoxやChromeでも高い相互運用性を持って実装されています。Facebookでも実運用で活用しており、実用性はそれなりに高い技術と言えます。

ただ、利用するにはブラウザの内部的な動作を深く理解することが求められます。本稿でも、まずはブラウザの基本的なしくみから、順を追って説明していきましょう。

「ナビゲーション」とは

図2はブラウザでWebページを表示するときの処理の流れを表した図です。ユーザは、ハイパーリンクやFormのサブミットボタン、「戻る／進む」「更新」ボタンを押下するなどして、Webページの読み込みをブラウザへ依頼します。その後、さまざまなキャッシュ機構の支援を受け、HTMLドキュメントの情報をもとにWebページを表示していきます。この一連の動作を、Web標準では「ナビゲーション」と呼びます。

パフォーマンスには、メモリの使用量やCPUコストなどさまざまな側面がありますが、Navigation Timingはナビゲーションの速度面の評価を行うことを目的としています。

ナビゲーションで注目すべき点は、大きく2つあります。インフラ側の問題が現れやすい「ファイル取得」と、フロントエンド側の問題が現れやすい「レンダリング」です。

ファイル取得は、**図2**の左半分の構成要素にて行われます。ほとんどがキャッシュの機構です。戻る／進むボタンのパフォーマンス改善を行うためのブラウジング履歴用のキャッシュ、RFC2616を用いたキャッシュを実現するHTTP Cache、HTML5から追加されたApp Cacheなどさまざまです。ドメイン名の名前解決を行う際にも、透過的にDNSのキャッシュの恩恵を受けています。ファイル取得のしくみは複雑ですが、レンダリングを行うタイミングでは抽象化され、実装を意識する必要はありません。

レンダリングは、**図2**の右半分の構成要素で行われます。キャッシュか通信か、どちらを使っていても一度はバッファリングが行われ、HTMLドキュメントはバッファリングされている分を逐次パースしていきます。そして、参照されるJavaScriptやCSS、画像ファイルなどの外部リソースを読み込んで、DOMコンテンツというデータベースを作ります。その内容がリアルタイムに、Webページとして画面上に表示されていきます。

Navigation Timingの計測のしくみ

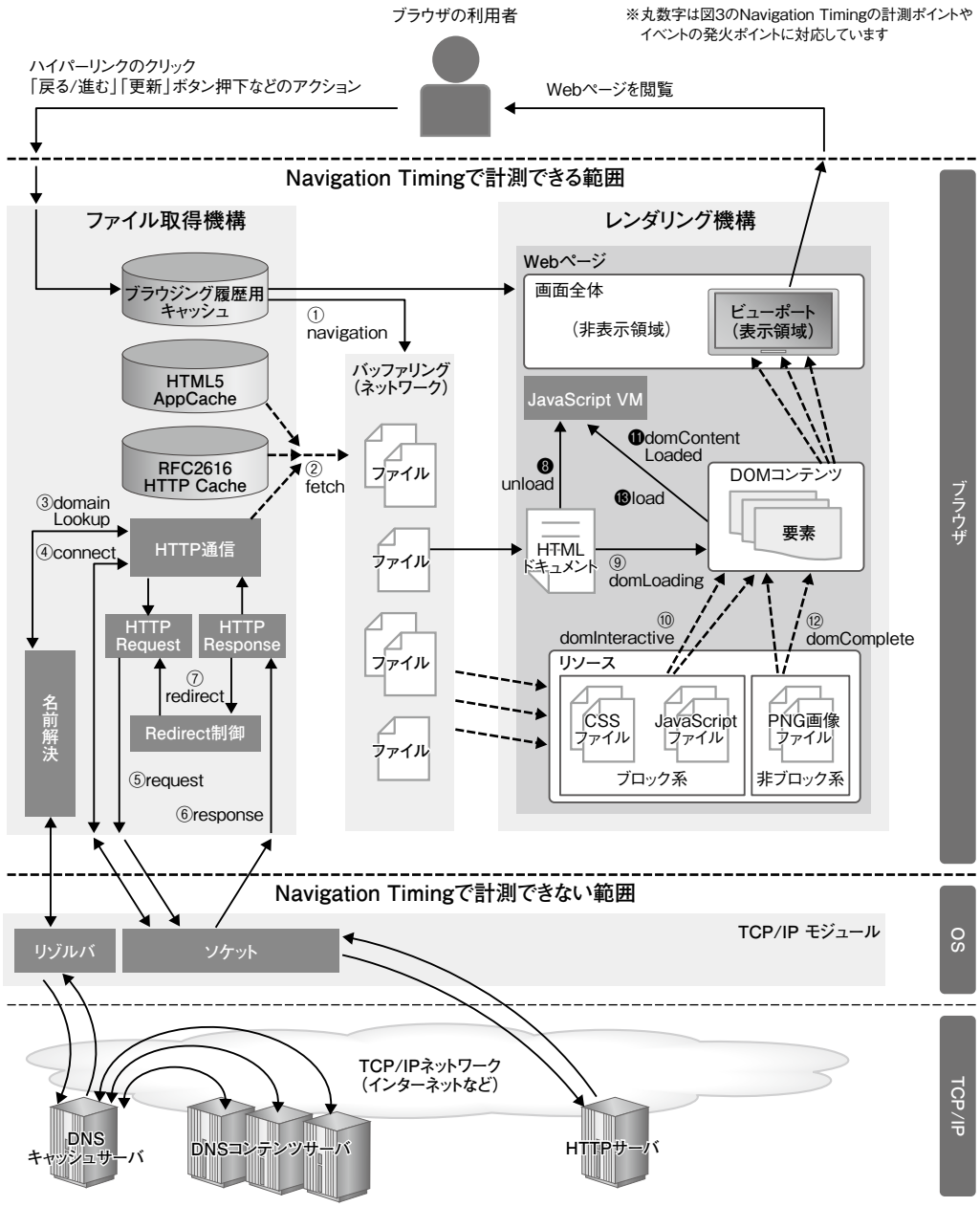
ファイル取得におけるブラウザの動作

Navigation Timingは、ナビゲーションの処

理の中で発生した動作／イベントを、ミリ秒単位の時刻（タイムスタンプ）として記録し、あとからJavaScriptのプロパティとして参照できる機能です。Navigation Timingで記録される時刻の意味を理解するため、ブラウザ内部の処理の流れを把握してみましょう。**図2**と**図3**を見ながら読み進めてみてください。

「戻る／進む」ボタン押下のアクション発生時は、まずブラウジング履歴用のキャッシュが確認されます。利用者が戻るボタンを高速と感じるのは、このキャッシュが再利用されるからです。戻る／進む以外のアクションが発生した場

▼図2　ブラウザ内の論理アーキテクチャ



合、もしくはブラウジング履歴のキャッシュがないと判断された場合、ブラウザはナビゲーションを開始します。その際「navigationStart」(①)の時刻が記録されます。その後、HTMLドキュメントの取得を開始する際「fetchStart」(②)の時刻が記録されます。HTMLドキュメントはすぐにサーバへ取得しにいくのではなく、まずHTML CacheやHTML5 AppCacheといったキャッシュの有無を確認します。

キャッシュが見つからない場合は、サーバから取得しなくてはいけないため、IPアドレスを把握するためドメイン名の名前解決を行います。その際「domainLookupStart」(③-S)と「domainLookupEnd」(③-E)を名前解決処理の前後に記録します。

IPアドレスが取得できると、Webサーバへアクセスするために、3ウェイハンドシェークなどのTCP接続処理を始めます。その前後を「connectStart」(④-S)と「connectEnd」(④-E)として記録します。SSLの場合は、SSLの確立の直前に「secureConnectionStart」(④-A)として値が記録されます。TCP/SSL確立後、HTTP Requestの送信前に「requestStart」(⑤)が記録され、HTTP Responseの受信開始／完了時には、それぞれ「responseStart」(⑥-S)と「responseEnd」(⑥-E)を記録し、サーバとの通信を完了させます。

HTTPヘッダからリダイレクトであることが発覚した場合は、「redirectStart」(⑦-S)を記録し、fetchStart(②)から再開します。リダイレクトが終われば「redirectEnd」(⑦-E)が記録されます。

レンダリングにおけるブラウザの動作

ブラウザはHTMLドキュメントの取得を開始しバッファ内への蓄積を開始すると、それを8KB単位でパースし、DOMコンテンツを作ります。最初の8KBをパースする際、前のWebページに仕込んでいた「unload」(❽)イベントが発火し、DOMコンテンツ内がクリアされます。この時点でブラウザは、前のWebページの表示が消え真っ白な見た目へと変化しています。

HTMLパースが開始される際、直前で「domLoading」(⑨)に時刻が記録されます。その後ブラウザは、HTMLを8KBパース→DOMコンテンツへ登録→HTMLを8KBパース→DOMコンテンツへ登録、という流れをひたすら繰り返します。HTMLの内容がDOMコンテンツへ登録される都度、画面内の表示はアップデートされるため、閲覧者は画面が徐々に読み込まれているという感覚を得ます。

HTMLのパース時には、JavaScriptやCSSといった、即時にコンパイルし実行／反映する必要のあるリソースが見つかることがあります。これらは「ブロック系」と分類され、処理が完了するまでの間は、HTMLのパースを一時的に停止(ブロック)してしまいます。一方で、画像や音声、動画ファイルは「非ブロック系」のリソースに分類されます。ナビゲーション時にWebページ上の複数の画像が、同時に表示されていく様子を見たことがある方もいるでしょう。これは、非ブロック系のリソースが、HTMLパースを継続して行いつつ、リソースが見つかる都度、平行してDOMコンテンツへ反映していくというしくみを持つからです。

HTMLのパースが完了すると、この時点の時刻「domInteractive」(⑩)を記録します。そして「domContentLoaded」(⓫)イベントが発火します。HTMLパースは終わっても、平行して行われていた非ブロック系のリソースは、この時点ではまだ読み込みが完了していないことがあります。これらのリソースが読み込みを終えると、今度は「domComplete」(⑫)の値が更新され、「load」(⓭)イベントが発火します。こうして、ブラウザのナビゲーションは完了します。

JavaScriptで計測値を取得

Navigation Timingで取得できる値の意味が理解できたところで、今度はそれをJavaScript

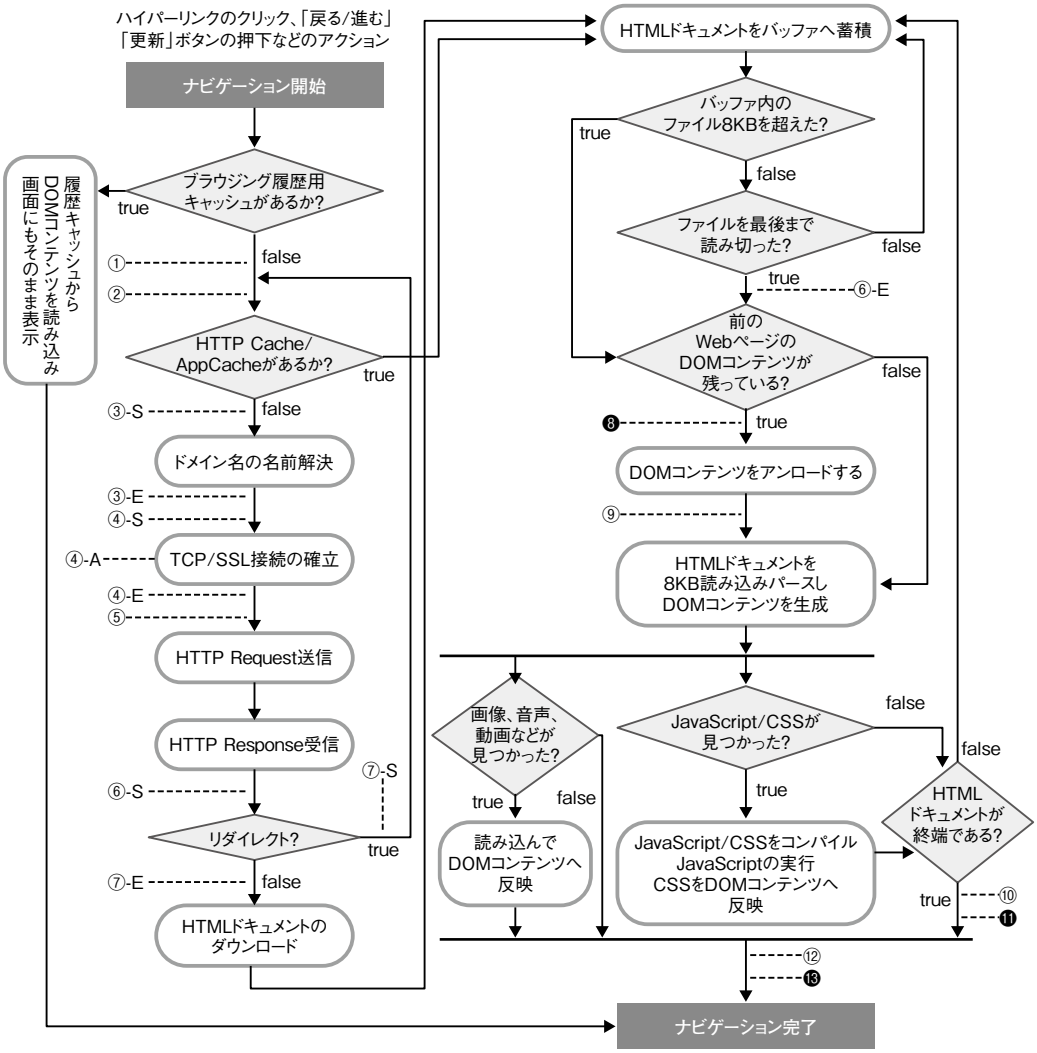
で取得してみましょう。**リスト1**のサンプルコードを確認しながら読み進めてみてください。

windowオブジェクト配下にはperformanceというパフォーマンス関連のAPIを扱うオブジェクトがあり、その配下にtimingというオブジェクトがあります。timingには、パフォーマンスに関連した計測値がread onlyのプロパティとして並んでいます。**リスト1**ではtimingを通じて、ブラウザが蓄積したNavigation Timingの計測値にアクセスし、価値のある情報へと加工しコンソールへ出力しています。

新機能に対応していないブラウザへの対処法

また、**リスト1**では、新しい機能に対応して

▼図3　ナビゲーションの流れと計測ポイント



いないレガシーブラウザ対策に「プログレッシブエンハンスメント」という考え方を適用しています。新しいブラウザには新しい機能を用いた100%の力を発揮させ、古いブラウザでは不足する機能について、実装をあきらめるか、劣化してでもほかの機能で代替させることで対応してしまおうという考え方です。ほかの機能による代替は「フォールバック」と言います。たとえば、jQueryのready()だと、domContentLoadedイベントが実装されていないIE8以下では、loadイベントを使ってフォールバックしています。

フォールバックするにせよ、実装をあきらめるにせよ、ブラウザの機能の対応状況に応じて動作を切り替えなくてはいけないのですが、その際、Web標準では「Feature Detection(機能検出／動作検出)」という手法が推奨されています。ブラウザの種類やバージョンを検出するのではなく、機能の有無によって動作を切り替えることで、未知のブラウザにも備えておこうという考え方です。

プログレッシブエンハンスメントとFeature Detection、この2つを実現しているのが、**リスト1**の最初の3行です。1〜2行目では、ブラウザがNavigation Timingの機能を実装している場合、すなわち、window配下にperformanceオブジェクトが存在する場合はオブジェクトをそのまま返し、そうでない場合はプロパティのない空オブジェクトを返すようにしています。3行目では、performanceオブジェクト内のプロパティ「timing」と「navigation」の有無を確認することで、Navigation Timingの機能を検出し、loadイベントが発火するように作りこんでいます。

Navigation Timingでは、loadイベントの実行開始／終了時間を取得できるのですが、loadイベント自体が終了していないことには、終了時間の時刻が記録されません。**リスト1**では、こうした問題への対策として、setTimeoutを利用して処理の実行タイミングを遅延させています。

Navigation Timingで取得した情報の出力

パフォーマンスに関連する情報は、a〜dに分類し加工して出力しています。Navigation Timingに記録されているのは時刻ですので、ナビゲーション開始後からの相対的な経過時間を取得したい場合は、ナビゲーション開始時間をそれぞれの計測値から減算する必要があります。サンプルコードの「a. 開始からの経過時間」が、その実装サンプルになります。

「b. 処理別の実行時間」では、ナビゲーション内の特定の動作やイベントの実行時間を出力しています。たとえば「loadイベント実行時間」は、loadイベントの終了時刻から、loadイベントの開始時刻を減算することで、実行時間を得ています。

「c. ナビゲーションの種類」は、どのようなアクションによりナビゲーションが発生したのかを出力しています。「TYPE_NAVIGATE」はハイパーリンクやFormのサブミット、ブックマークからのナビゲーションを意味します。対して「TYPE_BACK_FORWARD」は戻る／進むボタンを押下した際にブラウジング履歴用キャッシュにキャッシュがなかった場合のナビゲーション、「TYPE_RELOAD」は更新ボタン押下やスーパーリロードされた際のナビゲーションを意味します。

Resource Timingで取得した情報の出力

「4. リソース別の取得時間」では、「Resource Timing」というWeb標準のAPIを用いて、リソースごとのURIと取得時間を出力しています。ここでいうリソースとは、JavaScriptやCSS、画像ファイルを指します。動画や音声のようなメディア系のファイルは、Web標準ではまだ検討中という状況です。

リソースは、URIをキーとしてDOMコンテンツから参照されます。たとえば、imgタグでPNG画像を参照している場合、同一のURIを指すimgタグは何度使っても、最初に読み込ん

▼リスト1　Navigation Timingの計測値を取得する例

```
var timing = ( window.performance || {} ).timing;
var navigation = ( window.performance || {} ).navigation;
timing && navigation && window.addEventListener( "load", function() { // Navigation Timingの機能検出
  setTimeout( function() {
    console.log( "------------------------------------------------------------" );
    console.log( "ナビゲーショントータル時間:" + (timing.loadEventEnd-timing.navigationStart) + "ms" );
    console.log( "------------------------------------------------------------" );
    console.log( "- a. 開始からの経過時間(Navigation Timing) -" );
    console.log( "ナビゲーション開始:0ms" );
    console.log( "キャッシュ確認開始:" + (timing.fetchStart - timing.navigationStart) + "ms" );
    console.log( "ドメイン名前解決開始:" + (timing.domainLookupStart - timing.navigationStart) + "ms" );
    console.log( "Webサーバ接続開始:" + (timing.connectStart - timing.navigationStart) + "ms" );
    console.log( "HTTP Request送信開始:" + (timing.requestStart - timing.navigationStart) + "ms" );
    console.log( "HTTP Response受信開始:" + (timing.responseStart - timing.navigationStart) + "ms" );
    console.log( "HTMLパース開始:" + (timing.domLoading - timing.navigationStart) + "ms" );
    console.log( "HTTP Response受信終了:" + (timing.responseEnd - timing.navigationStart) + "ms" );
    console.log( "ブロック系リソース読込終了:" +
                 (timing.domInteractive - timing.navigationStart) + "ms" );
    console.log( "非ブロック系リソース読込終了:" + (timing.domComplete - timing.navigationStart) + "ms" );
    console.log( "loadイベントの終了:" + (timing.loadEventEnd - timing.navigationStart) + "ms" );
    console.log( "------------------------------------------------------------" );
    console.log( "- b. 処理別の実行時間(Navigation Timing) -" );
    console.log( "unloadイベント実行時間:" + (timing.unloadEventEnd - timing.unloadEventStart) + "ms" );
    console.log( "domContentLoadedイベント実行時間:" +
                 (timing.domContentLoadedEventEnd - timing.domContentLoadedEventStart) + "ms" );
    console.log( "loadイベント実行時間:" + (timing.loadEventEnd - timing.loadEventStart) + "ms" );
    console.log( "リダイレクト実行時間:" + (timing.redirectEnd - timing.redirectStart) + "ms" );
    console.log( "キャッシュ確認時間:" + (timing.domainLookupStart - timing.fetchStart) + "ms" );
    console.log( "名前解決実行時間:" + (timing.domainLookupEnd - timing.domainLookupStart)  + "ms" );
    console.log( "TCP/SSL接続確立時間:" + (timing.connectEnd - timing.connectStart) + "ms" );
    console.log( "HTTP Request前処理時間:" + (timing.requestStart - timing.connectEnd) + "ms" );
    console.log( "HTTP Response待ち時間:" + (timing.responseStart - timing.requestStart) + "ms" );
    console.log( "HTTP Response応答時間:" + (timing.responseEnd - timing.responseStart) + "ms" );
    console.log( "HTML/JS/CSSパース実行時間:" + (timing.domInteractive - timing.domLoading) + "ms" );
    console.log( "非ブロック系リソース読込時間:" + (timing.domComplete - timing.domInteractive) + "ms" );
    console.log( "------------------------------------------------------------" );
    console.log( "- c. ナビゲーションの種類(Navigation Timing) -" );
    console.log( "ナビゲーション:" + (navigation.type === navigation.TYPE_NAVIGATE ? "TRUE" : "FALSE") );
    console.log( "戻る/進む:" + (navigation.type === navigation.TYPE_BACK_FORWARD ? "TRUE" : "FALSE") );
    console.log( "リロード:" + (navigation.type === navigation.TYPE_RELOAD ? "TRUE" : "FALSE") );
    console.log( "------------------------------------------------------------" );
    console.log( "- d. リソース別の取得時間(Resource Timing) -" );
    if( performance.getEntries ) { // Resource Timingの機能検出
       var entries = performance.getEntries();
       for( var i=0 ; i<entries.length ; i++ ) {
           console.log( entries[i].name + ":" +
                        parseInt(entries[i].responseEnd-entries[i].redirectStart)+"ms" );
       }
    }
  },1);
},false);
```

だPNG画像を流用し続けます。同じURIのファイルを、何度もWebから読み込んだりはしません。このような独特のしくみを持つリソースのパフォーマンスを計測するための機能がResource Timingです。

Navigation Timingは整数のミリ秒単位で時刻が記録されているのに対し、Resource Timingは実数のミリ秒単位(実質的にミリ秒以下まで表現可能)で時刻が記録されています。そのままだと実数型となるため、parseIntで整

数型へ変換してから出力しています。Resource Timingは対応ブラウザも、まだそこまで多くないうえに、Web標準としてのステータスも2014年5月8日現在、「勧告候補」という状況です。普及にはもう少し時間がかかるかもしれません。

Beaconで計測値をサーバに送信

リスト1では、計測結果をそのままコンソールに出力しています。しかし、実際の運用では、計測値を分析して活用することが求められるため、情報をサーバへ送信し蓄積する必要があります。サーバへの送信には、ナビゲーションを伴わずにサーバへアクセスできる「XMLHttpRequest」が有用です。しかし、Webページを表示しているときにこれを行うと、ほかの処理とぶつかる可能性があります。パフォーマンスを改善するために採用した機能が、逆にパフォーマンスを劣化させてしまっては本末転倒です。

こうした問題もあり、情報の送信はWebページのパフォーマンスに最も影響を与えないタイミングである「unload」イベントに仕込むという作法があります。本記事でも一度紹介しましたが、Webページへunloadイベントを仕込んでおくと、そのWebページ上でナビゲーションが発生した際、次のWebページのDOMコンテンツが読み込まれる直前、DOMコンテンツがクリアされるタイミングで発火します。unloadはWebページで捕捉できるあらゆるイベントの中で一番最後に発火するため、ほかの処理を邪魔しません。情報の送信には、最適なタイミングと言えます。Web標準でも、すべてのパフォーマンス計測が終わったタイミングを選ぶ必要があるためunloadイベントが適切である、と説明されています。

Webアプリの本筋とは異なる情報を送信するしくみは、Web標準では「Beacon」として標準化されています。**リスト2**はそのサンプルコードです。かつては「Web Beacon」と呼ばれる、1×1ピクセルの画像を読み込ませてHTTPリクエストヘッダの情報を収集するというアイデアが普及しましたが、Web標準のBeaconはより先進した機能を提供します。Beaconは、JavaScript上から得られる環境に関する情報を収集し、サーバへ送信することを支援します。今回紹介したNavigation Timingの情報を送信するには、最適な機能と言えるでしょう。ただ、Beaconはまだ仕様が固まっておらず、インターフェースも変更が予定されており、**リスト2**のようにXMLHttpRequestでフォールバックすることは必須です。SD

▼リスト2　Beaconの実装例

```
// 情報をあらかじめ詰めておく
var analyticsData = [];
var timing = (window.performance||{}).timing;
if( timing ) {
  analyticsData[“TurnAroundTime”] =
    timing.loadEventEnd - timing.navigationStart;
}
(...略...)
// 情報をWebサーバへ送信する
window.addEventListener('unload', function() {
  var SERV_URL = “/log”;
  if( beacon ) { // beaconが実装されている場合
    beacon("POST", SERV_URL, analyticsData);
  } else { // beaconのフォールバック
    var client = new XMLHttpRequest();
    client.open("POST", SERV_URL, false); //第3引数のfalse→送信が完了するまで制御を返さない
    client.setRequestHeader(
      "Content-Type", "text/plain;charset=UTF-8");
    client.send(analyticsData);
  }
}, false);
```

BOOK no.1 パケットキャプチャ入門 [第3版]
LAN アナライザ Wireshark 活用術

竹下 恵 【著】
B5判、 448ページ ／価格＝2,800円＋税／発行＝リックテレコム
ISBN＝978-4-89797-950-2

本書は、Wiresharkを使ってパケットキャプチャのやり方を学ぶための本だが、それだけではなくネットワークプロトコルの基本を学ぶのに良い教材にもなる。Ethernet Ⅱ /ARP/IP/UDP/DNS/TCP/HTTP/SIPのそれぞれのダンプ解析のやり方を説明しつつ、パケットフォーマットやパケットフローについても解説する。とくにパケットの中身まで示してくれているので、ネットワーク上で流れるデータのイメージがつかめるはずだ。ぜひ、本書に従って実際にパケットキャプチャを試してみてほしい。パケットの中身を見て、その意味がわかるようになればトラブルシューティングもしやすくなるし、具体的な障害内容もつきとめられるようになるだろう。

BOOK no.2 小さな会社のLAN構築・運用ガイド
Windows 8 シリーズ /7/Vista 対応

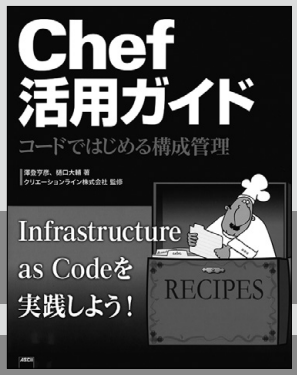
橋本 和則 【著】
B5変形判、 264ページ／価格＝2,400円＋税／発行＝翔泳社
ISBN＝978-4-7981-3228-0

書名どおり、LAN構築時に小さな会社で起きうる課題をやさしく解説した本。パソコンに詳しくない初心者でもWindows 8/7/Vistaなどの混在環境で社内LANを作り上げるためのヒントがまとめられている。

ハブ・ルータ・LANケーブルの商品選択ポイントから、OS設定、無線LANの導入、サーバの構築まで、写真や画面のキャプチャを使って具体的に解説されている。また、TCP/IPのしくみやルータの役割といった基礎の知識も載せられており、初心者にもやさしい。巻頭には、Windowsベースの社内LANを実際に構築した企業の成功事例も掲載されている。

BOOK no.3 Chef活用ガイド

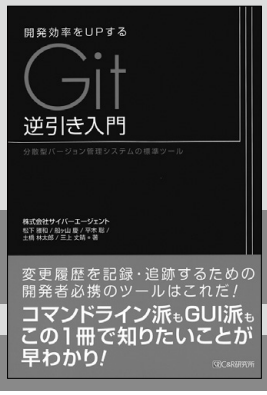
澤登 亨彦、 樋口 大輔 【著】 ／クリエーションライン株式会社 【監修】
B5変形判、 672ページ／価格＝3,900円＋税／発行＝ KADOKAWA
ISBN＝978-4-04-891985-2



Chefの数ある構成要素をきちんと理解し、適切な使い方を実践するのはなかなか難しい。本書はそんな悩みに応えるべく、Chef-Server、Chef-Client、Ohai、Workstationなどの構成要素ごとに、目的、導入方法、使用例などを解説する。後半はRole、Cookbook、Environmentなど個々のリソースや設定ファイルも扱う。Recipeの書き方ならば「何をAttributeにすべきか」「特定プラットフォームごとの設定をするにはどうすればいいのか」といった疑問にも答えてくれる。Recipeの書き方の違いでChefの動作はどう変わるか、そんな観点での解説もある。Chefを導入したなら、もっと効率的に使えないか、本書を片手に確認してみてはどうだろう。

BOOK no.4 開発効率をUPするGit逆引き入門

株式会社サイバーエージェント　松下 雅和、 船ケ山 慶、 平木 聡、 土橋 林太郎、 三上 丈晴 【著】
A5判、 224ページ／価格＝2,200円＋税／発行＝シーアンドアール研究所
ISBN＝978-4-86354-146-7



バージョン管理システム「Git」の入門書。逆引きの形式で書かれ、自分がやりたいことや今困っていることといった疑問から各機能・操作を調べることができ、Git初心者にお勧めだ。また、すべての操作に対して、コマンドライン／GUIでの手順が載せられている。分散型管理という仕様上、集中型のシステムよりも複雑な操作が求められるGitだが、ローカル／リモートリポジトリに関して、ファイルの遷移が簡潔な図でわかりやすく解説されている。

Gitを利用したホスティングサービス「GitHub」（本書では付録ページで紹介）もOSSの開発者を中心に、急激な利用者の伸びを見せている。この機にGitを始めてみてはいかがだろうか。

リアルタイム／分析機能／スケーラブル が武器

複雑化するサーバ環境の監視を変える「OpenTSDB」

後編

サーバ管理者の皆さん、昨今の複雑化・肥大化するサーバ環境の監視業務は、大きな負担になってきてはいないでしょうか。あるいは蓄積されたログを分析していたのでは障害の発見が遅れてしまう事態になっていないでしょうか。本稿で紹介する監視ツール「OpenTSDB」には、もしかしたら欲しかった機能があるかもしれません。一度試してみませんか。

Writer sap

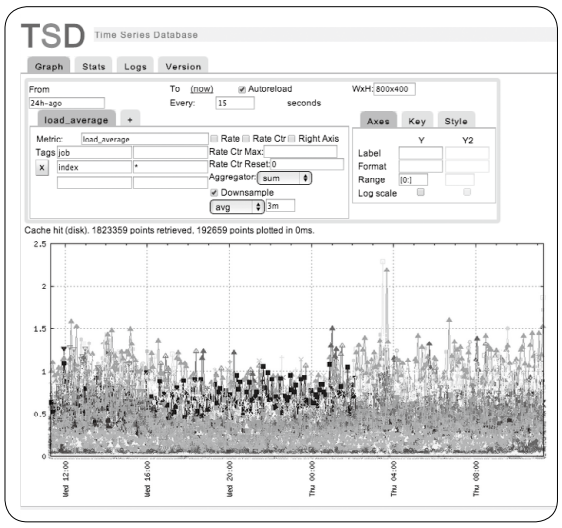
OpenTSDBのコンソールを操作してみよう

先月号に掲載の前編を執筆した新井氏から引き継いで、後編はsapが担当します。前編では導入ということで、OpenTSDBの特徴を簡単に説明し、VagrantとシングルHBaseを使用してスタンドアローンで実際に動作する環境を構築しました。後編はOpenTSDBの実践的な使い方を中心に説明したいと思います。

OpenTSDBを実践的に使用するには、付属のコンソール（Webブラウザで各種設定を行うためのGUI操作画面）を使いこなすことがとても重要です。

図1のように、OpenTSDBに付属のコンソールにはたくさんのボタンやテキストボックスがあり、はじめてOpenTSDBに触れる方にはハードルの高いコンソールだと思います。かなりいろいろな機能が隠れていますので管理者のさまざまなニーズに応えてくれますが、使いこなすには訓練が必要です。まずは本稿でコンソールの基本的な使い方を知り、活用の足がかりにしてください。

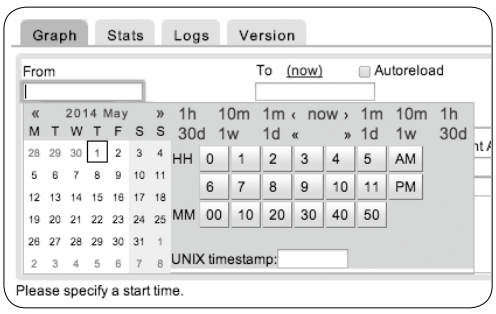
▼図1　OpenTSDBの付属コンソール



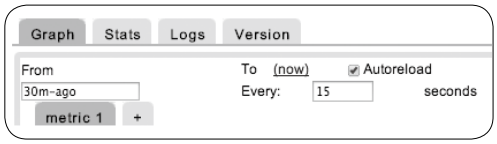
期間の設定

最初に期間の変更の仕方について説明します。**図2**のように、OpenTSDBのコンソールの左上にある「From」と「To」のボックスをクリックすると日付・時間入力用のカレンダーが表示さ

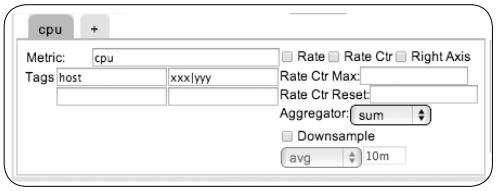
▼図2　コンソールの期間設定



▼図3　期間設定のago機能の例



▼図4　Tags機能



▼図5　ブラウザ上のURLをPNGファイル作成用のURLに変更

```
                q?png&  ←──┐さしかえ
http://localhost:4242/#start=10m-ago&m=sum:rate:tsd.rpc.received&o=&yrange=%5B0:%5D&key=
out%20center%20top%20box&wxh=800x400&autoreload=15
```

れます。日付と時間を設定すると、その期間のモニタリングのデータが表示されます。これだけだと単純な期間設定ですが、**図3**のようにFromのボックスに「30m-ago」と入力すると30分前のデータを、「24h-ago」と入力することで24時間前のデータを表示することもできます。これはかなり便利な機能です。たとえば今から1週間前のデータが見たいというときには、Fromに「1w-ago」と入力してください。「7d-ago」と入力しても同じです。

データ量が多いときや複雑な計算をOpenTSDBにさせているときは、グラフを表示するのに多少時間がかかるかもしれません。しかし、OpenTSDBでは一度見たデータはキャッシュされるので、次に見るときは前回のときより速く表示されます。

Tags機能

図4の「Tags」ボックスは、OpenTSDBの分析機能のうちの1つであるAggregation(集合・集約)を使用するために使います。Tagsの具体的な使用例は後述しますが、たとえば「host」というホストマシンを意味するTagがあるとします。その場合はTagsボックスのキー(左のボックス)に「host」、バリュー(右のボックス)に「*」を入れると、すべてのホストマシンのグラフが一覧表示されます。

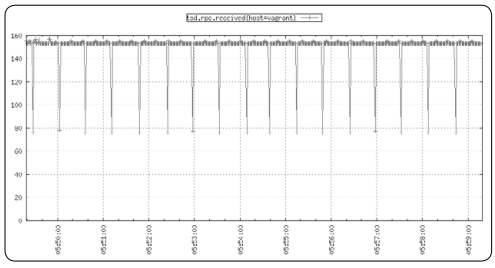
また、xxxとyyyというホストだけを表示させたい場合は**図4**のように「xxx|yyy」と入力すれば、その2つのホストだけのグラフが得られます。hostの値に何も入れず、デフォルトのAggregatorである「sum(合計)」を使用すると、全ホストのMetric(メトリック)合計値のグラフが表示されます。

グラフのPNGファイル作成

OpenTSDBでは、コンソールのURLを修正してリロードするだけでいろいろなことができます。監視用のWebページを作るのに、コンソールのグラフだけほしいといった例を考えましょう。コンソールに表示されているグラフのPNGファイルを作成したい場合は、そのブラウザ上のURL文字列の適切な個所に「png&」をくっつけます。たとえば、**図5**のURLの場合は、「/」の次の「#」を取り除いて「q?png&」とします。あとはブラウザをリロードするだけで、**図6**のようなグラフのPNGファイルが取得できます。

このPNGファイル機能を使用すれば、自前のダッシュボードなどが簡単に作成できます。まずはダッシュボードに載せたいグラフのURLをさきほど紹介したように変更し、PNG

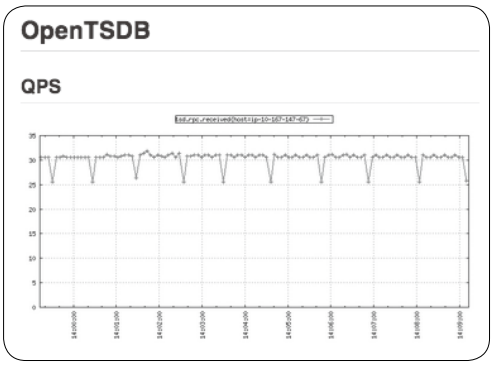
▼図6　作成されたグラフのPNGファイル



▼図7　README.mdの中身

```
OpenTSDB
==============
## QPS
![qps](http://ec2-xx-xx-xx-xx.ap-northeast-1.compute.amazonaws.com:4242/q?png&start=10m-ago&m=sum:rate:tsd.rpc.received&o=&yrange=%5B0:%5D&key=out%20center%20top%20box&wxh=800x400&autoreload=15 "qps")
```

▼図8　GitHub上のREADME.md



ファイルを作成します。あとはダッシュボードのWebページに、作成したPNGファイルのURLを載せるだけです。そのWebページをリロードすれば、いつでも最新のグラフを表示するダッシュボードができあがります。

たとえば、**図7**のようにGitHubのrepositoryのREADMEの中に、作成したPNGファイルのURLを書くと、GitHub上のREADMEは**図8**のように表示されます。ページをリロードすると最新のグラフに更新されるのが確認できます。

asciiモードで生データを取得

さきほどはグラフのPNGファイルを取得しましたが、今度はURLを修正してグラフの生データを取得してみましょう。**図9**のようにURLの「#」を「q?ascii&」に修正してください。修正したら、さきほどと同じようにブラウザをリロードしてください。すると今度はグラフではなく、**図10**のようなグラフの生データがasciiモードで表示されます。このasciiモードのデータをcurlコマンドなどで取得して、awkやsedなどで加工、保存すれば、ExcelやGoogle Docsなどへデータを移して活用するといったこともできます。

No Key表示

Tagでセパレートする値の種類が少ないときは問題ないのですが、Tagの値の種類が多いときにはグラフ表示が見にくくなることがあります。このようなときは、コンソールの「No key」のチェックボックスに印をつけてください（**図11**）。実際、**図1**のグラフはNo keyに印をつけて表示させています。

OpenTSDBにTop 10機能[注1]といったものがあればこのような問題は避けることができるのですが、まだ実装されていません。OpenTSDBのGitHubにこのIssueが上がっているので、将来実装されて、Tagの値の種類が多いときでも見やすいグラフを表示できるようになる日は遠くないと思います。

URLでグラフ共有

PNGファイル作成とasciiモードのところでURLを修正しましたが、OpenTSDBではURLがとても重要な役割を持っています。システムを運用するアドミニストレータ（アドミン）にとって、ほかのアドミンと情報を共有することはとても大切です。調査で使ったグラフのスナップショットをとってメールに添付したり、JIRA[注2]のチケットに貼り付けたりすることは日常茶飯事です。しかし、ほかのアドミンはそのスナップショットの瞬間のグラフは共有できても、そこからでは自分の見たい情報を探

注1）Top 10機能とは、値の大きい順に10個表示させるといった機能のこと。

注2）おもにバグトラッキング、課題管理、プロジェクト管理に用いられているソフトウェア。

▼図9　ブラウザ上のURLをasciiモード用のURLに変更

```
        q?ascii&  ←さしかえ
http://localhost:4242/#start=1h-ago&m=sum:rate:tsd.rpc.received&o=&yrange=%5B0:%5D&key=
out%20center%20top%20box&wxh=800x400&autoreload=15
```

▼図10　asciiモードの出力結果（一部）

```
tsd.rpc.received 1399725064 25.5 host=vagrant
tsd.rpc.received 1399725069 30.599999999999998 host=vagrant
tsd.rpc.received 1399725074 30.599999999999998 host=vagrant
tsd.rpc.received 1399725079 30.599999999999998 host=vagrant
tsd.rpc.received 1399725084 30.599999999999998 host=vagrant
tsd.rpc.received 1399725089 30.599999999999998 host=vagrant
tsd.rpc.received 1399725094 30.599999999999998 host=vagrant
tsd.rpc.received 1399725100 25.5 host=vagrant
tsd.rpc.received 1399725105 30.599999999999998 host=vagrant
tsd.rpc.received 1399725110 30.599999999999998 host=vagrant
tsd.rpc.received 1399725115 30.599999999999998 host=vagrant
tsd.rpc.received 1399725120 30.599999999999998 host=vagrant
tsd.rpc.received 1399725125 30.599999999999998 host=vagrant
tsd.rpc.received 1399725130 30.599999999999998 host=vagrant
tsd.rpc.received 1399725135 30.599999999999998 host=vagrant
tsd.rpc.received 1399725140 30.599999999999998 host=vagrant
tsd.rpc.received 1399725145 30.599999999999998 host=vagrant
tsd.rpc.received 1399725150 30.599999999999998 host=vagrant
tsd.rpc.received 1399725155 30.599999999999998 host=vagrant
```

して調査するのが難しいときがあります。

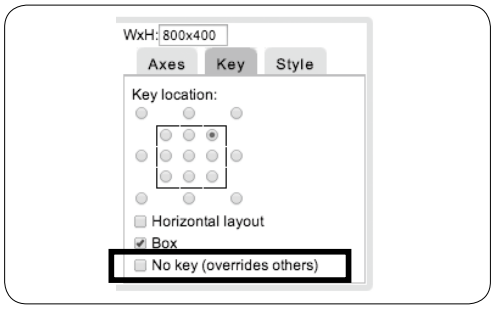
アドミンが実際にアラートを受け取りシステムを調査するとき、SkypeやHipChatといったチャットの中で、ほかのアドミンや調査チケットに更新できるグラフを貼り付けられると、皆がそれぞれに状態を追えるので効率よく調査が行えます。OpenTSDBのURL貼り付けは、それを可能にしてくれます。

TSDBとは?

さて、今までOpenTSDBについて紹介してきましたが、ここであらためて、TSDBについて少し説明したいと思います。

TSDBとは何でしょうか？　TSDBはTime Series DataBaseの略です。日本語にすれば時系列データベースのことで、各時刻ごとの値が集まったデータベースです。**表1**は単純なTSDBの例です。1分おきの時系列「アクセス総数」が4つあります。これは日時2014/05/12 12:34:00にサーバへのアクセス総数が100回、それから1分後の12:35:00にアクセス総数が110回ということを表しています。この表は非

▼図11　Tagの値が多いときはNo keyにチェック



常に単純で、X軸を日時、Y軸をアクセス総数としたグラフとしても表すことができます。

OpenTSDBを使ったTSDBの使い方

OpenTSDBはTSDBを賢く使うことで、ITシステムの見える化が簡単に行えるソフトウェアです。とくにスケールアウト型システムを利用している環境で、サーバ台数が大幅に増加した場合に非常に効率よく可視化できます。その肝は、前述したOpenTSDBのTags機能です。**表2**は、アクセス総数に「index」というTagを付けた例です。

index=server1のアクセス総数が100、110、150、200と増えていき、index=server2のアクセス総数が200、210、220、230ということがわかります。このようにTagをつけることで、同じ内容の値をserverごとに複数保持できます。

スケールアウト型のシステムでは、ユーザからのアクセスはロードバランサなどによって各処理サーバへ分散されます。したがって、1台のサーバに対するアクセス総数を監視していても、システム全体がどのように動作しているのかを判断することは容易ではありません。

▼表1　TSDBの例

時刻	2014/05/12 12:34:00	2014/05/12 12:35:00	2014/05/12 12:36:00	2014/05/12 12:37:00
アクセス総数	100	110	150	200

OpenTSDBは複数のTSDBをTagごとに統計処理できるようになっています。**表2**の例でいえば、ユーザからのアクセスの総数を計算するには、server1とserver2の両方の合計値を計算する必要があります(**表3**)。また、合計のほかにも平均値や最大値などの統計処理が可能です(**表4**)。

カウンタとゲージ

ところで、今までの例で、TSDBの値に「アクセス総数」というものを用いました。アクセス総数というのは積算のアクセス数で、そのサーバが起動してからその時刻までのアクセスの総数です。しかし、サーバへの負荷を知るには、アクセスの総数よりもアクセスが一度にどれぐらい起きているかを測定することが重要です。いわゆるQPS(Queries Per Second)というもので、1秒あたりのクエリ(アクセス)の数です。なぜQPSではなく、アクセス総数を例に使ったのでしょうか？　これには理由があります。

QPSは1秒あたりのアクセス数なので、QPSを保管しようとすると1秒ごとにその値をTSDBに保存しなければなりません。したがって、1日あたり86,400の時系列情報が必要になります。全体のシステムとしてはこれにサーバの数分の行ができますので、データ量が膨大に膨れ上がってしまいます。データ量が膨れ上がるとそのデータを保存するためにシステムの負荷が上がるため、システム本来の仕事に影響が出てきてしまいます。

そのため1秒あたりのシステムの使用状況を知るために、使用状況の値そのものを保存するのではなく、積算されたシステム使用状況をTSDBに保存し、それを数学処理することで1秒あたりのシステム使用状況を計測する方法が効果的です。

表5では、「QPS(合計)」は、「Δアクセス総数(合計)／Δt」で求めています。たとえばQPS(合計)0.33の値は、(320 − 300) ÷ 60で計算しています。すなわち、12:34:00から12:35:00までの1分の間の平均QPSです。実際には、この1分の間にもっと急激なアクセスがあったかもしれませんが、それはこの表からはわかりません。12:34:00から12:35:00まで安定して0.33QPSがあったのかもしれませんし、もしかしたら12:34:59から12:35:00の1秒の間

▼表2　複数サーバのアクセス総数のTSDB

時刻	2014/05/12 12:34:00	2014/05/12 12:35:00	2014/05/12 12:36:00	2014/05/12 12:37:00
アクセス総数(index=server1)	100	110	150	200
アクセス総数(index=server2)	200	210	220	230

▼表3　アクセス総数の合計値を計算

時刻	2014/05/12 12:34:00	2014/05/12 12:35:00	2014/05/12 12:36:00	2014/05/12 12:37:00
アクセス総数(index=server1)	100	110	150	200
アクセス総数(index=server2)	200	210	220	230
アクセス総数(合計)	**300**	**320**	**370**	**430**

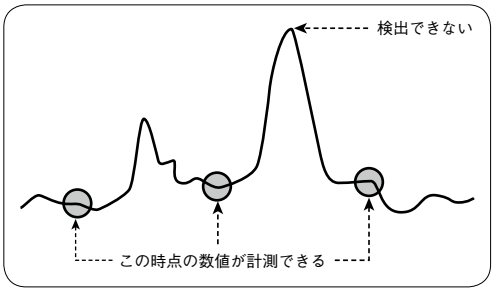
▼表4　アクセス総数の平均値、最大値を計算

時刻	2014/05/12 12:34:00	2014/05/12 12:35:00	2014/05/12 12:36:00	2014/05/12 12:37:00
アクセス総数(index=server1)	100	110	150	200
アクセス総数(index=server2)	200	210	220	230
アクセス総数(合計)	300	320	370	430
アクセス総数(平均)	**150**	**160**	**185**	**215**
アクセス総数(最大)	**200**	**210**	**220**	**230**

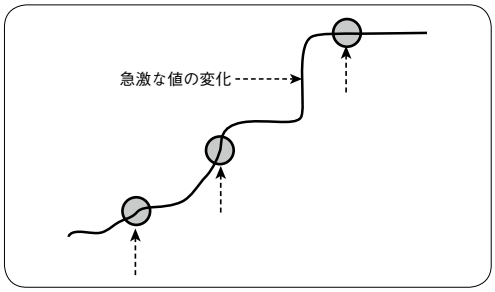
▼表5　QPS（合計）を算出

時刻	2014/05/12 12:34:00	2014/05/12 12:35:00	2014/05/12 12:36:00	2014/05/12 12:37:00
アクセス総数（index=server1）	100	110	150	200
アクセス総数（index=server2）	200	210	220	230
アクセス総数（合計）	300	320	370	430
QPS（合計）	**N/A**	**0.33**	**0.8**	**0.96**

▼図12　ゲージのグラフの例



▼図13　カウンタのグラフの例



に20回のアクセスがあったかもしれません。

もし、各時刻における正確なQPSを求めたい場合は、TSDBのサンプリングレート（現在は1分ごと）を1秒ごとにアップサンプリングする必要があります。ただし、こうすると前述したようにTSDBのデータ量が増えてしまいますので、システム監視の要件に応じて最適なサンプリングレートを設定することになります。

このようにシステム負荷を求めるために記録する積算値のことを「カウンタ」と呼びます。カウンタとはいわゆる数値カウンタのことで、値が単調増加していくタイプの値です。カウンタは、サーバにアクセスがあったときにサーバ自身が保持する1つの変数をインクリメントするだけなので非常に効率が良く、サーバの負荷もほとんどありません。

これに対し、サーバ自身がシステム負荷を計算し、記録する値のことを「ゲージ」と呼びます。ゲージはサーバ自身が値を計算しなければならず、サーバ負荷が上がりやすくなります。

また、ゲージの場合、サンプリングの中間で起きた現象を値として取り出すことができません。たとえば**図12**のようなタイミングで値を抽出してる場合、そのタイミング以外で起きた値の変化を捉えることはできません。

一方、カウンタはサンプリングタイミング以外に起きた現象も捉えることができます。**図13**のようにサンプリングタイミングの外で起きた急激な値の変化がカウンタに値として残ります。これを、前回のサンプリングからの経過時間で除算することで、値の単位時間あたり変異が求まります。形式的には、その値とはグラフの傾きのことです。

OpenTSDBはグラフの傾きを計算することが簡単にできます。**図4**を見てください。ここにあるOpenTSDBコンソールの「Rate」をチェックすると、傾きが計算できます。

OpenTSDBとグラフ

さて、ここまでの説明がOpenTSDBコンソールを使うと実際にどういうふうに見えるかを見ていきましょう。

QPSのグラフ

図14は、あるシステムのQPSのグラフです。このグラフは各サーバが保持するアクセス総数をもとにし、上記で説明したロジックを用いてアクセス総数からQPSを算出しています。

合計QPSを計算すると、**図15**のようなグラ

フになります。各サーバのQPSの合計が確認でき、システム全体で最大6,000QPSくらいあることがわかります。

CPU利用率のグラフ

図16は、あるシステムのCPU利用率のグラフです。このグラフは各サーバの/proc/<pid>/statsにある、積算CPU時間をもとに生成したCPU利用率のグラフです。積算CPU時間を記録して、Rate関数を使うことで1秒あたりのCPU時間を計算し、グラフ化しています。単位は「CPU時間／秒」です。

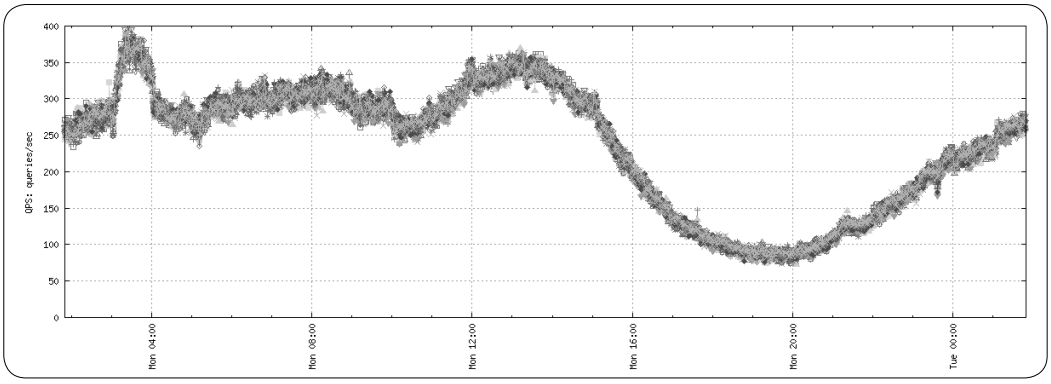
このグラフから各サーバは1秒あたり0.3～0.5のCPU時間を使っていることがわかります。値が1のとき、1つのCPUを100%使っているという意味と同じです。値が2のときは2つのCPUを100%使っていることになります。これらのことから、どのサーバもほぼ同じぐらいのCPU利用率、つまりユーザからの処理をほぼ均等に分散できていることがわかります。仮に分散に偏りがあったとすると、CPU利用率に顕著な差が出てきます。

ほかにも、①異なるCPUを用いてサーバクラスタを構成している場合、②仮想環境などを利用していて特定のホストマシンの負荷だけが高くなった場合にCPU利用率に差が出ます。どちらの場合も、2～複数の束がグラフに現れてきます。

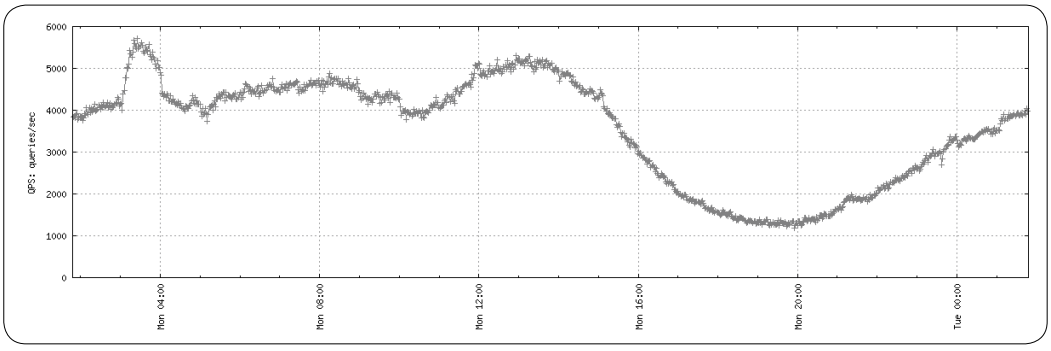
①のグラフでは、すべての物理マシンで同じCPUを使うという運用をするなら、束が1つになることを目視することでマニュアルテストの代用にもなります。②のグラフからは、ホストマシンの負荷を平準化するために、マイグレーションなどのオペレーションが必要であることが読み取れます。

さて、合計値を計算してみましょう。**図17**はCPU利用率の合計を計算させたグラフです。これを見ると、このサーバは全体で10～14

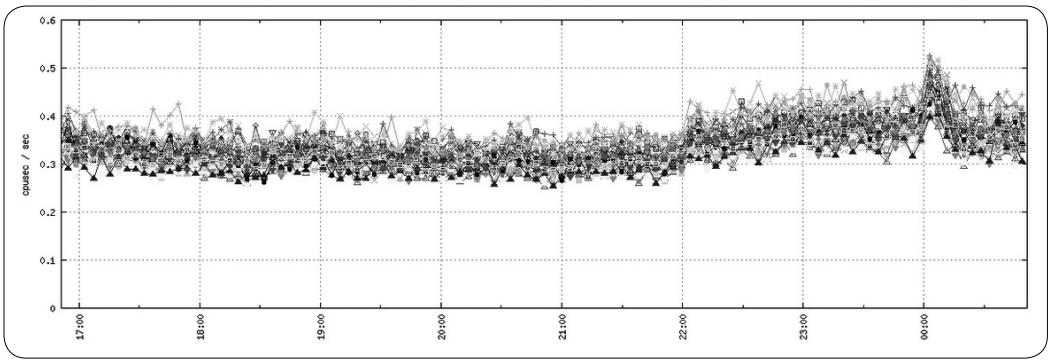
▼図14　QPSのグラフ



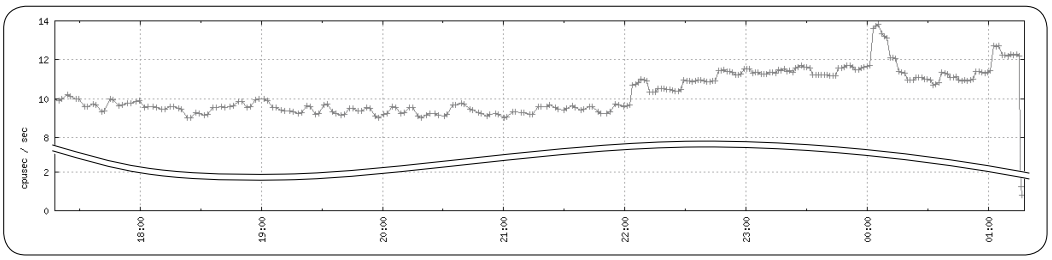
▼図15　合計QPSの例



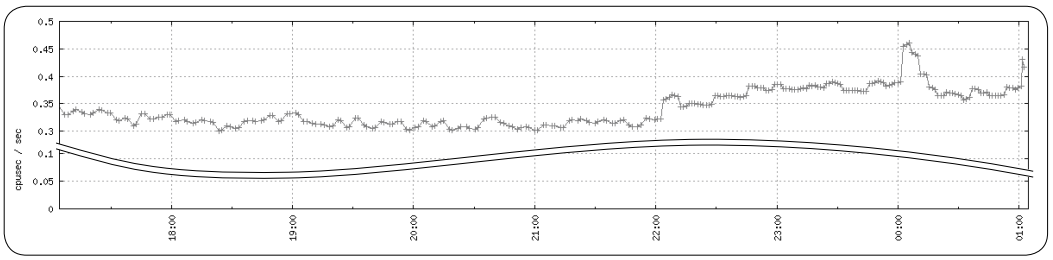
▼図16　CPU使用率のグラフ



▼図17　CPU使用率の合計のグラフ



▼図18　CPU使用率の平均のグラフ



CPUを使っていることがわかります。したがって、16コアのサーバであれば、1台の物理マシンでもユーザからのアクセスが処理可能であることがキャパシティの観点からはわかります。

平均ではどうなるでしょうか。**図18**がCPU使用率の平均のグラフです。0.3～0.45の間に収まっています。この環境は1VMに2CPUを割り当てていますので、まだかなりの余裕があることがわかります。

OpenTSDBの最新情報

2014年5月6日にOpenTSDB 2.0.0が正式にリリースされ、ホームページも新しくなりました。OpenTSDBのGitHub[注3]から、最新の2.0.0のソースコードやrpmとdebパッケージがダウンロードできます。本稿前後編で使用していたOpenTSDBは2.0.0の不安定版なので機能はほとんど同じですが、バグなどがたくさん修正されています。また、現在のOpenTSDBのバックエンドはHBaseだけですが、今後Cassandoraにも対応する可能性があることがドキュメントに書いてあります。今後のOpenTSDBのアップデートにご期待ください。SD

注3）https://github.com/OpenTSDB/opentsdb/releases

短期集中連載

アプリケーション開発者がインフラ担当に!

Rettyのサービス拡大を支えた“たたき上げ”DevOps

第3回(最終回) ❯ やっぱり楽しい! トレンドに乗ったインフラ改善

実名ユーザたちによるお勧めからレストランを探せるグルメ系Webサービス「Retty」。急成長するサービスの裏側では、融通のきかない古いシステムから大規模システムへの移行という難題が立ちはだかっていました。それを乗り越えたのはインフラ経験なしのアプリケーションエンジニア。スマートなだけではすまされない、現場でのInfrastructure as Code実践を紹介してもらいます。

Writer 梅田 昌太(うめだ しょうた) Retty㈱ チャーハン担当 Twitter @ebisusurf

インフラを楽しく整備しましょう

最近AWS Elastic Beanstalk(Amazon Web Servicesのサービス)がDockerをサポートしだして興奮している、Retty チャーハン担当の梅田です。クラウドサービスの進化がすごくてキャッチアップしていくのが大変ですがとても楽しいです。

全3回の本連載もとうとう最終回となりました。今回はラストを飾るべく、モダンな開発環境へのシフトについて書いてみたいと思います。

これまでの記事では割と保守的な感じで、「今あるしくみを動かしつつ、どう改善していくのか?」について書いてきました。ですが、それだけでは開発速度が上がらないし、いつまでもレガシーなプロダクトをメンテナンスし続けるというのはあまりテンションが上がりません。ということで、“少しずつ”でもトレンドな環境に寄せていくためにRettyがどのような活動をしているのか、一部ですが紹介します。

格好良くテンションが上がる言葉「Immutable Infrastructure」

ここ最近Immutable Infrastructure(イミュータブル・インフラストラクチャ)というものが大人気ですね。“もの”というか概念なんだとは思うのですが、個人的にも未来感があってとても好きです。「Immutableとか言われても現状のしくみをいきなり変えることはできない」ということをずっと書いてきたのですが、やはり世の中が「Immutable」とか「Disposable」といった言葉に代表されるような「軽量で使い捨てな環境」に向かっているのは間違いありません。モダンな開発環境へのシフトとして、**Elastic Beanstalkで Immutable Infrastructureを実践!** というテーマでまとめたいと思います[注1]。

Immutable Infrastructureとは? というお話はもっと詳しい方々がたくさんいらっしゃるのでそちらにお任せします[注2]。そして「本来のImmutableな構成」ということを書きだしたらそれだけで話が終わってしまうので、筆者は大まかに「デプロイのフローを新しい考え方でやってみよう」と理解しているということで先に進ませてもらいます。

Rettyにおける Immutableな インフラ構成とは?

おかげさまで最近Rettyはユーザ数300万人を突破しました! そこで現在のRettyのアプリケーションサーバのCPU使用率とユーザ伸び率をざっと見てみたら、今のアプリケーション

注1) 最終回はMapReduceで集計ネタ(AWSだとElastic MapReduce)と迷ったのですが、筆者がとても気に入っているサービス&流行の言葉が使えそうだったのでこちらを選びました。

注2) インターネットで検索すればたくさん出てきます。

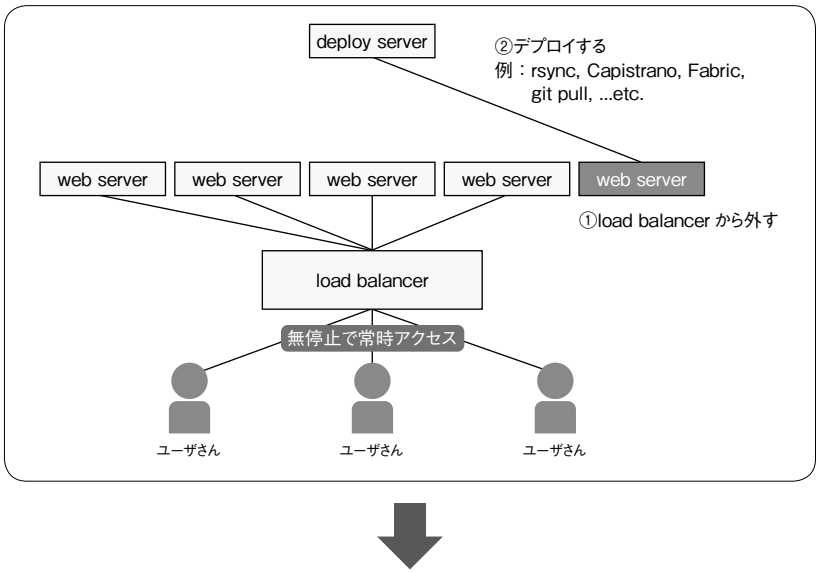
サーバ郡とオペレーションでは500万人くらいで頭打ちが来そうだと予想できました。この将来的な課題への対策にはいろいろなアプローチがあると思いますが、「Immutableな構成」で解決できそうなのは「PHPやRubyアプリのWebサーバへのデプロイ」であろうと狙いを定めました。

Elastic BeanstalkによるImmutable実践の前に、これまで筆者が経験してきたデプロイのフローについて説明しておきましょう。流れとしては、

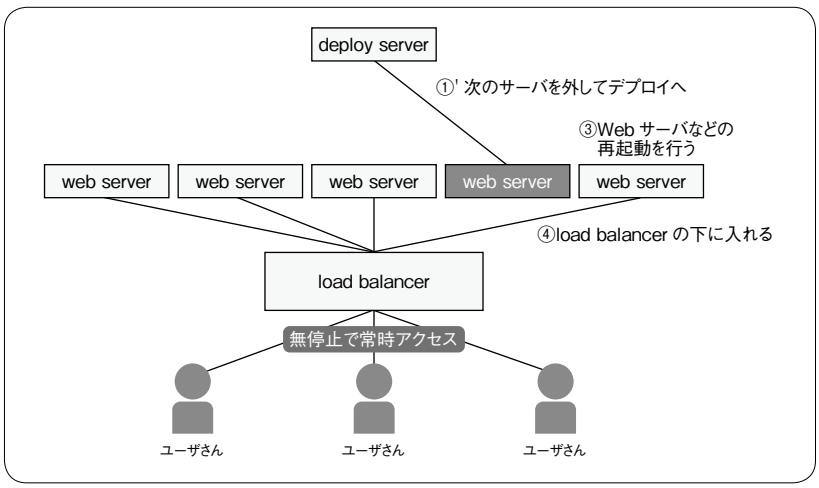
①Webサーバからロードバランサから外す
②rsync、Capistrano、Fabricのようなデプロイツールを使ってファイルを転送
③Webサーバの再起動などを行う
④ロードバランサ配下に入れなおす

これを複数サーバに対して順番にやっていくことになります。図解すると**図1、2**のような感じです。

▼図1　これまでのデプロイのフロー①、②



▼図2　これまでのデプロイのフロー③、④



これまでのデプロイ方法の何が問題なのか?

[1]デプロイ中に環境差異が出る

2、3台なら問題は起こりにくいとは思いますが、100台、1,000台といったインスタンスに対して実行していると転送するだけで30分以上かかったりして、デプロイを行っている時間に環境差異が生まれてしまうことがあります。その間のサービス稼働について考慮しなくてはいけませんでした。

アプリケーションからのデータの読み込み／書き込みによる不整合を起こさないためのロジックを組んだり、バックエンドの先行リリースをするなどしていました。また、staticファイル(cssや画像など)を配信しているサーバの場合は、デプロイ中に表示差異が出てしまいます。

筆者も1,000台以上のインスタンスで稼働しているサービスで開発をしていたときには、上記のようなことを意識してクリティカルな障害とならないように配慮しつつも、デプロイ中に起こりうる差異は諦めていました。

[2]ロールバックが大変

単純にアプリケーションのロールバックをすれば済むのかというと、そうでもありません。[1]の問題と同様にロールバック中に環境差異が出ることで2次障害を生みますし、何より再デプロイとなると単純に時間がかかります。

[3]オペレーションが複雑

ロードバランサから外して、外れたイベントを拾って、適切にファイルを転送して、プロセスを適切に処理して、ロードバランサに戻して、サービスインしたイベントを拾って……。自動化するにしてもタスクが山盛りです。

新しい設計の方針

そこで、Immutable Infrastructureという言葉を持ってきてみます。これは「BlueGreenDeployment」とか「フローティングIPパターン」とか言われていたりもするのですが、インフラデザインとして、

> クラウドなんだから、デプロイはアプリケーションを上書きするんじゃなくて、今動いている構成とまったく同じ構成のものを隣にもう1つ作って、そこでゆっくり落ち着いてデプロイを全部済ませる。その後IPを付け替えるなり(フローティングIPパターン)、ロードバランサの設定を切り替えるなり、URL Swap(Elastic Beanstalkの機能)させるなりしてユーザさんには新しい環境に接続してもらいましょう

という設計です。

問題がなければ古いインスタンスをすべて削除。ロールバックが必要な場合は古い環境に付け替える。本連載第1回目にも書きましたが、こういうインフラデザインは本当にクラウドならではだなーと思います。瞬間的なコスト増はインフラのコストとして積んでおきましょう。

呼び名はいろいろありますが、クラウドのインスタンスは立ち上げたり下げたりをひたすら繰り返すような設計思想にしておくのが今どきの思想なんだと思います。

とはいえ自前でデプロイのshellを書いている時間はありません。**AWSのサービスに乗っかりましょう。**

AWS Elastic Beanstalkとは?

Elastic Beanstalk(以下、EB)はAWSが提供している環境構築のサービスです。筆者はサブドメインサービスのowner.retty.me開発時に使い始めて、とても良かったので、メインサービス(retty.me)にも導入することにしました。ドキュメントを読むとEC2やRDS(Relational Database Service)に比べてとっつきにくかったり、何ができるのかよくわからない印象を受け

ますが、要は「ミニHeroku」「AWS上にパーソナルなPaaSが作れる」という認識でOKだと思います。

AWSの資料にもあるのでここでは省略しますが、類似のサービスにOpsWorksやCloud Formationというサービスもあります(**図3**。筆者はOpsWorksは使ったことがあります。Cloud Formationは未経験です)。

Elastic Beanstalkがやってくれることは?

EBは一般的なWebアプリケーションに必要な設定を組んで、Public DNS名を割り振ってURLでアクセスできるようにしてくれます。AWSマネジメントコンソール、もしくはAWS CLIから、

- 作るアプリはWebアプリか? Workerか?
- RDSは必要か、不要か? Multi-AZにするか?
- ELB(Elastic Load Balancing)を使うか? 1インスタンスで動かすか?

といった選択肢を選んで行くと完成します。

EBが作ってくれるといっても、中身は普通のEC2インスタンスやRDSインスタンスなので、セキュリティグループに関するノウハウやEC2に関するノウハウなどはそのまま使えます(本連載第2回参照)。EBが作ったものは、EC2、ELB、RDSインターフェースのラッパーとして動いてくれるイメージです。

アプリケーションのデプロイをImmutableに

さて、本題に戻りましょう。EBを使えば、Herokuと同じようにgitのリポジトリをそのままデプロイできます。

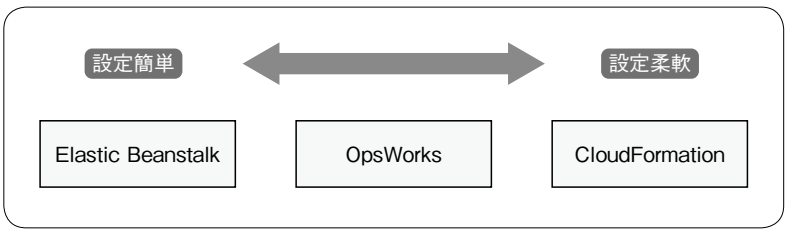
```
git aws.push --environment retty master
```

といった形でenvironmentとgitのブランチを指定します。後はEB側で勝手にデプロイをやってくれます。

詳しいことはAWSの資料にお任せするとして簡単に説明すると、EBにはapplicationとenvironmentという論理構成があります。論理構成といってもただの名前なので気構える必要はありません。ユースケースとして考えると、ドメインレベルでapplicationを用意して、Role単位でenvironmentを用意すると良いと思います(主観です)。そしてenvironment宛にデプロイをします。environmentごとの固有設定は、EBの環境変数に持たせることができます。第2回までにも書いた環境固有にハードコードされた設定は、ここで環境変数に押し込んでおく必要があります。

また、デプロイ時に行うマイグレーションタスクのようなものは、.ebextensionsというディレクトリをgitリポジトリ内のトップディレクトリに作っておいて、yamlでタスクを記述できます。rpmパッケージを取得するようなコマンドもありますし、用途が足りなければ自前のshellを動かすこともできます。.ebextensionsの書き方は、Treasure DataがEBへの

▼図3 Elastic Beanstalk、OpsWorks、CloudFormation



Fluentd（td-agent）のインストール資料[注3]を見るとわりやすいと思います。

筆者も最初はWebサーバのレシピをChefで書いてEC2を管理していたのですが、最近Chefのレシピはproxyサーバやagent向けのサーバだけになってきました。Webサーバの管理はEBに任せています。

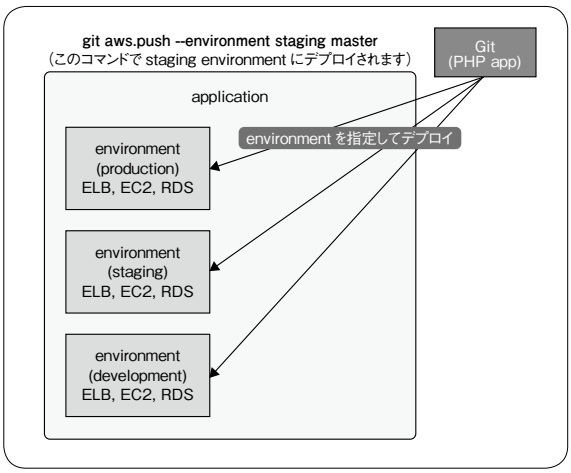
EBをRetty的に組むと、構成は**図4**のようになります。しかし、この状態では**Immutable Infrastructureが実践できているとはいえません**。Immutableに扱うには、インスタンスごと入れ替えられる構造になっている必要があります。そのためには**Elastic BeanstalkのURL Swap機能を使う**と簡単に行えます。

Rettyのリリースフローに合わせると、staging環境でテストするたびにenvironmentを立ち上げて（**図5**）、テストが完了したらstagingのenvironmentをそのままURL Swapでproductionから付け替えます（**図6**）。

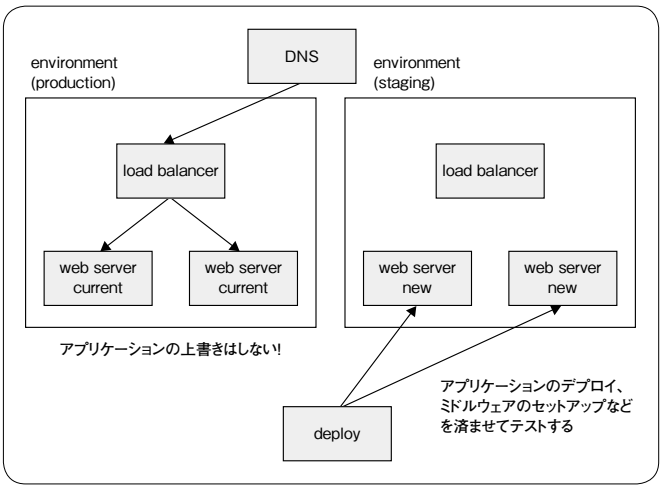
このデプロイフローがImmutable Infrastructureかどうかはさておき、計画していた「インスタンスを使い捨てにするデプロイフロー」になりました。これにより、

- デプロイ中に環境差異が出る問題 → 環境丸ごと切り替わるので起こりません
- ロールバックが大変 → リクエストの向き先を戻すだけなので

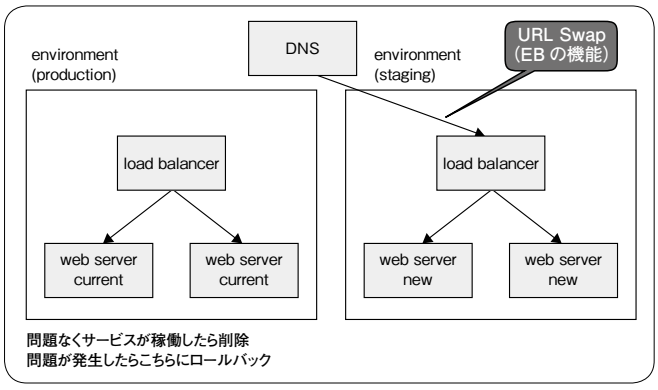
▼図4　EB導入後のWebサーバへのデプロイ



▼図5　stagingで新しいWebサーバインスタンスを構築



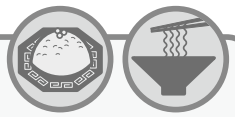
▼図6　productionからstagingに付け替え



注3）https://github.com/treasure-data/elastic-beanstalk-td-agent

Column

とっても便利なCross-Zone Load Balancing



連載第2回(2014年6月号 p.114)で「ELBはZONEごとに均等にリクエストを割り振るので、EC2インスタンスはZONEごとに均等に配置しましょう」と書きました。これはこれで間違っていないのですが、ELBのアップデートがあって、追加の機能としてCross-Zone Load Balancing[注4]が使えるようになりました(結構前ですね……)。筆者も最近まで知らなかったので早速試しました。

この機能により、これまでELBでZONEごとに均等に割り振っていたのを、ZONEをまたいで均等に割り振ってくれるようになります(要するにA-ZONE 2台、C-ZONE 3台の合計5台でも均等に割り振ってくれます)。残念ながらAWSマネジメントコンソールに設定画面がないので、AWS CLIから実行します。

AWS CLIとは

AWS CLI(コマンドラインインターフェイス)のインストールは公式ページ[注5]に書かれているとおり、pipからのインストールが楽です。

```
pip install awscli
```

AWS CLIとは、EC2インスタンスを起動するなどの操作をAWSマネジメントコンソールからではなく、コマンドラインから行えるようにするツールです。コマンドラインということで一見とっつきにくいですが、RDSやELBなどほとんどのサービスに対応しており、一度覚えてしまえば作業が格段に早くなります。APIを叩いてJSONの標準出力を受け取るというのが基本なので、jqコマンドと組み合わせて使うのが定石だと思います。適当にフィルタリングしたら、後はパイプでlessするとかでも良いと思います。

たとえばWebサーバの再起動のように、インスタンスのメンテナンスを行いたいときは、

①awsコマンドでELB配下のEC2のインスタンスを調べる
②該当するインスタンスをawsコマンドでELBから外す
③メンテナンス作業を行う
④awsコマンドでELBの中に入れる

といったことができます。定型作業ならshellスクリプトにしてしまうと良いでしょう。

Cross-Zone Load Balancingの使い方

さて。本題に戻ってCross-Zone Load Balancingを有効にしてみましょう。ロードバランサの名前を適切に変えて次を実行します。

```
aws elb modify-load-balancer-attributes --load-balancer-name youreBalancerName --load-balancer-attributes '{"CrossZoneLoadBalancing": {"Enabled": true}}'
```

これで適用されました。確認してみましょう。

```
aws elb describe-load-balancer-attributes --load-balancer-name youreBalancerName

        "CrossZoneLoadBalancing": {
            "Enabled": true
        },
```

となっていれば適用されています。

ゾーンをまたぐと通信料がかかったり、レイテンシの問題があるので、一概に適用したほうが良いとは言い切れないのですが、可用性の観点から筆者は有効にしてます。

注4) http://aws.amazon.com/jp/about-aws/whats-new/2013/11/06/elastic-load-balancing-adds-cross-zone-load-balancing/
注5) http://aws.amazon.com/jp/cli/

素早く安全
・オペレーションが複雑 → 環境構築だけ自動化してしまえばURL Swapさせるだけ

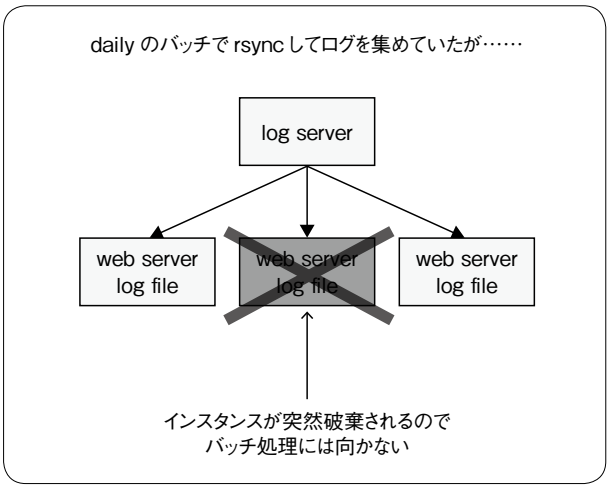
というように、最初にあげていた従来の3つの問題点が一気に解消しました。

EBのオートスケール時に気をつけておきたい点

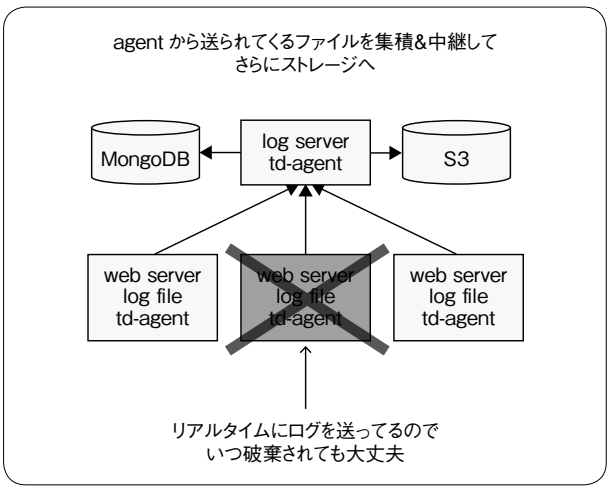
筆者はEBの導入と同時にオートスケールも導入しました。オートスケールとはAWS上のサービスで、細かいことはこれまた公式ドキュメントにお任せするとして、大まかに言うと「一定の時間CPU使用率がNN%超えたら勝手にWebサーバ追加します」という設定をしておけるサービスです。反対にCPU使用率が下がってきたらサーバが削除されます。設定してみるとインスタンスの追加／削除がガンガン通知されてくるので結構面白いです(調べてみたらインスタンスの追加／削除を1日200回以上やってくれてて、とてもエコですね)。

しかし、これには注意すべき点もあります。Webサーバが使い捨てになるので、**Webサーバ上にはデータをstoreしておけない**、ということです。すぐに思いつくのはログファイル系ですね。Webサーバのログだけでなく、独自に出力してる行動ログファイルなどもあるでしょう。たとえば日時でrsyncしてログを集めてくるようなしくみだった場合(Rettyがまさにそうでした)、オートスケールは勝手にインスタンスを破棄してしまうので(保持させる最低時間は設定できますが)非常に相性が悪いです(**図7**)。

▼図7　オートスケール利用時に相性の悪いログ収集処理例



▼図8　オートスケール利用時にはtd-agentがログ収集に便利



そこで使いたいのが、ebextensionsのサンプルでも紹介したtd-agent(Fluentd)です。td-agentはWebサーバ上にagentが常駐して、設定ファイルどおりにファイルの中身を別のサーバに送ってくれます。いろいろな使い方はありますが、単純にテキストファイルを集めてきたいだけ、というようなときにもすごく便利です。S3へのアダプタ、MongoDBへのアダプタもあるのも使いやすい理由です。

Rettyでは**図8**のようにしました。いろいろと凝った使い方もしたくなりますが、「とりあえずそのまま保存しておく」という方針ではじめてみるとよいと思います。ほぼリアルタイ

ムに全サーバのログを見られるので、「awkやgrepで検索する」でも相当な情報を探れます(たとえばデプロイ直後にエラーが起こった、特定のページに大量のアクセスがあった、etc.)。

全3回を通したまとめ

いかがでしたでしょうか。まだまだ書ききれないことがたくさんあるのですが、当初から一貫して伝えたかったことは3つです。

1. 今あるしくみを安易に否定しない
2. できることから1つずつ改善していく
3. 平行して(楽しみながら)理想的な姿に近づけていく

「Immutable Infrastructure」「Infrastructure as Code」「CI(Continuous Integration)」、それにこの連載に書いていないことなども含めて、ここ最近のインフラ周りの進化は目を見張るものがあってとてもテンションが上がります。格好良いし、未来感があります。ですが、いきなり未来の道具を入れようとすることよりも、今動いているしくみを少しずつでも改善しておくということは非常に大事だと思います。

本編では触れませんでしたが例を出すと、筆者が入社する前はweeklyデプロイだったのをdailyデプロイにしました。フットワーク軽くKPIを回すためです。モダンではないデプロイ方法だったので、デプロイタスク(stagingデプロイ、productionデプロイなど)だけで日に2、3時間取られていました。これでは改善のための時間が取れません。ということで、自前のデプロイツールを回収してdailyのデプロイ時間を30分に縮めました(stagingデプロイは自動化、productionデプロイは時間の短縮化)。

ひたすらshellスクリプトの書き直しです。テンションの上がる作業ではありません。ですが、1日2時間の短縮は相当大きいです。こういう時間を浮かせる開発は時間が経てば経つほど効果が出ます。たとえ改修に丸2日かかったとしても、完成すれば8日程度でペイします。後はひたすら時間の貯金ができます。

毎日やってることの時間が縮められる開発であれば、積極的に時間を割いたほうが良いと思います。「忙しいから改善する時間がない」ではなく、「改善しないから忙しい」んです。

それと平行して新しい技術(サービス)については適度に試しておくことも大事です(度合いが難しいですね)。それはインフラのアーキテクチャが単体で成り立つわけではなく、複合的な技術(サービス)の組み合わせだからだと思います。

AWSを例に取ってみると、「VMベースのインスタンスをどうデザインしていくのか?」に対して「Dockerのようなコンテナベースのしくみをどうデザインしていくのか?」という選択肢が出てきました。これはAWSのサービスがどうとかいう前に、Docker自体に触れておかないと選択の幅が狭まります。手元の環境でだけでもいいので触れておくと、サービスとして提供されたとき(本稿冒頭にもあるように、EBでDockerがサポートされたとか)に選択の余地がでます。

そして、技術(サービス)が想定している理想的な使い方(未来)に合わせてインフラデザインをしていくことも重要なことです。技術(サービス)を無理矢理自分たちの使い方に合わせることに執着しないようにしたいものです。

これまで書いてきたことは筆者がRettyで実践してきたことですが、何かの参考になれば幸いです。おつきあいくださりありがとうございました。これからもRettyをよろしくお願いします。SD

Rettyアカウント
http://user.retty.me/1059418/

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://d.hatena.ne.jp/rubikitch/
http://www.rubyist.net/~rubikitch/

第3回 反復練習に勝るものなし——打鍵すべし! 設定を書くべし!

Emacsは、ユーザに向かって開かれています。しかしその分厚い扉を何度でも叩き開こうとしなければ、Emacsのゾーン(zone)には入れません。何事も反復が大事なのです。今回は、困ったときの対処法とヘルプシステムおよび使いやすい初期設定を紹介します。

ちょっとずつ慣れてきましたか?

ども、Emacs廃人のるびきちです。本連載はEmacsを初めて使う人にでもEmacsのとりこになってもらうというコンセプトです。前回、前々回とEmacsの基礎概念について触れてきました。コードは1行も出てきていません。Emacsをすでに使っている人にとっては釈迦に説法だったと思いますが、今回からは実践的な内容になってきますので、楽しみにしててください。

前回の最後に紹介したように、<f1> t(F1を押したあとにtを押せば日本語チュートリアルが起動します。これによりEmacsの基本操作を手を動かしながら独習できます。Emacsは手を動かしながら学ぶのが一番です。まだEmacsに慣れていないのであればチュートリアルをしながらコマンド表を自分で作成してください。

そして、チュートリアルを何度か繰り返してください。子供の頃、何度も転びながらも、くじけずに練習した末やっと自転車に乗れましたよね。新たなスキルを習得するには反復練習が不可欠です。

Emacsの熟練度が低いうちは、ちょっとしたことで困るかと思います。たとえば、

・意図しないコマンドを実行してしまったとき
・打ち間違いをしたとき
・コマンドがわからない
・詳しい説明が欲しい
・いろいろと不便で使いづらい

などといったことです。ご安心ください。Emacsは困ったときにでも柔軟に対応できる力があります。その方法を初心者段階で知っておけば後々楽になることは間違いありません。今回は、困ったときにどうすればいいか、そして起こり得る不満を解消した設定をお話します。

はい、「設定」という言葉が出てきましたね。Emacsは「The extensible self-documenting text editor」と言われており、extensible(拡張可能)が大きな特徴です。好みの設定にするどころか、機能をガンガン拡張できるのです。ついに今回、初めてEmacs Lispのコードが登場します。

困ったときには

Emacsを使っていて、困ったと思うときがありますが、大きく分けて次の種類があります。

1. 現在のコマンドを抜ける➡C-g(Ctrl+g)
2. Emacsが固まる➡C-g
3. 編集をやり直す➡C-/(Ctrl+/)
4. カーソルが迷子になった➡C-/やC-u C-SPC

(Ctrl + u　Ctrl + Space)

5. 使い方がわからない➡ヘルプ

とにかく今の状況を抜け出す C-g

Emacsを使っていて行き詰まったら、とりあえずC-gを押してください。C-gはEmacsに「今の動作をやめろ」と伝えるコマンドです。

たとえば、C-x C-f(Ctrl + x　Ctrl + f)でファイルを開こうとしたとき、ミニバッファにてファイル名を訊いてきます。そこで「やっぱやーめた」という場合、C-gを押せばQuitと表示されてC-x C-fのコマンドがなかったことにされます。C-x b(Ctrl + x　b)などミニバッファを使うほかのコマンドでも同じです。

あるいは、時間がかかりすぎるコマンドを実行してしまったとき、それを取り止めたいケースもあります。この場合もC-gを押せばそのコマンドの動作はキャンセルされます。

たとえば、M-!(Alt + !)はシェルコマンドを実行するのですが、普通に使う場合は実行終了までEmacsは待ち状態になります。あまりに待たされて「時間かかりすぎ、もういいよ」と思ったときにはC-gを押します。なお、こういうコマンドを実行するときはM-!の代わりにM-&(Alt + &)を使えば、シェルコマンドを実行しながらEmacsが使えます。

極論を言えばEmacs Lispで無限ループを実行させても、C-gで中止できます。M-:(Alt + :)の後に(while t)を入力したとき、無限ループのためにEmacsが固まりますが、C-gを押せば何もなかったかのように元の状態に戻ります。

もし、C-gを押しても戻らないときは再度C-gを押してください。

C-gにはもともとEmacs Lispの実行を中止したり、ミニバッファから抜ける役割がありますが、モードによっては他のコマンドが割り当てられていることもあります。それでも「動作をやめろ」という意味のコマンドが割り当てられていることが普通です。それ以外のコマンドが割り当てられているときには、Emacsユーザは強い違和感を感じます。

編集を取り消す C-/

Emacsを使っていると意図しない編集をしてしまうことがあります。誤ったコマンドを実行して変な文字列が挿入されたとかそういうケースです。その場合は落ち着いてC-/(Ctrl + /)を押してください。これはundoというコマンドで、直前のバッファの変更を元に戻してくれます。

C-/を繰り返すと、過去の変更をどんどん元に戻していきます。そして、最初の変更まで遡った後さらにC-/を押すと、戻りようがないのでエラーになります。

Emacs標準のundoにはやや癖があります。undoしすぎた場合にundoを取り消すにはC-f(Ctrl + f)などの無害なコマンドを実行したあとC-/を押します。すると、「undo」という編集をundoするため、undoが取り消されるのです。この挙動が嫌な人はredo+.elを導入してください。

なお、undo情報を記録するかしないかは明示的に指定できます。デフォルトでは通常のバッファでは記録し、隠しバッファ(スペースから始まるバッファ名)では記録しません。もしなんらかの事情でundo情報が記録されていない場合は、M-x(Alt + x)buffer-enable-undoで記録を再開させてください。逆にM-x buffer-disable-undoで現時点での記録を破棄し、以後記録をしないようになります。

カーソル位置が迷子になったら

編集していくうちに現在のカーソル位置を見失ってしまうこともありますね。いわゆる迷子です。

もし、編集している位置を見失ってしまったときはundo情報を使います。C-/を押したとき、バッファの内容が元に戻るだけでなく、カーソル位置もundoされた位置に移動してくれるのでそれを利用してあげます。具体的にはC-/ C-f

C-/を押してundoしてそのundoを取り消します。これは便利なので、過去の編集位置に戻るコマンドを定義しているgoto-chg.elが存在します。

ほかにも意図せずに大域移動コマンドなどを実行してしまったときにも戻る方法があります。M-<(Alt + <)(バッファの先頭へ移動)やM->(Alt + >)(バッファの末尾へ移動)やM-g M-g(Alt + g Alt + g)(指定した行番号へ移動)やC-s(Ctrl + s)・C-r(Ctrl + r:インクリメンタルサーチ)がそれに該当します。これらのコマンドを実行した後には、後で戻れるように暗黙のマークがつけられます。いわば実行前にC-SPC(Ctrl + Space)が押されているようなものです。

マークは範囲(regionという)を設定するためだけに使われていると思いがちですが、実は位置を記憶するためにも使われています。過去のマークに戻るにはC-u C-SPC(Ctrl + u Ctrl + Space)を押します。繰り返し実行することで、もっと遡ることができます。

ヘルプシステム

Emacsのヘルプ機能はきわめて強力で、わからないことがあったらまずはEmacsに質問するべきです。

The extensible "self-documenting" text editor

前述のとおりEmacsはThe extensible self-documenting text editorと呼ばれています。先ほどはextensibleという点に触れましたが、今度はself-documentingという点に注目してください。自分自身を説明している、つまり、Emacsは自分自身についてよく知っているのです。これが何を意味するかというと、Emacsのコマンドや変数には説明文を入れることができ、いつでも参照できるのです。

多くのEmacsのコマンドはEmacs Lispというプログラミング言語によって記述されていますが、説明文を組み込む機能が言語レベルで搭載されています。これにより、標準コマンド以外にも外部Emacs Lispプログラムでもself-documentingの恩恵が受けられます。

self-documentingな性質は、Emacsに備わっているヘルプ機能で発揮されます。それではヘルプ機能について見てみましょう。

<f1> <f1>

ヘルプコマンドは<f1>(F1)またはC-h(Ctrl + h)から始まります。C-hはシェルではバックスペースとして使われていてヘルプには使いたくないと思うので、<f1>と覚えてください。

<f1>から始まるコマンドは多数存在しますが、今回はとりあえず<f1> <f1>(F1 F1)だけ覚えてください。これはヘルプのメニューがポップアップし、行頭に書いてあるキーを押すことでその項目を実行できるようになっています。たとえば、次の行はbを押せば現在使えるキーバインドをすべて表示します。最初から書くと<f1> <f1> b(F1 F1のあとにb)になります(**表1**)。

```
b           Display all key bindings.
```

各ヘルプコマンドはメニューを介しても介さなくても起動できます。たとえば<f1> <f1> bを縮めて<f1> b(F1のあとにb)とも操作できます。うろ覚えの場合はメニューから、そうでない場合は<f1>からというスタンスでかまいません。

プレフィクスキーから始まるコマンドのリストを表示する

Emacsには無数のコマンドがキーに割り当てられています。普通のGUIのショートカットでは1ストロークですが、Emacsでは2ストローク以上で起動するコマンドがたくさんあります。1ストロークでは収まりきれないほどのコマン

▼表1 主なヘルプコマンド(拙訳)

b	使えるキーバインドを表示する
c KEYS	そのキーのコマンド名をエコーエリアに表示する
e	エコーエリアのログ(*Messages*バッファ)を表示する
f FUNCTION	関数やコマンドの説明を表示する
F COMMAND	コマンドの説明をInfoで読む
i	Infoを開く
k KEYS	キーに割り当てられたコマンドの説明を表示する
K KEYS	キーに割り当てられたコマンドの説明をInfoで読む
l	最近300個のキーを表示する
m	カレントバッファのモードの説明・キーバインドを表示する
n	EmacsのNEWSを見る
r	emacsのInfoを開く
t	チュートリアルを開く
v VARIABLE	変数の値と説明を表示する
w COMMAND	コマンドが割り当てられたキーを表示する

ドを登録するには、2ストローク以上必要になるからです。

最も代表的な例としてC-x(Ctrl + x)から始まるキーが挙げられます。たとえばC-x C-f(Ctrl + x Ctrl + f)でファイルを開き、C-x C-s(Ctrl + x Ctrl + s)でファイルに保存します。このC-xをプレフィクスキーといいます。プレフィクスとは接頭辞という意味で、C-xを前置しているということです。C-x 4 C-fにおいては、C-x 4がプレフィクスキーです。一般にNストローク必要なコマンドにおいてN-1番目までのキーがプレフィクスキーになります。

<f1> b(F1のあとにb)は現在使えるキーバインドをすべてリストしますが、特定のプレフィクスキーから始まるものを列挙したいときがあります。プレフィクスキーを押したのはいいものの、その次がうろ覚えのときに役立ちます。

プレフィクスキーから始まるキーバインドを列挙するには、プレフィクスキーの後にC-hを押してください。たとえば、C-x C-hやC-c C-hのようになります。何気に地味な機能ですが、Emacsを使い始めて間がないときや、新しいEmacs Lispプログラムを使うときには重宝します。

設定ファイル

Emacsは自由自在に設定可能です。ちょっと挙動を変更したり、欲しい機能を導入したり、はたまた自分でコマンドを作成することもできます。それにより、Emacsは十人十色な使い方ができるのです。そこがEmacsの奥深さであり、楽しさであり、難しさでもあります。

設定ファイルは~/.emacs.d/init.elに書きます。Emacs設定ファイルの代名詞として「.emacs」と言うこともありますが、かつて~/.emacsや~/.emacs.elに設定を書いていたからです。今でもそれらに記述することもできますが、非推奨なのでinit.elに書いてください。

現在おもに使われているEmacsであるGNU Emacsは、今年で30歳になります。それくらい歴史のあるエディタです。Emacsの基本的な挙動や操作方法は昔と大きく変わらないように初期設定されています。Emacsは初期設定のままでも使えますが、歴史的事情から古くさいクセがあります。そのためデフォルトのままだと使いにくいと感じるケースが多々あります。

また、EmacsはEmacs Lispという拡張言語によっていくらでも拡張できるので、Emacsユーザは好き勝手に自分の欲しい機能が作れます。長年に渡り多くのEmacsユーザによって作られたEmacs Lispプログラムは膨大なものであり、同じ目的を達成するEmacs Lispプログラムが複数個あることは日常茶飯事です。現存のプログラムのちょっとしたことが気になってみんなこぞって新たに自作してしまうからです。

Emacsには標準添付のEmacs Lispプログラムが多数含まれています。その中でも同じ目的を達成するものが複数個含まれているものもあります。たとえば、バッファメニューのM-x ibufferやM-x bs-show、IRCクライアントの

M-x rcircやM-x ercといった具合です。

標準の挙動を改善するEmacs Lispプログラムもたくさんあります。M-x show-paren-modeは対応する括弧を知らせてくれます。M-x global-hl-line-modeは現在行をわかりやすくハイライトしてくれます。saveplace.elは開いたファイルの位置を永続的に記録して、再度開いたときにはその位置から開いてくれます。本来ならばテキストエディタとして当たり前とされる機能がなぜか無効になっていて、設定して初めて使えるということがEmacsではよくあるのです。せっかく標準添付なのだから有効にしないともったいないです。

外部Emacs Lispコミュニティの方も活発で、標準添付にはない斬新な機能が実装されていたりします。

このように、init.elに設定を書いていくことで、Emacsをあなた好みの使いやすいエディタに育てていけるのです。まるで新しい武器を見つけて装備するかのように、面白いEmacs Lispプログラムの設定を加えていくさまはRPGや育成ゲームを思わせます。

ちなみに筆者のinit.elの行数はかつて15,000行を超えていましたが、現在は4,500行くらいに落ち着いています。もう使わない設定を削除したり、関数群をパッケージングしたことで大幅に行数を削減できたのです。

さあ、設定を書いてあなたのEmacsも覚醒させましょう！

初期設定

それでは、おすすめの初期設定を**リスト1**に示しておきます。

Emacs24からパッケージシステムが標準でついてくるようになりました。PerlのCPAN、RubyのRubyGemsのように外部パッケージを簡単に導入できるようになりました。依存するパッケージもインストールしてくれるので、外部Emacs Lispプログラムを導入する手間を大幅に削減できます。この4行はおまじないと思ってinit.elの先頭に書いてください。

紹介したばかりのC-u C-SPCですが、過去のマークを辿るにはC-u C-SPC C-u C-SPC ……とC-uを毎回押す必要がありました。ですが、set-mark-command-repeat-popを設定することで、C-uを一度押した後はC-SPCを連続的に押すだけでマークを遡れます。

uniquify.elを使えばディレクトリ違いの同じファイル名のファイルを開いたときにバッファ名にディレクトリが付くようになります。デフォルトでは<2>などという数字が付くだけなのでバッファ名で識別できず不便です。

C-hはシェルと同様にBackSpace同様の挙動になるようにしています。シェルを使っていると打ち間違えたときに反射的にC-hを打つので、Emacsでもそろえておきます。

その他、表示関係を中心とした細々とした設定をしています。

一緒にEmacs力を高めませんか？

筆者は毎週土曜日にEmacsのメルマガを発行しています。多くの解説ではその機能の説明に終始しており具体例に乏しいため、理解するのにとても時間がかかるうえ、今の自分に必要なのかどうかを見極めることも難しいです。メルマガではチュートリアル形式で手を動かして学ぶ形式になっているので、たった5分でその内容を習得できるようになっています。わかりにくい資料で長時間悪戦苦闘するか、月々527円で時間差を買うかはあなた次第です。無制限メール相談権も付けています。初月無料なので安心して登録してください。

http://www.mag2.com/m/0001373131.html

Happy Emacsing！SD

▼リスト1　init.el（オススメ初期設定）本誌7月号サポートページに掲載予定

```
;;; パッケージの設定
(require 'package)
(add-to-list 'package-archives '("marmalade" . "http://marmalade-repo.org/packages/"))
(add-to-list 'package-archives '("melpa" . "http://melpa.milkbox.net/packages/") t)
(package-initialize)

;;; 右から左に読む言語に対応させないことで描画高速化
(setq-default bidi-display-reordering nil)

;;; splash screenを無効にする
(setq inhibit-splash-screen t)

;;; 同じ内容を履歴に記録しないようにする
(setq history-delete-duplicates t)

;; C-u C-SPC C-SPC ...でどんどん過去のマークを遡る
(setq set-mark-command-repeat-pop t)

;;; 複数のディレクトリで同じファイル名のファイルを開いたときのバッファ名を調整する
(require 'uniquify)
;; filename<dir> 形式のバッファ名にする
(setq uniquify-buffer-name-style 'post-forward-angle-brackets)
;; *で囲まれたバッファ名は対象外にする
(setq uniquify-ignore-buffers-re "*[^*]+*")

;;; ファイルを開いた位置を保存する
(require 'saveplace)
(setq-default save-place t)
(setq save-place-file (concat user-emacs-directory "places"))

;;; 釣合う括弧をハイライトする
(show-paren-mode 1)

;;; インデントにTABを使わないようにする
(setq-default indent-tabs-mode nil)

;;; 現在行に色をつける
(global-hl-line-mode 1)

;;; ミニバッファ履歴を次回Emacs起動時にも保存する
(savehist-mode 1)

;;; シェルに合わせるため、C-hは後退に割り当てる
(global-set-key (kbd "C-h") 'delete-backward-char)

;;; モードラインに時刻を表示する
(display-time)

;;; 行番号・桁番号を表示する
(line-number-mode 1)
(column-number-mode 1)

;;; GCを減らして軽くする
(setq gc-cons-threshold (* 10 gc-cons-threshold))

;;; ログの記録行数を増やす
(setq message-log-max 10000)

;;; 履歴をたくさん保存する
(setq history-length 1000)

;;; メニューバーとツールバーとスクロールバーを消す
(menu-bar-mode -1)
(tool-bar-mode -1)
(scroll-bar-mode -1)
```

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4回　AWS利用環境の構築(中編)　請求関連の設定

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

2014年5月上旬に「AWS認定SysOpsアドミニストレーター――アソシエイトレベル」の日本語での提供が始まりました。これはおもにAWS上のシステムを運用するエンジニア向けの認定試験で、昨年5月に日本語での提供が開始された「AWS認定ソリューションアーキテクト――アソシエイトレベル」に続く2つめの認定区分となるものです。筆者もGW連休明けすぐに受験してきましたが、実践的でかつ比較的詳細な知識が問われ、歯応えのある問題が多かったように思います。

業務で認定資格を必要とするエンジニアの方々はもちろん、自己研鑽などでAWSに触れているエンジニアのみなさんにとっても、AWSに関する知識レベルを客観的評価で確認するためのツールとして利用してみても良いのではないかと、受験を終えてほっと一息つきながら思ったのでした。さて今月号の本題に入ります。

今回の流れ

前回は、AWSアカウントの作成と、AWS APIを快適に利用するうえでやっておくと良い次の4つの作業のうちAWS CloudTrailとIAMユーザの作成について解説しました。

1. AWS CloudTrail(APIのロギング)の設定
2. 作業用AWS Identity and Access Management(以下IAM)ユーザの作成
3. 請求関連の設定
4. 多要素認証(MFA)の設定(次号掲載予定)

今回は請求関連の設定を解説します(MFAは次号予定です)。

請求関連の設定

請求レポートの設定

まず、請求に関する設定をしていきましょう。AWSでは、請求関連の機能についてAPIが公開されていないため、基本的にWebブラウザから設定します。

なお、第2回の記事でも言及しましたが、AWSマネジメントコンソールの操作においては、Google Chromeの利用を推奨いたします。

月次請求書の設定

AWSでは、Billingマネジメントコンソールにアクセスし、月額利用料をダッシュボードで目視確認したり、請求書データをCSV(*Comma Separated Values*; カンマ区切り)形式でダウンロードしたり印刷することができます(**図1**)。

しかし、毎月決まったタイミングで手動で確認するのは、手間でもあり忘れてしまうことも

多いので、電子メールでPDF形式の月次請求書を受け取る設定をしておきましょう。

手順①

Billingマネジメントコンソールの設定画面(**図1**)にアクセスします。AWSマネジメントコンソールからは、右上の[アカウント名]→[Billing & Cost Management]→[Preference]の順にクリックすると設定画面が表示されます。

手順②

設定画面(**図2**)では、右下の[Choose Language(言語の選択)]欄のプルダウンで日本語を選択できます。

手順③

[電子メールでPDF版請求書を受け取る]にチェックを入れて、ページ下の[設定の保存]ボタンをクリックします(**図2**)。これで月次請求書の設定は完了です。

▼図1　Billingマネジメントコンソール
(https://console.aws.amazon.com/billing/home#/)

▼図2　電子メールでPDF版請求書を受け取る

請求レポート(Billing Report)の設定

月次請求書よりも詳細なレポートが欲しい場合は、Amazon Simple Storage Service(以下「S3」)経由で時間単位での請求レポートを受けとれます。この設定をすると、Billingマネジメントコンソールの[レポート]画面やS3マネジメントコンソールからレポートファイルをダウンロードしたり、AWS API経由で請求レポートのデータにアクセスできるようになります。

手順①　Billingマネジメントコンソールでの設定

先ほどのBillingマネジメントコンソール設定画面で、[S3バケットに保存:]と記載された入力フォームに作成予定のバケット名を入力します。ここでの名前はAmazon S3全体でユニークである必要があるので、たとえば"組織名-billing"というような名称を入力します。すぐ下にある[ポリシー]というリンクをクリックするとサブウィンドウが開きます(**図3**)。下半分に表示されているポリシー定義[注1]を全選択してクリップボードにコピー(Mac OS Xの場合

注1) このポリシー定義は、[S3バケットに保存:]に入力したバケット名が反映された内容になっています。手順②で、作成しようとしているS3のバケット名がすでに誰かに使われていた場合は、再度[S3バケットに保存:]の入力からやりなおすか、ポリシー定義の該当部分を修正する必要があります。

は[Command]+[C])します。

コピーしたら[完了]ボタンをクリックします(サブウィンドウが閉じます)。

●手順② S3マネジメントコンソールでの設定

次にS3マネジメントコンソール(https://console.aws.amazon.com/s3/home)にアクセスし、[Create Bucket]ボタンをクリックします。先ほどポリシー定義をコピーするときに決めておいたバケット名を入力し、[Create]ボタンをクリックします。バケットの一覧に、今作成したバケットが表示されるのでリンクをクリックします。そして右上にある[Properties]ボタンをクリックします。Permissionsタブ内にある[Add bucket policy]ボタンをクリックするとBucket Policy Editorウィンドウが開きます(**図4**)。

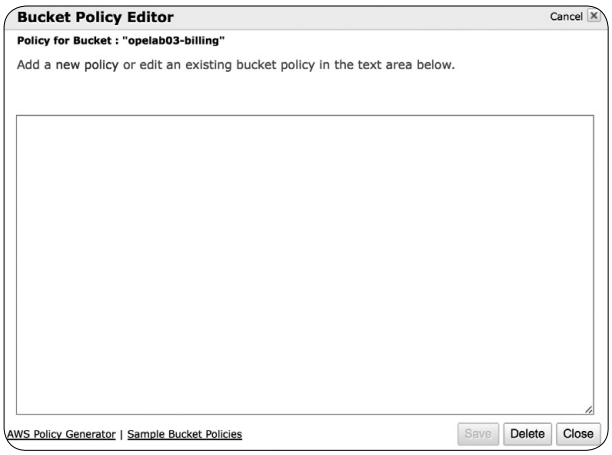
先ほどコピーしたサンプルポリシーを貼り付け(Mac OS Xの場合は[Command]+[V])、[Save]ボタンをクリックするとサブウィンドウが閉じます。

手順③ Billingマネジメントコンソールでの設定

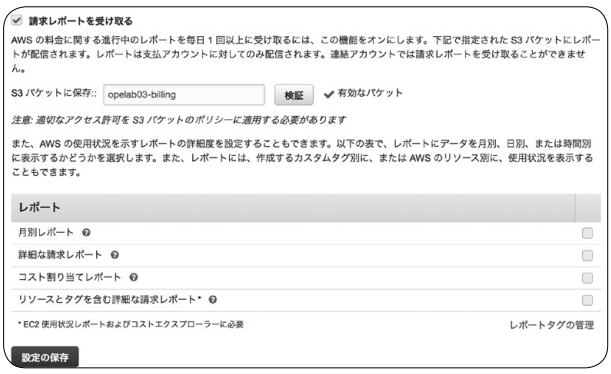
ふたたび、Billingマネジメントコンソールの設定画面(**図1**)にアクセスします。[請求レポートを受け取る]にチェックを入れ、[S3バケットに保存]のフォームに先ほど作成したバケット名を入力し[検証]ボタンをクリックします。[検証]ボタンの右側にグリーンのチェックマークとともに[有効なバケット]と表示され、選択できるレポートの一覧が表

▼図3 サンプルポリシーをコピーしておく

▼図4 Bucket Policy Editorウィンドウ



▼図5 請求レポートの受け取り方の選択



示されるようになります(**図5**)。[レポート]欄の全項目にチェックを入れ、一番下の[設定の保存]ボタンをクリックして設定を保存します。

以上で請求レポートの設定は完了です。

請求アラートの設定

請求アラート(Billing Alerts)の設定

AWSでは請求金額が想定金額を超えたときにアラートを上げるしくみが用意されています。これはAmazon CloudWatch(以下「CloudWatch」)とAmazon Simple Notification Service(以下「SNS」)を利用します。

具体的には、CloudWatchがSNSに対してアラーム(Billing Alerms)を送信し、SNSはそのアラームを(たとえば経理などの)担当者に対してアラート(Billing Alerts)としてプッシュ通知する役割を担います。請求金額にビクビクしながら使うのは不安だと思いますので、ぜひ設定しておきましょう。

手順①　SNSトピックの作成

SNSは、HTTP/HTTPSやE-mail、SMS、モバイルプッシュ通知など多様な方法でメッセージをプッシュできるサービスです。

SNSとよく似たサービスに、Amazon Simple Queue Service(SQS)がありますが、SNSはプッシュ型サービス、SQSはプル型サービスという違いがあります。いずれのサービスもシステム間のメッセージ交換に使われますが、SNSは通知の中継用途で使われることも多いのが特徴と言えます。

またSNS/SQSともに、メッセージを送信した側(今回はCloudWatch)はそのメッセージが最終的にどこに渡されるかを意識する必要がないため、疎結合なシステムを構築するうえでは欠かせない、AWSの中でも非常によく使われる重要なコンポーネントとなっています。

SNSでは、トピックという通知の一次受付となる器のようなものを作成します。トピックに通知されてきたメッセージは、そのトピックの設定内容によって、関係者に対してメールで通知されたり、ほかのシステムに対してHTTPリクエストで通知されたりします。ここでは、BillingAlertという名前のSNSトピックを作成し、amz@example.jpというメールアドレスにメールをプッシュする設定をAWS CLIで行う手順について紹介します。

まず、請求アラート用のトピックをSNSに作成します。

・コマンド：

```
% aws sns create-topic --name BillingAlert
```

・結果(例)：

```
{
  "TopicArn": "arn:aws:sns:us-east-1: ➥
xxxxxxxxxxx:BillingAlert"
}
```

トピックを作成したら、トピックに届いたメッセージのプッシュ通知先としてメールアドレスを登録します。

・コマンド：

```
% aws sns subscribe --topic-arn `aws sns ➥
list-topics | grep BillingAlert | sed -e ➥
's/¥"//g' | awk '{print $2}'` --protocol ➥
email --notification-endpoint amz@example.jp
```

・結果：

```
{
  "SubscriptionArn": "pending ➥
confirmation"
}
```

通知先として登録されたメールアドレスに確認用のメールが送信されてきます(**図6**)。

メール本文中の[Confirm subscription](リンク)をクリックすると、Amazon SNSが生成した[Subscription confirmed!]のページが表示され、そのメールアドレスに関して登録が完了します。

登録が完了したことをAWS CLIで確認して

みましょう。

・コマンド：

```
% aws sns list-subscriptions-by-topic ➥
--topic-arn `aws sns list-topics | grep ➥
BillingAlert | sed -e 's/¥"//g' | awk ➥
'{print $2}'`
```

・結果（参考例）：

```
{
  "Subscriptions": [
    {
        "Owner": "xxxxxxxxxxx",
        "Endpoint": "amz@example.com",
        "Protocol": "email",
        "TopicArn": "arn:aws:sns:us-east- ➥
1:xxxxxxxxxxxx:BillingAlert",
        "SubscriptionArn": "arn: ➥
aws:sns:us-east-1:xxxxxxxxxxx: ➥
BillingAlert:xxxxxxx-xxxx-xxxx-xxxx- ➥
xxxxxxxxxxx"
    }
  ]
}
```

SubscriptionArnの値が、**図6**で示したように“arn:aws”で始まる値になっていればSNSでの設定は完了です。

手順② Billingマネジメントコンソールでの設定

ふたたび、Billingマネジメントコンソールの設定画面（**図1**）にアクセスします。

［請求アラートを受け取る］にチェックを入れ、一番下の［設定の保存］ボタンをクリックして設定を保存します。

設定を保存したら［請求アラートを受け取る］の設定項目[注2]のところに表示されている［請求アラートを管理する］と書かれたリンク（**図7**）をクリックし、CloudWatchの設定画面にアクセスします。

注2） Billing AlertsはバージニアリージョンのCloudWatchマネジメントコンソールで設定するため、上記リンクをクリックするとCloudWatchのダッシュボードに移動します（ https://console.aws.amazon.com/cloudwatch/home?region=us-east-1）。

手順③ CloudWatchマネジメントコンソールでの設定

CloudWatchは、AWSクラウド上で利用できるモニタリングサービスで、AWSリソースやユーザのアプリケーションやサービスを監視できます。請求関連では、請求アラームの設定があらかじめ用意されており、これを有効にすることで請求金額が一定額（ドル建て）に達したときにアラームを上げられます。

では、次にCloudWatchの設定をします。CloudWatchマネジメントコンソールの左のリストからBilling[注3]を選択すると、Billing Alarmsの設定画面が表示されます（**図8**）。

［Create Alarm］ボタンをクリックします。［exceed］欄にアラームの閾値（いきち）となる金額（たとえば“25USD”）を入力し、［send a notification to:］欄で先ほど作成したSNSのトピック名（例では“BillingAlart”）を選択し、［Create Alarm］

注3） バージニアリージョン以外のCloudWatchマネジメントコンソールでBillingを選択するとエラーが出ます。

▼図6　確認用のメールが送信される（SNSでの例）

差出人: Notifications AWS >　隠す

宛先: [redacted] >

AWS Notification - Subscription Confirmation

2014年4月28日 12:36

You have chosen to subscribe to the topic:

arn:aws:sns:us-east-1:[redacted]:BillingAlert

To confirm this subscription, click or visit the link below (If this was in error no action is necessary):

Confirm subscription

Please do not reply directly to this e-mail. If you wish to remove yourself from receiving all future SNS subscription confirmation requests

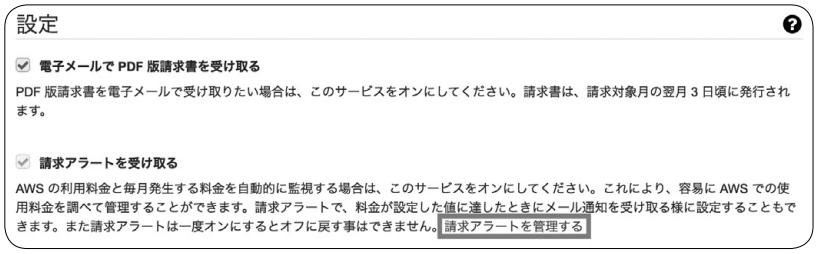
ボタンをクリックします(**図9**)。

CloudWatchマネジメントコンソールのDashboard(**図1**)にアクセスし、[Alerm Summary]欄の[BillingAlarm]に緑色のチェックマークが表示されれば(**図10**)、請求アラートの設定はすべて完了となります。

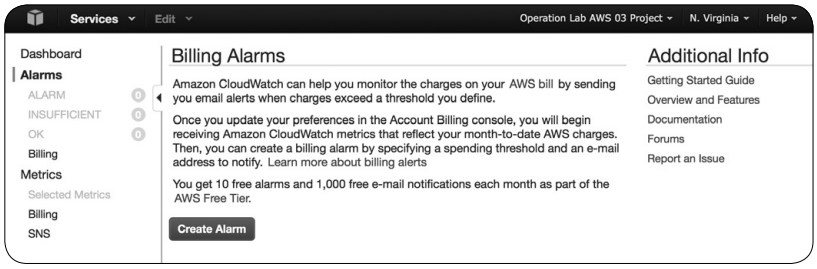
次回は

ここまでの設定で、AWSアカウントでなければできないことは請求情報の変更や請求レポートの閲覧以外ほぼなくなり、日常運用については前回作成したIAMユーザで行うことができるようになりました。普段使わなくなったAWSアカウントについては、多要素認証(Multi-Factor Authentication)を導入して防御を固めておきたいところです。次号では、このMFAの設定について解説する予定です。SD

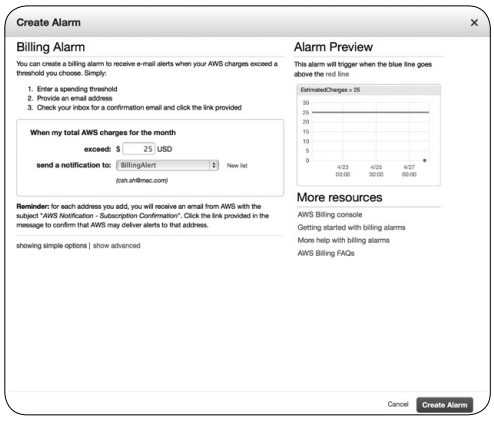
▼図7　請求アラートを管理する



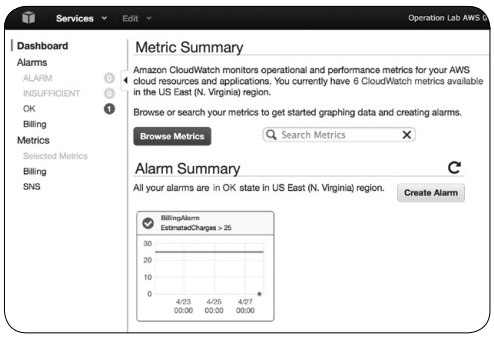
▼図8　Billing Alarms



▼図9　Create Alarmの画面下方で、[Create Alarm]ボタンをクリック



▼図10　[BillingAlarm]に緑色のチェックマークを確認



第48回

アプリの脆弱性を気にしてますか?

学習・点検ツール「AnCoLe」

モバイルデバイス初のオープンソースプラットフォームとして、エンジニアから高い関心を集めるGoogle Android。いち早くそのノウハウを蓄積したAndroidエンジニアたちが展開するテクニックや情報を参考にして、大きく開かれたAndroidの世界へ踏みだそう!

谷口 岳　TANIGUCHI Gaku
タオソフトウェア㈱
Twitter @tao_gaku

開発者のセキュリティ意識と知識を高めたい

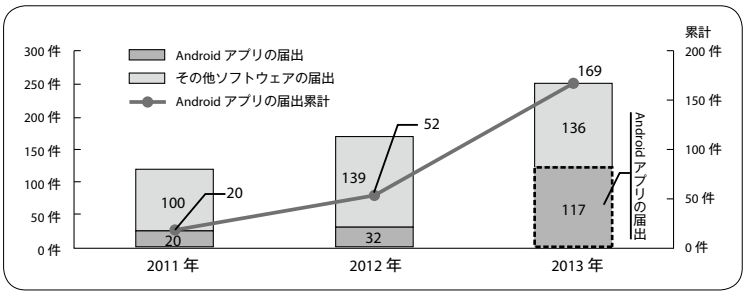
ここ数年のスマートフォンの普及速度はめざましく、従来型の携帯電話ではオマケ扱いだったアプリケーションの使用者は、携帯電話＝スマートフォンを使用する人すべてとなり爆発的に増加しました。また、アプリケーションのマーケットサイトが気軽に利用できるサービスとして提供され、決済が非常に楽になり、個人でも簡単にアプリケーションを販売できるようになりました。このため、今までアプリケーション開発に縁のなかった企業や個人もアプリケーションを作り、リリースしています。しかしながらセキュリティについて十分に考察されないまま開発し、公開されているケースが多く見られます。そもそもセキュリティって何?という開発者も存在します。

IPA(独立行政法人情報処理推進機構)による、ソフトウェア等の脆弱性(ぜいじゃくせい)についての届け出件数に関する統計データ(**図1**)を参照すると、2011年には2割に満たなかったAndroidアプリケーションの脆弱性が2013年には約5割を占め、件数も大幅に増加しています。この件数は届け出件数なので、氷山の一角と言えるでしょう。発見しても届け出をしていないもの、また発見されていないものを含めると膨大な数になると考えられます。

ユーザが脆弱性のあるアプリケーションを使用すると、ユーザの情報が漏えいしたり、改ざんされるなどの被害が発生する恐れがあります。故意にユーザの情報を盗み取る"悪意のあるアプリケーション"も存在しますが、悪いことをする気がなくても、悪意のあるアプリケーションと間違えられて炎上したり、あなたが作成したアプリケーションの脆弱性を悪意のあるアプリケーションが攻撃することで、ユーザに被害を与えることもありえます。

この先、誰もが安心して利用できるアプリケーション市場にしていくためには、このようなことが起らないように、より多くの開発者がAndroidアプリケーションにおける脆弱性に対して

▼図1　IPAによるAndroidアプリの脆弱性の届け出件数



引用元:ソフトウェア等の脆弱性関連情報に関する届出状況[2013年第4四半期(10月～12月)]
http://www.ipa.go.jp/security/vuln/report/vuln2013q4.html

の基本的な対策方法を習得する必要があります。このような背景から、脆弱性が作り込まれてしまう原因や対策について実習形式で学べるようにと開発されたのが、今回紹介する「AnCoLe（アンコール）」です。

AnCoLeの概要

AnCoLeは、2014年4月11日にIPAから無償でリリースされた自由に使える学習ツールです。なぜ無償かといえば、IPAは「日本におけるIT国家戦略を技術面、人材面から支える」ことを目的とした独立行政法人だからです。IPAといえば情報処理技術者試験の運営で有名ですが、「利用者視点に立った複雑、膨大化する情報社会システムの安全性・信頼性の確保」を理念として活動しており、セキュリティに関していろいろな取り組みをしています。たとえばWebサイトからの個人情報などの流出は、古くて新しいニュースでなかなか収まりませんが、IPAではこの問題に対処するために、Webの脆弱性体験学習ツール「AppGoat」を無償で提供しています。

今回リリースされたのは、Androidアプリの脆弱性体験学習ツールになります。

インストール

インストールは非常に簡単で、AnCoLeのサイト[注1]から「ancole.zip」をダウンロードし、解凍したzipファイルに含まれるjarファイルをEclipseの「dropins」フォルダにコピーするだけです。

公式では対応OSはWindowsのみとなっていますが、Eclipseのプラグインとして提供されていますのでMacでも動きます。現在の開発環境がプラグインで汚れるのが嫌だという人は、Eclipseをもう1つインストールしてから動かすのがいいかもしれません。

jarファイルをコピーした後、Eclipseを起動すると、鉛筆アイコンと虫眼鏡アイコンのボタンがツールバーに追加されます（**図2**）。AnCoLeはこの2つのアイコンが起点となります。

AnCoLeの機能

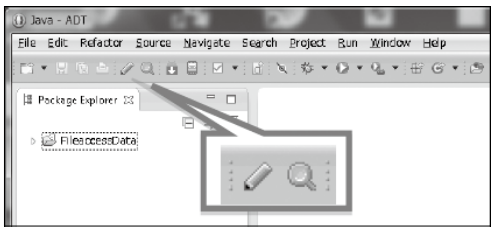
2つのアイコンは、鉛筆アイコンが「学習機能」、虫眼鏡アイコンが「点検機能」です。AnCoLeには大きく分けるとこの2つの機能が存在します（**図3**）。実際にアプリケーションを公開されている方は、点検機能で自分のアプリケーションを点検することから始めるのがいいでしょう。体系的に脆弱性について学びたい方は、学習機能に含まれるシナリオを通して、アプリケーション開発時に注意すべき脆弱性の情報や脆弱性への対策方法について学習をしてください。

学習機能について

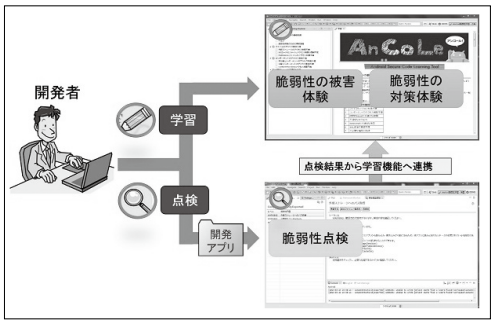
AnCoLeには次の7つの学習テーマが存在します。

1. ファイルアクセス制限の不備
2. コンポーネントのアクセス制限不備

▼図2　ツールバーアイコン



▼図3　AnCoLeの機能　©2014 IPA



注1）http://www.ipa.go.jp/security/vuln/ancole/download.html

3. 暗黙的Intentの不適切な使用
4. 不適切なログ出力
5. WebViewの実装不備
6. SSL通信の実装不備
7. 不必要な権限の取得

各テーマには複数のシナリオ（合計11）が入っています。詳しい内容についてはAnCoLeのドキュメントを参考にしてください。

また、AnCoLeの1つの特徴は、実際に脆弱性のあるアプリケーションを作製して攻撃ができることです。攻撃を体験できることで、どのようにして守る必要があるのかを実感することが可能です。

SDカードのファイルアクセス制限の理解不足シナリオを誌上体験

各シナリオは次のような流れで進みます。読むだけでなく、サンプルファイルを編集、実際に攻撃も行うため、じっくりと腰を据えると1シナリオ20分ぐらい学習時間がかかります。

❶脆弱性の概要説明
❷サンプルコードの解説
❸脆弱性体験
❹サンプルアプリの脆弱性説明
❺脆弱性対策方法の説明
❻脆弱性対策の体験

ここでは「1. ファイルアクセス制限の不備」テーマの中の「SDカードのファイルアクセス制限の理解不足」シナリオを通して、どのような流れで学習が進むのかを解説します。

▼図4　脆弱性のあるソースコード

```
81  public void onClickSave(View v) {
82      // 入力内容の取得
83      String data = csvData.toString();
84      FileOutputStream fileOutputStream = null;
85
86      // 既存のファイルを一度削除した後、新たにファイルを作成
87      File file = new File(SD_FILE_PATH);
88      file.delete();
89
90      file.getParentFile().mkdir();
91
92      try {
93
94      // ▼▼▼脆弱性のあるソースコード▼▼▼
95      // 書き込み処理を行う
96      fileOutputStream = new FileOutputStream(file, true);
97      fileOutputStream.write(data.getBytes());
98      // ▲▲▲脆弱性のあるソースコード▲▲▲
```

❶ 脆弱性の概要説明

学習対象の脆弱性がどのようなものか、どのような被害につながるのかを学習します。実際に報告されている事例についても解説します。

SDカード上にファイルを保存する場合と、内部ストレージ上にファイルを保存する場合では大きな違いがあります。内部ストレージ上に保存したファイルは、アプリケーション自身が許可しない限り、ほかのアプリケーションからは読み込みや書き込みのアクセスができません。一方、SDカード上に保存したファイルは、SDカードにアクセスできるすべてのアプリケーションからファイルの読み取りが可能になります。したがって重要なデータをSDカード上に保存した場合、その情報が盗まれる可能性があります。

❷サンプルコード解説

脆弱性のあるサンプルコードを解説します。

図4は脆弱性のあるコードです。ファイルをSDカード上に作成しているため、ファイル内に書かれたデータが重要なデータの場合は脆弱性となります。

❸脆弱性体験

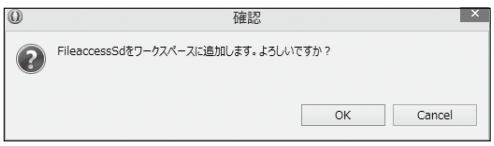
実際に脆弱性があるアプリケーションを実行し、攻撃を行います（サーバを立てる必要があったり、体験が簡単にできないものに関しては体験が含まれていない場合もあります）。

■ソースインポート

実際に脆弱性のあるアプリケーションを作製

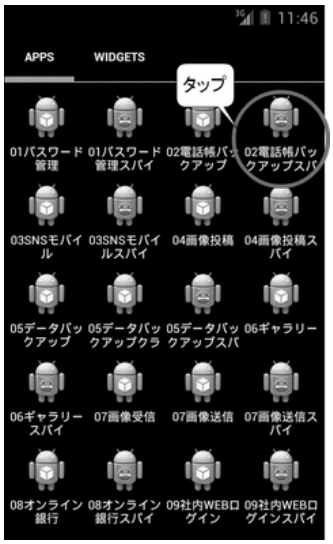
▼図5　ソースインポートボタン

▼図6　ソースインポートの確認ダイアログ



▼図7　脆弱性のあるアプリ実行

▼図8　攻撃アプリの起動



▼図9　攻撃アプリによるデータ取得

して攻撃を体験します。脆弱性のあるコードを手入力する必要はありません。AnCoLeには脆弱性があるサンプルアプリケーションが用意されているので、「サンプルアプリをインポート」のボタンを押して、プロジェクトをインポートします(**図5、6**)。

インポート後、コンパイルエラーが出る場合は、対応するAndroid SDKがインストールされていない場合がほとんどです。詳細はAnCoLeのドキュメントを参照してください。

■攻撃アプリの準備

攻撃アプリケーションもAnCoLeに付属しています。「攻撃アプリをダウンロード」ボタンを押すと、ファイルの保存先選択ダイアログが表示されます。次の2つのファイルが保存されます。

- FileaccessSdSpyware.apk
- Install.bat

攻撃用アプリケーションは、FileaccessSdSpyware.apkです。Install.batは中にadb installコマンドが記載されているだけなので、必要に応じて使用してください。

■各アプリの実行

実機もしくはエミュレータに、コンパイルした脆弱性のあるサンプルアプリケーションと攻撃アプリケーションの両方をインストールします。サンプルアプリケーションは電話帳をバックアップするアプリケーションです。したがって端末の電話帳にデータが入っている必要があります。そこで、サンプルの脆弱性アプリケーションを動作させたときに、電話帳にデータが1件もない場合は電話帳にテストデータを入れるといった便利機能も付いています。

まず脆弱性のあるアプリケーションを実行します(**図7**)。そして、「電話帳バックアップ」ボタンを押します。このとき、このアプリケーションは端末の電話帳からデータを吸い出し、SDカード上にデータを保存します。

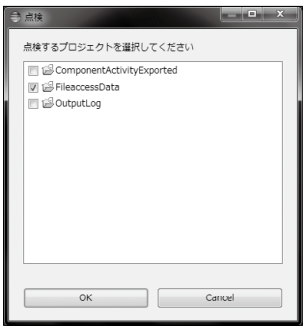
次に攻撃アプリケーションを実行します(**図8**)。この攻撃アプリケーションには電話帳へのアクセス権限はありませんが、SDカード上のデータを読み込むことで電話番号を取得しています(**図9**)。この攻撃アプリケーションはサンプルとしてバックアップした電話帳の内容を画面に表示しましたが、作り変えることで内容を利用者に気づかれないように外部に送信することも可能になります。

▼図10　脆弱性の修正

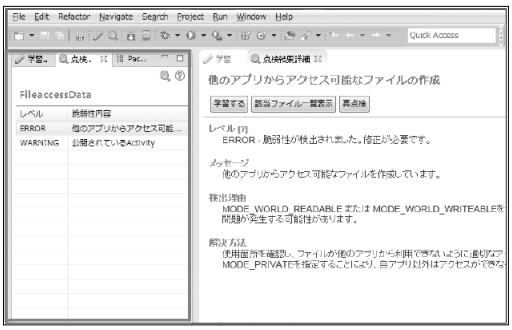
```
81  public void onClickSave(View v) {
82      // 入力内容の取得
83      String data = csvData.toString();
84      FileOutputStream fileOutputStream = null;
85
86      try {
87
88          // ▼▼▼脆弱性を修正したソースコード▼▼▼
89          // 書き込み処理を行う
90          fileOutputStream = openFileOutput(FILE_NAME,  MODE_PRIVATE);
91          fileOutputStream.write(data.getBytes());
92          File file = getFileStreamPath(FILE_NAME);
93          // ▲▲▲脆弱性を修正したソースコード▲▲▲
94
95          // 完了メッセージ表示
96          Toast.makeText(this, String.format("%s\r\nにバックアップが完了しました。",
    <
119 public void onClickDelete(View v) {
120     deleteFile(FILE_NAME);
121     Toast.makeText(this, "ファイルを削除しました", Toast.LENGTH_LONG).show();
122 }
```

▶図11　攻撃アプリによる脆弱性修正確認

▼図12　プロジェクトの選択



▼図13　点検結果一覧画面



❹サンプルアプリの脆弱性説明

脆弱性のあるサンプルコードを解説し、脆弱性の原因について理解します。

実際に自分で攻撃することで、確かに問題のあるアプリケーションであるのがよくわかったと思います。脆弱性説明セクションでは、どのような問題があるのかを詳しく説明します。

ここでは、電話帳のデータをSDカードにそのまま保存する設計が良くありませんでした。内部ストレージに保存、暗号化して保存、サーバに保存するなど解決策はいろいろありますが、ほかのアプリケーションからアクセスできないように設計すべきです。

❺脆弱性対策方法の説明

脆弱性の対策方法について学習し、実際にサンプルアプリケーションのソースコードを修正します。

どのように修正すべきかソースコードレベルで解説されます。たとえば内部ストレージにデータを保存するように変更します(図10)。

❻脆弱性対策体験

修正したアプリケーションを実行し、攻撃アプリケーションを使って攻撃します。対策されたことにより攻撃が成立しないことを確認します。

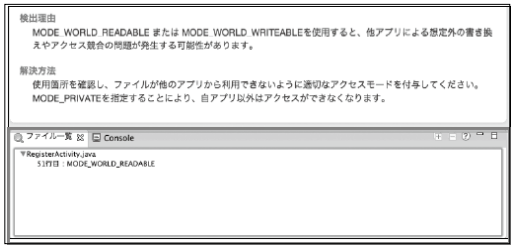
脆弱性体験のときと同じように、修正したアプリケーションをインストールし実行して、攻撃アプリケーションを動かします。攻撃アプリケーションで電話帳が参照できないのが確認できます(図11)。

以上で学習機能の体験は完了です。

点検機能について

点検機能はAnCoLeの大きな機能の1つです。既存アプリケーションのソースコードを読み込ませることで、脆弱性になり得る個所がないか

▼図14　脆弱性が存在するファイル一覧



を確認できます。

点検機能を使用するには、虫眼鏡アイコンをクリックします。するとプロジェクト選択画面になりますので、点検するプロジェクトを選択します(**図12**。あらかじめ、点検するプロジェクトをEclipseにインポートしておいてください)。

プロジェクトを選択すると自動的に点検作業を開始し、「点検結果一覧」画面が表示されます(**図13**)。左側に検出された脆弱性リストが表示され、クリックすると右側にその説明が表示されます。右側に表示されている「学習する」ボタンを押すと、該当する脆弱性に関する学習コンテンツページに移動します。これにより脆弱性に関する詳しい学習が可能になります。

図13で「該当ファイル一覧表示」ボタンを押すと、画面の下に「ソースコード一覧」が表示されます(**図14**)。検出された脆弱性が含まれるソースコードと行番号、内容のサマリが表示され、ファイル名をクリックするとソースコード編集画面が表示されます(**図15**)。

このように、点検機能で検出された内容の理解からソースコードまでを連続して表示することが可能なため、手間なく修正が可能となります。修正した後は、再度点検をかけて問題がないことを確認します。

注意すべき点は、AnCoLeで検出されるすべての項目を消すのを目的にはしないことです。アプリケーションの仕様によっては削除不可能な項目もありますし、「AnCoLeの点検機能で1件も検出されないこと」を目標にしてしまうと、脆弱性を理解しないまま修正を行うため、脆弱性が修正されない事態も起こる可能性があります。

▼図15　脆弱性があるソースコード

また、AnCoLeは学習ツールであるため、学習内容は比較的理解しやすい事項に絞っています。アプリケーションを作製するうえで気を付けなければいけないことはほかにもありますので、AnCoLeで点検したから安全というわけではない点に注意してください。

最後に

本記事では、IPAからリリースされたAnCoLeの解説をしました。無料で使用できますので、多くのAndroid開発者の方に使ってもらい、世の中から脆弱性が少しでもなくなっていけば良いと思っています。

最近はセキュリティに関する仕事をしておりますが、セキュリティの分野は注意をしている人には情報が届くのですが、すべての人に情報を届けるのは非常に難しいと感じています。今回少しでも役に立てばと思い、この記事を書かせていただきました。Androidの開発者の方はぜひ一度使ってみてください。また知り合いに開発者がいる方は、こんなツールがあるよと伝えていただけたらうれしく思います。SD

谷口 岳（たにぐち がく）　タオソフトウェア株式会社 代表取締役

Android OSの発表直後からAndroid開発を開始。Androidのセキュリティ本を出版後、Androidアプリの脆弱性検査ツール「TaoRiskFinder」をリリース。Androidのセキュリティに関する情報を広める活動を行っている。

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3回 開発フローを考慮したサーバの安定運用

前回はRed Hat Enterprise LinuxとRPMのバージョンについて紹介しました。今回はさらにそれらを開発するフローについて理解し、サーバを可能な限り安定的に運用する方法について考えてみましょう。

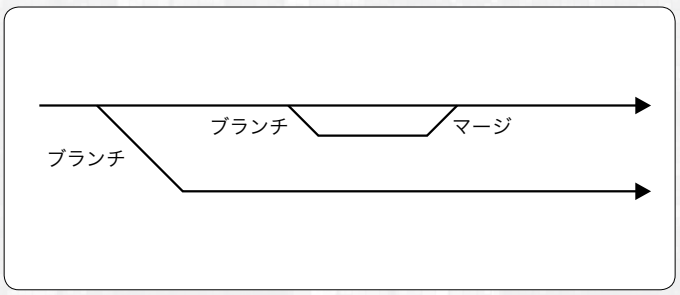
レッドハット(株)グローバルサービス本部プラットフォームソリューション統括部
ソリューションアーキテクト部長　藤田 稜(ふじた りょう)

アップストリームとRHELの関係

ブランチはマージされる

OSSに慣れ親しんだ方であれば、アップストリームという言葉は聞いたことがあると思います。Linuxのカーネル[注1]であればkernel.org、OpenStackであればopenstack.orgがコミュニティの中心であり、これをアップストリーム・「上流」と呼びます。さらにその成果物、つまりソースコードのツリーが上流となり、そこから派生した下流は枝分かれしていることから「ブランチ」と呼びます。ブランチは「マージ」という工程によって、メインのツリー、つまり主流に吸収されることもあります(**図1**)。

▼図1　ブランチとマージの関係(主流に吸収される場合)

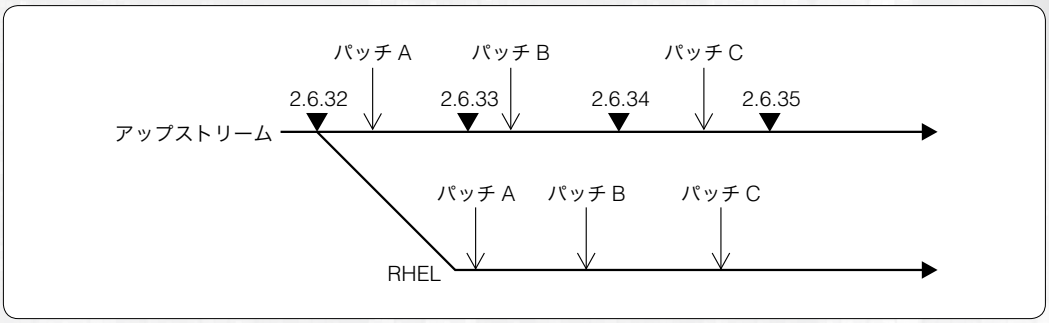


アップストリーム・ファースト

レッドハットの開発フローは前述のアップストリームを起点としており、このことを「アップストリーム・ファースト」と呼んでいます。Linuxカーネルを例にとればRed Hat Enterprise Linux(以降、RHEL)のカーネルはアップストリームに対するブランチとなります[注2]。アップストリームでは、さまざまな企業に属する、あるいは個人として参加している人々がソースコードに対するパッチを開発しています。さて、ここである問題がカーネルに発見され、そのパッチをレッドハットのカーネルエンジニアが書いたとしましょう。するとそのパッチは社内のメーリングリストに投稿され、シニアのカーネルエンジニアによってレビューを受けます。レビューをパスすると、しかるべきアップストリームのメーリングリスト[注3]に投稿されレビューを受け、場合によっては修正されたうえでコミットされます。そして、そのコミットされたパッチがRHELのカーネ

注1) Linuxは狭義にはカーネルそのものを指すが、ここでの"Linux"はOSのディストリビューションを指す。
注2) RHELをベースに開発されているCentOSなどは、「RHELのブランチ」という見方ができる。
注3) Linuxカーネルはネットワークやファイルシステムなど、個別の機能ごと(サブシステム)に開発責任者が置かれメーリングリストが運用されている。

▼図2 アップストリーム・ファースト（アップストリームとRHELの関係）



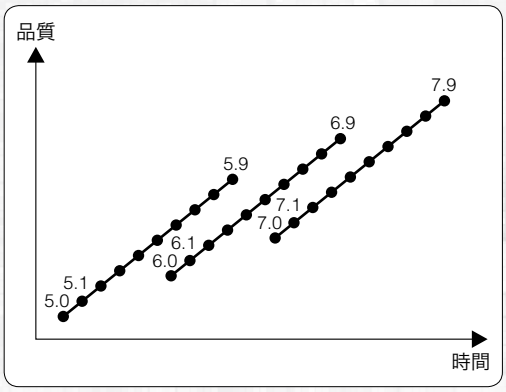
ルにも適用される、というわけです（図2）。

ブランチだけで開発しない理由

ここで「なぜそんな一見して回りくどいことをしているのか？」という疑問が湧いたかもしれません。つまり、RHELのあるバージョンで必要なパッチはそのブランチの中で留めておけば、複数回のレビューや修正案のやり取りの必要がないため、もっと短い期間でパッチを開発し適用できるのではないかということです。実際、かなり昔の話ではあるのですが、レッドハットの社内でブランチをメンテナンスしていたことがあり、一部の顧客に向けてカスタムカーネルなどが提供されていました。しかし、ブランチのメンテナンスには大きなコストがかかる[注4]ことと、もう1つ後述する理由によって前述のアップストリーム・ファーストに完全移行したという経緯があります。

RHELをブランチだけで開発しない大きな理由はツリーの連続性を維持するためです。RHEL 6において開発されたkernel-2.6.32向けのパッチを例とすれば、社内のバージョン管理システムの上だけで保持しアップストリームに還流しなかった場合、RHEL 7で採用されるkernel-3.10にはそのパッチが入っていない可能性があります。もしそうなった場合、RHEL 6では解決された問題がRHEL 7で再発し、デグレードしてしまうわけです。

▼図3 バージョンと品質の関係（模式図）



RHELの品質はメジャーバージョンで下がる？

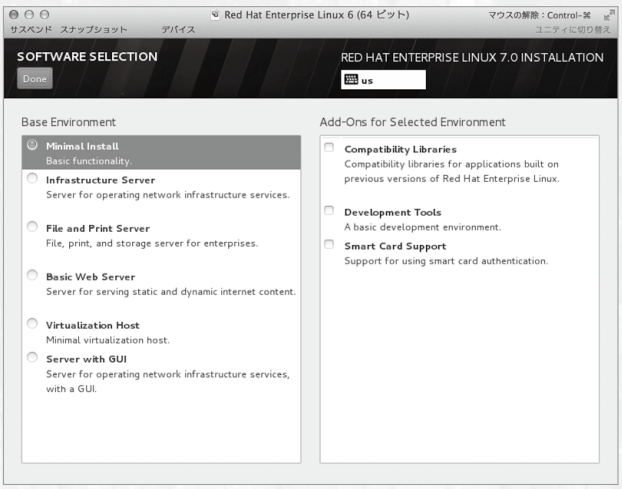
レッドハットではソフトウェアの品質を向上するために、ツールを使ったソースコードの品質チェックが行われています。残念ながらその結果を本稿で紹介することはできないのですが、各リリースにおける品質はおおむね図3のように模式化できます。

メジャーリリースの最初のマイナーリリース、つまり「.0」は同じタイミングの前メジャーリリースのマイナーリリースと比較すると、全体とし

注4）社内だけでブランチをメンテナンスするのと、アップストリームでメンテナンスするのでは、コストの制約があるためにコードを開発・レビューする人数が大きく異なることも理由として挙げられる。

ての品質はどうしても下がります。ですが、前述したエンジニアリングフローと付き合わせると、品質が低下する理由を説明することは簡単です。2つのメジャーリリース間で共通の機能があった場合、その機能を実装しているソフトウェアについてはツリーの連続性を保つことで品質が向上していることが期待できます。一方で、メジャーリリースには新機能が追加されるのが通例なので、その実装部分については長年メンテナンスされて「枯れて」いるソフトウェアと比較して品質が劣りがちです。従って全体としては一時的に品質が低下するものの、利用する機能やソフトウェアを限定することでその影響を最小化することが可能です。

▼図4　最小インストールをするだけでも脆弱性が下がる



この基本的な理解があれば、レッドハットのトレーニングコースやドキュメントで推奨している「必要なRPMパッケージだけをインストールする方法」が、バグやセキュリティの脆弱性といったトラブルを回避する最善の方法であることにも納得できるはずです。残念ながら問題を起こさない「完全なソフトウェア」というものは過去にも、そしておそらく将来にも実現するのが非常に難しく、問題を踏む確率を最小限に抑えることしかできないのが現実です。

安定稼働の観点から考えたRHELのインストール

脆弱性を回避する方法とは

RHELをインストールするには、メディアキットやその元となっているISOイメージファイルを用います。インストール後、Red Hat Networkに登録し、最新のRPMパッケージで更新することがセキュリティの脆弱性を回避する最善の方法ですが、RHEL上で稼働させるアプリケーションや稼働テスト済みの環境とそろえるために、たとえば「RHEL 6.3」という特定のバージョンをインストールせざるを得ないこともあるので、その方法についても簡単に紹介しましょう。

まず、RHELのインストールメディアからマシンを起動するとanacondaというインストーラが実行され、インストールの一連の作業をインタラクティブに進められます。この際、“Minimal Install”あるいは「最小インストール」というパッケージグループを選択します。**図4**はRHEL 7のインストーラ画面の例です。実はこの「最小インストール」を選択するだけでエラータを適用しなければならない確率を数分の1以下に下げられます。実際、筆者の手元にあるインターネットに接続されているRHEL 6.5のサーバ上では、ウェブサーバやDNSサーバ、PostgreSQLサーバなど各種サービスを稼働させていますが、インストールされているのは900に満たないRPMパッケージです。RHEL 6.5のメディアキットには約3,700程度のRPMパッケージが含まれていますので概ね4分の1しか利用していないことになり、当然のことながら適用しなければならないエラータも確率論としては4分の1になるわけです。

▼図5　yumコマンドを利用してインストール

```
# mount -o loop /tmp/rhel-server-6.5-x86_64-dvd.iso /mnt/
# mkdir /opt/rhel65_repo
# cp -prv /mnt/* /opt/rhel65_repo/

# cat /etc/yum.repos.d/rhel-local.repo
name=Red Hat Enterprise Linux
baseurl=file:///opt/rhel65_repo/
enabled=1
gpgcheck=1
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-redhat-release
```

確実なバージョンアップの方法

インストールの終了後、Red Hat Networkへの登録、あるいはサブスクリプションマネージャによるサブスクリプションの有効化を行えば、yumコマンドによって自動的にRPMパッケージ間の依存性が解決され、必要なRPMパッケージがすべてインストールされます。前述したように最新のRPMパッケージではなく、メディアキットに含まれるRPMパッケージだけでインストールをしたい場合には、メディアキットをマウントして含まれているファイルをすべてローカルHDDにコピーし、**/etc/yum.repos.d/**以下にレポジトリの定義ファイルを作成することで、yumコマンドで追加インストールが行えます。**図5**にその手順の例を記しておきますので、参考にしてください。

LINUXCON JAPAN 2014

ちょうど本稿の締め切りの翌日・5月20日から3日間、目白の椿山荘でLinuxCon Japan 2014が開催されました。レッドハットもスポンサーとして名を連ね、海外から多くのエンジニアが来日しLinuxカーネルやその周辺の関連事項について説明するとともに、ブースには多くの方にお越しいただきました(**写真1**)。

▼写真1　LINUXCON JAPAN 2014(レッドハット社ブース)

また併催されたGluster Community Dayでも同僚の中井氏や岩尾氏がプレゼンテーションを行い、分散ファイルシステム「Red Hat Storage」とOpenStackの統合や、分散オブジェクトストレージ「Ceph」の技術的な説明を行いました。

さらに本誌連載陣と会場で「進捗どうですか？」という挨拶[注5]も交わすことができるというなかなかに貴重なイベントですので、読者諸氏も来年はLinuxCon Japanにぜひ足を運んでみてください。

次回は

近々発表される予定のRHELの最新版であるバージョン7について、評価版の入手、インストール、基本設定や、導入前に確認すべきドキュメントなどについて紹介します。SD

注5) 実際には青田氏や浅田氏らと筆者の間で「締め切り、昨日でしたね」といった感じの言葉を交わした[注6]。

注6) 「進捗〜」は、当社書籍編集部の傳智之の口癖。SNSで進捗にまつわるエントリーを数時間おきに書き込み、ライターさんを恐怖のズンドコに陥れるので一部で問題視されている(w。

Be familiar with FreeBSD.

チャーリー・ルートからの手紙

第9回 ◇ 仮想ディスクのサイズ調整が便利になったgrowfs(8)

◉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。

パワーアップしたgrowfs(8)、マウントしたままファイルシステムを拡張!

ちょっと地味目な機能ですが、FreeBSD 10.0-RELEASEで登場した便利な機能に「**growfs**(8)のオンライン対応」といったものがあります。この機能が動作すると仮想環境で動作させているFreeBSDサーバの管理が便利になります。

FreeBSDは4.4-RELEASEの段階でgrowfs(8)と呼ばれる機能を導入しました。この機能はUFSファイルシステムのサイズを拡張するという機能です。従来のやり方ですと、次の手順を踏むことでファイルシステムの拡張が可能でした。

①ファイルシステムのアンマウント
②パーティションテーブルの書き換え
③growfs(8)によるファイルシステムの拡張
④ファイルシステムのマウント

10.0からはマウントした状態のままこの作業ができるようになりました。つまり作業を次の手順で完了させることができます。

①パーティションテーブルの書き換え
②growfs(8)によるファイルシステムの拡張

ファイルシステムをマウントしたまま作業ができますので、サービスを止めることなく利用するファイルシステムを広げることができます。運用状況に合わせて、使用する仮想ディスクのサイズを引き上げるといった使い方をする場合に便利です。

仮想化アプリケーション側の設定

実際に作業して、どういったものか感じてみましょう。まず、20GBの仮想ディスクにFreeBSD 10.0-RELEASEをインストールした環境を用意します。ここでは仮想化アプリケーションとしてVMware Fusion 6.0.3を使いました。**図1**のように、20GBのうち18GBほどをファイルシステムに、残りはスワップに扱われています（パーティションでは19GB確保していますが、ファイルシステムはそれよりも小さいサイズを表示します）。

VMware Fusion 6.0.3では動作させた状態のまま仮想ディスクのサイズを変更することはできませんので、いったん仮想環境を終了してからサイズの引

▼**図1 20GBの仮想ディスクにインストールされたFreeBSD 10.0-RELEASE-p2**

```
% uname -a
FreeBSD virt.ongs.co.jp 10.0-RELEASE-p2 FreeBSD 10.0-RELEASE-p2 #0: Tue Apr 29 17:06:01 UTC 2014
root@amd64-builder.daemonology.net:/usr/obj/usr/src/sys/GENERIC  amd64
% df
Filesystem      Size    Used   Avail Capacity  Mounted on
/dev/da0p2       18G    6.2G     11G    37%    /
devfs           1.0K    1.0K      0B   100%    /dev
%
```

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。

き上げを実行します(**図2、3**。注意:VMware Fusion 6.0.3の場合、サイズを引き上げることはできますが、サイズを引き下げることはできません)。

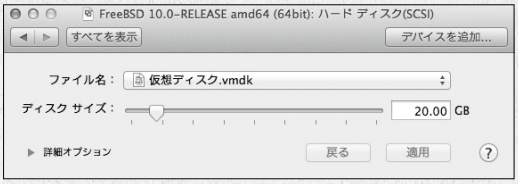
VMware Fusion 6.0.3では仮想ディスクのサイズを引き上げた場合、ゲストOS側でソフトウェアを実行して変更に対応するように、といったメッセージが出力されます(**図4**)。

ゲストOS側での見え方の違い

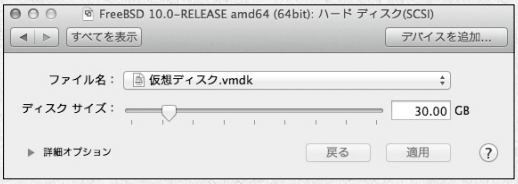
変更前と変更後で、ゲストOS側のFreeBSDからディスクの認識がどのように変わるか調べてみましょう。**dmesg**(8)の出力を比較すると、仮想ディスクのサイズが20GBから30GBへ増えていることを確認できます(**図5、6**)。

今度は**gpart**(8)を使ってパーティショニング情報を表示させてみます。**図7**のように、ディスクの最後に使われていない10GBのフリースペースが生えてきていることを確認できます。

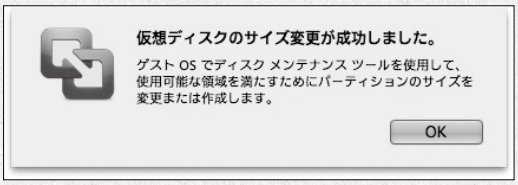
▼**図2　仮想化アプリケーションで設定している仮想ディスク(VMware Fusion 6.0.3)**



▼**図3　仮想化アプリケーション側では30GBの仮想ディスクへと扱いを変更**



▼**図4　設定を変更するとゲストOS側で対応が必要である旨が表示される**



▼**図5　仮想ディスクサイズ変更前**

```
% dmesg | grep da0
da0 at mpt0 bus 0 scbus2 target 0 lun 0
da0: <VMware, VMware Virtual S 1.0> Fixed Direct Access SCSI-2 device
da0: 320.000MB/s transfers (160.000MHz, offset 127, 16bit)
da0: Command Queueing enabled
da0: 20480MB (41943040 512 byte sectors: 255H 63S/T 2610C)
Trying to mount root from ufs:/dev/da0p2 [rw]...
%
```

▼**図6　仮想ディスクサイズ変更後**

```
% dmesg | grep da0
da0 at mpt0 bus 0 scbus2 target 0 lun 0
da0: <VMware, VMware Virtual S 1.0> Fixed Direct Access SCSI-2 device
da0: 320.000MB/s transfers (160.000MHz, offset 127, 16bit)
da0: Command Queueing enabled
da0: 30720MB (62914560 512 byte sectors: 255H 63S/T 3916C) ←ここの値が増えている
Trying to mount root from ufs:/dev/da0p2 [rw]...
%
```

▼**図7　10GBのフリースペースが追加されていることを確認**

```
% gpart show
=>       34  62914493  da0  GPT  (30G)
         34       128    1  freebsd-boot  (64K)
        162  39845760    2  freebsd-ufs  (19G)
   39845922   2097084    3  freebsd-swap  (1.0G)
   41943006  20971521       - free -  (10G)
%
```

チャーリー・ルートからの手紙

ファイルシステムを拡張してみよう!

この仮想環境はFreeBSD 10.0のデフォルトインストーラを使ってとくに設定を変更することなくインストールされたもので、仮想ディスクは次のように3つのパーティションに分割されています。

- 1: 64KB ブートローダ領域
- 2: 19GB ファイルシステム
- 3: 1GB スワップ領域

これが次のように変わったわけですから、

- 1: 64KB ブートローダ領域
- 2: 19GB ファイルシステム
- 3: 1GB スワップ領域
- 4: 10GB 未使用領域

ファイルシステムの領域を拡張するには、いったんスワップ領域を削除して作業する必要があります。次のような順序でアップデートを実施することになります。

- 1: 64KB ブートローダ領域
- 2: 19GB ファイルシステム
- 3: 1GB スワップ領域
- 10GB 未使用領域

⬇ (スワップ領域を削除)

- 1: 64KB ブートローダ領域
- 2: 19GB ファイルシステム
- 11GB 未使用領域

⬇ (ファイルシステムの領域を29GBへ拡張)

- 1: 64KB ブートローダ領域
- 2: 29GB ファイルシステム
- 1GB 未使用領域

⬇ (未使用領域をスワップ領域へ変更)

- 1: 64KB ブートローダ領域
- 2: 29GB ファイルシステム
- 3: 1GB スワップ領域

この手順どおりに作業してみましょう。まず、**swapoff**(8)コマンドを使ってスワップの使用を停止し、その後で**gpart**(8)コマンドでスワップパーティションを削除します(**図8**)。そうすると未使用の領域が11GBに増えることを確認できます。スワップパーティションを削除したので、インデックスから3が消え、1と2だけが残っていることも確認できます。1番がブートローダの領域、2番がファイルシステムです。

次に、**gpart**(8)コマンドを使ってファイルシステムのパーティショニング情報を29GBへ広げます(**図9**)。この状態ではパーティショニングテーブルの情報が書き換わっただけで、ファイルシステムは19GBのままです。

続いて、**growfs**(8)コマンドを実行して実際にファイルシステムの領域を29GBへ広げます。**図10**のように確認を求められるので、Yesと入力して作業を進めます。このコマンドを実行してはじめてファイルシステムが実際に広がります。

メインの領域を広げることができましたので、最後に未使用領域をスワップ領域としてパーティションを作成して、**swapon**(8)コマンドでスワップとしての機能を有効にします(**図11**)。最終的に**図12**のようにファイルシステムを広げたことが確認できます。

ZFSだけで対応できるのは?

こうしたファイルシステムの拡張といった作業はZFSのほうが便利です。しかし、ZFSは動作に大量

▼**図8　スワップの利用停止とパーティションテーブルからのスワップ領域の削除**

```
# swapoff /dev/da0p3
# gpart delete -i 3 da0
da0p3 deleted
# gpart show
=>       34  62914493  da0  GPT  (30G)
         34       128    1  freebsd-boot  (64K)
        162  39845760    2  freebsd-ufs  (19G)
   39845922  23068605       - free -  (11G)

#
```

▼**図9　メインのパーティションを29GBへ拡張**

```
# gpart resize -i 2 -s 29g da0
da0p2 resized
# gpart show
=>       34  62914493  da0  GPT  (30G)
         34       128    1  freebsd-boot  (64K)
        162  60817408    2  freebsd-ufs  (29G)
   60817570   2096957       - free -  (1.0G)

#
```

のメモリが必要で、仮想環境のようにリソースが限定された環境での利用には向いていません。また、UFSと比較するとスピードという面ではどうしても劣ってしまいます。リソースが限定された仮想環境ではUFS、リソースが豊富なオンプレミスではZFSというのが、ひとつのロジカルな選択方法です。

ちなみに、マウントした状態のままgrowfs(8)を実行する機能は、UFSに新しく追加された「書き込みサスペンド」という機能を使って実現されています。オンライン(マウントしたまま)の状態でファイルシステムを拡張したいという要望はこれまでも出ていたもので、10.0で実現されました。

ノンストップサーバ構築に活用

ここではデフォルトインストールの状態でサンプルを示しましたので、スワップ領域の削除と追加という手間が発生していますけれども、最初から仮想環境で運用することを前提にしてパーティショニングを実施しておけば、スワップ領域に触れることなくファイルシステムを動的に拡張するといったことができます。

FreeBSD 10.0にはいろんな新機能が追加されましたが、どの機能にも言えることは「手軽になった」ということだと思います。これまでの作業に比べるといろんな操作が簡単になりました。システムの構築も運用も、これまでと発想を変えることで本当に簡単に実現できるようになりました。活用方法は、ユーザのアイディア次第です :) SD

▼図10　growfs(8)でファイルシステムを29GBへ拡張

```
# growfs -s 29g /dev/da0p2
Device is mounted read-write; resizing will result in temporary write suspension for /.
It's strongly recommended to make a backup before growing the file system.
OK to grow filesystem on /dev/da0p2, mounted on /, from 19GB to 29GB? [Yes/No] Yes   ← Yesと入力
super-block backups (for fsck -b #) at:
 41031872, 42314112, 43596352, 44878592, 46160832, 47443072, 48725312, 50007552,
 51289792, 52572032, 53854272, 55136512, 56418752, 57700992, 58983232, 60265472
# df -h
Filesystem      Size    Used   Avail Capacity  Mounted on
/dev/da0p2       28G    6.2G     20G    24%    /
devfs           1.0K    1.0K      0B   100%    /dev
#
```

▼図11　スワップ領域の作成と有効化

```
# gpart add -t freebsd-swap -s 999m da0
da0p3 added
# gpart show
=>       34  62914493  da0  GPT  (30G)
         34       128    1  freebsd-boot  (64K)
        162  60817408    2  freebsd-ufs  (29G)
   60817570   2045952    3  freebsd-swap  (999M)
   62863522     51005       - free -  (25M)

# swapon /dev/da0p3
# swapinfo -h
Device          1K-blocks     Used    Avail Capacity
/dev/da0p3        1022976       0B     999M     0%
#
```

▼図12 最終的な状態

```
% df
Filesystem      Size    Used   Avail Capacity  Mounted on
/dev/da0p2       28G    6.2G     20G    24%    /
devfs           1.0K    1.0K      0B   100%    /dev
%
```

16

Debian Developer　やまねひでき　henrich@debian.org

効率よくリポジトリミラーを構築する方法

Debian Hot Topics

Ruby2.1移行とJessieリリーススケジュール

「Transition note: Ruby 2.0 is going away」というタイトルのメールがDebian関係者に流れてきました。「Debianのruby-defaultパッケージをRuby2.1に移行するので、対処が必要なパッケージのメンテナは対処せよ」という内容です。本誌6月号で、Ruby2.0に移行する話を伝えたあとに矢継ぎ早の話ですが、Ruby2.0と2.1ではさほど大きな非互換性は発生しないと思われるので、対象者が多いわりには移行はスムースに進むのではないでしょうか。

スケジュール的にRuby2.2のリリースは、おそらく2014年のクリスマスです。Debian 8「Jessie」の開発がフリーズする11月5日よりあとですから、JessieのRubyはRuby2.1でリリースされるでしょう。なお、リリースまでのスケジュールは次のようになっています。

- 2014年9月5日：バージョン間での非互換性修正のための移行（transition）停止
- 2014年9月5日：リリース対象アーキテクチャの最終確認
- 2014年10月5日：パッケージの「urgency」を「low」に固定、unstableからtestingへの移行は10日間の猶予が必要となる（それまでは「medium」で5日が標準）
- 2014年10月5日：セキュリティチームがサポート外のパッケージを決定
- 2014年11月5日：フリーズ、リリースクリティカルバグ修正の追い込み開始

「フリーズ」（新たなパッケージの受け入れ停止）の日程は決まっていますが、「リリース」の日程は決まっていないのにはご注意ください[注1]。

「sparc」除外と各アーテキテクチャの状況

また、次期リリース「Jessie」からsparcが取り除かれたことが告げられました[注2]。

この件について誤解されているコメントが散見されました。Jessieのリリースターゲットから外れたとはいえ、まだsparcアーキテクチャ上でのDebianパッケージのビルドは続けられており、完全にDebianでの開発の対象外になったわけではありません。本稿執筆時では「さらにUnstableでの開発を継続するかどうか」について議論されています[注3]。

なぜsparcが次期バージョン「Jessie」から外れたのかというと、現状のDebianでのsparcアーキテクチャの問題として、①安定性と②ハードウェアが挙げられます。

①は、Linuxカーネルの安定性がない（特定のバージョンであれば起動する）ことと、gcc-4.8関連パッケージがビルドできていないこと

注1）リリースクリティカルバグの修正スピードしだいです。

注2）URL https://lists.debian.org/debian-devel-announce/2014/04/msg00012.html

注3）debian.orgでのサポートアーキテクチャから外れた場合、debian-ports.orgへ移管されることになります。

により多くの依存パッケージがインストールできないという問題があります。ただ、作業者の増加や新バージョンでの修正により対応の目処が立つのではないかと思われます。

②は、そもそもベンダがOracle社一択(一応、富士通もありますが)で、彼らはSPARCでLinuxを動作させることには興味がないでしょうから、支援は期待できません。「eBayなら安く買えるよ!」という意見もありますが、財政的な基盤が強くないDebianで、将来性が不透明なアーキテクチャのビルドマシンに大枚をはたくかどうかも議論の的となるでしょう。

そのほか、リリースチームから各種アーキテクチャについて状況の説明が出ています[注4]。かいつまんで内容を紹介します。

- **「sparc」はJessieから除外**
- **「kfreebsd-amd64/i386」はJessieに入るが対象を縮小する**
  GNOMEデスクトップのサポートは要求しないが、GNOME関連ライブラリのサポートは必須(例:gtk+3.0)
- **「armel」をサポート対象に戻す**
  新たなハードウェアを寄付してもらい、とくに問題は見られない
- **「armhf」は検討中**
  新たなハードウェアを入手したがまだ不安定。移植作業者が問題を調査中で、近いうちに再度状況を評価する予定
- **「mipsel」はほぼ正常に戻った**
  新たなハードウェアを受け取り、ほぼすべてが正常動作中。残り2台がセットアップできたら正常に戻ったとみなす
- **「mips」は検討中のまま**
  新たなハードウェアの購入計画はあるが未実施。7月中旬予定
- **「hurd-i386」はJessieについて未検討**
- **「amd64」「i386」「powerpc」「s390x」は正常**

筆者にとって意外だったのはarmelの開発を継続することです。Jessieの対象アーキテクチャを決定する9月5日まではまだ間がありますので、引き続き動向を注視していきたいと思います。

BeagleBone Blackのデフォルトイメージ

みなさんは「Raspberry Pi」(以下、RasPi)をご存じでしょうが、同じカテゴリに入る製品として「BeagleBone Black」(以下、BBB)があります。BBBはRasPiと同等の価格ながら若干CPUが速いという特徴があるのですが、RasPiの影に隠れていて初耳……という方もいるかもしれません。

先日、このBBBで利用されるデフォルトのディストリビューションについて「ÅngströmからDebianに移行する」と関係者がインタビューで答えていたのを目にしました。実際、配布イメージにそれまでオプション扱いだったDebianが登場しています[注5]。お手軽に使えるARMのLinuxマシンは「デフォルトがDebianベース」という状況が加速していますね。イベントのDebianブースでもBBBを展示する予定ですので、興味のある方はお立ち寄りください。

リポジトリミラー構築のすすめ

さて、話題を変えてちょっとしたTipsを。同一構成マシンのデプロイなどを大量にする方、あるいは実験で同じ構成のサーバを何回も作りなおすような方は「パッケージの取得に毎回時間がかかるなー」という感想を持たれることでしょう。そんな大量のトラフィックが発生する作業を行う場合は「ローカルネットワーク上にリポジトリミラーを構築する」ことを考えてみましょう。

牛刀をもって鶏を割く? ローカルミラー構築

ミラーを作るためのパッケージを探した人が、まず見つけるであろうパッケージとして、その

注4) URL https://lists.debian.org/debian-devel-announce/2014/05/msg00000.html

注5) URL http://beagleboard.org/latest-images

名もズバリ「apt-mirror」[注6]があります。**図1**、**リスト1**のようにして利用します。

ただし、apt-mirrorの中身は単にwgetでファイルを並列に取得するスクリプトなので、取得先のサーバからは迷惑な大量ダウンロードと区別がつきづらく、非常に筋が悪い方法です。不要なパッケージの大量取得はほかのユーザにも迷惑となり、場合によっては攻撃とみなされてサーバから接続拒否(ban)されることもありえますので、ご注意ください(最低でもデフォルトの20同時接続を変更することを検討してください)。

ちなみに、「公式ミラーサーバを引き受けたいので、全アーキテクチャを含めた完全なミラーを構築／運用したい！」という奇特な人(筆者など)に対しては「ftpsync」というスクリプト[注7]が用意されています。

注6) URL http://apt-mirror.github.io/

注7) URL http://www.debian.org/mirror/ftpmirror　公式のミラーはこちらのスクリプトを使ってrsyncしています。経験的にはtarballではなくGitで公開されているスクリプトをcloneして使うほうがハマりが少ないです。

より効率的に、プロキシミラーの利用

先に挙げたapt-mirror/ftpsyncスクリプトは「フルミラー」の構築に使われるものです。同じ構成のマシンをデプロイする場合はサーバ関連の一部のパッケージを使うだけであり、デスクトップ環境やTeX関連などの巨大なパッケージ群は時間をかけてミラーしてもサーバの構築では利用されることはありません。フルミラーはデータを取得する時間が何時間もかかるうえに、数十～数百GBというディスク容量も必要ですし、何より上流のミラー元サーバに大きな負荷をかけます。できれば、使うパッケージだけで効率よくミラーを構築できたらすばらしいですよね。

そんな期待に応えるのがsquid-deb-proxyなどのプロキシミラー[注8]の利用です。squid-deb-proxyはプロキシとして有名なsquid3を利用してdebパッケージのキャッシュに特化したものです。クライアント側ではsquid-deb-proxy-clientパッケージを入れるだけで、Avahiを利用して設定不要でプロキシサーバを自動で検出できます[注9](**図2**)。たいていの場合はインストールする

注8) ほかにもapprox、apt-cacher、apt-cacher-ngなどがあります。過去にはapt-proxyというのもありましたが、すでに削除されています。URL https://bugs.debian.org/576821参照。

注9) OS X ServerでもAppStoreのソフトウェアのダウンロードをキャッシュする同様の機能があります。設定の詳細は/etc/apt/apt.conf.d/30autoproxyを参照。

▼図1　apt-mirrorを導入する

```
$ sudo apt-get install apt-mirror
$ sudo vi /etc/apt/mirror.list
  （ファイルを編集、リスト1参照）
$ sudo apt-mirror
 ↓ミラー完了後、適当なhttpサーバを入れて他マシンから
  参照可能にする
$ sudo apt-get install apache2
 ↓ここではrootディレクトリでdebianディレクトリが見えるように設定
$ sudo ln -s /var/spool/apt-mirror/mirror/ftp.jp.debian.org/debian /var/www/debian
 あとはクライアント側のapt lineをローカルミラーを参照するように書き換える
```

▼リスト1　mirror.listの編集内容

```
（編集前）
set nthreads     20
  （略）
deb http://ftp.us.debian.org/debian unstable main contrib non-free
deb-src http://ftp.us.debian.org/debian unstable main contrib non-free

（編集後）
 ↓並列で動くwgetの数を抑える
set nthreads     3
  （略）
 ↓取得先を日本のサーバに変更、ソースパッケージは取得しない
deb http://ftp.jp.debian.org/debian unstable main contrib non-free
```

だけで設定も不要というお気楽さですが(wheezyの場合はbackportsから導入)、大きな効果が出ます。筆者の検証用VirtualBox環境に導入してみたところ、キャッシュをSSDに置いたせいもあってかダウンロード速度が26.0MB/sも出ました。その威力がわかるかと思います。

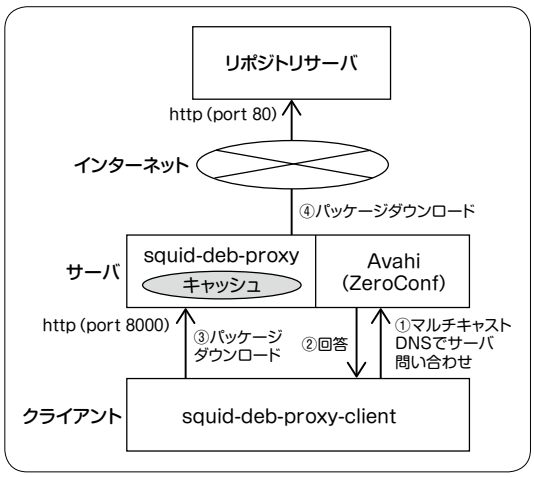
図3、リスト2の作業を行ったあと、サーバ側で**sudo tail -f /var/log/squid-deb-proxy/access.log**などとしてクライアントからのアクセスを確認しつつ、クライアント側でパッケージのインストールを実施してみましょう。動作していればパッケージを取得しているログが流れてくるはずです。

squid-deb-proxyには弱点が1つあり、リポジトリサーバがリダイレクトアクセスを提供している場合はリダイレクト先も/etc/squid-deb-proxy/mirror-dstdomain.aclに追加してやらないとアクセスエラーが出ますので注意してください(有名どころだとdeb-multimedia.orgなど)。

ほかのソフトウェア、たとえばapt-cacher-ngについては、gihyo.jpの連載「Ubuntu Weekly Recipe」の「apt-cacher-ngを使ってAPT用キャッシュプロキシの構築」[注10]で取り上げられていますので、そちらを参照してください。

プロキシミラーを使うと、ミラー対象のパッケージを最新に保つのが容易になるというメリットがあります(apt-mirrorですといちいち全部のパッケージを舐めていくという、非常に効率の悪い手法しか取れません)。複数台のDebianマシン運用時に時間も帯域もディスク容量も節約できるsquid-deb-proxyやapt-cacher-ngなどのプロキシミラー活用の検討をしてみていただければ幸いです。SD

▼図2　squid-deb-proxyを使ったプロキシミラー



▼図3　squid-deb-proxyの導入

```
(サーバ側)
$ sudo apt-get install squid-deb-proxy
↓参照先のサーバがdebian.org以外の場合は、
  以下を編集して許可を追加
$ sudo vi /etc/squid-deb-proxy/mirror-dstdomain.acl
(ファイルを編集、リスト2参照)

(クライアント側)
$ sudo apt-get install squid-deb-proxy-client
```

▼リスト2　mirror-dstdomain.aclの編集内容

```
# mirror-dstdomain.conf
#
# network destinations that are allowed by this cache

# the default mirror names on debian are ftp[0-9]* .[a-z]+ .debian .org
# but that would require (slow) regexp matching, so for now we allow
# www.debian.org and friends here too
.debian.org
cdn.debian.net
http.debian.net
debian-mirror.sakura.ne.jp   ←.debian.orgドメイン以外なので追加
dl.google.com   ←3rdパーティのリポジトリなので明示的に追加が必要
(以下略)
```

注10) URL http://gihyo.jp/admin/serial/01/ubuntu-recipe/0315

Enjoy Open Source!!

橋本 賢弥
HASHIMOTO Takahiro
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グループ
ソリューションアーキテクト

恵比寿からこんにちは

はじめまして。レッドハットでプラットフォームのソリューションアーキテクトをやっている橋本と申します。2014年の2月から「あー、某恵比寿のオープンソースの会社」の一員となりました。Linuxはもちろん、仮想化、OpenStack、クラウド、ストレージといった、大きく広がりつつあるレッドハットのプラットフォーム系ポートフォリオをもって、オープンソースとレッドハットが提供できる価値を顧客にお届けすべく日夜(?)励んでおります。あ、堅苦しいですね(笑)。本誌を初めて手に取ったのはもう十年くらい前ですが、まさか自分が原稿を書くことになろうとは、と不思議な思いです。どうぞよろしくお願いします。

オープンソース天国は本当だった

筆者はメーカー系のベンダからレッドハットへ転職しました。以前は個人でオープンソースソフトウェア(以降OSS)にかかわることは多かったのですが、仕事ではプロプライエタリな製品を扱うことも多く、仕事のやり方やしくみも独自のものでした。まぁ、当然ですね。一方レッドハットでは、社内の活動はまるでOSSのプロジェクトに参加しているようです。社内のメーリングリストに登録して海外のエンジニアと議論をしたり、パッチが書かれたメールを眺めたり、Bugzillaに報告していろいろな状況を判断したり、(もちろんすべて公開されている)製品のソースコードを読んで動作を確認したり、技術情報を世間にアウトプットしたりと、あれ、これなんかみんな仕事以外でもやってるかも?

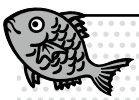
なーんて、妙にしっくりくるときが多くあります。そんなそばから「今度この業務システムを変更するよー、もちろんオープンソースだよー」というのがひょいっとメールで流れてくるなど、まさにオープンソース天国を謳歌しています。最近ではOSSを幅広く利用する会社も多くなってきていると思いますが、それでもこのレッドハットのオープンソースに根ざす文化は独特ではないかなと思います。あ、もちろん、ビジネスをしていますから、そういうところは他の会社と同じだと思います(笑)。

誰もがソースコードを読めるなら

OSSで、誰でもソースコードにアクセスできる世界では、メインの開発者だけでなく、誰もがバグを見つけたり、新しい機能を提案したり、そのためのパッチを書いて提出する機会が提供されていますね。この「誰もが」というところがとても大事だなぁと思います。ここでちょっと筆者の体験を紹介します。もう8年くらい前の話です。とあるUNIXサーバをお客様に納めるお仕事があり、筆者はチームリーダーをしていました。ある日メンバーから「○○○(ストレージ用ドライバに付属の、プロプライエタリなバックアップユーティリティ)が動作しません!」との連絡をもらいました。サポートに問い合わせて状況を説明しましたがなかなか理解してもらえず「これは仕様だ。そんな使い方は想定しな

い」との回答が。時間だけが過ぎていき、プロジェクトのスケジュールに影響を及ぼしかねない状況でした。筆者も個別に調査を始めてみると、このユーティリティで呼ばれているシェルスクリプトに原因があることがわかりました。そう、幸運なことにスクリプトなので、中身が読めます。バックアップとリストアのスクリプトの両方を眺めていくと、バックアップ側で実装されている動作をリストアの処理では考慮していない作りになっていることがわかりました。そこでリストアが正常に動作するように変更してパッチを作成し、「バックアップでは今回の動作を想定して実装されているが、リストアではこれを考慮していない。このパッチのように変更すればちゃんと動くから、これをマージしてほしい」という英語の説明とともにパッチを開発チームに送るよう依頼をしました。すると今度はすぐにアメリカの開発チームから「わかった。君の言うとおり、これは仕様ではなくバグだ。次のバージョンで必ず修正する」というコメントが来ました。さらに開発チームは修正の正式版が出るまで、筆者のパッチをお客様環境で使用することをサポートすると明言してくれました。そしてお客様にご迷惑をかけることなく、無事サービスを開始できました。めでたし。めでたし。

OSSにどっぷりの皆さんは「ん？　当たり前じゃない？　なんか特別なこと？」とおっしゃるかと思います。でも、もしこれがスクリプトではなくバイナリだけで提供されていたら、問題を解決するにはより多くの時間を必要とし、お客様にご迷惑をかけていたかもしれません。もちろんなんでも自分で解決すればいいというわけではありませんが、実際にソフトウェアを使用していて、早くバグ直したい、こう変更すべきだ、と思った人が誰でも、コードを読み、パッチを書いてきちんと説明することで、その提案を取り込んでソフトウェアを改善することができる「チャンス」を持っているということはとてもすばらしいことだと、皮肉にもOSSではないソフトウェアで実感したのでした(笑)。OSSなら、提供されるすべてのソフトウェアでこれが実現できる可能性があります。それは、当時の筆者にOSSの未来を確信させるのに十分な出来事でした。そして今は、それに賛同してくれる人がたくさんいる時代になったのだと思います。



SDxはストレージに注目を

最後に少しストレージの話をします。今、“Software Defined”なインフラが注目されています。“Software Defined Storage”という考え方がストレージ業界でも浸透し始めていますが、最近ストレージのトレンドは大きく二極化していると感じます。一方はメディアにFlashやSSDを使用して、コントローラにFPGAで専用の処理を実装し、高負荷な特定のワークロードを一極集中で担う専用の超高性能ストレージ、もう一方はコモディティ化したハードウェアと柔軟な機能を備えたOSSのソフトウェアを使用した大容量で統合されたスケーラブルなストレージです。前者は高I/O性能かつ容量集約率重視で、低I/O遅延と圧縮や重複排除機能がカギになってきています。後者はコモディティなハードウェアで実現できることや、スケーラビリティの高さ、そしてクラウドとの連携やデータ連携のAPIといった機能が重要視されています。今までストレージは「聖域」かのように触れることを避ける人が多く、こうしたことをいろいろ深く学ぶエンジニアはサーバやネットワークと比較すると少ないと感じていましたが、クラウド時代を迎え、Software Definedなストレージの必要性はますます高まっていてストレージのことをいろいろ学ぶいいチャンスです。そこでOSSで学習コストが低く、優れたSoftware Defined Storageのコア機能であるGlusterFSやCephに触れていただいて、ストレージ「も」得意なエンジニアになれば、いろいろなところで引っ張りだこになるかもしれませんよ？　そしてぜひ一緒に恵比寿で働きましょう(笑)。SD

Ubuntu 14.04 LTS 日本語Remixを作ってみる

Ubuntu Japanese Team
あわしろいくや　ikuya@fruitsbasket.info

Ubuntu 14.04 LTS 日本語Remixはubuntu-defaults-builderでインストールisoイメージの細かな修正を行い、これまでよりも一歩踏み込んだカスタマイズを行いましたので、ここでその方法をまとめます。自分なりのカスタマイズを行いたい場合のアドバイスもあります。

14.04の日本語Remix

Ubuntu 14.04 LTSの日本語Remixは、それまでのバージョンと比較して変更点が大きかったです。日本語Remixはその前身（名称変更前）も含めてずっと32bit（i386）版をリリースしてきましたが、1つ前の13.10で64bit（AMD64）版と両方リリースされ、ついに14.04ではAMD64版が主でi386版はいちおうリリースはされたものの推奨はされていません。今はAMD64に対応していないPCのほうが少ないので、こうなるのも必然といえます。

14.04での最大の変更点は、インプットメソッド（IM）をIBusからFcitxに変更したことです。過去に何度か書いてきましたが[注1]、IBusだと仕様変更によって今までと同じようには使えない部分もあるので、とくにユーザが多いであろうLTS（Long Term Support）の14.04では、比較的今までと同じように使用できるFcitxにすることにしました。

インストール中にスライドショーが表示されますが、実は差し替えることができます、というわけで日本語Remixユーザに便利な情報を提供するように差し替えてみましたが、今後メンテナンスすることを考えると変更は最小限度に抑えたほうがいいに越したことはありません。

今回は具体的にどのような変更を行ったのかを具体的に解説します[注2]。独自Ubuntuを作成してみたい場合の参考になれば幸いです。

注1） 先月号の第2特集でもさらっと書きました。

準備と最初のisoイメージの作成

まずは普通のUbuntuを作成してみましょう。次のコマンドを実行してください。

```
$ sudo apt-get install ubuntu-defaults-builder
```

次に、適当なフォルダにテンプレートをコピーします。

```
$ mkdir ubuntu-defaults-builder
$ cd ubuntu-defaults-bulider
$ ubuntu-defaults-template ubuntu-defaults-sd
```

ubuntu-defautlts-templateがテンプレートをコピーするコマンドで、ubuntu-defaults-sdがパッケージのソース名です。もちろんここは任意に変更していただいてかまいません。続いてパッケージのソースをパッケージ化し、isoイメージを作成します。

```
$ cd ubuntu-defaults-sd
$ dpkg-buildpackage -r -uc -b
$ cd ../
$ ubuntu-defaults-image --package ./ubuntu-
defaults-sd_0.1_all.deb
```

dpkg-buildpackageがパッケージを作成するコマンドで、ubuntu-defaults-imageがisoイメージを作成するコマンドです。実行の際、先だって作成したパッ

注2） 過去にも書いたことがありますが、今回はより踏み込んだアップデート版です。

ケージを引数で与えています。原則としてはこれだけで、すごく簡単であることがわかります。

ubuntu-defaults-imageのオプションとしては、ここでは-archと-releaseを覚えておくといいでしょう。前者はi386かamd64を指定できます。デフォルトでは現在実行しているアーキテクチャのisoイメージが作成されます。後者はUbuntuのコードネームを与えます。デフォルトではイメージを作成しているコードネームを与えています。具体的にはtrustyです。

isoイメージの作成には時間がかかるので、少しお待ちください。isoイメージはファイル名違いで2つ作成されますが、両者はまったく同じものです。それぞれにmd5sumコマンドを実行してみれば、同じものであることがわかります。

できあがったisoイメージは、仮想マシンで実行できるか確認してみるといいでしょう。

基本的な変更

すでにお気づきかもしれませんが、「パッケージのソースを変更してパッケージを作成し、isoイメージを作成する」を繰り返すことになります。ここでは日本語環境に必要な変更を加えることにします。

次のようにディレクトリを変更してパッケージのソースに戻り、ファイルを編集します。

```
$ cd ubuntu-defaults-sd
```

まずはdepends.txtですが、**リスト1**のように編集します。コメントアウトの部分は省略します。続いてhooks/chrootを**リスト2**に、i18n/keyboard.txtを**リスト3**に、i18n/langpacks.txtを**リスト4**に、language.txtを**リスト5**に提示しますので、このとおりに編集してください。

depends.txtは、その名のとおりisoイメージに含めるパッケージを列挙しています。本当はrecommends.txtと使い分けたほうがいいのですが、今回はしていません。

hooks/chrootはシェルスクリプトで、コメントにもあるとおりisoイメージ作成中のパッケージのインストールが完了したあとに実行されます。このスクリプトが終了した後、squashfsが作成されます。とい

リスト1　depends.txtの内容

```
language-pack-ja
language-pack-gnome-ja
libreoffice-l10n-ja
libreoffice-help-ja
firefox-locale-ja
thunderbird-locale-ja
fonts-takao-pgothic
fonts-takao-gothic
fonts-takao-mincho
poppler-data
cmap-adobe-japan1
ibus-anthy
ibus-mozc
kasumi
im-setup-helper
fcitx
fcitx-mozc
fcitx-libs-qt5
fcitx-frontend-qt5
mozc-utils-gui
```

リスト3　i18n/keyboard.txtの内容

```
jp
```

リスト4　i18n/langpacks.txt の内容

```
ja complete
```

リスト5　language.txtの内容

```
ja
```

リスト2　hooks/chrootの内容

```
wget -q https://www.ubuntulinux.jp/ubuntu-ja-archive-keyring.gpg -O- | sudo apt-key add -
wget -q https://www.ubuntulinux.jp/ubuntu-jp-ppa-keyring.gpg -O- | sudo apt-key add -
wget -q https://www.ubuntulinux.jp/sources.list.d/trusty.list -O /etc/apt/sources.list.d/ubuntu-ja.list
if [ -f /etc/apt/sources.list.d/zz-sources.list ]
then
  sed -i -e 's/http: / /archive .ubuntu .com/http: / /jp .archive .ubuntu .com/' /etc/apt/sources.list.d/z
z-sources.list
fi
apt-get update
apt-get upgrade --yes
apt-get clean
```

うことはシェルスクリプトでできることであれば何でもできるということになりますが、ここでパッケージをインストールしてもライブセッションでは有効なものの、インストーラ(Ubiquity)でインストールするとそのパッケージは削除されてしまいます。ここがちょっとしたコツというかハマりどころですので、お気を付けください。編集作業が終了したら、パッケージとisoイメージを作りなおします(**図1**)。

オプションが増えました。--ppaは、その名のとおりPPAを指定します。im-setup-helperはUbuntuのリポジトリにはありませんが、PPAにはあるのでここを指します。すなわち、PPAと組み合わせれば任意のパッケージを追加できます。もちろんそれだけではなく、ちょっとした不具合を修正したパッケージをPPAにアップロードしておけば、そのバージョンをダウンロードしてきます。具体的には、今回使用したPPAではMozcとunzipのパッケージをアップデートしています。

--componentsも見たままですが、universeにあるパッケージをダウンロードする場合に指定します。multiverseも指定していますが、今回はmultiverseにあるフリーではないパッケージは含まれていません。再配布を考えている場合はmultiveseにあるパッケージは含めないほうがいいでしょう。--localeもそのままです。

isoイメージの作成が途中で止まる場合は、今回変更したファイルに間違いがないか確認してみてください。ちょっとでも間違いがあるとすぐに停止します。

キーボードインジケーターを非表示にする

14.04からFcitxとIBusが両方インストールされている場合、前者のほうが優先して起動するようになりました。しかしFcitxが起動してもキーボードインジケーターは表示されており、これはIBus専用ですので使いどころがありません。非表示にするためには[システム設定]-[テキスト入力]の[メニューバーに現在の入力ソースを表示]のチェックを外せばいいのですが、それをすべてのユーザにやってもらうというのは非現実的です。というわけでパッケージの中で行ってログイン時にはオフになっているのが理想であり、今回挑戦することにしました。この設定はdconfというGNOMEの設定フレームワークで行われており、Unityでも全面的に採用されています。というわけで調べていくと、**図2**のコマンドで非表示にできます。あとはこれを自動実行すればいいだけです。自動実行のしくみはubuntu-defaults-builderにはなく、Debianパッケージの機能を使います。ですので、debianフォルダに20_ubuntu-defaults-sd.gschema.overrideというファイルを作成し、内容を、

```
[com.canonical.indicator.keyboard]
visible=false
```

とします。これだけだとパッケージに入らないため、同じフォルダにinstallというファイルを作成し、内容を**リスト6**のようにします。すなわち、debianフォ

図1　パッケージとisoイメージの作成

```
$ dpkg-buildpackage -r -uc -b

$ ubuntu-defaults-image --package ./ubuntu-defaults-sd_0.1_all.deb --ppa japaneseteam/ppa --component
s main,restricted,universe,multiverse --locale ja_JP
```

図2　IBusのインジケータを非表示にする

```
$ gsettings get com.canonical.indicator.keyboard visible false
```

リスト6　installファイルの中身

```
debian/20_ubuntu-defaults-sd.gschema.override usr/share/glib-2.0/schemas
```

図3　debianフォルダー以下にあるファイル

```
20_ubuntu-defaults-sd.gschema.override  changelog  compat  control  copyright  install  rules
```

ルダの中身は**図3**になります。

あとはパッケージとisoイメージを再作成し、起動してキーボードインジケーターが消えていることを確認してみてください。

スライドショーを変更する

スライドショーを変更するのはなかなかたいへんです。まずはテンプレートを取得する必要があります。そして、スライドショーが入ったソースを取得してビルドする必要があります。isoイメージ作成直後のフォルダにいると想定して**図4**のコマンドを実行してください。

これでubuntu-defaults-sd/ubiquity-slideshow/slides/ja_JP/以下に、8つのHTMLファイルがコピーされます。というわけで、あのスライドショーの正体はHTMLでした。画像の追加もできますが、今回は行っていません。HTMLファイルの追加は試していませんが、おそらくできないものと思われます。文言の変更やリンクの追加はHTMLを編集すればいいということになり、ここまで来てしまえばお手軽にできるのではないでしょうか。

UEFI非対応

これでいちおう完了しましたが、ubuntu-defaults-builderで作成したisoイメージはUEFI(Unified Extensible Firmware Interface)非対応です。これをUEFIに対応させるためには手動での作業が必要です。その方法はここには書きませんが、Ubuntu Japanse Teamリーダーの小林準氏がgihyo.jpに書かれたUbuntu Weekly Recipe 第300回[注3]の記事を、14.04に適合するように変更してみてください[注4]。

ほかにもカスタマイズできそうなところ

今回は行わなかったもの、ほかにもカスタマイズできそうなところはたくさんあります。というわけで、勘所をいくつか紹介します。ここでもフォルダー名／ファイル名で編集すべきファイルを提示します。

desktop/default-session.txt

デフォルトのセッションは当然Unityですが、変更できます。たとえばGNOME 2.xのルック&フィールに近いGNOME Flashbackにしたい場合は、depends.txtに「gnome-session-flashback」を追加し、desktop/default-session.txtに「gnome-fallback」を追記すればいいのですが、コメントにもあるように、この場合は今回の例で使用したubuntu-defaults-sdなどパッケージ名に「ubuntu」を入れることができなくなります。

unity/launchers.txt

Unityの左側に表示されているランチャーにアイコンを追加できます。原則は/usr/share/applications/*.desktopであれば何でも追加できます[注5]。よく使うアプリケーションを追加しておくと良いでしょう。

webbrowser/startpage.txt

Webブラウザ(今のところはFirefoxのみ)のいわゆるホームページを設定できます。ほかにもブックマーク(メニューとツールバー)と検索エンジンも変更できますが、後者は非推奨のようです。SD

注3) http://gihyo.jp/admin/serial/01/ubuntu-recipe/0300?page=2

注4) 広く公開される日本語RemixであればUEFIに対応していないのはナンセンスで、このような作業が必要になりますが、手元で作成する場合はWindowsとデュアルブートにでもしなければUEFIの対応は必須と言えないと思うのですが、どんなものでしょうか。

注5) とはいえ、実際にはUnityでは非表示とかも可能なので、そういうのは追加されないはずです。

図4 スライドショーを変更する

```
$ sudo apt-get build-dep ubiquity-slideshow-ubuntu
$ apt-get source ubiquity-slideshow-ubuntu
$ cd ubiquity-slideshow-ubuntu-*
$ dpkg-buildpackage -r -uc -b
$ mkdir ../ubuntu-defaults-sd/ubiquity-slideshow/slides/ja_JP/
$ cp build/ubuntu/slides/l10n/ja/*  ../ubuntu-defaults-sd/ubiquity-slideshow/slides/ja_JP/
```

第28回

Linux 3.15の新機能
～dm-eraとFile private POSIX lock～

Text：青田 直大　AOTA Naohiro

今月はLinux 3.15で追加される新しい機能を紹介していきます。今回はdm-eraとFile private POSIX lockについて解説します。

dm-era

dm-eraはLinux 3.15で追加される新しいDevice mapperターゲットです。ユーザが決めた“era”（翻訳すれば、時代、時期）と呼ばれる期間に、どのブロックに書き込みが行われたのかを追跡する機能を備えています。

まずはdm-eraがどのようなものかを見てみましょう（**図1**）。

❶ はじめにdm-eraで管理したいデータが置かれる領域（/dev/vg2/eraorigin）を作っておきます。

❷ dm-eraのメタデータを置く領域として、/dev/sdd1を使います。このパーティションの先頭ブロックはdm-eraのフォーマットが走るようにゼロ埋めしておきます。

❸ dmsetupコマンドを用いて、dm-eraデバイ

▼図1　dm-eraの動作

```
# lvcreate -L 10G -n eraorigin vg2  ……❶
  Logical volume "eraorigin" created
# dd if=/dev/zero of=/dev/sdd1 bs=4096 count=1  ……❷
1+0 records in
1+0 records out
4096 bytes (4.1 kB) copied, 0.000791587 s, 5.2 MB/s
# dmsetup create vg2-eradev --table "0 20971520 era /dev/sdd1 /dev/vg2/eraorigin 4096" ……❸
# era_dump /dev/sdd1   ……❹
<superblock uuid="" block_size="4096" nr_blocks="5120" current_era="1">
  <era_array>
    <era block="0" era="0">
    <era block="1" era="0">
    <era block="2" era="0">
…… (中略) ……
    <era block="5117" era="0">
    <era block="5118" era="0">
    <era block="5119" era="0">
  </era_array>
</superblock>
# dmsetup status vg2-eradev   ……❺
0 20971520 era 8 268/4161600 1 -
```

スvg2-eradevを構築します。開始セクタ0から20,971,520セクタ(512×20,971,520＝10GB)を、/dev/sdd1をメタデータデバイス、/dev/vg2/eraoriginをオリジナルデータデバイスとして、dm-eraで追跡される「ブロック」のサイズを4,096セクタ単位としています。

❹ era_dumpコマンドをメタデータデバイスに対して実行することで追跡データを見ること

▼図2　era deviceへの書き込み

```
# dd if=/dev/urandom of=/dev/mapper/vg2-eradev bs=4096 count=1  ……❻
1+0 records in
1+0 records out
4096 bytes (4.1 kB) copied, 0.0049059 s, 835 kB/s
# era_dump /dev/sdd1   ……❼
<superblock uuid="" block_size="4096" nr_blocks="5120" current_era="1">
  <era_array>
    <era block="0" era="0">
    <era block="1" era="0">
    <era block="2" era="0">
……(中略)……
    <era block="5117" era="0">
    <era block="5118" era="0">
    <era block="5119" era="0">
  </era_array>
</superblock>
# dmsetup message vg2-eradev 0 take_metadata_snap   ……❽
# era_dump /dev/sdd1   ……❾
<superblock uuid="" block_size="4096" nr_blocks="5120" current_era="2">
  <writeset era="1" nr_bits="5120">
    <bit bit="0" value="1">
    <bit bit="1" value="0">
    <bit bit="2" value="0">
……(中略)……
    <bit bit="5117" value="0">
    <bit bit="5118" value="0">
    <bit bit="5119" value="0">
  </writeset>
  <era_array>
    <era block="0" era="0">
    <era block="1" era="0">
    <era block="2" era="0">
……(中略)……
    <era block="5117" era="0">
    <era block="5118" era="0">
    <era block="5119" era="0">
  </era_array>
</superblock>
# era_dump --logical /dev/sdd1   ……❿
<superblock uuid="" block_size="4096" nr_blocks="5120" current_era="2">
  <era_array>
    <era block="0" era="1">
    <era block="1" era="0">
    <era block="2" era="0">
……(中略)……
    <era block="5117" era="0">
    <era block="5118" era="0">
    <era block="5119" era="0">
  </era_array>
</superblock>
# dmsetup status vg2-eradev     1   ……⓫
0 20971520 era 8 272/4161600 3 268
# dmsetup message vg2-eradev 0 drop_metadata_snap
# dmsetup status vg2-eradev
0 20971520 era 8 268/4161600 3 -
```

ができます。まだ何もしていないので、すべて0ですね。

❺ “dmsetup status”を使うと、メタデータデバイスの情報を取得できます。“era”の次がメタデータの1ブロックあたりのセクタ数で、これは8に固定されています。その次は“使用中のブロック数/全ブロック数”です。その次には現在のeraが表示され、最後にメタデータのsnapshotの位置が(あれば)表示されます。

では、era deviceの初期化が終わったので書き込んでみましょう(**図2**)。

❻ era deviceにデータを書き込みます。

❼ それだけではera_dumpしても結果に反映されてはいません。

❽ dmsetup messageコマンドでtake_metadata_snapを送り、メタデータのスナップショットをとります。これで変更が読み込めるようになります。また同時にeraがインクリメントされます。

❾ ここでera_dumpすると、current_eraが2にインクリメントされています。また、“writeset”という項目が追加されています。writesetを見ることで、era=1に書き込まれたかどうかを見ることができます。一方で、era_arrayは最後にどのeraに書き込まれたかを出力していますが、まだこの時点では反映されていません。era_arrayへの反映は次のeraに移るときに行われます。

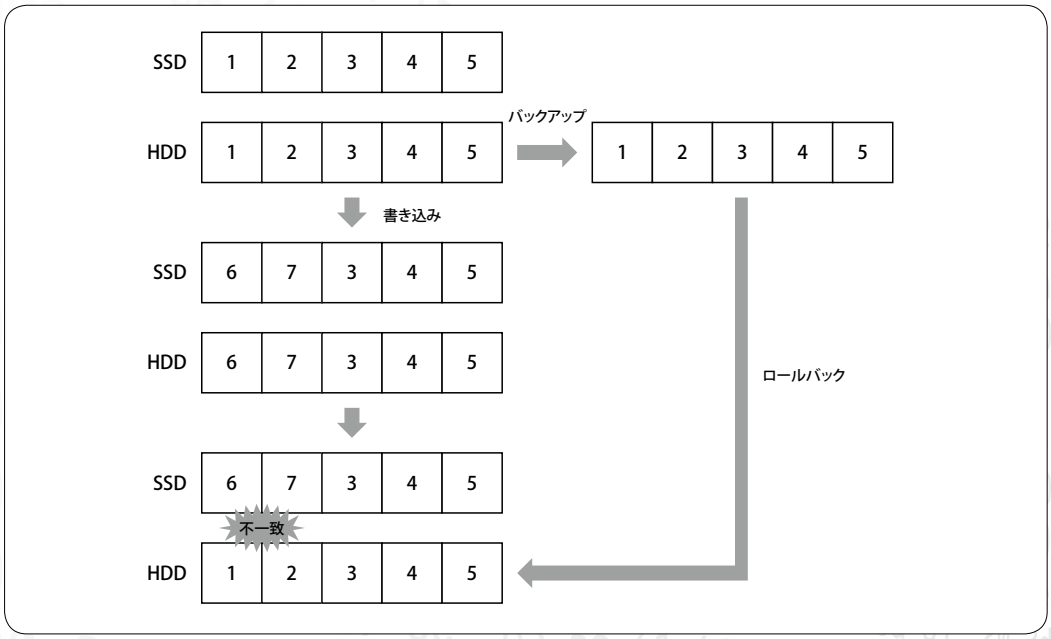
❿ --logicalオプションを使うことで、era_dumpの中でこの情報をマージして表示できます。

⓫ snapshotをとってstatusを見ると、使用ブロック数が増え、メタデータのsnapshotの位置が表示されていることがわかります。

dm-eraとdm-cache

このようにdm-eraを使うことで、デバイスのどの部分が変更されたかを追跡できるようになります。用途の1つとしては、変更された部分だけを転送する効率的なバックアップというも

▼図3　SSDのcacheとロールバック後のHDDに不一致部分ができる



のを考えることができます。また、もう1つ想定されているユースケースとして、dm-cacheと一緒に使うことが挙げられています。

図3のように、HDDの上にSSDのcacheデバイスを設定します。データがcacheにすべて載っていて読み込みはすべてSSDから行われる状態です。ここでデータをHDDからバックアップします。そして、その後書き込みを行ったあとで、先ほどのバックアップからロールバックすることを考えてみましょう。ロールバックを行うと、cacheとHDD上のデータが一致しなくなってしまいます。ここでcacheをすべて捨ててしまうこともできますが、それではcacheが一致している部分も捨ててしまうのでもったいないことになります。

dm-eraを使えばこの問題を解決できます。バックアップをとったあとに、checkpointメッセージを使ってeraをインクリメントし、そのeraを記録しておきます。cacheをpassthroughモード(cacheを無視して、元のデバイスに読み書きしにいく)にして、バックアップからロールバックします。その後、記録しておいたera以降に書き込みのあったブロックに対応するcacheだけを捨ててしまえばcacheとHDDの内容が一致し、安全かつ効率的にロールバック後のcacheの一貫性を保つことができます。

file-private POSIX lock

次に、file-private POSIX(Portable Operating System Interface for UNIX) lockについて紹介します。file-private POSIX lockは、今までのロックとは動作が異なるファイル用のロックです。まずは、今までのロックの動きを見てみましょう。今までのロックにはfcntlによって設定するPOSIXロックと、flockによって設定するBSDスタイルのロックがあります。

リスト1はPOSIXロックをファイルAとBとに行うコードです。AとBをロックしてから3秒、Bをcloseしてから(それによって自動的にBのロックが解除されてから)3秒スリープします。Bをcloseしてもロックは残っている、ことが期待されますね？　では、**リスト2**でロックしているプロセスを表示してみましょう(**図4**)。

❶ まずはファイルA、Bを普通に作って試してみます。"A and B is locked" のあとに、locking

▼リスト1　plock.c

```
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>

int main()
{
  struct flock lk = {F_WRLCK, SEEK_SET, 0, 1024, 0};
  int fda = open("A", O_RDWR);
  int fdb = open("B", O_RDWR);
  if(fda < 0 || fdb < 0)
    return -1;
  if(fcntl(fda, F_SETLKW, &lk) < 0 || fcntl(fdb, F_SETLKW, &lk) < 0)
    return -1;
  printf("A and B is locked n");
  sleep(3);
  close(fdb);
  printf("close B: still remain lock of A? n");
  sleep(3);
  close(fda);
  printf("close A n");
  return 0;
}
```

▼リスト2　watch.c

```
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>

int main()
{
  int i;
  int fda = open("A", O_RDWR);
  for(i=0; i<10; ++i) {
    struct flock lk = {F_WRLCK, SEEK_SET, 0, 1024, 0};
    if(fcntl(fda, F_GETLK, &lk) < 0)
      return -1;
    printf("locking pid: %d n", lk.l_pid);
    sleep(1);
  }
  close(fda);
  return 0;
}
```

pidとして(Aのロックをとっている)plockのプロセスIDが表示されています。その後、"close B"しても期待通りにAのロックは残っていることが見てとれます。

❷ 次にBをAからハードリンクしてみましょう。この場合は、なんとBをcloseしたあとにAのロックも解除されてしまっています！ このようにPOSIXロックでは、あるプロセスがとっているロックは、そのロック対象ファイルに関連しているファイルデスクリプタを1つでもcloseすると、そのファイルに関わるほかのロックもすべて解除されてしまいます。

▼図4　ロック状態の表示

```
$ dd if=/dev/zero of=A bs=1024 count=1
$ cp A B
$ ./watch& ./plock    ……❶
[2] 4179
locking pid: 0
A and B is locked
locking pid: 4180
locking pid: 4180
locking pid: 4180
close B: still remain lock of A?
locking pid: 4180
locking pid: 4180
close A
locking pid: 0
locking pid: 0
locking pid: 0
locking pid: 0
[2]  - done       ./watch
$ rm B; ln A B
$ ./watch& ./plock    ……❷
[2] 13558
locking pid: 0
A and B is locked
locking pid: 13559
locking pid: 13559
close B: still remain lock of A?
locking pid: 0
locking pid: 0
locking pid: 0
close A
locking pid: 0
locking pid: 0
locking pid: 0
locking pid: 0
[2]  - done       ./watch
```

▼図5　unlockを待たずにロックを取得している

```
$ ./thread
file locked
file locked
unlocked
unlocked
```

もう1つのPOSIXロックの問題をthread.cの例で見てみましょう。**リスト3**のプログラムは、2つのスレッドを起動し、それぞれのスレッドに同じファイルのロックをとらせます。一方はすぐにロックをとり、5秒スリープし、もう一方は2秒待ってからロックをとろうとして3秒スリープします。後者のスレッドは前者のunlockを待つことが期待されますが、実際に動かしてみると**図5**のようにunlockを待たずにロックを取得したことになってしまっています。このようにPOSIXロックはプロセス単位で取得され、マルチスレッドなアプリケーションの多い今では使いにくいものとなっています。

▼リスト3　thread.c

```
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <pthread.h>

void *f(void *arg)
{
  int sec = (int)arg;
  struct flock lk = {F_WRLCK, SEEK_SET, 0, 1024, 0};
  int fd = open("A", O_RDWR);
  if(fd < 0)
    pthread_exit(NULL);
  if(fcntl(fd, F_SETLKW, &lk) < 0)
    pthread_exit(NULL);
  sleep(sec);
  printf("file locked n");
  sleep(5-sec);
  close(fd);
  printf("unlocked n");
  pthread_exit(NULL);
}

int main()
{
  pthread_t t1, t2;
  pthread_create(&t1, NULL, f, (void *)0);
  pthread_create(&t2, NULL, f, (void *)2);
  pthread_exit(NULL);
  return 0;
}
```

▼図6　モジュールロード時

```
[ 1139.471972] [netcat]: netcat - Cycles Per Instruction - Kernel
Module Edition - 2014
[ 1139.471976] [netcat]: netcat is Brandon Lucia, Andrew Olmstead, and
David Balatero
[ 1139.471977] [netcat]: On the web at http://netcat.co
[ 1139.471979] [netcat]: 'ogg123 - < /dev/netcat' to play.
```

▼図7　曲名の表示

```
[ 1318.288195] [netcat]: Now playing track 1 - Interrupt 0x7f
[ 1385.893090] [netcat]: Now playing track 2 - The Internet is an Apt
Motherfucker
```

もう一方のBSDスタイルのロックは、flockの定義からわかるように、POSIXロックのようなファイルの内の位置を指定したロックをかけることができず、常にファイル全体にロックがとられてしまいます。その一方でBSDスタイルのロックには、前述したような1つでもcloseすることでロックが外れてしまったり、プロセス単位でロックが行われたりするような問題点はありません。

Linux 3.15では、これら2つのロックを統合したようなFile private POSIX lockが導入されました。このロックはファイル中の指定した部分を、BSDスタイルのロックのようにopenしたファイルごとに独立したロックを取得することができます。使い方はこれまでのPOSIXロックのF_SETLKWなどをF_SETLKPWというようにPを追加したものに置き換えるだけです。

Kernel Module形式のアルバム

最後に、今月はちょっと変わったカーネルモジュールについて紹介します。GitHubでnetcatというバンドの"Cycles Per Instruction"というアルバムがLinux Kernel Moduleの形式で公開されています[注1]。リポジトリをcloneしてmakeすると、netcat.koというカーネルモジュールが作られます。このモジュールをinsmodすると/dev/netcatというデバイスが作られます。さらにdmesgを見てみると**図6**のような説明も表示されています。

説明文にあるようにogg123コマンドを使うと(著者はogg123ではなぜかノイズが出たので、mplayer2を使いました)音楽が流れ出します。さらにdmesgには**図7**のように、音楽の進行に合わせて曲名も表示されていきます。

このリポジトリに、Linuxカーネルのstableメンテナとして有名なGreg K-Hがコードを Linuxカーネルモジュールのコードスタンダードに合うようにするPull Request[注2]や、同時に複数の読み込みができるようにするPullRequest[注3]が寄せられるなど注目を集めています。さらにはFreeBSDサポートのパッチ[注4]まで寄せられています。

まとめ

今月はdm-eraとFile private POSIX lockについて紹介しました。Linux3.15-rc6は、LinusがLinuxCon Japanに来ていたため東京でリリースされました[注5]。このメールによれば、rc7が最後のrcとなるかもしれないということなので、この号の発売の時期には3.16の開発も始まっているでしょうか。SD

注1） https://github.com/usrbinnc/netcat-cpi-kernel-module

注2） https://github.com/usrbinnc/netcat-cpi-kernel-module/pull/11

注3） https://github.com/usrbinnc/netcat-cpi-kernel-module/pull/13

注4） https://github.com/usrbinnc/netcat-cpi-kernel-module/pull/42

注5） https://lkml.org/lkml/2014/5/21/610

July 2014

NO.33

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

技術は変わり、インフラは不変になる

今回は5月に沖縄で行った研究会の模様をお伝えします。

沖縄大会

■AWSで行うImmutable Infrastructure
【講師】米須 渉（JAWS-UG沖縄）
【司会】法林 浩之（日本UNIXユーザ会）
【日時】2014年5月24日（土）15:15～16:00
【会場】沖縄コンベンションセンター
会議場B1（B棟2F）B会場

今回はJAWS-UG沖縄の代表を務めている米須さんを講師に迎え、最近流行しつつある使い捨て方式によるシステム構築や、それをAWS（Amazon Web Services）で実現することの利点などについて講演していただきました。本研究会の舞台となったオープンソースカンファレンス2014 Okinawaの総参加者数150人に対して本セッションに30人の参加があり、高い関心を感じさせました（**写真1**）。

■Immutable Infrastructureとは「使い捨て可能な環境」のこと

講演の前半はImmutable Infrastructureに関する話でした。直訳すると「不変なインフラ」ですが、直観的にはやや理解しにくい言葉です。同義語として使われるDiposable Components、すなわち「使い捨て可能な部品（サーバ群）」のほうがイメージしやすいでしょう。サーバを使い捨てにするとは、サーバの設定を変更するようなメンテナンスを行うときはサーバを作りなおすことを意味します。これにより、本番環境に手を入れるということがなくなり、作業ミスが減るだろうという考え方です。

写真1　沖縄大会の様子

この考え方の基礎になったのが、マーチン・ファウラーさんが提唱したBlue-Green Deploymentという手法です。本手法の根底には、本番環境へのデプロイが一番危険であり、またダウンタイムは最小にしたいという考えがあります。そこで本手法では、Blue（現在稼動中）とGreen（新バージョン）の2つの本番環境を用意し、ロードバランサを用いて本番環境全体を瞬時に切り替えることによりデプロイを行います。

この手法を実践するのにAWSは非常に適しており、ダウンタイムが短く、切り戻しも楽、さらに環境ごとに地域やAZ（Availability Zone）を分けられるのでディザスタリカバリ（DR：Disaster Recovery）の観点からも良いシステムを作ることができます。

■捨てても良い環境／捨てられない環境の見極め

ここで講師から「クラウド脳」という言葉が提示さ

れました。歴史上の流れとして、昔は自前で物理サーバを構築していたのが、データセンターのホスティングサービスを利用するようになり、次はVMWareやXenなど仮想化ソフトウェアが登場し、続いてそれがデータセンター上に導入されたVPSを利用し、さらに最新動向としてはOpenStackなどのクラウドソフトウェアやAWSなどのクラウドサービスを利用するようになってきました。このような状況になるとシステム作りに対する考え方が根本的に変わってきて、落ちないサーバ群を用意するのではなく落ちてもサービスを提供できるように考える必要がある、それがクラウド脳であるというものです。

クラウド脳では、サーバをStatelessとStatefulに分けて考えます。Statelessとは状態を持たないサーバで、捨てても良い状態で運用します。コンテンツを見せるだけのWebサーバやアプリケーションサーバなどはStatelessで十分です。ただし、ログはStatelessなサーバに置いて消えてしまってはいけないので、syslogやfluentdを利用してStatefulなサーバに集めます。一方、Statefulは状態を持つサーバで、ファイルサーバやDBサーバなどが該当します。これらについては従来と同じ運用が必要です。

ここでDRなどの観点から複数拠点に環境を構築することを考えると、データセンターを複数契約するよりもAWSを利用して複数のリージョンで構築するほうが安価で済みます。このように、クラウド脳ではできるだけSingle Point of Failure（単一障害点）をなくすように考えることが肝心です。

■Immutable Infrastructureを加速する？ コンテナ型仮想化の実演

ここから後半に入り、Dockerの話題に移行しました。Dockerはコンテナ技術を用いたオープンソースの仮想化ソフトウェアで、最近急速に開発が進んでおり、注目を集めています。それまでの仮想化ソフトウェアと比べても、さらにすばやくサーバ台数の増減ができることなどが歓迎されているようです。

ハイパーバイザー型仮想化基盤ではハードウェアの上に仮想化ソフトウェアがあり、その上に仮想マシンがありますが、コンテナ型仮想化基盤では仮想化ソフトウェアの階層がなく、ホストOSの上に、ライブラリやアプリケーションをグループ化したものが複数稼動するという構造になっています。OSではなくプロセスの起動になるため、「起動／終了が速い」、「必要なサービスだけを動かせる」、「仮想マシンの上でも動く」などの利点がありますが、慣れないと動作を少しイメージしにくいのが難点です。また、動作環境は基本的にLinux系に限られます。

そして最後にDockerのデモがありました。講師のノートPCで動作するUbuntuの画面を提示し、コマンドライン上でDockerのコマンドを実行することにより次の操作が行われました。

- docker runでCentOSを起動し、/etc/issueを見せてCentOSであることを示し、CentOSを終了
- docker runでCentOSを起動し、その上で/bin/bashを起動
- bash上でyum updateを実行するとyumが動作するが、CentOSを終了するとOSは捨てられるので、実際にOSが更新されるわけではないことを示す

時間がなくなったので実演はここまでになりましたが、yumでOSを更新したあと、commitすれば更新したOSのイメージが保存されるので、これを用いてシステムの更新履歴を残したり、何かトラブルが起きてもその前の状態に戻せるという説明がありました。

◆ ◆ ◆

Immutable InfrastructureやDockerなど最新技術が紹介されましたが、単に技術用語を羅列するだけの内容ではなく、重要なポイントや考え方を伝えることに重点を置いてわかりやすく解説していただきました。これからこの分野を勉強したい人にとって非常に良いきっかけとなるセッションでした。今回の講演の資料は講師により公開されています[注1]ので、そちらも参考にしてください。SD

注1）URL http://www.slideshare.net/asumaslv/ocs2014

Hack For Japan
エンジニアだからこそできる復興への一歩

●Hack For Japanスタッフ
鎌田 篤慎　KAMATA Shigenori
Twitter @4niruddha

第31回

Hack For Japan 3.11～3年のクロスオーバー振り返り（後編）

“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」。前回に引き続き、3年間の活動を振り返ったイベントの報告です。

先月号に引き続き、各地でHack For Japanの活動に共感し、共に活動してきた開発者達と東日本大震災からの3年を一緒に振り返るイベント「Hack For Japan 3.11～3年のクロスオーバー振り返り」の様子をレポートします。Google+ハングアウトオンエアで宮城県の仙台と石巻の2会場、岩手県の釜石会場、福島県は中通りの郡山、浜通りの南相馬、会津のそれぞれにある3会場、そのほかに東京会場や大阪会場、海外のシドニーをインターネット中継で結んで行われました。

新しい世代の芽と本当の復興が見えつつある会津

福島県会津からの発表はHack For Japanのメンバーでもある佐々木陽さんによる、会津地方の紹介と福島の現状報告です。

原発の問題が依然として継続し、立ち入りできない土地が現在も数多くある福島の今を冒頭に説明。震災以降、会津で実施されたHack For Japan関連のハッカソン、最新のハードウェアやガジェットを使った新しい街作りを目指す「Hack For Town in Aizu[注1]」（**図1**）の活動を紹介いただきました。

◆**図1　Hack For Town in Aizu**

そうした活動がきっかけで、先日市長賞を受賞した五十嵐太清くんが会津で育つ若い芽として紹介されました。彼は会津工業高校に寄付されたAndroid端末を用いてガイガーカウンターなどを開発し、その流れでハッカソンに参加するようになり、会津大学進学後も継続して活動に参加しています。

五十嵐くんは前回のレポートでも紹介した、宮城県石巻のイトナブ[注2]に通う若者達と連携した活動も始めています。各地で若い芽が着実に育ち、つながっていっています。そして、Hack For Japanの流れから生まれたCode for Aizuによって行政の人達とも緩やかなつながりが生まれ、地域に根ざした活動になりつつあります。このような活動が最近では頭に「震災」の文字が付かずとも地域のために行われるようになってきたことから、本当の意味での復興が見えてきたとのことでした。

「ふくしまの未来＝希望」を目指すエフスタ!!郡山

同じく福島県郡山からは、本連載2013年2月号でも紹介した「ITスキルアップコミュニティ エフスタ!![注3]」の大久保仁さんからの発表です（**図2**）。

福島県のITエンジニア達は大きく、浜通り、中通り、会津といった土地柄による隔たりがありましたが、今回の震災、そしてこのクロスオーバー振り

注1　http://blog.hack4.jp/2014/03/hack-for-town-in-aizu.html
注2　http://ishinomaki2.com/
注3　http://efsta.com/。郡山での活動から徐々に東北や東京まで活動の幅を拡げています。

◆図2　福島県郡山からのハングアウト中継

◆図3　福島県南相馬からのハングアウト中継

返りイベントで集い、つながったことに対する喜びの声から始まりました。

福島の中でも浜通り、中通りは原発の問題が継続しており、制限された生活を強いられています。食べることや遊ぶことなど多くの面で課題があります。そうしたことから福島では現状復帰という意味での復興よりも、その前提としての「安心して暮らせる」ということが求められており、エフスタ!!は勉強会を中心に活動し、福島の今を伝える活動を東北各地、そして東京などでも行っています。「震災前より元気な福島、福島は面白いぞ！」というのを福島の子供達にも感じ取ってもらえるよう頑張っているとのことでした。

発表の最後はエフスタ!!に参加するメンバーひとりひとりが、中継を見ている参加者に一言ずつメッセージを発表するという形で締めくくられました。

放射線被災下でITに活路を見出す南相馬

福島第一原発がある浜通りの南相馬からは、「南相馬ITコンソーシアムhttps://www.facebook.com/minamisoma.it」の立ち上げに関わられている田中章広さんからの発表となり、その報告は日本各地、世界各地から寄せられた南相馬への支援の数々への感謝から始まりました（**図3**）。

南相馬がある浜通りは津波被害が大きな地域と、事故当時の風向きで放射線の影響が大きくなってしまった地域などが入り組んだ複雑なもので、それぞれに違った対応が迫られています。中でも人口流失の問題は深刻で、震災直後は原発の影響で9割近くの住民が避難し、震災から3年が経った今でも以前の6割程度の住民しか戻ってきていないとのこと。

とくに南相馬の産業の中心だったサービス業などに従事していた若年層の流出が著しく、新たな産業を南相馬で見出さなければ街がダメになってしまうという強い危機感のもと、放射線被災下においてもやっていける新規事業を模索するうちにIT産業に活路を見出したそうです。しかし、IT技術を教育しても他地域に技術者が流出してしまう問題は解決しません。そんな課題から、教育プログラムから雇用までをサポートする南相馬ITコンソーシアムを立ち上げたとのことでした。

当初はエンジニアが1人もいないところからスタートしたため、国内でもITに早い段階から力を入れていた岐阜県大垣市に協力を依頼し、南相馬へプロジェクトマネージャーを派遣してもらいました。合宿形式でシステム開発の習得に腐心した結果、エンジニアも育ち、現在までの実績として映画や球団、TV番組の公式アプリ開発を21本ほど受注、納品されています。そしてほかの被災地域同様、プログラミングワークショップなどの形で若年層の育成にも力を入れはじめているとのことでした。ここでも会津と同じように、石巻のイトナブと復興に向けて協力していくつながりが生まれています。

次なる減災のフェーズへ東京

東京会場からの発表は、宮城県石巻市出身でTwitterアカウント @UN_NERV[注4]を使った迅速な災害情報の告知で活躍する石森大貴さんから、その

注4　https://twitter.com/UN_NERV

アカウント誕生の経緯、これまでとこれからを発表していただきました。

3月12日、都内で大規模停電の知らせが拡がります。首都圏での電力不足を補うため輪番停電の実施が決定されたことを受け、アニメ『新世紀エヴァンゲリオン』に登場する組織「NERV」を表すTwitterのアカウント（@UN_NERV）を持っていた石森さんが「ヤシマ作戦」を訴え節電への協力をインターネット上で呼びかけはじめました。「ヤシマ作戦」とは『新世紀エヴァンゲリオン』の劇中に登場する、敵を倒すために行った日本全国の電力を集める作戦「ヤシマ作戦」になぞらえたものです。

その活動が注目を集め、実際のアニメを模したサイトやアプリケーションが生まれることにつながります。その時点では著作権などへの問題が指摘されていたそうですが、震災直後の緊急時という点と少しでも被災地のためにできることとして、石森さん自身が責任を取れる範囲であれば構わず続けるという意思のもと、優先すべきことを優先するという姿勢を貫いたそうです。そのような姿勢に多くの人々も賛同し、サーバの提供なども含めさまざまな協力者が現れはじめます。

震災から2日目、活動当初から指摘されていた新世紀エヴァンゲリオンの版元に対する著作権の問題が進展します。Twitterのフォロワーから版元と調整してくれる人物が名乗り出て調整してくれたことで、『新世紀エヴァンゲリオン』公式ブログからヤシマ作戦が正式にアナウンスされました。石森さんの活動が多くの人達を動かし、人々が求める情報を配信するインフラ作りの第一歩になった瞬間でした。

届けなければならない情報が届いていない。そうした背景があり「生命と財産の保護が最優先」というミッションのもと、石森さんはインターネットの力を使ってこの課題の解決に取り組んでいきます。ブログやTwitterなどで被災地のために地道に続けていた活動が大きな動きとなったことで、気象庁からは防災気象情報、総務省からは公共情報、IIJからは緊急地震速報などの防災、減災に関する情報提供などさまざまな組織から情報が提供されるようになりました。

こうした石森さんの活動は東日本大震災だけに留まらず、日本で起こる災害全般に視野が拡がっています。インターネットから情報を届けられるデバイスが増加していることを受け、現在はカーナビとの連携も予定されているそうです。一斉配信でも輻輳（ふくそう）が起こらない放送波を使うしくみの利用を考えられているとのことでした。

釜石の復興と若者のITスキル育成のために

釜石会場からの発表は、地方におけるICTの利活用やさまざまなライフスタイルを通した地域振興の発展を目指す「LiFESTYLE Lab.[注5]」を運営されている西条佳泰、さやかご夫妻からの発表となりました。お二人は震災後に都内のIT企業と広告代理店をそれぞれ退職し、ふるさと支援の目的で岩手県釜石市にUターン。ITスキルを活用して地元の復興のために尽力されています。

釜石も南相馬などと同様に震災後から人口の流出が止まらず、従来の産業だけでは先細りが見えているため、地元に新しい活路を見出すべく、西条さんはITの利活用を地元で推進していきます。その活動は震災のあった2011年から始まり、我々Hack For Japanのハッカソンなどを中心に、釜石でいくつかの活動を開催されてきました。その流れで、地元の中小企業の人達にITスキルを身につけさせるコーディネーターとしても活躍し、現在も釜石でITを根付かせる活動をしているとのことでした。

2013年の夏には、「東北TECH道場[注6]」が第4期から釜石でもスタートし、下は小中学生から上は50代まで、幅広い層の人間から支持を集めているそうです。そうした活動や西条さんの人となりから、地元との連携に発展し、やがて地域の高校と連携が始まります。2013年9月には震災後3年ぶりに地元のお祭りである「釜石よいさ」が復活し、地域の高校生の授業の一貫として行われたお祭りの公式サ

注5 https://www.facebook.com/pages/LiFESTYLE-Lab/576418645734458
注6 https://sites.google.com/site/tohokudojo/

イトを指導者として一緒に制作されました。岩手県の故郷をテーマにしたCMでは釜石よいさのCMが賞を受賞し、釜石は震災後、これ以上ないほど人が集まったそうです。

現在、釜石は震災後からの再開発が進み、中心市街地ではイオンタウンが建設中です。その集客力を地域振興に活かすべく、再開発にかかわるメンバーの一員としても参加しています。とくにICTの効果的な利用を狙って、釜石市の地域情報、観光情報、市役所、市民記者、企業や事業者の情報を1ヵ所に集約する「情報交流センター」を企画し、自立したICT活動やイベントが行えるハブとなることで、中高生のIT教育、地元民のITリテラシーの向上を目指しているとのことでした。

「東北TECH道場」釜石道場の道場主である奥さんの西条さやかさんからの発表は、釜石道場に通う中学生2人のお話でした。大人も交じる釜石道場の中で、その2人は最初はプログラムに苦戦し、くじけそうになりつつも釜石道場に通い続けました。そのため今では、周囲の大人達を超える勢いで成長しているそうです。そうした成長してゆく子供達の姿を見て、道場に通う大人達も、道場主である西条さん自身も大いに刺激を受けているとのことでした。今後はその2人の中学生が高校生になるので、高校生の参加者も増やしていきたいという展望をうかがいました。

石巻のイトナブや会津の五十嵐くん、南相馬の若者など次世代のITエンジニア達が育ち、そして緩やかにつながっていく様は東日本大震災からの復興だけでなく、日本の将来も明るく照らすことを伝える発表でした。

大槌から復興に留まらない地域振興を

大阪会場のトップバッターは「KAI OTSUCHI[注7]」を運営されている鷲見英利さんからの発表となりました（**図4**）。

KAI OTSUCHIの成り立ちは、大阪でのアプリ開発企業の経営、横浜での外国人向け不動産会社の経営、貿易会社経営など、さまざまな業態の企業を営んでいる鷲見さんがその経験を買われ、関西大学の与謝野有紀教授より鷲見さん宛に、被災地において仕事を創出する活動を考えてほしいという依頼があったことに端を発します。鷲見さんが以前に中国へのオフショア事業に携わっていた経験から日本の競争力をあらためて考えたこともあって、日本の若手育成の観点から、ニアショア事業と復興支援、その先にある地域振興を意識した自立した成長を支援する活動を行う目的で始められたとのことでした。

◆**図4　KAI OTSUCHIのホームページ**

当時、大槌町にはプログラムができる人材がそもそもいませんでした。そこで人材をゼロから教育してアプリ開発ができるようになるところまでを目指し、若者の職場づくりを加速させる目的でスタートしました。その結果、半年で電子書籍のアプリを作れるようになっていったそうです。当初はそうした若者達が首都圏での就職が容易になるよう、彼らの仕事の成果として名前を露出することまで想定していました。しかし、そうするとせっかく育った若者が首都圏に移住し、地元の人材流出を加速させる懸念があります。そこで現在では、一般社団法人を作り、地元での雇用の創出、人材の育成にシフトしているそうです。

そうした中で開発された「京都カメラ[注8]」をきっかけに、観光地との連携が商材になりつつあります。「義援金より仕事を」というキャッチコピーで、ただ教育するだけでなく、実作業、実績をつくっていくことを目指した活動になっています。

振り返りイベント当日も、ただプロダクトを作る

注7　http://kai-otsuchi.com/

注8　http://kai-otsuchi.com/camera.html

だけでなく、そのコンセプトがほかの観光地などにも水平展開できるところがビジネスとしての継続性もあって素晴らしいという意見もあがりました。

大阪からのイノベーションを目指し東北と連携

続いて大阪会場より「Osaka Innovation Hub[注9]」を運営する大阪市役所の角勝さんからの発表です。ハッカソンやオープンデータをキーワードに、イノベーションの創出を試みている関西イノベーション国際戦略総合特区にあるイベント会場を軸にした活動の紹介です。

日本以外の国、大阪以外の地域で継続的なイノベーションが生まれる環境を目指しているとのことで、グローバルに人材、情報、資金が入り込む環境で、ほぼ毎日イベントが開催されています。その中でも、我々が開催するようなハッカソンをかなり実施しているとのことでした。

とくに最近ではハードウェア系のものに力を入れいており、そうした活動が認知されて、パナソニックやシャープといった大手企業とも連携したハッカソンにまで発展しているとのことでした。そうした活動の中、東北との連携では「Earth Communication Award 2013[注10]」で、石巻や仙台でハッカソンを実施し、成果をあげているというお話でした。

ハッカソンではどうしても単発的なアウトプットとなってしまいがちな点を考慮して、継続的な活動、事業展開が見込める成長を目指し、ハッカソンからのプロダクトを展示する展示会を開催したりしています。やっただけで終わりにしない施策にも力を入れており、ハッカソン発のプロダクト、サービスの支援を行っているというお話でした。こうした部分はかねてからHack For Japanの反省でも指摘されているとおり、大切な部分だと思います。

防災と地域活性化を目指すFandroid Kansai

「防災と地域活性化のためのITのあり方をコミュニティから考える」をテーマに「Fandroid Kansai[注11]」設立記念ワークショップの様子を同コミュニティの佐藤拓也さんより紹介いただきました。

Fandroid Kansaiは「Fandroid EAST JAPAN[注12]」という、“震災復興から地域振興まで目指すAndroidアプリ開発”を活動の軸とした東北地方のコミュニティの関西支部です。自立から飛躍を目指し、関西地域の防災、地域振興のナレッジを東北にもつなげていこうという試みです。また、今回の発表では阪神大震災というバックボーンを持つ関西地域ならではの視点もあった、設立記念ワークショップで行われた講演やワークショップの内容紹介でした。

基調講演は、京都大学 防災研究所の准教授 畑山満則さんによる東日本大震災、阪神大震災などの教訓から今後の防災に活かす研究結果の発表です。ITの役割が阪神大震災のときと東日本大震災のときとでは役割が大きく変化してきており、防災への活用はこれから本格化していくと予想。また、被災時に提供できるものとして「安全」と「安心」の2つがありますが、ITが寄与できる領域は物理的な安全より、安心を届けられる点のほうが、今後より見込まれてくるだろうという話があったとのことでした。

また、同イベントのほかのパネルディスカッションの様子として、一般社団法人東日本大震災復興サポート協会の代表である遠藤雅彦さんと、南三陸震災復興推進課まちづくり推進室長の畑文隆さんの対談が紹介されました。災害時、向こう3軒の近接住民間のコミュニケーションが大切と捉え、それを日頃からITの力で補助しつつ、作り上げていくことが大事だという話でした。これは災害時にあらゆるインフラが一時的に麻痺しがちな状況下で、人と人との日頃からのコミュニケーションがいかに大切かという部分で、非常に印象に残るお話でした。

そのほかのパネルディスカッションのお話としては、大阪市旭区役所 総務課防災等担当課の有信博孝さんによる、子供向けの防災教育プログラム「カエルキャラバン[注13]」の活動紹介や、音楽活動のCD

注9 http://www.innovation-osaka.jp/ja/
注10 http://ec-award.com/
注11 https://www.facebook.com/FandroidKansai
注12 http://fandroid-ej.org/
注13 http://kaerulab.exblog.jp/

の売上を全額陸前高田に寄付され、iTunesの売上を子供支援協会などに寄付しているMusic ActivisのShihoさんのお話が紹介されました。Shihoさんのお話の中で「ITによって他人事を自分事にすることができれば、人はもっと動く」というメッセージは参考になる視点です。また、西宮経済新聞編集長である林拓真さんの発表は、日頃からメディアやアプリを使って地元のお店の情報発信を行っているインフラを活用し、災害時に利用するというアイデアのお話でした。

そうした講演をインプットとして受け、Fandroid Kansai設立記念ワークショップに参加された参加者全員でアイデアソンを実施し、今後の防災に役立つアイデアを出し合ったとのことでした。

「これまで」と「これから」と

今回のクロスオーバー振り返りイベントの締めくくりとして、Hack For Japanの立ち上げメンバーでもある及川卓也さんより、出張先のシドニーから、本イベントの総括とこれまでのHack For Japanの振り返り、そしてこれからのことをお話しいただきました（**図5**）。

今年でちょうど3年が過ぎ4年目に入って、及川さんの当時のTweetを振り返ると、東京で何もできないジレンマから、同じ想いを持つ人達を集めるべくHack For Japanを2011年3月18～20日の震災直後に立ち上げ、そこから自分の世界が変わったという話は、会場にいる誰もが同じことを想い、そして今に至るまで活動してきた背景から非常に共感できる話でした。

当時はハッカソンを主軸にした活動でもあったため、停電や余震の影響を受けて、被災していない西日本の会場やオンライン会場、また海外のチームと時差を利用して効率的に作業を進めました。そうしたことがきっかけで多くの人がつながり、3年経った今も今回のイベントのように多くの人が、多くの場所で集まり、振り返りを行えています。しかし一方で、ハッカソンなどの活動で成果がなかなか生まれなかったことも事実としてありました。これが1

◆**図5　及川さんのハングアウト中継**

つのきっかけとなって、継続性を意識しているCode for Japanや東北TECH道場、イノベーション東北の活動に通じていると思います。

また、ヤシマ作戦と同じように批判などはあっても、意志をもって継続することの大切さや、新たな産業振興や教育についてITを軸に据えて未来を見ることの大切さ、そうした活動を日本だけでなく世界にも視野を拡げて見てみようという意識を、今回のイベントを通じて会場にいる参加者全員で得ることができました。

3.11は悲しい出来事でした。しかし、その悲しみを乗り越えて復興、地域の活性化を図るためには「笑いあいながら、楽しみながら」。そうしたメッセージが今回の発表でも随所にありました。むしろ、そうした明るさから未来が見えてくるのではないかという及川さんのメッセージで今回のイベントは締めくくられました。

震災からの3年間を振り返って

東日本大震災からの3年間を振り返ったこの記事をご覧の読者の皆さんも、当時を思い出されたでしょうか。当時できなかったこと、今ならできること、時間の流れと共に私たちにできることも変わります。技術者にしかできないことというのも数多くあります。この記事が読者の方々にとって、「こういうことなら私にも支援できる」「災害に備えてこういうことを準備しておこう」「もし次に災害が起きてしまったときにはこう動こう」など、3.11の教訓を生かすために少しでも参考になれば幸いです。SD

温故知新
IT むかしばなし

第35回

プリンタ

北山 貴広　KITAYAMA Takahiro　kitayamat@gmail.com　Twitter：@kitayama_t

はじめに

今回は筆者が使っていた機種を中心に、記憶に残っているプリンタについてお話したいと思います。

放電破壊プリンタ

1980年ころ、筆者が通っていた高校の成績処理に「コモドール PET 2001」が導入されました。定期テスト（中間、期末テスト）や実力テストの結果が、放電破壊プリンタで印字されて配られました。あまり見やすいものではなく、保存性もよくなかったのですが、各教科の個人の点数と平均点と順位と偏差値が印字されており、成績を伝えるには十分でした。

放電破壊プリンタとは、導電性のあるアルミニウムが蒸着された用紙（チューインガムの包み紙のような銀紙）にヘッドを近づけて放電させ、表層を破壊して印字するものです。用紙はスーパーのレジ用紙のように巻紙になっていました。印字結果は読みやすいとは言えず、文字部分が若干黒くなった程度でした。当時はアルファベットと数字しか印字できないものでした。

ちなみに高校には、「Canon のキャノーラ」というプリンタが付いたプログラム電卓がありました。サイズは蓋を閉じた「東芝 SPARC LT AS1000」より少し大きく、電卓と呼ぶには巨大なもので、当時でも誰も使わずに放置されていました。実行速度は、1から10までの三重ループでピタゴラスの定理を満たす直角三角形の3辺を印刷するプログラムを放課後に実行して帰宅し、翌朝登校してやっと結果が印刷されるほどの遅さでした。

EPSON MP-80/RP-80

「EPSON MP-80」は、1980年11月にEPSONから発売された9×9のドットマトリックス（文字を行と列の点で表現したもの）を打ち付けて印字する方式の、ドットインパクトプリンタです。

ドットインパクト方式は、ヘッドと呼ばれる、点（ドット）を打ち付ける（インパクト）ピンが並んだ部分と、インクが付いたリボンを重ね、用紙にインクを打ち付けるようにして印字するものです。ヘッドが横に動くことで1行分が印字されます。

ドットインパクト方式のプリンタは、物理的に圧力をかけて印字するので、宅配便の伝票でも利用されている複写式の用紙に今も活用されています。動作音が高い周波数で大きいため、夜に印刷するときは座布団をかぶせたりしたこともありました。

この機種は海外ではMX-80という型番で、米国のIBM-PCの普及とともによく売れて、EPSONのサイトにもマイルストンプロダクツとして掲載されています[注1]。この機種のType IIIではEPSONの標準プリンタコマンド体系の「ESC/P」が初めて採用されました。用紙は紙送り機構のために両端に穴の開いたフォームフィード紙を使用します。

筆者が最初に購入したプリンタは、このMP-80 Type IIIの後継機種の「RP-80」で、当初はApple IIで使ってました。前の機種に比べて印字速度が上がって、価格も89,800円と5万円も

注1）http://www.epson.jp/ms/1980_10.htm

値下がりして10万円を切り、当時としては普及価格帯になりました。1986年に、NEC PC-9801 VX2を購入した際、一太郎で書いた卒業論文をこのRP-80で印字してみたことがあります。RP-80は漢字プリンタではないため、ソフトウェアで全部ビットイメージに変換して印字する方式でした。A4、1枚分の印刷に約30分かかったので実用的ではなかったのですが、ソフトウェアで処理すれば漢字の印字も可能なんだという実感を持ちました。

Apple LaserWriter

「Apple LaserWriter」は1985年にAppleから発売された、Macintosh用の300dpiのレーザープリンタです。LaserWriterはページ記述言語にPostScriptを採用したことでも有名で、MacintoshとLaserWriterと当時発売されたページレイアウトソフトのAldus PageMakerとの組み合わせは、デスクトップパブリッシング(DTP)の幕開けをもたらしました。大学の研究室にMacintosh PlusとLaser Writerが導入されて印刷したときは、美しい英字のベクトルフォントで書籍と同等(印刷屋さんからは怒られそうですが)なものが手元の環境でできるということで非常に感動しました。

HP DeskWriter

「HP DeskWriter」は筆者が1991年にMacintosh IIciと一緒に購入したヒューレットパッカード(HP)のインクジェットプリンタです。1998年2月にHPはDeskjetを初めて発売し、その後Macintosh向けにDesk Writerという名前で発売しました。このプリンタもLaserWriterと同様300dpiの解像度があり、デザインもかっこいいものでした。当時の値段は198,000円と(LaserWriterに比べると)とても安く、同じようにきれいな印刷ができました。Macintosh用のプリンタを買う際、ほかのプリンタとの比較検討した記憶がないので、個人で買うときにはこれで決まりだったのかもしれません。

EPSON AP-500

「EPSON AP-500」は漢字フォントを本体に持った熱転写プリンタで、この印字方式はドットインパクトの打ち付けが熱に変わり、インクリボンから3色のカラーフィルムに変わったものです。インクは3色順繰りになっていて、1行印刷するのにカラーフィルムが色数の分だけ繰り返し送られます。このプリンタは印刷が遅かったのと、インクリボンがとてももったいなくてよっぽどのことがない限りカラー印刷は使うことはありませんでした。

また、使用済みインクリボンの熱転写された個所は透明になっているため、あとからインクリボンをもとに戻すと印刷した内容が反転した形で見えるのでセキュリティ的には問題がある方式でした。

EPSON PM-A900

「EPSON PM-A900」は2004年10月に発売されたEPSONの複合機で、35mmフィルムスキャナ機能が付いた最上位機種です。この頃には、インクジェットが普及してデジカメの普及とともに写真印刷もプリンタを使うようになり、年賀状と写真の印刷が重要な要素となりました。またプリンタ単機能から、スキャナとの組み合わせでスキャンとコピーとしても使える複合機が主流となりました。

Canon PIXUS MP-960

「Canon PIXUS MP-960」は、2006年10月上旬から発売された複合機の最上位機種で、これも35mmフィルムスキャナ機能が付いていて、つい先日まで筆者のところでは現役でした。それまでずっとEPSONを使ってきたのですが、最上位機種が2年で故障してしまったことと、このころのEPSONの複合機は不必要に大きくてデザインが好みではなく、継続して買う気にはなれなかったのです。

終わりに

今回執筆しながら購入プリンタを眺めていたら、プリンタの機能も方式も形も、ずいぶん進化したものだと感じました。SD

Report

サイバーエージェント、「Ad Engineering Summit 2014」を開催

2014年5月15日と16日、東京ミッドタウンホール（東京都港区）において、「Ad Engineering Summit 2014」（㈱サイバーエージェント主催）が開催された。

同イベントは、広告／IT業界の技術者や経営者に向けて、アドテクノロジー（以下、アドテク）の最新動向や専門技術についてのセッションと、来場者同士のつながりの場を提供するのが目的。セッションにはアドテク業界の第一線で活躍する人々（約30名）が登壇した。

キーセッションでは、Criteo社のジェイソン・モース氏、Appier社のチハン・ユー氏、TapCommerce社のマーク・ヘール氏、StrikeAd社のマリオ・ピロ氏、㈱medibaの菅原健一氏によりパネルディスカッションが行われた。

現在、アドテク業界では「RTB（Real Time Bidding）」というリアルタイムに広告枠の入札を行う技術が隆盛を極めている。この技術の導入のおかげで、今ではWebの1日の広告件数は45億件にものぼり、この状況は「広告流通革命」と呼ばれている。ディスカッションでは、この広告流通革命のこれからについて意見が交わされた。議論の内容は、次のサイトでレポートを掲載しているので、そちらを参照のこと。

▼アドテク業界のキーパーソンが集結！「Ad Engineering Summit 2014」開催
http://gihyo.jp/news/report/2014/05/2102

アドテク業界は、日本では注目され始めたばかりだがすでに多くの目標や課題を抱えている。イベントでも優秀な人材を求めていることが感じられた。腕に覚えのある人は、今後注目してみてほしい。

▲パネルディスカッションの様子

CONTACT ㈱サイバーエージェント
URL http://www.cyberagent.co.jp/

Hardware

ベルキン、キーボード一体型カバー「iPad Air対応QODE Thin Typeキーボード」を発売

ベルキン㈱は4月15日、「iPad Air対応QODE Thin Typeキーボード」を発売した。

同製品は、1枚のアルミ板を加工して設計されたユニボディデザインのiPad Air用キーボード一体型カバー。最薄部が4mm、重量が340gという薄型軽量が特徴で、キーボードカバーの中では世界最薄（2013年12月時点）となる。かさばらないので、持ち運びの際に便利であるほか、次のような特徴がある。

- 音楽の再生や停止、コピー&ペーストなど便利なショートカットキーを搭載
- カバーを開けば自動的に電源がONになり、閉じればスリープ状態になるオートウェイク機能を搭載
- 正確なタイピングが可能なベルキンTrueTypeキーを採用。キーひとつひとつの間隔をあけてデザインされている
- スナップセキュアマグネットによりiPad Airをしっかり固定できる
- 耐久性能の高いカバーは、持ち運びの際にもiPad Airが傷つくのを防ぐ
- iPad AirとはBluetoothで接続する
- ビューアングルは、iPad Airを使用し作業するうえで一番疲れない角度（約35度）を採用
- 最大79時間のバッテリーを内蔵。充電は同梱のマイクロUSBケーブルを使用してを行う
- iPad Airの重さによって自動でON／OFFが切り替わるスイッチを搭載
- カラーはグレー（型番：F5L155qeGRY）とホワイト（型番：F5L155qeWHT）の2色を用意

▲iPad Air対応 QODE Thin Typeキーボード（ホワイト）

CONTACT ベルキン㈱
URL http://www.belkin.com/jp/

Service

アシスト、「LibreOffice」の研修用テキストを公開

㈱アシストは5月7日、オープンソースのオフィスソフトウェア「LibreOffice」の研修サービス提供のために作成したテキストを、改変および商用利用が可能なクリエイティブ・コモンズ・ライセンスで公開した。これにより、誰もがテキストをカスタマイズしたり、同テキストを利用した研修コースを社内外で自由に実施できる。

同社は、OpenOffice.orgを2007年2月に社内標準ソフトウェアとして運用を開始、その経験を活かして導入支援、ヘルプデスク、集合研修などの各種支援サービスを顧客企業に提供してきた。2012年6月には派生ソフトであるLibreOfficeを全社で採用し、OpenOffice.orgと同様のサービスを提供してきた。今回の公開は、LibreOfficeを利用する企業や団体が増え、LibreOfficeを取り巻くエコシステムの発展に少しでも寄与したいと考えてのことだという。なお、本テキストはLibreOffice4.0.4を対象バージョンとしている。

▼テキスト公開URL
http://www.ashisuto.co.jp/product/category/oss/libreoffice/download/

CONTACT ㈱アシスト
URL http://www.ashisuto.co.jp/

Hardware

NEC、アーカイブストレージ「iStorage HS6」発売

NECは5月7日、「iStorage シリーズ」のラインナップを強化し、大容量データを安全に長期保管するアーカイブ専用モデル「iStorage HS6」を発売した。ビッグデータ活用を背景に、増え続ける大容量データの継続的な保管に最適だ。新製品の特長として、次の3点がある。

- 48TB～7.9PB（ペタバイト）まで増設可能なストレージにより、柔軟な拡張性と安全な長期保管を実現
- 分散冗長配置技術により、HDDの故障時にもデータの復元が可能。暗号化技術・改ざん防止機能も装備
- NEC独自の画像圧縮エンジン「StarPixel」により画像データを高速・高効率に圧縮・復元

▲iStorage HS6

価格（税別）は、ベースノードが530万円～、増設用ハイブリッドノードが530万円～、増設用ストレージノードが350万円～となっており、出荷開始日はいずれも6月30日。

CONTACT 日本電気㈱
URL http://jpn.nec.com/

Service

リンク、「ベアメタル型アプリプラットフォーム」の提供を開始

㈱リンクは、高負荷サイトやスマホアプリ、ゲーム事業者向けサーバサービスとして展開している専用サーバパッケージ「アプリプラットフォーム」を「ベアメタル型アプリプラットフォーム」として、5月28日より提供を開始した。

2010年から展開されているアプリプラットフォームは、NAND型フラッシュメモリ「ioDrive2」を搭載したサーバ・回線・その他機器をパッケージ化した、初期費用0円で利用できるホスティングサービス。今回提供開始する新サービスはその後継サービスであり、仮想サーバを利用する感覚で、物理サーバを利用できるものだ。ユーザはコントロールパネルから物理サーバを作成／削除でき、追加も最短20分で行える。OSまたはハイパーバイザ（KVM）をコントロールパネルからリモートインストールでき、仮想サーバでテスト、物理サーバで本番運用するなどの使い方もできる。テンプレートからのサーバ作成やスナップショットといったクラウドならではの機能も利用可能だ。

価格は、初期費用0円、月間利用量14,800円～となっている。

CONTACT ㈱リンク
URL http://www.link.co.jp/

Dで SF

小飼弾

第7回

『ディアスポラ』（グレッグ・イーガン／早川書房）

「数学は科学の女王」とのたまったのはガウスだそうですが、だとしたらScience Fictionの女王もまた数学的フィクションということになるでしょう。数学的フィクション――「数学ガール」シリーズ（結城浩）――確かに数学「を」語るフィクションではあるけれど、数学「で」語るフィクションとは言えませんよね。数学「で」語るフィクションなんてあるのでしょうか――あります。そう。SFならね。

それが、グレッグ・イーガンの諸作品。『順列都市』なんて、タイトルからしてもう数学の臭いがプンプンしてきて、数学が苦手な人は鼻とつむじをまげてしまいそうですが、しかしイーガンが数学者ではなく作家であるのは、話の骨格はとっても数学的なのに、肉付けが文字通り肉感的なところ。

ここでは『ディアスポラ』を取り上げましょうか。話の始まりは西暦30世紀頃。大多数の人は「ポリス」という「データセンター」の仮想インスタンスとして不老不死の存在となった一方で、我々と同様に血と肉でできた肉体を持つ[肉体人（fleshers）]も残っている時代。主人公ヤチマがポリス自身によって生み出されるところからはじまります。本作は「人間」の親を持たない「孤児」である彼（v）／女（e）の個人史であると同時に地球史であり宇宙史でもあるのですが、とあるきっかけで地球はDNAとタンパク質で駆動する生物が生存できない環境になってしまいます。この事件についた呼び名が肉滅（carnevale）。こんな肉々しい地球滅亡劇がありますか？

しかし本作のすごいのはここから。仮想インスタンスだけあって、肉滅をあっさり生き延びたポリス人たちは単に不老不死であるにとどまらず、簡単にコピーがとれるわけですが、彼らはポリスごとコピーを取ったうえでそのコピーを宇宙にばらまくことに決めました。いつか地球外生命体と出会うことを夢見て。これが本書のタイトルともなった放散（ディアスポラ）なのですが、あ、ありのまま今起ったことを話すぜ。ち……地球外生命を見つけたと思ったら宇宙外生命だった。頭がどうにかなりそうだった……。

もっと恐ろしいものの片鱗は本書で存分に味わっていただくとして、個人的に印象的だったのは、肉滅の難民オーランドのトイレシーン。元肉体人だけあって仮想肉体も持っているのですが、便意は削除しても尿意はわざと残してるってのが。排便感と排尿感の甲乙つけがたい私としては「オーランドのくそったれ！」と叫びたい気分ですが、クソまじめなようで結構笑いのツボもついてきます。肉滅前にご一読を！SD

題字・イラスト／aico

ひみつのLinux通信

作)くつなりょうすけ
@ryosuke927

第7回 コマンドヒストリに時を刻め

五月病を克服したあとに「夏休み恋しい病」を発症したくつな先生に愛のボーナス（ツイートとかメールとか）募集中！
◆先輩に成敗してほしいネタもね！

to be Continued

sudoを使うと実行ユーザとコマンドがログに残るのでメンテナンス履歴として重宝しますが、sudo bashで管理者権限になると以降のコマンドがログに記録されずにシェルヒストリに記録が残ります。複数人で管理する際にその様な使い方をする人がいると困ることになります。その対策として、標準ではコマンドヒストリには記録されない実行時刻を出力する設定にすることをお勧めします。bashならHISTTIMEFORMAT="%Y-%m-%d %H:%m:%S "などを設定するとhistoryコマンドに実行時刻を表示する記録をします。HISTSIZEも大きくすればモアベターです。一番楽なのはメンテナがsudoでシェルを起動しないよう設定・教育することですが……。

Letters from Readers

そういえば、ワールドカップですが……

毎年5月と6月はIT系のイベントが多いです。編集部からも企画のネタ探しを兼ねていくつか取材に行くのですが、新しい技術やサービスを目にするのは、心躍ります。本誌が発売されるころには、FIFAワールドカップが始まっているのでしょうが、きっと編集部内では、Google I/Oのほうが気になっているのだろうなあ。

2014年5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　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

あきみちさんとaicoさんのコラボレーションによるインターネットのしくみ講座をお届けしました。

今までなんとなくしかわかっていなかったところが理解できたので、とてもよかった。イラストが小悪魔さんでわかりやすく読みやすかった。サンプルコードがCで掲載されていて、細かい実装説明もあったのでとてもよかった。

東京都／山内さん

「投げっぱなしジャーマンスープレックス」の話は必要だったのでしょうか（笑）

神奈川県／epoxyさん

CCNAを保有していたころは知ってて当たり前だったのに、今はあいまいになっていた。いい復習になった。そして、aicoさんのイラストがかわい過ぎる。

東京都／binaさん

「投げっぱなしジャーマンスープレックス」のたとえに反応した読者が異様に多くてびっくりしました。ちなみに、担当はネットワーク輻輳状態を表現した、うさぎさんがオラオラやかめはめ波をしている絵が気に入っています。

第2特集　UNIX必須コマンドトレーニング

連載漫画「ひみつのLinux通信」のくつなりょうすけさんに、新人にまず知っておいてほしい実用的なコマンドを紹介していただきました。

コマンドとかも普段なにげに使っていますが、確認のために読んでみると、非常に勉強になります。

京都府／クラウドマンさん

よく忘れるコマンドなどが記載されていてとても良いです。

東京都／tekitoizmさん

オプションを解説しないで、「これだけ打てばこの作業ができる」というようにコードを教えるのは、実使用上では若干いただけない印象がある。

東京都／TH.Lさん

今回は、「よく使うコマンドをオプションとセットで覚えてほしい」「まずはそのまま入力して試してほしい」という意図から、逆引きのクックブック形式にしてみました。いかがでしたでしょうか?

短期集中連載　Rettyのサービス拡大を支えた“たたき上げDevOps”

スタートアップ企業が急速な成長をするとき、インフラ担当者はどう対応すべきか。「Retty」の事例を紹介しました。

本連載の今後に期待しています。とくに専門家が構築したわけでないシステムをあとから参画して、どうやってDevOpsで改善していったのかが興味あります。

千葉県／今井さん

DevOpsの内容もだが、単純にいちエンジニアの話としておもしろかった。

東京都／藤田さん

RettyのサービスはAWS上に構築されていますが、読者にもAWSユーザは多いようで「参考になった」という声が多かったです。

短期集中連載　さらに踏み込む、Mac OS Xと仮想デスクトップ

本連載も最終回。これまでの内容を総動員して、ハイブリッドなデスクトップ環境の構築方法を紹介しました。

自分がMacBookやMacBook Proに興味があるのでためになった。

東京都／サバ鼻炎さん

Mac OS Xの見えないところが見えておもしろかったです。

東京都／吉田さん

複雑で高度な知識が必要ですが、Macユーザには一度試してもらいたいですね。構築するだけでもさまざまなノウハウが身につきそうです。

短期集中連載　Web標準技術で行うWebアプリのパフォーマンス改善

HTML5というとリッチなコンテンツを作る手段として扱われがちですが、少し視点を変えてパフォーマンス改善の面からHTML5などの技術を紹介しました。

大規模データをWebアプリで閲覧するようなしくみを考えています。今後のためにも、こういうユーザレベルQoSを重視した記事については、さらに深く掘り下げてもらえるとありがたいです。

愛知県／NGC2068さん

昔ながらのダイヤルアップの感覚で使っていると、先読みするというのは、確かに思いつきにくい気がする。最近の技術が知れておもしろかった。

北海道／村橋さん

HTML5は見た目だけでなく、結構、実用的な機能があることがわかってもらえたのではないでしょうか。

連載

「温故知新　ITむかしばなし」がいつまで続くのか、とても気になります。頑張ってほしい!

神奈川県／とーふやさん

本連載は3名ほどの著者が入れ替わり執筆しています。きっとネタが尽きるまで続くでしょう。長く続けるためにも、応援をよろしくお願いします。

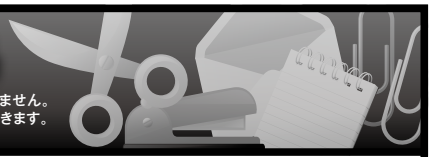
「セキュリティ実践の基本定石」で諜報分野における「オープンソース」の意味を知ってびっくり。専門用語は難しいですね。私は、以前からすずきひろのぶさんが「ハッカー」と「クラッカー」を区別していることに賛成しているのですが、社会的には何でも「ハッカー」という用語を使用しているのが歯がゆいです。

東京都／psiさん

「ライフハック」などの用語のおかげで、以前よりは「ハッカー」の誤用が減ってきているように思います。ですが、テレビの報道などを見ていると、「まだまだだなあ」と思うこともありますね。

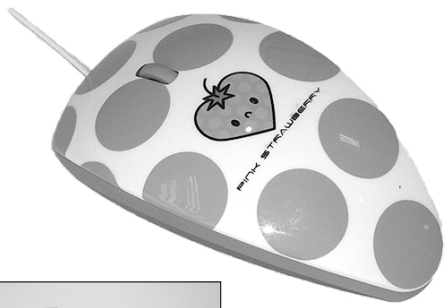
エンジニアの能率を高める一品

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。



mouse fit

3,240円（税込）／㈱アートファクトリー　http://www.art-factory.ne.jp/



◀写真1
マウス裏面は、体のツボをマッサージしやすいよう凸状のでっぱりがある。

今回紹介するのはキュートな柄の光学マウス。しかも単なるマウスではありません。マウス側面のボタンをONするとバイブレータが起動。ブブブブ……と振動した状態のまま自分の肩にあてがえば、肩をマッサージできるのです（写真1、2）。もちろん肩以外に首でも、太ももでもお好きなところをマッサージしてください。側面ボタンを何回か押せば、振動パターンを変えられます。バイブレータは側面ボタンを長押しすることでONになるので、マウス使用中に振動が起きるなんてことはありません。小型ながら振動はなかなか強力で、手にしているだけで、思わず目が覚めるという二次効果もありそう。本当に癒されたいときは、本製品を友人などに持ってもらって自分の肩にあてがってもらうのが一番気持ちよさそうです。仕事の休憩時間に仲間同士で代わりばんこにマッサージしあうっていうのはどうでしょうか？　（読者プレゼントあります。P.16参照）

▲写真2　マッサージ使用例

5月号のプレゼント当選者は、次の皆さまです

①サクス・タブレット7 ……東京都　菅原　剛様
②現代用語の基礎知識1995～2014 ……神奈川県　三輪時弘様

★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
August 2014

2014年8月号
定価（本体1,220円+税）
176ページ

7月18日発売

［第1特集］身近なシステムログからWebやデータベース、そしてビッグデータの基礎まで

ログを読む技術

コンピュータに残ったログは宝の山！ 故障の原因追求からユーザ動向までまるわかり!

そもそもログはどこにたまる？／syslogを調べてみよう!／WebサーバApacheとNginxの場合／MySQLのログ分析／ホスティング現場のログ管理ノウハウとは？／Fluentd＋MongoDBで小さくはじめるログ活用

［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

Linuxのしくみを深いレベルから理解すると、さまざまな局面でエンジニアとして実力を試すことができます。カーネルが読めるようになると世界が開けます!

■特別企画

・ITビジネスの足下を揺るがす大きなバグ「OpenSSLの脆弱性 “Heartbleed” の教訓（後編）」

・ボンディングによる高速NASはどこまで速くなる？

※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。

休載のお知らせ

「ハイパーバイザの作り方」（第21回）は都合によりお休みいたします。

お詫びと訂正

以下の記事に誤りがございました。読者のみなさま、および関係者の方々にご迷惑をおかけしたことをお詫び申し上げます。

■2014年6月号　連載「シェルスクリプトではじめるAWS入門」第3回

●P.136　左段・~/.aws/configの行順に間違いがありました。正しくは次のようになります。

```
[default]
aws_access_key_id=AKIAIOSFODNN7EXAMPLE
```

SD Staff Room

●年度末の機材更新にともない、余ったディスプレイで大きめのものを拝借したら、仕事しやすい！ こんなにも作業効率に影響を与えるとは思いもよらず。4年もののMBPもSSD化＋メモリ増設で今どきのマシンレベルにしたらストレス減ってうれしくなった。道具にお金かけるのはライフハックだね。（本）

●既に今年も半分過ぎようとしている。半年間のトピックスは、年始の健康診断で2,000kcal/日以下にするように言われ、それを守れてきたことだろうか。おかげさまで年始の最高体重ー現在体重は12kg。平日歩きと休日チャリも効果があったようだ。しかし、ズボンの買い換えで思わぬ出費が……。（幕）

●娘のピアノの発表会に息子と自転車で行きました。会場付近の駐輪場にはGoogle検索＆Mapsで迷わずに到着。あとは歩いて5、6分。着いたも同然……ん、あれ？ あわててGMapsを開くもGPSをロスト……なんとかたどり着けましたが、家族の信用を回復するには時間がかかりそう。（キ）

●LinuxConの取材に行きました。Linus氏はゲーム、アジャイル、TDD、仮想化にはあまり興味がなくて、組込みやハードウェアが好きだそう。「なんだ、おじさんじゃん」なんて口が裂けても言えません。「ワタシハ リナックス チョットデキル」Tシャツをゲットしたので、次号のプレゼントにします。（よし）

●電気製品やパソコン書を買いに、よく秋葉原へ行きます。お店の数が多いので、大型量販店で見つからなかったものが、小さな個人店で見つかるといったことが多々あります。ECサイトを使うと数秒で済むことも、道草しながら時間をかけて行うと、やりがいのある趣味に感じられて楽しいですね。（な）

●新生活を始めてから半年が経ち、ようやく生活のリズムも定着してきたので、前からやろうと思っていた家庭菜園をしようと構想中。引っ越し当初は毎朝のようにカラスが遊びに来ていたけど、最近はほとんど来なくなったので、今がチャンスかな？ とりあえず、休日にベランダの片づけから始める予定です。（ま）

ご案内

編集部へのニュースリリース、各種案内などがございましたら、下記宛までお送りくださいますようお願いいたします。

Software Design 編集部
ニュース担当係

［FAX］
03-3513-6173

※FAX番号は変更される可能性もありますので、ご確認のうえご利用ください。

［E-mail］
sd@gihyo.co.jp

Software Design
2014年7月号

発行日
2014年7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。