Special Feature 1 決定版Docker
Special Feature 2 SNMPの本格活用

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

Software Design

2015年12月18日発行
毎月1回18日発行
通巻368号
（発刊302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

2015 December 12

Special Feature 1

決定版 Docker 自由自在

実用期に入った
Linuxコンテナ技術

HashiCorp
Docker Machine
Docker Swarm
Docker API
Docker Trusted Registry
Vagrant
Packer
Terraform
Serf
Consul
Vault
Atlas

最新情報をキャッチアップ！

Special Feature 2

ネットワーク・
システム管理はおまかせ！

SNMPの教科書

基礎からわかる・
すぐ応用できる

Extra Feature

短期集中連載

クラウド時代の負荷試験再入門

スマホ・Web開発環境の徹底改善

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。

2015
December
12
Software Design

第1特集

決定版 Docker 自由自在

実用期に入ったLinuxコンテナ技術

contents

前佛 雅人 017

第1章 Dockerのキホン 018
環境構築から基本操作をマスターしよう

第2章 Dockerのイメージ管理と基本操作 027
レジストリとDockerクライアントの操作法

第3章 Dockerの操作と管理 035
便利なコマンド／オプションを一挙に学ぼう

第4章 Docker環境を自動構築 044
オーケストレーションツール「Docker Machine」

第5章 Docker Swarmでコンテナのスケジューリング 051
コンテナをクラスタリングしリソースを管理

第6章 Docker環境のコード化とオーケストレーション 059
DockerfileとDocker Composeで作業の省力化・効率化

第7章 HashiCorpの自動化ツールとDockerの連携 064
HashiCorp道の真髄

2015
December
12
Software Design

contents

第2特集

ネットワーク・システム管理の定石

SNMPの教科書

堅実な監視で障害をキャッチ　本文:山下 薫　コラム:馬場 俊彰　067
セクション1　SNMPはなぜ必要なのか　068
セクション2　SNMP入門編　070
セクション3　SNMP実用編　077
セクション4　SNMP応用編　084
セクション5　SNMPの関連技術　091
セクション6　まとめ　093
コラム1　SNMPの動作形態　073
コラム2　SNMPのデータモデル　075
コラム3　CentOSでの利用方法　080
コラム4　SNMPのバージョン　083

短期連載

クラウド時代のWebサービス負荷試験再入門[新連載]
クラウド時代における負荷試験とは何か　仲川 樽八　096
SMB実装をめぐる冒険[2]
File System for Windowsの作り方　田中 洋一郎　106

Catch up trend

ConoHaで始めるクラウド開発入門[5]
OpenStack APIを使ったCLI操作をConoHaでやってみる　郷古 直仁　186
迷えるマネージャのためのプロジェクト管理ツール再入門[10]
テストにもっと光を! 言うは易く行うは難し。テスト工程を改善しよう!　廣田 隆之、網野 勉、大塚 和彦　190

アラカルト

ITエンジニア必須の最新用語解説[84] Kinetic Open Storage Project　杉山 貴章　ED-1
読者プレゼントのお知らせ　016
SD BOOK REVIEW　095
バックナンバーのお知らせ　161
SD NEWS & PRODUCTS　194
Readers' Voice　198

Column

digital gadget [204]
コンピュータグラフィックスの祭典SIGGRAPH 2015[後編] 安藤 幸央 001

結城浩の再発見の発想法 [31]
Library 結城 浩 004

[増井ラボノート]コロンブス日和 [2]
Gyump 増井 俊之 006

軽酔対談　かまぷの部屋 [17]
ゲスト:吉岡 弘隆さん 鎌田 広子 010

ツボイのなんでもネットにつなげちまえ道場 [6]
SPIで通信してみる 坪井 義浩 012

Hack For Japan〜エンジニアだからこそできる復興への一歩 [48]
Connect 2015 in Koriyama, with UDC 及川 卓也、清水 俊之介 180

温故知新 ITむかしばなし [49]
シリアル通信〜高速モデムとホストマシンとの接続〜 速水 祐 184

ひみつのLinux通信 [23]
数字が見える! くつなりょうすけ 197

Development

Vimの細道 [3]
VimでJavaを使う(補助プラグイン編) mattn 114

るびきち流Emacs超入門 [20]
標準機能から「yasnippet」まで　Emacsの入力支援 るびきち 120

書いて覚えるSwift入門 [10]
例外を避ける? 小飼 弾 124

Erlangで学ぶ並行プログラミング [9]
プロセス辞書とETSによる暗黙のデータ共有 力武 健次 128

セキュリティ実践の基本定石 [27]
同じ轍を踏まないために。IoT時代に向けてできること すずきひろのぶ 135

Sphinxで始めるドキュメント作成術 [9]
ドキュメントに図を入れよう——さまざまなグラフィックツールとの連携 小宮 健、清水川 貴之 140

Mackerelではじめるサーバ管理 [10]
Mackerelの監視ルールをコードで管理しよう 田中 慎司 146

OS/Network

Red Hat Enterprise Linuxを極める・使いこなすヒント .SPECS [17]
Identity Managementを使おう 藤田 稜 152

Be familiar with FreeBSD〜チャーリー・ルートからの手紙 [26]
bhyveでOpenBSDファイアウォール on FreeBSDを構築 後藤 大地 156

Debian Hot Topics [30]
DebConf15レポート(前編)と最新トピック やまねひでき 162

Ubuntu Monthly Report [68]
UbuntuとSkylake 柴田 充也 166

Linuxカーネル観光ガイド [45]
Linux 4.1の新機能〜ext4の暗号化機能 青田 直大 172

Monthly News from jus [50]
リングに再び帰るLLイベント 法林 浩之 178

[ロゴデザイン]
デザイン集合ゼブラ+坂井 哲也
[表紙デザイン]
藤井 耕志(Re:D)
[表紙写真]
Life On White /gettyimages
[イラスト]
フクモトミホ
[本文デザイン]
*岩井 栄子
*近藤 しのぶ
*SeaGrape
*安達 恵美子
*轟木 亜紀子、阿保 裕美、佐藤 みどり
(トップスタジオデザイン室)
*伊勢 歩、横山 慎昌(BUCH+)
*森井 一三
*藤井 耕志(Re:D)
*石田 昌治(マップス)

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。

BOOKS INFOMATION

エンジニアとして世界の最前線で働く選択肢

渡米・面接・転職・キャリアアップ・
レイオフ対策までの実践ガイド

●竜盛博　著

A5判／216ページ
定価（本体1880円+税）

ISBN978-4-7741-7656-7

世界のソフトウェア開発の本場で働けたら—
エンジニアならば一度は考える可能性を実現するにはどうすればいいか？
アメリカで職を得るために必要なこと、レジュメを書くときの注意点、面接官の前で実際にコーディングをする面接を突破するためのコツ、日本との仕事環境の違い、転職やレイオフとの向き合い方までを具体的に教えます。
シリコンバレーやシアトルで計15年、従業員数十人のスタートアップでも10万人以上の大企業でも働き、面接する側も数多く経験した著者だからこそ書けるリアルが満載。

ITエンジニアのための機械学習理論入門

機械学習のしくみを学び
データサイエンスの
本質を理解する

●中井悦司　著

A5判／256ページ
定価（本体2580円+税）

ISBN978-4-7741-7698-7

現在話題となっている機械学習（Machine Learning）のツールやライブラリは内部でどのような計算をしているのか？計算で得られた結果にはどのような意味があり、どのようにビジネス活用すればよいのか？
—という疑問を持つエンジニアが増えています。本書は機械学習理論を数学的な背景からしっかりと解説をしていきます。そしてPythonによるサンプルプログラムを実行することにより、その結果を見ることで機械学習を支える理論を実感できるようになります。

Python ライブラリ厳選レシピ

●池内孝啓、他　著

A5判／320ページ
定価（本体2880円+税）

ISBN978-4-7741-7707-6

Pythonには、豊富な標準あるいはサードパーティライブラリ・モジュールがあります。ライブラリに関するドキュメントも充実していますが、初心者にとってはそのボリュームゆえに、まずどんなライブラリを活用できればよいのか、わかりづらい側面があります。本書では、「これだけは押さえておきたい」ライブラリとその機能を、標準ライブラリだけでなくサードパーティのものも交え、カテゴリごとに、その活用法を紹介します。バージョンはPython 3.4をメインとします。

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

Pocket Reference

小さくても、中身充実！

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

鶴長鎮一／馬場俊彰 著
四六判／400ページ
定価（本体2780円+税）
ISBN978-4-7741-7633-8

宮前竜也　著
四六判／224ページ
定価（本体2180円+税）
ISBN978-4-7741-7740-3

沓名亮典 著
四六判／608ページ
定価（本体2380円+税）
ISBN978-4-7741-7404-4

石田つばさ 著
四六判／448ページ
定価（本体2180円+税）
ISBN978-4-7741-4836-6

土井毅／高江賢／飯島聡／高尾哲朗 著　山田祥寛 監修
四六判／488ページ
定価（本体2580円+税）
ISBN978-4-7741-4948-6

高橋晶／安藤敏彦／一戸優介／楠田真矢／道化師／湯朝剛介 著
四六判／544ページ
定価（本体2980円+税）
ISBN978-4-7741-7408-2

朝井淳 著
四六判／640ページ
定価（本体1980円+税）
ISBN978-4-7741-3835-0

高江賢 著　山田祥寛 監修
四六判／536ページ
定価（本体2580円+税）
ISBN978-4-7741-4592-1

大垣靖男 著
四六判／648ページ
定価（本体2580円+税）
ISBN978-4-7741-7229-3

岡本隆史／武田健太郎／相良幸範 著
四六判／272ページ
定価（本体2480円+税）
ISBN978-4-7741-5184-7

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

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]サーバ/インフラエンジニア養成読本　管理/監視編
養成読本編集部 編
定価 1,980円+税　ISBN 978-4-7741-6424-3

[改訂新版]サーバ/インフラエンジニア養成読本　仮想化活用編
養成読本編集部 編
定価 1,980円+税　ISBN 978-4-7741-6425-0

[改訂新版]サーバ/インフラエンジニア養成読本
養成読本編集部 編
定価 1,980円+税　ISBN 978-4-7741-6422-9

iOSアプリエンジニア養成読本
髙橋俊光、諏訪悠紀、湯村翼、平屋真吾、平井祐樹 著
定価 1,980円+税　ISBN 978-4-7741-6385-7

[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

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

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

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

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

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

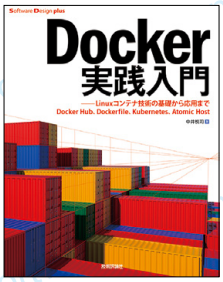
データベースエンジニア養成読本
データベースエンジニア養成読本編集部 編
定価 1,980円+税　ISBN 978-4-7741-5806-8

小銅弾のコードなエッセイ
小銅 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

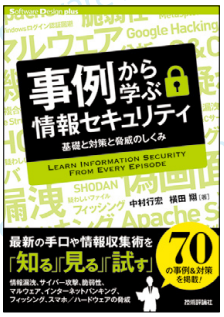
サウンドプログラミング入門
青木 直史 著
定価 2,980円+税　ISBN 978-4-7741-5522-7



中井悦司 著
B5変形判・200ページ
定価 2,680円(本体)+税
ISBN 978-4-7741-7654-3

養成読本編集部 編
B5判・192ページ
定価 2,280円(本体)+税
ISBN 978-4-7741-7631-4

岩永信之、山田祥寛、井上章、伊藤伸裕、熊家賢治、神原淳史 著
B5判・128ページ
定価 1,980円(本体)+税
ISBN 978-4-7741-7607-9

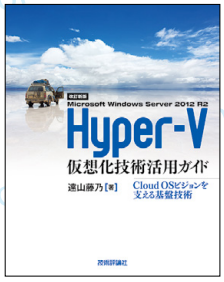


中村行宏、横田翔 著
A5判・320ページ
定価 2,480円(本体)+税
ISBN 978-4-7741-7114-2

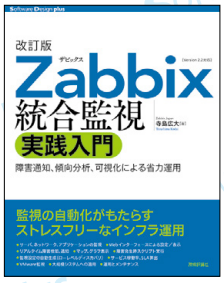
川本安武 著
A5判・400ページ
定価 2,980円(本体)+税
ISBN 978-4-7741-6807-4

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

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



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

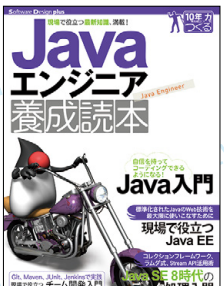


寺島広大 著
B5変形判・440ページ
定価 3,500円(本体)+税
ISBN 978-4-7741-6543-1

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

養成読本編集部 編
B5判・128ページ
定価 1,980円(本体)+税
ISBN 978-4-7741-7320-7

養成読本編集部 編
B5判・164ページ
定価 1,980円(本体)+税
ISBN 978-4-7741-7057-2

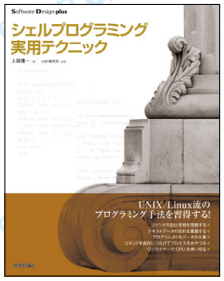


養成読本編集部 編
B5判・168ページ
定価 1,980円(本体)+税
ISBN 978-4-7741-6931-6

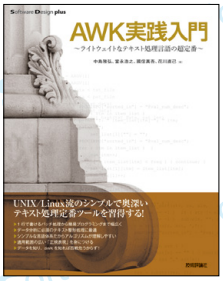
吾郷協、山田順久、竹馬光太郎、和智大二郎 著
B5判・136ページ
定価 1,980円(本体)+税
ISBN 978-4-7741-6797-8

技術評論社

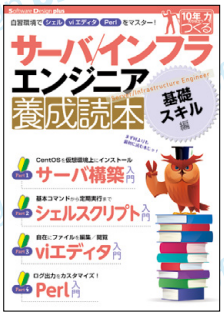
養成読本編集部 編
B5判・112ページ
定価 1,980円(本体)+税
ISBN 978-4-7741-7441-9



上田隆一 著
USP研究所 監修
B5変形判・416ページ
定価 2,980円(本体)+税
ISBN 978-4-7741-7344-3



中島雅弘、富永浩之、
國信真吾、花川直己 著
B5変形判・416ページ
定価 2,980円(本体)+税
ISBN 978-4-7741-7369-6



福田和宏、中村文則、
竹本浩、木本裕紀 著
B5判・128ページ
定価 1,980円(本体)+税
ISBN 978-4-7741-7345-0

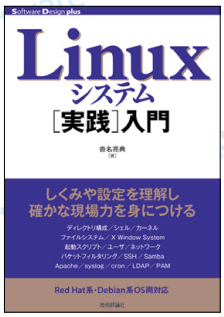
川瀬裕久、古川文生、
松尾大、竹澤有貴、
小山哲志、新原雅司 著
B5判・156ページ
定価 1,980円(本体)+税
ISBN 978-4-7741-7313-9

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

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

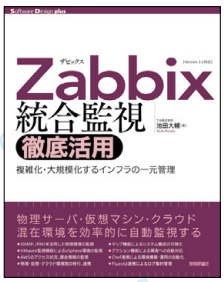
久保田光則、アシアル㈱ 著
A5判・384ページ
定価 2,880円(本体)+税
ISBN 978-4-7741-6211-9

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



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

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

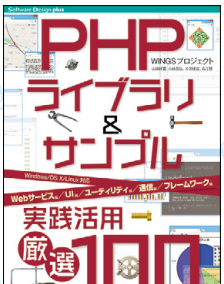
沼田哲史 著
B5変形判・360ページ
定価 3,200円(本体)+税
ISBN 978-4-7741-6076-4

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

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

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

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



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

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

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

紙面版
A4判・16頁
オールカラー

電脳会議

DENNOUKAIGI

一切無料

新規購読会員受付中!

『電脳会議』は情報の宝庫、世の中の動きに遅れるな!

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

今が旬の情報満載!

新規送付のお申し込みは…

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

電脳会議事務局　検索

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

一切無料!

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

毎号、厳選ブックガイドも付いてくる!!

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

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

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）

第84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。

Kinetic Open Storage Project

Kinetic Open Storage Project とは

Linux Foundationは2015年のLinuxConにおいて、大手ハードディスクベンダーやネットワーク機器ベンダーなどと共同で「Kinetic Open Storage Project」と呼ばれる新しいプロジェクトを開始することを発表しました。Kinetic Open Storageは、ストレージサーバによる管理なしに、アプリケーションがイーサネット経由で直接ディスクドライブやSSDにアクセスできるようにするしくみです。同プロジェクトでは、この技術のオープン化と対応するディスクドライブの開発を推進することを目的として、具体的なAPIやプロトコルの策定を行うとのことです。

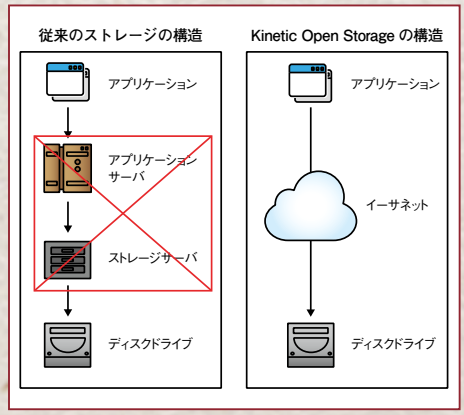
Kinetic Open Storageでは、ディスクドライブ自身がIPアドレスを持つことで、中間にWebサーバやストレージ管理サーバを置かず、アプリケーションから直接イーサネット接続してデータの格納や参照ができます。データの保管はキーバリューストア方式で行われ、アプリケーションからはAPI（Kinetic API）経由でデータを操作するしくみになっています。

従来のオブジェクトストレージでは、ディスクドライブとアプリケーションの間に、キーバリューストアを実現するためのストレージサーバや、アプリケーションとの仲介を行うアプリケーションサーバなどのレイヤーが必要でした。この構造はアプリケーションからストレージ内部の実装を隠蔽できるメリットがある一方で、中間レイヤーのオーバーヘッドによる性能劣化や、管理するサーバが増えることによる運用コストの増加などといったデメリットがありました。

Kinetic Open Storageではこれらの中間レイヤーが省略できることから、次のようなメリットが得られるとのことです。

- システム全体の構造が簡略化できる
- ストレージをコンピューティングから分離できる
- ラックの密度を高められる
- 冷却のためのファンを最小化できる
- データセンターの構築や運用管理のコストを削減できる

▼図1　イーサネット接続でのストレージシステム



誕生の背景

Kinetic Open Storageは、もともとはSeagate社が開発を進めていた技術です。その誕生の背景には、大規模なデータ駆動型アプリケーションが急増したことによって、ストレージインフラに求められる性質が変わってきたという事情があります。保管するデータの種類が大きく変化していることに加えて、容量とコスト、そして速度に対する要求は増し続けており、従来の枠組みのままでは対応が追いつかなくなりつつあります。

そこで非効率性の原因となっている中間のレイヤーを取り除き、モダンなエンタープライズアプリケーションの要求に最適化した新しいしくみとして生み出されたのが、Kinetic Open Storageというわけです。中間レイヤーを省略して必要なリソースにダイレクトにアクセスするというコンセプトは「サーバレスアーキテクチャ」と呼ばれており、ストレージ以外の分野でも普及しつつあるものです。

Seagateは当初OpenStackのSwiftおよびAWS S3互換のRiak CS向けにAPIをリリースし、それと並んでオープン化やサードパーティへの普及を推進してきました。今回のプロジェクト発足はその延長にあるもので、東芝やWestern Digitalといった競合大手が参加しているほか、ネットワークベンダーやOSベンダーなどからも広く支持されていることがポイントです。ストレージの重要性がますます大きくなっていることから、プロジェクトが順調に進めば、Kinetic Open Storageは広く普及する可能性が高いと言えるでしょう。SD

Kinetic Open Storage Project
http://www.openkinetic.org/index.php

DIGITAL GADGET vol.204

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

コンピュータグラフィックスの祭典SIGGRAPH 2015[後編]
映画の都ロサンゼルス。映像技術編

SIGGRAPH 基調講演と表彰から

コンピュータグラフィックスとインタラクティブ技術に関する世界最大の学会・展示会である、第42回SIGGRAPH 2015が8月9日から13日の5日間、米国ロサンゼルスで開催されました。先月号に続いて、デジタルガジェット視点でレポートをお届けします。

今年のSIGGRAPHの基調講演は、マサチューセッツ工科大学(MIT)メディアラボの所長、伊藤譲一氏でした。例年、コンピュータグラフィックス直球ではなく、その周りの専門家を呼んでお話ししてもらうことで、参加者に示唆を与えたり、今後の指針となるような話題の人物が選定されています。過去には、著名なゲーム作家であるウィル・ライト氏や、コンセプトアートのシド・ミード氏、SF作家のブルース・スターリング氏らが講演されています。

伊藤氏の講演は、前メディアラボ所長のニコラス・ネグロポンテ氏の言葉を借り、「Bio is the new digital」を前面に打ち出したものでした。過去を振り返ると、インターネット登場以前と以後では世の中や社会のしくみが大きく変わってきました。バイオ技術の進化と、デジタル技術との組み合わせやその応用によって、今までにはなかったモノが作れたり、見たことのないアートの世界にバイオ技術が活用されたりした事例が紹介されました。バイオ技術とデジタル技術の組み合わせによって、インターネットによる革新と同じくらい新しいことが起きつつあるということを、濃縮された情報とともに伝えた60分の講演でした。

また、今年のSIGGRAPHの表彰では、CG映像制作には不可欠なBSP Treeというアルゴリズムの考案や、黎明期のVR(バーチャルリアリティ)技術発展への貢献が認められ、Henry Fuchs氏がクーンズ賞を獲得しました。Fuchs氏はBSP Tree (Binary Space Partitioning:バイナリ空間分割)と呼ばれる、三次元空間内に描こうとしている物体が存在するかどうかを再帰的に分割していく手法で知られています。BSP Treeは現在でも、ゲームやロボット工学における衝突判定には欠かせない重要な考え方です。

そのほか、アーティストのLillian Schwartz氏が、長年のデジタルアートコミュニティへの貢献が認められ、優秀アーティスト賞を受賞しました。Schwartz氏は1980年頃からデジタル技術を活かした、2D、3Dの新たな表現方法を切り開いた作家として知られています。いくつかの作品はMOMAへの永久所蔵品として収められています。

また今年はちょうど、さまざまな

SIGGRAPH会場内、VR展示の様子

SIGGRAPH会場内、さまざまな機器を試用、試作できるスタジオ展示の様子

静止した車に乗りながら、ヘッドマウントディスプレイを装着し、VRを楽しむ様子

バイオとデジタル技術をテーマにした伊藤譲一氏の基調講演

※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います。

映画の特殊効果を手がけるILM（Industrial Light & Magic）が40周年を迎え、そのお祝いとなる記念セッションも開催されました。深海に住む未知の生命体を描いた映画「アビス」に始まり、現在作成中の最新作の「スター・ウォーズ」まで、さまざまな映画を振り返りながら、CG/VFXの歴史を振り返るセッションでした。また、最新の取り組みであるILMの研究所ILMxLABによる、VRを活用した撮影環境の話題が紹介されました。

技術の進化と映像の進化

今年のSIGGRAPHでの注目の話題は、パノラマ映像と安価なHMD（ヘッドマウントディスプレイ）によるVRコンテンツの浸透です。VR（バーチャルリアリティ）は技術や機材的にも研究開発的にもSIGGRAPHが得意とする領域ですが、過去にも何度かブームがあり、盛り上がっては盛り下がっていった中で、近年のVRブームは一般ユーザ向けのため、今度こそ一大ブームになると意気込む人々が増えてきている印象です。

日本でも個人によるさまざまな試作コンテンツ、広告キャンペーンなどでVRが活用されていますが、欧米では中規模のゲーム制作規模の予算がついたり、アーティストのプロモーションやミュージックビデオでの活用、ハリウッド映画を制作していたような映像チームがパノラマ映像を撮影しはじめたりなど、産業として広がりつつあることが見て取れます。

Side Effectsブースでの HELP制作のメイキング

Googleが進めるパノラマ映像コンテンツ視聴用アプリ「Google Spotlight Stories」（iOS/Android）では、5分の短編「HELP」が公開。これはイギリスの大手映像プロダクションThe Mill制作によるもので、監督はワイルド・スピードシリーズを手がけるJustin Lin氏です。6K解像度のRED EPIC DRAGONカメラ4台で同時撮影し、リアルタイムプレビューしながらパノラマ撮影するという、スタッフ構成も機材的にも現在考えうる最高の組み合わせで作られたパノラマ映像です。

Googleが攻める パノラマ映像機材と パノラマ映像ソリューション: Google JUMP

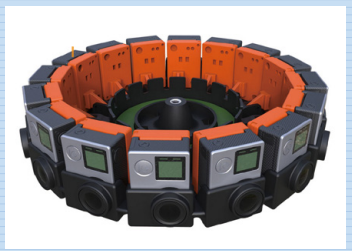
Googleはハイエンドからローエンドまでパノラマ映像の展開を網羅し始めており、安価なものでは、ダンボールで作られたCardboardを展開し、サードパーティ製のものも含め、世界で100万人のユーザがCardboardアプリを使い始めているそう。会場ではGoogle JUMPと呼ばれる、アクションカムのGoPro HERO4を16台つなぎ合わせた力技的なパノラマ撮影リグ（台座）もお披露目されていました。現在撮影プロジェクトを募集中で、企画が通った撮影案にはJUMP機材が貸与されるそうです。

VulkanというAPIの浸透、進化

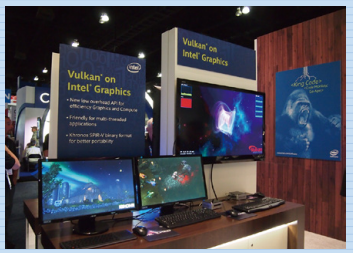
現在、リアルタイムで描画するコンピュータグラフィックスの世界はOpenGLとDirectXが主流ですが、アーキテクチャ的には古くなりつつあり、最新のハードウェアを最大限に活かしきれなくなってきています。そういった声を反映して、VulkanというOpenGLの次に続く新しいAPI群が予定されています。VulkanはAppleのMetal APIに相当するもので、並列化された最近のハードウェアのしくみに合わせ、それらの性能を最大限に活かしたアプリを作ることのできる環境を目指しています。

一方で、いままでグラフィックスドライバが面倒を見てくれていた並列処

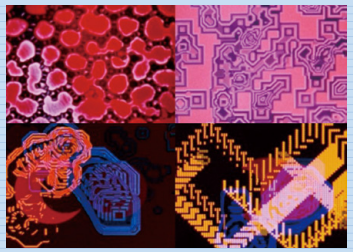
Henry Fuchs氏の クーンズ賞受賞の様子



GoPro HERO4を16台搭載した パノラマカメラリグ「Google JUMP」



Vulkanによる描画の展示 （インテルブース）



Lillian Schwartz氏のデジタル アート作品（lillian.com より）

超高解像度のデジタルビデオカメラ4台を 搭載した、パノラマ撮影用カメラセット

WHAMアプリの一体感を楽しむ観客

理や各種設定、負荷分散などを自前で用意しなければいけないということで、労力の割には速度的メリットは少ないとも言われています。これからのチューニングは、ハードウェア性能の向上が期待されます。

これからのコンピュータグラフィックスの進化

SIGGRAPH開催中の夜に、エレクトロニックシアターというCG短編の受賞作十数本を2時間ほど観られるイベントが開催されます。その開始までの待ち時間に、「WHAM」というアプリを使ったお楽しみがありました。このアプリによって会場にいる個々人のスマートフォン全部が連動し、映像を映し出したり、フラッシュが連動して光ったりして、たいへん盛り上がったのです。知らない人同士が集う会場で、デジタル技術であっという間に一体感が創り出されたのがとても感動的でした。

映画でしか触れることのできなかった最先端のコンピュータグラフィックス技術が、手のひらの中のスマートフォンで実現し、ネットワークやコンピュータパワーを気にすることなく、瞬時に手に入れることができる世界が身近にやってきています。一昔前なら1秒分の映像を制作するのに、数十億円もする高額なスーパーコンピュータが必要だった時代から、ノートパソコン1台、スマートフォン1台で、リアルタイムで素晴らしい表示がなされるようになりました。しかし、人間の欲望と表現への欲求は限りがありません。

来年夏のSIGGRAPH 2016は米国アナハイムで開催されることが決まっています。また、今年11月2日から5日の4日間開催されるSIGGRAPH ASIA 2015は、2009年以来の日本開催となる、神戸での開催です（http://sa2015.siggraph.org/）。アジア各国から集まるCG作品や、最新技術展示に多くの期待が集まっています。SD

Gadget 1

Wobble Strings

http://nae-lab.org/~naemura/publication/

弦楽器の震えプロジェクション

Wobble Stringsは、東京大学大学院情報理工学系研究科・苗村研究室による研究開発中のプロジェクトで、デジタルカメラなどでみられるローリングシャッター現象を活用したものです。たとえば高速に回転するプロペラや扇風機などをデジタルカメラで撮影したとき、ぐにゃぐにゃに曲がった写真が撮影された経験はないでしょうか？　Wobble Stringsはギターのピックアップから得られた信号をMIDIデジタルデータに変換し、その音データに応じた投影データを、ギターの弦がはってあるネックの部分に投影するというものです。

Gadget 2

Sensel Morph

https://www.kickstarter.com/projects/1152958674/the-sensel-morph-interaction-evolved

変化するタブレットデバイス

Sensel Morphは、表面カバーを付け替えることで、さまざまな用途に変化するタブレットデバイスです。圧力によって入力情報が変化し、押す強さに応じて入力値を変化させることができます。上に載せるシートによって、通常のQWERTYキーボードのほかにも、ピアノ鍵盤やDJコントローラ、筆ペンなどに変化します。現在は、クラウドファンディングのKickstarterで資金集めに成功し、製品化に向けて邁進しているそうです。2016年の中頃に249ドルで一般販売が予定されています。

Gadget 3

nod ring

https://nod.com/

指輪型モーションデバイス

nod ringは、指につけて、腕や手の動きを検知するためのモーションデバイスです。モーションセンサー、タッチセンサーが搭載されています。おもにゲームのコントローラ用途として、ディスプレイの前で使うことを想定しています。バッテリは約1日持ち、防水となっています。最大の特徴は、わざわざ新しいジェスチャを覚えたり、登録したりする必要はなく、ごく自然で単純な手や指の動きを検知し、それをトリガーにして操作が可能なことです。開発者用に99ドルで先行販売していましたが、今後予定されている一般販売の際は149ドルとなる見込みです。

Gadget 4

SKATER SCOPE

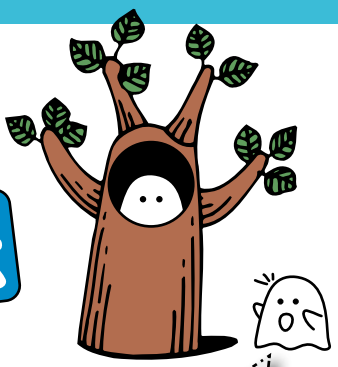
https://www.pstechnik.de/en/skater-scope-pl-pl/a-1059/

マクロレンズ接続用アダプタ

蟻サイズの小さなスーパーヒーローが登場する映画「アントマン」の撮影に用いられたカメラレンズアダプタ。実際の撮影には、Mocoというモーションコントロールカメラ（ロボットのように精密に動きを何度も繰り返すことのできる装置）の先に、レンズ方向を自由に設定できるこのSKATER SCOPEを取り付け、さらにその先に一眼レフカメラ用の100mm焦点マクロレンズを装着して映画撮影用のカメラとしたそうです。SKATER SCOPEの価格は約400万円。1日レンタルでも数万円します。極小の撮影ができますが、価格は小さくありませんね。

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

結城浩の再発見の発想法 31



Library

Library——ライブラリ

ライブラリとは

ライブラリ(Library)とは、ほかのプログラムから利用されることを前提としたプログラムで、通常はそれ単独での動作を前提としないものです。数学関数を集めたライブラリ、多倍長演算を行うライブラリ、暗号やセキュリティに関する関数を集めたライブラリなど、用途に応じた多種多様なライブラリが開発されています。

プログラマがアプリケーション(以下、アプリ)の開発を進めるとき、自分が使いたい機能がライブラリとして使えるなら、開発の手間は大きく軽減されます。ライブラリはプログラミング言語ごとに開発／提供されるので、使おうとするプログラミング言語にどんなライブラリが存在するかは重要な情報になります。良いライブラリが充実していれば開発は楽になるでしょう。

もともと、ライブラリは「図書館」という意味で、“library”という英単語はラテン語の“liber”(本)から派生したものです。調べものをしたい人が図書館に行って目的に合った本を利用するように、アプリを開発したい人は、ライブラリをうまく利用して必要な機能を実装するのです。

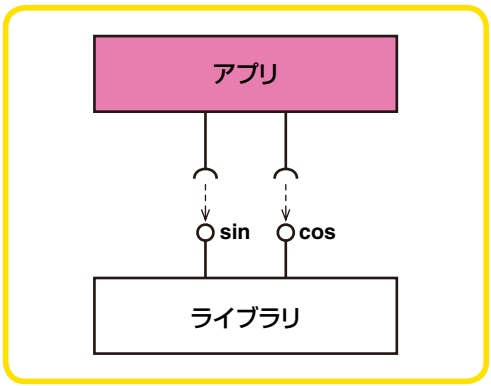
“library”という英単語には「書庫」という意味もあります。書庫という意味では、アーカイブ(archive)という言葉もあり、こちらは作ったものを変更せずに保存しておくニュアンスが強い言葉です。

用意されているメリット

ライブラリを使う大きなメリットは、プログラマがいちいち自分でプログラミングしなくても、ライブラリで提供されている関数やクラスを利用できる点にあります(**図1**)。たとえば、ゲームのアプリを作ろうとしてsinやcosの計算が必要になったとします。もしも三角関数ライブラリを使わなければ、プログラマは自分でsinやcosを計算するコードを書かなければなりません。これでは、本来やりたいゲーム開発になかなか取りかかれません。ライブラリは必要な関数やクラスがすでに用意されているからこそ便利なのです。それは、図書館に行けば必要な本がすでにそろっていて、すぐに利用できる状況とよく似ています。

ところで、図書館で調べものをする場合「必

▼図1　アプリはライブラリを利用する



要な本をすばやく見つけることができるか」は大切ですね。そのためには書籍の検索ができることや、そもそも司書がしっかり配架を整理しておくことが必要でしょう。アプリが利用するライブラリも同様に、ライブラリの開発者がしっかりと関数やクラスを設計し、ドキュメントやサンプルが準備されていることが大事になります。つまり、必要な関数やクラスがそろっているだけではなく、アプリのプログラマが使いやすい状態になっていることが求められるのです。

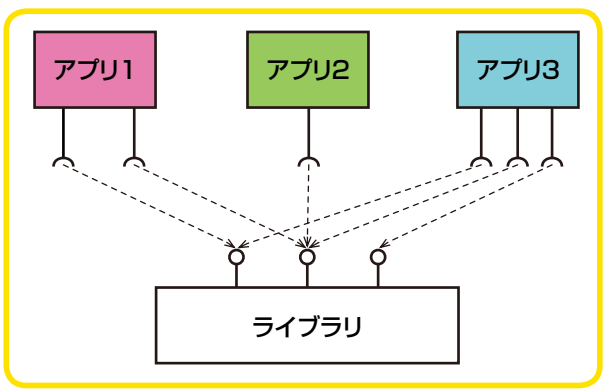
分離しているメリット

アプリの開発が楽になることだけがライブラリを使うメリットではありません。トラブルが発生したときにも、アプリとライブラリが分離していることが重要な意味を持ちます。それは、発生したトラブルがアプリとライブラリのどちらに起因しているのかを調べて、トラブルの原因を切り分けることができるからです。

また、実行環境ごとに異なるライブラリを用意し、それを切り換えるようにすれば、アプリの修正をほとんど行わずに複数の環境に対応できるプログラムが作れます。アプリとライブラリとが分離していればテストも容易になるでしょう。

アプリとライブラリが分離していれば、ライブラリはアプリと独立に性能アップに取り組むこともできます。汎用性の高いライブラリは、たくさんのアプリから利用されます(**図2**)から、1つのライブラリの性能を上げるだけで、たくさんのアプリの性能がアップすることになります。これはいいことですね。アプリとライブラリが分離しているからこそ、ライブラリをうまく共有できるのです。

▼図2　ライブラリはアプリに共有される



日常生活とライブラリ

プログラミングのライブラリが持つ発想法を日常生活に生かせるでしょうか。ライブラリはもともと、私たちの図書館から借りてきたメタファです。しかし、図書館以外にもライブラリの発想を生かせる場合がたくさんありそうです。

たとえば、何かを作るとき、「自分ですべてを作ろう」とするのではなく、「すでに世の中に存在するものを利用できないか」と考えるのは大事です。さらに「そもそも、自分が今から作ろうとしているものは、過去に誰か似たものを作っているのではないか」という発想も重要でしょう。これは、アプリ開発でライブラリを利用することに相当します。

もっと言うなら「自分が作り上げたものをほかの人に利用してもらえないか」という発想もありますね。これは、ライブラリを開発するプログラマの発想です。その際には、ほかの人が利用しやすくするための工夫も必要でしょう。

考えてみると、現代のインターネットは、「他人の成果物を利用する」あるいは「自分の成果物を他人に利用してもらう」ことを実現できる基盤と言えます。

プログラミングの分野では、インターネットの力を生かしてプログラマ同士が成果物を共有し合っています。それ以外の分野でも同じように、成果物をうまく共有できたらいいですね。

◆　◆　◆

あなたが何かを作るとき、「すべてを自分で作ろうとしていないか」あるいは「成果物を他人に利用してもらうことはできないか」とぜひ考えてみてください。SD

増井ラボノート

コロンブス日和

第2回　Gyump

エンジニアというものは「楽をするためならどんな苦労も厭わない[注1]」ものだと言われていますが、コロンブスの卵のようなゴキゲンな発明によって頑張らずに楽できるなら、それに越したことはないでしょう。私はコンピュータ上の簡単な工夫で楽をする方法を考えるのが好きで、長年にわたっていろんなシステムを開発してきています。今回の連載では、私がこれまで開発して長年実際に利用しているような単純かつ便利なシステムをたくさん紹介していきます。

Author　増井 俊之（ますい としゆき）　慶應義塾大学

長いURLを便利にしたい

あらゆるものがURLで表現可能になりつつありますが、URLで表現する対象が膨大な量になってくると、長いURLを扱う機会が増えてきています。長いURLは取り扱いが面倒ですし、記憶するのもたいへんですから、次のような工夫がよく用いられています。

ブックマークを利用する

よく使うURLに手軽にアクセスできるようにするために、ブラウザの**ブックマーク機能**が広く使われています。ブックマーク機能はMosaicのような初期のブラウザでも実装されていたほどポピュラーなもので、あまり機能が進化しないまま現在のブラウザでも利用されています。

ブラウザのメニューからよく使うURLにアクセスできるのは便利ですが、気軽に登録しているとあっという間にメニューが一杯になってしまいます。階層的にブックマークを管理すれば良いのですが、きちんと分類して管理するのは面倒なので、結局利用の機会が減ってくることが多いようです。

ブックマーク情報を、複数のマシンやブラウザで共有しづらいのも面倒です。最近のブラウザではブックマーク情報を複数マシンで同期できますが、こういう設定は面倒なものです。また、ブックマークをブラウザ上に記憶する代わりに、クラウド上に記憶して共有する**ソーシャルブックマーク**のようなサービスもありますが、ソーシャルブックマークは、Webページにコメントを付ける用途で利用されることが多く、頻繁に利用するURLにアクセスする用途にはあまり使われていないように思われます。

長いURLを短いURLで置き換える

長いURLに簡単にアクセスしたり他人にURLを知らせたりする場合のために、長いURLを短くしてくれるTinyURL、Bit.ly、Goo.glなどの**URL短縮サービス**がよく使われています。これらのサービスを利用すると、`http://pitecan.com/`……のような長いURLの代わりに、`http://goo.gl/jx7VZy` や `http://bit.ly/1LnYzAt` のような短いURLを利用できます。

URLが短ければ記憶が可能かもしれませんし、他人とURLをやりとりする場合にも便利なのですが、「jx7VZy」のような暗号的な文字列を記憶することは困難です。覚えることができないため、ファイルなどに書いておく必要があるのであれば、長いURLを利用するのとあまり変わらないかもしれません。短い名前を自分で選べるサービスもありますが、好きな名前が使えるとは限りませんし、一度登録したURLを後で変更できないのが普通です。

また、短いURLと長いURLの対応はサーバに記憶されているので、サービスにトラブルがあったり、サービス自体が終了したりすると使えなくなるのは心配です。普通は対応データベースを取得できませんから、データをバックアップしておくこともできません。

注1）http://thinkit.co.jp/free/article/0709/19/

▼図1　短縮URLサービスでcurlコマンドを試す

```
% curl -w '[%{http_code}]¥n' http://goo.gl/jx7VZy
<HTML>
<HEAD>
<TITLE>Moved Permanently</TITLE>
</HEAD>
<BODY BGCOLOR="#FFFFFF" TEXT="#000000">
<H1>Moved Permanently</H1>
The document has moved <A HREF="http://pitecan.com/">here</A>.
</BODY>
</HTML>
[301]
%
```

そもそも個人的に使うURLは短くて良いはずです。私の周囲で「SFC」といえば慶應義塾大学湘南藤沢キャンパスのことに決まっており、スーパーファミコンのことではありません。自分の周囲で言及することが多いものは、たいてい数文字で表現できてしまうでしょうから、よく使うものから順番に短い名前を割り当てておけば便利でしょうし、情報圧縮や効率化の面でも有利なことは間違いありません。今回は自分用の短い名前を気軽にブックマークとして活用できるGyump（ジャンプ）というサービスを紹介します。

Gyump：柔軟なURL短縮サービス

URL短縮サービスの1つであるGoo.glにhttp://pitecan.com/を登録するとhttp://goo.gl/jx7VZyのようなURLが利用できるようになります。curlコマンドで確認してみると、**図1**のようなHTMLとステータス301（Moved Permanently）が返ってくることがわかります。

Goo.glや、そのほかのURL短縮サービスは、短い名前と長い名前の対応データベースを保持しており、短い名前でアクセスされたとき**図1**のようなHTMLとステータスコードを返すという単純なしくみで動いているようです。このようなものは自分で作るのも簡単です。

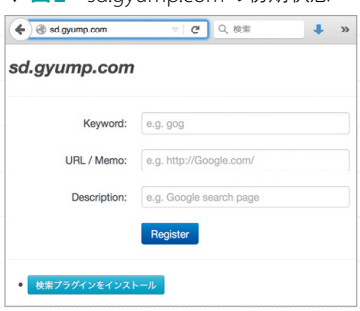
Gyumpは、ユーザが指定した名前を使って任意のURLにアクセスできるようにするサービスです。たとえば「sd」というIDを持つユーザが「map」というキーワードで東京駅の地図にアクセスしたいとき、http://gyump.com/sd/mapというURLを利用できるようにするというものです。既存の短縮サービスと異なり、Gyump.com以下の名前は任意のものが使えるので、自分のID（e.g. sd）と短い名前（e.g. map）のような任意の組み合わせを利用できます。

何も登録されていない状態でGyump.com/sd/にアクセスすると、**図2**のように登録フォームだけが表示されます。

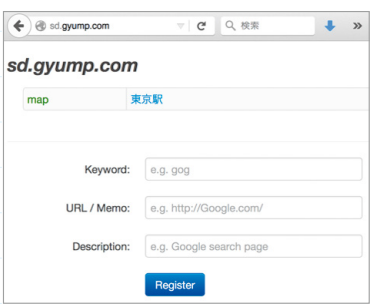
ここで「map」というキーワードと東京駅の地図のURLを入力して登録すると、東京駅のURLが「map」という名前で登録され、登録されたURLが**図3**のようにリストされます。この状態ではhttp://sd.gyump.com/mapというURLで東京駅の地図にアクセスできます（**図4**）。

リスト中の「map」をクリックすると編集モードになります。http://sd.gyump.com/map!のように、短いURLの最後に「!」を付けたURLにアクセスすると、直接編集ページに飛ぶことができます（**図5**）。

▼図2　sd.gyump.comの初期状態

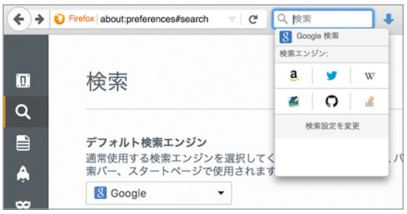


▼図3　東京駅を登録後のリスト



▼図4　http://sd.gyump.com/mapにアクセスして表示される地図

▼図5　編集フォーム



◀図6
Firefoxの検索エンジン選択機能

◀図7
デフォルト検索エンジンとして「sd.gyump.com」を指定

ブラウザの検索窓に登録

短いとはいってもhttp://sd.gyump.com/mapのようなURLをブラウザに入力するのは面倒でしょう。Firefoxなどではopensearch機能を使って検索窓をカスタマイズできるようになっているので(図6)、Gyumpを検索システムとして登録しておけば、検索窓にmapと入力するだけでhttp://sd.gyump.com/mapにアクセスできるようになるので便利です(図7)。

このように、Gyumpをデフォルト検索サービスとして登録しておくと、検索枠に「map」と入力するだけで東京駅の地図を開くことができます。Gyumpに登録されていない文字列を入力するとGoogle検索するようになっているので、検索エンジンを切り替える必要は多くありません。

階層的な情報管理

たくさんのURLを扱いたい場合はGyump.com/sd/maps/tokyoのような階層的な名前も利用できます。この場合、Gyump.com/sd/maps/akiba、Gyump.com/sd/maps/shibuyaのような名前を登録しておけば、Gyump.com/sd/maps/で地図のリストを見ることができます。

「sd」や「map」のような名前は自由に選択できますし、登録URLを後から変更することもできるので、次に示すようにさまざまな使い方ができます。

Gumpの活用例

その1「地図へのアクセス」

初めての場所に行こうとするとき、行先をあらかじめGoogle Mapsなどで調べておく人が多いと思いますが、パソコンで調べたURLにスマホからアクセスするためには、なんらかの方法でURLを送る必要があるので面倒です。Google MapsのURLはとても長いのでメールで送るのも手軽ではありません。

私は、行先の地図のURLを常にmyname/mapのような名前でGumpに登録するようにしており、Gyump.com/myname/mapのショートカットをスマホのホーム画面に登録してあるので、スマホ上でこれをタップするだけで、常に行きたい場所の地図を表示できるようになっています。行先が変わった場合でも、同じURLから目的地の地図が表示されるのでとても便利です。

その2「予定表」

私は予定表をWebで管理しており、Gyump.

com/myname/sのような短いURLでアクセスできるようにしています。予定表のURLは毎月変わるのですが、現在の日付からURLを計算して今月の予定表ページに飛ぶプログラムを用意しているので、常に同じURLで今月の予定表にアクセスできるようになっています。

その3「買い物／メモ」

GyumpのURL登録欄にURL以外の文字列を書いておくと、そのURLに飛ぶかわりに、登録した文字列が表示されるようになっているので、私は買い物メモなどはGyump.com/myname/buyのようなところに書くようにしています。

その4「よく使うショートカット」

よく利用するサービスに飛ぶためのブックマークとしても私はGyumpを活用しており、たくさんの固定URLを登録して使っています。「tenki」で天気予報にアクセスしたり、「jor」でジョルダン乗り換え案内にアクセスしたり、「hon」で本棚.orgのページにアクセスしたり、あらゆる状況で頻繁にGyumpを利用しています。

その5「一時的な仕事での利用」

不慣れなAPIなどを調べて使いたいような場合、マニュアルやブログなどたくさんのページを参照しながら理解を深めるものですが、検索したページをすべて開いたままにしておくとブラウザがタブだらけになってしまいます。このような場合、必要になるかもしれないページを Gyump.com/sd/api/1、Gyump.com/sd/api/2のようなアドレスにどんどん登録しておくようにすれば、タブの数などを気にすることなく後で簡単にアクセスできます。

その6「登録URLのチェックとバックアップ」

通常のURL短縮システムと異なり、Gyumpでは登録されたURLのリストを眺めることができますから、そのページをバックアップしておけばGyumpが使えなくなった場合でも安心です。

Gyumpアドレスのブックマーク

Gyump.com/sd/mapのようなURLをブラウザやスマホでブックマークしたい場合、ブラウザ上でこのようなURLを入力すると、ここに登録されている地図URLにジャンプしてしまうためGyump.com/sd/mapという短いURLをブックマークできません。Gyumpでは、Gyump.com/sd/で表示されるリスト画面の中から「map」を選択するとGyump.com/sd/mapのようなアドレスで登録画面が表示されるので、ここで短いURLをブックマークできます。

Gyumpの利用経験

私はこのシステムを長年活用しているのですが、同様のシステムを活用している人は多くないようなのが不思議です。さまざまなWebページにアクセスしようとするときは、毎回Google検索したり、ブラウザのURL補完機能やブックマーク同期機能を利用したり、Google Mapsのようなサービスが用意している登録機能を利用したり、Evernoteのような情報管理ツールを利用したり、人それぞれにいろいろな方法が利用されているようですが、Gyumpは単純な割に応用が広く、「sd/map」のような文字列さえ覚えておけば、あらゆるブラウザで使えるので安心ですし、他人に口頭でURLを伝えるのにも便利だと思います。

Gyumpのようなシステムはもちろん万能ではありません。「map」のような適切な名前を思いつかない場合もありますし、登録したものを他人に書き換えられてしまう可能性もあります。しかし運用を工夫すればそれほど困ることはありませんから、メリットを活かして活用するのが良いと思っています。

しくみの単純さと効果の大きさを掛けたものを、筆者は**コロンブス指数**と呼んでいるのですが、単純なしくみにもかかわらず応用範囲が広く有用だという意味で、Gyumpのコロンブス指数はかなり高いと言えるでしょう。SD

軽酔対談

かまぷの部屋

第17献 ゲスト：吉岡 弘隆さん

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

吉岡 弘隆（よしおか ひろたか）さん

1958年生まれ。楽天株式会社技術理事、ビジネス・ブレークスルー大学教授、産業技術大学院大学客員教授。1984年、慶應義塾大学大学院修了後、日本DEC研究開発センター入社、ソフトウェアの日本語化、国際化などに従事。日本Oracleを経て、2000年にミラクル・リナックスを創業、取締役CTOを歴任。2009年より現職。1999年よりカーネル読書会を主宰。日本におけるオープンソース、ハッカー、勉強会文化の普及・啓蒙に努める。TwitterID：@hyoshiok

子供時代、何が好きでしたか？

◀本が好きで、本屋に何時間居ても飽きない子でした。将来は文字を書くのが仕事になればいいと思っていて、今でも小説家に憧れています。

吉岡さんのコンピュータとの出会いはいつごろですか？

◀中学が大学に付属していた関係で、大学にあるコンピュータを使って、『電子計算機入門』という課外授業があり、それが最初です。コンピュータといっても、その当時はパンチカードで入力する計算機です。

パンチカードって実物を見たことないのですが、どんなものですか？

◀紙に穴をあけて1枚が1行分です。今の若い人たちは写真も見たことない人が多いと思いますが、1枚1円ぐらいで、タイプミスをすると、その1枚が無駄になるのでたいへんです。よく考えてからプログラムを書くという経験をしました。今は画面でいくらでもやりなおしができるけど、そのころはパーソナルコンピュータがない時代で、もちろんキーボードも触ったことがない。キーを探してポチポチ打つ。そうすると機械がパンチカードに穴を開け（パンチカードに穴をあける機械を穿孔機（せんこうき）と呼ぶ）、それが何枚もできる。それを読み込ませて初めてプログラムが計算機に入力されるわけです。プログラミングをすると何かが出てくる。その体験がすごく楽しかった。この環境があったのはラッキーだったと思います。

パソコンが出て来たのはいつごろですか？

◀高校2年生のとき、日経サイエンスという雑誌の記事に「マイクロプロセッサ」が紹介されていました。このチップの誕生でコンピュータの原理が成り立つんですよね。その記事には、きれいなICのカラー写真があって、これにメモリをつけるとコンピュータとして動作すると書いてある。当時の計算機は数千万円で、何百ドル（数万円）で同じものができるとわかって驚きました。

大学では、どんな研究をされたのでしょうか？

◀大学では、工学部の統計の研究室に入りました。大学院ではデータベースの研究室に進みました。論文を集めるのが趣味で、山のように論文を集めました。

新卒でDECに入社されて、エンジニアの職に就いたのですか？

◀はい、私は新卒でDECの研究開発センターに入社しました。そこではCOBOLの日本語化が初めての仕事でした。入社したのは1984年のことですが、入社してすぐに会社主催の花見の会がありました。そこで米国本社から来ていたエンジニアを紹介してもらったりして、いきなりのことに驚きました。米国の技術者は当時の自分には雲の上の存在でしたから。

DECからOracleへ転職されたのは、何がきっかけだったのですか？

◀DECの景気が悪くなって、希望退職を募っていたので手を挙げました。お世話になった方々へ退職メールを書いて送ったのですが、DEC RDB（という製品があって、Oracleへ部門ごと売却された）でお世話になった人が米国Oracleにいて、「Oracleの国際化どう？」と、職を紹介してくれました。トントン拍子で




米国に行きました。Oracleでのシリコンバレー時代は楽しくてしょうがなかったです。

シリコンバレーではとくに何が楽しかったですか？

スタンフォード大学の授業に潜入し、データベースの授業を聞き、毎週のように参加しました。米国大学のいいところは、社会人が授業に参加していても問題なくて、大学以外でも勉強会のようなものはいたるところにありました。その費用は10ドル、20ドルで、参加しやすいのもよい点です。CPUを作っている人、OSを作っている人、コンパイラを作っている人、いろんな人が集まっていて、盛んに情報交換していました。約3年半、貴重な時間を過ごしました。

カーネル読書会を始めたきっかけは何ですか？

日本に戻って、シリコンバレーで体験したことを日本でもできるか試験的にやってみたんです。やってみたらみんな楽しんで盛り上がりました。第1回は川崎の溝の口に30名ほど集まりました。それがだんだん規模も大きくなり、気がついたら100回を越えることができました。実際のところは、システムコールの実装が知りたいな、と思ったことがきっかけでした。

GitHubを勧めていますよね。

もっとGitHubの文化が伝わるといいと思っています。利用しているところは利用していると思うのですが、大企業はまだまだじゃないですかね。オープンソースの世界では当たり前のように理解されていますが、IT技術（ソフトウェア開発）という範囲では、まだまだ認知が浅いと思います。

楽天へ転職されたきっかけはなんだったんでしょうか？

ずっと外資だったので、国内に本社がある会社に勤めてみたかったのです。東京で働くなら日本の会社ですね。それとオープンソースに力を入れている会社に入りたかった。日本でオープンソースに力入れている会社はWeb系が多く、その中で楽天にお世話になることになりました。社長の三木谷さんは、インターネットやオープンソースのことをよく理解している経営者ですね。日本の経営者の中では一番理解している一人じゃないかな。そして入ってみて実感していることですが、楽天の偉いところは、人材育成に本気で力を入れているところですね。

楽天の有名な英語公用語化はその一部なのでしょうか？

そうですね。英語以外にも教育メニューは充実しています。人間はソフトウェアと一緒で変化し続ける必要がありますから、力を入れて環境を与えてしっかりと育てる。それを愚直にやっています。英語化を始めるときはTOEICのスコアなどを指標としていましたが、そういう指標も変化しつつあります。最近では隣の席が外国人というのが当たり前になってきたので、英語が話せないと話にならないです。スコアがあるだけではダメです。英会話に力を入れていなかった人にとっては、たいへんな環境だと思います。会社は伸びる社員をサポートします。そして会社が躍進するためには「教育」が欠かせない要素ですね。

吉岡さん個人は10年後、どうなっていたいですか？

著述業かな。あと、タモリ倶楽部に出たいです。本当は笑っていいともテレフォンショッキングに出て、徹子の部屋に出て、タモリ倶楽部という順で考えていました。ブラタモリが復活してうれしいです。タモリさん大好き。

今日はたくさんのお話、どうもありがとうございました。 SD




ツボイのなんでもネットにつなげちまえ道場

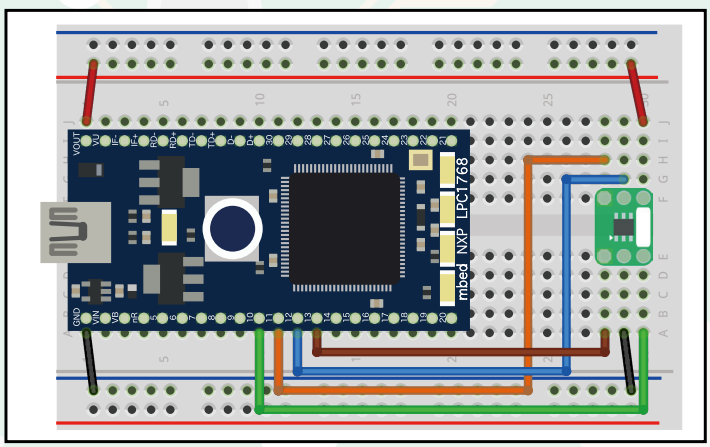
第六回 SPIで通信してみる

Author 坪井 義浩（つぼい よしひろ） Mail ytsuboi@gmail.com @ytsuboi
協力：スイッチサイエンス

はじめに

前回はさまざまな通信プロトコルを紹介しました。UART（Universal Asynchronous Receiver Transmitter）は一般的ですし、皆さん簡単に使うことができるでしょう。そこで今回は、前回紹介したSPI（Serial Peripheral Interface）を実際に使ってみましょう。mbedには、多くの電子部品にライブラリが作られ、Componentsとして登録されています。OSで言うところのデバイスドライバがすでに提供されているようなものです。ですから、SPIを直接操作しなければならないのは、Componentsに登録されていない部品を使うときくらいでしょう。今回は、筆者が使いたいと思って中国から買い付けてきた、「GT20L16J1Y」という日本語フォントが入ったROMをSPIで読み出して使ってみたいと思います。

▼図1　配線図



GT20L16J1Y

このチップのメーカーのページ注1を見ると、「GT20L16J1Y規格书V2.0I_B」注2というデータシートが掲載されているのが見つかります。中国語で書かれているので、とても読みづらいですが、データシートを見て、どのようにアクセスをすればよいのか確認しましょう。

まず、基本的なところを確認します。GT20L16J1Yを使うための電圧と、アクセス方法です。4ページを見ると、SPI、30MHz、工作电压：2.7～3.6Vといった情報が読み取れます。このチップは、2.7～3.6Vで動き、30MHz以下のSPIでアクセスすればよいようです。10ページには、ピン配置と、SPIのタイミング図が載っています。このタイミング図を見る限り、SPIのモード0あるいはモード3でアクセスするようです。また、マイコンとROM（GT20L16J1Y）の配線については、先ほどのピン配置に加えて11ページに配線図が載っていました。これを参考に、mbed LPC1768とGT20L16J1Yを**図1**のように配線します。

ここでは、mbedのp10（CS）をROMのCS#(3)に、p11(MOSI)をSI(6)に、p12(MISO)を

注1）http://www.genitop.com/Products/indexlist_GT20L16J1Y.html
注2）http://www.genitop.com/admin/upload/20150511144125736.pdf

SO(5)に、p13(SCLK)をSCLK(1)にそれぞれ接続しています。mbed LPC1768がSPIのマスタに、ROMがスレーブになっています。そういえば、前回はCS(チップセレクト)ではなく、SS(スレーブセレクト)と記していました。昨今ではスレーブ(奴隷)という語感がよろしくないということで、こういった置き換えをよく見ます。

ソフトウェア

接続をしたところで、実際にSPIを使ってアクセスしてみましょう。mbedでSPIを使うには、まず、

```
SPI spi(p11, p12, p13);
```

といった具合に、SPIとして使うピン名を指定し、コンストラクタを呼んで初期化します。ここで、spiというオブジェクトができますので、

```
spi.format(8,3);
spi.frequency(1000000);
```

と、データフレームの長さと、モード、また、SPIのクロックの速さを設定します。この場合、1,000,000Hzですので、1MHzです。

先ほどのデータシートの8ページ目を見ると、このROMを使うには、まず0x03を書き込んでREADモードにして、続いて読み出したい番地を3byte(24bit)送るようです(**図2**)。SPIの設定を終えたので、SPIでアクセスしてみましょう。SPIでアクセスする前に、CSをHIGHからLOWにして、今、通信している相手のチップを明確にします。そのうえで、

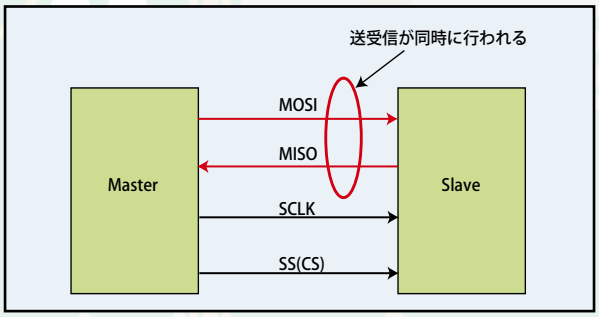
```
spi.write(0x03);
```

とすると、0x03をマイコンからROMに送ることができます。この調子でuint32_tのAddressに入っている番地を、1byteずつ転送します。

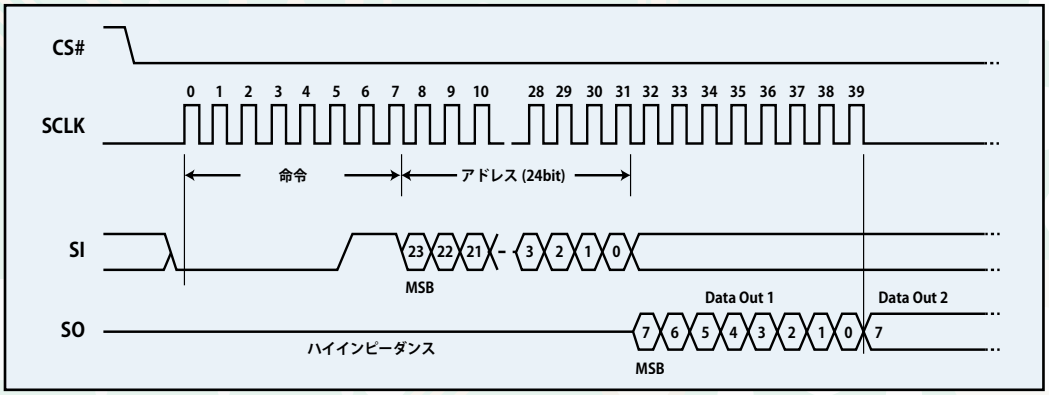
```
spi.write(Address >> 16  & 0xff);
spi.write(Address >> 8   & 0xff);
spi.write(Address        & 0xff);
```

これで、読み出したい文字が記録されている番地を送ることができました。次に、ROMから送られてくるデータを読み出しましょう。前回記したように、SPIの通信はマスタ主導で行う必要があります。SPIは、SCLKに合わせてMOSI(Master-Out Slave-In)とMISO(Master-In Slave-Out)とが同時に通信を行います(**図3**)。

▼図3 SPIの通信



▼図2 フォントROMを読み出すときの信号

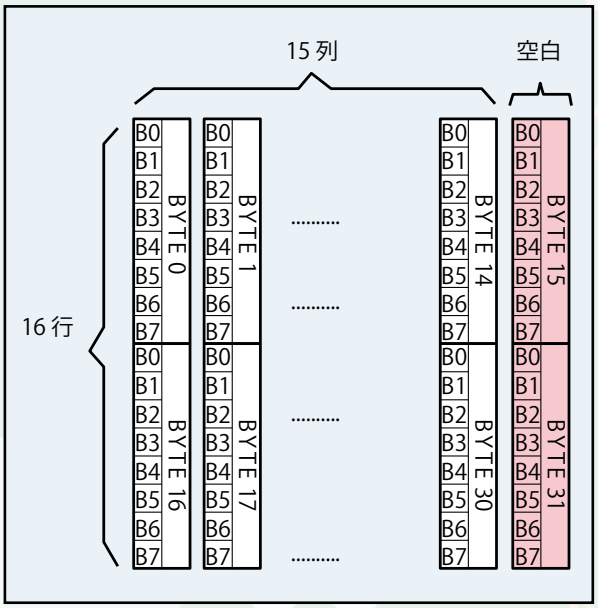


マスタが何かを送ろうとしてSCLKを出さなければ、スレーブからは何も戻ってきません。ですので、マスタからは、0x00などのデータを送ります。スレーブから送られて来た値はspi.write()の戻り値になっていますので、次のように読み出します。

```
for (int i = 0; i < 32; i++) {
    matrixdata32[i] = spi.write(0x00);
}
```

32回ループを回しているのは、全角文字の場合、フォントのデータが1文字あたり32byteあるからです。**図4**のように15×16ドットフォントが1列のスペース付きで格納されています。最後に、CSをLOWからHIGHにするとSPIの通信を終了させることができます。上記の例では、matrixdata32という配列にフォントのビットマップが入っているので、これを液晶などに表示して使います。

▼図4　フォントの格納イメージ



▼図5　フォントが表示される

```
SJIS, 0x8b5a
JIS X up8 = 0x35 downn8 = 0x3b
MSB = d21 LSB = d27
Address = 59456
------XX-----------XX----------
------XX-----------XX----------
------XX-----------XX----------
------XX----XXXXXXXXXXXXXXXXX--
XXXXXXXXXXX--------XX----------
------XX-----------XX----------
------XX-----------XX----------
------XX--XX--XXXXXXXXXXXXX----
------XXXX------XX--------XX----
----XXXX--------XX------XX------
XXXX--XX----------XX----XX------
------XX----------XX--XX--------
------XX-----------XX----------
------XX----------XX--XX--------
--XX--XX------XXXX------XX------
----XX----XXXX-----------XXXX--
```

サンプルコード

今回は、UARTで読み出したフォントを表示してみるサンプルコード[注3]を用意してみました。mbed LPC1768は、パソコンにUSBで接続すると、ドライブだけでなく、シリアルポートも備えた複合デバイスとして認識されます。OS Xではドライバのインストールが不要ですが、Windowsでは、ドライバのインストールが必要です。mbedのシリアルポートのドライバは、配布ページ[注4]でダウンロードできます。

サンプルコードを“Import this program”ボタンをクリックして自分の開発環境にインポートし、“Compile”ボタンをクリックすると、コンパイルが実行され、生成されたバイナリのダウンロードが始まります。ダウンロードされたバイナリをmbedのドライブにコピーし終えたら、mbed LPC1768のリセットボタンを押してください。

次にシリアルポートを、お手持ちのシ

注3）https://developer.mbed.org/users/ytsuboi/code/GT20L16J1Y_sample/
注4）https://developer.mbed.org/handbook/Windows-serial-configuration

リアルターミナルで開きます。mbedでUARTを使用するとき、とくに指定をしなければ、9,600bps、8bit、パリティなし、ストップビット1でUARTが動きます。お手元のシリアルターミナルも、この設定にしてください。また、このフォントROMには、JIS X 0208の並びでフォントが格納されています。サンプルプログラムは、Shift_JISからフォントROM内のアドレスに計算して読み替えを行っています。ですので、シリアルターミナルの文字コードをShift_JISにすることも忘れないでください。

シリアルターミナルでmbedのシリアルポートに接続し、「技」と入力すると、**図5**のようにROMに格納されていたフォントが表示されます。

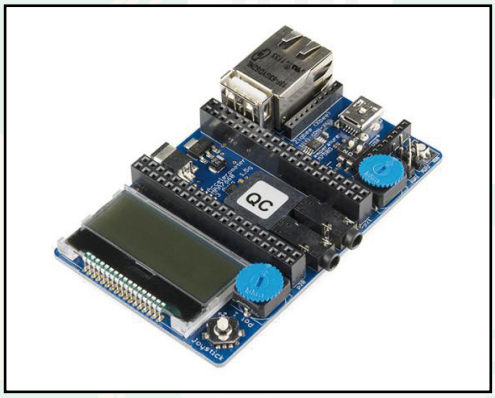
ライブラリ

ここまでは、SPIを使って、直接フォントROMにアクセスしてきました。しかし冒頭で紹介したように、mbedのComponentsページには、多くのライブラリが登録されています。このGT20L16J1Yのライブラリもすでに作られ、登録されています[注5]。このComponentsページにあるサンプルコードは、UTF-8をJIS X 0208に変換するテーブルも含まれています。このため、ソースコードに書いた日本語を、mbedアプリケーションボード[注6]という基板(**写真1**)に搭載されている液晶に表示することができます(**写真2**)。ちなみに、この液晶もSPIでmbedに接続されています。SD

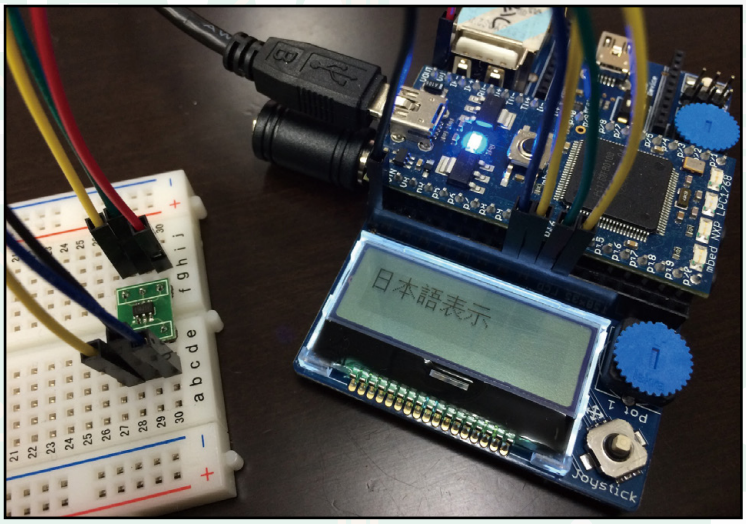
注5) https://developer.mbed.org/components/GT20L16J16Y-Japanese-font-ROM/

注6) https://www.switch-science.com/catalog/1276/

▼写真1　mbedアプリケーションボード



▼写真2　液晶に漢字を表示させてみた



コラム　クワッドSPI

SPIは、「MOSI」「MISO」「SCLK」の3本の信号線を使うことから3線式シリアルとも呼ばれます。より大きな帯域幅を求めて、SPIを拡張して作られたクワッドSPIという規格も存在します。SPIはMOSIとMISOの2本を使った全二重の通信を行いますが、クワッドSPIは4本の信号線を使い、半二重の通信を行います。複数のデータ線を使っているので、厳密にはシリアル通信とは言えませんね。

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」にアクセスし、アンケートにご協力ください（アンケートに回答するにはgihyo.jpへのお名前と住所のアカウント登録が必要となります）。ご希望のプレゼント番号を記入いただいた方の中から抽選でプレゼントを差し上げます。締め切りは**2015年12月17日**です。プレゼントの発送まで日数がかかる場合がありますが、ご了承ください。

ご記入いただいた個人情報は、プレゼントの抽選および発送以外の目的で使用することはありません。アンケートの回答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。記入いただいた個人情報は、作業終了後に責任を持って破棄いたします。

01 ハイレゾ対応USBパワードスピーカー Olasonic『TW-S9』

1名

卵型のPC用スピーカー。96kHz/24bitまでのハイレゾ音源入力に対応しています。パソコンへはUSBケーブル1本で接続し、バスパワーで動作します。卵型キャビネットは音響的に理想的な形状で、剛性が高く、箱鳴りなどの不要音の発生や定在波を防ぎ、高音質再生を実現しています。OSは、Mac OS X 10.1以降、Windows Vista/7/8/8.1/10に対応しています。

提供元 東和電子 http://www.olasonic.jp/

02 D456 USB Desktop アクアリウム

1名

デスクで魚が飼える癒しグッズ。USBから循環ポンプとLEDライトに給電します。ペンスタンドと液晶表示の時計・カレンダー・温度計も付いて、通常のツールとしても活躍。忙しい作業空間に癒しをお届けします。

提供元 パソコン工房 http://www.pc-koubou.jp

03 ウイルスバスター モバイル

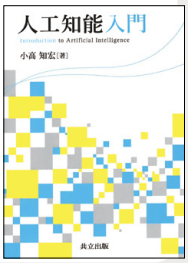
1名

不正アプリ対策、Web脅威対策、保護者による使用制限、盗難／紛失時の対策、迷惑SMS／迷惑着信対策が行えるスマートフォン・タブレット用のセキュリティアプリ。「1年1OS用」をプレゼントします。

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

04 人工知能入門

小高 知宏 著



探索、知識表現と推論、学習、自然言語処理、進化的計算と群知能など、人工知能の基本分野を解説した1冊。ビッグデータ解析やディープラーニング、技術的特異点などの現代的な話題も紹介しています。

提供元 共立出版 http://www.kyoritsu-pub.co.jp

2名

05 実践 JUnit

Jeff Langr, Andy Hunt, Dave Thomas 著

Javaユニットテストを自動化するフレームワーク「JUnit」の解説書です。ユニットテストの基礎からチーム開発でのユニットテストまで、実践的なテクニックを解説しています。

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

2名

06 あなたの知らない超絶技巧プログラミングの世界 遠藤 侑介 著

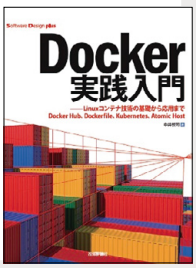
自身のコードを出力する「Quine」、特定の文字だけでコードを組む「○○禁止プログラミング」、コードのアスキーアート化など、一般的な技術書ではお目にかかれないプログラミングテクニックが満載です。

提供元 技術評論社 http://gihyo.jp

2名

07 Docker 実践入門

中井 悦司 著



Dockerが動くしくみから、GitHubと連携したデプロイ方法、Dockerfileの書き方、kubernetesとの連携方法、Atomic Hostでの使い方など、実践を想定したDockerの使い方を解説しています。

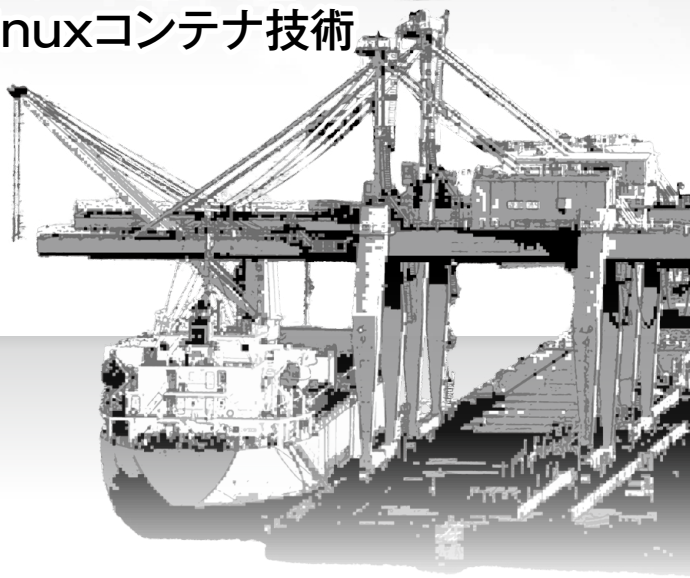
提供元 技術評論社 http://gihyo.jp

2名

第1特集

[決定版] Docker自由自在

実用期に入ったLinuxコンテナ技術



すでに普及段階に入ったDockerとコンテナ技術。本特集ではDockerのおさらいから、さらに使いこなすためのノウハウをまとめました。

第1章ではDockerコマンドを使い、コンテナを操作する方法を紹介します。第2章ではDockerイメージの使い方とDockerクライアントの操作方法を紹介します。第3章ではDockerデーモンの管理方法を学習します。第4～6章では、Dockerによる自動処理や、クラスタ管理のツール、設定ファイルによる自動化などの手法を紹介します。最後の第7章では、HashiCorp社のツールを使ったDockerの環境構築や管理を紹介します。

第1章 Dockerのキホン
環境構築から基本操作をマスターしよう ……P.18

第2章 Dockerのイメージ管理と基本操作
レジストリとDockerクライアントの操作法 ……P.27

第3章 Dockerの操作と管理
便利なコマンド／オプションを一挙に学ぼう ……P.35

第4章 Docker環境を自動構築
オーケストレーションツール「Docker Machine」 ……P.44

第5章 Docker Swarmでコンテナのスケジューリング
コンテナをクラスタリングしリソースを管理 ……P.51

第6章 Docker環境のコード化とオーケストレーション
DockerfileとDocker Composeで作業の省力化・効率化 ……P.59

第7章 HashiCorpの自動化ツールとDockerの連携
HashiCorp道の真髄（しんずい） ……P.64

Author 前佛 雅人（ぜんぶつ まさひと） クリエーションライン㈱ Twitter @zembutsu

第1章 Dockerのキホン

環境構築から基本操作をマスターしよう

日本国内でも日に日にDockerやコンテナについての認知が広まりつつあります。そもそもDockerやコンテナとは何であり、どのような機能や利便性をもたらすのでしょうか。本章ではまず始めに、Dockerの概要と環境構築のしかたを学びます。それから、Dockerコマンドを使い、一通りのコンテナの操作方法を習得しましょう。

Author 前佛 雅人（ぜんぶつ まさひと） クリエーションライン㈱ Twitter @zembutsu

Dockerの成り立ち

Dockerは、Docker社[注1]がオープンソースとして開発／提供しているコンテナ管理用のプラットフォームです。一般的にDockerと呼ばれる場合は、このDocker社を指す場合と、中心となるプログラムのDockerエンジン（Dockerデーモン）と周辺ツール（Docker Toolbox、Docker Machineなど）やサービス（Docker Hub、Docker Trusted Registry）を総称している場合があります。

Dockerが初めて発表されたのは、2013年3月15日、カリフォルニア州のサンタクララで開催されたPython開発者向けイベントPyCon 2013でした。dotCloud社のCEOだったSolomon Hykes氏により、わずか5分間のライトニングトーク[注2]で発表されました。このとき、概要の説明と「docker」コマンドでDockerイメージを使い、Dockerコンテナを実行できるというデモが行われました

その発表から1週間ほどして、Dockerのプロジェクトとソースコードが GitHub上に公開されました。ソースコードはオープンソースとして公開され、社内外から多くの人たちが開発に参加するプロジェクトがスタートします。そして、2015年10月現在の段階で、GitHubのスター数は25,000件を超え、18,000以上ものコミット、累計1,300名のコントリビュータが参加しており、GitHub上でもトップ20以内のプロジェクトになるまでに成長しています。

Dockerそのものも、世界で幅広く使われるようになりました。Dockerイメージを共有するDocker Hubは、過去1年間で24万人の利用者、15万件のリポジトリ、5億ダウンロード（pull）[注3]という、世界でも有数の開発プラットフォームに成長しています。

dotCloud社はもともとPaaS（Platform-as-a-Service）事業を展開していました。発表後、Dockerを事業の主力として取り扱うことを決め、社名をDocker社に変更しています。

Dockerの普及に伴い、業界動向も大きく変わりつつあります。当初はこのDocker社だけでコンテナの仕様取り決めや方向づけを行っていました。しかし、2015年6月に開催されたDockerConでは、The Open Container Project[注4]の発表を通し、さまざまな開発会社やベンダ、クラウド事業者が協力して、共通のコンテナ仕様を決めることが発表されました。

Dockerが解決する問題

Dockerが幅広く使われるようになった理由を紐解（ひも）いてみましょう。Dockerのよいところは、

注1）https://www.docker.com/
注2）The future of Linux Containers https://www.youtube.com/watch?v=wW9CAH9nSLs
注3）http://www.slideshare.net/Docker/dockercon-15-keynote-day-2/16
注4）http://www.opencontainers.org/

簡単なコマンドを実行するだけで、コンテナを実行できる点にあります。アプリケーションだけでなく、ミドルウェア環境もすべてコンテナ化することにより、開発環境／テスト環境／本番環境を問わず、アプリケーションが必ず動く環境を提供します。

これにより、環境におけるライブラリやミドルウェアなどの違いによって「開発環境では動いたのに、本番環境では動かなかった」という問題を回避します。Docker社のサイトでも「Dockerはアプリケーションや自身の依存関係を含め、それらをソフトウェア開発における標準的なユニットにしたもの」と説明があります。

Dockerの扱うコンテナ環境(Dockerイメージ)は、実行が非常にスムーズです。詳しくは後述しますが、あくまでもDockerが実際に行う処理とは、隔離された環境でプロセスを起動するだけです。そのため、利用者からすると、コマンド実行後は瞬間的にアプリケーションやプログラムが起動するように見えます。これがDockerコンテナが速いと言われるゆえんです。

そのため、ちょっとしたコマンドやアプリケーションのテストを行いたいときDockerを使えば、わざわざOS環境からミドルウェアも含めてすべてを構築する必要はありません。使い終わったら簡単に環境を削除できるのも魅力の1つです。

Dockerイメージを共有するプラットフォームとして、Docker Hub[注5]の役割もDockerにおいて欠かせないものです。Docker Hubはパブリックレジストリと呼ばれており、公式のDockerイメージが提供されているだけでなく、GitHubのように、任意のイメージ登録・共有を行うことができるインターネット上のサービスです。イメージのダウンロードや検索、公開リポジトリの登録は、すべて無料で行うことができます。

そのほか、DockerとDocker HubはAPIを実装しています。このAPIを通せば、何らかの自動的な作業を行ったり、ほかのツールと連携した自動処理や、CI(継続的インテグレーション)やCD(継続的デリバリ)への応用も可能となります。

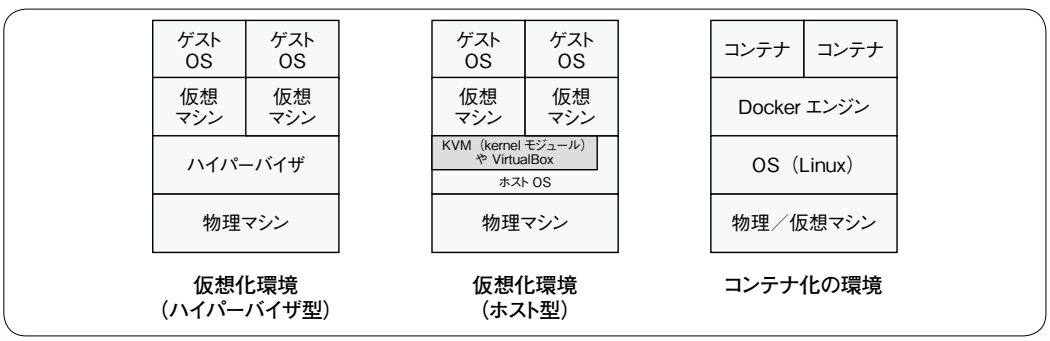
Dockerが解決しない問題

一見すると何でも便利に行えるかのように見えるDockerですが、よくある誤解は、既存の仮想化またはクラウド環境をおきかえるものだという考えです。Docker(Dockerエンジン)は、あくまでもLinuxカーネルの機能を使って、コンテナと呼ばれる各種リソースが隔離されたプロセス空間を作るためのものです。仮想化システムやクラウドで扱う仮想マシンは、ホスト側とは独立したリソースやOSを持ちます。一方、Dockerコンテナは OS側と同じkernelを使って動作する環境を作るものです(**図1**)。したがって、仮想化とDockerを使ってコンテナ化された環境は、しくみが違うために一概に比較できません。

Dockerで何らかのプロセスを実行したい場

注5) https://hub.docker.com

▼図1　仮想化とDockerの違い



合、仮想化やクラウド環境のように仮想マシン環境を用意する必要がありません。Dockerエンジンはプロセスを実行するにあたり、OSの環境内にユーザ空間と呼ばれる独立した環境を複数実行できます(**図2**)。

一方で、仮想マシンのように異なったOSでコンテナを動かすことはできません。たとえば、WindowsやMac OS上の環境でLinuxのコンテナは動きません。同様に、64bitのLinuxカーネル環境で動作するアプリケーションは、32bitの環境では動かないこともあるでしょう。

ほかにも、ChefやPuppetなどの構成管理ツールとDockerが比較されることもあります。これは、後の章で説明するDockerfileやDocker Composeを使うことで、Dockerイメージやコンテナ環境をコードで管理できるからです。DockerfileやComposeの役割は、あくまでコンテナのイメージやコンテナ間の情報を定義するものであり、、インストールや設定変更を行うための構成管理ツールとは役割が少し違います。すでに構成管理ツールを使っているのであれば、これまでの知見を活かしてコンテナ内の環境構築に活かすのも方法の1つです。そもそも、Dockerが動作しない環境では、やはり従来どおり、構成管理ツールを使うほうが便利な場合もあると思われます。

ほかにもDockerが解決しない問題としては、コンテナが動作する複数マシンのクラスタ管理とスケジューリングが挙げられます。複数のホストにまたがるクラスタ上で、どのようにコンテナを実行すべきかは、Docker単体では解決できません。そのためDocker単体では、ロードバランサや仮想ネットワークなどを扱う機能は提供されていません。

クラスタ管理には、ほかのツールと連携する必要があります。具体的には、後の章で扱うDocker Swarmや、Kubernetes、Apache Mesos、Nomadなどのツールを使うか、Amazon EC2 Container Service (ECS)などのクラウド事業者の提供するサービスを使う方法があります。

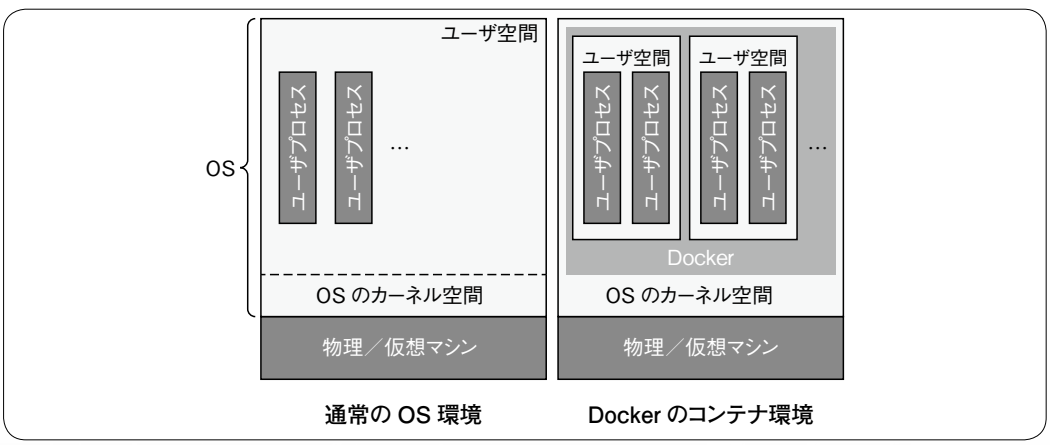
Dockerのアーキテクチャ

DockerとLinuxコンテナの違い

混同されがちですが、Dockerとは「コンテナ」のことではありません。Dockerはさまざまなツールやサービスの総称です。その中でもDockerエンジンが、コンテナを配布・実行するためのプラットフォームの役割を持っています。

それではDockerコンテナの「コンテナ」とは何でしょうか。「Dockerコンテナ」はDockerがさまざまなLinuxカーネルの機能を扱い、複雑な設定を抽象化してコンテナを作り上げたもので

▼図2　通常のOSとDockerの違い



す。つまり、ここでのコンテナとは、Linuxカーネルが提供する次の2つの機能を使い、プロセス、ファイルシステム、ネットワークを分離する環境のことを指します。

・Control Groups（cgroups）

特定のプロセスに対してリソースの上限を設定するための機能です。Dockerはこの機能を使い、すべてのコンテナに対するメモリやCPUリソースを管理します。また、コンテナのリソースの使用状況を把握するdocker statsコマンド使用時にも利用します。

・Namespaces（名前空間）

コンテナごとにリソースを分離する機能であり、コンテナ中では自分自身のリソースしか見えません。リソースにはmount（ファイルシステム）、UTS（コンテナ自身がホスト名、ドメイン名を持つ）、IPC、PID、ネットワーク、ユーザ（UID・GID）があります。

これらの技術を使うので、コンテナの中では通常のファイルシステムやネットワークが存在しているように見えます。また、複数のコンテナが起動したとしても、各々は独立して見えますし、コンテナの中からホスト側の情報を見ることもできません。ただし、OS側とLinuxカーネルは常に共有している状態です。

このコンテナと似ている、従来からある手法がchrootです。chrootはプロセスとファイル空間を分離するために使う機能です。Dockerも隔離された環境においてアプリケーションを実行していますが、カーネルの機能を使ってリソース管理なども含めて「コンテナ」として扱えるようにしています。

Dockerのコンテナと同じような処理は、Linuxコンテナ（LXC）を使っても実現できます。LXCもLinuxカーネルが持つコンテナ関連機能を使うためのライブラリやツール群です。LXCは2008年から開発が始まっており、2014年にLXC 1.0がリリースされました。これはDockerとは異なったコンテナ管理のアプローチです。

Dockerもバージョン0.9まではLXCを使っていましたが、以降のバージョンではlibcontainerというGo言語で書かれた独自ライブラリを使ってLinuxカーネルを操作しています。なお、現在もDockerデーモン起動時のオプションで、LXCドライバを使うことはできます。

コンテナやLXCについては、次の資料が参考になります。

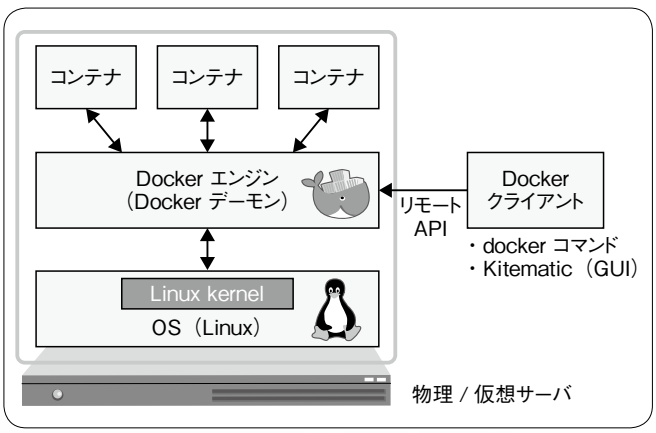
・Dockerを支える技術（中井悦司氏）[注6]
・Linuxコンテナと LXC 入門（加藤泰文氏）[注7]

サーバ／クライアント型のアーキテクチャ

Dockerはサーバ／クライアント型のシステムです（**図3**）。Dockerエンジンは Linuxカーネルのコンテナ機能の制御や、Dockerイメージを操作する役割があります。いわば、Dockerエンジンが、プラットフォームにおける中心的な役割を果たします。このDockerエンジンの実体こそが、サーバ上で稼働しつづけるDockerデーモンです。

注6） http://www.slideshare.net/enakai/docker-34668707
注7） https://speakerdeck.com/tenforward/1st-kistudy

▼図3　Dockerはサーバ／クライアント型



デーモンはTCPポートまたはUNIXドメインソケットをオープンし、Dockerクライアントからの要求を受け付けます。クライアントは、Dockerデーモンに対してAPIを通したリクエストを行います。クライアントには、コマンドラインで利用可能なdockerコマンドと、GUIを備えるKitematic(WindowsおよびMac OS向け)が提供されています。

実際にコンテナを使うには、利用者がクライアントを使ってデーモンに対してさまざまな命令を送ります。たとえば、コンテナを使うためにはdocker runコマンドを実行し、ホスト上で動作しているデーモンに対して、コンテナの実行を命令します。デーモンとクライアントは同じ環境だけに限定されません。クライアント側の設定により、命令先のホストを任意に切り替えることもできます。

Dockerイメージ

Dockerの概念の中でも独特なのは、コンテナを実行するときに必要となるDockerイメージです。これは仮想化やクラウド環境における仮想マシンイメージとは異なります。仮想マシンの場合は、ブート可能なOSが含まれる単一ファイルのことであり、ファイルの形式や環境によって互換性がない場合があります。

Dockerイメージには、コンテナを実行するときに使うためのすべての要素が含まれています。たとえばファイル群です。アプリケーションや何らかのサーバを動かすためには、バイナリファイルやライブラリ、設定ファイルなどが必要となります。イメージの中には、ほかにも外部に公開するポート番号の情報や、コンテナとして標準で実行されるべきコマンドの情報も含みます。

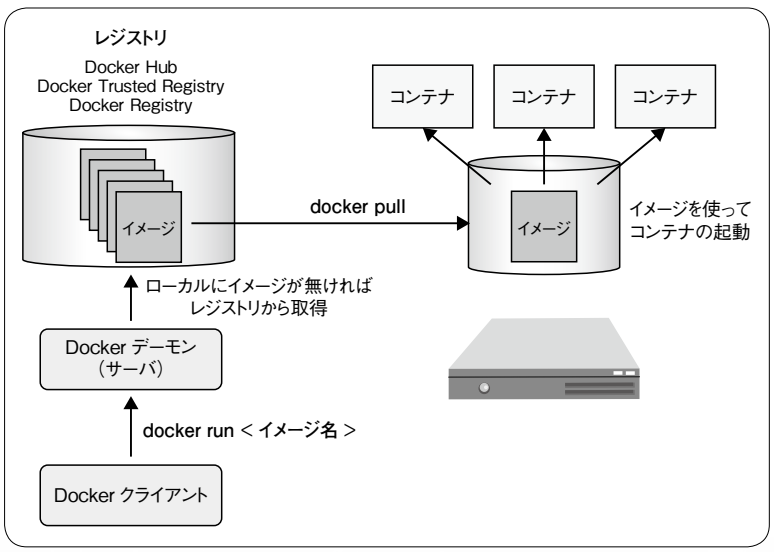
イメージ管理とレジストリの役割

Dockerがコンテナを起動するまでの流れを整理しましょう。Dockerでコンテナを実行する前に、まずDockerイメージを用意する必要があります。通常は、Docker Hub[注8]でパブリックに公開されているイメージをDockerの実行環境にダウンロードします。このイメージは、自分で作成することもできます。

Dockerクライアントからコンテナを実行しようとすると、まずローカル環境上に実行しようとするイメージが存在しているかどうか確認します(**図4**)。もしイメージがなければ、Dockerデーモンは Docker Hubから自動的にイメージのダウンロードを試みます。イメージの取得後、コンテナを起動します。

なお、イメージはダウンロードするだけでなく、リモート環境上にアップロードすることもできます。このイメージを取得・保管しておく場所が、レジストリです。レジストリには3種類あります。一般的にパブリックに公開されている

▼図4 コンテナの起動とイメージの関係



注8) https://hub.docker.com

Docker Hubでは、他人とイメージを共有するだけでなく、自分や特定のグループだけが参照できるプライベートな環境を持てます。あるいは、自分自身でサーバ環境を利用して使えるDocker Registry(レジストリ)と、Docker Trusted Registry(トラステッドレジストリ)があります。詳細は第2章で紹介します。

Dockerの環境構築

Dockerを使うにはDockerデーモンとクライアントの準備が必要です。用意する環境は、OSごとに異なります。

- **Linux**
  ⇒Docker社が提供しているセットアップ用スクリプトやパッケージを使う方法
  ⇒ディストリビューションが提供しているパッケージを使う方法
- **WindowsまたはMac OS X**
  ⇒Docker Machine(boot2docker)を使い、VirtualBoxの仮想環境を準備する方法

Linux環境へ準備する方法

LinuxでDockerを使い始めるのは、最もシンプルな方法です。物理・仮想化の環境を問わず、Dockerの動作環境を準備できます。ディストリビューションごとにさまざまなセットアップ方法があります。

- インストール用のスクリプトを使う方法
- Docker社が提供しているバイナリパッケージを使う方法
- ディストリビューションごとに提供されているバイナリパッケージを使う方法

インストール方法によりバージョンが異なる場合があります。この違いとは、誰がメンテナンスをしているかです。Docker社のスクリプトおよびバイナリは、Docker社自身によってメンテナンスされており、常に最新の安定版を利用できます。一方、ディストリビューション版は各々のコミュニティや提供会社によってサポートされています。そのため、最新版への対応は時間がかかりがちです。

Docker社としては、推奨すべきディストリビューションはありません。どちらを使うかは、最新版の必要性や、ディストリビューションによるサポートの有無をもとにご検討ください[注9]。

以降では、Docker社が提供する最新安定版をセットアップする方法を見ていきます。

動作環境

Dockerの動作のためには、Linux上に次の環境が必要です。

- iptables 1.4以上
- Git 1.7以上
- procps(またはpsコマンドを実行可能なもの)
- XZ Utils 4.9以上
- cgroupsの階層を参照可能なこと

Linuxカーネルは3.10以上が推奨されています。ですが、ディストリビューションによってはサポートされておらず、導入によってOSのサポートが受けられなくなる場合がありますので、ご注意ください。

インストールスクリプトの使用

Ubuntuなどの対象OS[注10]であれば、スクリプトを使ったセットアップが最も手軽です。DockerとDockerに依存関係のあるパッケージを自動的にセットアップします。

```
# curl -sSL https//get.docker.com | sh
```

正常終了すると、画面にインストールしたDockerデーモンとクライアントのバージョン情

注9) 原稿執筆時点では、商用サポート(CS)版のDocker動作環境は、Red Hat Enterprise Linux 7およびUbuntu 14.04 LTSです。
注10) Ubuntu、Fedora、Gentoo、Debian、CentOSなどの主要ディストリビューションに対応しています。

報が表示されます。以後はすぐに利用できます。docker versionコマンドで再度バージョン番号を確認できます。

公式docker-engineパッケージを使う方法

自動インストール用のスクリプトを使わず、手動でパッケージを確認しながら作業を進めることができます。それはDocker社がメンテナンスしているリポジトリを使い、docker-engineパッケージを使う方法です。ただし、各ディストリビューションが提供しているパッケージとは内容やバージョンが違うため、注意が必要です。

・Ubuntu/Debianの場合

```
$ apt-key adv --keyserver hkp://p80.
pool.sks-keyservers.net:80 --recv-keys
58118E89F3A912897C070ADBF76221572C52609D
$ vi /etc/apt/sources.list.d/docker.list
```

ファイル中にリポジトリと対象環境のディストリビューションを記述します。以下はDebian Wheezyの場合です。

```
deb https://apt.dockerproject.org/repo
debian-wheezy main
```

Ubuntuの場合は、次のように指定できます。

```
deb https://apt.dockerproject.org/repo
ubuntu-precise main
```

利用可能なリポジトリは、http://apt.dockerproject.org/repo/dists/ をご確認ください。

パッケージをインストールするには、apt-get install docker-engineを実行します。

・CentOS/Fedoraの場合

図5の「centos/7」にあたるディストリビューション名やバージョンは、自分の環境に合わせて書き換えてください。

インストールするには、yum install docker-engineを実行します。

WindowsまたはMac OS X構築方法

現時点で選択肢は2つあります。ローカルに環境を構築するのであれば、Docker Toolboxを使う方法が、現時点で推奨されています[注11]。あるいは、仮想化・クラウド上にLinuxをセットアップし、その上でDockerを入れる方法もあります。

あるいは、常にクラウドの利用が前提であれば、Docker Machineのバイナリのみをセットアップする方法もあります。

Docker Toolboxを使う方法

Docker Toolboxは次のものがパッケージ化されたセットアップ用のツールです。

- Docker Machine
- Docker Compose（MacOS Xのみ）
- Kitematic
- Virtualbox

Docker Machineは、さまざまな仮想化環境／クラウドに対してDockerの実行環境を自動的に構築するためのツールです。デフォルトでは、Toolboxに梱包されている、VirtualBox用のドライバを使用します。コマンドdocker-machineを使うと、自動的にDockerがインストールされた環境を手軽に構築したり、使い終わったあとは削除したりできるため、便利です。

注11）以前はboot2dockerというツールが提供されていました。Docker Machineはboot2dockerの機能を拡張し、汎用性を高めたものです。

▼図5　CentOS/Fedoraのリポジトリ追加

```
# cat >/etc/yum.repos.d/docker.repo <<-EOF
[dockerrepo]
name=Docker Repository
baseurl=https://yum.dockerproject.org/repo/
main/centos/7
enabled=1
gpgcheck=1
gpgkey=https://yum.dockerproject.org/gpg
EOF
```

Docker Toolboxのセットアップ

Docker Toolboxを使った環境に対応しているのは、次の環境です。

・Windows 7.1、8、8.1[注12]
・Mac OS X 10.8以上

ダウンロード用のページ[注13]にアクセスし、自分の環境にあった「Download」のリンクをクリックします。ファイルのダウンロードが終わったら、インストーラを起動し、Docker Toolboxのセットアップを進めます。「セットアップ後は、メニューからDocker Quickstart Terminal」を実行します。内部でDocker Machineを使い、Dockerが動作するLinux仮想マシンを起動します。

Dockerの基本操作

Dockerを使ったコンテナの起動

Dockerの実行環境を整えたあとは、実際にコンテナを起動してみましょう。コンテナの起動には、Linux環境上にあるDockerエンジンを使います。コンテナを起動するにはdocker runコマンドを実行します。ここではテスト用の「hello-world」コンテナの起動を試みます。

```
$ docker run hello-world
```

コンテナの起動や停止／再起動というと、仮想マシンの操作を想像されるかもしれません。しかし、実際には通常のプロセスを起動するのと同様の操作であり、そこにコンテナ独特の機能が付加されているものです。

初回実行時は、ローカル環境上に「hello-world」イメージが存在しません。そのため、Docker Hubからイメージのダウンロード(pull)を自動的に行います(**図4**参照)。ダウンロードが終わったら、その後、自動的にコンテナを起動します。正常に実行されると、次のようなメッセージを表示して、コンテナは停止します。

```
Hello from Docker.
This message shows that your ➪
installation appears to be working ➪
correctly.
```

次はubuntuコンテナを起動し、bashを使って仮想サーバのように操作してみます。今度は、新しく2つのオプションを使います。

・-i …… 標準入力を受け付ける
・-t …… 疑似ターミナルを使用する

さらに、最後にコンテナの中で実行するコマンドを指定します。次のようにdocker runコマンドのオプションと、最後に/bin/bashを指定すると、bashのみ実行するコンテナ環境を開始します。/bin/bashを指定します。

```
$ docker run -i -t ubuntu /bin/bash
root@87c50014765b:/#
```

コマンド「docker run」を実行すると、Dockerエンジンは次の処理を行います。

①ubuntuイメージがローカルになければリモートのDocker Hubからイメージを取得
②イメージを使って、新しいコンテナを作成
③ファイルシステムを割り当て、新たに読み書き可能なイメージ層を追加
④コンテナがローカルホストと通信できるように、ネットワークとIPアドレスを割り当て
⑤指定したプロセス/bin/bashを実行
⑥プロセスの標準入出力を画面に表示

コマンドを実行すると、「root@コンテナID」の形式でシェルが起動します。以降の操作は、コンテナの中で行います。コンテナは本章の冒頭で解説したように、隔離されたプロセス空間です。psコマンドを実行すると、bashのプロセスのPIDが1になっていることがわかります(**図6**)。

またdfコマンドを実行すると、ホスト側とは

注12) Windowsの場合は、ハードウェア側がBIOS設定で仮想化機能(Intel Virtualization Technologyなど)に対応している必要があります。
注13) https://www.docker.com/toolbox

異なるパーティション構成が見られます(**図7**)。

同様にネットワークも隔離されていることをifconfigコマンドで確認しましょう(**図8**)。こちらもホスト側と異なり、172.17.0.0/16のネットワーク範囲内でIPアドレスが自動的に割り振られていることが確認できます。

最後にコンテナを停止するには、exitコマンドを実行し、bashを終了します。このbashプロセスが停止すると、自動的にコンテナは停止状態となります。

```
root@87c50014765b:/# exit
exit
```

通常のプロセスと違うのは、コンテナを起動するたびに、都度、コンテナに対する新しいイメージ層(ファイルシステム)が自動的に作成されます。そのため、忘れがちですが、最後に使い終わったコンテナを削除する作業が必要です。これにはdocker runに--rmオプションを指定するか、コンテナ停止後にdocker rmで削除します。そうしない限りファイルは残り続けますので、注意が必要です。今回はdocker rm 87c50014765bと実行してコンテナを削除します。

◆ ◆ ◆

以上がコンテナを使った基本的な操作となります。具体的なDockerクライアントの操作方法は、次章で見ていきましょう。SD

▼図6 psコマンドでPIDを確認する

```
root@87c50014765b:/# ps -ef
UID        PID  PPID  C STIME TTY
          TIME CMD
root         1     0  0 07:52 ?        00:00:00 /bin/bash
```

▼図7 dfコマンドでパーティション構成を見る

```
root@87c50014765b:/# df
Filesystem                1K-blocks    Used Available Use% Mounted on
rootfs                     20511356 7073044  12373356  37% /
none                       20511356 7073044  12373356  37% /
tmpfs                        250896       0    250896   0% /dev
shm                           65536       0     65536   0% /dev/shm
tmpfs                        250896       0    250896   0% /sys/fs/cgroup
/dev/disk/by-label/DOROOT  20511356 7073044  12373356  37% /etc/hosts
tmpfs                        250896       0    250896   0% /proc/kcore
tmpfs                        250896       0    250896   0% /proc/latency_stats
tmpfs                        250896       0    250896   0% /proc/timer_stats
```

▼図8 ifconfigでネットワーク状況を確認する

```
root@87c50014765b:/# ifconfig
eth0      Link encap:Ethernet  HWaddr 02:42:ac:11:00:0f
          inet addr:172.17.0.15  Bcast:0.0.0.0  Mask:255.255.0.0
          inet6 addr: fe80::42:acff:fe11:f/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:8 errors:0 dropped:0 overruns:0 frame:0
          TX packets:9 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:648 (648.0 B)  TX bytes:738 (738.0 B)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

第2章

Dockerのイメージ管理と基本操作

レジストリとDockerクライアントの操作法

Dockerを使ったコンテナ操作の中心となるのがDockerイメージの管理です。本章はイメージの基本的な扱い方と、Docker Hubを含めたレジストリの利用方法、そして基本的なDockerクライアントの操作方法に対する理解を深めます。

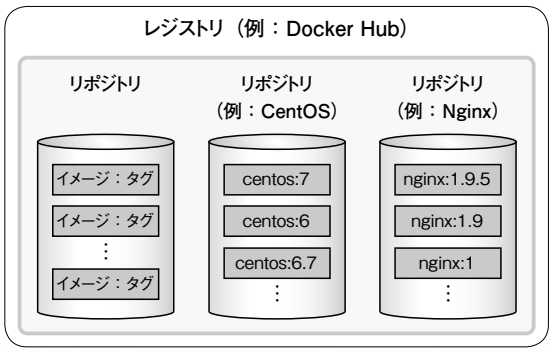
Author 前佛 雅人(ぜんぶつ まさひと)　クリエーションライン㈱　Twitter @zembutsu

Dockerイメージの基本概念

Dockerイメージとは？

Dockerでコンテナを作成／実行するときに必要になるのが、Dockerイメージです。このイメージとは、読み込み専用のテンプレートであり、コンテナを実行する際に必要なすべての情報(例：プログラムやライブラリなど)が格納されています。さらに、イメージの中に含まれるのは、「コンテナ実行時に何のコマンドを実行するか」、「どのポートを利用するか」、「どのボリュームをマウントするか」などの情報も含まれる場合があります。そのため、仮想化やクラウドにおける仮想マシンイメージに近い概念ですが、その実体は異なるものです。

▼図1　レジストリとリポジトリの関係



レジストリとリポジトリ

Dockerイメージを取得してコンテナとして利用するには、レジストリ(Registry)からイメージをダウンロード(pull)します。レジストリとは、Dockerイメージを格納しておく場所のことです。一番有名なレジストリはDocker Hub[注1]です。これはDocker社による公開リポジトリとして提供されており、誰でも利用できます。そのほか、自分が自由に使えるプライベートなレジストリを構築することもできますし、サポートを受けられるDocker Trusted Registryも利用できます。

各々のレジストリの中にリポジトリ(Repository)があります。このリポジトリには、さまざまなアプリケーションやミドルウェアなどの情報が集まっており、この中にDockerイメージが格納されています(**図1**)。

Docker Hubには公式(Official)リポジトリがあります(例：Ubuntu、CentOS、Nginx、MySQLなど)。これは公式のイメージが配布されています。公式という名前が付くとおり、Docker社による認証／精査済みのイメージであり、Docker Hub上で配布されています。ただし、あくまで場所が提供されているだけであり、イメージのメンテナンスやサポートを行うのはDocker社で

注1) https://hub.docker.com/

はなく、それぞれの公式リポジトリの提供者です。

Docker Hubでは公式リポジトリ以外にも、さまざまな方が作成したリポジトリからイメージをダウンロードできます。また、自分のリポジトリにアップロードしたイメージを共有することもできます。共有したくない場合は、プライベートリポジトリを作ることもできます。

イメージのバージョン管理

各イメージには、イメージ名とタグが割り当てられています。イメージ利用時にタグを指定しない場合は「latest」(最新)を指定したものとみなされます。

具体的な例を見ていきましょう。CentOSのコンテナを実行するにはdocker run centosと入力します。これはcentos:latestというイメージを指定したのと同義です。もしCentOSのバージョンを明示したいのであれば、docker run centos:6のように<イメージ名>:<タグ>の形式で入力する必要があります。

イメージの構造

Dockerイメージはイメージ層(image layer)の積み重ねにより構成されています(**図2**)。各イメージは、必ずベースイメージを持っています。公式リポジトリで配布されているイメージは、公式のベースイメージ(UbuntuやCentOSなど)を利用しています。一方、自分で利用する場合は、自分でベースイメージを作成することもできます。

また、複数のコンテナで共通しているイメージ層を利用している場合、Dockerはイメージ層を共有して利用します。たとえば、共通のUbuntuベースイメージを利用しているDockerイメージを使いたいとき、すでにローカルにベースイメージをダウンロード済みであれば、Dockerはあらためてベースイメージをダウンロードしません。このしくみのため、迅速なイメージのダウンロードや、ディスク容量の節約を実現しています。

イメージはすべて読み込み専用として配布されます。コンテナとして実行するときに、初めて書き込み可能な層が割り当てられます。そのため、コンテナ内で何らかの処理を行い、設定を反映したい場合にはコミット(docker commit)を必ず行わなくてはいけません。また、docker runを繰り返す度に、コンテナ内のファイル階層が初期化されるように見えるのは、その都度、新しい書き込み可能な層が割り当てられているためです。

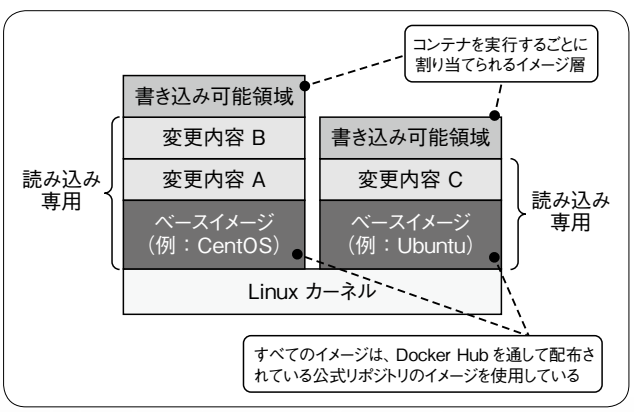
イメージの構築方法

Dockerイメージを構築するには3つの方法があります。

- コンテナの変更内容を新しいイメージとしてコミット(docker commitコマンド)
- Dockerfileをもとにイメージを構築(docker buildコマンド)
- Dockerにtar形式のバイナリを取り込みイメージ作成(docker importコマンド)

開発環境で頻繁に使用されるのが、docker commitコマンドを使ってコミットする(内容を確定)方法です。開発が完了しており状況を再現するために使われるのがDockerfileを使ってビルドする方法です。Docker

▼図2 イメージ層の構成とコンテナ



Hubで配布されている各公式イメージは、Dockerfileも配布されていますので、どのような過程でイメージを構築したのかを確認できます。

Dockerイメージの管理と配布

Dockerイメージを保管する場所をレジストリと呼ぶと説明しましたが、Docker社によるレジストリは、ライセンスや利用形態によって3つに分類されます(**表1**)。どの場合も、イメージの送信手順は共通しています。まずはコンテナに対する変更内容をdocker commitで確定し、docker tagでタグ付け(名称変更)を行い、docker pushでイメージを送信します(**図3**)。

Docker Hub (パブリックレジストリ)を使う方法

前述のとおり、Docker Hubは誰もが利用可能なパブリックレジストリの位置づけです。公式リポジトリを通して、各種の公式イメージが配布されているだけはありません。アカウント登録を行えば、自分の作成したイメージを登録・共有できます。

Docker Hubの利点としては、自分や特定グループのみ参照可能なプライベートレジストリが利用可能なことです。1つだけなら無料ですが、5つまでは7ドル/月額、以降はリポジトリの数に応じて課金されます[注2]。また、リポジトリごとにWebhookを指定できます。そのため、イメージ内容の更新をトリガとして、GitHubや外部のサービスと連携する使い方もできます。

以降では、実際にアカウントを作成し、自分で編集したbusybox[注3]コンテナのアップロードを行ってみます。

Docker Hubの登録

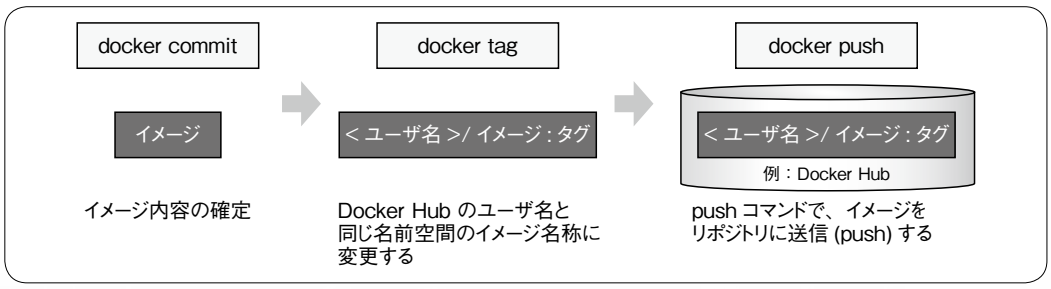
アップロードのためには、アカウントの登録が必要です。登録は無料で行えます。Docker Hubのサイト[注4]に移動します。**図4**の画面上「user name」(登録した名前は、自分のリポジトリ名としても利用されます)、「email」、「password」各項目を入力し、「Sign Up」ボタンを押します。登録確認用のメールが届きますので、メール本文

注2) Docker Hub 料金表 https://www.docker.com/pricing
注3) https://hub.docker.com/_/busybox/
注4) https://hub.docker.com/

▼表1　3つのレジストリの比較

レジストリ	説明	イメージ管理	GUI	認証	リソース表示	ログ	TLS	サポート	動作環境	料金
Docker Hub	誰もが利用可能	○	○	○	×	×	○	△	Docker動作環境	無料～
Docker Registry	ローカルにレジストリを準備	○	×	×	×	×	△	×		-
Docker Trusted Registry		○	○	ユーザ認証 LDAP認証	○	○	○	○	RHEL 7.0,7.1、Ubuntu 14.04 LTS	1 Registry、10 Engine 150ドル/月～

▼図3　リポジトリへイメージをpushする流れ



のURLをクリックすると登録が完了します。

リポジトリの作成

イメージをDocker Hubにアップロードする前に、イメージ用のリポジトリを新規に作成します。作成には、Docker Hubにログインし、画面右上にある「Create Repository」（リポジトリ作成）ボタンをクリックします。その次の「Create Repository」画面（**図5**）では、次の項目を選択／入力します。

・ネームスペースの選択（必須）

通常は自分のDocker Hubユーザ名ですが、組織（Organization）に登録されている場合は、選択肢から選べます

・リポジトリ名（必須）

mybusyboxなど、任意のリポジトリ名を入力します

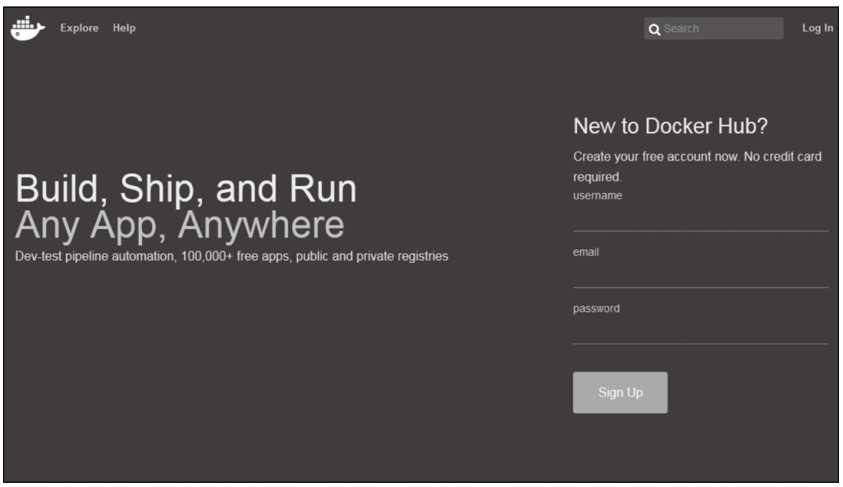
・短い説明（必須）

何のリポジトリなのか、説明を入力します

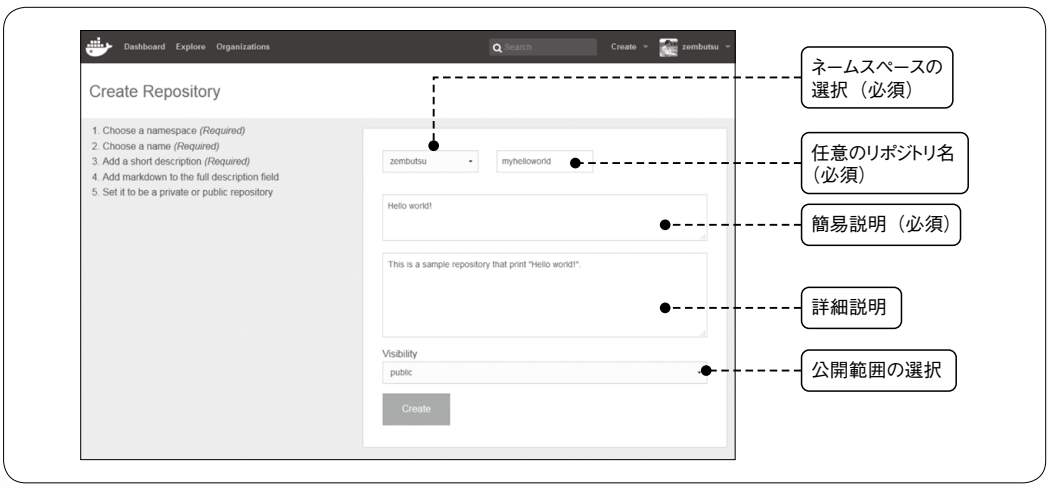
・詳細説明

通常のテキスト形式のほか、Markdown形式で記述することもできます

▼図4　Docker Hubログイン画面



▼図5　リポジトリの登録画面



・公開範囲
publicまたはprivateを選択します

入力後は「Create」ボタンを作成すると、画面が切り替わり(**図6**)、リポジトリ作成が完了したことがわかります。これがリポジトリの基本画面です。この画面から必要に応じて、説明を書き換えたり、Webhookの設定、リポジトリの削除を行います。

アップロード用イメージの作成

ここではbusyboxイメージをアップロードします。busyboxは公式イメージの1つで、容量が2.5MB程度の小さなLinuxディストリビューションです。

アップロードする前に、busyboxイメージを使ってコンテナを実行します。次のようにコマンドを実行するとbusyboxコンテナを起動し、コンテナ内のルートディレクトリにファイルを置きます。最後にexitを実行するとコンテナを終了します。

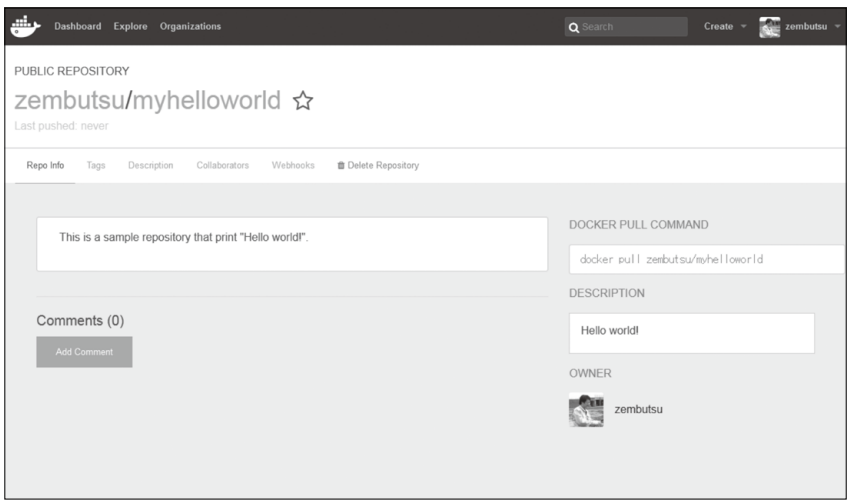
```
$ docker run -t -i busybox
/ # echo "Hello World" > hello.txt
/ # exit
```

次にイメージ内容をコミットします。docker ps -lで、最後に実行したコンテナのコンテナIDを確認し、commitコマンドで<Docker Hubユーザ名>/mybusyboxというイメージ名としてタグ付します。**図7**の例ではコンテナIDは7ab41e8ac9ffですが、実際の出力に合わせて書き換えます。

作成したイメージの中に、hello.txtが作成されていることを確認しておきます。

```
$ docker run -ti zembutsu/mybusybox ↵
cat /hello.txt
Hello World
```

▼図6　作成したリポジトリ



▼図7　コンテナIDを指定してイメージをコミットする

```
$ docker ps -l
CONTAINER ID        IMAGE               COMMAND             CREATED             ↵
STATUS                      PORTS               NAMES
7ab41e8ac9ff        busybox             "/bin/sh"           27 seconds ago      ↵
Exited (0) 5 seconds ago                        pensive_stallman
$ docker commit 7ab41e8ac9ff zembutsu/mybusybox
a8a72c90493844ba589c7e94dea4c114ba9877efe4d6fb4267c95cd22892a5a2   ← 新しく作成したイメージID
```

これで準備が整いました。次はDocker Hubに作成したイメージをアップロードします。

なお、この時点で不要なコンテナを削除しておきます。正確には、コンテナを実行するときに自動的に作成された、新しいイメージ層です。docker ps -aコマンドを実行すると、停止中のコンテナ情報がすべて表示されます。これらは再利用しませんのでdocker rm <コンテナID>コマンドを実行して削除しておきます。

Docker Hubにアップロード

アップロードする前に、docker loginコマンドを実行し、認証を行います(**図8**)。認証に成功すると、Docker Hubのトークンが発行され、~/.docker/config.jsonに保管されます。認証を解除するにはdocker logoutコマンドを実行します。

次にdocker pushコマンドでイメージをDocker Hubに送信します。このコマンドを実行すると、ローカルに保管されているイメージを送信します(**図9**)。

それでは、ローカルのイメージを削除して、Docker Hubからダウンロードしなおしてみましょう。イメージを削除するにはdocker rmiコマンドを実行します(**図10**)。

これでローカルのイメージは削除されました。

▼図8　docker loginによって認証を行う

```
$ docker login
Username: zembutsu      ← Docker Hub のアカウント
Password:               ← パスワード
Email: foo@example.jp   ← 登録メールアドレス
WARNING: login credentials saved in /home/zem/.docker/config.json
Login Succeeded
```

▼図9　docker pushでイメージをDocker Hubに送信する

```
$ docker push zembutsu/mybusybox
The push refers to a repository [docker.io/zembutsu/mybusybox] (len: 1)
a8a72c904938: Image successfully pushed
0f864637f229: Image successfully pushed
79722f6accc3: Image successfully pushed
cf2616975b4a: Image already exists
latest: digest: sha256:1b7fe4531632d6cab92adb0d80bf5def056a62185c124955bb83eef26de06000 ↵
size: 6507
```

▼図10　docker rmiでイメージを削除する

```
$ docker rmi zembutsu/mybusybox
Untagged: zembutsu/mybusybox:latest
Deleted: a8a72c90493844ba589c7e94dea4c114ba9877efe4d6fb4267c95cd22892a5a2
```

▼図11　docker runでコンテナを実行する

```
$ docker run -ti zembutsu/mybusybox cat /hello.txt
Unable to find image 'zembutsu/mybusybox:latest' locally   ←ローカルにイメージがない
latest: Pulling from zembutsu/mybusybox                    ←最新のイメージを取得(pull)
a8a72c904938: Pull complete
cf2616975b4a: Already exists
79722f6accc3: Already exists
0f864637f229: Already exists
Digest: sha256:1b7fe4531632d6cab92adb0d80bf5def056a62185c124955bb83eef26de06000
Status: Downloaded newer image for zembutsu/mybusybox:latest
Hello World
```

次に再びdocker runコマンドを使ってコンテナを実行します(**図11**)。今度はローカルにイメージがありませんので、自動的にDocker Hubからイメージを取得したあと、実行します。

今回はパブリックリポジトリに登録しました。そのため、自分以外の誰でもダウンロードできますし、イメージの検索対象に含まれますので注意しておく必要があります。

不要になったリポジトリは削除できます。Docker HubのGUIにログインし、対象のリポジトリを開きます。メニューの「Delete Repository」(リポジトリの削除)をクリックし、確認メッセージの「Delete」をクリックするとリポジトリの削除が完了します。

このように便利なDocker Hubですが、使うためには必ずインターネットに接続可能な環境が必要です。ローカルの閉じた環境内や、自分が持っている環境上でリポジトリを使うには、次に紹介するレジストリを使う方法があります。

Docker Registryを使う方法

Docker Registry(レジストリ)とは、Docker Hubを使わずに自分でリポジトリを管理できるレジストリのことです。イメージの保管場所には、ローカルのディスクを使うこともできますし、Amazon S3 や Azure、Google Cloud Storage、OpenStack Swiftなどのストレージにも対応しています。

Docker RegistryはDocker Hub上で配布[注5]されており、誰でも自由に利用できます。使用するにはDocker Hub上のイメージを使う方法と、ソースコードから構築する方法があります。

開発はGitHubのコミュニティ[注6]を通して行われています。ただし、サポートなどはありませんし、Docker HubのようなGUIも準備されていません。サポートや高度な機能が必要な場合はDocker Trusted Registryをご検討ください。

ローカルにイメージを保管するregistry環境

Docker Registryをコンテナとして動かすには、Dockerが動作している環境が必要です。実行するためにはdocker run -p 5000:5000 registryのように実行します。ここでは-pオプションを使い、コンテナ内のポート5000をホスト側のポート5000に割り当てています。実際には、次のようにコンテナをデタッチドモードで実行する-dオプションも使用します。

```
$ docker run -d -p 5000:5000 registry:2.0
```

docker psコマンドを実行すると、**図12**のようにコンテナが動作していることがわかります。

このRegistryにイメージを送信するには、docker pushコマンドを使います。イメージを送信する前に、docker tagコマンドを使い、イメージ名をタグ付しておく必要があります。

```
$ docker tag <イメージID> <ホスト名:5000>
/<イメージ名>:<タグ>
```

ホスト名の個所は、IPアドレス(パブリックまたはプライベート)もしくは名前解決が可能なホスト名を指定する必要があるので注意が必要です。

注5) https://hub.docker.com/r/library/registry/

注6) https://GitHub.com/docker/docker-registry

▼図12　docker psでコンテナの動作を確認する

```
$ docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED
STATUS              PORTS                    NAMES
5383bf7675b3        registry:2.0        "registry cmd/registr"   3 seconds ago
Up 2 seconds        0.0.0.0:5000->5000/tcp   clever_torvalds
```

```
$ docker tag <イメージ名> localhost:5000/myapp:1.0
```

実行したあとは、docker pushコマンドでイメージを送信します(**図13**)。

Registryにどのようなタグが登録されているかを調べるには、curlなどでAPIの情報を取得する必要があります。

```
$ curl http://localhost:5000/v2/myapp/tags/list
{"name":"myapp","tags":["1.0"]}
```

イメージを取得するのは、Docker Hubと同じようにdocker pullコマンドを使用します(**図14**)。

このとき、リモートの環境からイメージの取得をする場合、Dockerデーモンのオプションで--insecure-registry <ホスト名/IPアドレス>:5000のオプションが必要になります。接続先がローカル環境(127.0.0.0/8)の場合のみ、この設定は不要です。

なお、Regisryの環境は動作するポート番号さえわかっていれば、どのような環境からもアクセスができます。ローカル環境ならまだしも、インターネットに公開された環境で使う場合には十分な注意が必要です。

Docker Trusted Registry

Docker Trusted Registry(以下DTR)は、自分のインフラ上で安全にDockerイメージを保管するためのリポジトリです。基本構造はDocker Registryと同等ですが、次の機能拡張が行われています。

- Webで設定可能なGUIやダッシュボード
- 認証機能(パスワード認証、LDAP認証)
- SSL証明書を使った暗号化のサポート
- ログの記録・監査機能

DTRを使うには、ライセンスの取得が必要です。評価用として、30日間無償利用可能なライセンスが配布されています[注7]。

なお、DTRは通常のDockerエンジンと異なり、商用サポート版のDockerエンジンが提供されています。動作環境は現時点でRed Hat Enterprise Linux 7.0、7.1、Ubuntu 14.04のみです。SD

注7) https://hub.docker.com/enterprise/

▼図13 docker pushでイメージを送信する

```
$ docker push localhost:5000/myapp:1.0
The push refers to a repository [localhost:5000/myapp] (len: 1)
8c2e06607696: Image successfully pushed
6ce2e90b0bc7: Image successfully pushed
cf2616975b4a: Image already exists
1.0: digest: sha256:9215bd2c4aa49481dd66881d03c69ab3909e8aa384d3c2ccd974b4c35d1b8ea0 size: 5047
```

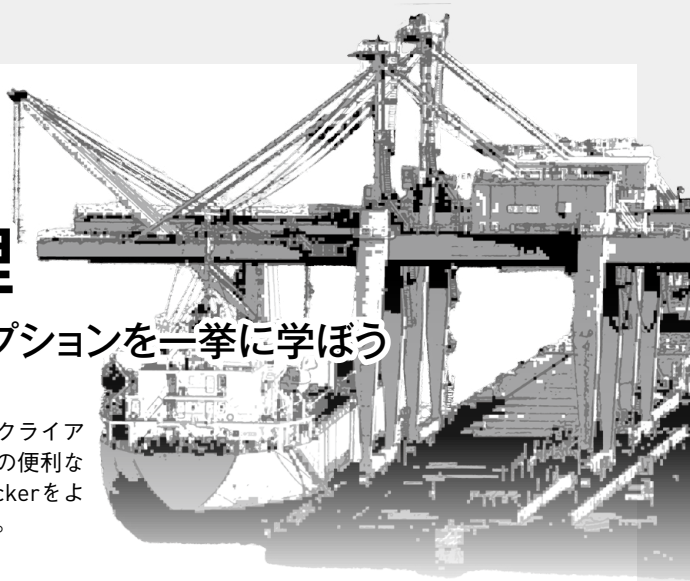
▼図14 docker pullでイメージを取得する

```
$ docker pull localhost:5000/myapp:1.0
1.0: Pulling from myapp
cf2616975b4a: Already exists
6ce2e90b0bc7: Already exists
8c2e06607696: Already exists
Digest: sha256:9215bd2c4aa49481dd66881d03c69ab3909e8aa384d3c2ccd974b4c35d1b8ea0
Status: Image is up to date for localhost:5000/myapp:1.0
```

第3章

Dockerの操作と管理

便利なコマンド／オプションを一挙に学ぼう



Dockerを日常的に使ううえで欠かせないのが、クライアントの操作方法とデーモンの管理です。コマンドの便利な使い方やオプションの指定方法を学ぶことで、Dockerをより便利に使いこなせるようになっていきましょう。

Author 前佛 雅人（ぜんぶつ まさひと） クリエーションライン㈱ Twitter @zembutsu

Dockerクライアントでコンテナ操作

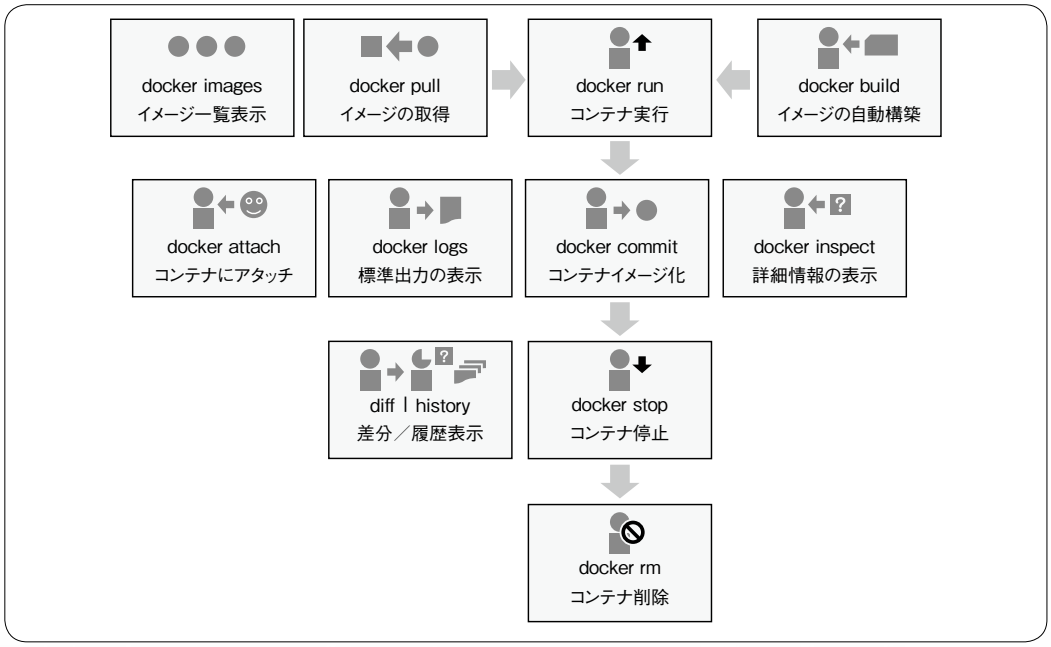
Dockerコンテナに対する操作は、**図1**のようなライフサイクルを持ちます。基本的にコンテナを実行して停止するまでの流れは、通常のLinux上のプロセスと何ら変わらないものです。そこに、Docker独特の要素として、イメージの取得／構築といったコマンドや、イメージの内容を確定するコマンド、差分／履歴を確認する管理用のコマンドが準備されています。

コンテナの実行と停止

コンテナを操作するには、dockerコマンド（**表1**）で特定のコンテナに対して命令を出します。ここでは各コマンドの詳しい動きを見ていきます。

コマンドdocker runはコンテナを起動します。正確には特定のプロセスを独立した（隔離された）コンテナとして実行するためのコマンドで

▼図1 おもなdockerコマンドとコンテナのライフサイクル



す。第1章で学んだように、プロセスを稼働するとともに、独立したファイルシステム、ネットワーク、プロセスツリーを持ちます。コンテナ起動時に、どのDockerイメージを使い、どのようなコマンドを実行するか、どのポートを公開するかなどの指定を行います。

書式
```
$ docker run [オプション] イメージ名[:tag] [コマンド] [引数]
```

Ubuntuイメージを使い/bin/bashを実行するには、次のようにコマンドを入力します。

```
$ docker run -i -t ubuntu:latest /bin/bash
root@30edd8bb4427:/#
```

コマンドを実行するとプロンプトがroot@30edd8bb4427:/#に切り替わり、コンテナの中でbashを操作していることがわかります。コンテナの中でps -efなどのコマンドを実行すると、自身のPIDが「1」として実行していることが確認できます(**図2**)。

プロンプト上のホスト名にあたる個所に表示されている英数字30edd8bb4427はコンテナIDであり、コンテナを実行するたびにランダムで割り振られるものです。コンテナを停止するときや、各種操作を行うときに、このコンテナIDを使って命令します。もう少し詳しく説明すると、各コンテナは64桁のロングIDと呼ばれるコンテナIDを持っています、そのうち先頭から12桁だけのものをショートIDと呼び、Dockerのコマンド実行時に多用します[注1]。

なお、先の例ではubuntuイメージのタグにlatestを指定しています。もしタグを指定しないと、自動的にlatestが割り当てられます。たとえば、centosのイメージを指定するときにタグを指定しないと、自動的にcentos:latestになります。これは、現時点ではcentos:7を指定するのと同じことになるため注意が必要です。

ターミナルを操作、exitを実行すると、実行しているbashプロセスを終了するとともに、コンテナそのものも終了状態となります。コンテナのプロセスを停止せずにホスト側に戻りたい場合は、コンテナ内のターミナル上でCtrl+P+Qキーを同時に押します。

▼表1　dockerコマンド(実行と停止にかかわるもの)

コマンド	説明
docker run	コンテナの起動
docker ps	コンテナの一覧表示(実行中／停止中を含む)
docker attach	コンテナにアタッチ
docker exec	コンテナ内でコマンドの実行
docker logs	コンテナのログ(標準出力)を表示
docker stop	コンテナの停止(SIGTERMシグナル送信)
docker kill	コンテナの停止(SIGKILLシグナル送信)
docker start	停止したコンテナの再起動
docker restart	コンテナを停止したあとで再起動

▼図2　コンテナの中におけるpsコマンドの実行例

```
root@30edd8bb4427:/# ps -ef
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0 02:22 ?        00:00:00 /bin/bash
```

▼図3　docker psの実行例

```
$ docker ps
CONTAINER ID   IMAGE   COMMAND                  CREATED         STATUS         PORTS                     (略)
3c2ad14fc213   nginx   "nginx -g 'daemon off"   2 seconds ago   Up 2 seconds   0.0.0.0:32769->80/tcp,(略)
0e37159335c5   ubuntu  "/bin/bash"              2 hours ago     Up 2 hours                               (略)
```

コンテナの状態を確認

コンテナが稼働しているか停止しているか、どのようなコマンドを実行しているかなどを確認するにはdocker psコマンドを実行します。 コマンドを実行すると、コンテナID、利用しているイメージ名、コンテナの中で実行されているコマンド、作成した時間、状態、ポートの割り当て状況を確認できます(**図3**)。

注1) dockerコマンドでコンテナを指定するとき、ロングIDとショートIDのどちらも指定できます。正確には、IDが前方一致するコンテナを操作します。

docker psコマンドはプロセスが実行中のものだけを表示します。便利なオプションは**表2**のとおりです。これらは組み合わせて利用可能です。たとえば、最後に実行したコンテナのIDのみを表示するには、次のように実行します。

```
$ docker ps -lq
30edd8bb4427  ←コンテナID
```

コンテナをバックグラウンドで実行

コンテナを対話型ではなく、バックグラウンドでデーモンとして動かすには、docker run -dを使用します。これをDockerでは「デタッチド・モード(detached mode)で実行する」と言います。たとえば、pingコマンドをデタッチド・モードで実行するには、**図4**のように実行します。画面にはコンテナIDが表示されるだけで、実行結果は表示されません。このときdocker logs <コンテナID>コマンドを実行すると、実行中のpingコマンドの状況が表示されます。logsはコンテナ内のログ(出力)を表示して終了します。コマンドの実行結果を見続けたい場合はdocker logs -f <コンテナID>を実行すると、処理が終わるまで表示されるので便利です。

デフォルトではコンテナを実行してからすべてのログを表示します。表示量が多い場合は--tail <行数>オプションを使うことで、直近のログのみを表示します。次の例は直近の10行以降のログを表示し続ける指定です。

書式

```
$ docker logs --tail 10 -f <コンテナ>
```

次に、ログの表示を停止したい場合はCtrl＋Cを実行します。

ログを画面に表示するだけでなく、実際にコンテナ内を操作したい場合はdocker attach <コンテナID>コマンドを実行します。コンテナにアタッチすると、先ほどのログと違い、直接実行中のプロセスを操作することができます。そのため、何らかのコマンドやデーモンを実行中であればCtrl＋Cを実行することで、プロセスが停止してしまうので注意が必要です。アタッチしている状態でホスト側に戻るにはCtrl＋P＋Qを実行します。

execコマンドで追加プロセス実行

コンテナの中で操作するには、docker attachでプロセスにアタッチする命令のほかに、docker execコマンドを使う方法があります。これはコンテナの中で追加のプロセスを実行するものです。たとえば、何らかのデーモンが稼働中のコンテナがあり、その中でSSHログインするように操作／デバッグを行いたい場合は、次のようにコマンドを実行します。

書式

```
$ docker exec -i -t <コンテナID> /bin/bash
```

これはコンテナ内で稼働しているプロセスとは別の/bin/bashプロセスを起動します。そのため、作業後にexitで終了しても、コンテナそのもののプロセスは停止しません。

コンテナの停止

コンテナを停止するにはdocker stop <コンテナID>、またはdocker kill <コンテナID>を実行します。docker stopは

▼表2　docker psのオプション

オプション	説明
-a	停止したコンテナを含めてすべて(all)表示
-l	直近(latest)に実行したコンテナの情報を表示
-q	コンテナIDのみ表示する静かな(quiet)モード
--filter	状態(STATUS)で表示結果をフィルタ 例：docker ps -a --filter "Exited=1"

▼図4　デタッチド・モードで実行するとpingの結果ではなくコンテナID(ロングID)を表示

```
$ docker run -d centos ping 127.0.0.1 -c 60
e6c12622cdec0adec3141157a4522364eb431217f97ed3079bbc8ea101084ce9
```

SIGTERMシグナルを送信後、一定期間経過後にSIGKILLシグナルを送信します。一方のdocker killはただちにSIGKILLシグナルを送信するという違いがあります。

書式
```
$ docker stop <コンテナID>
$ docker kill <コンテナID>
```

コンテナの再起動

停止したコンテナのプロセスは、docker startコマンドを使うことで再開できます。オプションを指定しない場合、以前と同じオプションでコンテナは起動します。このとき-aオプションを使うと、コンテナを再起動しつつ、コンテナ内にアタッチすることができます。

書式
```
$ docker start -a <コンテナID>
```

実行中のコンテナを再起動したい場合は、docker restartコマンドを使うこともできます。

▼表3 dockerコマンド(コンテナ管理にかかわるもの)

コマンド	説明
docker cp	ホスト／コンテナ内でのファイルの複製
docker top	コンテナ内のプロセス表示
docker inspect	コンテナの詳細情報を表示

▼図5 docker cpの書式

```
$ docker cp <ローカルのパス> <コンテナID>:<コンテナ内のパス>
$ docker cp <コンテナID>:<コンテナ内のパス> <ローカルのパス>
```

▼図6 docker cpの実行例：コンテナ内の/etcをローカルの/archiveにコピー

```
$ docker cp 0e37159335c5:/etc /archive/
```

コンテナの削除

コンテナを実行し終えたあと、docker rmコマンドで対象のコンテナを削除します。逆に、docker rmで削除しない限り、コンテナの実行ごとに作成された情報は削除されないため、注意が必要です。

書式
```
$ docker rm <コンテナID>
```

コンテナ管理コマンド

コンテナでファイルを扱ったり、情報を表示したりするコマンドとして**表3**のものがあります。

ファイルの複製

docker cpコマンドはLinuxのcpコマンドのように、ホスト上のファイルやディレクトリをコンテナ上にコピーしたり、コンテナ上のものをホスト上にコピーしたりできます(**図5、6**)。

停止中／実行中、どちらの状態でもコピーすることができます。ただし、コンテナとコンテナの間で直接コピーすることはできません。

プロセス表示

docker topは、コンテナ内で何のプロセスが稼働しているかを知るためのコマンドです(**図7**)。プロセスの状況のみを知りたい場合、都度コンテナにアタッチする必要がなく便利です。

コンテナの詳細情報

docker inspectコマンドは、コ

▼図7 docker topの実行例

```
$ docker top 3c2ad14fc213
UID         PID         PPID        C           STIME       TTY         TIME        CMD
root        32172       15258       0           00:38       ?           00:00:00    nginx: ➥
master process nginx -g daemon off;
sshd        32178       32172       0           00:38       ?           00:00:00    nginx: ➥
worker process
```

ンテナIDをもとに、コンテナ内のホスト名やコマンドだけでなく、公開するポート、ボリュームなどに関する情報を表示します。

書式
```
$ docker inspect <コンテナID>
```

実行すると、結果はデフォルトでJSON形式として表示されます。項目が多数あるので、特定の項目のみ知りたい場合は--formatオプションを使うと便利です(**図8**)。

Dockerイメージの管理

イメージを管理するために、**表4**のようなコマンドが用意されています。

イメージの一覧

ローカルにあるイメージの一覧を表示するにはdocker imagesコマンドを使います(**図9**)。ここではリポジトリ名、タグ、イメージIDやサイズの情報が表示されます。

▼表4　dockerコマンド(Dockerイメージ管理にかかわるもの)

コマンド	説明
docker images	イメージのリポジトリやIDを一覧表示
docker search	リモートのイメージ情報を検索
docker pull	イメージをリモートから取得
docker push	イメージをリモートに送信
docker tag	コンテナにタグ付け
docker commit	コンテナの内容を、名前を付けてイメージに保存
docker rmi	イメージの削除

イメージの検索

docker searchコマンドは、リモート・レジストリ(Docker Hub)に登録されているイメージ情報の一覧を表示します。ブラウザでDocker Hubを表示して調べることもできますが、コマンドライン上でも検索できます。

書式
```
$ docker search <検索したいキーワード>
```

イメージの取得

リモート・レジストリからイメージを取得するには、docker pullコマンドを使います。通常、何も指定しない場合は、Docker Hubからダウンロードを試みます。

書式
```
$ docker pull <リポジトリ名>:<タグ>
```

対象のリポジトリに多数のタグがある場合、まとめてダウンロードするには--all-tagsオプションの利用が便利です。ただし、リポジトリによってダウンロードに時間がかかる場合や、ディスク容量を消費する場合があるので、注意が必要です。

```
$ docker pull --all-tags centos
```

なお、第2章で紹介した任意のレジストリやDocker Trusted Registryを使う場合は、ホスト名やポート番号の指定が必要になります。

```
$ docker pull localrepo:5000/myapp:1.0
```

▼図8　docker inspectの実行例：コンテナに割り当てられているIPアドレスを確認

```
$ docker inspect --format='{{.NetworkSettings.IPAddress}}' 3c2ad14fc213
172.17.0.13
```

▼図9　docker imagesの実行例

```
$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             VIRTUAL SIZE
busybox             latest              0f864637f229        2 weeks ago         2.433 MB
centos              latest              7322fbe74aa5        3 months ago        172.2 MB
```

イメージをレジストリに送信

ローカルのイメージをレジストリに送信するにはdocker pushコマンドを使います。対象ホストの指定がない場合は、Docker Hubへのアップロードを試みます。

書式
```
$ docker push <リポジトリ名>:<タグ>
```

Docker Hub以外のレジストリに送信する場合は、docker pushコマンドを実行する前に、docker tagコマンドを使って、送信先のホスト名やポート番号を指定する必要があります。たとえば、ホスト名がdev.docker.jp、ポート番号が5000の場合、次のように指定します。

```
$ docker tag myapp:1.0 dev.docker.jp:5000/myapp:1.0
```

docker tagコマンドはリポジトリ名だけでなくイメージIDも指定できます。なお、docker tagコマンドを実行しても、新しいイメージが作られるわけではないため、ディスク容量は増えません。docker imagesコマンドでは、見かけ上イメージが増えたように見えます。しかし、イメージが別名でタグ付けされただけであり、コンテナIDは共通です。

イメージの削除

イメージが存在し続けると、ローカルのディスク容量を圧迫していきます。使わないイメージはdocker rmiコマンドで削除可能です。

書式
```
$ docker rmi <イメージID>
$ docker rmi <リポジトリ名>:<タグ>
```

イメージの作成

使用しているコンテナの情報をもとに、新しいイメージを作成するにはdocker commitコマンドを使います。コミットとは「コンテナの内容を確定する」という意味です。

書式
```
$ docker commit <コンテナID> <リポジトリ名>:<タグ>
```

新しいイメージを作成するには、このコミット以外にも、Dockerfileという設定ファイルを使ってイメージを作る方法もあります。詳しくは第6章で扱います。

イメージのexportとimport

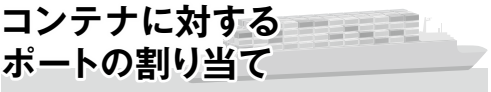
サーバを越えてイメージをやりとりするには、Docker Hubなどのレジストリを使う方法以外に、イメージ用のファイルを直接やりとりする方法があります。docker exportコマンドは、tar形式でイメージを出力します。

書式
```
$ docker export <出力ファイル名>.tar
```

出力した環境を別のDocker環境で取り込むには、docker importコマンドを実行します。パスに指定できるのはローカル上だけでなく、http://で始まるURLも指定できます。

書式
```
$ docker import <パス> | - <リポジトリ名>:<タグ>
```

コンテナに対するポートの割り当て



Dockerコンテナ内外の通信は、ホスト側のdocker0という仮想ブリッジを通して行われます。そのため、コンテナの内外で通信したい場合は、使うポートを明示する必要があります。

手動でコンテナ内とホスト側のポートを割り当てる場合は、docker runの-pオプションを使用します。-p <ホスト側ポート>:<コンテナ側ポート>の形式で指定します。ホスト側ポート8080に、Nginxコンテナのポート80を割り当てる場合は次のように実行します。

```
$ docker run -d -p 8080:80 nginx
```

また、コンテナによっては-P(大文字)を指定すると、コンテナが使用するポートを自動的にホスト側に割り当てることもできます[注2]。

```
$ docker run -d -P nginx
```

このときに割り当てられるのは、ホスト側の49151～65535の範囲内の空きポートです。docker psコマンドで確認するほかに、docker portコマンドでも確認できます。

```
$ docker port 931ea28cc678
443/tcp -> 0.0.0.0:32768
80/tcp -> 0.0.0.0:32769
```

また、Webサーバとデータベース間のようにコンテナとコンテナの間で安全に通信を行いたい場合は、--linkオプションを使うと便利です。通常、コンテナが起動するときにホスト名やIPアドレスを割り当てることはできません。しかし--linkオプションを使えば、コンテナ起動時に対象コンテナのエイリアス(別名)を指定できます。さらに、このエイリアスを使ってポート番号やIPアドレスを知ることができます。次の例はコンテナIDbf058dc12286にfrontというエイリアス名を指定したものです。

```
$ docker run -it --link bf058dc12286:front ubuntu bash
```

この新しいコンテナの中では、FRONTで始まるポート番号やIPアドレスなどの環境変数を持ちます。

envコマンド実行例
```
FRONT_PORT_80_TCP_ADDR=172.17.0.6
FRONT_PORT_80_TCP=tcp://172.17.0.6:80
```

これらの情報が取得できるので、IPアドレスやホスト名がわからなくても、--linkの設定をもとにコンテナ間の通信設定を行いやすくなります。

なお、先の--linkオプションの実行例では、オプションの中でコンテナIDを用いましたが、Dockerのコンテナ名(未指定時はランダムな形容詞と科学者／ハッカーの名前が割り当てられます[注3])も利用できます。コンテナ名を変更する場合はdocker rename <コンテナ名またはID> <新コンテナ名>が利用可能です。

コンテナのボリューム・オプション

Dockerではコンテナでデータを扱うために、ボリュームと呼ばれるディレクトリを指定することができます。これはコンテナ起動時に指定するとコンテナとは別の領域として認識／マウントされ、docker commitを実行しても、対象ディレクトリはコンテナに反映されません。また、コンテナを削除してもボリュームの情報は残り続けます(docker rm時に-vオプションを付けると、ボリュームも削除します)。

ボリュームをマウントして使うには、docker run実行時に-vオプションを使用します。オプションは複数回の利用が可能です。

```
$ docker run -d -v /mydata ubuntu
```

このとき、ボリュームがマウント／利用しているのは、単純にホスト上のディレクトリ(docker inspectで表示されるvolumeのパス)を利用しているだけです。つまり、暗号化などはされていませんので、パーミッションなどデータの取り扱いには注意が必要です。

ボリュームを定義するほかに、ホスト側の既存ディレクトリをマウントすることも可能です(**図10**)。たとえば/wwwディレクトリにコンテンツがあるとして、それをNginxコンテナのドキュメント・ルートと共有したい場合は、**図11**のようにコンテナを実行します。ホスト側の/wwwにindex.htmlを作成しておけば、ブラウザなどでホスト側のポート8080にアクセスすると、その内容が表示されます。

注2) 正確には第6章で扱うDockerfileにおいて、コンテナが使用するポート(EXPOSE)が明示されていると、コンテナ実行時、自動的にポートが割り当てられます。

注3) https://github.com/docker/docker/blob/master/pkg/namesgenerator/names-generator.go

高度な使い方としては、ほかのコンテナが利用しているボリュームを参照する方法もあります。新しいコンテナを起動するときにdocker runのオプションとして、--volumes-from <コンテナID>を指定すると、新しいコンテナも指定したボリュームを参照可能になります。

その他の管理用コマンド

これまでに紹介したもの以外に**表5**のようなコマンドがあります。

統計情報の出力

docker statsコマンドは、コンテナの使用しているリソース情報を表示するためのコマンドです。CPU、メモリ、ネットワークI/Oに関する情報が表示されるため、特定のホスト内でどのコンテナが多くリソースを使っているか、あるいはリソースを使っていないかを確認することができます(**図12**)。コンテナは1つまたは複数の指定が可能です。

Docker環境の情報表示

docker infoは、クライアントが接続しているホスト側の情報を表示するためのコマンドです。Dockerを動かす環境上で、どれだけのコンテナが実行されているか、イメージが保管されているか、どのようなシステム状況なのかを確認できるようになります(**図13**)。

バージョンの表示

docker versionはクライアントおよびサーバ側のDockerバージョンを表示します(**図14**)。

リモートのDockerデーモンに接続するには

dockerコマンドを実行するとき、クライアントが何も指定しない場合は、ローカル環境上のUnixソケット/var/run/docker.sockなどを参照しようとします。Dockerデーモンを起動していない場合や、リモートに疎通できない場合には、コマンドを実行しても**図15**のようなエラーが

▼図10　ホスト側の既存ディレクトリをマウントする際の書式

```
$ docker run -v <ホスト側パス>:<コンテナ側パス> <イメージ名>
```

▼図11　ホスト側の既存ディレクトリをマウントする際の実行例

```
$ docker run -d -p 8080:80 -v /www:/usr/share/nginx/html nginx
```

▼図13　docker infoの実行例

```
$ docker info
Containers: 60                          ←コンテナ数
Images: 246                             ←イメージ数
Storage Driver: aufs                    ←使用中のストレージ・ドライバ
 Root Dir: /var/lib/docker/aufs         ←ルート・ディレクトリ
 Backing Filesystem: extfs              ←ファイルシステム
 Dirs: 366                              ←ディレクトリ数
 Dirperm1 Supported: false              ←aufsマウント時のオプション
Execution Driver: native-0.2            ←実行ドライバのバージョン
Logging Driver: json-file               ←ロギング・ドライバ
Kernel Version: 3.13.0-57-generic       ←kernelバージョン
Operating System: Ubuntu 14.04.3 LTS    ←ディストリビューションのバージョン
CPUs: 1                                 ←ホスト側CPU数
Total Memory: 490 MiB                   ←ホスト側メモリ
Name: dev.docker.jp                     ←ホスト名
```

▼表5　dockerコマンド(その他管理用)

コマンド	説明
docker stats	統計情報の出力
docker info	Docker環境の表示
docker version	バージョン情報の表示

▼図12　docker statsの実行例

```
$ docker stats 3c2ad14fc213 0e37159335c5
CONTAINER           CPU %               MEM USAGE/LIMIT     MEM %               NET I/O
0e37159335c5        0.00%               1.81 MB/513.8 MB    0.35%               9.286 MB/532.2 kB
3c2ad14fc213        0.00%               2.466 MB/513.8 MB   0.48%               50.35 kB/738 B
```

表示されてしまいます。

Dockerデーモンをリモートから接続するためには、Docker起動時にオプションでTCPポートを開くための指定が必要になります。具体的には、UnixソケットとTCPポートの両方が有効になるように、Dockerの設定ファイルを編集します。Ubuntu/Debianの場合は/etc/default/dockerを開き、Fedora/CentOS/RHELの場合は/etc/sysconfig/dockerを開き、DOCKER_OPTS行を**リスト1**のように書き換えます。設定を有効にするため、Dockerデーモンを再起動します。

```
Ubuntu/Debianなど
$ service docker restart
Fedora/RHELなど
$ systemctl docker restart
```

リモートのDockerへの接続

リモートに接続するためには、2つの方法があります。1つはdockerコマンド実行時に都度-Hオプションでリモートの Dockerデーモンを指定する方法(**図16**)か、あるいは、環境変数DOCKER_HOSTを定義したあと、dockerコマンドを実行する方法(**図17**)です。**図16、17**はいずれも同じ結果になります。

注意点としては、このクライアントの通信が暗号化されていないことです。セキュリティ上のリスクを避ける方法として、TLS(Transport Layer Security)をDockerでも利用できます。通常の設定方法は多少複雑ですが、第4章で紹介するDocker Machineを使って環境を構築することで、自動的にTLSも有効化できます。SD

▼図14　docker versionの実行例

```
# docker version
Client:
 Version:      1.8.1
 API version:  1.20
 Go version:   go1.4.2
 Git commit:   d12ea79
 Built:        Thu Aug 13 02:35:49 UTC 2015
 OS/Arch:      linux/amd64

Server:
 Version:      1.8.1
 API version:  1.20
 Go version:   go1.4.2
 Git commit:   d12ea79
 Built:        Thu Aug 13 02:35:49 UTC 2015
 OS/Arch:      linux/amd64
```

▼図15　Dockerデーモンに接続できない場合のエラー

```
$ docker ps
Get http:///var/run/docker.sock/v1.20/containers/json: dial unix /var/run/docker.sock: ↵
permission denied.
* Are you trying to connect to a TLS-enabled daemon without TLS?
* Is your docker daemon up and running?
```

▼リスト1　Dockerの設定ファイルで、UnixソケットとTCPポートを有効にする

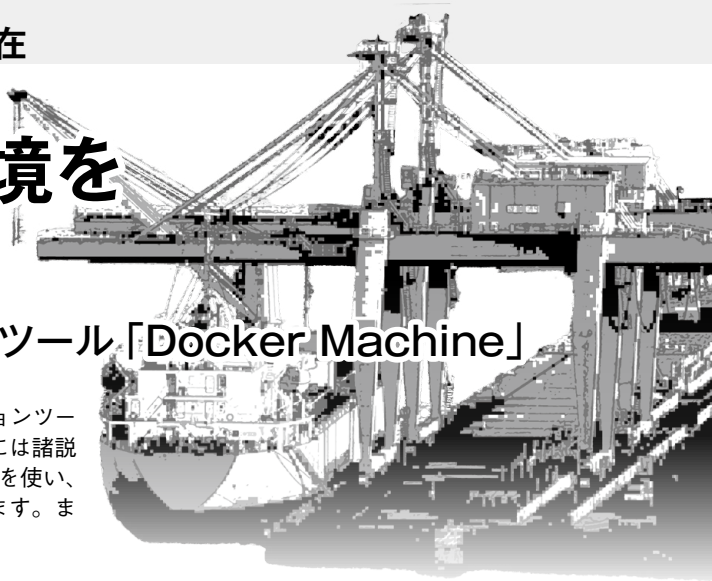
```
DOCKER_OPTS="-H tcp://0.0.0.0:2375 -H unix:///var/run/docker.sock"
```

▼図16　dockerコマンド実行時に都度、-Hオプションでリモートの Dockerデーモンを指定する例

```
$ docker -H tcp://192.168.0.10:2375 ps
CONTAINER ID    IMAGE        COMMAND       CREATED       STATUS        PORTS         NAMES
(以下略)
```

▼図17　環境変数DOCKER_HOSTを定義したあとに、dockerコマンドを実行する例

```
$ export DOCKER_HOST="tcp://192.168.0.10:2375"
$ docker ps
CONTAINER ID    IMAGE        COMMAND       CREATED       STATUS        PORTS         NAMES
(以下略)
```



第4章 Docker環境を自動構築

オーケストレーションツール「Docker Machine」

本章からはDockerを取り巻くオーケストレーションツールをとりあげます。オーケストレーションの意味には諸説ありますが、ここではDocker社が提供するツールを使い、複数のサーバにまたがる自動化処理のことを指します。まずは環境構築を行うDocker Machineです。

Author 前佛 雅人（ぜんぶつ まさひと） クリエーションライン㈱ Twitter @zembutsu

Docker Machine概要

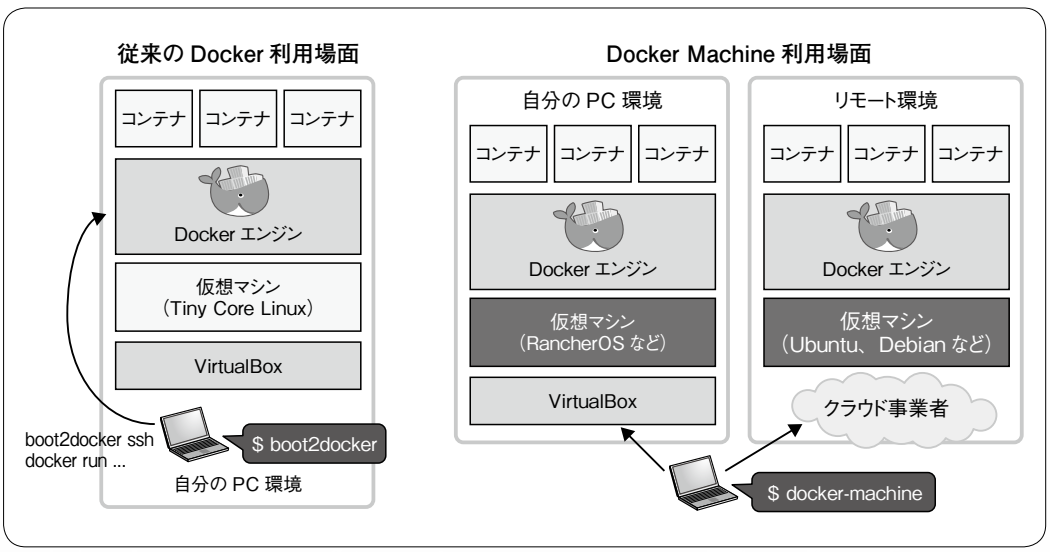
Docker Machineは、コマンドライン上でDocker環境を管理するためのツールです。仮想化やクラウドの環境を問わず、Dockerを使い始めるためには仮想マシンを用意し、そのうえでDockerエンジン（デーモン）をセットアップします。また、リモートで管理する場合は、デーモンの設定追加やTLSで安全に使う設定のほか、環境の切り替えが手間になりがちです。

Dockerを使えばいろいろなコンテナ環境を手軽に実行できるのに、そもそも、Dockerが動作するPCやサーバ環境の管理が手間になってしまうのは本末転倒です。

Docker Machineを使えば、Docker環境における面倒な作業を自動化します。コマンド1つ実行するだけで、仮想マシン環境の自動起動と、Dockerデーモンのプロビジョニングを行います。さらに、複数の環境についてもコマンドを通して管理でき、利用が終わればコマンド1つで環境を破棄することもできます（**図1**）。

Docker Machineを使う利点の1つに、さまざまな環境をすべてコマンドライン上から操作で

▼図1 Boot2DockerとDocker Machineの比較



きることが挙げられます。専用のGUIにログインすることなくDockerが動くマシンを作成できるだけでなく、SSHでログインするためのIPアドレスをGUIで確認する必要もありません。また、Dockerエンジンは TLS通信による暗号化設定も自動的に行いますので、クライアントはリモートから安全にDockerを操作できます。

対応している環境はVirtualBoxなどのローカルPC上の仮想環境だけではありません。Amazon Web Services(EC2)、SoftLayer、Azure、DigitalOceanなどの幅広いクラウドプロバイダに対応しているのも特徴です。

Boot2Dockerとの違い

これまでWindowsやMac OSなど、ローカルPC上における環境構築にはBoot2Docker[注1]の利用が推奨されていました。2015年8月にDocker Toolbox[注2]が新しく発表[注3]されてからは、環境構築としてはDocker Machineを推奨する旨がアナウンスされています。

Docker Toolboxとは、次のツール群を1つにまとめたパッケージです。

- Dockerクライアント：コマンドライン上で操作するdockerコマンド
- Docker Machine：Docker動作環境の自動構築と管理
- Docker Compose：複数のコンテナの状態をコードで管理(Mac版のみ)
- Kitematic：GUIのコンテナ管理ツール
- VirtualBox：Oracle社が提供する仮想マシン管理ツール

この中でも、Docker MachineがBoot2Dockerの発展型という位置づけです。どちらもローカル環境上のVirtualBoxを使い、Dockerの動作環境を構築します。一方、Docker Machineは複数のローカル環境を扱うことが可能です。それだけでなく、リモートのクラウド環境上におけるDocker環境の構築や削除・管理にも対応しています(**図2**)。詳しい機能の違いは**表1**をご覧ください。なお、どちらもオープンソースとして開発・公開されており、自由に使うことができます。

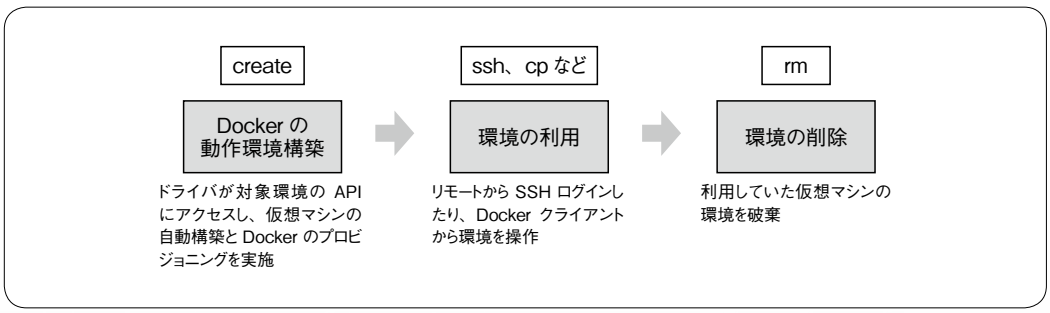
対応環境

Docker Machineは「ドライバ」と呼ばれる単位でさまざまな環境を混在して利用できます。現時点(v0.4.0)で対応しているシステム環境は次のとおりです。

- クラウドやVPS環境：Amazon Web Services、Microsoft Azure、DigitalOcean、Exoscale、Google Compute Engine、OpenStack、Rackspace、SoftLayer
- 仮想化システム：Microsoft Hyper-V、Virtual Box、VMware vCloud Air、VMware Fusion、

注1) http://boot2docker.io/
注2) Docker Toolbox配布サイト
https://www.docker.com/toolbox
注3) https://blog.docker.com/2015/08/docker-toolbox/

▼図2　Docker Machine利用の流れ



VMware vSphere

それぞれのドライバごとに、独自のオプション(リージョンやセキュリティに関する設定など)が存在します。各ドライバの詳細は対応ページ[注4]をご確認ください。

対応環境とセットアップ方法

Docker Machineの実体はdocker-machineという単一のバイナリファイルです。セットアップには2つの方法があります。1つはDocker Toolboxを使ってセットアップする方法です。PC上では手軽な方法ですが、Docker Machineだけを欲しい場合は冗長です。

もう1つの方法は、docker-machineのバイナリをダウンロードする方法です。ダウンロード用のページ[注5]に移動します。ダウンロードの一覧に、各OSやアーキテクチャごとのバイナリが配布されています。Linux 64bit環境の場合は「docker-machine_linux-amd64」、Mac OSは「docker-machine_darwin-amd64」、Windowsは「docker-machine_windows-amd64.exe」をダウンロードします。Windows以外の環境では、ダウンロード後は実行可能なパーミッションを設定します(**図3、4**)。

正常にセットアップされたかどうかは、次のようにバージョン情報が表示されるかどうか確認します。もし「No such file or directory」と表示された場合は、/usr/local/bin/にパスが通っているかどうかを確認ください。

```
$ docker-machine -v
docker-machine version 0.4.1 (e2c88d6)
```

Docker Machineで環境管理

Docker環境を構築・管理・削除するまで、一連の流れを見ていきます。

VirtualBoxで環境構築

コマンドdocker-machine createを次の書式のように実行することで、ドライバで指定した環境上に仮想マシンを構築し、Dockerのセッ

注4) Supported Drivers https://docs.docker.com/machine/drivers/
注5) https://github.com/docker/machine/releases/

▼表1 機能比較表

	ローカル環境の構築	複数のローカル環境	クラウド環境	TLS設定	初期リリース
Docker Machine	VirtualBox、VMwareなど	○	○	○	2015年2月
Boot2Docker	VirtualBoxのみ	×	×	×	2013年12月

▼図3 Linuxの設置例

```
$ wget -O docker-machine https://github.com/docker/machine/releases/download/v0.4.1/docker-machine_linux-amd64
$ chmod 755 ./docker-machine
$ sudo mv ./docker-machine /usr/local/bin/docker-machine
```

▼図4 Mac OSの設置例

```
$ curl -L https://github.com/docker/machine/releases/download/v0.4.1/docker-machine_darwin-amd64 > ./docker-machine
$ chmod 755 ./docker-machine
$ sudo mv ./docker-machine /usr/local/bin/docker-machine
```

トアップを行います。

書式
```
$ docker-machine create --driver <ドライバ名> <ホスト名>
$ docker-machine create -d <ドライバ名> <ホスト名>
```

ドライバを指定しない場合、デフォルトではvirtualboxが適用され、ローカルのVirtualBox上に環境を構築します。オプションで-d virtualboxと環境を明示することもできます。

書式
```
$ docker-machine create -d virtualbox <オプション> <ホスト名>
```

一番簡単な構築方法は、ホスト名を入力するだけです。**図5**のコマンドを実行するとlocalという名称の仮想マシンが自動的に構築されます。このとき、VirtualBoxをPC上で起動しておく必要はありません。バックグラウンドで自動的に環境構築が進行します。また、デフォルトではCPU 1プロセッサ、1MBのメモリ、HDD容量20GBを使用する仮想環境を構築しますが、オプション(**表2**)を指定することで仮想マシン環境をカスタマイズできます。次のコマンド例では、CPU 2プロセッサ、メモリ 512MB、ディスク容量を2GBに設定しています。

```
$ docker-machine create -d virtualbox --virtualbox-memory 512 --virtualbox-cpu-count 2 --virtualbox-disk-size 2000 <ホスト名>
```

初回実行時はVirtualBoxの中で使うためのLinuxディストリビューション(Boot2Docker付属のCore Linux)をダウンロードするため少々時間がかかります。次回以降はこのダウンロードの必要がないため、数分程度で環境が構築できるようになります。バックグランドでは仮想環境の構築と、SSHログイン用の鍵ペアの作成、Dockerデーモンのセットアップが自動的に進行します。この時点で、自分のホームディレクトリ直下の.docker/machine/ディレクトリに、各種の証明書やマシンごとのSSH鍵、VirtualBoxの場合はISOイメージなどが保管されます。

コマンド実行後、エラーが出なければ環境構築は完了です。docker-machine lsコマンドを実行すると、仮想マシンの稼働状態が一覧形式で表示されます(**図6**)。STATE(状態)の個所が

▼表2　VirtualBoxドライバのオプション

オプション	説明	デフォルト値
--virtualbox-memory	仮想マシンのメモリ容量	1024
--virtualbox-cpu-count	仮想マシンのCPUコア数	1
--virtualbox-disk-size	仮想マシンのディスク容量(単位MB)	20000(20GB)

▼図5　仮想マシンlocalの実行

```
$ docker-machine create -d virtualbox local
Creating CA: /Users/zem/.docker/machine/certs/ca.pem
Creating client certificate: /Users/zem/.docker/machine/certs/cert.pem
Image cache does not exist, creating it at /Users/zem/.docker/machine/cache...
(省略)
```

▼図6　仮想マシンの稼働状態を確認

```
$ docker-machine ls
NAME    ACTIVE   DRIVER       STATE     URL                         SWARM
local            virtualbox   Running   tcp://192.168.99.100:2376
```

Runningであれば正常に実行中です。なお、複数のマシンをcreateすると、作成されたホスト情報がすべて表示されます。

Docker環境に接続

Docker環境を操作するには2つの方法があります。1つはSSHで仮想マシンにログインする方法と、もう1つはDockerクライアント(dockerコマンド)から操作する方法です。

SSHでログインするにはdocker-machine ssh <ホスト名>コマンドを使います(**図7**)。仮想マシンのIPアドレスを知らなくても自動的にSSHログインが可能です。これは後述するAWSなどのクラウド環境のドライバを使った場合も同様で、自動的にパブリック側のIPアドレスに対してログインします。

ホスト名の後にコマンドを指定すると、SSHコマンドのように、ログインせずにリモート環境上でコマンドを実行することもできます。次の例はlocalという名称のホスト上でuptimeコマンドを実行します。

```
$ docker-machine ssh local uptime
 03:55:45 up  1:49,  1 users,  load ➥
average: 0.00, 0.01, 0.04
```

一方、複数のDocker環境を切り替えて使う場合であれば、Dockerクライアントを使うほうが便利です。クライアントは何も指定しない場合、ローカル環境上のDockerを参照しようとします。クライアント上の環境変数を切り替えると、リモートのDockerデーモンを直接操作できます。環境変数の確認はdocker-machine env <ホスト名>コマンドで行います。**図8**はWindows + Cygwin環境での実行例です。通常は不要ですが、このようにシェル環境を明示する場合は、--shellオプションを使います。

クライアントを切り替えるには、それぞれのexport行を実行するか、eval $(docker-ma

▼図7　仮想マシンlocalへのSSHログイン

```
$ docker-machine ssh local
                        ##         .
                  ## ## ##        ==
               ## ## ## ## ##    ===
           /"""""""""""""""""""¥___/ ===
      ~~~ {~~ ~~~~ ~~~ ~~~~ ~~~ ~ /  ===- ~~~
           ¥______ o          __/
             ¥    ¥        __/
              ¥____¥_______/
 _                 _   ____     _            _
| |__   ___   ___ | |_|___ ¥ __| | ___   ___| | _____ _ __
| '_ ¥ / _ ¥ / _ ¥| __| __) / _` |/ _ ¥ / __| |/ / _ ¥ '__|
| |_) | (_) | (_) | |_ / __/ (_| | (_) | (__|   <  __/ |
|_.__/ ¥___/ ¥___/ ¥__|_____¥__,_|¥___/ ¥___|_|¥_¥___|_|
Boot2Docker version 1.8.1, build master : 7f12e95 - Thu Aug 13 03:24:56 UTC 2015
Docker version 1.8
```

▼図8　Dockerクライアントを使ったログイン

```
$ docker-machine env local --shell bash
export DOCKER_TLS_VERIFY="1"
export DOCKER_HOST="tcp://192.168.99.100:2376"
export DOCKER_CERT_PATH="C:¥Users¥zembutsu¥.docker¥machine¥machines¥local"
export DOCKER_MACHINE_NAME="local"
# Run this command to configure your shell:
# eval "$(C:¥cygwin64¥bin¥docker-machine env local)"
```

chine env <ホスト名>)コマンドを実行します。以降docker runやdocker psコマンドを実行すると、リモートのDockerを操作します。

なお、接続先を切り替えるなど環境変数をリセットしたい場合には、次のコマンド実行が便利です。

```
$ eval $(docker-machine env -u)
```

ファイルを複製するscp

開発環境のコードをコンテナに送りたい場合や、コンテナ間でデータを複製したい場合にはdocker-machine scpコマンドを使うと便利です。

書式
```
$ docker-machine scp [-r] <コピー元ホスト>:<パス> <コピー先ホスト>:<パス>
```

たとえば、ローカルにある/etc/temp.txtを、ホストlocalの/opt/etc/にコピーする場合は次のように入力します。

```
$ docker-machine scp /etc/temp.txt local:/opt/etc/
```

また、ディレクトリ単位でまとめてコピーする場合は、-rオプションを使います。

```
$ docker-machine scp -r ./code/ local2:/home/docker/code
```

コピー先がリモートの場合、ディレクトリには適切な権限が必要です。VirtualBoxの場合はdockerユーザでログインするため、このユーザ権限の範囲の場所にファイルを置く必要があります。

環境の削除

使い終わった環境は、docker-machine rmコマンドで削除できます。このとき、確認もなくすべての情報が削除されるため、必要なデータがあれば事前にバックアップが必要です。

```
$ docker-machine rm local
Successfully removed local
```

なお、VirtualBox上もしくはクラウド上のマシンをDocker Machineを使わずに削除した場合は、ホームディレクトリ以下にある.docker/machine/machines/にある対象マシン用ディレクトリを直接削除する必要があります。

そのほかの仮想マシン管理コマンド

これまで紹介したほかにも、**表3**にまとめたような管理用のコマンドが用意されています。

クラウド環境で環境構築

Docker Machineはドライバを使い分けることでさまざまな環境を利用できます。ここでは主な環境における構築方法を見ていきましょう。

Amazon Web Servicesを使う場合

Amazon Web Services(AWS)上のEC2インスタンスを自動的に起動し、その中にDocker環

▼表3　主な管理用コマンド

書式	動作
docker-machine start <ホスト名>	仮想マシンの起動
docker-machine kill <ホスト名>	仮想マシンの強制停止
docker-machine restart <ホスト名>	仮想マシンの再起動
docker-machine stop <ホスト名>	仮想マシンの停止
docker-machine ip <ホスト名>	IPアドレスの表示
docker-machine inspect <ホスト名>	仮想マシンの詳細表示

境を自動的に構築します。Docker Machineを使えばGUIにログインせず、迅速な環境の構築や、利用後の環境削除もスムーズに行えます。

利用時は-d amazonec2ドライバの指定や、各種のオプションを指定します。オプションの中でもアクセスキーとVPC IDの指定が必須なため、利用する環境にあわせて適切なものを入力します(**図9**)。

オプションのうち、リージョン(デフォルトはus-east-1)、インスタンス(t2.micro)、OSやディスクサイズ(16GB)などを変更可能です。あるいは、環境変数を使った定義も可能です。指定できるオプションおよびデフォルト値はドキュメント[注6]をご覧ください。注意点としては、インターネットに接続できない環境でインスタンスを準備した場合、インスタンスの起動は可能ですがDockerを自動セットアップできません。

DigitalOceanを使う場合

DigitalOcean[注7]は世界中に複数のデータセンタを持つVPSサービスとして、世界中で広く使われているサービスです。Docker Machineを使えば、DigitalOcean上でもDockerがすぐに利用できるドロップレットの構築・管理が可能です。

DigitalOceanの場合、必須のパラメータはAPIトークンです。管理画面上で事前にAPIトークンを発行し、控えておきます(**図10**)。デフォルトのパラメータを変更したい場合は、ドキュメント[注8]を参考に調整ください。

汎用的なgenericドライバを使う方法

これまで紹介してきた仮想化・クラウド環境のほかにも、SSHが可能な環境であれば、genericドライバを使うことでDocker Machineの管理対象とすることができます。これにより、任意のクラウド環境や任意のディストリビューション上を操作できます。ただし、条件としてリモートログインするSSHアカウントはrootないし同等の権限を持っている必要があります(**図11**)。SD

注6) https://docs.docker.com/machine/drivers/aws/
注7) https://www.digitalocean.com/
注8) https://docs.docker.com/machine/drivers/digital-ocean/

▼図9　AWS利用時に最低限必要なオプション

```
$ docker-machine create \
    --driver amazonec2 \
    --amazonec2-access-key <アクセスキー> \
    --amazonec2-secret-key <秘密アクセスキー> \
    --amazonec2-vpc-id <VPC ID> \
    <ホスト名>
```

【参考情報】

- Docker Machine ドキュメント
  https://docs.docker.com/machine/
- Docker Machine 0.3.0 Deep Dive
  https://blog.docker.com/2015/06/docker-machine-0-3-0-deep-dive/

▼図10　DigitalOcean利用時に最低限必要なオプション

```
$ docker-machine create \
    --driver digitalocean \
    --digitalocean-access-token <APIトークン> \
    <ホスト名>
```

▼図11　genericドライバを使った構築の例

```
$ docker-machine create -d generic \
    --generic-ssh-user <ログイン名> \
    --generic-ssh-key <SSHログイン用公開鍵のパス> \
    --generic-ip-address <IPアドレス> \
    <ホスト名>
```

第5章

Docker Swarmでコンテナのスケジューリング

コンテナをクラスタリングしリソースを管理

コンテナの可用性を高めたり分散したりして処理を行うために、複数のDockerホストを1つのリソースの集まりとして扱うことができます。このクラスタを管理するためにさまざまなツールが開発・提供されています。ここではDocker Swarmを使ったリソースとコンテナの管理方法を見ていきます。

Author 前佛 雅人(ぜんぶつ まさひと)　クリエーションライン㈱　Twitter @zembutsu

クラスタのコンテナを管理するDocker Swarm

コンテナをどのサーバ上でどのように起動するのか決めることを、コンテナのスケジューリングと呼びます。Docker Swarm[注1]は、このスケジューリングを行うためのツールの1つです。おもにDocker社によって開発が進められていますが、ソースコードはGitHubで公開されており、オープンな開発が行われています。

Docker Swarmのシステムは、Swarmマネージャとエージェントがクラスタを形成し、1つのリソースプール(Dockerホスト上のリソースがまとめられた状態)を形成しています(**図1**)。このDocker Swarmを使えば、Dockerが動作する複数の環境を1つのリソースプールとして扱うことができます。ただし、現時点では開発中のバージョンであり、機能はコンテナをスケジューリングすることに特化しています。

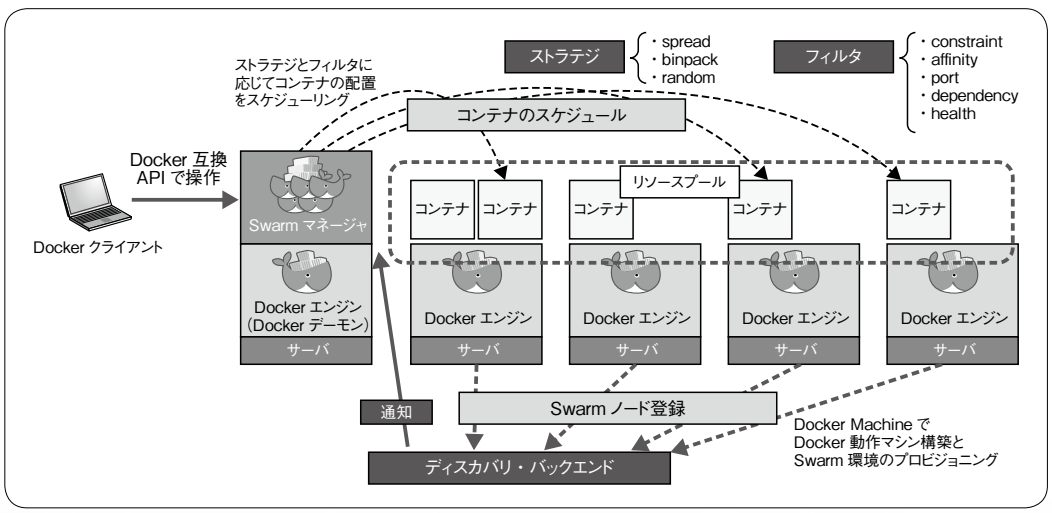
負荷分散や高度なリソース管理を行うことはできませんので、このような機能を使いたい場合は、後述する別のツールをご検討ください。

Dockerエンジンと互換性のあるAPI

Swarmの大きな特徴の1つに、DockerのAPI

注1) https://www.docker.com/docker-swarm

▼図1　Docker Swarmの構成



と互換性がある点が挙げられます。これは、通常のDockerクライアントがリモートのDockerエンジンを操作するのと同じように、Swarmが公開しているポートに対してアクセス可能です。

つまり、通常のdockerコマンドと同じコマンドを使い、クラスタ全体のコンテナ操作が可能なことを意味します。たとえばdocker runコマンドを実行すると、通常はDockerエンジンが稼働している環境でコンテナが起動します。リモートのDockerを操作するには環境変数を切り替えます。これと同様にSwarmのホストやポートを指定すると、Swarmが管理するリソースプール内のいずれかのサーバ上でコンテナを起動できます。

リソースプール上でコンテナを起動するとき、Dockerエンジンが動作するノードの情報を意識する必要はありません。しかし全体的なスケジューリングの方向性としての「ストラテジ」や、より具体的なコンテナの実行方針として「フィルタ」を指定できます。

Docker Swarmの環境構築

Docker Swarmのクラスタは、Swarmマネージャ、Swarmノード、ディスカバリ・バックエンド(**図1**)によって構成されます。Swarmマネージャと Swarmノードは、Docker社から公式リポジトリが公開されていますので、こちらを使うのが便利です。必要があればGitHubで公開されているGo言語のソースコード[注2]を元に、バイナリをビルドすることもできます。

クラスタ全体の構成は**図2**のように単純なものです。SwarmマネージャがDockerクライアントのAPIを受け付け、Swarmノード上にコンテナをスケジュールしたり、各Swarmノード上のリソース情報やプロセスの情報を収集する役割があります。

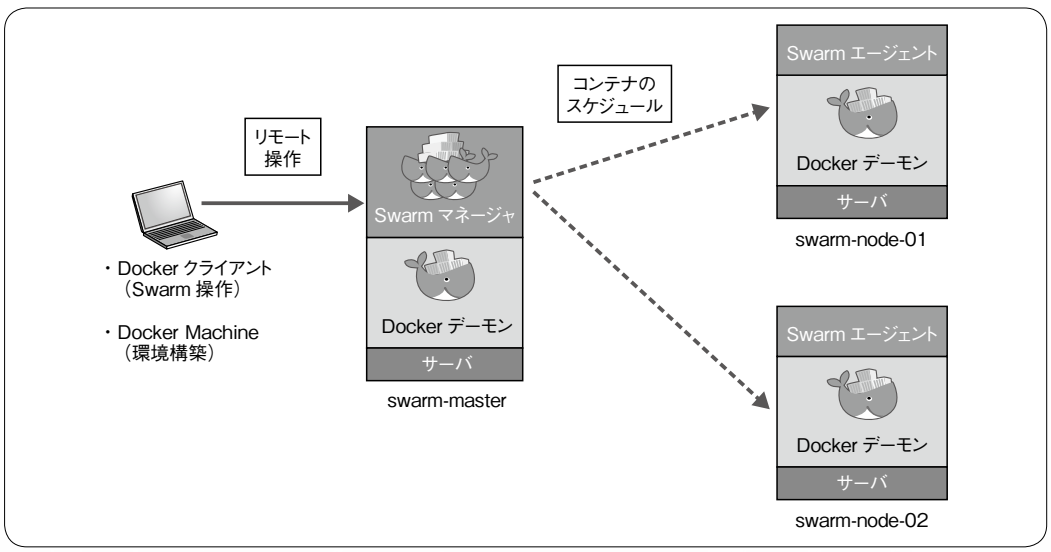
ディスカバリ・バックエンド

Swarmはクラスタを形成するとき、Swarmが動作するノード管理に「ディスカバリ・バックエンド」を使用します。これは、ノードを登録したり、障害があるノードを検知し、Swarmマネージャに伝える役割を持ちます。

ディスカバリ・バックエンドには、Docker Hubを使うホステッドディスカバリを使うか、

注2） https://github.com/docker/swarm

▼図2　Docker Swarm動作環境の構築



自分でローカルにKVSを構築する方法(etcd、Consul、Zookeeperなど)を選べます。どちらも利用できますが、インターネットに接続可能な環境であれば、ホステッドディスカバリを使う方法が簡単です。

ホステッドディスカバリはトークンを使ってクラスタを形成します(**図3**)。まずあらかじめ`swarm create`というコマンドを通して、ランダムな文字列のトークンを作成します。次に、リソースプールに追加したいノード上では、自分のIPアドレスとDockerが稼働しているポート番号、そして発行したトークンを指定して`swarm join`コマンドを実行します。あとは自動的にSwarmマネージャがノードを認識できるようになります。

トークンの作成

ホステッドディスカバリを使う場合、まずはじめにSwarmクラスタを識別するために、トークンを作成する必要があります。いずれかの環境で、次のコマンドを実行します。

```
$ docker run --rm swarm create
(...省略...)
↓最後に表示される文字列がトークン
7546a86262b847f7f34785ab2e0d6118
```

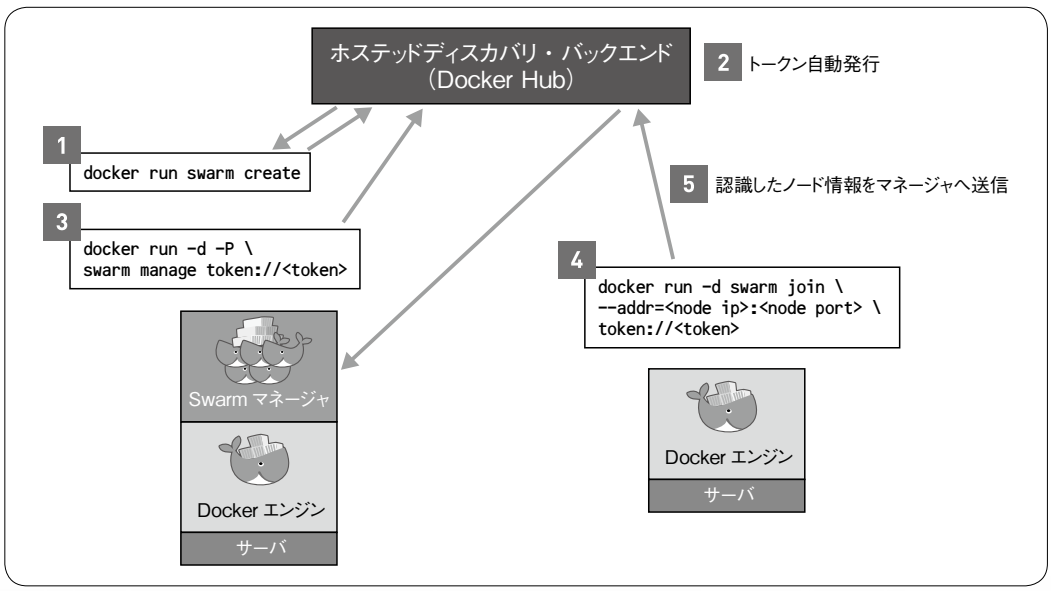
Docker Machineでクラスタを構築

Docker Machineを使えば、Swarmマネージャもノードも比較的簡単に構築できます。仮想マシンの構築を行えるだけでなく、DockerおよびSwarmマネージャまたはノードのプロビジョニングを自動的に行えます。また、環境の管理や削除もDocker Machineを使えばスムーズに行えます。

まずはじめにSwarmマネージャを起動します。**図4**の例は`swarm-master`という名称のホストを起動します。ここではDigitalOceanの環境を使用していますが、VirtualBoxやAWS向けなど、任意のドライバを利用できます。

次にSwarmノードを、同じくDocker Machineを使って起動します(**図5**)。先ほどとコマンドは似ていますが、`--swarm-master`というオプションが外れ、ホスト名が`swarm-node-01`に

▼図3　ホステッドディスカバリ・バックエンドとトークン



なっている点のみ違います。

同様に、ホスト名をswarm-node-02として起動すると、自動的に2台目のノードが起動し、Swarmクラスタに追加されます。この状態でdocker-machine lsを実行すると、**図6**のようにホスト情報が追加されていることがわかります。

Dockerクライアントからリモートでクラスタを操作するには、次のコマンドを実行し、画面に表示される環境変数を有効化します。

```
$ docker-machine env --swarm swarm-master
```

以後dockerコマンドを使った操作は、Swarmクラスタ全体に対する操作となります。たとえばdocker infoは特定のDockerホストに関する情報表示ではなく、クラスタ全体のリソース情報を表示するようになります(**図7**)。

この状態でdocker runコマンドを実行すると、クラスタ内のいずれかのホスト上でコンテナが実行されます。次のコマンドを実行したあと、各ノードにSSHでログインし、どこでコンテナが起動しているか確認しましょう。

```
$ docker run -d -P nginx
```

さらに、各ノードのDockerエンジンはSwarmクラスタの管理下に入ります。試しに、ノード内でdocker runコマンドで任意のコンテナを起動してみましょう。それから、Swarmに対してdocker psを実行すると

▼図4　Swarmマネージャの起動

```
$ docker-machine create \
  --driver digitalocean \
  --digitalocean-access-token <DigitalOceanのトークン>  \
  --swarm \
  --swarm-master \
  --swarm-discovery token://<トークン> \
  swarm-master
```

▼図5　Swarmノードの起動

```
$ docker-machine create \
  --driver digitalocean \
  --digitalocean-access-token <DigitalOceanのトークン>  \
  --swarm \
  --swarm-discovery token://<トークン> \
  swarm-node-01
```

▼図6　ホスト情報の表示

```
$ docker-machine ls
NAME            ACTIVE   DRIVER         STATE     URL                         SWARM
swarm-master             digitalocean   Running   tcp://128.199.179.153:2376  swarm-master
(master)
swarm-node-01            digitalocean   Running   tcp://128.199.157.183:2376  swarm-master
swarm-node-02            digitalocean   Running   tcp://188.166.252.45:2376   swarm-master
```

▼図7　クラスタ全体のリソース情報を表示

```
$ docker info
Containers: 4
Images: 1
Role: primary
Strategy: spread
Filters: affinity, health, constraint, port, dependency
Nodes: 1
 node1: 188.166.252.158:2375
  m Containers: 4
  m Reserved CPUs: 0 / 1
  m Reserved Memory: 0 B / 514.5 MiB
  m Labels: executiondriver=native-0.2, kernelversion=3.13.0-57-generic,
operatingsystem=Ubuntu 14.04.3 LTS, provider=digitalocean, storagedriver=aufs
CPUs: 1
Total Memory: 514.5 MiB
Name: 0477faab5932
```

ノード名/コンテナ名として自動的にコンテナが認識されていることがわかります。

なお、この状態でSwarmクラスタに参加しているホスト情報を知るには、swarm listコマンドを使う方法もあります。

```
$ docker run swarm list token://<トークン>
103.253.146.176:2375
188.166.252.158:2375
```

手動でクラスタ構築

手動でSwarmクラスタを構築することも可能です。ただし、マシン環境は自分で構築しなくてはいけません。すでに稼働中のマシン環境があり、そこを利用する場合には有用です。

まず、構築時に必要なのは、Swarmマネージャの起動です。起動コマンドは次のように実行します。

```
$ docker run -d -P swarm manage
token://<トークン>
```

2台のノードを用意したら、それぞれのノード上で次のコマンドを実行することで、自動的にクラスタに登録されます。

```
$ docker run -d swarm join --addr=<ノードのIPアドレス>:2375  token://<トークン>
```

このとき、マネージャ側のサーバでdocker psコマンドを実行すると、ホスト側のどのポートがSwarmに対して割り当てられているか確認できます(**図8**)。

今回の例ではポート32768がSwarmのポート番号だとわかります。あとは、Dockerクライアントは環境変数を指定するか、あるいは次のように-Hオプションを指定し、Swarmにアクセスできます。

```
$ docker -H tcp://127.0.0.1:32768 info
```

Swarmクラスタに対してdocker infoコマンドを実行すると、クラスタ全体のノード情報やリソース情報が表示されます。

3種類のストラテジ

コンテナ実行時、Swarmはどのノードでコンテナを実行するのか、大まかなスケジューリング方針をあらかじめ決められます。この方針はストラテジ(strategy)と呼ばれ、それぞれのストラテジのアルゴリズムに従ってSwarmノードをランク付けし、順位の高いノード上でコンテナを実行しようとします。

ストラテジは3種類あり、「spread」「binpack」「random」いずれか適用されます。以降で1つ1つのストラテジを見ていきます。

spreadストラテジ

spreadストラテジは、より多くのノードを使いコンテナを分散しようとする方式であり、Swarmのデフォルトのストラテジです(**図9**)。この方式では、実行中のコンテナ数に応じてノードをランク付けします。初回実行時など、複数のコンテナで実行しているコンテナ数が同じになった場合は、ランダムにノードが選ばれます。

binpackストラテジ

binpackストラテジは、できるだけ少ないノードにコンテナを集約する方式です。対象となるノードで利用可能なリソース(CPU、メモリ)の上限に達するまで、コンテナの起動を試みます

▼図8　ホスト側の割り当てポート確認

```
$ docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED
STATUS              PORTS                    NAMES
0477faab5932        swarm               "/swarm manage token:"   4 seconds ago
Up 3 seconds        0.0.0.0:32768->2375/tcp   trusting_mcclintock
```

(**図10**)。リソース上限に達すると、別のノードを使用します。なお、こちらも初回実行時はランダムに選択されます。

ただし、他のストラテジと異なり、コンテナを実行するたびにコンテナが使用するリソース(CPU、メモリいずれか、または両方)を必ず指定する必要があります。この指定を行わずにコンテナを稼働しても、Swarmはリソースの計算ができないため、1つのノード内で多くのコンテナが稼働することになります。

リソースを指定するには、コンテナ実行時にオプションを指定します。次のコマンドは-mオプションを使い、tomcatコンテナが利用可能なメモリ上限を512MBとしています。

```
$ docker run -d -m 512MB tomcat
```

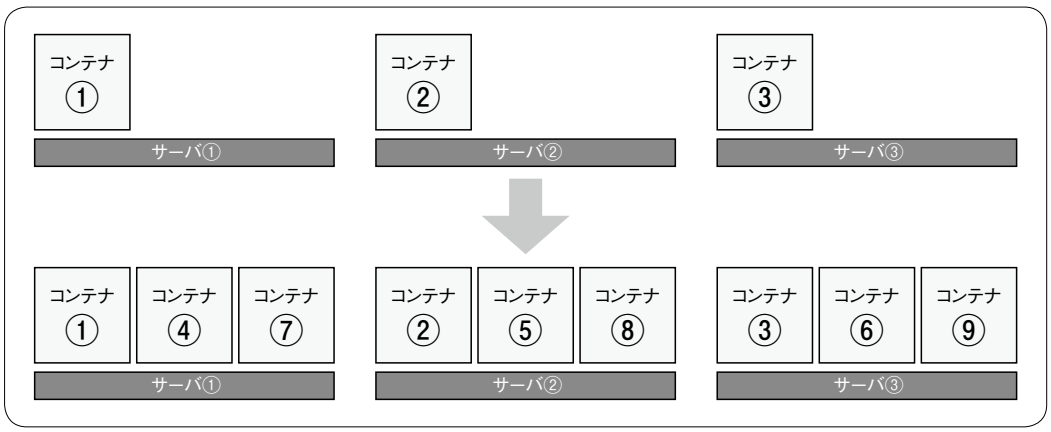
randomストラテジ

randomストラテジは、spreadやbinpackと異なり、ランキングのためのアルゴリズムを使用しません。おもにデバッグ用途で使用します。ランダムに選びたくない場合は、後述するフィルタを使って具体的なノードを選択します。

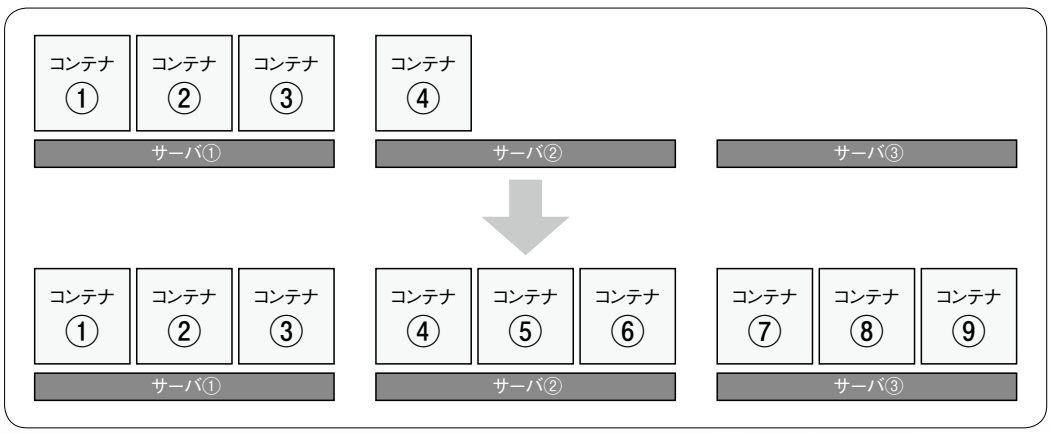
ストラテジの使い分け

デフォルトのspreadストラテジは、分散してコンテナを実行したい場合に向いています。1つのノードで障害や停止が発生しても、全体に

▼図9　spreadストラテジ



▼図10　binpackストラテジ



対する影響が少ないため、たくさんのノードを持っている場合は有効なストラテジです。

一方のbinpackストラテジは、限られたノード内のリソースをできるだけ使おうとします。そのため、リソースの使われていないノードが増えることはありません。ただし、障害が発生すると多くのコンテナに影響が出ます。

ストラテジの指定方法

ストラテジを指定するには、Swarmマネージャの起動時にオプションで指定します。以下の例では、オプション--strategyでbinpackストラテジを指定しています。

```
$ docker run -d -P swarm manage ➡
token://<トークン> --strategy binpack
```

5種類のフィルタ

スケジューリングにおいてストラテジが大まかな方針を決めておくのに対し、フィルタはより具体的なスケジューリング先を指定するためのものです。フィルタの指定は、ストラテジよりも優先されます。

具体的な使い方としては、特定のOSやハードウェア環境を選択してコンテナを起動する場合に重宝します。また、コンテナの依存関係やノードの稼働状態によって、システム側で自動的に決定されるフィルタもあります。以下でそれぞれのフィルタの役割を見ていきましょう。

Constraintフィルタ

Dockerのラベル(label)、システム情報、ノード名に応じて、コンテナのスケジューリング条件を指定することができるフィルタです。

ノードに任意のラベルを付けるには、Dockerデーモンの起動オプションに--labelを使います。たとえば、リージョンtokyoというラベルを指定したい場合は--label region=tokyoをDocker起動時に指定します(デーモン起動時のオプションは/etc/default/dockerなどで指定できます)。

こうしておけば、このノードでコンテナをスケジュールしたい場合、コンテナ実行時に-eオプションを使ってラベルを指定することでフィルタリングの条件として機能させられます。次の例はラベルregionがtokyoの環境で、nginxコンテナをスケジュールします。また、条件の設定には==(同じ)演算子または!=(否定)演算子が利用できます。

```
$ docker run -d -e ➡
constraint:region==tokyo nginx
```

2つめは、standard constraintと呼ばれる固有のDocker環境に関する条件です。これはdocker infoコマンドを実行したときに表示されるストレージドライバ(storagedriver)、実行ドライバ(executiondriver)、カーネルのバージョン(kernelversion)、OS情報(operatingsystem)によって指定できます。

3つめは、ノードconstraintと呼ばれるもので、ノード名を直接指定することができます。次の例はノード名web1に対してnginxをスケジュール「しない」ものです。

```
$ docker run -d -e constraint:node!=➡
web1 nginx
```

affinityフィルタ

既存のコンテナ名、またはダウンロード済みのイメージ名に対応して、コンテナをスケジュールします。すでに稼働中のコンテナと同じノード上でコンテナを使いたい場合、あるいは使いたくない場合に指定します。たとえばコンテナ名cacheが存在するノードでtomcatコンテナを実行するには、次のように実行します。

```
$ docker run -d -e ➡
affinity:container==cache tomcat
```

イメージの指定が有効な場合があります。

Swarmクラスタ上でコンテナを起動しようとするとき、その対象ノードにイメージがない場合はダウンロードを開始します。このイメージが巨大なファイルサイズの場合、コンテナの稼働に時間がかかります。そこで、あらかじめイメージが取得済みのノードを指定することにより、迅速に稼働したい場合にお勧めのフィルタです。次の例は、tomcat:6コンテナを持っているノード上でコンテナを実行します。

```
$ docker run -d -e ↵
affinity:image==tomcat:6 tomcat:6
```

ポートフィルタ

Swarmクラスタ上でコンテナを実行するとき、-pオプションでポートを指定すると、自動的にポートが未使用なノード上でコンテナを起動しようとします。たとえば、docker runをSwarmに対して実行し、オプションで-p 80:8080(コンテナのポート8080を、Swarmの80に割り当て)を指定すると、ポート80が使われていないノードのいずれかでコンテナを実行します。

依存関係フィルタ

依存関係フィルタとは、コンテナの依存関係がある場合に自動的に適用されるフィルタです。特定のコンテナとリンクするオプション--linkか、ボリュームを参照するオプション--volumes-fromを使いコンテナを起動しようとすると、同じノード上でコンテナを起動しようとします。

ただし、同じノードにコンテナがスケジュールできない場合(リソースがないなど)、コンテナを実行することはできません。たとえば、新しいコンテナを起動するとき、複数のコンテナに対するリンクを指定しているとします。もし、対象のコンテナが別々のノードで稼働している場合、コンテナは起動できません。

ヘルスフィルタ

ノードの死活状況に応じて自動的に適用されるフィルタです。ディスカバリ・バックエンドによって正常に稼働していないとみなされるノードは、自動的にスケジュールの対象から除外されます。ノードが正常に復帰すると、スケジュール対象にも自動的に復帰します。

その他のスケジューラ

Docker Swarm以外にもさまざまなスケジューラを使うことができます。最後に、Dockerと連携する主なツールを**表1**に紹介しておきます。SD

【参考情報】

・Docker Swarm
https://docs.docker.com/swarm/

▼表1 Docker Swarm以外の主なスケジューラ

名称	URL	概要
Kubernetes	http://kubernetes.io/	Google社によって開発され、オープンソースとして開発が続けられている。ラベルやポッドの概念を使い、コンテナを管理することが可能
OpenShift	https://www.openshift.com/	DockerとKubernetesを組み合わせたPaaS環境を実現しており、RedHat社によるサポートを受けられる
Rancher	http://rancher.com/	ロードバランサや仮想ネットワークをサポートしており、プライベートなコンテナ利用環境を目指している。Rancher社が開発しており、オープンソースとして公開
Nomad	https://nomadproject.io/	スケジューリングだけでなく、リソース管理も可能。コンテナ以外にも仮想化システムでの処理やバイナリの実行にも対応。HashiCorpが開発中であり、オープンソースとして公開

第6章

Docker環境のコード化とオーケストレーション

DockerfileとDocker Composeで作業の省力化・効率化

Dockerイメージを使うには、前章までに紹介したほかにも、設定ファイルを使うことによって自動的にイメージを構築できます。また、複数のコンテナの構築だけでなく同時に起動／停止することもできます。本章では、これらの作業を自動化するため、Dockefileを使ったイメージの構築方法と、Docker Composeを使った複数コンテナの構築やオーケストレーションのしかたをみていきます。

Author 前佛 雅人（ぜんぶつ まさひと） クリエーションライン㈱ Twitter @zembutsu

設定管理のコード化、その恩恵と問題

アプリケーションやミドルウェアの環境をどのように構築するか。これは、Dockerコンテナだけに限らず、物理環境やクラウド環境でも大きな課題になりつつあります。かつての環境構築は、人手によって設定手順書に従いコマンドを実行するものでした。しかし、この人手による作業には、次のような課題があります。

・作業者によって作業に時間がかかる
・作業ミスを引き起こすリスクと、ミスを回避するためのチェック体制
・手順書の変更やその管理・共有方法

構成管理ツール

これを解決する手法の1つが、Chef、Puppet、Ansibleなどのツールを使う手法です。これらは構成管理ツールと呼ばれており、あらかじめ設定ファイル（レシピ、プレイブックなど）でインストールすべきパッケージの情報や、サーバ内の設定を定義しておきます。あとは環境構築時にツールを実行することで、定義された環境を正確・迅速に自動的に構築します。

さらにこのファイルを、ソースコードのような「コード」として扱うことで、変更差分の確認を容易にできるようになりました。ただし、設定ファイルは各ツールによって互換性がなく、ある程度自由に使いこなすためには、それぞれの設定言語（DSL）を学ぶ必要があります。

マシンイメージの活用

構成管理ツールと似たような手法として、仮想マシンイメージを作る方法があります。これは頻繁に使う開発環境や共通するアプリケーションを、あらかじめテンプレート的な環境として仮想サーバ上に構築します。このとき、構成管理ツールと組み合わせることもできます。環境が整うと、それを頻繁に使うマシンイメージとして保管し（これをゴールデンイメージと呼ぶ場合もあります）、必要に応じて使う手法です。

一見すると便利な手法ですが、アプリケーションやミドルウェアのバージョンが上がる場合（仕様変更やセキュリティ対策などで）、都度、マシンイメージの再構築が必要になります。また、イメージの数が増えてくると、管理がたいへんになりがちという新しい課題も出てきました。

DockerfileでDockerイメージを構築

Dockerでは、Dockerfileと呼ばれる設定ファイルを使い、コンテナイメージを自動構築できます。Dockerfileでは何のDockerイメージをもとにコンテナを準備し、コンテナ内でど

のようなコマンドを実行するか定義します。これにより、Dockerイメージを手作業で構築するよりも、効率的に作成できるようになります(**図1**)。

具体的な使い方としては、特定のパッケージをコンテナ内にセットアップしたり、設定ファイルを書き換えたり、ソースコードやバイナリをコンテナ内にコピーできます。このファイルを使いdocker buildコマンドを実行することにより、自動的にDockerイメージを構築するしくみになっています。

作成後のイメージはコンテナとして実行できるだけでなく、Docker Hubなどのレジストリを通して共有できます。イメージの容量が大きい場合でも、Dockerfileであれば手軽にファイルを共有できますし、GitHubのようなリポジトリを使ってバージョン管理も行えます。

このような機能から、docker buildは構成管理ツールやマシンイメージによる環境の構築と管理を組み合わせたような概念を持っていると言えるでしょう。ただし、必ずしもDockerfileを使う必要はありません。もし、これまでにChef、Puppet、Ansibleなどの構成管理ツールの知見があるのであれば、それを活用するのも方法の1つです。とはいえ、Docker Hubに自分のイメージを公開・配布するときは、Dockerfileを通して構築手順が明らかになっている方が、多くの方にとって経緯がわかりやすく、再利用しやすくなると思われます。

コンテナ実行時の環境も定義

Dockerfileはイメージ構築時のコマンドを定義するだけはありません。コンテナ実行時に必要となる、次の情報を定義することもできます。

- コンテナがデフォルトで実行するコマンドやオプション
- ホスト側に公開するポート番号
- 参照するボリューム
- 動作用のディレクトリやユーザ権限

さらにDockerには、複数のコンテナの構築や操作を行うDocker Composeというツールが提供されています。これは、複数のコンテナの状態をdocker-compose.ymlファイルで定義したり、Docker Composeを使い同時に操作(オーケストレーションと呼ばれる一斉作業)したりできるようになります。

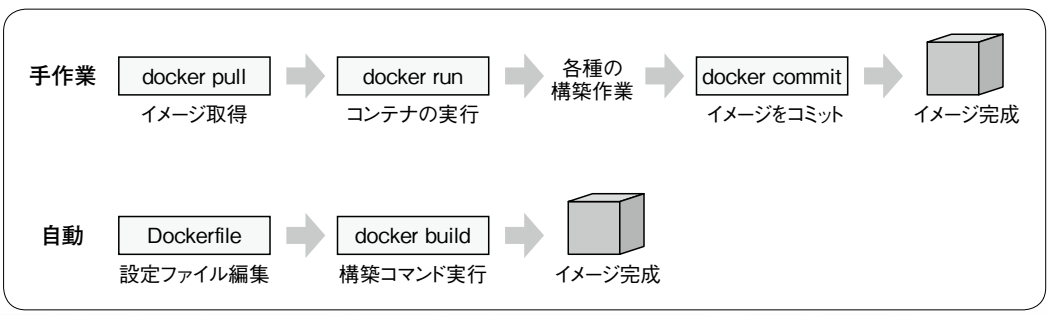
イメージを自動構築するDockerfile

Dockerfileとは、Dockerイメージの構築時の命令を記述するファイルのことです。イメージを構築するにはdocker buildコマンドを使います。

書式
```
$ docker build <オプション> <Dockerfileのパス>
```

典型的な使い方は、作業用のディレクトリを作成し、Dockerfileを配置・編集したあと、docker buildコマンドでイメージを構築しま

▼図1 Dockerイメージ構築手法の比較



す。このとき、-tオプションでタグの指定と、.(カレントディレクトリ)にあるDockerfileを指定します。

簡単なコンテナの作成

例として、WebサーバとしてNginxを起動するためのDockerfileを作成してみます。コンテナは次のような条件を指定するとします。

・ベースとするイメージ
　centos:7
・インストールするパッケージ
　epel-release、nginx
・ローカルのindex.htmlをドキュメントルートに複製
・ポート80を公開
・Nginxの起動

まず、任意の名前のディレクトリを作成し、中に移動します。エディタを使い、Dockerfileを作成し、次のとおりに編集・保存します。詳細な命令の意味については後述します。

```
FROM centos:7
MAINTAINER <自分の名前>
RUN yum update -y
RUN yum install -y epel-release
RUN yum install -y nginx
ADD index.html /usr/share/nginx/html/
EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]
```

それから、ドキュメントルートとして表示するためのindex.htmlを作成します。

```
$ echo '任意の文字列' > ./index.html
```

docker buildコマンドでイメージを構築します。次の例ではmynginxとタグ付けしていますが、任意の名称が利用できます。

```
$ docker build -t mynginx .
```

もしエラーが出る場合は、記述が違っている可能性がありますので、内容を見直します。問題なければ、このmynginxイメージを使い、コンテナを起動します。

```
$ docker run -d -p 80:80 mynginx
```

実行後、ブラウザから対象マシンのポート80にアクセスすると、先ほど作成したindex.htmlの内容が表示されています。

Dockerfileの命令

ファイルの中ではコンテナに関するさまざまな命令を記述できます。

・FROM……コンテナ構築時に使用するイメージ(必須)。
　例：FROM nginx
・MAINTAINER……コンテナの管理担当者名やメールアドレスなど任意の文字列。
　例：MAINTAINER foo@example.jp
・RUN……コンテナ内でコマンドを実行。
　例：RUN apt-get install -y wget
・CMD……コンテナ実行時にデフォルトで実行するコマンドと引数。ただしコンテナ実行時に引数があると、この指定は無視。
　例：CMD["ping","127.0.0.1","-c","30"]
・ENTRYPOINT……コンテナ実行時に実行するコマンドを指定。CMDと違いコンテナ実行時に--entrypointオプションを指定しない限り、必ず実行する。また、CMDと併記する場合はその内容がENTRYPOINTで指定したコマンドのオプションになる。
　例：ENTRYPOINT["ping"]
・COPY……ホスト上の特定のファイルやディレクトリをコンテナ内に複製。
　例：COPY <ホスト上のパス> <コンテナ内パス>
・ADD……COPYと同様の機能を持つがtar形式の展開やURLの指定も可能。
・WORKDIR……コンテナ内でRUN・CMD・ENTRYPOINTなどの命令を実行するディレクトリを指定。
・ENV……コンテナの中で参照可能な環境変数

を定義。

例：ENV WEB_PORT 80

・VOLUME……ボリュームのマウント先。

例：VOLUME["/var/log/"]

・PORT……コンテナが使用するポート番号。ただし、コンテナの中でのポート指定であり、docker run時に-pまたは-Pオプションの利用が必要。

例：PORT 80 443

・USER……RUN、CMD、ENTRYPOINTの実行時のユーザ名またはUID。

例：USER nginx

詳細な使い方やサンプルはリファレンス[注1]をご覧ください。

コンテナのクラスタを管理するDocker Compose

Docker Composeは複数のコンテナやアプリケーションを構築・管理するためのコマンドラインツールです。設定にはdocker-compose.ymlというYAML形式のファイルを使い、Dockerfileと連動して環境構築ができます。Composeはコンテナ環境を自動的に構築するだけでなく、複数のコンテナの起動／停止といったコンテナ管理もできます(**図2**)。

Docker Hubで公開されている公式パッケージを中心に、複数のコンテナを組み合わせる環境構築を簡単にする方法として、docker-compose.ymlを配布する流れもあります。

Docker Composeのセットアップ

Composeを使うにはdocker-composeという単一のバイナリファイルを使います。基本的なセットアップ手順はDocker Machineと同様です。ダウンロード用のページ[注2]に移動し、各OSやアーキテクチャごとのバイナリをダウンロードし、適切なパーミッションを設定します(**図3**)。

正常にセットアップされたかどうかは、バージョン情報が表示されるかどうかで確認します。

```
$ docker-compose -v
docker-compose version: 1.4.2
```

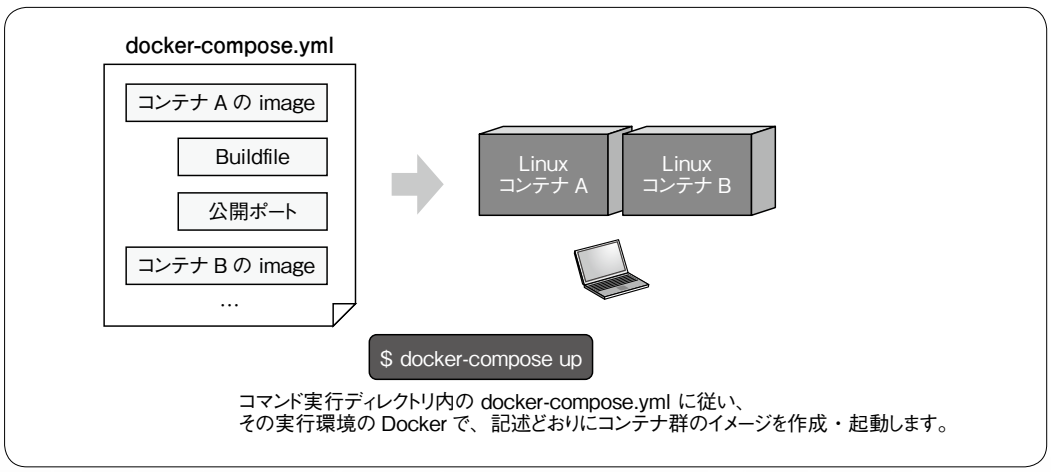
Composeの利用例

WordPressを使うためのコンテナを起動してみます。任意のディレクトリでdocker-compose.ymlファイルを作成し、内容を**リスト1**のよ

注1) Dockerfile reference https://docs.docker.com/reference/builder/

注2) https://github.com/docker/compose/releases/

▼図2 Docker Composeでコンテナ群の自動起動



うに編集します。これはwordpressとdbという名前のサービスを定義し、それぞれの利用するイメージ名、リンク、ポートの公開に関する情報を定義しています。

このファイルを作成したあと、docker-compose upを実行すると、コンテナが起動します。メモリが少ない環境は起動できない場合がありますのでご注意ください。あとはブラウザからポート8080にアクセスすると、WordPressの初期画面が表示されます。このとき Ctrl+C を実行するとコンテナが停止します。

利用可能な主なコマンドは次のとおりです。

・docker-compose up
コンテナ群の起動。-dオプションをつけると、デタッチドモードとしてバックグランドで動作
・docker-compose ps
対象のコンテナ群の状態を表示
・docker-compose stop
コンテナ群を停止
・docker-compose kill
コンテナ群を強制停止
・docker-compose logs
コンテナ群のログを表示
・docker-compose rm
コンテナが使用したファイルを削除。確認画面でyを入力すること

詳細な設定方法やオプションについては、ドキュメント[注3]をご参照ください。

Composeは開発途上のツールであり、現時点では1つのサーバ上のコンテナのみ管理できます。言い換えれば、複数のサーバにまたがってコンテナを配置することはできません。しかし、今後のバージョンアップにより、たとえばDocker Swarmと連携するなどし、アプリケーションを複数のサーバ上で展開できるようになれば、Composeはコンテナを使った運用において欠かすことができないツールになるでしょう。SD

注3） Docker Compose https://docs.docker.com/compose/

▼図3　Mac OSとLinuxの設置例

```
・Mac OSの設置例
$ curl -L https://github.com/docker/compose/releases/download/1.4.2/docker-compose-
Darwin-x86_64 > ./docker-compose
$ chmod 755 ./docker-compose
$ sudo mv ./docker-compose /usr/local/bin/docker-compose

・Linuxの設置例
$ curl -L https://github.com/docker/compose/releases/download/1.4.2/docker-compose-
Linux-x86_64 > ./docker-compose
$ chmod 755 ./docker-compose
$ sudo mv ./docker-compose /usr/local/bin/docker-compose
```

▼リスト1　docker-compose.yml

```
wordpress:
  image: wordpress                  ←wordpressイメージの使用
  links:
    - db:mysql
  ports:
    - 8080:80                       ←コンテナ内のポート80をホスト側8080に公開

db:
  image: mariadb                    ←mariadbイメージの使用
  environment:
    MYSQL_ROOT_PASSWORD: example    ←rootパスワードをexampleで指定
```

第7章 HashiCorpの自動化ツールとDockerの連携

HashiCorp道の真髄（しんずい）

ここまでの章では、Docker社が提供する各種のツールを紹介してきました。最後に、HashiCorpのツールを使い、Docker環境の構築や管理を行う手法を紹介します。

Author 前佛 雅人（ぜんぶつ まさひと） クリエーションライン㈱ Twitter @zembutsu

HashiCorpとDocker

HashiCorp社[注1]は、開発環境の自動構築として有名なVagrantを始めとし、多くのDevOps問題を解決するためのさまざまなツールを提供している会社です。

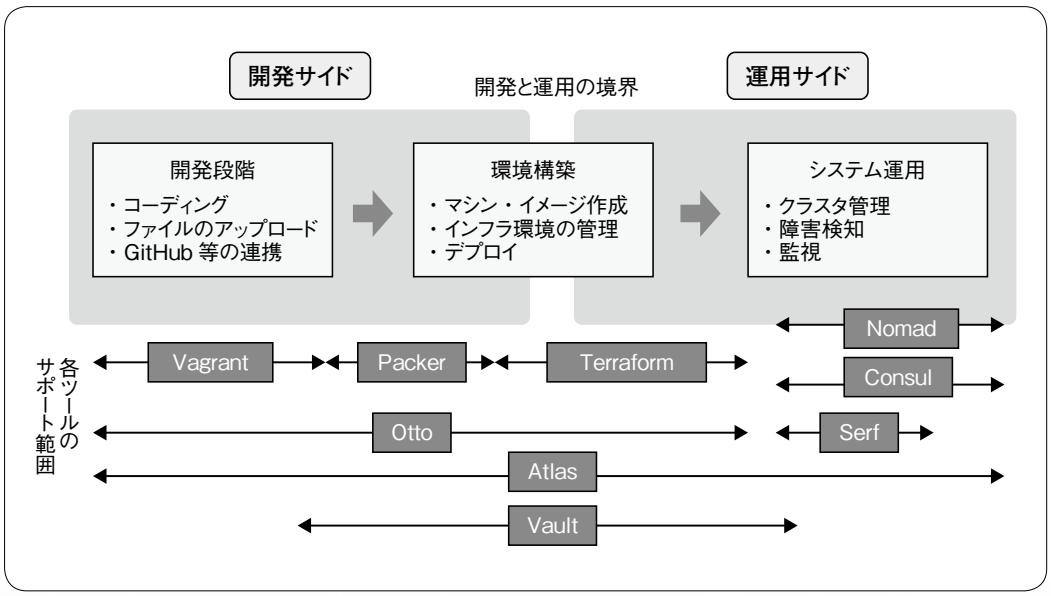
HashiCorpのよいところは、「何かをするためには、使った方が楽だよね」というツールを提供していることです。これはHashiCorp道（The Tao of HashiCorp[注2]）と呼ばれる開発方針にもとづき開発を進めているためです。彼らはまず、最適な業務フローを検討し、それを達成するためのツールがなければ自分たちで開発する方式を採っています。現在9つのプロダクトが提供されていますが、それぞれが異なった作業範囲における目的を解決するためのものです（**図1**）。

これまでHashiCorpでは、おもに仮想化環境

注1） https://hashicorp.com/

注2） https://hashicorp.com/blog/tao-of-hashicorp.html

▼図1 HashiCorpのツールがカバーする範囲



やクラウドに対応したツール開発が行われてきました。2015年以降はDockerのコンテナ環境にも対応し始めています。そのため、これまでHashiCorpのツールを使ったことがある方なら、これまでと同じ感覚でDocker環境も扱えます。

たとえばVagrantであれば、開発環境の選択肢の1つとしてDockerを選ぶこともできます。PackerであればDockerイメージを構築しますし、Terraformはコンテナ環境をデプロイします。現時点での各ツールとDockerの対応状況は**表1**をご覧ください。

今年9月に開催されたHashiConfではDockerのスケジューリングに特化したNomad[注3]と、Dockerにも対応する開発・デプロイ用ツールとしてOtto[注4]の発表も行われており、今後も活発に開発が進められるものと思われます。

注3）https://nomadproject.io/
注4）https://ottoproject.io/

各ツールの対応状況

TerraformでDocker環境の構築

Terraformはインフラ環境を自動的に構築し、管理するためのツールです。プロバイダを切り替えることにより、クラウド環境だけでなく、Dockerにも対応できます。

TerraformのDockerプロバイダは、Docker APIを通してDockerデーモンまたはDocker Swarmと通信します。現時点の機能では、Dockerコンテナを動かす下準備として、各サーバ上にコンテナやDockerイメージを自動的に設置／削除するのに便利です。

例として、Ubuntuイメージを使ったDockerの環境を構築します。docker.tfというファイルを準備し、内容を**リスト1**のとおりにします。

▼表1　HashiCorpの各ツールとDockerの対応状況

名称	URL	ツールの役割	Docker対応状況
Vagrant	https://www.vagrantup.com/	開発環境の自動構築と管理	Dockerイメージを使った環境構築に対応
Packer	https://packer.io/	マシンイメージの自動構築	Dockerイメージの構築に対応
Terraform	https://terraform.io/	インフラのコード化と自動構築	Docker環境の構築（簡易）
Serf	https://serfdom.io/	クラスタ管理とオーケストレーション	Dockerに依存しない
Consul	https://consul.io/	サービス検出やオーケストレーションなど	Dockerに依存しない
Vault	https://www.vaultproject.io/	秘密鍵やトークンなど秘密情報の管理	Dockerに依存しない
Atlas	https://atlas.hashicorp.com/	開発から運用に至るフローを一体化	Dockerイメージの管理に対応
Nomad	https://nomadproject.io/	コンテナのスケジューリング	Dockerコンテナのスケジューリングに対応
Otto	https://ottoproject.io/	開発環境の自動構築とデプロイ	アプリケーションタイプの1つとしてDockerに対応

▼リスト1　docker.tf

```
provider "docker" {
    host = "tcp://127.0.0.1:2375/" ← Dockerデーモンの IPアドレスとポート番号
}

resource "docker_container" "web" { ←Dockerコンテナのリソースを定義
    image = "${docker_image.uginx.latest}" ←使用するイメージはリソースdocker_image
    name = "web"
}

resource "docker_image" "nginx" {  ←Dockerイメージのリソースを定義
    name = "nginx:latest"
}
```

このあとインフラに対する変更内容を確認するterraform plan、そして反映するterraform applyを実行すると、自動的にNginx用のDockerイメージをダウンロードし、webという名称でコンテナを起動します(**図2**)。使い終わったあとは、terraform destroyコマンドを実行すると、コンテナの停止と削除を行いますので、環境構築まわりの面倒な作業を楽にできます。

ここで紹介したほかにもさまざまなオプションが利用できます。詳細についてはドキュメント[注5]を参照ください。

VagrantとDocker開発環境

Vagrantでは、デフォルトのVirtualBox環境のほかにもさまざまな環境を、プロバイダと呼ばれる単位で操作できます。このプロバイダの1つにDockerも対応しており、Dockerコンテナを開発環境として利用できるようになります。Vagrantを連携する利点は、Dockerイメージをそのまま使えるだけでなく、Dockerfileがあればイメージをもとに環境を構築することもできることです。詳細はドキュメント[注6]を参照ください。

PackerとDockerイメージ

Packerはマシンイメージを自動構築するためのツールです。単純にイメージを作るだけでなく、Chef・Puppetなどの構成管理ツールを自動的に実行することもできます。Docker向けの機能としては、特定のDockerイメージを自動的にコミットしたり、tar形式のファイルとしてエクスポートする機能があります。こちらも詳細なオプション設定は、ドキュメント[注7]を参照ください。

Serf・Consulについて

SerfやConsulはクラスタ管理やオーケストレーションを実施するためのツールです。いずれもOS上で動作するアプリケーションのため、Docker環境の利用にあたって特別な設定・操作などはありません。

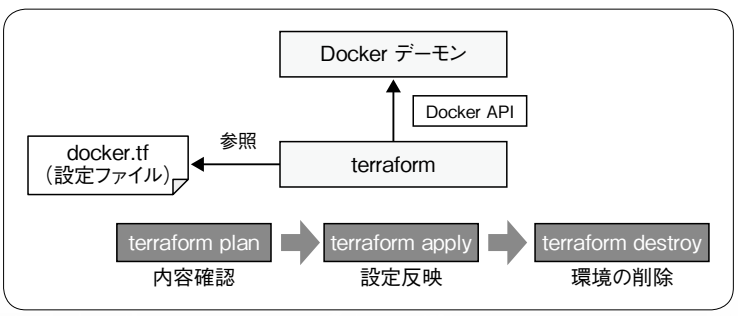
DockerとHashiCorp

個人的に、DockerとHashiCorpが提供する手法は似通っているように思います。たとえばDockerの場合はdocker runでコンテナ環境を実行しますし、Vagrantはvagrant upコマンドで自動的に環境を構築します。ただし、Dockerはあくまでコンテナありきな操作や機能が中心です。それに対して、HashiCorpのツールの場合はプラットフォームを自由に選べます。Dockerに限らず、複数の環境を併用したり使い分けたりできるのが大きな違いです。加えて、既存の構成管理ツールやサービスとの連携も容易になるしくみを提供しています。

開発・運用現場の規模や考え方によって、合うツールや合わないツールがあるかもしれません。筆者はDockerやHashiCorpのツールを必ずしも皆が使うべきとは考えていません。みなさんの現場において、今回の特集の中で「もしかして使えるかも？」という利用シーンがあれば、積極的に活用していただければと思っています。SD

注5) https://terraform.io/docs/providers/docker/index.html
注6) http://docs.vagrantup.com/v2/docker/index.html
注7) https://www.packer.io/docs/builders/docker.html

▼図2 TerraformとDocker連携



第2特集

ネットワーク・システム管理の定石

SNMPの教科書

堅実な監視で障害をキャッチ

サーバ、スイッチ、ルータ、プリンタ……、ネットワークにつながったあらゆる機器を、同じルールで管理できるプロトコル「SNMP」。本特集では、「“運用”とは何か」「なぜ、SNMPが必要なのか」という疑問を出発点に、入門編、実用編、応用編とステップアップしながらSNMPについて学びます。実際の運用でのハマリどころも扱っていますので、日々の運用にぜひ役立ててください。

また、コラム①～④では、本文で出てきた各テーマについておさらい、深堀りをしています。こちらも併せてお読みください。

CONTENTS

❶ SNMPはなぜ必要なのか 68
❷ SNMP入門編 70
❸ SNMP実用編 77
❹ SNMP応用編 84
❺ SNMPの関連技術 91
❻ まとめ 93

セクション❶～❻：Author 山下 薫（やました かおる） URL http://www.linkedin.com/in/kaoruyamashita

コラム 1 SNMPの動作形態 73
コラム 2 SNMPのデータモデル 75
コラム 3 CentOSでの利用方法 80
コラム 4 SNMPのバージョン 83

コラム❶～❹：Author 馬場 俊彰（ばば としあき） ㈱ハートビーツ CTO ／ Twitter @netmarkjp

1 SNMPはなぜ必要なのか

Author 山下 薫(やました かおる) URL http://www.linkedin.com/in/kaoruyamashita

システムの運用に利用する「SNMP」とはどのようなプロトコルなのか、その説明の前にまずはシステムの「運用」とはどのような段階を指すのかをお話しします。

運用にトラブルはつきもの

「SNMP(Simple Network Management Protocol)」は、情報システムの「運用」を支える、地味ですが広く用いられているプロトコルです[1]。では、そもそも「運用」とは何でしょうか。

ネットワークを含む情報システムを新しく導入する場合、構築作業が必要です。具体的には、ユーザの要件に合わせてシステムを設計し、その構成要素を選び、開発または設定して全体を組み立て、テストを行います。テストの結果、要件を満足できていれば、本番稼働に入ります。本番稼働後の情報システムを正常に稼働させ続け、万が一のトラブルに対応する段階が「運用」です。

情報システムの多くが、定期メンテナンスなどを除けば24時間365日、いつも正常に動いていることが当然になっている今日、運用の役割はとても重要です。では、情報システムの運用の際には、どのようなトラブルが発生するのでしょうか。運用担当者が対処しなければならないトラブルの例を2つ挙げます。

1つめは、サーバやルータのCPU負荷の上昇によるアプリケーションの応答時間の遅延です。アプリケーションの種類によりますが、ユーザが体感できるかどうかのごくわずかな遅延でも、使い勝手に影響が出てきます。入力に対する応答に、たとえば10秒の待ち時間が発生すると、アプリケーションによってはユーザ離れを招きます。これを防ぐために、CPU負荷を監視しておき、想定していた値を超えるようであればサーバを増設するなどの対策を施します。

2つめは、ネットワーク機器のインターフェースへの過負荷によるパケットロスです。たとえば、1Gbpsの帯域を持つインターフェース(ギガビットイーサネット)に、1G以上の負荷がかかると、転送できないパケットが発生(パケットロス)します。パケットロスが少ないうちは、ユーザへの影響はほとんどありませんが、ロスが多くなってくると、1つめの例と同じくアプリケーションの応答が遅くなったり、正常に動作しなくなってしまいます。これを防ぐためには、インターフェースの利用率や、転送できないパケットの数を監視し、過負荷が続くようであれば帯域を増やすなどの対策をとります。

もしSNMPがなければ

先に挙げた2つのトラブルの例において、もしSNMPが使えないとしたら、どうやって機器の状態を監視すれば良いのでしょうか。

CPU負荷やパケットロスは、サーバやネットワーク機器にsshなどでログインして、テキストベースのコマンド(CLIまたはCUI)を入力することで知ることができます。**図1**は、Ciscoルータの例です。ここでは「line vty」および「line con」以下に、「exec prompt timestamp」を設定しています。

また、専用のGUIやWeb画面経由で必要な値を知ることができる機器もあります。**図2**は、無線LANアクセスポイント(自律型)にブラウザで

アクセスして、統計情報を表示させたものです。

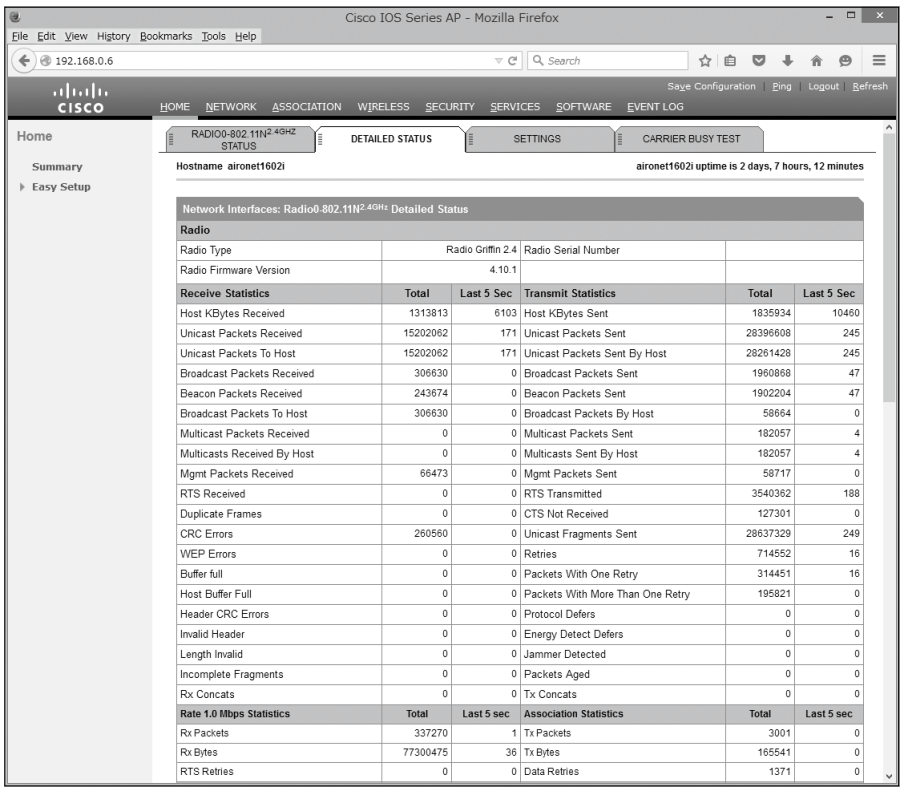
しかし、これらのアクセス手段は、システムを構築する際の各機器の設定と動作確認のために用意されていることがほとんどです。構築担当エンジニアにはお馴染みのものであっても、運用担当者にとって必ずしも使いやすいものとは限りません。たとえば、画面に表示される項目の数や情報量が多過ぎて、監視

▼図1　CLIによる監視の例

```
cisco1921#show interfaces GigabitEthernet 0/0
Load for five secs: 0%/0%; one minute: 1%; five minutes: 2%  → CPU負荷
Time source is NTP, 16:20:28.150 JST Sat Oct 17 2015

GigabitEthernet0/0 is up, line protocol is up  → リンクしているか?
...(中略)...
  5 minute input rate 1282000 bits/sec, 147 packets/sec → 受信トラフィック量
  5 minute output rate 1353000 bits/sec, 159 packets/sec  → 送信トラフィック量
     12694238 packets input, 2985991471 bytes, 0 no buffer
     Received 0 broadcasts (0 IP multicasts)
     0 runts, 0 giants, 0 throttles  → 受信時のエラー
     4269 input errors, 0 CRC, 0 frame, 4269 overrun, 0 ignored
     0 watchdog, 0 multicast, 0 pause input
     6913178 packets output, 434240556 bytes, 0 underruns
     0 output errors, 0 collisions, 0 interface resets  → 送信時のエラー
     8 unknown protocol drops
     0 babbles, 0 late collision, 0 deferred
     0 lost carrier, 0 no carrier, 0 pause output
     0 output buffer failures, 0 output buffers swapped out
```

▼図2　Web UIによる監視の例



の対象だけをピックアップすることが困難だったりします。

また、たとえばCPU負荷を監視したいとしても、対象の機器の種類やベンダによって方法が違うことがほとんどです。テキストベースのCLIでは、CPU負荷を知るためのコマンドが違ったり、もしコマンドが同じであったとしても、出力内容が異なったりすることがあります。これはGUIやWeb UIでも同じです。さらに、同じベンダの製品でも、機種が違えば監視のための方法が違ってくる場合があります。

さらに、**図1**のようなテキストベースのコマンドの出力において、どこが監視対象の項目なのかを特定できたとしても、手動でその値を継続して監視し続けるのは一苦労です。

そのため、監視のための操作を自動化することが必須になります。しかし、機器にログインし、コマンドを入力し、出力を解析するまでのステップすべてを自動化することは簡単ではありません。もし自動化のためのスクリプトを書けたとしても、監視対象の機器の種類によってスクリプトの内容を変えなければなりません。また同じ機器であっても、ソフトウェアアップグレードを実施すると、コマンドの出力が変更されてしまい、自動化のためのスクリプトが正しく動かなくなってしまうこともあり得ます。

以上のような監視のための課題を解決するプロトコルが、SNMPです。SNMPがあれば、情報システムのさまざまな構成要素を、同じ方法で監視できます。具体的には、サーバ、スイッチ、ルータなどが、SNMPで規定されている共通のプロトコルとデータモデルに従うことによって、一元的な監視が可能になるのです。

2 SNMP入門編

Author 山下 薫(やました かおる) URL http://www.linkedin.com/in/kaoruyamashita

ここでは、SNMPの概要について解説します。基本的なコマンドとその実行結果、またデータモデルを見ながら、SNMPがどのように動作するのかをつかんでください。

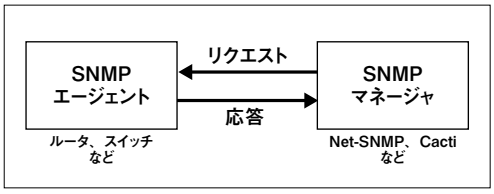
あらゆる監視対象を「OID」で管理

SNMPでは、監視する側を「マネージャ」、監視される側を「エージェント」と呼びます(**図3**)。この図のような「リクエスト—応答」型の場合は、マネージャがクライアント、エージェントがサーバのように働きます。

SNMPの「N」は「Network」の略ですが、SNMPで監視される対象はネットワーク機器に限りません。本特集で解説するルータ、スイッチの監視に限らず、ネットワークカメラ、温度計、気圧計、さらには工場のなかの機器をSNMPで監視している例[2]もあります。今ふうに言うとIoT(Internet of Things)のためのプロトコルとして用いられていると言えるかもしれません[3]。

さまざまな種類の機器を同じプロトコルで取り扱うために、SNMPでは独特の「データモ

▼図3 SNMPマネージャとエージェント



デル」を定義しており、対象となる機器すべてがこのモデルに従います。ただし、このデータモデルには相当「くせ」があるため、とっつきにくいと感じられる方が多いのではないでしょうか。このあたりの実際を、具体例を用いて紹介していきます。

たとえば、対象となる機器の2番めのインターフェースが受信したバイト数を、監視対象として取得したい場合は、

```
IF-MIB::ifInOctets.2
```

という名前で指定します。この監視対象(オブジェクト)の名前を「OID(Object ID)」と呼びます。また、「MIB」は「Management Information Base」の略で、OIDを定義している一種のデータベースです。**IF-MIB**という名前は、インターフェースに関するMIBであることを意味しています。**ifInOctets**は、インターフェース(if)で受信(In)したオクテット数(Octets[注1])をそれぞれ略し、連結した名前です。

SNMPはどう動くか

ここからは、フリーのSNMPマネージャである「Net-SNMP」の一部として配布されているユーティリティ「net-snmp-utils」と、本誌2015年3月号と4月号の記事『Cisco VIRLでネットワークのシミュレーション』で紹介した「Cisco VIRL[4]」を用いて、実際のSNMPの動作とデータモデルを見ていきます。

net-snmp-utilsは、次の手順でインストールできます。

・CentOS、Fedoraなど

```
# yum install net-snmp-utils
```

・Ubuntu(必要に応じてsudoを併用)

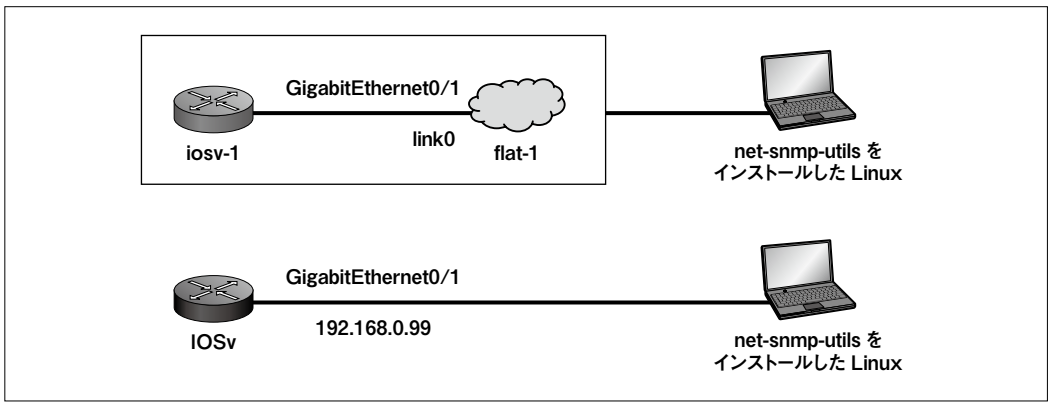
```
# apt-get install snmp
# apt-get install snmp-mibs-downloader
# ln -s /var/lib/mibs/* /usr/share/snmp/mibs
このあと、/etc/snmp/snmp.confの最後の行を「mibs +ALL」に変更
```

さらに、監視対象として、**図4の上側**の構成を用意します。Cisco VIRLに含まれている仮想化されたIOSルータ「IOSv」と、net-snmp-utilsをインストールしたLinux(CentOS 7.1)を接続した、とても単純な構成です。

図4中の「flat-1」という雲のアイコンは、Cisco VIRLと外部を接続するためのレイヤ2のブリッジです。これによって、論理的には**図4の下側**のように、IOSvとLinuxマシンが直接接続さ

注1) オクテットは、8ビットと同じ意味です。

▼図4　ネットワーク構成図(上：実際の構成、下：論理的な見え方)



れているように見え、IOSvにはIPv4アドレス「192.168.0.99」でアクセスできます。筆者が確認した範囲では、SNMPの動作に関しては、IOSvは従来のCisco IOSルータとまったく同様に動作します。ですから、この記事で紹介する動作例は、実機のIOSルータでも利用できます。

まず、IOSvにSNMPを使用するための最低限の設定をします。具体的には次のコマンドを1行入力します。

```
IOSv(config)#snmp-server community ↵
public RO
```

snmp-serverはSNMPエージェントを意味しています。**community public**によって、SNMPのパスワードに相当する「コミュニティ名」に**public**[注2]という文字列が含まれたリクエストを受信した場合に限り、アクセスを許可します。**RO**は「Read Only」の略で、リクエストによって値を読み込むだけであり、書き換えたりはしないことを意味します。このコマンドによって、IOSvの内部で動作しているSNMPエージェントに外部からアクセスすることが可能になります。

次に、Linux側でSNMPマネージャに相当する**snmpget**コマンドを用いて、SNMPリクエストを送信し、応答を確認します。

```
$ snmpget -v2c -c public 192.168.0.99 ↵
IF-MIB::ifInOctets.2
IF-MIB::ifInOctets.2 = Counter32: 84339
```

Counter32: 84339という応答が得られました。これは、このインターフェースの受信バイト数を、32ビット整数で表現したものです。**snmpget**のオプションのうち、**-v2c**はSNMPプロトコルの「バージョン2c[注3]」を、**-c**はコミュニティ名**public**をそれぞれ指定しています。

SNMPバージョン2cの場合、SNMPエージェント(ここではIOSv)側の設定と、リクエストの際に指定されたコミュニティ名が一致すれば、アクセスが許可されます。**192.168.0.99**は、**図4の下側**のIOSvルータに設定したIPv4アドレスで、このアドレスを用いてSNMPリクエストと応答を送受信します。

念のために、インターフェースの名前を確認します。**ifDescr**は、インターフェース(if)の説明(Description)[注4]を意味しています。

```
$ snmpget -v2c -c public 192.168.0.99 ↵
IF-MIB::ifDescr.1
IF-MIB::ifDescr.1 = STRING: ↵
GigabitEthernet0/0
```

このインターフェースは、この構成では使用していない**GigabitEthernet0/0**に対応しています。さらに、2番目のインターフェースを確認します。

```
$ snmpget -v2c -c public 192.168.0.99 ↵
IF-MIB::ifDescr.2
IF-MIB::ifDescr.2 = STRING: ↵
GigabitEthernet0/1
```

このインターフェースが、SNMPマネージャと接続している**GigabitEthernet0/1**であることがわかります。これで先に得られた、**IF-MIB::ifInOctets.2 = Counter32: 84339**が、IOSvルータの**GigabitEthernet0/1**の受信バイト数を示していることが確認できました。

SNMPのデータモデル

では、SNMPのデータモデルの正体に、少

注2) 例として用いている**public**は、コミュニティ名のデフォルトとしてよく用いられているものです。実際のネットワークでは、パスワードに準じて推測されにくい文字列を設定するか、この記事の後半で解説する「SNMPバージョン3」を使用してください。

注3) **v2C**と、「C」だけを大文字で書く場合もあります[5]。

注4) Cisco IOSでインターフェースのコメントを設定する**description**コマンドとは異なり、インターフェースの名前そのものを指します。

コラム 1

SNMPの動作形態

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

SNMPの世界では監視・管理する側をマネージャ、監視・管理される側をエージェントと呼びます。データの流れとしては次の2種類があります。

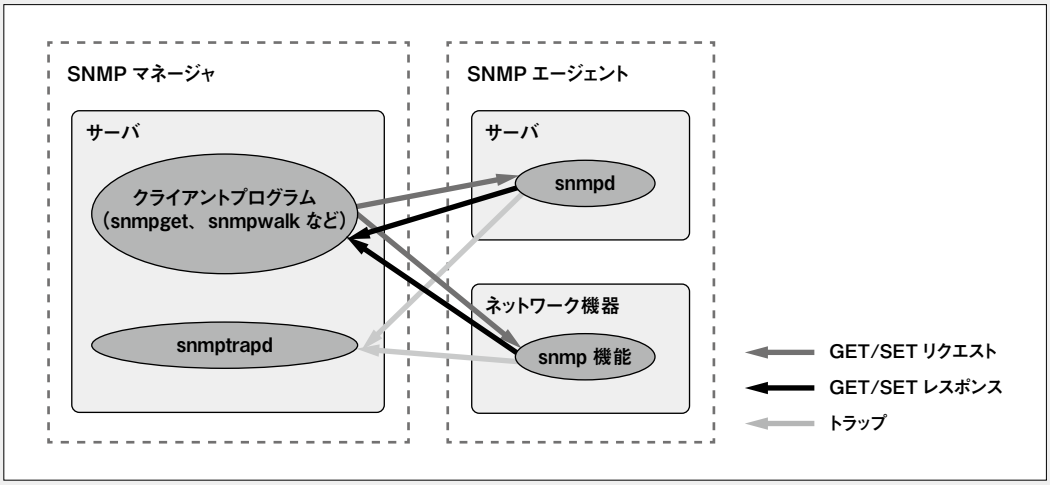
・マネージャがエージェントにリクエストを発行し、エージェントがレスポンスを返す
・エージェントからマネージャにデータを送信する(トラップ)

CentOSなどで採用されているSNMPマネージャ側のプログラムはSNMPのget/setリクエストを発行するsnmpget、snmpwalkなどと、SNMPトラップを受け付けるsnmptrapdがあります。

エージェント側のプログラムにはsnmpdがあります(**図A**)。

監視の基本はマネージャからの定期リクエストです。マネージャから定期的にアクセスし状態を確認することで、エージェントが何の前触れもなく突然ダウンしたときにも漏らさず検知できます。定期リクエストによるデータ取得の間隔よりも早く、よりリアルタイムに状況変化をキャッチしたい場合は、エージェントからのトラップを利用することで対応できます。家庭用のスイッチングHUBなどはとくに何も設定できないのですが、専門事業者が使うネットワーク機器ならば、telnetやsshでたいていログインでき、SNMPでの監視・管理ができます。このようにログインできたりSNMPでの監視・管理が可能な機能を俗にインテリジェント機能と呼びます。たとえばインテリジェント機能つきのL3スイッチをインテリL3スイッチと呼んだりします。データセンターの重要な個所で利用されるネットワーク機器(定価10万円程度以上のもの)には基本的にインテリジェント機能があります。

▼図A　SNMPマネージャとエージェントの構成例



し踏み込んでみます。次のコマンドは、先のコマンドにオプション-Of(full)を追加したものです。

```
$ snmpget -v2c -c public 192.168.0.99 ➥
-Of IF-MIB::ifInOctets.2
.iso.org.dod.internet.mgmt.mib-2.➥
interfaces.ifTable.ifEntry.ifInOctets.2 ➥
= Counter32: 84969
```

応答の前にあるOIDが、リクエストの際とはまったく異なる長い文字列になっています。これは、OIDを省略せずにテキスト形式で表現したものです。また、応答はCounter32: 84969となっており、最初のときより値が増えています。これは、このインターフェースが上記のSNMPのリクエストと応答そのものを送受信しているためです。

さらに、オプションを-On(numerically：数値で)に変更すると、次のようになります。

```
$ snmpget -v2c -c public 192.168.0.99 ➥
-On IF-MIB::ifInOctets.2
.1.3.6.1.2.1.2.2.1.10.2 = Counter32: 85296
```

.1.3.6.1.2.1.2.2.1.10.2は、OIDをSNMPリクエストとして実際に送受信されるフォーマット、すなわち、数値で記述したものです。

ここまでの例で現れた次の3つは、いずれも同じOIDを示しています。

① IF-MIB::ifInOctets.2
② .iso.org.dod.internet.mgmt.mib-2.interfaces.ifTable.ifEntry.ifInOctets.2
③ .1.3.6.1.2.1.2.2.1.10.2

もし想定しない形式でOIDが表示されてしまった場合は、snmptranslateコマンドに先ほどと同じ-Ofや-Onオプションを付けて実行することで変換できます。

説明を先のばしにしてきましたが、実はOIDは深い階層を持つ木構造のデータです。このOIDの場合、階層は**図5**のようになっています。木の頂点(Root)を除くすべての分岐点と葉、すなわち「ノード」には、数字と名前が割り当てられています。これらの数字と名前を、Rootに近いほうから順番に並べ、「.」で連結したものがOIDです。OIDは、身近なものに例えると、Linuxのファイルシステムのパス名で用いられている「/」の代わりに、「.」を用いて木構造を表現したものだととらえるとわかりやすいでしょう。

OIDの階層構造は相当深くなっています。これは、「OSI 7層モデル」に名前が残っている「OSI」という普及しなかったプロトコルの一部となれるよう、データモデルが定義されたためです。

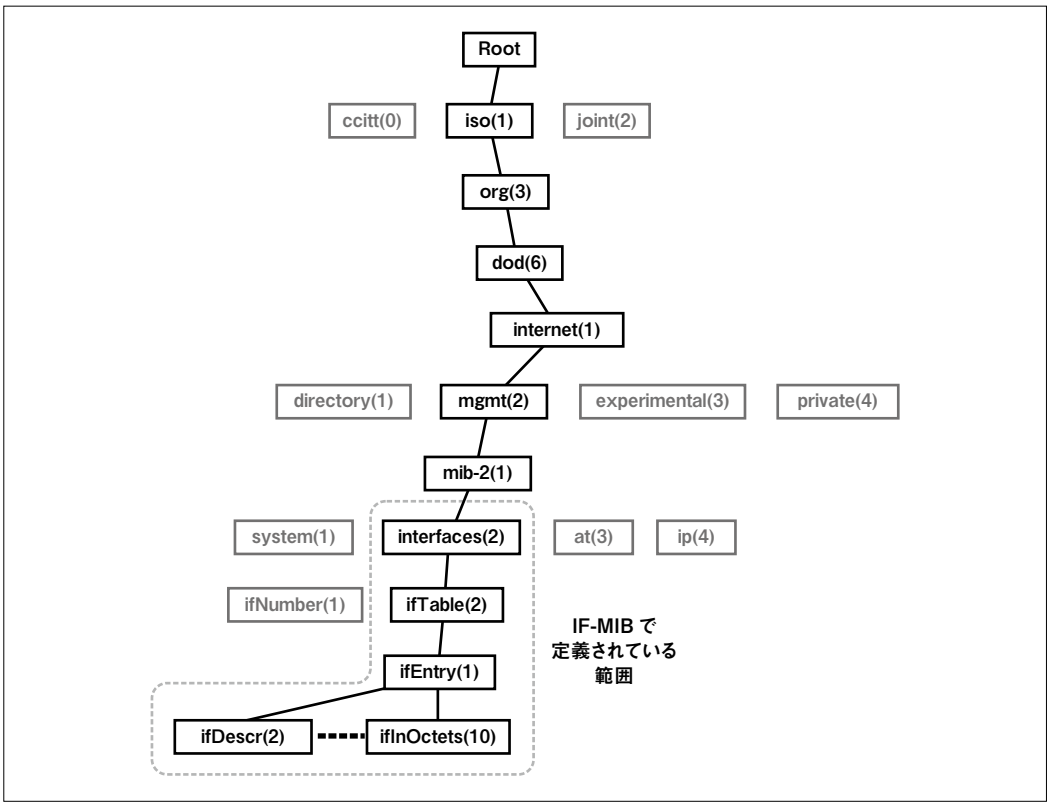
しかし、TCP/IPを用いる今日のネットワークでは、OIDの先頭にある.1.3.6.1はいつも同じです。ですので、この無駄な部分を削ってしまいたいところなのですが、OIDの互換性が失われてしまうため今日でも残っています。その代わりに、IF-MIB::ifInOctets.2のような省略形によって、OIDを指定したり表現できるよう工夫されています[注5]。

OIDの一番最後の部分、木構造の葉の1つ上にあたる「ifDescr(2)」や「ifInOctets(10)」は、監視対象になるデータの種類と型を表します。先に説明した動作例では、前者がSTRING、後者がCounter32というデータ型になっています。

インターフェースのように同じデータ構造が複数ある場合は、OIDの最後の数値(木構造の葉)が配列の「添字」の役割を果たします。IF-MIB::ifInOctets.2やIF-MIB::ifDescr.2の最後の「2」が、同じ2番めのインターフェースの、受信バイト数と名前を指定するために用いられています。

注5) 筆者が確認した限り、この省略形(モジュール形式)のOID名はNet-SNMPでのみ使えるようです。Net-SNMPを含まないほかのSNMPマネージャでは、省略形が使えない可能性があります。

▼図5　SNMPのデータモデル



コラム 2

SNMPのデータモデル

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

SNMPの世界では管理上で欲しいデータを表す方法としてMIB(Management Information Base)を使います。MIBは.(ドット)で始まるツリー構造になっていて、各階層・要素が数字で表されています。これをOID(Object Identifier)と呼びます。

CentOS 7のデフォルトインストール状態では、**.1.3.6.1.2.1.1**(SNMPv2-MIB::system)、**.1.3.6.1.2.1.25.1.1**(HOST-RESOURCES-MIB::hrSystemUptime)にアクセスできるよう設定されています。

各OIDはそれぞれ名前が付いています。数字のOID(**.1.3.6.1.2.1.1**)とテキストの名称(SNMPv2-MIB::system)は同じものを指します。そのためSNMPリクエストを発行する際に**.1.3.6.1.2.1.1**と指定するのと、SNMPv2-MIB::systemと指定するのは同じ意味になります。この2つは**snmptranslate**コマンドで変換できます。

```
[root@manager ~]# snmptranslate ↵
.1.3.6.1.2.1.1
SNMPv2-MIB::system
```

SNMPを利用してエージェントとデータのやりとりをする際には、このOIDを数字またはテキストで指定し、データを取得します。

```
[root@manager ~]# snmptranslate .1
iso
[root@manager ~]# snmptranslate .1.3
SNMPv2-SMI::org
[root@manager ~]# snmptranslate .1.3.6
SNMPv2-SMI::dod
[root@manager ~]# snmptranslate .1.3.6.1
SNMPv2-SMI::internet
[root@manager ~]# snmptranslate .1.3.6.1.2
SNMPv2-SMI::mgmt
[root@manager ~]# snmptranslate ↵
.1.3.6.1.2.1
SNMPv2-SMI::mib-2
[root@manager ~]# snmptranslate ↵
.1.3.6.1.2.1.1
SNMPv2-MIB::system
```

なおテキストから数字に変換する場合は**-On**オプションを利用します。

```
[root@manager ~]# snmptranslate -On ↵
SNMPv2-MIB::system
.1.3.6.1.2.1.1
```

MIBは独自拡張可能なため、各ネットワーク機器ベンダは独自のMIBを持っています。UP/DOWNなどポートのステータスや、ポートごとのトラフィックなどを取得できます。また変わったところではJVM(Java VM)もMIBを持っており、SNMPでステータスを取得できます。

CentOS 7の**snmpd**では特定のOIDにアクセスすると任意の処理を実行させる設定ができます。**/etc/snmpd/snmpd.conf**で**exec**を設定することで実現できます。

myuptimeという名前で**uptime**コマンドを実行するOIDを定義する例は次のとおりです。

```
exec myuptime /bin/uptime
```

このように設定すると、**.1.3.6.1.4.1.8072.1.3**(NET-SNMP-AGENT-MIB::nsExtensions)配下にアクセスすることで、このコマンドを実行し結果を取得できるようになります(**図B**)。

▼図B　snmpwalkコマンド実行例

```
[root@manager ~]# snmpwalk -v 2c -c public agent .1.3.6.1.4.1.8072.1.3
NET-SNMP-EXTEND-MIB::nsExtendNumEntries.0 = INTEGER: 1
NET-SNMP-EXTEND-MIB::nsExtendCommand."myuptime" = STRING: /bin/uptime
NET-SNMP-EXTEND-MIB::nsExtendArgs."myuptime" = STRING:
NET-SNMP-EXTEND-MIB::nsExtendInput."myuptime" = STRING:
NET-SNMP-EXTEND-MIB::nsExtendCacheTime."myuptime" = INTEGER: 5
NET-SNMP-EXTEND-MIB::nsExtendExecType."myuptime" = INTEGER: exec(1)
NET-SNMP-EXTEND-MIB::nsExtendRunType."myuptime" = INTEGER: run-on-read(1)
NET-SNMP-EXTEND-MIB::nsExtendStorage."myuptime" = INTEGER: permanent(4)
NET-SNMP-EXTEND-MIB::nsExtendStatus."myuptime" = INTEGER: active(1)
NET-SNMP-EXTEND-MIB::nsExtendOutput1Line."myuptime" = STRING:  08:11:31 up  3:53, ↵
1 user,  load average: 0.00, 0.01, 0.05
NET-SNMP-EXTEND-MIB::nsExtendOutputFull."myuptime" = STRING:  08:11:31 up  3:53,  1 user, ↵
load average: 0.00, 0.01, 0.05
NET-SNMP-EXTEND-MIB::nsExtendOutNumLines."myuptime" = INTEGER: 1
NET-SNMP-EXTEND-MIB::nsExtendResult."myuptime" = INTEGER: 0
NET-SNMP-EXTEND-MIB::nsExtendOutLine."myuptime".1 = STRING:  08:11:31 up  3:53,  1 user, ↵
load average: 0.00, 0.01, 0.05
```

3 SNMP実用編

Author 山下 薫(やました かおる) URL http://www.linkedin.com/in/kaoruyamashita

ここでは、SNMPを使った運用の現場でよく使われるコマンドと、「トラップ」のしくみについて解説します。また、SNMPの新しいバージョン「3」の特徴をまとめました。

snmpwalkでOIDと値をまとめて取得

ここまでのSNMPの動作例では、監視対象となるOIDが前もってわかっていることが前提でした。これに対して、Net-SNMP(net-snmp-utils)には、ファイルシステムに対する`ls -R`コマンドと同様の動作をする、`snmpwalk`コマンドが含まれています。

たとえば、**図5**にある「IF-MIBで定義されている範囲」に含まれるすべてのOIDとその値は、**図6**のようにして取得できます。この例でアクセスしているIOSvには、2つの物理インターフェースと、「Null0」という論理インターフェースがあり、それぞれの状態がわかります。Null0は、Linuxの/dev/nullに相当する論理インターフェースです。

`ls -R /`に相当する操作も可能で、木構造のRootに対応するOIDである「.」を指定する[注6]と、対象になる機器のすべてのOIDとその値の一覧が出力されます(**図7**)。このように、OIDがわからなくても一覧が取得できるので、監視対象にしたい値に対応するOIDが存在するかどうかを、対象となる機器(SNMPエージェント)に実際に問い合わせて確認できます。

一方、実際にアクセスしてOIDの木構造を調べるのではなく、Net-SNMP側にインストールされているMIBをもとに、SNMPマネージャ側だけでOIDの構成を表示させることもできます。この目的にも、`snmptranslate`コマンドが使えます。**図5**の「IF-MIBで定義されている範囲」に対して`snmptranslate`を実行すると、**図8**のような出力が得られます。Rootからたどることもできます。

注6)「.」を省略しても、同じ動作をします。

▼図6　図5中「IF-MIBで定義されている範囲」に含まれるすべてのOIDとその値を取得

```
$ snmpwalk -v2c -c public 192.168.0.99 .iso.org.dod.internet.mgmt.mib-2.interfaces
IF-MIB::ifNumber.0 = INTEGER: 3
IF-MIB::ifIndex.1 = INTEGER: 1
IF-MIB::ifIndex.2 = INTEGER: 2
IF-MIB::ifIndex.3 = INTEGER: 3
IF-MIB::ifDescr.1 = STRING: GigabitEthernet0/0
IF-MIB::ifDescr.2 = STRING: GigabitEthernet0/1
IF-MIB::ifDescr.3 = STRING: Null0
...(中略)...
IF-MIB::ifAdminStatus.1 = INTEGER: down(2)
IF-MIB::ifAdminStatus.2 = INTEGER: up(1)
IF-MIB::ifAdminStatus.3 = INTEGER: up(1)
IF-MIB::ifOperStatus.1 = INTEGER: down(2)
IF-MIB::ifOperStatus.2 = INTEGER: up(1)
IF-MIB::ifOperStatus.3 = INTEGER: up(1)
...(以下略)...
```

OIDの値を書き換える

SNMPでは、OIDの値を読み出すだけではなく、書き換えることで機器の状態を変えることもできます。実際にやってみましょう。まず、Cisco IOS(IOSv)に設定を追加します。

```
# snmp-server community private RW
```

コミュニティ名として**private**[注7]を、読み書き両方が可能なことを**RW**(Read/Write)でそれぞれ指定します。

では、IOSvのインターフェースの状態を変更してみましょう。先ほどの**snmpwalk**の結果をよく見ると、1番目のインターフェースである**GigabitEthernet 0/0**の名前とOID、値は次のようになっています。

```
IF-MIB::ifDescr.1 = STRING: GigabitEthernet0/0
IF-MIB::ifAdminStatus.1 = INTEGER: down(2)
IF-MIB::ifOperStatus.1 = INTEGER: down(2)
```

ifAdminStatusは管理者によって設定された状態を示し、**ifOperStatus**は実際のインターフェースの状態を示しています。この時点では、**GigabitEthernet 0/0**には**shutdown**コマンド

注7) 説明をわかりやすくするため、デフォルトでよく用いられているものをコミュニティ名としてそのまま使っています。実際のネットワークでは、セキュリティに配慮してください。

▼図7 対象になる機器のすべてのOIDとその値の一覧を出力

```
$ snmpwalk -v2c -c public 192.168.0.99 .
iso.0.8802.1.1.2.1.3.1.0 = INTEGER: 4
iso.0.8802.1.1.2.1.3.2.0 = Hex-STRING: FA 16 3E 34 45 EB
iso.0.8802.1.1.2.1.3.3.0 = STRING: "IOSv"
iso.0.8802.1.1.2.1.3.4.0 = STRING: "Cisco IOS Software, IOSv Software
(VIOS-ADVENTERPRISEK9-M), Version 15.5(2)T,
...(以下略)...
```

▼図8 SNMPマネージャ側だけでOIDの構成を表示

```
$ snmptranslate -Tp .iso.org.dod.internet.mgmt.mib-2.interfaces
+--interfaces(2)
   |
   +-- -R-- Integer32 ifNumber(1)
   |
   +--ifTable(2)
      |
      +--ifEntry(1)
         |  Index: ifIndex
         |
         +-- -R-- Integer32 ifIndex(1)
         |        Textual Convention: InterfaceIndex
         |        Range: 1..2147483647
         +-- -R-- String    ifDescr(2)
         |        Textual Convention: DisplayString
         |        Size: 0..255
...(中略)...
         +-- -RW- EnumVal   ifAdminStatus(7)
         |        Values: up(1), down(2), testing(3)
         +-- -R-- EnumVal   ifOperStatus(8)
         |        Values: up(1), down(2), testing(3), unknown(4), dormant(5),
notPresent(6), lowerLayerDown(7)
...(以下略)...
```

が投入されていて、インターフェースが実際には何かに接続されていても、ダウンしたままになります。

ここからはsnmpsetコマンドを用いて、インターフェースの状態を変更してみます。snmpgetの場合とは異なり、i 1でOIDの新しい値を指定します。iは「INTEGER」の略で、1は「up」を意味しています。

```
$ snmpset -v2c -c private 192.168.0.99 ↵
IF-MIB::ifAdminStatus.1 i 1
IF-MIB::ifAdminStatus.1 = INTEGER: up(1)
```

すると、IOSvルータの側でもインターフェースの状態が変化します(**リスト1**)。192.168.0.148は、SNMPマネージャNet-SNMPが動作しているLinuxマシンのIPv4アドレスです。IOSvの側でshow run int Gi0/0を実行して確認すると、もともと入っていたshutdownコマンドが消えています。

SNMPトラップ

ここまでで解説したSNMPのプロトコルを用いて実際に監視をするには、定期的に監視対象にアクセスして値を取得すること、すなわち「ポーリング」が必要です。もし監視対象に何らかの変化があっても、次回のポーリングまではそれを知ることができません。このため、監視対象であるSNMPエージェントから、SNMPマネージャに対して変化があったことをすぐに通知することで、より早く監視対象の変化を把握できるしくみが用意されています。これが「SNMPトラップ」です(**図9**)。

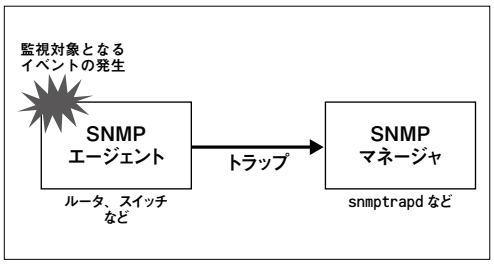
snmpgetやsnmpsetでは、SNMPエージェントがサーバのように働き、リクエストを待っていました。SNMPトラップではこれと逆になり、SNMPエージェントから送信されるトラップを、SNMPマネージャが待ち受けます。Net-SNMPには、「snmpd」と「snmptrapd」の2つのデーモンが含まれていますが、snmpdはそれが動作しているLinuxマシンにSNMPエージェントの役割を持たせるためのもので、トラップを待ち受ける機能はありません。代わりに、snmptrapdがSNMPトラップを受信し、ログファイルにその内容を記録したり、管理者にメールで通知したりします。

SNMPは、バージョンに関係なくUDP(User Datagram Protocol)の161番ポートをリクエストと応答に、162番ポートをトラップに用います。もしsnmpdとsnmptrapdの2つのデーモンを同じLinuxマシンで動作させても、それぞれ161番と162番ポートで別途パケットを待ち受けるため、誤動作することはありません。

では、実際にCisco IOS(IOSv)とNet-SNMPを用いて、SNMPトラップを送受信してみましょう。まず、snmptrapdの設定ファイル「/etc/snmp/snmptrapd.conf」を次のように書き換えて、snmptrapdを再起動します。

```
authCommunity   log,execute,net public
traphandle default /usr/bin/logger
```

▼図9　SNMPトラップ



▼リスト1　snmpsetで変更されたインターフェースの状態

```
*Oct 18 01:01:37.471: %SYS-5-CONFIG_I: Configured from 192.168.0.148 by snmp
*Oct 18 01:01:39.340: %LINK-3-UPDOWN: Interface GigabitEthernet0/0, changed state to up
*Oct 18 01:01:40.340: %LINEPROTO-5-UPDOWN: Line protocol on Interface GigabitEthernet0/0, ↵
changed state to up
```

コラム 3

CentOSでの利用方法

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

CentOS 7でSNMPを利用する場合はnet-snmpパッケージを利用します。net-snmpパッケージには、SNMP要求を受け付けるsnmpdと、SNMPトラップを受け付けるsnmptrapdが含まれています。また、net-snmp-utilsにsnmpgetやsnmpwalkなどのクライアントツールが含まれています。

agentというホスト名のサーバでsnmpdをインストールする方法は次のとおりです。

```
[root@agent ~]# yum install net-snmp ➥
net-snmp-utils
[root@agent ~]# systemctl start snmpd
[root@agent ~]# firewall-cmd --add-port ➥
161/udp
```

自動起動設定と、設定の永続化も忘れずに実施しておきましょう。

```
[root@agent ~]# systemctl enable snmpd
[root@agent ~]# firewall-cmd --add-port ➥
161/udp --permanent
```

インストール直後の/etc/snmp/snmpd.confは**図C**のとおりです。コミュニティ文字列public、SNMPバージョンv1、v2cで、.1.3.6.1.2.1.1(SNMPv2-MIB::system)、.1.3.6.1.2.1.25.1.1(HOST-RESOURCES-MIB::hrSys0temUptime)にアクセスできる設定です。

syslocationやsyscontactを設定できるあたりが、システムリソース利用状況だけでなく、システム管理そのものも考えられている感がありますね。

snmpgetやsnmpwalkを使うことでエージェントからデータを取得できます。snmpgetはMIBのOIDを決め打ちで取得し、snmpwalkはMIBのOIDを指定するとその配下を一通り取得します。

たとばmanagerというホスト名のサーバからagentというホスト名のサーバにアクセスし、前出のsysLocationをsnmpwalkで取得する方法は次のとおりです。snmpwalkの場合は、sysLocation.0ではなくsysLocationと指定しても値を確認できます。

```
[root@manager ~]# snmpwalk -v 2c -c ➥
public agent sysLocation.0
SNMPv2-MIB::sysLocation.0 = STRING: ➥
Unknown (edit /etc/snmp/snmpd.conf)
```

snmpgetの場合は次のとおりです。snmpgetの場合はsysLocationではダメで、sysLocation.0まで指定する必要があります。

▼図C インストール直後の/etc/snmp/snmpd.conf

```
com2sec notConfigUser  default  public
group   notConfigGroup v1       notConfigUser
group   notConfigGroup v2c      notConfigUser
view    systemview     include  .1.3.6.1.2.1.1
view    systemview     include  .1.3.6.1.2.1.25.1.1
access  notConfigGroup ""       any  noauth exact  systemview none none
syslocation Unknown (edit /etc/snmp/snmpd.conf)
syscontact Root <root@localhost> (configure /etc/snmp/snmp.local.conf)
dontLogTCPWrappersConnects yes
```

```
[root@manager ~]# snmpget -v 2c -c ➥
public agent sysLocation.0
SNMPv2-MIB::sysLocation.0 = STRING: ➥
Unknown (edit /etc/snmp/snmpd.conf)
```

snmpwalkの場合は**.**(ドット1つ＝ルートという意味)を指定して、取得できる全データを確認できます。ぜひ一度試してみてください。

```
[root@manager ~]# snmpwalk -v 2c -c ➥
public agent .
```

CentOS 7 の **snmpd** は **ucd-snmp/proxy** モジュールが組み込まれており、proxy機能が利用できます。これを利用することでmanager→agent→agentという踏み台構成を実現できます。

proxy機能を使うためには**/etc/snmp/snmpd.conf**にcom2secとproxyを設定します。

コンテキストは**snmpd.conf**内でユニークな文字列であればよいです。このコミュニティ文字列は中間エージェントにアクセスする際のコミュニティ文字列です。

```
com2sec -Cn コンテキスト notConfigUser ➥
default コミュニティ文字列
```

コンテキストはcom2secと同じものを指定します。このコミュニティ文字列は最終エージェントにアクセスする際のコミュニティ文字列です。

設定例は次のとおりです。

```
proxy   -Cn コンテキスト -v 2c -c ➥
コミュニティ文字列 ホスト OID
```

このように中間エージェントにアクセスする際のコミュニティ文字列を最終サーバごとに異なるものにすることで、各最終エージェントは同じコミュニティ文字列(今回はpublic)を受け付ける設定にしておくことができます。

```
com2sec -Cn remote-child-1 ➥
notConfigUser default child-1-public
proxy -Cn remote-child-1 -v 2c -c ➥
public child-1 .1.3
```

このように設定すると、コミュニティ文字列**child-1-public**を指定して中間エージェントにアクセスすることで、最終エージェントchild-1の値を取得できます。

この方法で1つの踏み台を経由して大量のバックエンドにアクセスできます。

また、IOSvに次の設定をします。

```
IOSv(config)#snmp-server enable traps ➥
snmp linkdown linkup
IOSv(config)#snmp-server host ➥
192.168.0.148 version 2c public
```

これで、インターフェースの状態が変化した際に、SNMPトラップがsnmptrapd(この例では192.168.0.148で動作)へ送信されます。snmptrapdが動作しているLinuxマシンで、**tail -f /var/log/messages**を実行すれば、SNMPトラップの内容がわかります。たとえば、IOSvの**GigagitEthernet0/0**がダウンした際には、**リスト2**のようなログメッセージが記録されます。

snmptrapdは、トラップの受信をトリガにメールを送信したり、コマンドを実行することもできます。また、Cisco IOSでは、**リスト2**の例のインターフェースのリンクアップ／リンクダウン以外にも、SNMPトラップの対象になるイベントを細かく設定できます[6]。単に**snmp-server enable traps**だけを設定すると、

サポートされているすべての種類のSNMPトラップが送信されるようになってしまいますので注意してください。

SNMPバージョン3（SNMPv3）

ここまで説明したSNMPバージョン2cは、コミュニティ名さえ一致していればアクセスを許可するものでした。また、SNMPバージョン2cでは通信はまったく暗号化されていないので、コミュニティ名は平文でネットワーク上を流れます。このため、SNMPのパケットを傍受できれば、コミュニティ名を知ることはとても簡単です。いったんコミュニティ名が漏れてしまうと、監視対象の詳細な情報が知られたり、機器の状態を勝手に変えられたりてしまいます。

このため、セキュリティを強化した「SNMPバージョン3」が標準化されています[5]。

SNMPバージョン3は、OIDなどのデータモデルはバージョン2cとまったく同じで、認証と暗号化のしくみを強化したプロトコルです。

SNMPバージョン3では、認証と通信の暗号化にそれぞれ別の方式を用います。また、USM（User-based Security Model）という概念が追加されています。具体的には、ユーザとグループ、さらにOIDのどの範囲のアクセスを許可するかなどを細かく設定できます。

図10に、Cisco IOS（IOSv）をSNMPバージョン3のエージェントとして動作させるための設定例を示します。(1)でグループを登録します。グループには**view**という属性があり、**図5**のOIDの木構造のどの部分にアクセスできるかを制限できます。この例の(2)と(3)では**iso**を指定しているので、事実上すべてのOIDにアクセスできます。(4)が、ユーザ名とパスワードの設定です。ユーザ認証とSNMPプロトコルの暗号化が分かれているので、それぞれの暗号化方式とパスワードを指定します。**auth md5 authPasswd**は、認証に**authPasswd**というパスワードを用いてMD5を使い、**priv des privPasswd**は暗号化に**privPasswd**というパスフレーズを用いてDESを使うことを、それぞ

▼リスト2　IOSv（Cisco IOS）のGigagitEthernet0/0がダウンし、SNMPトラップが送信された際のログメッセージ

```
Oct 18 20:01:39 localhost snmptrapd[3279]: 2015-10-18 20:01:39 <UNKNOWN> [UDP: ➥
[192.168.0.99]:62684->[192.168.0.148]:162]:
DISMAN-EVENT-MIB::sysUpTimeInstance = Timeticks: (21122011) 2 days, 10:40:20.11
SNMPv2-MIB::snmpTrapOID.0 = OID: IF-MIB::linkDown
IF-MIB::ifIndex.1 = INTEGER: 1  IF-MIB::ifDescr.1 = STRING: GigabitEthernet0/0
IF-MIB::ifType.1 = INTEGER: ethernetCsmacd(6)
SNMPv2-SMI::enterprises.9.2.2.1.1.20.1 = STRING: "Lost Carrier"
Oct 18 20:01:39 localhost logger: UDP: [192.168.0.99]:62684->[192.168.0.148]:162
Oct 18 20:01:39 localhost logger: DISMAN-EVENT-MIB::sysUpTimeInstance 2:10:40:20.11
Oct 18 20:01:39 localhost logger: SNMPv2-MIB::snmpTrapOID.0 IF-MIB::linkDown
Oct 18 20:01:39 localhost logger: IF-MIB::ifIndex.1 1
Oct 18 20:01:39 localhost logger: IF-MIB::ifDescr.1 GigabitEthernet0/0
Oct 18 20:01:39 localhost logger: IF-MIB::ifType.1 ethernetCsmacd
Oct 18 20:01:39 localhost logger: SNMPv2-SMI::enterprises.9.2.2.1.1.20.1 "Lost Carrier"
```

▼図10　IOSvをSNMPバージョン3のエージェントとして動作させるための設定例

```
IOSv(config)#snmp-server group V3Group v3 auth read V3Read write V3Write          (1)
IOSv(config)#snmp-server view V3Read iso included                                 (2)
IOSv(config)#snmp-server view V3Write iso included                                (3)
IOSv(config)#snmp-server user V3User V3Group v3 auth md5 authPasswd priv des privPasswd (4)
IOSv(config)#snmp-server host 192.168.0.148 version 3 auth V3User                 (5)
```

コラム 4

SNMPのバージョン

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

SNMPはバージョンとしてversion 1(v1)、version 2(v2、v2p、v2c、v2u)、version 3(v3)があります。最新版のv3を使うのがよさそうなものですが、筆者がCentOSで使ってみたところ、どうにもうまく動かないことが多く、高負荷時や大量取得時にデータがなぜか取得できない事態が頻発したためv2cを利用しています。CentOS 7の**snmpd**でv3を利用するには、いくつかの段取りが必要です。

今回はOID**.1**にreadonlyアクセス可能なユーザとしてユーザ名myuserを作成します。まずはユーザの設定を**/etc/snmp/snmpd.conf**に記載します。

```
rouser myuser priv .1
```

次にcreateUserを**/etc/snmp/snmpd.conf**に記載します。今回はハッシュ方式SHA、認証パスワード**authp@ssw0rd**、暗号化方式AES、暗号化パスフレーズ**cryptp@ssw0rd**を利用することにします。

```
createUser myuser SHA authp@ssw0rd AES ➥
cryptp@ssw0rd
```

rouserと**createUser**を記載したら**snmpd**を再起動します。

```
[root@agent ~]# systemctl restart snmpd
```

再起動すると**/var/lib/net-snmp/snmpd.conf**にmyuser関連の記載が追加されます。

ここまでできたらユーザ作成完了なので**/etc/snmp/snmpd.conf**の**createUser**行を削除しておきます。v3を利用して**snmpwalk**でアクセスする方法は次のとおりです。

v2のときと比較するとコミュニティ文字列の指定がなく、**-u -l -a -A -x -X**の指定が必要となります。

```
[root@manager ~]# snmpwalk -v 3 -u ➥
myuser -l authPriv -a SHA -A ➥
authp@ssw0rd -x AES -X cryptp@ssw0rd ➥
agent sysLocation.0
```

v3と比較してv2cではデータの暗号化ができません。パスワードに相当するコミュニティ名も平文で流れます。v3を利用することでデータを暗号化できるようになります。SNMPはManagement Protocolの名のとおり、情報取得(GET)だけでなく、設定変更(SET)もできます。とはいえプロトコルがUDPということもあり、セキュリティ対策としての接続元制限はあまり有効に機能しない可能性が考えられます。通常SNMPは通信にUDPを使いますが、Net-SNMP 5.6以上であればTCPも選択可能となるため、TLSによる経路暗号化が利用できます。ただし最近はSSHやプロビジョニングツールを使うのが主流ですので、正直なところSNMPでの設定変更は出番がありません。

れ意味します[注8]。(5)で、SNMPバージョン3でアクセスするマネージャとユーザ名を設定します。

Net-SNMPからSNMPバージョン3でアクセスする際は、認証と暗号化のためのパラメータが増えるだけで、それ以外はSNMPバージョン2cと同じです。

```
# snmpwalk -v3 -u V3User -a MD5 -A ↵
authPasswd -x DES -X privPasswd -l ↵
authPriv 192.168.0.99 .1.3.6.1.2.1.2.2.1
```

-a、**-A**と**-x**、**-X**が、Cisco IOS側の(4)で設定した認証と暗号化のためのパラメータに対応します。これら4つのすべてが一致する場合に限り、アクセスが許可されます。

注8) Cisco IOSでは、(4)を設定しても**show run**では表示されません。

Column セキュリティのためのTips

Cisco IOSが動作しているルータやスイッチでは、SNMPマネージャのIPアドレスを制限することによって、ある程度ですがセキュリティを高めることができます。たとえば、次の設定をすると、192.168.0.148というIPv4アドレスを持ったSNMPマネージャからのリクエストのみを受けつけ、ほかは拒否します。

```
IOSv(config)#access-list 1 permit 192.168.0.148
IOSv(config)#snmp-server community public RO 1
```

4 SNMP応用編

Author 山下 薫(やました かおる) URL http://www.linkedin.com/in/kaoruyamashita

ここでは、スイッチの扱い、ベンダ固有のOIDなど、SNMPを利用するうえで気をつけておきたいことを解説します。また、OIDのビット数、可視化の方法、時刻同期といった少し踏み込んだ知識もお伝えしたいと思います。

スイッチにおけるSNMP

今日のスイッチは、ルーティングなどルータの基本機能の多くを持っています。また、Cisco Catalystのように、ルータと共通のOSが動作するスイッチでは、SNMPの挙動はルータと大きく変わりません。とはいえ機器の構造が異なっているので、SNMPで監視する際の振る舞いが異なることがあります。この項では、SNMPという切り口から見たスイッチの2つの特徴を説明します。

インターフェースの多さ

1つめは、ルータに比べてインターフェースが多いことです。また、設定の変更やハードウェアの増設によって、インターフェースの数が増減することが、ルータに比べると頻繁に起こります。

具体例を見ていきましょう。**図11**のリストは、先に説明したルータと場合とまったく同じやり方(**snmpwalk**)で、インターフェースとそれ

に対応するすべてのOIDを列挙したものです。ここで用いているスイッチはCisco Catalyst 3560で、**192.168.0.2**はこのCatalystに割り当てているIPv4アドレスです。

まず、**IF-MIB::ifNumber.0 = INTEGER: 16**が、インターフェースが全部で16個あることを示しています。OIDの最後の**.0**は、このOIDが配列ではなく、要素を1つしか持たない値に対応していることを意味します。

次の**ifIndex**が、各インターフェースの論理的な番号(インデックス)です。インターフェースが増減しても番号を振りなおさなくて済むように、飛び飛びの値が割り振られています。具体的には**1**、**100**、**128**がVLANインターフェース(SVI)、**10101**から**10112**が物理インターフェース、**10501**がNull0にそれぞれ対応します。

先に説明したルータの例では、**GigabitEthernet0/1**の受信バイト数に対するOIDは**IF-MIB::ifInOctets.2**で、ifIndexは**2**でした。

このスイッチの例では、**ifDescr**から**GigabitEthernet0/1**の**ifIndex**が**10101**であることがわかります。ですから次のやり方で、ルータと同じように**GigabitEthernet0/1**の受信バイト数を知ることができます。

```
$ snmpget -v2c -c public 192.168.0.2 ➥
IF-MIB::ifInOctets.10101
IF-MIB::ifInOctets.10101 = Counter32: ➥
17257969
```

▼図11　インターフェースとそれに対応するすべてのOIDを列挙

```
$ snmpwalk -v2c -c public 192.168.0.2 .iso.org.dod.internet.mgmt.mib-2.interfaces
IF-MIB::ifNumber.0 = INTEGER: 16
IF-MIB::ifIndex.1 = INTEGER: 1
IF-MIB::ifIndex.100 = INTEGER: 100
IF-MIB::ifIndex.128 = INTEGER: 128
IF-MIB::ifIndex.10101 = INTEGER: 10101
IF-MIB::ifIndex.10102 = INTEGER: 10102
IF-MIB::ifIndex.10103 = INTEGER: 10103
IF-MIB::ifIndex.10104 = INTEGER: 10104
IF-MIB::ifIndex.10105 = INTEGER: 10105
IF-MIB::ifIndex.10106 = INTEGER: 10106
IF-MIB::ifIndex.10107 = INTEGER: 10107
IF-MIB::ifIndex.10108 = INTEGER: 10108
IF-MIB::ifIndex.10109 = INTEGER: 10109
IF-MIB::ifIndex.10110 = INTEGER: 10110
IF-MIB::ifIndex.10111 = INTEGER: 10111
IF-MIB::ifIndex.10112 = INTEGER: 10112
IF-MIB::ifIndex.10501 = INTEGER: 10501
IF-MIB::ifDescr.1 = STRING: Vlan1
IF-MIB::ifDescr.100 = STRING: Vlan100
IF-MIB::ifDescr.128 = STRING: Vlan128
IF-MIB::ifDescr.10101 = STRING: GigabitEthernet0/1
IF-MIB::ifDescr.10102 = STRING: GigabitEthernet0/2
IF-MIB::ifDescr.10103 = STRING: GigabitEthernet0/3
IF-MIB::ifDescr.10104 = STRING: GigabitEthernet0/4
IF-MIB::ifDescr.10105 = STRING: GigabitEthernet0/5
IF-MIB::ifDescr.10106 = STRING: GigabitEthernet0/6
IF-MIB::ifDescr.10107 = STRING: GigabitEthernet0/7
IF-MIB::ifDescr.10108 = STRING: GigabitEthernet0/8
IF-MIB::ifDescr.10109 = STRING: GigabitEthernet0/9
IF-MIB::ifDescr.10110 = STRING: GigabitEthernet0/10
IF-MIB::ifDescr.10111 = STRING: GigabitEthernet0/11
IF-MIB::ifDescr.10112 = STRING: GigabitEthernet0/12
IF-MIB::ifDescr.10501 = STRING: Null0
...(以下略)...
```

OID値の更新の遅れ

もう1つのSNMPにおけるスイッチの特徴は、OIDの値の更新の遅れです。

たとえばスイッチにおいて、インターフェースの受信バイト数が急激に増加していても、SNMPリクエストの応答がリアルタイムでは増加しないことがあります。これは、スイッチにおけるパケット処理とSNMPエージェントの実装が理由です(**図12**)。

今日のスイッチでは、パケット処理はすべて専用のハードウェア(ASIC)が担当しています[注9]。このため、パケット数や送受信のバイト数がリアルタイムで記録されるのは、ASICの内部にある各種ハードウェアカウンタです。SNMPリクエストに応じて、この値を毎回読み出すのであれば、OIDの値として必ず最新のものが返されるはずです。

しかし、SNMPリクエストのたびにASICにアクセスしていると、スイッチにとってもっとも優先度の高いASICの制御の邪魔になり、最悪の場合スイッチの動作が不安定になる可能性があります。そこで、SNMPリクエストとは無関係に、一定間隔でASICのハードウェアカウンタを読み出してソフトウェア側に値を反映しています。

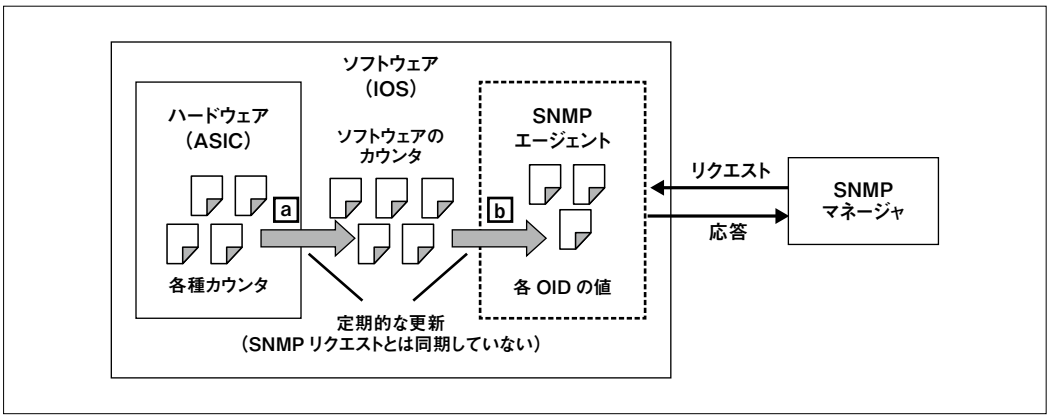
Cisco IOSが動作しているCatalystスイッチでは、IOSのアーキテクチャに基づいたカウンタと、それをSNMP OIDに対応するよう変換したものの2種類のソフトウェアカウンタを管理しています。

図12に示すように、ASICのハードウェアカウンタが「**a**」の矢印の向きで、IOSのソフトウェアカウンタに一定間隔で反映されます。さらに「**b**」の矢印の方向へ、別の時間間隔でIOSのソフトウェアカウンタがOIDの値に変換されます。実際に筆者が試してみたところ、Catalyst 3560では1秒間隔で「**a**」が実行されていました。また、Cisco IOSでは「**b**」はデフォルトで5秒間隔で実行されるため、連続してインターフェースの受信バイト数の値を読み出しても、5〜6秒程度は同じ値が返ってきます。

この挙動が問題になるのは、SNMPで定期的にスイッチのOIDの値を確認し、値の増加が止まった場合には何らかの問題が発生したと判断して、バックアップ系に切り替えるといったスクリプトをしかけるような場合です。このスクリプトで、「値の増加が止まった」と判断するまでの時間を短くし過ぎると、誤動作する可能性があります。Cisco IOSでは、`snmp-server hc poll` や `service counters`

注9) OvSやハイパーバイザ内蔵のvSwitchのように、パケットをソフトウェア処理している場合もあります。

▼図12 スイッチでのSNMPの実装



max ageコマンドによって、OIDの値の更新間隔を短くすることは可能[7]ですが、CPU負荷が高い場合には応答が遅れる場合があります。

また、モジュール型の大型のスイッチなどでは、パケット転送を担当するASICとそれを制御するCPUが複数あり、並列化されているため、小型のスイッチに比べるとOIDの値の更新がさらに遅くなる可能性があります。ですから、モジュール型スイッチでは極端に短い間隔でOIDの値を取得することは避けたほうが良いと思います。

ベンダ独自のOID（プライベートMIB）

Cisco IOSが動作している機器のCPU負荷は、

```
cisco1921#show processes cpu
CPU utilization for five seconds: 42%/
/39%; one minute: 21%; five minutes: 9%
```

のように、過去5秒間、1分間、5分間の3通りの時間軸で計測されています。過去5秒間は、2つのパーセンテージが表示されており、この例ではCPU負荷の合計が42%で、そのうちの39%が割り込み処理（ほとんどがパケット転送）に消費されています。この過去5秒間のCPU負荷に対応する最新のOIDは、**リスト3**の3つです。

しかし、デフォルト設定のNet-SNMPでは、これらのOIDに名前でアクセスできません（**図13**）。これは、OIDがベンダ（メーカー）によって独自に割り当てられているためです。ベンダ独自のOIDの定義とそれを含むファイルを「プライベートMIB」と呼び、**図14**のように構成されています。

OIDの5番めと6番めの数字が**4.1**（private.enterprise）である場合、その次の数値がベンダ固有の番号として使われます。Ciscoの番号は**9**です。ほかのベンダにも固有の番号があり、その下の階層は各ベンダが独自に定義しています。

では、CiscoのプライベートMIBをNet-SNMPに追加して、CPU負荷のOIDに名前でアクセスできるようにしてみましょう。まず、CiscoのWebサイト「SNMP Object Navigator[8]」で、OIDを数値で検索します（ユーザ登録しなくても利用できます）。機械翻訳で日本語で読むこともできますが、誤訳があるので必要に応じ英語でアクセスしてください（**図15**）。下のほうにある「MIB」の「CISCO-PROCESS-MIB」をたどっていくと、MIBファイルをダウンロードできます。Net-SNMPのデフォルトでは、「/usr/share/snmp/mibs/」にMIBファイルがありますので、ここに存在しないファイルを選び、「~/.snmp/private-mibs/」にダウンロードしてください。この場合、必要なファイルは次の3つです。

▼リスト3　過去5秒間のCPU負荷に対応するOID

```
CISCO-PROCESS-MIB::cpmCPUMonInterval.1 (.1.3.6.1.4.1.9.9.109.1.1.1.1.9.1)
CISCO-PROCESS-MIB::cpmCPUTotalMonIntervalValue.1 (.1.3.6.1.4.1.9.9.109.1.1.1.1.10.1)
CISCO-PROCESS-MIB::cpmCPUInterruptMonIntervalValue.1 (.1.3.6.1.4.1.9.9.109.1.1.1.1.11.1)
```

▼図13　デフォルト設定では名前からOIDにアクセスできない

```
$ snmpget -v2c -c public 192.168.0.99 CISCO-PROCESS-MIB::cpmCPUTotalMonIntervalValue.1
MIB search path: /usr/share/snmp/mibs
Cannot find module (CISCO-PROCESS-MIB): At line 0 in (none)
CISCO-PROCESS-MIB::cpmCPUTotalMonIntervalValue.1: Unknown Object Identifier
```

・CISCO-SMI.my
・CISCO-TC.my
・CISCO-PROCESS-MIB.my

さらに、「~/.snmp/snmp.conf」を作成し、次の2行を書き込みます。このファイルでは、ホームディレクトリを絶対パスで指定してください。

```
MIBDIRS /usr/share/snmp/mibs:/home/kaoru/.snmp/private-mibs
MIBS all
```

これで、Cisco独自のOIDに、名前でアクセスできるようになります(図16)。

64ビットカウンタ

筆者が初めてSNMPに触れたころは、イーサネットインターフェースの最高速度がやっと1Gbpsになったばかりで、まだ主流は100Mbpsでした。先に紹介したインターフェースの受信バイト数のOIDは32ビット整数だったので、100Mbpsで100%の負荷がかかった場合、約5.7分で桁あふれが発生して0に戻ります。より高速のインターフェースでは、1Gbpsだと約34秒、10Gbpsになると3秒と少しでカウンタが0に戻ってしまうことになります。

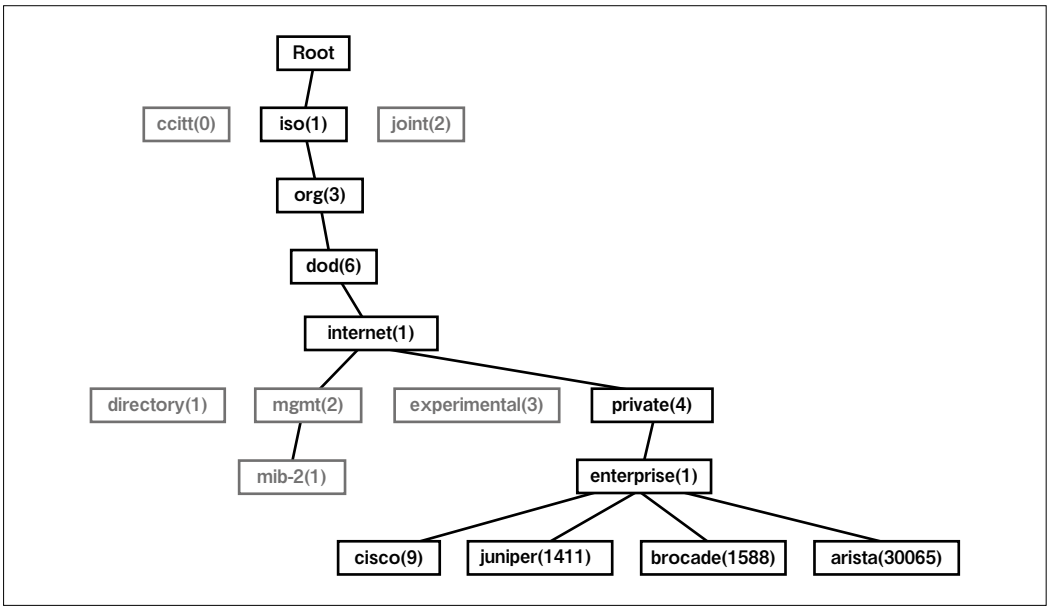
これでは、たとえば1分間隔でOIDの値を取得して値が増加していたとしても桁あふれが起きたか否かがわからず、正確なバイト数を知ることができません。このために、64ビットのOIDが用意されています[9]。

```
$ snmpget -v2c -c public 192.168.0.99
IF-MIB::ifInOctets.2
IF-MIB::ifInOctets.2 = Counter32:
3256237645

$ snmpget -v2c -c public 192.168.0.99
IF-MIB::ifHCInOctets.2
IF-MIB::ifHCInOctets.2 = Counter64:
16141143794
```

64ビットカウンタのOIDの名前に含まれているHCは「High Capacity(大容量)」の略です。32ビットカウンタで表現できる最大値が4,294,967,295($2^{32}-1$)であるのに対して、64ビットカウンタでは18,446,744,073,709,551,

▼図14 プライベートMIB



615($2^{64}-1$)まで数え上げることができます。

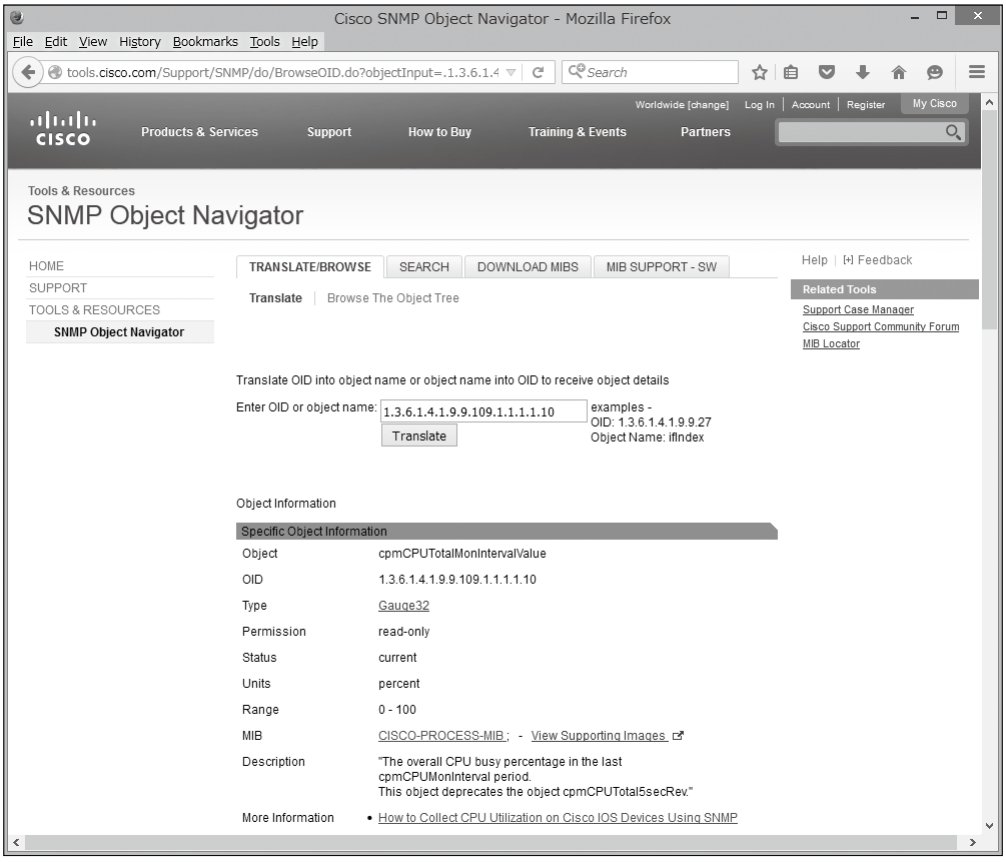
この例で、32ビットカウンタと64ビットカウンタの値を比較すると、前者は3回桁あふれを起こして0に戻っていたことがわかります。64ビットカウンタであれば、現時点で最高速の100Gbpsのインターフェースでも、桁あふれまでに約47年かかります。標準化が始まっている400Gイーサネットでも12年弱ですから、当面は問題にならないと思います。

可視化ツール「Cacti」

SNMPでは、その瞬間のOIDの値はわかりますが、過去にさかのぼることはできません。また、Net-SNMPにはOIDの値を可視化する機能はありません。そんな中「Cacti[注10][10]」は、監視対象の機器のさまざまな情報を蓄積し、

注10)「カクタイ」と発音されることが多いようです。

▼図15　Cisco SNMP Object Navigator



▼図16　Cisco独自のOIDに名前でアクセス

```
$ snmpget -v2c -c public 192.168.0.99 CISCO-PROCESS-MIB::cpmCPUMonInterval.1
CISCO-PROCESS-MIB::cpmCPUMonInterval.1 = Gauge32: 5 seconds
$ snmpget -v2c -c public 192.168.0.99 CISCO-PROCESS-MIB::cpmCPUTotalMonIntervalValue.1
CISCO-PROCESS-MIB::cpmCPUTotalMonIntervalValue.1 = Gauge32: 42 percent
$ snmpget -v2c -c public 192.168.0.99 CISCO-PROCESS-MIB::cpmCPUInterruptMonIntervalValue.1
CISCO-PROCESS-MIB::cpmCPUInterruptMonIntervalValue.1 = Gauge32: 39 percent
```

グラフ化してくれるツールです。SNMPを用いて機器にアクセスでき、かつあまりSNMPを意識しなくても良いように工夫されています。

Cactiは、5分ごとに監視対象の機器にアクセスして、その結果をMySQLやMariaDBのデータベースに蓄積していきます。グラフは、データベースに蓄積されたデータをもとに生成するので、グラフ化の時間軸の範囲と刻み幅を選んで、さかのぼって情報を確認できます。

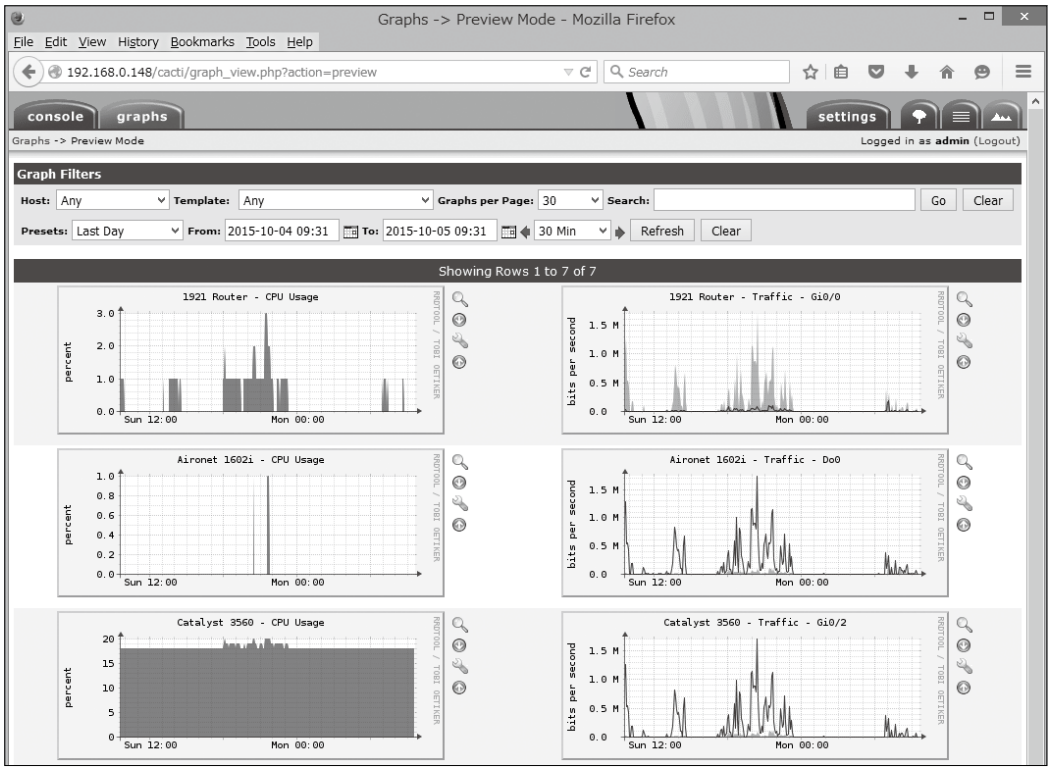
SNMPで取得したデータのグラフ化には、古くから「MRTG[11]」が用いられてきました。しかし、MRTGは古いソフトウェアであり、メンテナンスもあまり頻繁に行われなくなっているため、代替ツールとしてCactiが広く使われています注11。

注11) この稿では紹介しませんが、Cacti以外にも監視ツールはたくさんあります。

図17は、Ciscoルータ、Catalystスイッチ、無線LANアクセスポイント(Aironet)の3台の機器のCPU負荷と、インターフェースのトラフィックをグラフ化した例です。CentOS 7.1-1503にCacti 0.8.8bをインストールし、実際に動作させています[12]。

この6つのグラフを見ると、いくつか興味深いことがわかります。右側の縦に3つ並んだトラフィックの変動は、3台ともほぼ同じです。しかし、CPU負荷がトラフィックに連動しているのは、左上のルータの場合だけです。これは、このルータのみがCPUでパケット転送処理をしており、AironetとCatalystでは、CPUが直接パケット処理に関わっていないためです。実際に、Aironetではグラフの中央付近でCPU負荷が2回ほど高くなっていますが、トラフィックには連動していません。Catalystスイッチではまた挙動が異なり、トラフィッ

▼図17 Cactiによるグラフ化



クに無関係にCPU負荷が20%弱のまま、ほぼ一定になっています注12。

時刻同期について

SNMPバージョン3では、SNMPのパケットをキャプチャして再生する攻撃に対処するために、タイムスタンプを用いています。具体的には、受信したメッセージがリクエストの送信時刻から前後150秒の範囲に収まっているかを確認します[5]。この時刻の取得には、SNMPエージェントが起動してからの経過時間を用いています。ですから、NTPなどによるSNMPエージェントとマネージャの時刻同期は、必須ではありません。しかし、監視の精度を上げるために、できる限りNTPなどを用いて関連する機器すべての時刻を同期することをお勧めします。

注12) 詳細は、Cisco社のバグID「CSCtn42790」を参照してください[13]。

5 SNMPの関連技術

Author 山下 薫(やました かおる) URL http://www.linkedin.com/in/kaoruyamashita

SNMPトラップと似た「syslog」、そしてSNMPそのものを発展させた「NETCONF」および「YANG」を紹介します。

syslog

本特集で紹介したSNMPトラップと同様の役割を持つしくみとして「syslog」があります。syslogは、1台のサーバが複数の機器からのログメッセージを受信し、1つのファイルにまとめて書き込んだり、複数のファイルに分けることができます。1台の機器から複数のサーバへログを送信することも可能で、障害発生時にもできる限りログを残すようにできます。syslogサーバは、ログのファシリティ(種類)とレベル(重要度)に応じて、対応するファイルに受信したメッセージを書き足します。

SNMPトラップは、対象になるイベントでどのOIDの値が変化したかを通知するので、"何が起こったか"を機械的に判定することが容易です。これに対して、syslogはテキストデータでメッセージを取り扱うので、syslogサーバ側で何が起きたかを判別する処理が難しくなります。

syslogサーバとしてもっともよく用いられるのはLinuxだと思いますが、最近はログ管理をjournaldへ移行する方向です。しかし、たとえばCentOS 7では従来のsyslogサーバと互換性を持つrsyslogdが並行して動作しており、当面は外部機器のログをrsyslogdで記録する方法が残るようです。

rsyslogdに、Cisco IOSが動作しているルータからのログを受信させ、ファイルに記録させてみましょう。まず、rsyslogdの設定ファイル「/etc/rsyslog.conf」の次の3行のうち、2行目と3行目の行頭のコメント#を削除します。

```
# Provides UDP syslog reception
$ModLoad imudp
$UDPServerRun 514
```

また、次のようにログのファシリティとログファイルの場所を記述して、rsyslogdを再起動します。

```
local6.* /var/log/gihyo.log
```

一方、IOSに次の設定を追加します。

```
IOSv(config)#logging facility local6
IOSv(config)#logging host 192.168.0.148
↑rsyslogdが動作しているサーバのIPv4アドレス
```

以上で、rsyslog.confで指定したファイルにログが記録されるようになります(**図18**)。このログメッセージは、SNMPトラップの動作例と同じく、インターフェース**GigabitEthernet0/0**がダウンした場合の出力例です。SNMPトラップでは、送信されるのはOIDの形で構造化されたメッセージでしたが、syslogでは普通のテキストデータが送信されます。

NETCONF

(※「NETCONF」と「YANG」の項には、筆者の主観が多く含まれます。ご了承ください)

SNMPでは、先に書いたsnmpsetを用いて機器の設定を変更できます。しかし、SNMPだけを用いた機器の設定は、次のような理由により現実的ではありません。

①機器の設定すべてを、OIDだけで表現することが難しい
②変更内容が実際に受けつけられるか、事前に確認できない
③設定がまったく入っていない状態から開始するためのしくみがない

①がもっとも重要な課題であり、これを克服するべくベンダ、機種に依存しない設定と運用のための「共通言語」または「データモデル」を確立しようという、ネットワーク界の「聖杯」[注13]を求めての試行錯誤が続いてきました。この目標は「オーケストレーション」「プログラマビリティ(プログラム可能)」「SDNコントローラ」などの用語でも表現され、ネットワーク機器の設定を楽にするだけにとどまらず、より高い自由度や柔軟性を持たせよう、という試みが行われています。

一方、前記の課題を克服して、機器の設定を一元化し、SNMPを置き換えることを目標にしているプロトコルが「NETCONF」です。筆者が最初に触れたNETCONFは、SNMPのOIDやMIBの代わりにXMLを用いる「NETCONF/XML」と呼ばれるものでした[14]。

2008年から2010年ごろのお話です。当時まだ発売されたばかりのとあるネットワークOSは、過去のOSのしがらみから自由になるために、SNMPのサポートは必要最小限にとどめ、NETCONF/XMLに全面的に移行するという大胆な方針で市場に投入されました。ところが、XMLによる記述の自由度が高過ぎて、どう書いたら良いのかがわからない状況が続出し、またSNMPを用いる既存のマネージャとの互換性がなさ過ぎたため、ユーザの支持を得ることができませんでした。このため、SNMPのサポートを従来機種と同じレベルまで復活させることになり、NETCONF/XMLはごく一部でだけ使われるにとどまっています。

注13)「見果てぬ夢」でもあります。

▼図18　ログが記録されているか確認

```
$ tail -f /var/log/gihyo.log
Oct 13 20:15:56 192.168.0.99 129: *Oct 13 20:16:26.080: %LINEPROTO-5-UPDOWN: Line ➥
protocol on Interface GigabitEthernet0/0, changed state to down
Oct 13 20:15:56 192.168.0.99 130: *Oct 13 20:16:27.080: %LINK-3-UPDOWN: Interface ➥
GigabitEthernet0/0, changed state to down
```

YANG

実は、もともとNETCONFのプロトコルを策定する際に、データモデルが一番難しいということは早い段階からわかっていました。そこで、XMLに代わるデータモデルである「YANG(Yet Another Next Generation：さらに別の次世代)」が、2010年にRFC 6020として標準化されました。YANGは、「設定(コンフィグ)」に特化したプログラミング言語といった感じです。設定する対象を表現するためのさまざまなデータ型や、プログラミング言語によくある構造体、共用体などが使えるようになっています。また、ルーティングテーブルなどの表形式のデータを扱うためのデータ型と、それに対する操作なども定義されています。ただし、通常のプログラミング言語が持っている条件分岐や繰り返しの機能はほとんどありません。

おもなネットワークベンダのYANGへの取り組みとしては、BrocadeとCiscoがGitHubでサンプルなどを公開しています[15]。ただし、Ciscoはハイエンドルータ用のOSである「IOS XR」向けのサンプルしか、現時点では公開していません。代わりに、Ciscoは買収したTail-f Systemsが開発した「ConfD」の機能制限版である「Basic ConfD[16]」を、無償で配布しています。Basic ConfDはYANGをサポートしており、設定サンプルなども含まれています。

YANGに関する網羅的な情報は、筆者が調べた限りでは日本語で書かれたものを見つけられませんでした。代わりに、英語ではありますが、Tail-f Systemsが公開しているビデオ[17]とプレゼン資料[18]が、現時点ではもっともよくまとまっていると思います。

6 まとめ

Author 山下 薫(やました かおる) URL http://www.linkedin.com/in/kaoruyamashita

あらゆる機器を同じルールで管理できるのがSNMPの強み。本番環境ではどのようにSNMPを活用すればいいのか、そのヒントをお伝えします。

最後に、この記事で解説したSNMPによる監視とは何だったのか、何に留意しなければならないのかをまとめます。「監視」の目的は、大別すると次の2つです。

❶ 定常運用に入った情報システムが正常に動作しているか、確認し続けること
❷ 異常を検出した場合、その原因と正常状態への復旧手段(対策)を知るための手掛かりを見つけること

❶を実施し続けるためには、自動化が必須です。そのために、情報システムを構成するさまざまな機器の状態を知るためのプロトコルが、SNMPでした。しかし、実際にやってみるとわかりますが、監視対象のすべての機器に対して`snmpwalk .`を実行すると、膨大なOIDとその値がリストアップされます。ですので、どのOIDを監視対象にするのかを絞り込むことが重要です。

次に、何をもって「正常」と判断するかの基準を決めます。SNMPリクエストに対して応答があることがまず大前提ですから、監視の対象となる機器へネットワーク経由で到達できることが、正常動作の確認の第一歩です。

SNMPエージェントだけが動作しなくなることは少ないので、SNMPリクエストに対する応答の有無で、この到達性は確認できます。

SNMPとは別に、マネージャから定期的にpingして、到達性と、対象機器が応答できる状態にあるかどうかを確認することもよく行われています。

さらに、監視対象のOID、たとえばCPU負荷、残りメモリ、インターフェースの状態と負荷、パケット送受信数などが、どんな値の範囲になるかを見ます。正常稼働時に、このような値がどんな範囲に収まっているのかが「ベースライン」です。ベースラインは、運用に入る前のテストで見極めておくのが理想ですが、実際は本番運用をしばらく続けている内に分かってくることも多いと思います。ですので、Cactiなどでそれっぽいグラフが並ぶと運用できている気分にはなりますが、ベースラインがわからず、正常時と異常時を区別できないのであれば、グラフは無意味です。

さらに、SNMPトラップをうまく使うことができれば、異常動作の自動検出と通知ができます。ただし、自動的に正確に異常が検出できるのは、たとえば常に必ずリンクアップしていなければならないインターフェースがダウンしたなど、症状が明確なものに限られます。

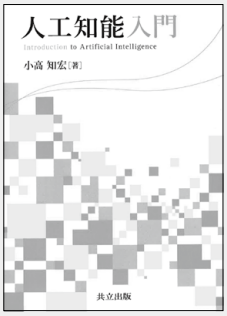
もしSNMPでは障害が検出できず、ユーザの申告などで判明した場合も、該当する時間帯に送信されていたSNMPトラップが原因特定に使えることもあります。

複数の機器から集まってきていたsyslogによるログメッセージと突き合わせることも、原因特定と対策を調べるために役立ちます。過去の障害と同じかどうかは、Cactiなどで蓄積した履歴と比較することで、ある程度判断できます。

以上、SNMPの概要について説明しました。みなさんのネットワークとシステムが、無事稼働し続けますように。SD

○参考資料

[1]『入門SNMP』, Douglas R. Mauro, Kevin J. Schmidt著, 土本 康生監訳, 福田剛士訳, オライリー・ジャパン, ISBN = 978-4-87311-090-4
[2]http://www.ibsjapan.co.jp/tech/details/successstory/building-automation-security/intrusion-detection-with-snmp-and-ip-video-in-factory.html
[3]http://embedded-computing.com/articles/internet-things-requirements-protocols
[4]http://www.lansw-book.net/VIRL/
[5]『実践SNMP教科書』, 山居正幸著, CQ出版社, ISBN = 978-4789818759
[6]http://www.cisco.com/cisco/web/support/JP/100/1008/1008143_snmp_traps-j.html
[7]http://networkengineering.stackexchange.com/questions/14630/snmp-if-mib-stats-export-interval-in-cisco-ios
[8]http://tools.cisco.com/Support/SNMP/do/BrowseOID.do?local=en
[9]http://www.cisco.com/cisco/web/support/JP/100/1006/1006423_faq-snmpcounter-j.html
[10]http://www.cacti.net/what_is_cacti.php
[11]http://oss.oetiker.ch/mrtg
[12]http://www.server-world.info/query?os=CentOS_7&p=cacti&f=1
[13]https://supportforums.cisco.com/ja/document/106446
[14]http://www.janog.gr.jp/meeting/janog22/jointevent/data_3/netconf-01atarashi.pdf
[15]https://github.com/YangModels/yang/tree/master/vendor
[16]https://developer.cisco.com/site/confD/
[17]http://www.tail-f.com/confd-training-videos
[18]http://www.slideshare.net/tailfsystems/presentations



人工知能入門

小高 知宏 著
A5判／192ページ
2,300円＋税
共立出版
ISBN＝978-4-320-12389-2

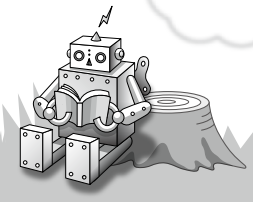
人工知能研究の歴史はアルゴリズムの発見と計算機の処理能力向上の歴史でもある。本書で解説する人工知能研究の産物は、現在「ビッグデータ」や「ディープラーニング」といった言葉で語られる、“膨大なデータを高速に処理し、そこに意味を見いだす技術”の基礎である。著者は福井大学大学院で知能情報学を専攻する教授であり、教科書らしい内容で、即座に実践に役立つ類いの本ではない。しかし、データ分析の基礎となる考え、そして全体像をつかむうえでは、一読しておくとしっかりとした土台が築ける。ちまたで話題の“計算機の能力が人間の知的能力を上回る”時点（シンギュラリティ）については本書でも最後の章で触れているが、それは本書で解説する工学的な観点での人工知能とは別のお話。

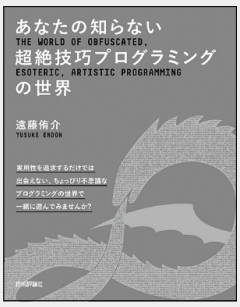
実践JUnit

Jeff Langr、Andy Hunt、Dave Thomas 著、牧野 聡 訳
A5判／272ページ
2,800円＋税
オライリー・ジャパン
ISBN＝978-4-87311-730-0

Javaのテスティングツール「JUnit」を題材に、Javaでのテストコード作成における勘所をまとめている。ユニットテストと聞くと機械的な作業をイメージしてしまうが、本書は独特なフレーズを使って、ユーモラスに説明している。たとえば、良いテストの条件として「FIRST（Fast・Isorated・Repeatable・Self-validating・Timely）」という合言葉を作り、テスト作成時の指針とするよう書いている。また、あとから見返すと内容が理解できないような悪いテストコードの特徴を“テストの臭い”と表現し、それを嗅ぎ分けるためのポイントが解説されていたりもする。本書はEclipse上でJUnitを使うことを想定してはいるが、普遍的な内容が多く、NetBeansやIntelliJ IDEAのユーザにも、別のテストツール「TestNG」のユーザにも有用だ。

SD BOOK REVIEW

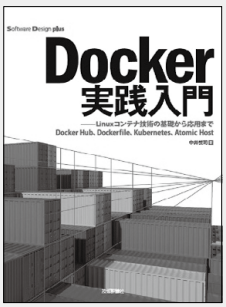




あなたの知らない超絶技巧プログラミングの世界

遠藤 侑介 著
B5変形判／272ページ
2,680円＋税
技術評論社
ISBN＝978-4-7741-7643-7

超絶技巧プログラミングとは「実用的なプログラミング言語を使って全く実用的でないものを作る遊び」と著者が本の中で述べているもの。「実用的なプログラミング言語」とあるように、コードはすべてRubyまたはC言語で書かれているので、手元の環境で実行できる。扱っているプログラムは、著者が得意としている「Quine」形式で作成されたものがもっとも多い。Quineとは、それ自体の文字の並びが絵に見えるソースコードを実行すると、元のコードと同じ文字列が出力されるというもの。本書表紙のウロボロスの絵も、実はQuine形式のプログラムである。Ruby→Scala→Scheme……と、100種類のプログラミング言語を経由したのちに、元のウロボロスの絵を出力するという大傑作なのだ。



Docker 実践入門

中井 悦司 著
B5変形判／200ページ
2,680円＋税
技術評論社
ISBN＝978-4-7741-7654-3

Dockerは、時代にマッチした技術だ。一言でいえばそうなる。ハイパーバイザで過剰なまでの仮想化を行うのではなく、必要な機能を自由に使い、必要がなくなったら捨てるというスマートな計算機資源の管理ができる。これがいい。クラウド環境を構築し、その中でサーバインスタンスを多数立ち上げ、トラフィックの増減に対応しなければならない現代の環境が求めたものなのだ。Immutable Infrastructure（不変なインフラ）という言葉があるが、しばらく時が経てばITエンジニアにとって、自然なものになるだろう。必要な機能をもったサービスを立ち上げて、ニーズがなくなれば捨てる（消去する）というスタイル。これを支える重要な技術の1つがDockerだ。まずは本書で基礎を固めてほしい。

短期集中連載

開発に効く数字の測り方

クラウド時代のWebサービス負荷試験再入門

Author 仲川 樽八（なかがわ たるはち） ㈱ゆめみ Twitter@tarupachi

第1回 クラウド時代における負荷試験とは何か

はじめに

皆さん、はじめまして。仲川樽八です。筆者は2000年頃の携帯向けWebサービスの黎明期に起業したベンチャー企業に新卒として入社、その後15年間Webサービスの開発に携わってきました。入社後しばらくの間は物理サーバを利用して携帯向けの公式サイトの構築、運用業務に携わっていたのですが、ここ数年間はクラウドを利用したシステムの構築、運用を中心とした業務を行っています。

クラウド以前・以後

AWSを始めとしたクラウドの登場はシステム構築の現場を大きく変えました。一般的にはクラウドが与えた恩恵と言えば、中小のサービス事業者や開発ベンダにおいても巨大なリソースを扱えるようにしたことです。しかしクラウドがシステムリソース調達のスピードを劇的に変えたことは、それ以上に大きな恩恵であり、クラウド時代の本質であると、筆者は考えています。

負荷試験の必要性

オンプレミス時代は、常に余剰システムリソースの在庫を抱えている体力のある事業者でなければ、新規リソースの追加は選定から発注、納品まで1ヵ月以上の期間がかかることがよくありました。サービスリリース時点および将来の必要リソースを見積もることはサービスの継続のための必須要件であり、そのための負荷試験は非常に重要視されていました。また、その負荷試験を行うためにも、試験用サーバリソース調達のための期間が別途必要であり、スケジュール的に十分な余裕をとっておく必要がありました。

クラウドの登場はシステム調達の速度を、数ヵ月という単位から数分といった単位に変えました。これにより開発ベンダの必要リソースの見積もり方や負荷試験の位置づけが大きく変わりました。負荷試験をしてあらかじめ必要リソース量を見積もっておかなくても、まず動くものを作ってサービスインしてしまい、実際のユーザのアクセスに合わせて動的にシステムリソースの調達をするといった、まるで後出しジャンケンといってもいい手法が採れるようになったのです。

オンプレミス時代の負荷試験とは？

クラウド以前は、システムの可用性に対する要求も現在ほど高くなかったこともあり、突発的なアクセス増に対して動的にシステム構成を変えないものが主流でした。また、多くの場合において、システムリソースの選定はシステムが組み上がる前に行っておく必要があったため、負荷試験を行う段階では対象のシステムリソース構成はほぼ決定していました。その固定されたシステムについて、利用可能な単位時間あたりのユーザ数の見積もりを出すことが負荷試験の目的でした。

サービスインまでのスケジュールに余裕がある場合ならば、負荷試験の結果をもとにサーバの構成を変更し、あらためて追加リソースの発注をかけることができましたが、システムの構成が決定するまでは試験もできないというジレ

ンマがあるため、商用環境と同じ構成での負荷試験を行うことも難しいことがほとんどでした。

もちろん、同じシステムリソースではより多くのユーザにサービスを提供することが求められますから、負荷試験時には同時にシステムのプロファイリング、ボトルネックの特定を行い、チューニングしていくことはクラウド時代においても変わらない、負荷試験の目的の1つです。

クラウド時代の負荷試験とは?

クラウド時代には、後出しジャンケンのような手法が採れるようになったと説明しました。しかし、その一方でシステムは24時間365日稼働して当たり前になりました。また、高速な通信や処理が可能なスマートフォンの普及などにより、サービスの利用ユーザの規模も、1人あたりのリソース要求量も膨れ上がりました。さらにユーザ数の増加に伴う負荷の増大や、突発的なアクセス増などに対しても“**システムリソースを(動的に)増強することでシステムの応答性能を向上させることができるシステム**”つまり“**スケーラブルなシステム**”を設計、構築することが当然になりました。

しかし、そのシステムが本当に“**スケーラブルなシステム**”であるということは、どうやって担保されるのでしょうか。それは負荷試験を実施しなかった場合には、システムをリリースしたあとに、まさに必要に迫られてシステムリソースを増強しようとした、そのときまでは実際にはわからないのです。そしてわかったときには、すでに遅かったりします。

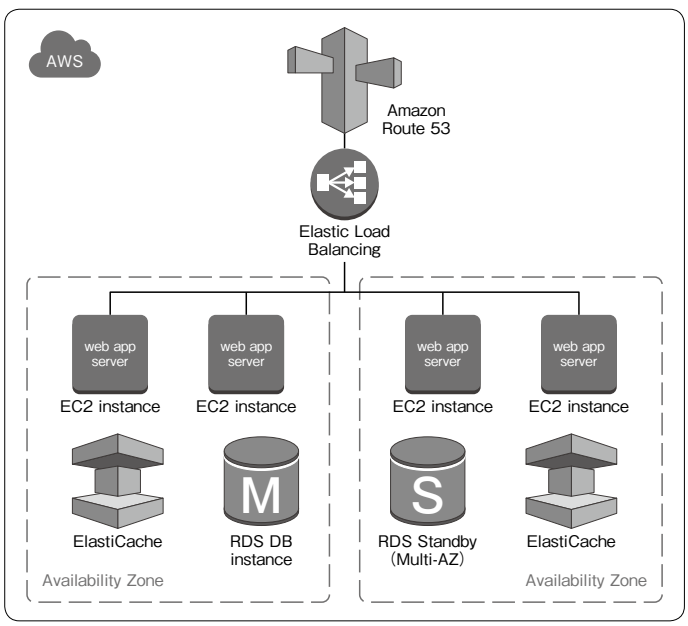
クラウド時代においての負荷試験の主な目的はすでに固定された構成における性能を保証することではなく、構築された対象のシステムがまさに“**スケーラブルなシステム**”であることを確認することなのです。

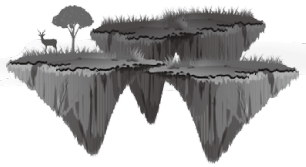
スケーラブルなシステムの設計の復習

サーバ構成を変えるスケールの手法には単一のシステムリソースの処理能力の増減を行う方法とシステムリソースの台数を増減させる方法の2つがあります。言うまでもなく、これらの手法はクラウド登場以前からありますが、クラウドによって非常に利用しやすくなり、積極的に設計に取り入れられるようになりました。また、上記の2つの方法とは別にクラウド業者がサービスとしてスケール対応であることを保証するフルマネージドサービスと呼ばれるストレージやロードバランサ、KVS(Key-Value Store)などのサービスもあります。

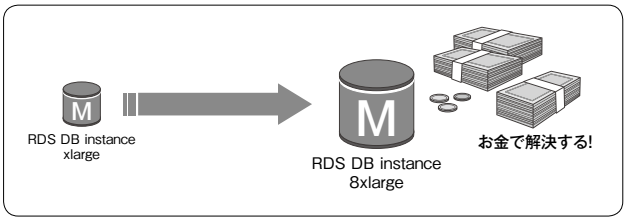
スケーラブルなシステムを構築する際には要件や規模に応じてこれらを組み合わせて考えます(**図1**)。Webサーバ部分、データベース部分、キャッシュ部分とサブシステムに分解していき、すべての部分で冗長性およびスケーラビリティーが確保された設計となるようにします。

▼図1 冗長化対応、スケール対応のシステムの例





▼図2　若干のメンテナンス時間さえとれればスケールアップできる



冗長性の確保について

システム上の、どのサーバノードが停止してもサービスを継続して受けることができるようにします。それには、複数台のサーバを利用することで単一障害点を作らないとともに、データも常に最新のデータの二重化を行う必要があります。

たとえばRDSというDBサービスではMySQLやPostgreSQLなどのDBを利用できますが、その際にMulti-AZというオプションがあり、それを利用することで最新のデータを複数のAvailability Zone(データセンターにあたると考えてください)間にまたがった別々のサーバ上で保持しますので、どちらかのサーバまたはAvailability Zoneに障害が発生してもそのままデータの欠損を生むことなくサービスを継続できます。オンプレミス時代には難しかった、データセンターをまたいだ形でのシステムの冗長化を簡単に行うことができるというのもクラウド(AWS)の大きなメリットの1つです。

スケールアップ・スケールダウン

システム負荷が上がったときに、機器そのものをより高機能なものに変更することをスケールアップ(**図2**)、逆にシステムリソース使用状況に余裕があった場合に適切なリソースになるまでより低機能な機器に変更することをスケールダウンと言います。

スケールアップのメリット

アプリケーションやミドルウェアが未対応でも簡単に該当リソースの応答性能の増強ができます。

スケールアップのデメリット

デメリットを次に挙げます。

・スケールアップさせた機器の応答性能が、その機器を利用するコストに比例しては上がらないことが少なくない(とくにすでにかなり良い機器を利用していた場合には、2倍の値段の機器を利用してもスループットがほとんど変わらないことがある)
・提供されている機器の最大サイズがシステムの上限となってしまい、それ以上の増強が不可能
・中間的なサイズのリソースが提供されていないことが多い(2倍→4倍→8倍→16倍といった感じでリソースも増強されるがコストも上がる)
・システムのリプレイス時に一定時間のダウンタイムが発生することが多い

スケールアウト・スケールイン

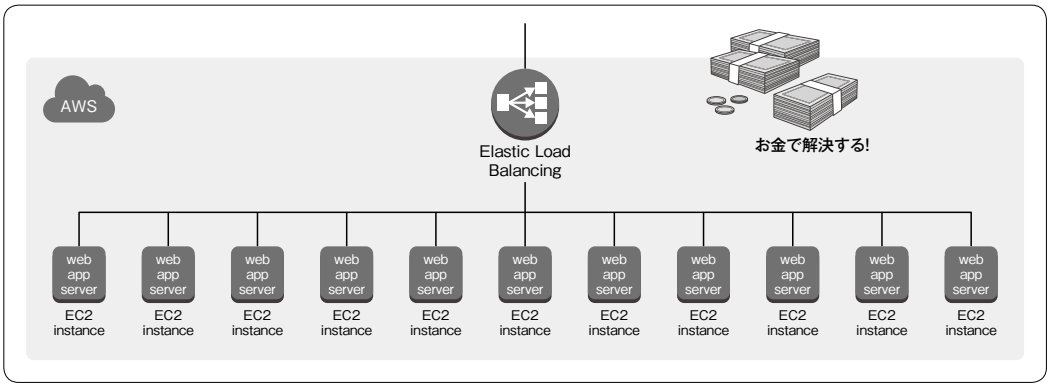
システム負荷が上がったときに、機器の台数を追加することでスループットの改善を図るのがスケールアウト(**図3**)、逆にシステムリソース使用状況に余裕があった場合に適切なリソースになるまで機器の台数を減らすことがスケールインです。

スケールアウトのメリット

メリットを次に挙げます。

・スループットの増減は機器の台数に比例して追従しやすい(スケールアップの場合と比較してコストパフォーマンスに優れることが多い)
・システムの応答性能の上限がサーバ1台の最大能力に制限されないため、スケールアップを行う場合より性能上限を引き出しやすい
・機器を複数台利用することで性能向上を狙うと同時にシステムの冗長化を図ることができる
・オートスケールといった手法を取ることにより、負荷に応じて自動的に機器の台数を増減させることも可能

▼図3　高負荷のときはスケールアウトする



・マスタ、スレイブ構成のレプリケーションを組んでいる際に、スレイブ側の負荷分散は比較的容易に可能

スケールアウトのデメリット

デメリットを次に挙げます。

・追加された機器を適切に利用するために、アプリケーションもしくはミドルウエアに対応が必要となることが多い
・コネクションの上限の問題やネットワーク的な問題が発生することがある
・マスタ、スレイブ構成のレプリケーションを組んでも更新側の負荷が大きなコンテンツには対応できない
・分散構成に対応していないDBへの更新負荷を分散させるためにはデータ垂直分割[注1]、水平分割[注2]などを考える必要があり、分割されたデータ間ではJOINができないなどの制約が発生するため、設計、実装レベルでシステムの複雑度が上がる

クラウド提供業者によるフルマネージドサービスの利用

ファイルサーバとして、AWSのS3を利用する、ロードバランサとしてELB(Elastic Load Balancing)を利用するなど、利用状況に応じて自動でスケールするフルマネージド[注3]なサービスもあります。また、システムのスループットの予約を動的に変更することができるDynamoDBのようなサービスもあります。デメリットとしては提供されている機能が要件に合わなければ利用できないことが挙げられますが、逆に要件を整理してでも積極的に利用することで設計や運用をかなり楽にできます。

Webシステム設計事例と負荷試験の事例

携帯向けのWebコンテンツの開発に筆者が長くかかわってきたために事例は偏ってしまうのですが、2000年以降のシステム開発の設計事例と、負荷試験、その結果の事例を紹介します。

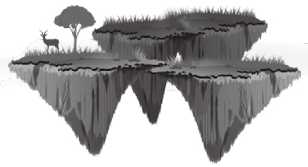
[オンプレ時代]フィーチャフォン黎明期の携帯向け公式サイトシステムの事例

Webブラウザが搭載された携帯電話が相次いで発表されたころ、それをビジネスチャンスととらえたサービス事業者が次々と参入しました。筆者が所属する会社もまさにその時期に起業したベンチャーで、当時は筆者を含めた全員がまだ学生でした。関係者の誰もが携帯向けコンテンツの開発経験がないなか、最初に立ち上げた携帯向け公式サイトの概要が**図4**です。現

注1) 扱うデータの種類によりデータの格納先DBを変える手法、扱うDBごとに異なるテーブルを管理する形になります。

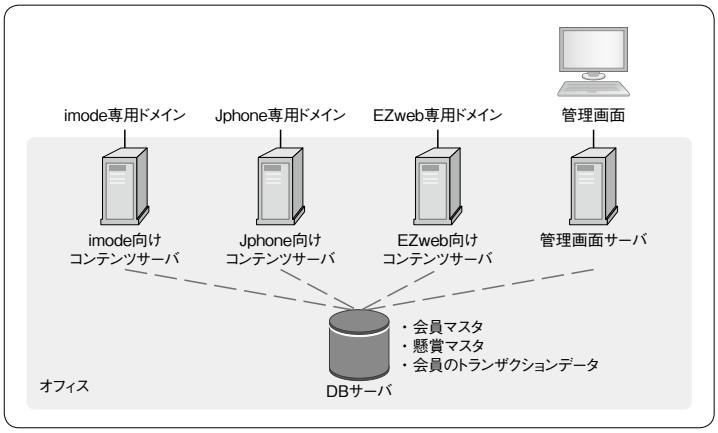
注2) すべてのDBで同じテーブルのスキーマを利用しますが、たとえばユーザの区分ごとにDBを分けるといった管理をします。こちら、分割された個々のDBをシャードDBと呼ぶこともあります。

注3) フルマネージドサービスを利用していても、実際にその能力をすべて引き出すことができるかは、構築したシステム依存であることに注意ください。



在のシステムと比較すると障害にかなり弱い設計です。すべてのサーバが単一障害点で、どのサーバが障害を起こしたとしても代替サーバのセットアップが完了するまでサービスも停止し、また完全なデータ保全も保証されていませんでした。

▼図4　システム概要（[オンプレ時代]フィーチャフォン黎明期）



その当時の負荷試験について

数週間ほどオフィスに寝泊まりしながらコンテンツの開発を行い、リリース直前までデバッグ作業を行った結果、まともな負荷試験をかけることなくサービスインしてしまいました。今考えるとそれがシステムの冗長度の低さと合わさって、その後の長い苦しい運用の始まりだったのですが、その結果としてサービスイン後は実際のユーザアクセスを利用して商用環境に高負荷をかける“**実践的な負荷試験**”を毎晩かけることができました。しかしながらその“**実践的な負荷試験**”（もちろん皮肉です）は次のような代償を伴っていました。

- 夜間に接続不能という状況が続き、ユーザの利便性を大きく損ない続けた
- ほかのメディアと連動した短期的にユーザが集中する系統の施策や、会員数を増やすための有効な施策が打てない
- 負荷が高いときにはサーバにログインできなくなるので、その分析もできない
- 単体部分の試験ができないので特定のリソースの逼迫（ひっぱく）の原因がどこにあるのかの分析が困難
- 対策コードのリリースやより高価なシステムリソースへのリプレイスを行っても、応答性能は変わらなかったケースがある

反省事項

このシステムにかかわったことで、本来であれば負荷試験の重要性や冗長化されたシステムのありがたさを学び取ることができたはずなのですが、残念ながら当時は経験も浅く、こういったシステムが当たり前だと思ってしまっていました。

それらの重要性がわかったのは、そのあとのシステム開発において実際に負荷試験を行ったり冗長化されたシステムを構築したりして、その恩恵がおもに開発者に対して向けられることを実感してからとなります。

[オンプレ時代]フィーチャフォン最盛期の携帯向け公式サイトシステムの事例

前節のシステム開発から数年、数々の公式サイト開発の事例を積み重ねたあとに、より大規模なエンタープライズ向けのシステムを構築することとなりました。この案件では何よりも24時間365日無停止であることが重要視され、そのために当時のシステムとしてはかなり潤沢なサーバリソースを利用しました。

サイト内容

携帯3キャリア（imode、Softbank、EZweb）向けのクーポンサイトです。ユーザの居住エリア情報や性別、店舗での利用状況といった情報を基にユーザごとにクーポンを発行し、またその利用を追跡、プロファイリング可能とします。システム概要は**図5**のとおりです。

[オンプレ時代]のシステム構成と負荷対策

システム構成を次にまとめます。全体的に冗

長化され障害にも強くなり、また参照系の負荷分散やアプリケーションサーバの負荷分散がされていましたが、MasterDBサーバは冗長化されていないという状態でした。サービスイン当初の負荷対策は次のようになります。

・Apache＋PHP＋MySQL構成
（PHPアクセラレータの導入）
・Webサーバをロードバランサの下に配置することで冗長性およびスケール性能を確保
・データの冗長化はMySQLのレプリケーション機能と定期バックアップを組み合わせた
・アプリケーションサーバは各キャリア向け共通で、DB操作が必要なページ以外はすべて静的ファイルとして構築
→静的ファイルの一部のデータや絵文字の変換をApacheモジュールで処理することでPHPを利用せずに高速に応答させる
→ユーザの認証機構もApacheモジュールで担保するため、会員制コンテンツ部分も高速に応答
・DBの参照負荷はMySQLのレプリケーション機構を利用して対策

負荷試験について

この案件ではJmeterを利用した負荷試験を複数回ほど実施できました。しかし、オンプレミスのため、システムリソースの選定、発注から稼働開始まで2ヵ月間程度は余裕を見ておく必要がありました。そのため負荷試験の結果を見てからのサーバ調達はできず、すでに準備された環境に対する負荷試験となってしまいました。それでもサービスインまでには負荷試験の結果を受けて各種の対策を実施でき、サービスイン後しばらくの間順調にサービスを行い続けました。しかしながらそのあと、会員数が数百万人規模に順調に増加していったことと、クーポンアプリケーションのリリースとともにクーポンをすべてユーザごとに配布するという方式に変えたことでMasterDBサーバの能力が不足し、次の負荷対策が追加必要となりました。

リリース後の負荷対策

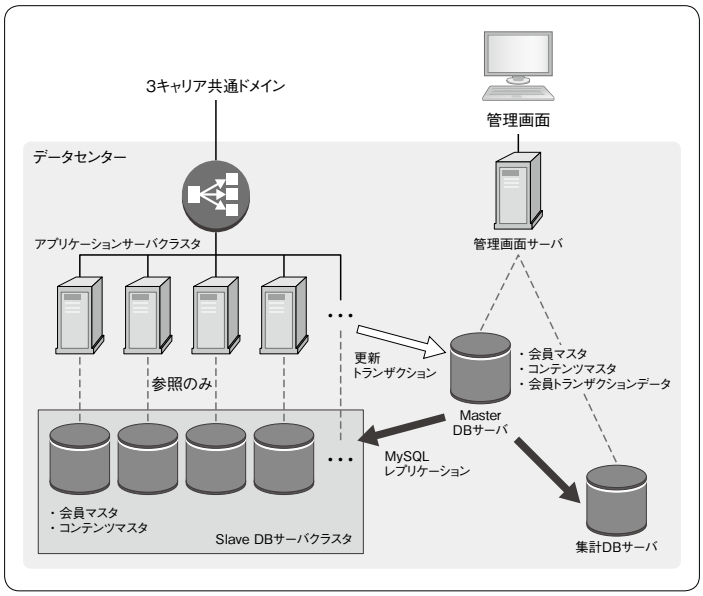
次に記載する負荷試験を行うため、専用の環境を構築しましたが、実際のサイズのものを準備することが難しく、商用環境の数分の1のサイズのシステムでの負荷試験となりました。

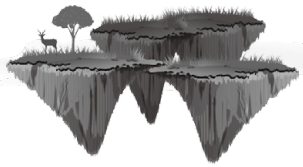
・DBサーバ機器の入れ替え
・会員トランザクションデータを書き込むDBを会員ごとに分割するシャードDB対応
・DB/SQLチューニング
→Indexの見直し、追加
→ロックを発生させないように更新処理はすべて主キーを指定したクエリになるように書き換え

反省事項

ユーザ数の増加、要件の追加によりバックエンドシステムのリプレイスを必要としましたが、

▼図5［オンプレ時代］フィーチャフォン最盛期のシステム概要（データセンターにサーバを設置）





負荷試験を行うスケジュールが確保できたことおよびシステムの冗長構成を取ることができたことで、全体的には非常に安定した運用を行えるようになりました。

また、事前に負荷試験をすることで各種の不具合を未然に防げました。一方でサービスインしたあとにデータ移行を伴う大きな設計変更を加えることは非常にコストが高くつくので、可能ならば最初の設計時点ですべての場所でスケール性を考慮しておくべきであることを痛感しました。

［クラウド時代］スマホアプリケーションAPIサーバ

ユーザの利用デバイスの中心がスマートフォンに移行し始めたころ、スマートフォン向けソーシャルゲームアプリケーションのバックエンド側APIの構築をしました。ユーザごとのトランザクション量が非常に多いため負荷対策も重要で、またサーバ上で管理させるデータや処理とアプリケーション上で管理させるデータや処理との同期方法などに工夫が必要なサービスです。筆者の所属するチームでは、このころからシステム構築の中心をAWS利用に移行し始めており、この案件も同様です。システム概要は**図6**のとおりです。

システム構成と負荷対策

エンドユーザからアクセスされるすべての部分[注4]が冗長構成となっており、単一のノードの障害やゾーン障害が発生してもサービスに影響を与えることなくそのまま利用可能であり、またデータの欠損も生じさせない構成です。

これらは以前の案件ではエンタープライズ向けの一部のお客様でしか取れなかったような贅沢な構成でした。しかしクラウドにより、こういった構成は案件の規模にかかわらず導入可能となり、また、逆にそうすることが当たり前の時代となってきました。そのときの負荷対策は次のようになります。

- Apache＋PHP＋MySQL構成（PHPアクセラレータの導入）
- 横断検索の必要なデータはリードレプリカを利用する（落ちていたときはマスタ参照）
- 横断検索の必要ないユーザのトランザクションデータはシャードDBに格納（あらかじめ60シャードの論理DBを作成しておくことで、DBノード追加時のユーザデータの移行を行いやすくした[注5]）
- 複数のAPIを同一のHTTPリクエストとしてコールできるフレームワークを自社開発
  →HTTPリクエスト数を絞ることが可能
  →複数のシャードDBおよびキャッシュデータをまたいだ更新であっても、同時にコールされたリクエストはすべて同一のトランザクションとして扱う。この機構によりアプリケーション側からトランザクション範囲を適切に制御可能
- Webサーバをロードバランサの下に配置することで冗長性およびスケール性能を確保
- データの冗長化はRDSのMultiAZオプションを利用する
- キャッシュ機構を利用することでDBの参照負荷を減らす

負荷試験について

Jmeterを利用した負荷試験を複数回実施し、次のような障害を発見、解消しました。重要なことは、これらの障害は負荷試験を実施しなかった場合にはサービスイン後に顧客に見える障害として発生してしまう問題だったことです。

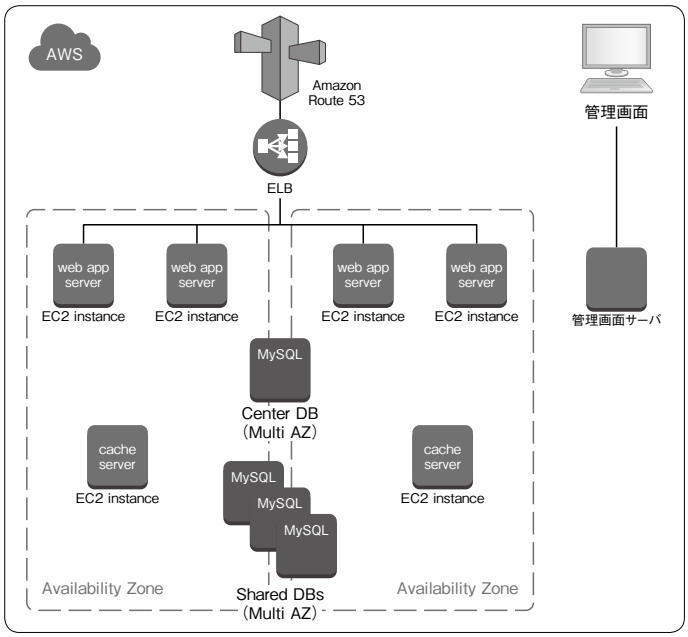
- 最初に選定したPHPアクセラレータはApacheを巻き込んで停止してしまうものであったためにアクセラレータを変更

注4） 逆にユーザから見えない部分の管理画面サーバなど一部のサーバは冗長化されていませんが、障害時であってもあらかじめ取得しておいたサーバのイメージを起動するだけですぐに復旧ができます。

注5） シャード用のDBノードとして2台準備した場合は30論理DBずつ利用、3台準備した場合は20論理DBずつ利用、と4台、5台、6台と台数を少しずつ追加しやすい数字のため。また、論理DBをあらかじめ分けておくと、テーブルの中から一部のユーザのデータを切りだすといった作業がなく、簡単にDBのコピーを行うことができます。

・高負荷時にRDS for MySQLへの接続が失敗することがある
→接続リトライ機構を組込み解決
・高負荷時にMemcachedへの接続が失敗する[注6]
・Webサーバを追加していっても応答性能が上がらない
→MySQL接続で可能な部分は永続的接続を利用する(接続上限数を超えない場合)
→Memcached接続で永続的接続を利用する

▼図6 [クラウド時代]スマホアプリケーションAPIサーバシステム概要(AWS上ですべて構築)



反省事項

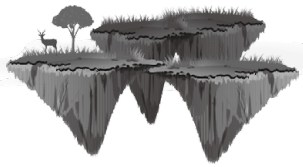
負荷試験およびプロファイリングは、まさに試行錯誤の中でボトルネックと対策を見つけていく作業ではありますが、この頃はまだ負荷試験をかけるにあたって効率的なツールや、手順などを意識していなかったことで正解にたどり着くまでに遠回りをしてしまうことや、間違った解答を導き出してしまうことが今よりも多くありました。

負荷試験を行わなかったことによる失敗事例(と思われるもの)

クラウドにより、スケーラブルなシステムの構築が当たり前となりました。ただ、それでもなお負荷対策不足に起因する事故は至るところで発生しており、それらは決して他人事ではありません。これらの事故は想定よりもはるかに多くのアクセスがあったために発生した不可避の不幸な事故の可能性も考えられますが、多くの場合は適切な負荷試験を行い、その結果の数字を尊重することにより回避できた可能性は高いと考えています。最近の事例でも次のようなことを聞いたことがありますし、中には皆さんも思い当たる事例もあるかと思います。

・開発環境では動作していたのに商用環境ではまともに動かない
・負荷対策はスケールアップで対応させる予定だったが、最大クラスのインスタンスを使用しても負荷に耐えられなかった
・スケール対応のフルマネージドシステムとつなぎ込んでいるはずなのにまったく性能が向上しない
・Webサーバをいくら追加してもまったくスケールしない
・ECサイトリニューアル以降、数週間サイトにアクセスできない状態が続く
・大々的な告知をした期間限定のキャンペーンサイトが、期間が終了するまで落ち続ける

注6) 接続失敗とそのときに発生するオートフェールオーバー機構の副作用でMemcachedが更新前の古いデータを取得してしまう現象が発生してしまい、結果としてユーザデータの不整合を発生させます。これを回避するためにElastiCacheではなく、KyotoTycoonの利用に変更しました。こちらの問題に関しては[Qiita:本当は怖いMemcached<http://qiita.com/taruhachi/items/a844bf373623991873ff>]に詳細を記載しています。



負荷試験が難しいケースとその対策

もちろん負荷試験が難しいケースもいろいろ存在しますが、最低限の目標として構築したシステムがスケール性能を保有していることを担保するところまでは必ず試験が必要です。たとえば次のような場合においては試験が難しいと感じることもあるかもしれません。

[ケース1]負荷試験環境が存在しない場合

クラウド時代のメリットを最大限に利用して、実際の環境を模した専用の負荷試験環境をぜひ構築してください。たとえ巨大なシステムであっても、時間貸しのシステムで構築して数時間だけしか使わないという利用方法をした場合には驚くほど安く利用できるのがクラウドです。たとえばAWSで考えたときに、月額で100万円の利用料のかかるシステムであっても、1時間あたりで計算すると1,400円以下となります。料金シミュレータで計算するとこれはc4.largeインスタンスを100台同時に稼働させることができる値段[注7]となります。

また、どうしても同等レベルの環境を構築することが難しかった場合には各サーバのインスタンスタイプをスケールダウンしたり、スケールインした環境を利用して全体のスループットを推測するという方法もありますが、こちらはスケーラビリティを確認するのには良いのですが、実際のシステムのスループット上限値を推定することが非常に難しくなりますので注意してください。また、この手法を採るときには次のことが重要になります。

- 冗長性に関しては商用環境とまったく同じ構成にする[注8]
- 性能の低いインスタンスで得られた結果と性能の高いインスタンスで得られた結果はインスタンスタイプの値段やスペックシートに正比例はしない。よって必ず両方で試験をして、それぞれにおけるスループットの比率を計測しておく
- 投入したダミーデータの量が負荷試験結果に重大な影響を及ぼすシステムと結果には影響をほとんど与えないシステムがあるので、そこは見極める必要がある

[ケース2]外部システムとの結合がある場合

外部システムとの結合があり、かつ結合先の環境に余計な負荷をかけられないケースなどは負荷試験をためらいがちになります。しかしながら、一般的にシステムの処理時間の多くは外部システムへの通信部分が占めることが多く、その部分がシステム全体のスループットやスケール性能を左右することが多いですので、逆にこういった場合こそ負荷試験を諦めてしまってはいけません。このような場合では次のいずれか方法をとります。

①該当の外部システムのダミー応答をするスタブサーバを構築する
②プログラム内部で外部システム連携部分のスタブを準備する

これらの方法のうち、強くお勧めするのは①の方法です。外部システム自体がどんなに高速に応答を返しても、実際に連携をさせた場合には思ったような性能が出ないということはよくあります。相手は静的なファイルの設置でもかまいませんので、ぜひ通信のシミュレーションを含めた試験を行ってください[注9]。

注7) 2015年11月現在：AWSの料金はドル建てですので、円・ドル相場によって変動します。また、AWSの料金改定による値下げは随時発表されていますのでこちらは参考までにしてください。

注8) 商用環境でMultiAZオプションを利用するのなら負荷試験環境でも利用するなど。

注9) どちらの場合であっても、実際の応答が得られる時間を想定したSleep()処理を入れておいてください。Sleep()処理は処理サーバのCPU負荷を上げずに全体の処理時間を伸ばすという意味で、システムに対して実際の外部システム連携をした場合と似た負荷をかけられます。

[ケース3]システムの性能が高すぎるため負荷をかけきれない場合

負荷試験をかけるためのツールはたくさんありますが、大きく次の軸で考えらえます。

①お手軽に使えるツール（ApacheBench、Locust等）
②複雑な攻撃シナリオの記載、実施が可能なツール（JMeter、Locust等）
③高負荷をかけることが得意なツール（tsung、Jmeter等）

この中で③の「とにかく高負荷をかけることが得意なツール」を利用する必要があります。また、負荷試験環境が存在しない場合と同じく商用環境を小さくしたセットに対して負荷試験をかける方法もありますが、この場合の注意点は前述のとおりです。

[ケース4]セッションIDやパスワード等、リクエストごとに異なるパラメータを付ける必要がある場合

次の2つの方法があります。これはどちらの方法でもかまいませんがスタブを利用する場合はその部分の負荷のシミュレートが難しくなる点に注意が必要です。

- 負荷試験をされる側のプログラムを改変して、パラメータを利用する部分をスタブにしたり、プログラムの先頭でパラメータを生成して埋め込んだりする
- 前項における「複雑な攻撃シナリオの記載、実施が可能なツール」を利用する

[ケース5]予算・スケジュールがない

負荷試験は必ず見積もりに含めてください。仮にお客様が実施するので納品物としては含めないという契約の場合であっても、スケーラブルなシステムであることを担保することは必要です。その担保のためにはやはり内部での工数は必要です。工数として省いてはいけません。

負荷試験がほぼ不要なケース

筆者は、次の場合においては負荷試験はほぼ不要なのでケースに合わせて省略してかまわないものと考えています。逆に言えば、次のケースに当てはまらない場合は、原則的に負荷試験を実施すべきと考えてください。

- ELBやCloudFrontなどのベンダがスケール性能を保証しているシステムに対する負荷試験
- すでにスケール性能が担保されている既存システムに対する少改修[注10]
- トラフィックが“絶対に”増えないケースやスケール性能を担保しないシステム

負荷試験のアンチパターン集

最後に負荷試験におけるアンチパターンを紹介します。それぞれの理由は次回以降に説明していきます。

- 対象のシステムに合わない負荷試験ツールを利用する
- SSLページの負荷試験をする
- 負荷試験環境を全部入りサーバで構築する
- 負荷試験のスケジュールをシステムリリース直前に持ってくる
- 最初からシステム全体の負荷試験をする

次回予告

次回は各種負荷試験ツールの紹介およびツールを使用した負荷試験の基本的な考え方の紹介をします。SD

注10）ただし、データリソースへのアクセス、外部APIの利用など、外部システムとの結合個所が1ヵ所でも増える場合はそこがシステム全体の応答性能を大きく左右してしまう可能性がありますので、負荷試験を行うべきです。

手がかりを探せ！

SMB実装をめぐる冒険 第2回

File System for Windowsの作り方

探す、調べる、ソフトを作る喜び

こんにちは。よういちろうです。「Windows共有フォルダをChromeOSのファイルアプリにマウントする」ことができるChromeアプリを開発してリリースしました。これを開発するためには、SMB（Server Message Block）と呼ばれるプロトコルを理解し、SMBプロトコルを話すクライアントコードをJavaScriptで書くことが必要でした。これは「File System for Windows」という名前でChromeウェブストアにて無料で公開していますので、Chromebookを持っている方はぜひ使ってみてください。今回は、前編に引き続き探偵風に開発過程を紹介します。

Author 田中 洋一郎（たなか よういちろう） Blog https://www.eisbahn.jp/ Twitter @yoichiro

第4部 身分を証明しろ

ユーザ認証のしくみを探る

クライアントとサーバの間で使用するダイアレクトが決定したあとに行うべきこと――それは「ユーザ認証」です。つまり、サーバに対して身分を証明することで、そのユーザの権限が決まります。このユーザ認証は、コンピュータの普及に伴ってセキュリティリスクが高まるたびに、仕様が次々と追加されてきた歴史があります。SMBプロトコルが扱っている「ファイル共有」という機能の性質上、ユーザ詐称ができてしまえば他人のファイルを盗むことができてしまうため、被害がとても大きくなります。昔はパスワード文字列をそのままLANに垂れ流していることも普通に多かったと思いますが、今日では許されないことです。

ユーザ認証を安全に行うこと、これがSMBプロトコルの最初の関門です。

さまざまな認証方式

SMBプロトコルでのユーザ認証方式は、いくつか存在します。適用される認証方式は、次の条件から決定されます。

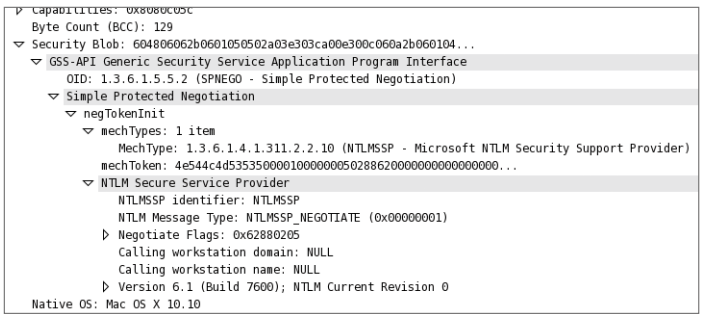
・capabilities値に含まれるextended_security値（拡張セキュリティ）が1である→Kerberos認証方式、NTLMSSP認証方式など
・上記のextended_security値が0である→LM/NTLM/NTLMv2 Challenge Response認証方式

拡張セキュリティには、ほかにもGSS-API、LDAP、MS-RPC、SPNEGOなども仕様上は利用できます。これらの「サブプロトコル」の存在が、SMBプロトコルをさらに難解にしている原因です。もちろん「SMBプロトコルは柔軟性がある」と言うこともできるのですが、そもそもパケットキャプチャして解析している身としては、サブプロトコルの存在はうれしくありません。

今回僕が実装した認証方式は、NTLMSSP認証方式と、LM/NTLM/NTLMv2 Challenge/Response方式の2つです。この2つの方式それぞれを理解し、正しく使い分けができるようになるまでには、多くの時間がかかりました。それはなぜか？――今振り返ると、その要因は2つありました。

1つは、そもそもNTLMSSP認証方式はSMBプロトコルとは独立した仕様であるため、SMBプロコトルに関する仕様をいくら読んでも説明されていなかった、ということです。参考にしていた@ITの記事やImplementing CIFS本にも、NTLMSSP認証方式に関する説明は載っていません。

▼図1　NTLMSSP認証方式が適用されたパケットのキャプチャ



そしてもう1つは、拡張セキュリティと認証方式の関係がまったくわからなかったことです。Implementing CIFS本には、LM/NTLM/NTLMv2 Challenge/Response認証方式のことが説明されていました。その章を読んで当然「よし、これでユーザ認証できる！」と思い込み、意気込んでLM/NTLM/NTLMv2 Challenge/Response認証方式を実装して、いざサーバにリクエストを送ってみても、STATUS_INVALID_PARAMETER 0xC000000D が返ってきて途方に暮れました。結論としては、拡張セキュリティが適用されたサーバに対して、拡張セキュリティではない場合の認証方式でリクエストし続けていたからエラーになっていた、ということだったのですが、これに気がつくまでに1週間程度かかったのを記憶しています。

NTLMSSP認証方式

サーバがずっとエラーを返し続けて悩んでいたとき、ふと初心に返って、OS XがSMB1/CIFSでどのようにユーザ認証要求を送っているのか、Wiresharkを使ってパケットキャプチャをしてみようと思い立ちました。すると、**図1**のような結果が表示されたのです。そこには僕が知らない謎のSecurity Blobバイト列が指定されていました。

これはImplementing CIFS本には載ってない……いや、載ってました。拡張セキュリティ、つまりextended_security値が1の場合には、あるアルゴリズムで計算されたパスワードのダイジェスト値を使う方法(LM/NTLM/NTLMv2 Challenge/Response認証方式)ではなく、Kerberos認証方式、NTLMSSP認証方式などが採用される、ということが書かれていました。これでやっと拡張セキュリティと認証方式の関係に気がついたのですが、肝心のNTLMSSP認証方式に関する詳しい手順は説明されていません。困りました。

インターネットで探せばあるはずだ、と思い、再度文献を探したところ、幸いにも見つけることができました。

[The NTLM Authentication Protocol and Security Support Provider][注1]
http://davenport.sourceforge.net/ntlm.html

この内容を読んでいくと、実はNTLMSSP認証方式では3つのType Messageが存在していて、クライアントとサーバ間のやりとりは1回ではなく2回必要になる、ということがわかりました。そして、ユーザ名とパスワードをいかにして計算していくかについても、例も含めてとても丁寧に解説されていました。Wiresharkで先ほどパケットキャプチャした際の内容ともどうやら一致している模様です。光が見えてきました。

3つのType Messageは、それぞれ異なる内容を持ちます。具体的には次のような感じです。

- Type 1 Message：認証要求を示す各種フラグ値などを持つ。これは、クライアントからサーバに送られる
- Type 2 Message：チャレンジなどを持つ。これは、サーバからクライアントに送られる

注1）(日本語訳)http://www.monyo.com/technical/samba/translation/ntlm.html

・Type 3 Message：ユーザ名、ドメイン名、チャレンジに対するレスポンスなどを持つ。これは、クライアントからサーバに送られる

Type 1 Message

1つずつ見ていきましょう。まずはType 1 Messageです。これは、ユーザ認証を行うためにクライアントがサーバに提示する条件を伝える役目を持っています。つまり、重要なのはフラグ値です。Type 1 Messageの構造を**リスト1**に示します。

flag値は4バイトもあります。つまり、指定可能なフラグは32個と多いのですが、基本的には次を指定しておけば良いでしょう。

・NEGOTIATE_UNICODE (0x00000001)
・REQUEST_TARGET (0x00000004)
・NEGOTIATE_NTLM (0x00000200)
・NEGOTIATE_OEM_DOMAIN_SUPPLIED (0x00001000)
・NEGOTIATE_OEM_WORKSTATION_SUPPLIED (0x00002000)
・NEGOTIATE_128 (0x20000000)

supplied_domainは、"?"を指定します。また、supplied_workstationには、クライアントのプログラム名を指定しておけば良いでしょう（今回作ったChromeOS向けアプリケーションでは"FSP_CIFS"と指定しています）。この2つは任意ですので、指定しなくても大丈夫です（OS Xでは指定されていません）。

▼リスト2　セキュリティバッファ

```
SecurityBuffer {
  USHORT length;          // セキュリティバッファ値のバイト数
  USHORT allocated_space;// 割り当て済みのバイト数
  UINT offset;            // Type 1 Messageの先頭からの位置
}
```

▼リスト3　Type 1 Messageの例

```
4e54 4c4d 5353 5001 0000 0005 3200 2001
0001 001b 0000 0008 0008 0020 0000 003f
4653 505f 4349 4653
```

▼リスト1　Type 1 Messageの構造

```
Type1Message {
  UCHAR[8] signature;          // "NTLMSSP"
  UINT message_type;           // 0x00000001
  UINT flag;
  UCHAR[8] supplied_domain_security_buffer;
  UCHAR[8] supplied_workstation_security_buffer;
  ANY supplied_domain;         // ASCII
  ANY supplied_workstation;    // ASCII
}
```

ここでセキュリティバッファについて説明しておきましょう。名前からは連想しにくいのですが、比較的長いバイト列を指定するために、実際のバイト列と「そのバイト列がどこにどんな長さで存在するか」を分けて指定する方式のことを指しています。具体的には、**リスト2**のように3つの値で構成されます。

flagとして上記の値を、SuppliedDomainは"?"、SuppliedWorkstationに"FSP_CIFS"を指定した場合のType 1 Messageのバイト列は、**リスト3**のようになります。

Type 2 Message

Type 1 Messageをクライアントがサーバに送信したあと、クライアントはサーバからType 2 Messageを受け取ることになります。このType 2 Messageは、クライアントから提示された条件をサーバが受け付けられるかどうかの返答、およびチャレンジとなるバイト列が含まれています。つまり、クライアントがType 3 Messageを作るために必要となる情報が、Type 2 Messageによってサーバから提供されるという流れです。Type 2 Messageの構造は**リスト4**になります。

サーバが返してきたflag値に応じて、クライアントはユーザのパスワードから作成するレスポンスの作り方を変化させていく必要があります。

Type 3 Message

Type 2 Messageを受け取ったクライアントは、サーバが提示してきたflag値に応じて、ユーザのパスワードをさまざまな計算を行うことで変化させてレスポンスを生成し、それをType 3 Messageに乗せて再度サーバに送る、ということを行います。その「さまざまな計算」にも複数の種類があります。**表1**は、執筆時点で存在する計算方法の一覧です。

ドメイン環境(NTドメインや、ActiveDirectoryによるドメイン)でKerberos認証方式が適用されている場合を除いて、現在はLMv2 ResponseおよびNTLMv2 Responseが広く利用されています。ここではこの2つのレスポンスの計算方法を紹介します。

LMv2 Responseの計算手順は**表2**になります。

また、NTLMv2 Responseの計算手順は**表3**になります。LMv2 Responseのときよりも、使用する値が増えています。

LMv2 ResponseおよびNTLMv2 Responseを持つType 3 Messageをサーバに送ることで、サーバはユーザ認証処理を行います。もちろん、Type 3 MessageにはLMv2 ResponseおよびNTLMv2 Responseを指定する個所があります。**リスト5**は、Type 3 Messageの構造です。

Type 1 Messageのsupplied_domainには"?"を指定しましたが、Type 3 Messageのdomainには、もしドメインコントローラによって管理されているユーザで認証したい場合には、正しいドメイン名を指定することが必要です。そうではなく、ワークグループレベルでのユーザ認証の場合には、このdomainについても"?"で大丈夫です。

▼リスト4　Type 2 Messageの構造

```
Type2Message {
  UCHAR[8] signature;          // "NTLMSSP"
  UINT message_type;           // 0x00000002
  UCHAR[8] target_name_security_buffer;
  UINT flag;
  UCHAR[8] challenge;          // レスポンス生成に利用
  UCHAR[8] context;
  UCHAR[8] target_information_security_buffer;
  ANY target_name;             // Unicode、レスポンス生成に利用
  ANY target_information;      // レスポンス生成に利用
}
```

▼表1　レスポンスの計算方法の一覧

名前	説明
LM Response	最初に策定された方式
NTLM Response	Windows 2000/XPなどのNT系OSで採用された形式
NTLMv2 Response	Windows NT SP4から導入された形式
LMv2 Response	LM Responseのセキュリティ強化形式
NTLM2 Session Response	NTLM2 Session Securityがネゴシエートされた際に利用する形式
Anonymous Response	匿名として認証する形式

ユーザ認証のためのメッセージ(拡張セキュリティの場合)

NTLMSSP認証方式の内容を把握できましたが、クライアントおよびサーバはType 1〜3 Messageを送受信できなければなりません。Type 1〜3 MessageはSMBプロトコルのメッセージではあり

▼表2　LMv2 Responseの計算手順

No.	説明	例
1	UnicodeベースのパスワードからMD4を使ってNTLMハッシュを計算する	"SecREt01"→0xcd06ca7c7e10c99b1d33b7485a2ed808
2	Unicodeベースのユーザ名およびtarget_name文字列をそれぞれ大文字変換した結果を結合し、NTLMハッシュ値を鍵としてHMAC-MD5によるNTLMv2ハッシュを計算する	("USER"+"SECRET01")×HMAC-MD5→0x04b8e0ba74289cc540826bab1dee63ae
3	ランダムな8バイトのclient_nonceを生成する	0xffffff0011223344
4	challenge値とclient_nonceを結合し、NTLMv2ハッシュを鍵としてHMAC-MD5によるダイジェスト値を計算する	(0x04b8e0ba74289cc540826bab1dee63aeffffff0011223344)×HMAC-MD5→0xd6e6152ea25d03b7c6ba6629c2d6aaf0
5	4のダイジェスト値とclient_nonceを結合して、LMv2 Responseとする	0xd6e6152ea25d03b7c6ba6629c2d6aaf0ffffff0011223344

▼表3　NTLMv2 Responseの計算手順

No.	説明	例
1	Unicodeベースのパスワードから MD4を使って NTLMハッシュを計算する	"SecREt01"→0xcd06ca7c7e10c99b1d33b7485a2ed808
2	Unicodeベースのユーザ名および target_name文字列をそれぞれ大文字変換した結果を結合し、NTLMハッシュ値を鍵として HMAC-MD5による NTLMv2ハッシュを計算する	("USER"+"SECRET01") × HMAC-MD5→0x04b8e0ba74289cc540826bab1dee63ae
3	現在日時を表現するタイムスタンプ値を生成する	2003/06/17-06:00:00→0x0090d336b734c301
4	ランダムな8バイトの client_nonceを生成する	0xffffff0011223344
5	署名、0、タイムスタンプ値、client_nonce、0、target_information、0を結合した blobを生成する	0x01010000 00000000 0090d336b734c301 ffffff0011223344 00000000 02000c0044004f004d00410049004e0001000c0053004500520056004500520004001400640006f006d00610069006e002e0063006f006d0003002200730065007200760065007200650072002e0064006f006d00610069006e002e0063006f006d0000000000 00000000
6	challengeと blobを結合する	0x0123456789abcdef+blob→0x0123456789abcdef010100000000000090d336b734c301ffffff0011223344000000000002000c0044004f004d00410049004e0001000c0053004500520056004500520004001400640006f006d00610069006e002e0063006f006d0003002200730065007200760065007200650072002e0064006f006d00610069006e002e0063006f006d000000000000000000
7	NTLMv2ハッシュを鍵として、5の結果の HMAC-MD5ダイジェスト値を計算する	0xcbabbca713eb795d04c97abc01ee4983
8	6の結果と blobを結合したバイト列を NTLMv2 Responseとする	0xcbabbca713eb795d04c97abc01ee49830x010100000000000090d336b734c301ffffff0011223344000000000002000c0044004f004d00410049004e0001000c0053004500520056004500520004001400640006f006d00610069006e002e0063006f006d0003002200730065007200760065007200650072002e0064006f006d00610069006e002e0063006f006d000000000000000000

▼リスト5　Type 3 Messageの構造

```
Type3Message {
  UCHAR[8] signature;                  // "NTLMSSP"
  UINT message_type;                   // 0x00000003
  UCHAR[8] lm_lmv2_response_security_buffer;
  UCHAR[8] ntlm_ntlmv2_response_security_buffer;
  UCHAR[8] domain_security_buffer;
  UCHAR[8] user_name_security_buffer;
  UCHAR[8] workstation_name_security_buffer;
  UCHAR[8] session_key_security_buffer;
  UINT flag;
  ANY lm_lmv2_response;                // LMv2 Responseを指定
  ANY ntlm_ntlmv2_response;            // NTLMv2 Responseを指定
  ANY domain;                          // ワークグループであれば"?"
  ANY user_name;
  ANY workstation_name;
  ANY session_key;
}
```

Column　「SMBタイムスタンプの計算方法」

SMBプロトコルでのタイムスタンプ値は、少し特殊な計算方法です。次の手順で計算します。

① 現在日時のUNIX時間(1970/01/01 00:00:00からの経過秒数)を得る
② 1601/01/01 00:00:00からの経過秒数とするために、①に116444 73600を加算する
③ ②の結果について、10000をかける
④ ③の結果について、リトルエンディアンの64ビット値として表現する

ませんが、SMB1/CIFSプロトコルではユーザ認証のためのコマンドが定義されています。それが、SMB_COM_SESSION_SETUP_ANDX 0x73です。このメッセージにType 1〜3 Messageのバイト列を格納して、クライアントとサーバ間でユーザ認証のためのやりとりを行います。

SMB_COM_SESSION_SETUP_ANDXメッセージの構造は、拡張セキュリティが適用されるかどうかによって変わってきます。**リスト6**は、拡張セキュリティが適用されていた場合のSMB_COM_SESSION_SETUP_ANDXリクエストの構造です。

ここで重要な値はsecurity_blobです。このsecurity_blob値として、Type 1〜3 Messageのバイト列を指定します。ほかの値は、ネゴシエートした際の

結果から指定すると良いでしょう。capabilitiesは、再度サーバに期待する動作を指定するフラグ値ですが、試行錯誤の結果、次の指定でうまくいきました。

・CAP_STATUS32 (0x00000040)
・CAP_NT_SMBS (0x00000010)
・CAP_UNICODE (0x00000004)

native_osにはクライアントが動作するOSの名前を、native_lan_manにはクライアント(アプリケーション)の名称をそれぞれ指定すると良いでしょう。今回僕が作ったChromeOS向けのアプリケーションでは、それぞれ“ChromeOS”、“File System for CIFS”と指定しました。

NTLMSSP認証方式では3つのメッセージをやりとりする必要があることを説明しました。つまり、SMB_COM_SESSION_SETUP_ANDXメッセージも1回のユーザ認証で「2往復」することになります。具体的には、**図2**のような流れになります。

Type 1 Messageをサーバに送信したあと、サーバがエラーコード(MORE_PROCESSING_REQUIRED)を返してくるのがポイントです。エラーとはいえ、もちろん継続できます。SMB_COM_SESSION_SETUP_ANDXレスポンスの構造は**リスト7**のようになります。

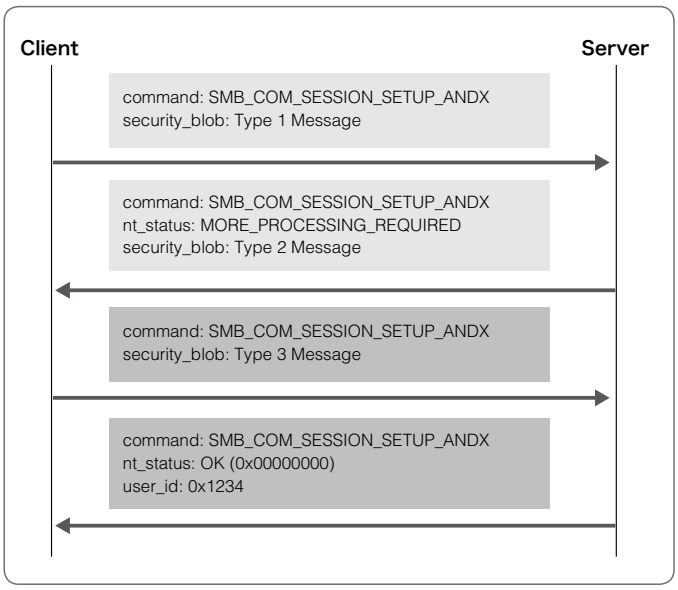
クライアントはType 2 Messageから各種レスポンスを計算し、それらを持つType 3 Messageをサーバに送ってユーザ認証を継続します。もし認証処理が成功すれば、サーバからの最後のメッセージのヘッダには、user_id値がセットされます。

LM Response、NTLM Responseを使う場合も、Type 3 Messageにそれらを含めて上記の

▼リスト6　拡張セキュリティの場合のSMB_COM_SESSION_SETUP_ANDXリクエストの構造

```
SMB_PARAMETER {
  UCHAR word_count;           // 12
  words {
    struct {
      UCHAR command;          // 0xff
      UCHAR reserved;         // 0x00
      USHORT offset;          // 0x0000
    } ANDX;
    USHORT max_buffer_size;
    USHORT max_mpx_count;
    USHORT vc_number;
    UINT session_key;
    USHORT security_blob_length;
    UINT reserved;            // 0x00000000
    UINT capabilities;        // サーバに期待する動作のフラグ値
  }
}
SMB_DATA {
  USHORT byte_count;
  bytes {
    ANY security_blob;        // Type 1,3 Messageが入る場所
    ANY native_os;            // Null-terminated String
    ANY native_lan_man;       // Null-terminated String
  }
}
```

▼図2　拡張セキュリティ適用時のメッセージの送受信手順



▼リスト7　SMB_COM_SESSION_SETUP_ANDXレスポンスの構造

```
SMB_PARAMETER {
  UCHAR word_count;
  words {
    USHORT security_blob_length;
    UINT reserved;              // 0x00000000
    UINT capabilities;          // サーバに期待する動作のフラグ値
  }
}
SMB_DATA {
  USHORT byte_count;
  bytes {
    ANY security_blob;          // Type 2 Messageが入る場所
    UCHAR[0 or 1] padding;      // 2バイト境界のためのパディング
    ANY native_os;              // Null-terminated String
    ANY native_lan_man;         // Null-terminated String
  }
}
```

手順を行えば、ユーザ認証ができるはずです。しかし、これらはすでにセキュリティ強度が非常に弱いことが知られているため、サーバによっては受け付けてくれない可能性があります。

ユーザ認証のためのメッセージ（拡張セキュリティではない場合）

拡張セキュリティをサーバが要求しなかった場合は、Type 1〜3 Messageは使用しません。そして、SMB_COM_SESSION_SETUP_ANDXリクエストの内容も若干変わります。変更個所は次になります。

- SMB_PARAMETER.word_count値を12から13に変更
- SMB_PARAMETER.security_blob_lengthを次の2つに変更
  USHORT case_insensitive_password_length;
  USHORT case_sensitive_password_length;
- SMB_DATA.security_blobを次の4つに変更
  ANY case_insensitive_password;
  ANY case_sensitive_password;
  ANY account_name;
  ANY primary_domain;

拡張セキュリティを適用しない場合は、LM Responseをcase_insensitive_passwordに、NTLM Responseをcase_sensitive_passwordにそれぞれセットします。先ほどLMv2 Response、NTLMv2 Responseを生成した際にはType 2 Messageの内容が計算に必要でしたが、LM ResponseおよびNTLM Responseを計算する際には、ネゴシエートした際にサーバから渡されたencryption_keyを使用します。

拡張セキュリティではない場合は、**図3**のように、SMB_COM_SESSION_SETUP_ANDXメッセージを1往復するだけでユーザ認証が完了します。

LM ResponseおよびNTLM Responseの計算方法は、先ほど紹介した"The NTLM Authentication Protocol and Security Support Provider"の記載内容を参照してください。最初に少し触れましたが、Implementing CIFS本に書かれている手順は、この拡張セキュリティを適用しない場合の手順でした。この手順を懸

Column　「ANDXとは?」

クライアントとサーバ間で行われる通信の回数が少なければ少ないほど、行われる処理全体のパフォーマンスはよくなります。また、1回の通信で複数の処理をクライアントから要求できれば、サーバは要求された複数の処理を一貫性を持って行うことができる可能性が高まります。SMB1/CIFSプロトコルには、末尾に"_ANDX"とついているコマンドが多くありますが、これらは「1回のリクエストで複数のコマンドを送信できる能力を持ったコマンド」です。

しかし、実際にはWindows系OSにおいてこのANDX系コマンドが使われていることはほとんどないらしいです。コマンドを連鎖させるためにSMB_PARAMETERの最初に「次のコマンドは○○ですよ」と指定できるのですが、本記事ではすべて0xff(次のコマンドはない)を指定しています。

Column 「やっかいなPadding」

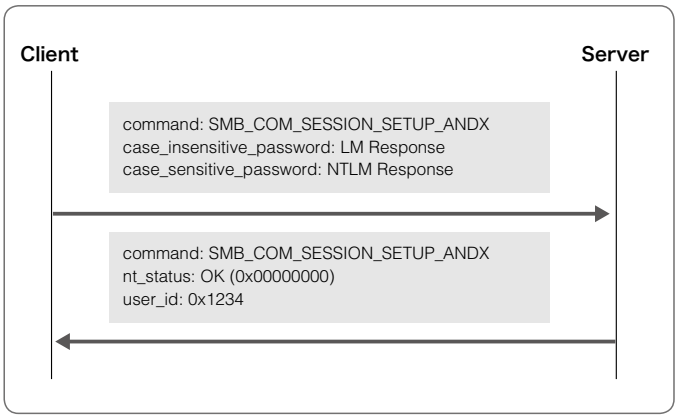
SMB1/CIFSプロトコルは、DOS時代から存在している非常に古い仕様です。これは僕の勝手な想像ですが、UNICODEサポートは後から追加されたものではないかと思っています。そのためなのか、SMBメッセージ内では「UNICODE文字列の開始は、ヘッダの先頭を起点として必ず16ビット境界であることを保証しなければならない」というルールがあります。つまり仮に"hoge"というUNICODE文字列(6800 6f00 6700 6500 0000)があったとすると、

・ff53 4d42……0102 0368 006f 0067 0065 0000 00 →16ビット境界から始まっていないのでNG
・ff53 4d42……0102 0300 6800 6f00 6700 6500 0000 →先頭に0x00を追加して16ビット境界から開始するように補正しているのでOK

というようにPadding値を追加するかどうかクライアントは判断しないといけません。これは、SMBプロトコルを難解にしている要因の1つと言えるでしょう。

命に拡張セキュリティを要求しているサーバに送っていたわけです。うまくいかなかったわけですね。

▼図3 拡張セキュリティ適用時のメッセージの送受信手順



ヘッダにuser_id値を指定する

以上の手順でユーザ認証ができるようになりました。これでuser_id値を得ることができましたので、敵の懐に飛び込んでいくことが可能になります。SMBプロトコルの大きな山を1つ超えたことになります。

これ以降、クライアントがサーバに送信するすべてのメッセージのヘッダに対してuser_id値を指定することで、ユーザ認証が完了していることをサーバに伝えます。サーバの動作は、user_id値から導き出される認証済みユーザの権限の範囲内で行われます。

次号はさらに真相に迫る!

ユーザ認証ができてしまえば、サーバの中に入っていろいろとファイルを入手したり書き込んだりできそうですが、その前にまだやるべきことがあります。それは「共有リソース一覧を入手」し、利用したい「共有リソースのTree IDを入手」する、という2つの手順が待ち構えています。OS XやWindowsからファイルサーバにアクセスした際に、最初に「どの共有リソースに接続しますか?」という問い合わせが表示されると思います。また、もしURIでファイルサーバにアクセスする場合は、"smb://server/share1"というように共有リソース名をパスとして記載するはずです。この共有リソースは1つのサーバに複数存在することが一般的ですので、クライアントは共有リソースの一覧を入手して、ユーザに選択してもらい、選択された共有リソースに接続してTree IDを得る、ということをします。

敵のアジトに入れたからといって、簡単に相手の資産に触れられるわけではないのです。その前に、各資産を管理している番人にお願いして、関所を越えるためのIDを発行してもらわなければならないのです。これは、ユーザ認証の次の壁となります。(以下次号に続く!)SD

一歩進んだ使い方のためのイロハ

Vimの細道

第3回

VimでJavaを使う（補助プラグイン編）

mattn
twitter:@mattn_jp

前回紹介したEclimは、高機能な反面、起動が遅く、自由度が低いという難点がありました。今回は、Vim上でのスピーディなJava開発を、vim-javacomplete2、sonictemplate-vim、java_checkstyle、google-java-formatという4つの補助プラグインを使って実現する方法を紹介します。

“VimでJava”への熾くなき挑戦

前回はVimからEclipseと同等の機能が扱えるEclimについて紹介しました。EclimはバックエンドでEclipseの機能を使っているだけあって、高機能かつ統合的な操作をVimに提供してくれます。あれはあれで非常に便利で、どっしりと構えた開発を行う場合にはかなり有用です。ただし、やはりEclimはEclipse上で行う開発手法をVimから間接的に実行するためのプラグインです。Eclipseでサポートされていない機能はもちろん使えませんし、Eclipseの決めごとに従い続けなければなりません。たとえば、プロジェクトは必ずworkspaceというEclipse上のルールに従って管理する必要があります。Mavenプロジェクトであれば一度Eclipseで取り込んでおいてからEclimで開き直す必要もあります。さらにはEclipseではMavenの扱いに癖があり、場合によってはファセットが正しく機能しない場合もあります。

ちょっとしたJavaのコードを書きたいがためだけに、Eclimdという重たいバッチプログラムを起動するのは億劫ですよね。たとえば、Java 8に導入されたStream APIの挙動を今すぐ試したい場合を考えてみましょう。Eclimの長い起動を待ち、ようやく起動が完了したEclimで新規にプロジェクトを作ってVimから接続し、ようやくコーディングを開始するというのは非常に大きな手間ですし、頭に浮かんだアイデアを逃してしまうこともあります。

Javaでの開発にはIDE（統合開発環境）を使うのが一般的と言われています。「Vimのようなテキストエディタでは補完も効かないし、Eclipseのようにimport文の簡単挿入も行えない」――みなさんそんなふうにお考えかもしれません。

しかしながら、Vimは進化し続けるテキストエディタです。“ないものは作る”のエンジニア魂により、これまで何度かVimでJava開発をする試みが行われてきました。

Java補完にvim-javacomplete2を使う

Vim上のJava補完機能、それは長い長い黒歴史

過去に幾度か、Javaの補完機能をVim上で実現するプラグインが現れました。しかし、そのほとんどはお世辞にも使えるというレベルには達しておらず、言ってしまえば“オモチャ”でしかありませんでした。単純な補完であれば実現できるのですが、Javaの構文の複雑さゆえ、コンテキストを意識した補完となるとEclipseをバックエンドに持つEclimを使うほかありませ

んでした。また、javacompleteという入力補完プラグインがあり、一部のユーザの間で使われていましたが、リフレクション[注1]のみでクラス解析を行っていたために補完候補を出すにはいくぶん貧弱で、コアなユーザからは使われていませんでした。そして、JavaにGenericsやLambdaが追加されてからは、もはや使いものにならなくなってしまいました。

vim-javacomplete2の誕生

「やはりVimには荷が重過ぎるのか」――多くのVimユーザがそんなふうに思っていた中、2015年5月にvim-javacomplete2[注2]が現れます（**図1**）。vim-javacomplete2は**表1**の機能を持ち合わせています。

vim-javacomplete2はjavacompleteをベースに開発されています。javaparserというライブラリを使うことで、リフレクションによる解析だけでなくパーサを用いたソースコードの解析も行っています。ですのでソースコードが多少不完全であっても、極力パースが継続されるように作られています。

バックエンドとしてJavaで書かれたサーバを起動していますが、最小限の機能しか持っていないため、サーバが起動してから補完が実行可能になるまで数秒しかかかりません。筆者の環境（Intel Core i5、メモリ8GB）であれば3秒程度です。一度起動すれば、Vimを終了するまで常駐します。使用しているパッケージの量にもよりますが、起動直後であれば10MB程度のメモリ使用量です。

筆者がこのプロジェクトを初めて見たときは、まだ粗削りで補完も完全ではなく、エラーもよく発生しました。しかしながらjavaparserを使うというデザインに惹かれ、いくらかコントリビュートもさせていただきました。

▼図1　vim-javacomplete2の入力補完

▼表1　vim-javacomplete2の機能

クラスのフィールドやメソッド、コンストラクタの入力補完
クラス自身やサブパッケージの入力補完
メソッドの引数情報、オーバーロードメソッドの入力補完
入力途中の単語を入力補完
JSPファイル内でsessionやrequestを認識した入力補完
Genericsを使ったソースコードの入力補完
Lambdaを使ったソースコードの入力補完
アノテーションの入力補完
ネストされたクラスの入力補完
使用しているクラスからimport文を生成
未使用のimport文を削除
pom.xmlを使ったクラスパスの追加
Eclipseの.classpathファイルからのクラスパスの追加

vim-javacomplete2を導入する

vim-javacomplete2はEclimのようにEcilpseのworkspaceに依存しません。pom.xmlがあれば、Mavenを使ってクラスパスを自動で検出し、依存ライブラリを認識します。入力補完を行うのであれば**リスト1**の設定をvimrcに追加するだけです。

プラグインを導入すると、初回の使用時にサーバを自動でビルドします。Mavenのコマンドmvnへパスを通しておいてください。ほかの面倒な設定は必要ありません。サードパーティのパッ

▼リスト1　入力補完を有効にする

```
augroup MyJavaFileType
    au!
    autocmd FileType java setlocal omnifunc=javacomplete#Complete
augroup END
```

注1）プログラムの実行中にプログラム自身の構造を読み取って解析すること。
注2）URL https://github.com/artur-shaik/vim-javacomplete2

▼リスト2　Javaのコードサンプル

```
package main;
import java.util.Date;

public class Foo {
    public static void main(String[] args) {
        System.out.println(new Date().);
    }
}
```

▼リスト3　生成されるテンプレート

```
/**
* Foo
*/
public class Foo {
    public static void main(String[] args) {
        //カーソルの位置
    }
}
```

ケージを使わないのであれば、プロジェクトに関する設定すらも必要ありません。たとえば、Java 8のStream APIを試したいだけならば、Vimで**リスト2**のJavaファイルを開いて.(ドット)の位置で<C-x><C-o>とタイプすればDateのメンバが表示されるようになります。

コマンド説明

現状vim-javacomplete2が提供する主要なコマンドは次の3つです。

- JCimportAdd
  カーソル下のクラス名から自動でimport文を生成
- JCimportsAddMissing
  未検出なimport文を生成
- JCimportsRemoveUnused
  未使用なimport文を削除

vim-javacomplete2が提供する補完機能とこれらのコマンドがあれば、Javaの編集作業はかなり捗ります。まだ現状リファクタリング機能がありませんが、GitHubリポジトリのissueにはすでに登録されています。いずれ実装されることになると思います。

次に、Javaの開発をさらに便利にするプラグインを見ていきましょう。

ひな形のコード生成にsonictemplate-vimを使う

Javaを書いていて頻繁に直面するのが、書き出しのコードやgetter/setterなどの決まりきったコードです。クラス名だけ決めれば自動でクラスのソースコードを生成してほしいですよね。sonictemplate-vim[注3]というプラグインを使うことで、こういった書き出しのソースコードや、getter/setterなどの部分的なソースコードを生成できます。

Java、C、C++、Golang、Python、Perl、Rubyなど多くのプログラミング言語に対応し、ある程度コンテキストに従ったテンプレートやスニペットを展開できます。

たとえば、Foo.javaというファイルを新規で開き、次のコマンドを実行します(Templateは Temくらいまでタイプして、Tabで補完可能)。

```
:Template main
```

すると、**リスト3**のソースコードが生成されます。ふとおもしろいソースコードが思い浮かんで、頭から消えないうちに書き出したいというときには最適です。

次に、このソースコードにgetter/setterを足したいと思います。**リスト4**のカーソル位置で:Templateまでをタイプし、コマンド引数をTabで補完すると、getterとgetter-setterという候補が現れます。getter-setterを選択するとname(名前)とtype(型)を聞かれるのでそれぞれfoo、Stringと入力します。すると**リスト5**のコードが生成されます。単純な操作で、ここまでのソースコードができあがるのです。あとはmain部分の処理を実装し、vim-quickrunで実行すれば、スクリプト言語をサクッと実行しているような使い心地になります。

注3) URL https://github.com/mattn/sonictemplate-vim

▼リスト4　宣言文を加えて、カーソルを移動

```
/**
* Foo
*/
public class Foo {
    private String foo;
    //カーソルの位置
    public static void main(String[] args) {
    }
}
```

sonictemplate-vimは各言語ごとに、baseと呼ばれるテンプレートとsnipと呼ばれるスニペットの集まりで構成されており、自分専用のテンプレートを簡単に作成できるようになっています。たとえば**リスト5**で使用したgetter-setterのテンプレートは**リスト6**のようになっています。

_input_でnameとtypeを入力させ、その値nameおよびtypeを使って、getterおよびsetterのメソッド名や引数を作成します。if_pythonやif_rubyなどを使用していないので、Vim scriptだけしか動かない環境でも間違いなく動作します。

ソースコードの検証にjava_checkstyleを使う

VimでJavaの開発ができるようになったのは良いのですが、自由になり過ぎてどんどん汚いソースコードを書いてしまいがちです。ここはしっかり文法チェックやコード規約のチェックをしたいところです。Javaでコード規約のチェックを行うにはcheckstyle[注4]を使います。これをVimから扱うプラグインに、**java_checkstyle.vim**[注5]があります。

しかし現在、checkstyleの出力結果が変わったためか正しく動作しておらず、かつモダンな作りになっていなかったので筆者が作りなおしたもの[注6]を使ったほうが良いです。

設定する変数名はjava_checkstyle.vimと同じになっています。Checkstyleコマンドが用意さ

▼リスト5　できあがったテンプレート

```
/**
* Foo
*/
public class Foo {
    private String foo;
    /**
    * get foo
    * @return foo
    */
    public String getFoo() {
        return foo;
    }

    /**
    * set foo
    * @param foo
    */
    public void setFoo(String foo) {
        this.foo = foo;
    }

    public static void main(String[] args) {

    }
}
```

▼リスト6　getter-setterのテンプレート

```
/**
* get {{_input_:name}}
* @return {{_var_:name}}
*/
public {{_input_:type}} get{{_expr_:substitute('{{_var_:name}}', '¥w+', '¥u¥0', '')}}() {
    return {{_var_:name}};
}

/**
* set {{_var_:name}}
* @param {{_var_:name}}
*/
public void set{{_expr_:substitute('{{_var_:name}}', '¥w+', '¥u¥0', '')}}({{_var_:type}} {{_var_:name}}) {
    this.{{_var_:name}} = {{_var_:name}};
}
```

れているので:Checkstyleと実行するか、前述の補完設定と合わせて**リスト7**のようにファイル書き込み時にCheckstyleコマンドを実行するようにしておくと良いでしょう。

このプラグインを使うには、SourceForgeの該当ページ[注7]からcheckstyle-6.11-all.jarをダ

注4）URL http://checkstyle.sourceforge.net
注5）URL https://github.com/vim-scripts/java_checkstyle.vim
注6）URL https://github.com/mattn/vim-java_checkstyle

注7）URL http://sourceforge.net/projects/checkstyle/files/checkstyle

▼リスト7　ファイル書き込み時にCheckstyleコマンドを実行する設定

```
augroup MyJavaFileType
    au!
    autocmd FileType java setlocal ➥
omnifunc=javacomplete#Complete
    autocmd BufWritePost *.java
:Checkstyle
    augrou  END
```

ウンロードし、~/.vim/libの下に置いておくか、変数g:Checkstyle_Classpathでパスを指定します。また、スタイル定義に用いるXMLファイルは、お好みのものをダウンロードし、同じフォルダに置いたうえで、変数g:Checkstyle_XMLで指定します。筆者はsun_checks.xmlというSun Code Conventionsに基づいたチェックができるXMLファイルを使っています。エラーが消えるまではQuickfixが閉じないようになっているので、必然的にきれいなコードになります(図2)。

Javaのコード整形にgoogle-java-formatを使う

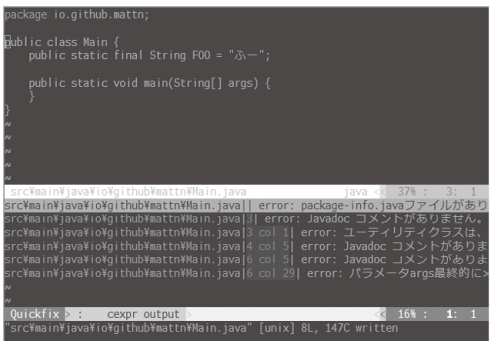
VimでJavaのコードを書いていると、Vimのユーザスキルによってはソースコードが汚くなりがちです。そこでgoogle-java-formatというコマンドを使い、ソースコードを成形します。google-java-formatはGoogleの開発者が提供しているソフトウェアで、IntelliJ IDEAのプラグインとして開発されていますが、コマンドラインから使えるjarプログラムとしても使えます。

まずリポジトリ[注8]からgoogle-java-formatを入手します。残念ながら執筆時点ではテストが通らないため、次のコマンドを実行してjarファイルを生成します。

```
$ mvn -Dmaven.test.skip=true package
```

エラーは発生しますが「core/target/google-java-format-0.1-SNAPSHOT.jar」というファイルが生成されるので、これを使ってJavaファイルを整形します。ただこのままだとVimから使いづらいため、筆者が用意したVimプラグイン[注9]を使用します。このプラグインを使うためには、先に作成したjarファイルを次のフォルダに格納しておく必要があります。

▼図2　checkstyleのソースコード検証



- Linux/MAC OS Xの場合
  ~/.vim/lib/google-java-format-0.1-SNAPSHOT.jar
- Windowsの場合
  %USERPROFILE%¥.vim¥lib¥google-java-format-0.1-SNAPSHOT.jar
  もしくは、
  %USERPROFILE%¥vimfiles¥lib¥google-java-format-0.1-SNAPSHOT.jar

あとはVimでJavaのソースコードを開き、

```
:JavaFmt
```

を実行するか、もしくはファイル名を指定して、

```
:JavaFmt＜Javaソースコードのパス＞
```

を実行することで、Google Java Coding Styleの形式に合わせてソースコードが整形されます(リスト8)。g:javafmt_optionsという変数によりgoogle-java-formatのオプションが指定できます。執筆時点ではvim-javafmtから使用できるgoogle-java-formatのオプションは次の--aospのみとなります。

注8) URL https://github.com/google/google-java-format

注9) URL https://github.com/mattn/vim-javafmt

```
--aosp, -aosp, -a
```

このオプションは、Google Style（4スペースインデント）の代わりにAOSP Style（8スペース）を使用する指定です（AOSPはAndroid Open Source Projectの略）。

ぜひ、きれいなソースコードを心がけてください。

◆ ◆ ◆

vim-javacomplete2の登場によって、VimでJavaの開発を行う障壁は大きく下がりました。Eclipseではできないような高速なJavaコーディングをぜひ体験してみてください。また、今回誌面の都合で紹介できなかったJava関連のプラグインを使えば、さらに便利に、さらに効率的にJavaの開発を行えるようになります。SD

▼リスト8　Google Java Coding Styleの形式に合わせてソースコードを整形

実行前

```
package sample;

public final class App
{
public static void main(final String[] args
) {
Foo foo = new Foo();
        System.out.println(
                "Hello World!" + foo.getMessage());
    }

    protected App() {
       }

}
```

実行後

```
package sample;

public final class App {
  public static void main(final String[] args) {
    Foo foo = new Foo();
    System.out.println("Hello World!" + foo.getMessage());
  }

  protected App() {}
}
```

Vim自報

より使いやすくなったctags

みなさん、Vimでソースコードを編集する際にctags[注A]はお使いでしょうか？　ctagsはソースコードからタグファイルを生成するプログラムです。Vimとctagsを連携することで、関数や変数、マクロの宣言位置へ高速にジャンプできるようになります。実は昔、ctagsはVimのリポジトリに含まれていました。それから時代が過ぎ、ctagsはSourceForge上で開発されてきましたが、作者であるDarren Hiebert氏が活発に開発できなくなってしまい、Universal Ctagsが開発を引き継ぐことになりました。

ctagsから生成されるtagsファイルは、個々のソースコードのエンコーディングが異なっている場合に、Vimから正しく扱えないという問題がありました。今回、Universal Ctagsへ開発の本流が移ったことを契機に、vim-jpではこの問題を修正するpull-requestを行いました。実際にはctagsで各言語のエンコーディングを指定できるようになりました。たとえば、`~/.ctags`に次の設定を追加しておくと、Javaとバッチファイルはシフト JISとして扱って、tagsファイルが生成されるようになります。

```
--input-encoding=utf-8
--input-encoding-java=cp932
--input-encoding-dosbatch=cp932
```

注A）URL https://ctags.io/

思考をカタチにするエディタの使い方

るびきち流 Emacs超入門

Writer | るびきち twitter@rubikitch | http://rubikitch.com/

第20回 標準機能から「yasnippet」まで Emacsの入力支援

Emacsの操作性アップシリーズ、今回は入力支援編です。Emacsに標準で備わる、動／静的略語展開、スケルトンといったEmacs標準の入力支援についておさらいしたあと、これらを統合したパッケージとも言える「yasnippet」を紹介します。

確実なテキスト入力を

ども、るびきちです。先々月、先月と、操作性を高めるためにはキーボードでの操作をしっかりと整えることが重要だとお伝えしました。キーバインドを適切に設定しておけば、いつでもすばやくコマンドが呼び出せるからです。それと同時に、入力支援機能を活用して、ミスタイプなく入力する確実な方法を確立すれば一気に使いやすくなります。

標準的な入力支援

動的略語展開

まず真っ先に使うべき入力支援機能はdabbrev（動的略語展開）です。長い文字列の入力を省力化する重要な機能です。この機能がないテキストエディタは正直、使いものになりません。

テキストエディタを使っていると、どうしても同じ単語を何度もタイプすることが多くなります。しかし、毎回馬鹿正直にタイプすると、時間はかかるしタイプミスが起こりやすくなります。Emacsではその問題に対する解決策が用意されています。

最初の数文字をタイプしてからM-/を押してみましょう。すると“魔法”が発動し、その数文字から始まる単語に補完されます。たとえば、interのあとにM-/を押すとinternet、interesting、interactive、interactivelyなどの単語に変化します。再度M-/を押すと別の単語になります。

そのカラクリはというと、入力された文字列から始まる単語をカーソルに近いほうから順次探索しています。カレントバッファで見つからない場合はほかのバッファからも探索します。知ってしまえば当たり前に思えるしくみも、初めて使うとあたかも魔法が発動したかのような感動を覚えることでしょう。入力支援基本のキとして、M-/は常用しましょう。

静的略語展開

動的略語展開が動的ならば、静的な略語展開もあるのではないか？　もちろんあります。

静的略語展開は、シンプルに略語から長い文字列を展開する機能です。動的略語展開が動的なのは、変化する単語の結果がバッファの内容に依存するからです。対して静的略語展開は変化する単語の結果が決まっています。たとえばinterから必ずinternetに展開されるように略語展開の設定ができます。設定したうえで、M-x abbrev-modeでマイナーモードを有効にし、interとタイプ後、スペースやカンマなどの単語区切り文字を入力すれば、internetと展開され

ます。この機能は後で紹介するyasnippetに完全に置き換わってしまうので、詳しくは述べません。

もし日本語入力にSKKを使っているのならば、アスキー文字から変換する機能があります。たとえば「/file」→「ファイル」のように変換できます。SKKは単語登録中心主義であるため、よく使う略語をガンガン登録すれば快適に入力できます。そう考えれば、SKKそれ自体が略語展開の機能を持っていると言えます。筆者は長年この機能を略語展開として使っています。

hippie-expand

M-x hippie-expandは動的略語展開、静的略語展開、ファイル名補完、シンボル補完などを統合した単語補完の“十徳ナイフ”です。M-/よりも高機能である反面、望みの補完をしてくれないことがあるのが玉に瑕です。

そこで筆者はhippie-exp-extパッケージにて、hippie-expandの機能性を活かしつつ補完の目的に沿ったコマンドを作成しました。M-x hippie-expand-dabbrev-limited-charsは、1バイト文字限定の動的略語展開および、「-」あるいは「_」から入力した場合に限り、長い文字列を途中から補完できるようにしたコマンドです。たとえば「-li」から「hippie-expand-dabbrev-limited-chars」と補完できます。

M-x hippie-expand-file-nameはファイル名補完に限定したコマンドです。

スケルトンによる定型文入力

コンピュータの世界におけるスケルトンとは骸骨……ではなくコードの骨格を意味します。Emacsにおけるスケルトンとは、パラメータを対話的に入力することで定型文を入力するコマンドです。スケルトンのコマンドを実行することを「スケルトンを展開する」といいます。

たとえば、次のコードを挿入するスケルトンを考えます。

```
(defun find-file--my-advice (&rest them)
  )
(advice-add 'find-file :around
            'find-file--my-advice)
```

これはEmacs 24.4から使える新しいアドバイス定義法で、関数を再定義せずに関数の挙動を変更できます。スケルトンの展開には、**表1**の情報が必要ですのでミニバッファから入力を求めます。これをスケルトンで表現すると**リスト1**のようになります。入力が複数ですので、スケルトンそのものではなく、コマンド定義とスケルトン展開の合わせ技です。スケルトン定義において> \nがインデントして改行するという指定で、「_」が展開後のカーソル位置です。M-x emacs-lisp-insert-advice-addを実行し、必要な情報を入力すれば先のコードが挿入されます。しかし、わかりづらいですよね。

▼表1　アドバイス定義で必要な入力

説明	変数名	具体的な文字列
元の関数名	symbol	find-file
場所	where	around
アドバイス名	name	my-advice

▼リスト1　スケルトンの設定テンプレート

```
(defun emacs-lisp-insert-advice-add (symbol where name)
  (interactive "s元の関数名: \ns場所: \nsアドバイス名: ")
  (skeleton-insert
   '("" nil    ;;;おまじない
     "(defun " symbol "--" name " (&rest them)" > \n
     _ ")" > \n
     "(advice-add '" symbol " :" where > \n
     " '" symbol "--" name ")" > \n)))
```

真打 yasnippet

略語展開とスケルトンの融合

略語展開は単に略語と展開結果の対応を表したもので、大した機能ではありません。入力作業全体から見てみれば略語→単語の略語展開による恩恵は微々たるものです。

一方で、略語展開の結果には関数(コマンド)を渡すことができます。スケルトンはコマンドですので略語展開の結果にスケルトンを割り当てられます。それをうまくやっているのがEmacs標準添付のpython.elによるpython-modeです。次の設定を加え、if、while、for、try、def、classのあとにスペースを押せばスケルトンが展開されます。

```
(setq python-skeleton-autoinsert t)
(add-hook 'python-mode-hook 'abbrev-mode)
```

スケルトンの問題

メジャーモード側で略語展開＋スケルトンの設定をしてくれているのは、ユーザからすれば親切といえます。しかし、スケルトンはS式であるため、細かい指定こそできるものの可読性が低いという欠点があります。穴埋めが複数あるスケルトンを定義するには、先の**リスト1**のようにコマンドを定義しなければなりません。

テンプレート展開はテキストエディタを効率よく使いたい一般ユーザとしてはぜひとも身につけておきたいところですが、elispプログラミングを要求するのは敷居が高過ぎます。読みづらいのはともかくとして、たかだか定型文のテンプレートを登録するのになぜelispの知識が必要なのでしょうか！？

yasnippet登場

スケルトンの使いづらさからか、テンプレート展開のelispは数多く存在します。スケルトンのようにS式ベースのテンプレート展開elispもありますが、やはりelispの知識を要求するので一般ユーザにはお勧めできません。elispがわかる人にとっても可読性の問題があり、お勧めできません。elispにはヒアドキュメントなどの高可読性の文字列表現がサポートされていないので、文字列を表現するには常に文字列リテラルを使う必要があります。elispの言語としての限界がそこにあります。

そうなると、必然的にテンプレートを独立したファイルに記述する方式が望まれます。この方式のelispもいくつか登場してきましたが、今ではyasnippetが定番です。

「スニペット」の簡単定義

先ほどのアドバイス定義のテンプレート(yasnippetではスニペットという)をyasnippetで定義すると、**リスト2**のようになります。暗号的なスケルトンと比較すれば可読性は明らかに上です。

冗長になっていますが、定義時(M-x yas-new-snippet)にあらかじめひな形が用意されるので丸暗記する必要はありません。スニペットの先頭から#で始まる行はコメントです。nameはスニペットの1行説明文、keyはそのスニペットに展開する略語です。スニペットの内容は# --行のあとに記述します。

スニペットの実体は、穴埋め部分を含むことができる定型文です。穴埋め部分が存在しない場合は、普通の略語展開と同じ機能です。

穴埋め部分は${1:symbol}のように、数字と表示文字列を指定します。数字は穴埋めされる順番で、表示文字列は穴埋めの説明の役割とデフォルト値の役割を果たします。スケルトンとの対比のために${4:where}と書いていますが、aroundと指定するケースが多いのならば、${4:around}と書いてデフォルト値として使いましょう。スニペット展開時はこの部分で文字列を入力することになります。表示文字列を書いていない$1のような指定は、穴埋め部分で入

▼リスト2 yasnippetでのスニペット定義テンプレート

```
# -*- mode: snippet -*-
# name: advice-add with function
# key: advice
# --
(defun ${1:symbol}--${2:name} (${3:&rest them})
  $0)
(advice-add '$1 :${4:where}
  '$1--$2)
```

力されたのと同じ文字列に置き換わります。最後に$0はスニペット展開後に移動するカーソル位置です。

インストールと設定

yasnippetはMELPAに登録されているのでM-x package-install yasnippetからインストールできます。パッケージをインストールするとyasnippet.el本体だけでなく、数多くのスニペットも同時にインストールされます。

そして、次の設定をします。

```
(require 'yasnippet)
(yas-global-mode 1)
;;; スニペット名をidoで選択する
(setq yas-prompt-functions
    '(yas-ido-prompt))
```

スニペットの置き場はyas-snippet-dirsで指定しますが、デフォルトは次のようになっています。

```
("~/.emacs.d/snippets"
 yas-installed-snippets-dir)
```

~/.emacs.d/snippetsは自分で定義したスニペットを置くディレクトリです。yas-installed-snippets-dirはパッケージによってインストールされたスニペットディレクトリの変数です。ほかのスニペットも使いたい場合は、この変数を適宜設定してください。

スニペットを展開する

スニペットを展開するには、略語(key)を入力してTabを押します。穴埋め部分が存在する場合はそこにカーソルが移動し、入力できるようになるので、入力したらTabで次の穴埋め部分に移動します。穴埋め部分を空白にするにはC-dです。すべての穴埋めが終われば展開終了です。

ミニバッファで情報を入力するスケルトンと違い、穴埋め部分にカーソルが移動するので、入力するべきテキストが明らかになるのが強みです。

略語が思い出せない場合はM-x yas-insert-snippetで展開します。またM-x yas-describe-tablesでスニペットを一覧します。

おわりに

今回は基本的な入力支援をおさらいしてから、yasnippetという強力なテンプレート入力の入口を紹介しました。yasnippetはそのまま使うだけでも入力を省力化できますが、可読性が高いフォーマットですので自分でスニペットを定義すれば、より便利なものとなります。次回はスニペット定義・テスト方法から始まり、いろいろな応用技を見ていきます。

筆者はサイト「日刊Emacs」を運営し、毎日パッケージの紹介記事を書いています。マイナーなものも紹介しているので、新たなパッケージを求めている人の役に立てば幸いです。また、EmacsユーザのQOLを上げるための厳選した情報を週間メルマガで配信しています。Emacsについてはもちろんのこと、ライフハックなどいろいろな分野について書いています。SD

登録はこちら➡http://www.mag2.com/m/0001373131.html

書いて覚える Swift 入門

第10回　例外を避ける？

Writer 小飼 弾（こがい だん）　twitter @dankogai

唯一変わったのは、そのすべて

「唯一変わったのは、そのすべて」。iPhone 6sのキャッチコピーですが、むしろそれはSwiftにこそふさわしい一言ではないかというぐらいSwiftは大きく変わりました。前回はそれを広く浅く紹介したのですが、今回からはそれぞれの変化を深く見ていきましょう。

Type？

Swiftの最大の特長は何かと問われたら、筆者はOptional型の多用だと答えます。本連載を最初から追いかけてくださっている読者の皆さんは納得していただけると思いますが、そうではない読者のために、ここで一度おさらいしておきましょう。

次のようなDictionaryがあったとします。

```
var supportedLanguages = [
    "C" : 1,
    "ObjectiveC" : 2,
]
```

次はなんとprintするでしょうか？

```
print(supportedLanguages["C"])
```

1ではなく、Optional(1)ですね。では次は？

```
print(supportedLanguages["Swift"])
```

nilとなります。

今度はOptional(1)ではなく1となります。この挙動を、型に着目して追っていきましょう。まずsupportedLanguagesの型は[String:Int]です。つまりStringを添字にすると、対応するIntが返ってくるデータ型なのですが、その中にないStringを添字には何を返したらよいでしょう？　クラッシュするか「何もない」を何らかの形で返すかのどちらかということになります。Swiftが採用したのは後者でした。この「何もない」のがnilで、「nilか値を返す」のがOptional型です。つまりsupportedLanguages[k]の型は、Intではなく Optional<Int> つまりInt?ということになります。

ところでSwiftには、enumがあります。Optional型をenumで表現するとどうなるでしょうか？　こんな感じでしょうか。

```
enum Optional<T> {
  case Nil
  case Some(T)
}
```

Swiftの実装は、まさにそのようになっています。[String, Int]が実はDictionary<String, Int>の構文糖衣であるように、Int?というのはOptional<Int>の構文糖衣に過ぎないのです。

There's more than one way to fail

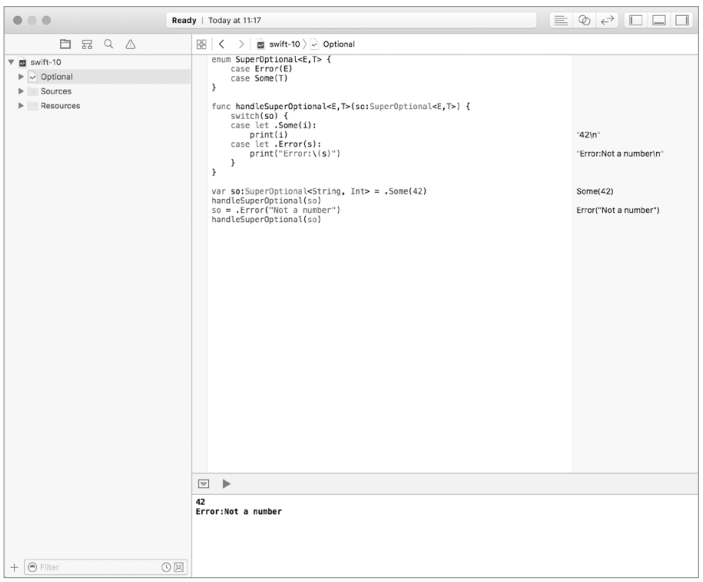
以上を踏まえて、次を見てみましょう。

```
var language = "C"
if let i = supportedLanguages[language] {
    print(i)
} else {
    print("Swift is not supported");
}
```

Optional(1)ではなく、1と表示されます。iの型はInt?ではなくIntで、ifに続く{}の中では100%例外なくiはIntであることが保証されている一方、elseに続く{}の中ではsupportedLanguages[language]がnilだったことが100%例外なく保証されているわけです。これがSwiftにおけるエラー処理の基本でした。Optional(と型変数)導入により、静的な型でもDictionaryのような動的に扱いたい型の扱いが動的言語なみに楽になったのです。

しかし、実際には「うまくいかない」だけではうまくいかないケースは少なくありません。「何がどううまくいかなかった」かによって、処理を変えたいケースも多いのです。たとえば「軽い」エラーならデフォルト値を代わりに使って続行し、「重い」エラーならプログラムを終了する。

▼図1　総称型enumの動作



そういった場合、どうしたらよいのでしょう？

SwiftのOptionalがenumで実装されていることを知っていれば、次のようなSuperOptionalを定義してしまえばその問題は解決しそうです。

```
enum SuperOptional<E,T> {
    case Error(E)
    case Some(T)
}

func handleSuperOptional<E,T>(so:SuperOptional<E,T>) {
    switch(so) {
    case let .Some(i):
        print(i)
    case let .Error(s):
        print("Error:¥(s)")
    }
}

var so:SuperOptional<String, Int> =
.Some(42)
handleSuperOptional(so)
so = .Error("Not a number")
handleSuperOptional(so)
```

ところが、Swift 1では型変数を複数持つ総称型enumはサポートしていなかったのです(**図1**)。

見てのとおりSwift 2では期待どおり動いていますが、Swift 1.xではコンパイラがクラッシュしてしまいます。

Swift 1におけるOptionalはenumで実装されていましたが、Swift 2におけるtry catchもenumによって実現されています。

give it a try

というわけでSwift 2のtry catchを実際に使ってみましょう。ここでは例題とし

て、「クラッシュしない配列」を実装してみます。

Swiftの Arrayの要素に範囲外の添字を与えると、問答無用でクラッシュします。

```
var ary = [0,1,2,3]
ary[4] // ここでクラッシュ
```

これはDictionaryとは異なる振る舞いです。

```
var dict = [0:0, 1:1, 2:2, 3:3]
dict[4] // nil
```

問答無用でクラッシュする代わりに、何らかのエラーを返すにはどのようにしたらよいでしょうか？

Swift 2では、まずどんなエラーを返すかを定義します。

```
enum ArrayError : ErrorType {
    case RangeError
}
```

見てのとおり、エラーはErrorType型を継承したenumです。次に、エラーを起こしうるメソッドを次のように定義します。

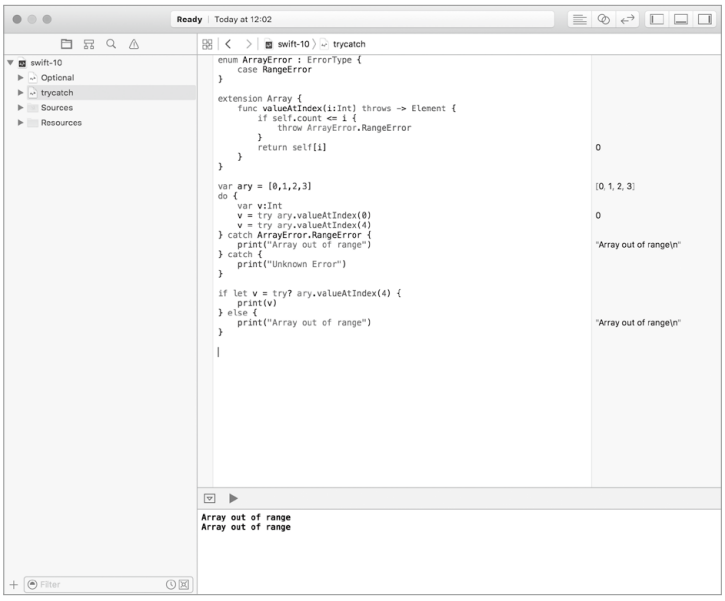
```
extension Array {
    func valueAtIndex(i:Int) throws ->
Element {
        if self.count <= i {
            throw ArrayError.RangeError
        }
        return self[i]
    }
}
```

通常のfuncと異なる点は2つ。1つは->の前にthrowsというキーワードが追加されていること、もう1つは範囲外であることを検出したうえで、その場合はthrow ArrayError.RangeErrorしていること。

あとはこれを使うだけ。

```
var ary = [0,1,2,3]
do {
    var v:Int
    v = try ary.valueAtIndex(0)
    v = try ary.valueAtIndex(4)
} catch {
    print("Array out of range")
}
```

▼図2　trycatchのエラー



たしかに今度はクラッシュせず、“Array out of range”と表示されるようになりました(**図2**)。ここでコードを見てみましょう。まず、実行ブロックがtryではなくdoで始まっています。そしてtryはary.valueAtIndex()の前についています。tryを取り除くとどうなりましたか？

Java(Script)と同様、catchは特定のエラーだけを捕まえることもできます。たとえば次のようにコードを書き換えてみましょう。

```
enum ArrayError : ErrorType {
    case OutOfBounds
    case NegativeBounds
}

extension Array {
    func valueAtIndex(i:Int) throws ->
Element {
        if i < 0 {
            throw ArrayError.NegativeBounds
        }
        if self.count <= i {
            throw ArrayError.OutOfBounds
        }
        return self[i]
    }
}

var ary = [0,1,2,3]
do {
    var v:Int
    v = try ary.valueAtIndex(0)
    v = try ary.valueAtIndex(-1)
} catch ArrayError.NegativeBounds {
    print("Array Index must be zero or
lager")
} catch ArrayError.OutOfBounds {
    print("Array Index too large")
} catch {
    print("Unknown Error")
}
```

例外を例外扱いしないSwift

さらにSwiftならではの特長として、catchでまとめて捕まえるのではなく、if letやguardで捕まえることもできます。

```
if let v = try? ary.valueAtIndex(4) {
    print(v)
} else {
    print("Array out of range")
}
```

もしくは

```
guard let v = try? ary.valueAtIndex(3) else
{
    print("Array out of range")
}
```

まとめると、次のとおりとなります。

- TypeErrorを継承したエラー型を定義
- エラーを起こしうるfuncには->の前にthrowsをつける
  （エラーを起こしたら定義したエラーをthrow）
- エラーを起こしうる関数／メソッドはtryする
  （エラーはcatchだけではなくif letやguardで使うこともできる）

このようにSwift 2のtry catch機構は、Java（Script）？のそれと比べると少し面倒ですが、例外ではなく飽くまで一般的なデータ型であるenumの自然な拡張として実現されている点が実に特長的です。構文の視点で見るとdo catchは「例外処理」ですが、型の視点で見ると飽くまでで普通の処理。Swift 2に合わせて改定された［The Swift Programming Language］[注1]でも、「例外」（exception）という言葉を避けて飽くまで「エラー処理」（Error Handling）としているのもそのためでしょう。

次号は

「なるべく一般的かつ包括的に」。それがさらに反映されているのが、Protocol Oriented Programmingという新たなスローガンでしょう。

次回は、いよいよこのProtocol Oriented Programmingを取り上げます。SD

注1）https://itunes.apple.com/jp/book/swift-programming-language/id881256329?l=en&mt=11

Erlangで学ぶ並行プログラミング

Author 力武健次技術士事務所 所長 力武 健次(りきたけ けんじ) http://rikitake.jp/

第9回 プロセス辞書とETSによる暗黙のデータ共有

この連載ではプログラミング言語Erlangとその並行プログラミングについて紹介していきます。今回は参照透過性の原則からあえて離れた暗黙の情報共有を可能にするErlangのプロセス辞書と、OTPのErlang Term Storage(ETS)という機能について紹介します(今回紹介予定だったOTPのデータベースMnesiaについては稿をあらためて次回紹介の予定です)。

OTP最新版の状況

本稿入稿時のErlang/OTPの最新版は18.1.3です[注1]。9月23日に18.1がリリースされました[1]。sslモジュールでTLS接続から単純なTCP接続へのダウングレードが可能になり、sshモジュールの鍵交換アルゴリズムにより強度の高いものが追加されたこと、またエラーログの量を制限できるようになったことなどがおもな変更点です。その後18.1.1[2]でinetsとmnesia両モジュールのバグ修正、18.1.2[3]と18.1.3(10月16日リリース)[4]ではsshモジュールの機能追加が行われました。なお、17.5系列ではバグ修正を含んだ17.5.6.4[5]が10月1日にリリースされています。

Erlangでの情報共有のスタイル、そして黒魔術としての暗黙のデータ共有

Erlangの基本的なプログラミングの方針として、状態が変化する前後の違いを明示的に記述し(参照透過性を確保し)、暗黙の情報共有を極力避ける(連載第2回を参照)という方針があります。この方針に沿った形で情報を共有したい場合は、原則として次の手法を採ります。

- 複数の関数の間で内部状態を共有する場合は、明示的に関数の引数や戻り値の中に含めて渡す[注2]。この典型的な例として、再帰による繰り返し実行がある
- 複数のプロセスの間で情報を共有する場合は、明示的にメッセージを送受信することで行う。データベースのように多数のプロセスで情報を共有する際は、共有する情報を内部状態として保持するプロセスを作り、当該プロセスにメッセージで読み書きを明示的に指示する

この参照透過性の確保を重視したデータの共有手法には、プログラムを読みやすくし、バグを出にくくするという効果があります。しかしその一方で、関数呼び出しやメッセージを介した1対1のやりとりの積み重ねとして処理を書くのが冗長な場合もあります。

複数の関数呼び出しにまたがる情報共有が必要な一例として、「1つ前に実行した関数の内部状態を使い演算をして結果を得て、その演算によって変化した内部状態を次の演算に使うために保存する」という作業があります。これを実現するには、参照透過性を確保するプログラミングスタイルでは関数に明示的な状態として

注1) 最新版はGitHubリポジトリを使いタグを指定することでビルドできます。詳細は「kerlでGitHub版のErlangをインストールする」(http://qiita.com/jj1bdx/items/4f7d7b5a53fcec32ab8d)を参照してください。

注2) Erlangでは参照渡しは行わないため、関数の引数や戻り値の受け渡しでは、値をコピーする処理をします。

「1つ前の内部状態」を引数に与えて、「現在の内部状態」を戻り値として得るという作業が必要です。仮に内部状態を複数の関数実行の間でどこか別の場所に置いておけるのであれば、内部状態をその場所から読み出して演算処理をして書き込むという作業ができるので、プログラミングは簡単になります[注3]。

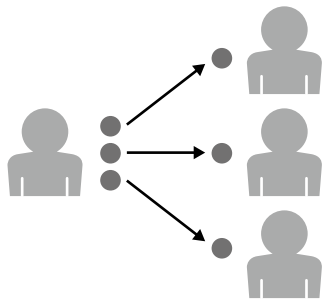
またほかの例として、ネットワークで言えばブロードキャストやマルチキャストなどの1対多の通信では、1対1のやり取りの繰り返しでは処理の記述が難解になり、また並列かつ並行に実行できる処理が逐次処理となってしまうため、速度の低下を招くことがあります。

たとえばErlangの基本的なプログラミングスタイルであるメッセージパッシングで複数の相手にデータ共有を行う際は(**図1**)、それぞれの相手に確実に伝達しその記録が取れるという利点の代償として、相手の数だけメッセージを繰り返して送らなければなりません。一方、共有メモリなど明示的でなく、書き込むのと読み出すのと場所が同じということを利用して暗黙の内にデータ共有をする際は(**図2**)、同一記憶場所に対するアクセスが並行かつ並列に行える限り速度を上げることができますが、その代償として何のデータがいつ書かれたのかについては別途通知し、かつロックなどで排他制御をしなければならなくなります。

Erlangでは極力メッセージパッシングのスタイルを取るということはこの連載で繰り返し書いてきました。しかし実際には、Erlang/OTPでもおもに速度の向上を目的として、暗黙の内にデータを共有する例外的なデータ構造(後述のプロセス辞書やETS)を使うことができるようになっています。ただし、これらの暗黙の共有を許容するデータ構造を使った場合、データ構造に対しては破壊的代入を行うため、参照透明性はなくなってしまいます。言い換えれば、データ構造の中身を変更する前と後の値を明示的に得ることができなくなるため、コードのデバッグは困難になります。また、暗黙の共有を許すデータ構造では、どのプロセスが書き込み、どのプロセスが読み出すかのタイミングを明確にしておかないと、競合状態(race condition)が発生して、予期しないバグに悩まされることになります。このよ

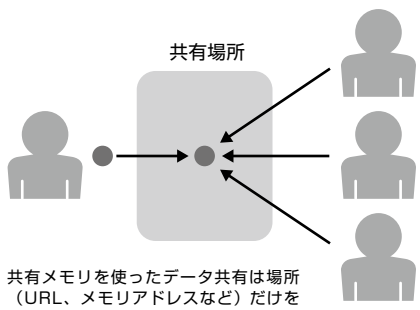
注3) たとえばOTPの疑似乱数モジュールrandにおいては、rand:uniform/{0,1}ではプロセス辞書に内部状態(rand:state()という型)を書き込みますが、rand:uniform_s/{1,2}では内部状態を関数の引数あるいは戻り値の中で明示的に受け渡しをします。

▼図1 メッセージパッシングによる共有



- メッセージパッシングによる共有では、送る側は何（メッセージ）を送るかを相手ごとに明示的に指定する必要がある。言い換えれば何を送ったかの記録を確実に取れる
- メッセージのやり取りの際には、誤り検出や抜けた情報の再送が可能なプロトコルを使うことで、相手に確実にメッセージを送りかつその確認ができる
- メッセージだけを考えて複数の相手に情報共有するには、仮に同じメッセージをすべての相手に送るとすれば相手の数だけ作業を繰り返さなければならないため、時間がかかるという欠点がある

▼図2 共有メモリを使ったデータの共有



- 共有メモリを使ったデータ共有では、書き込む側は読み出す相手の数に関係なく一度だけ書き込めばよい
- 読み出す側はそれぞれ自由なタイミングで読み出すことができる。書き込む側の一貫性さえ確保されていれば読み出す処理は並行かつ並列化できる
- この方式の欠点は、場所だけしかわからないので内容が書き換わった場合は別途メッセージパッシングなどで通知する必要が出てくることである

共有メモリを使ったデータ共有は場所（URL、メモリアドレスなど）だけを指定し、データをその場所にいつ誰が置くかは明示的にはわからない。そのため確実なデータのやり取りには別途同期をとったり一貫性を確保する排他制御（ロックなど）のしくみが必要になる

うな理由により、Erlangのプログラミングでは暗黙の共有を許容するデータ構造を必要もないのに使うことは推奨されていません[6][7]。これらの例外的なデータ構造は、いわば「黒魔術」として、普段は使うべきでないものと筆者は考えます(とはいえ、現実のプログラミングでは速度向上のために取らざるを得ない手法でもあります)。

プロセス辞書

Erlangのプロセスでは、プロセスに属する複数の関数間でのデータ共有のために、プロセス全体で1つだけプロセス辞書(Process Dictionary)というKey-value型データストア(KVS)を使うことができます。

プロセス辞書はプロセスの生成とともに作られ、初期値は空です。キーと値それぞれにErlangの項(アトムだけでなくタプルやリストなども可)を取ることができます。プロセスが終了するまでプロセス辞書は保持されますから、複数の関数呼び出しにわたってプロセス内部の状態を保持できます。これらの操作には組込み関数(BIF)のget/{0,1}, put/2, erase/{0,1}, get_keys/0で操作できます。プロセス辞書の中身は、pidがわかればBIFのprocess_info(Pid, dictionary)として外部から取得することもできます注4。

リスト1に簡単なカウンタ(連載第3回を参照)の例を示します。メッセージパッシングを使わずput/2とget/1で書いているため、Erlangの通常のプログラミングスタイルとはかなり違った感じになっています。シェルから実行した例を**図3**に示します。過去の連載で紹介したカウンタと一見同じ動作をしますが、その内容はプロセス辞書に書かれているため、同じプロセスから呼ばれている関数からしか操作できないのが欠点です。Erlangの並行動作はプロセス単位ですから、より実用的なものにするためには、複数のプロセスからアクセス可能なデータ構造を使う必要があります注5。

ETS:ノード内で使える共有データ構造

プロセス辞書が同一プロセス内でしか使えないのに対し、Erlang Term Storage(ETS)は、仮想マシンBEAMに実装されているメモリ上

注4) process_info/{1,2}では、プロセス辞書の内容のコピーを作り、それを戻り値として返します。そのため、大きなプロセス辞書、あるいは大量のプロセスに対して実行した場合は、BEAMのメモリ使用量が増大することでシステムの動作に影響を及ぼすことが指摘されています(http://videlalvaro.github.io/2015/05/erlang-process-dict.htmlを参照)。

注5) もちろんカウンタの内容を含むプロセス辞書を持つプロセスに対して、メッセージパッシング等でやり取りすることでほかのプロセスからカウンタの内容を読み書きすることはできますが、その方法は本連載の過去の記事にて各種実装として説明しているため今回は割愛します。別の言い方をするなら、プロセス辞書を使った実装はプロセス内部の状態の持ち方が変わるだけだとも言えます。

▼リスト1　プロセス辞書を使ったカウンタの例(msgcounter_procdict.erlモジュール)

```
-module(msgcounter_procdict).
関数名は連載第3回のmsgcounterモジュールに準じる
-export([start/1, stop/1, inc/1, dec/1, z
ero/1, val/1]).
辞書内のカウンタのキーは{?MODULE, カウンタの名前}とする。
?MODULEはモジュール名(msgcounter_procdict)を示すマクロであ
る。名前を引数に取るカウンタを使えるようにしている
-spec start(term()) -> term().
start(Name) ->
    put({?MODULE, Name}, 0),
    Name.
名前を引数に取るカウンタを消す
-spec stop(term()) -> ok.
stop(Name) ->
    erase({?MODULE, Name}),
    ok.
名前で指定したカウンタの値を1つ増やす
-spec inc(term()) -> integer().
inc(Name) ->
    New = get({?MODULE, Name}) + 1,
    put({?MODULE, Name}, New),
    put/2の戻り値は変更「前」の値なので明示的に変更後の値を返す
    New.
名前で指定したカウンタの値を1つ減らす
-spec dec(term()) -> integer().
dec(Name) ->
    New = get({?MODULE, Name}) - 1,
    put({?MODULE, Name}, New),
    New.
名前で指定したカウンタの値をゼロ(0)にする
-spec zero(term()) -> 0.
zero(Name) ->
    put({?MODULE, Name}, 0),
    0.
名前で指定したカウンタの値を返す
-spec val(pid()) -> integer().
val(Name) -> get({?MODULE, Name}).
```

▼図3 msgcounter_procdictモジュールの実行例

```
Eshell V7.1  (abort with ^G)  モジュールをロードする
1> l(msgcounter_procdict).
{module,msgcounter_procdict}
シェルで管理しているプロセス辞書には何も入ってない
2> get().
[]
3> msgcounter_procdict:start(c1).
1つめのカウンタc1を起動する
c1
4> msgcounter_procdict:start(c2).
2つめのカウンタc2を起動する
c2
5> msgcounter_procdict:inc(c1).
2つのカウンタが独立しているのがわかる
1
6> msgcounter_procdict:dec(c2).
-1
7> msgcounter_procdict:inc(c1).
2
8> msgcounter_procdict:val(c2).
-1
9> get().
プロセス辞書の状態を見るとカウンタの状態が反映されている
[{{msgcounter_procdict,c1},2},↵
{{msgcounter_procdict,c2},-1}]
10> get_keys().  プロセス辞書のキーだけをリストとして列挙する
[{msgcounter_procdict,c1},↵
{msgcounter_procdict,c2}]
11> get({msgcounter_procdict,c1}).
キーがわかれば同じプロセス内ならアクセス可能
2
12> self().  process_info/2で外部から様子をうかがえる
<0.36.0>
13> process_info(self(), dictionary).
{dictionary,[{{msgcounter_procdict,c1},2},
             {{msgcounter_procdict,c2},-1}]}
14> msgcounter_procdict:stop(c1).
カウンタを停止（プロセス辞書から消去）
ok
15> msgcounter_procdict:stop(c2).
ok
16> get().  消えているのがわかる
[]
User switch command
ここで【CTRL】+【G】を押してノード内に別のシェルを立ち上げる
 --> ?
  c [nn]            - connect to job
  i [nn]            - interrupt job
  k [nn]            - kill job
  j                 - list all jobs
  s [shell]         - start local shell
  r [node [shell]]  - start remote shell
  q                 - quit erlang
  ? | h             - this message
 --> s  ←新しくローカルシェルを起動する
 --> j  ←ノード内のジョブのリストを見る
   1  {shell,start,[init]}
  2* {shell,start,[]}
 --> c 2  ←新しいシェルに切り替える
Eshell V7.1  (abort with ^G)
1> self().  この新しいシェルのpidは前のものとは違う
<0.46.0>
2> process_info(pid(0,36,0), dictionary).
前のシェルのpidを指定すればプロセス辞書の中身が見える
{dictionary,[{{msgcounter_procdict,c1},2},
             {{msgcounter_procdict,c2},-1}]}
```

で動作するKVSで、ETSのあるノードで動くすべてのプロセスからアクセスできるようになっています。この特徴によりプロセス間の情報共有の速度を上げる方法の1つとして使うことができます。

ETSでは複数の連想配列（テーブル）を持つことができます。その上限値は既定値では約1400です[注6]。各テーブルはOTPのetsモジュール[8]に定義された関数を使って作成や消去をしたり、テーブルの内容を読み書きできます。テーブルを作成したプロセスは「オーナープロセス」とされ、オーナープロセスが終了すると作成されたテーブルも消えます[注7]。各テーブルは作成時に名前を付けたり、オーナープロセス以外のアクセス権を定めることができます。

ETSのテーブルのアクセス権には次の3つがあります。既定値のprotectedを使うことで、複数のプロセスが同時に書き込もうとすることによる競合を防ぐことができます。

- private：オーナープロセスしか読み書きはできない
- protected（既定値）：書き込みはオーナープロセスだけが可能だが、読み出しや検索はets:new/2で返されるテーブルID（整数あるいは指定した名前）を知っていればノード内のどのプロセスからもできる
- public：テーブルIDを知っていればノード内のどのプロセスからも書き込みと読み出し双方ができる

リスト2にmsgcounter_procdictモジュールと同様の機能を持つカウンタの例を示します。etsモジュールには多くの関数が用意されており、このモジュールは実質的にはetsモジュールのラッパーとして書くことができています。

注6） ETSのテーブル個数はBEAM起動時に環境変数ERL_MAX_ETS_TABLESを設定することで増やすことができます。

注7） ETSのテーブルを作成する関数ets:new/2にheirオプションを付けることで、プロセス終了時にほかのプロセスをオーナープロセスにできます。また、ets:give_away/3という関数を使うことで明示的に他のプロセスに所有権を移すこともできます。

▼リスト2　ETSを使ったカウンタの例（msgcounter_procdist.erlモジュール）

```
 関数名は連載第3回のmsgcounterモジュールに準じる
-module(msgcounter_ets).
-export([init/0, cleanup/0,
         start/1, stop/1, inc/1, dec/1, z
ero/1, val/1]).
 MODULEはモジュール名（msgcounter_ets）を示すマクロ。ETSのテーブル名は?MODULEとする。カウンタを列挙するETSテーブルを初期化する。アクセス権はpublicなのでどのプロセスからも読み書きできる。ETSテーブルを検索するキーは、keyposオプションで位置を指定できる（既定値は1でタプルの最初の要素）
-spec init() -> atom().
init() ->
    ets:new(?MODULE, [named_table, public]).
カウンタを列挙するETSテーブルを消去
-spec cleanup() -> true.
cleanup() ->
    ets:delete(?MODULE).
 名前を引数に取るカウンタを使えるようにする
-spec start(term()) -> term().
start(Name) ->
    ets:insert(?MODULE, {Name, 0}).
 名前を引数に取るカウンタを消す
-spec stop(term()) -> ok.
stop(Name) ->
    ets:delete(?MODULE, Name).
 名前で指定したカウンタの値を1つ増やす。ここで登場するets:update_counter/3はテーブル内のエントリに存在するタプルの指定した位置の要素をカウンタとして操作する。操作中は他のプロセスはテーブルにアクセスできないため値の更新に関する一貫性が保たれる
-spec inc(term()) -> integer().
inc(Name) ->
    ets:update_counter(?MODULE, Name, {2, z
1}).
 名前で指定したカウンタの値を1つ減らす
-spec dec(term()) -> integer().
dec(Name) ->
    ets:update_counter(?MODULE, Name, {2, z
-1}).
 名前で指定したカウンタの値をゼロ(0)にする
-spec zero(term()) -> 0.
zero(Name) ->
    ets:insert(?MODULE, {Name, 0}),
    0.
 名前で指定したカウンタの値を返す
-spec val(pid()) -> integer().
val(Name) ->
    ets:lookup_element(?MODULE, Name, 2).
```

msgcounter_etsモジュールの各関数をシェルから実行した例を**図4**に示します。カウンタはプロセス辞書での実装と同様の動作をしますが、大きな違いはオーナープロセスが存在している限り、ノード内の他のプロセスからもアクセスできることです。またets:update_counter/3という更新の際の一貫性を確保した関数を使うことで、同時にカウンタへのアクセスが複数のプロセスからあってもデータが壊れないようになっています[注8]。

ETSにはその他の機能として、テーブル中のエントリに対してマッチスペック（match specification）によるパターンマッチング検索を行う機能や、テーブルをファイルやDETS（ディスクベースのETS同様のKVS）[9]と相互に変換する機能など、インメモリデータベースとして使うために十分な機能がそろっています。ただし、ETS自身はデータの自動複製の機能などはないため、トランザクションかつ耐障害性の必要なデータベースを組むには別のしくみが必要です[注9]。

プロセス辞書とETSの違い

プロセス辞書とETSには次に述べる実装上の違いがあります[10]。

- プロセス辞書は各プロセスが占めるプロセスヒープ内に存在し、ガーベッジコレクション（GC）の対象となる。これは実行速度に影響する。一方、ETSはGCの対象にならない
- プロセス辞書の読み書きには内容のコピーを必要としない。一方、ETSのテーブルは読み書きに内容のコピーを必要とする。これは実行速度に影響する
- プロセス辞書は当該プロセスが終了すると消える。一方、ETSではテーブルのオーナープロセスが終了してもほかのプロセスに所有権を渡すことでテーブルの中身を保持し続けることができる
- プロセス辞書の内容は単純なキーの一致か不一致でしか検索できず、ほかのプロセスに割り込まれずに読み出しと書き込みを一度の操

注8） ETSで保証されているのは、テーブルの各要素の更新中には他のプロセスが影響することがなく、更新が完了するか失敗するかのどちらかであることです。ただし、ETSは本質的には暗黙の内にデータを共有しているデータ構造ですから、相互排除問題を解決するには、別途ロックなどの排他制御を行う必要があります。

注9） 耐障害性を確保するためには外部のデータベースバックエンドを使うこともできますが、Erlang/OTPではMnesia（http://erlang.org/doc/man/mnesia.html）という分散データベースが用意されています（次回紹介予定）。

▼図4 msgcounter_etsモジュールの実行例

```
Eshell V7.1  (abort with ^G)
1> l(msgcounter_ets).
{module,msgcounter_ets} ets:i/0はノード内にあるETSテーブルの概要を表示する。複数のテーブルがすでに存在していることがわかる
2> ets:i().
 id               name               type
size   mem      owner
 ----------------------------------------
---------------------------------
 1                code               set
278    13029    code_server
(中略)
 inet_db          inet_db            set
29     576      inet_db
 inet_hosts_byaddr inet_hosts_byaddr bag
0      305      inet_db
 inet_hosts_byname inet_hosts_byname bag
0      305      inet_db
 inet_hosts_file_byaddr inet_hosts_file_
byaddr bag   0      305      inet_db
 inet_hosts_file_byname inet_hosts_file_
byname bag   0      305      inet_db
ok
 ets:i/1で(アクセス権があれば)各テーブルの中身を見られる
3> ets:i(inet_db).
<1   > {res_usevc,false}
<2   > {res_alt_ns,[]}
<3   > {res_hosts_file,"/etc/hosts"}
<4   > {tcp_module,inet_tcp}
<5   > {udp_module,inet_udp}
 …… (中略) ……
<28  > {res_resolv_conf_tm,0}
<29  > {res_hosts_file_info,undefined}
EOT  (q)uit (p)Digits (k)ill /Regexp -->q
 ←インタラクティブなので【q】を入れて止める
ok
 カウンタを格納するETSテーブルを作る
4> msgcounter_ets:init().
msgcounter_ets
 ETSテーブルのリストを見ると、msgcounter_etsが入っていることがわかる
5> ets:all().
[msgcounter_ets,8207,file_io_servers,inet_
hosts_file_byaddr,
 inet_hosts_file_byname,inet_hosts_
byaddr,inet_hosts_byname,
 inet_cache,inet_db,global_pid_ids,global_
pid_names,
 global_names_ext,global_names,global_
locks,4098,1,ac_tab]
 カウンタを起動する
6> msgcounter_ets:start(c1).
true
7> msgcounter_ets:start(c2).
true
 カウンタが正常動作しているのがわかる
8> msgcounter_ets:inc(c1).
1
9> msgcounter_ets:dec(c2).
-1
10> msgcounter_ets:inc(c1).
2
11> msgcounter_ets:val(c2).
-1
 ETSテーブルの中では次のように表現されている
12> ets:i(msgcounter_ets).
<1   > {c1,2}
<2   > {c2,-1}
EOT  (q)uit (p)Digits (k)ill /Regexp -->q
 ←【q】を入力して止める
ok
 ets:info/1ではETSテーブルの諸情報を見ることができる
13> ets:info(msgcounter_ets).
[{read_concurrency,false},
 {write_concurrency,false},
 {compressed,false},
 {memory,319},
 {owner,<0.37.0>},
 {heir,none},
 {name,msgcounter_ets},
 {size,2},
 {node,nonode@nohost},
 {named_table,true},
 {type,set},
 {keypos,1},
 {protection,public}]
14>
User switch command ここでノード内に新しいシェルを起動する
 --> s ← シェルを起動
 --> j ← ジョブのリストを見る
   1  {shell,start,[init]}
   2* {shell,start,[]}
 --> c 2 ← 新しいシェルに制御を移す
Eshell V7.1  (abort with ^G)
 プロセス辞書の場合と違いカウンタの値が引き継がれる
1> msgcounter_ets:inc(c1).
3
 カウンタを停止(ETSテーブルからエントリを削除)する
2> msgcounter_ets:stop(c1).
true
3> msgcounter_ets:stop(c2).
true
4> ets:i(msgcounter_ets).
 エントリがなくなっているのがわかる
EOT  (q)uit (p)Digits (k)ill /Regexp -->q
 ←【q】を入力して止める
ok
5> msgcounter_ets:cleanup(). ETSテーブルを消去する
true
```

作で行うことができない。ETSではパターンマッチングを使って検索したり、fold演算(連載第3回を参照)や他のプロセスに割り込まれないカウンタとして使うなど、複雑な操作を行うことができる

・プロセス辞書の内容はノードが異常停止した際にエラーログに吐き出される。一方、ETSの内容はノードが異常停止した際には、オーナープロセスが停止するため、消滅してしまう。前者はエラーログの量が膨大になり、後者は情報の消失のために、いずれの場合もデバッグが困難になるという問題が起きる。これは参照透過性を犠牲にした破壊的操作の結果としては不可避であると筆者は考える

まとめ

今回はErlangのプロセス辞書ならびにOTPのETSという、2つの暗黙のデータ共有の手法について紹介しました。繰り返しますが、プロセス辞書もETSもErlang/OTPの大原則である非破壊代入や参照透明性といった特徴をあえて(おもに実行速度の向上のために)曲げて実装されている機能ですから、使う際は競合状態や相互排除問題が生じないかどうかについて細心の注意を払って使うことが必要です。

次回はより高度なOTPのデータベースであるMnesiaについて紹介する予定です。

ソースコードとサポートページ

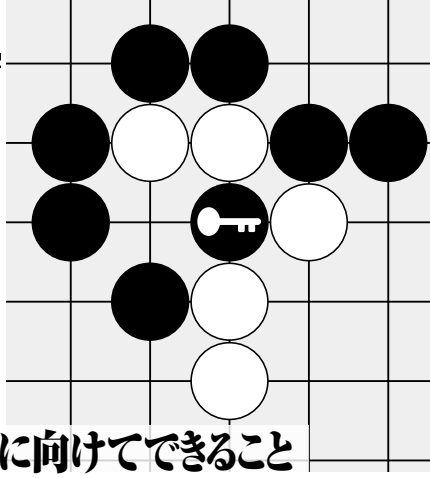
連載の記事で紹介したソースコードなどGitHubのリポジトリに置いています(https://github.com/jj1bdx/sd-erlang-public/)。どうぞご活用ください。SD

参考文献

[1] http://www.erlang.org/news/92
[2] http://erlang.org/pipermail/erlang-questions/2015-October/086285.html
[3] http://erlang.org/pipermail/erlang-questions/2015-October/086443.html
[4] http://erlang.org/pipermail/erlang-questions/2015-October/086474.html
[5] http://www.erlang.org/download/OTP-17.5.6.4.README
[6] Joe Armstrong, Robert Virding, Claes Wikström, and Mike Williams,"Concurrent Programming in Erlang" (Second Edition), Prentice-Hall,1996, ISBN-10: 0-13-508301-X, p. 132 (Section 9.6, "ProcessDictionary").
[7] Fred Hébert, "On the Use of the Process Dictionary in Erlang",2011-03-31,http://ferd.ca/on-the-use-of-the-process-dictionary-in-erlang.html
[8] http://erlang.org/doc/man/ets.html
[9] http://erlang.org/doc/man/dets.html
[10] Ulf Wiger, erlang-questions mailing list, 2007-04-30,http://erlang.org/pipermail/erlang-questions/2007-April/026330.html

セキュリティ実践の基本定石

すずきひろのぶ
suzuki.hironobu@gmail.com



みんなでもう一度見つめなおそう

【第二七回】同じ轍を踏まないために。IoT時代に向けてできること

先日、Linux.Wifatchというマルウェア（？）が家庭用ルータに侵入し勝手に設定を書き換え、セキュリティを高めていたという報道がなされました。善意のハッカーの登場ということで注目を集めた事件でしたが、少し冷静にこの事件を分析してみると、IoTのセキュリティには、インターネット黎明期やWebアプリケーション黎明期と同じような問題をはらんでいることが見えてきます。

マルウェア？ Linux.Wifatch

先日、Twitterで「善意のハッカー」が脆弱性のある家庭用ルータを勝手に書き換え、セキュリティを高めていたという話題が流れていました。

> GIGAZINE
> 「1万台ものルーターを何者かが勝手にハックしてセキュリティを高めていたことが発覚」[注1]

非常に興味深いので、情報ソースであるSymantec Official Blog[注2]の該当の記事を詳しく読んでみました。ここで説明されているLinux.Wifatchは、家庭用LinuxルータやIoT製品などに侵入し、その侵入先のシステムをコントロールするというマルウェアで、機能的にはこれまで何度となく聞いてきた典型的なボット型マルウェアと言えるでしょう。そういう意味ではちまたに溢れるマルウェアの1つですが、ただし、その目的があまり聞いたことのないものでした。その目的とは、

感染したルータやIoT製品のセキュリティを高める

だったのです。

ここまで読んで、頭の中にたくさんのクエスチョンマークが浮かんだ方もいるかと思いますので、以下にもう少し整理してみます。

ただし、筆者はこのマルウェア（善意のソフトウェアですので「マルウェア」と呼ぶのは不適当かもしれませんが）の目的の良し悪しではなく、技術的に興味深い部分をクローズアップしていきます。

Symantec社のサイトのマルウェア情報[注3]によれば、Linux.Wifatchはバックドア型トロイの木馬で、Linuxを搭載しているルータに侵入／感染します。Linux.Wifatchはシステムに侵入したらバックドアを作り、C&C（Command and Control）サーバに接続し、感染先ルータをコントロール下におきます。発見したのは2015年1月とのことです。

経済誌Forbesのサイト[注4]によれば、少なくとも10,000台のルータがLinux.Wifatchに感染していると見積もられています。

ルータをコントロール下において何をしているかと言えば、徹底したセキュリティ強化のためのセットアップです。Symantec社のサイトの情報を読んでいてわかったことは、「Linux.Wifatchに感染したルータは、Linuxセキュリティとしてやるべきことをほぼ一通り行われているので、かなり安全性の高い

注1）http://gigazine.net/news/20151002-hacked-routers-secure/
注2）Symantec Official Blog "Is there an Internet-of-Things vigilante out there?" 2015年10月1日
http://www.symantec.com/connect/blogs/there-internet-things-vigilante-out-there
注3）https://www.symantec.com/security_response/writeup.jsp?docid=2015-011216-2314-99
注4）http://www.forbes.com/sites/thomasbrewster/2015/10/01/vigilante-malware-makes-you-safer/

ルータになっている」ということです。どんなことを行われていたのか、次にいくつか例を挙げます。

- ルータのファイアウォール機能を強化し、外部からの通信をブロックし安全性を高める
- 強制的にパスワードを強化する設定にする
- 実行権限を整理して安全な設定にする
- 不必要なサービスを停止／削除する
- セキュリティ／アップデートを施す
- その他、多くのセキュリティ強化のためのチューニングを行う

しかも、意図的にtelnetポートは開いていて、そこにtelnetで接続するであろうルータの持ち主に次のような内容のメッセージを表示します。

「これ以上感染しないようにtelnetは使えなくしています。telnetを使用禁止にし、telnetのパスワードを変更し、そしてファームウェアをアップデートしてください。」

まだまだ存在するデフォルトのままのパスワード

後日、Linux.Wifatchの作者はSymantec Official Blogのインタビューで次のようにを答えています。

「Linux.Wifatchは侵入するために巧妙なバックドアやゼロデイアタックなどは使っていない」
「telnetなどのパスワードに簡単なもの（“password”など）を試しているだけ」

出荷時の共通の管理者アカウント名と共通のパスワード（すべての製品にデフォルトで、アカウント“admin”、パスワード“password”と設定しているようなもの）にしたまま販売しているネットワーク製品が、今でも多いことが容易に想像できます。

感染国の割合が示唆に富む内容で、**表1**のようになっています。

ヨーロッパの先進国や日本は上位に入っていません。急速に経済発展を遂げているブラジル、ロシア、インド、中国をまとめてBRICsと呼びますが、このリストには、そのうちの3つが入っています。ロシアの代わりにベトナム、トルコといった、やはり急速に力をつけてきている国が入っています。

インターネットにあるネットワーク機器の脆弱性をデータベース化しているSHODAN[注5]を使って検索してみれば、デフォルトのパスワードのままのルータに関する傾向がつかめるかもしれません。そこでSHODANで“router default password”という3つのキーワードで検索してみました。

国別にはトップ5はアメリカ、インド、中国、アルゼンチン、サウジアラビアという並びになりました。アプリケーション／サービスに関してのトップはダントツでtelnetでした。組織別のデータを見ると、先ほどの国にあるISPに連動しています。

日本でもそうですが、普通はインターネットサービスを契約した際、ISPに接続するルータはISPが提供する機材であるケースがほとんどです。ですから、ISP単位でネットワーク単一機種ルータが稼動しており、そこに脆弱性があれば（この場合は単一のログイン名と単一のパスワードですが）、このような結果になると考えるのが合理的だと思います。

つまり、インターネットが先行して普及した国の多くは、すでにこのような基本的なセキュリティに関しては、経験を積んでおり慎重になっていると考えていいと思います。たとえば、日本国内の家庭用ルータも、現在ではデフォルトパスワードが1台1台異なるようになっています。

◆**表1　Linux.Wifatchの感染国の割合**

国名	割合
中国	32%
ブラジル	16%
メキシコ	9%
インド	9%
ベトナム	7%
イタリア	7%
トルコ	7%
韓国	5%
アメリカ	5%
ポーランド	3%

注5）https://www.shodan.io/

問題はルータだけではない

しかし、インターネットに接続しているLinuxベースの機材はルータだけではありません。今や家電製品でもLinuxベースの製品は多々あります。ハードディスク・レコーダーなどの取扱い説明書を見てみると、最後にGPLライセンスが載っていることも、そんなに珍しいことではありません。

家電製品がLinuxベースというより、中身がデスクトップPCやサーバと区別がつかないものもあります。広く出回っているディストリビューションに少し手を入れただけの製品でも十分に動くので、それで出荷しているものもかなりあるかと思います。そして、そのような機材が一般家庭に設置され、インターネットに直接接続されて、簡単に乗っ取られてしまうという状況は否定できないようです。

SHODANでの検索ランキングの上位に顔を出すDreambox[注6]というネットワーク側からアクセスできる装置があります(**写真1**)。有料テレビや衛星テレビの番組を見るために使われるセット・トップ・ボックスだそうで、スペイン、ラトビア、スウェーデン、ドイツなどおもにヨーロッパ方面で使われているようです。

インターネットの上のブログなどを参考にすると、どうやらDreamboxのLinuxシステムのrootが単一パスワードになっているようです。もちろん、ユーザは本来rootのパスワードは知ることができません。しかし、ハードウェアを手に入れシステムのパスワードファイルを取り出し、パスワードをクラックしてしまえば、rootでログインできるようになってしまいます。英数字のみからなる6文字程度のパスワードは、今日の計算機の能力を使えば正しい値を見つけるのは難しいことではありません。

SHODANの情報を見ると、このボックスはtelnet経由でアクセス可能なようです。そして、検索ランキングが上位にあるということは、かなりの回数、興味をもって検索されているようです。この状況を見る限りデフォルトのパスワードが流出していると考えて良いのかもしれません。

どこかで見たような、これまでと同じ問題であっても、これまでの利用範囲ではなく、新しい分野の新しい製品で問題が繰り返し発生していく様子が、ここから見てとれます。

IoTとLinux

組み込みLinuxはIPv4/IPv6に強く、ライセンスの関係もあって、大量に使われるIoT製品には向いているようです。また、Raspberry Piに代表されるような、CPUがARMベースの小さなLinux Boxが脚光を浴びています。半導体メーカー最大手のIntel社もIoT向けのシステムEdison用のLinux Development Kitを配布しています。今後、IoTの分野で組み込みLinuxがさらに大きな存在になっていくでしょう。

しかし、Linuxは、スーパーコンピュータからIoTまで基本的に同一アーキテクチャです。つまり、これらの製品でセキュリティを作り込もうとすると、必要とする基礎知識はスーパーコンピュータでも、IoTでも同じなのです。

これまで組み込みはハードウェアの制限や、その制限されたハードウェアで動くソフトウェアでの制限がありました。ですので、まずは最小限の機能を載せるのが精一杯でした。しかし、今やハードウェアには十分な処理能力と容量があり、何か便利そうな機能があれば容易に載せておけます。一方で、1

◆**写真1　有料テレビや衛星テレビのためのチューナーDreambox**

(出典：https://en.wikipedia.org/wiki/Dreambox)

注6) http://www.dream-multimedia-tv.de/

つ1つの機能を吟味して、安全性を高める作業をしなければなりません。しかし、何が起こるかを想定し安全性を高めるのは、技術的にも時間的にもたいへんコストのかかる作業です。その安全性とコストのバランスをうまく取るのは至難の技と言えます。

ましてやまだ普及期にも届かないIoT技術です。十分なノウハウがたまっているとは言いかねます。また、組み込み系雑誌を読んでいてもIoT特集としてネットワークからアクセスできるアプリケーションの作り方を説明していますが、その安全性に対する説明はほとんど省かれています。紹介というレベルですのでしかたのないことかと思いますが、その背景にあるべき安全性に対する考え方まで伝わっているかどうかたいへん不安です。

Windows 95の悲劇再び？

インターネットが一般に普及したのはWindows 95以降と言われています。それまでインターネットを活用していたのは、大手企業や研究者などUNIXシステム、とくにUNIXワークステーションを使うようなユーザが中心でした。

インターネットのセキュリティ脅威は、1988年のモリスワームが引き起こした大規模な通信障害以来、常に懸念事項です。またインターネット上の通信は暗号化しない限り安全性が保てないことも、すでにそれまでのインターネットユーザには常識でした。また、外部から不正に接続されないようにフィルタリングをする、あるいは接続管理をするといったことも当たり前のようにされていました。もちろん、人間が管理する以上、今も昔も間違いや失敗はありましたし、そこから生じるセキュリティ上の問題も、今も昔も本質的には変わりがありません。

ところが、Windows 95の登場によって引き起こされた状況は、ちょっとそれとは異なります。Windows 95/98のTCP/IPの能力はLANを前提にして作られています。フォルダをLANに公開するのは簡単で、アクセスするにもパスワードは必須ではなく、LANにWindows 95/98をつなげばアクセスできる、「誰でも簡単につなげられる」ものでした。

さらに当時は、ファイアウォールのような機能もなしにインターネットに接続するような環境が当たり前のようにあったので、自分のPCのフォルダを全世界に公開するなんていうのも平気でありました。全世界とまではいかなくても、ケーブルテレビ会社のネットワーク経由でインターネットに接続すると、マンションが1つのLANになっていて、隣の家のフォルダが見えるといったこともありました。

高度と言うほどではない、それまでのネットワーク技術者にとって当たり前のような、ちょっとした常識レベルのセキュリティすらも、新しく参入してきたグループ（ソフトウェアベンダ、ISP、ユーザ）には伝わっていませんでした。

新規参入者には、「最先端」であるネットワーク技術を見よう見まねでもかまわないので取り入れることが最優先だったとしか筆者には思えません。

Webアプリケーションの経験

初期のWebアプリケーションも状況はひどいものでした。とにかくWebサービスのニーズは高いものですから、どんどん新規参入してきました。しかし、技術を学ぶのは一朝一夕にできるものではありません。何が起こったかというと、雑誌や本に書いてあるコードを見よう見まねで取り入れました。

雑誌や本において、サンプルコードは、理解を促進するために煩雑なエラー処理やセキュリティ機能などは省き、本質的な部分のみを示すのが一般的です。さらにOSの持つ機能は知っていることを前提に、説明を省いていることがほとんどです。そのため、WebサーバがUNIXのとき、OSとしてUNIXの持つ実効ユーザID／実効グループIDがどう作用するのかなど基本的な理解がないままプログラミングし、適切なアクセス管理ができていないこともしばしばでした。

コマンドインジェクションやSQLインジェクションなど、ユーザ入力を直接実行系に渡してしまい、問題を引き起こすようなことも普通に発生します。コンパイラを勉強すると必ず学ぶTombstone diagram（あるいはT-diagramsとも呼ぶ）の知識があれば、別言語体系で入力させ、変換したあとに実行するといった知恵も出てきたのかもしれません。

しかし、現実にはこのような教養とも言える基礎知識はWebアプリケーション作成の範囲から追い出されていて、別な知識として学ぶ必要がある体系になっています[注7]。

Webアプリケーション作成の現場だけではなく、レンタルサーバサービスの管理側が実行権限とファイルアクセス権限の関係性を理解できていないためにWebサーバとコンテンツマネージメントシステムの不整合が発生し、その間隙(かんげき)をついてユーザサイトが第三者によって改ざんされたという事例もあります[注8]。

しかしながら、このような苦い経験を経て、ベストプラクティスとしてコミュニティの中に知識として溜まり、少しずつではあるのですがセキュリティが改善してきてはいるのです。

IoTで繰り返さないために

IoTでは、少なくとも組み込みエンジニアが持っている知識と、ネットワーク技術者の知識に加え、これまでのインターネット上で繰り返されてきたセキュリティ侵害の知識が必要になります。

そうでなければ、Windows 95/98のころにあったPCからのインターネット利用の立ち上がり期でのトラブルや、Webアプリケーション立ち上がり期のトラブルが、今度はこれから立ち上がるIoTで繰り返されることになるでしょう。

これだけはほしいIoTセキュリティ機能

ルータの安全性を高めていたLinux.Wifatchですが、本来はこのようなことがないようにベンダがきちんとやるべきなのは言を待ちません。

では、LinuxベースのIoTを想定し、ネットワーク的にどのような形が望ましいかを考えてみます。

①工場出荷時のパスワードは個別にし、単一のパスワードなどは使わない

②パスワード設定が必要な部分は自動的にパスワードを生成するか、あるいはユーザの入力するパスワードの強度をチェックする

③必要のないポートはデフォルトでフィルタリングしておき、使うポートのみ明示的に通過させるようにする

④クライアント機能に徹しサーバ機能は持たせないで、通知にはプッシュ機能を使う

⑤通信は電子認証が可能でかつ暗号経路が可能な相手(たとえばTLSやVPN)としか行わない

⑥アップデート通知を受け取り、ユーザによらない自動アップデートを行う

⑦内部で動くプログラムは何でもrootではなく、適切なユーザ権限を割り当て、万が一のセキュリティ侵害でも被害を最小限に止める

⑧SELinuxを有効にし、意図しないアプリケーションの導入などを許さない

⑨正当な権限をもたないアクセスが頻繁にあった場合、通信を禁止にする

普通、スマートフォンはサーバの機能を持たずプッシュ通知により相手からの情報をもらうという形にしていますが、このような形にすることでセキュリティを高めることができます。

この手のマルウェアのノード(ボット)管理は、プッシュで命令を送ったり、通信に暗号を使ったりなど解析が難しいメカニズムを採用しているものもあります。裏を返せばセキュリティの問題がどこにあるのかがわかっている人たちが作るので、利用されている技術は皮肉というか当然というか可用性が高く安全性を考慮するものになっているのでした。

筆者はLinux.Wifatchの作者のようなやり方には賛同できませんが、上記のリストもLinux.Wifatchの作者が考えている具体的な対策と重なるところが多々あります。必要なセキュリティを考えれば、基本は同じ、ということなのかもしれません。SD

注7) 安全なウェブサイトの作り方 https://www.ipa.go.jp/files/000017316.pdf

注8) paperboy & co.レンタルサーバサービス「ロリポップ！レンタルサーバー」「第三者によるユーザーサイトの改ざん被害に関するご報告」2013年9月9日 http://lolipop.jp/info/news/4149/

Sphinxで始めるドキュメント作成術

小宮 健 Komiya Takeshi Twitter @tk0miya

第9回 ドキュメントに図を入れよう ——さまざまなグラフィックツールとの連携

今回のテーマ

今回のテーマは「図」です。図はドキュメントを書くうえで欠かせない要素の1つです。文章だけではイメージがつかみづらいことも、図を使って説明することでドキュメントを読む人に理解してもらいやすくなります。そこで今回は、Sphinxにおける図の使い方を取り上げます。

figureディレクティブ

ドキュメンテーションツールの中にはMicrosoft Wordのように作図機能を持つものがありますが、Sphinxが利用しているreStructuredTextには、それ単体で図を作る機能がありません。

reStructuredTextで図や画像を扱いたい場合はあらかじめ外部のグラフィックツール(ペイントなど)を利用して画像ファイルを作成しておき、figureディレクティブ(本連載第3回で紹介)を使ってドキュメントの中に画像を埋め込みます。

figureディレクティブの使用例
```
.. figure:: 画像ファイル.png

   サンプル画像  ←ここはキャプションになる
```

表1に示すように、figureディレクティブには画像の大きさやリンクに関するオプションがいくつかあります。

たとえば、scaleオプションを使って画像の大きさを50%にリサイズする場合は、次のように指定します。

scaleオプションの使用例
```
.. figure:: images/sphinx-logo.png
   :scale: 50%

   Sphinxのロゴ
```

オプションは、figureディレクティブの次の行に「**:オプション名: 設定値**」と指定します。

外部のツールで図を作成する

figureディレクティブは一般的な画像形式に対応しているため、さまざまなツールで作成し

▼表1 figureディレクティブのおもなオプション

オプション名	概要
align	画像の表示位置をleft、center、rightから選択する
alt	画像の説明を指定する。HTMLのalt属性などに利用される
width	画像をリサイズする。画像の幅をピクセル単位で指定する(例:120px)
height	画像をリサイズする。画像の高さをピクセル単位で指定する(例:120px)
scale	画像をリサイズする。画像の大きさを%で指定する(例:50%)
target	画像にリンクを張る。リンク先のURLを指定する

た図を利用できます。ここでは、よく使われているツールを紹介します。

Cacoo

Cacoo[注1]はNulab社の提供するオンライン・ドローイングツールです。ブラウザを利用して自由に作図できます(**図1**)。

Cacooで作成した図をSphinxドキュメントに埋め込むには、Cacooのエクスポート機能を利用して、作成した図を画像ファイルに変換します(**図2**)。エクスポート機能はさまざまな形式の画像ファイルへの変換が可能ですが、PNG形式の画像ファイルとしてSphinxプロジェクト内に保存すれば、figureディレクティブを使うことでドキュメントに埋め込めます。

■ sphinxcontrib-cacoo

エクスポート機能を使うとCacooの図をSphinxドキュメントに埋め込めますが、図を更新するたびにエクスポートを行わなくてはならないため、ドキュメントへの反映には手間がかかります。

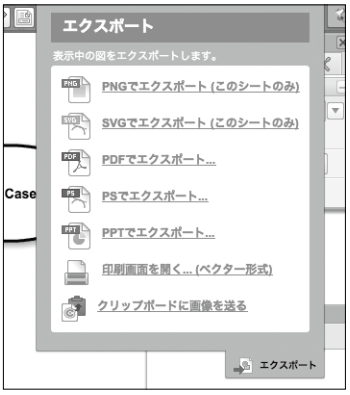
このエクスポートの手間を軽減するSphinx拡張が、sphinxcontrib-cacoo[注2]です。sphinxcon

注1) http://cacoo.com/

注2) https://pypi.python.org/pypi/sphinxcontrib-cacoo

▼図1 Cacooでの作図

▼図2 Cacooのエクスポート機能



COLUMN

figureディレクティブとimageディレクティブ

Sphinxの画像系ディレクティブには、figureディレクティブのほかに、imageディレクティブが用意されています。imageディレクティブはfigureディレクティブと同様に、引数に画像へのパスを指定して画像をドキュメントに埋め込みます。

imageディレクティブの使用例

```
.. image:: 画像ファイル.png
```

2つのディレクティブには次のような違いがあります。

- imageディレクティブはキャプションを持たない(設定できない)
- figureディレクティブはブロック要素(別の段落になる)として扱われるが、imageディレクティブはインライン要素(テキストの間に入れられる)として扱われる

用途に合わせて2つのディレクティブを使い分けると良いでしょう。

trib-cacooはCacoo APIを利用して図のエクスポートを自動的に行うため、手動でエクスポート操作を行うことなくCacooで作成した図をドキュメントに埋め込めます。

また、タイムスタンプを使って図の更新をチェックしているので、Sphinxの変換時に図が更新されていれば、自動的に再エクスポートを行い最新の図を取り込みます。

sphinxcontrib-cacooはサードパーティ製のSphinx拡張です。利用するには次のようにしてインストールを行う必要があります。

sphinxcontrib-cacooのインストール

```
$ pip install sphinxcontrib-cacoo
```

そして、conf.pyでsphinxcontrib-cacooを有効にします。同時に、CacooのAPIキーをcacoo_apikeyにセットします。

sphinxcontrib-cacooの設定（conf.py）

```
extensions = ['sphinxcontrib.cacoo']
cacoo_apikey = 'your apikey'
```

CacooのAPIキーは、Cacooの設定画面のAPIキーメニュー（**図3**）から生成します。

図を埋め込むには、figureディレクティブの代わりにcacoo-figureディレクティブを使います。cacoo-figureディレクティブの引数にはCacooで作成した図のURLを指定します（**リスト1**）。

cacoo-figureディレクティブはfigureディレクティブとの互換性があり、キャプションを指定できるほか、scaleやaltなどのオプションにも対応しています。

▼図3　CacooのAPIキーの生成画面

▼リスト1　cacoo-figureディレクティブの使用例

```
.. cacoo-figure:: https://cacoo.com/diagrams/mb53vvmYG38QGUPf

   Cacooで作成したユースケース図
```

ドキュメントの生成には、これまでどおりmake htmlを実行します。sphinxcontrib-cacooは自動的に図を取り込みます(**図4**)。

Microsoft Visio

Microsoft社のVisioも、よく利用される作図ツールです。Visioにはステンシルと呼ばれる図形テンプレート機能があり、図の部品を共有できます。Cisco社をはじめネットワーク関係の企業が多くステンシルを公開しているので、ネットワーク図などによく使われています(**図5**)。

Cacooと同様、Visioも作成した図をPNGなどの形式で保存できるため、figureディレクティブを使ってSphinxドキュメントに図を埋め込めます。

■ sphinxcontrib-visio

Visio専用のSphinx拡張として、sphinxcontrib-visio[注3]が提供されています。sphinxcontrib-visioはCOM(Component Object Model)を利用してVisioを呼び出し、Visio形式の図(.vsdx)を自動でPNGに変換してSphinxドキュメントに埋め込みます。また、sphinxcontrib-cacooと同様に図の更新をチェックしているので、図が更新されている場合はSphinxの変換時に自動的に取り込みます。

sphinxcontrib-visioは内部でVisioを呼び出しているため、Visioがインストールされたマシンでのみ動作します。また、sphinxcontrib-visioをインストールする際は、事前に依存ライブラリであるpywin32をインストールする必要があります。pywin32はPython for Windows ExtensionのWebサイト[注4]でインストーラが配布されています。インストーラはPythonのバージョン(2.6、2.7、3.4など)とプロセッ

注3) https://github.com/visio2img/sphinxcontrib-visio
注4) http://sourceforge.net/projects/pywin32/

▼図4　sphinxcontrib-cacooの出力結果

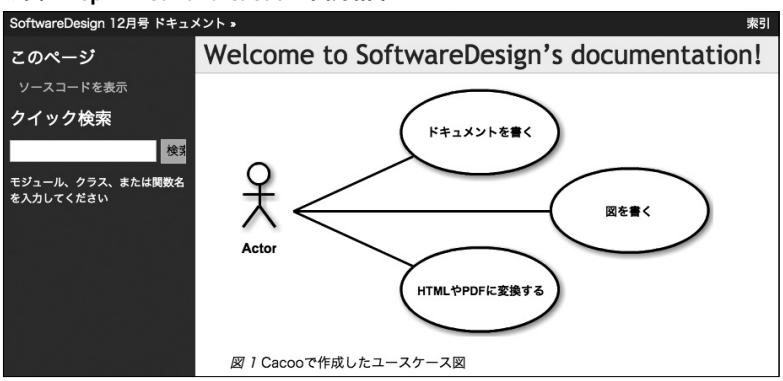
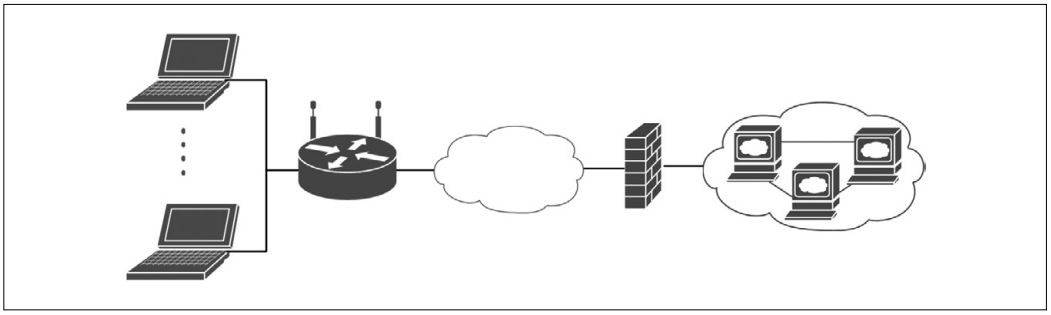


図 1 Cacooで作成したユースケース図

▼図5　Cisco社提供のVisioステンシルで描いた図



サ・アーキテクチャ(32/64ビット)によって分かれています。適切なバージョンを選択してインストールしてください。

次にsphinxcontrib-visioをインストールします。

sphinxcontrib-visioのインストール
```
$ pip install sphinxcontrib-visio
```

最後に、conf.pyでsphinxcontrib-visioを有効にします。

sphinxcontrib-visioの設定(conf.py)
```
extensions = ['sphinxcontrib.visio']
```

sphinxcontrib-visioを利用して図を埋め込むには、figureディレクティブの代わりにvisio-figureディレクティブを使います。引数にはVisioで作成したファイル(.vsdx)へのパスを指定します。

visio-figureディレクティブの使用例
```
.. visio-figure:: example.vsdx

   Visioで作成したユースケース図
```

複数のシートを持つVisioファイルを扱う場合は、pageオプションもしくはsheetオプションを使って、ドキュメントに埋め込むシートを指定します(**表2**)。

ドキュメントの生成には、これまでどおりmake htmlを実行します。sphinxcontrib-visioが自動的に図の取り込みを行います。

次回予告

今回はSphinxでの図の使い方と各種ツールとの連携方法について紹介しました。次回はテキストマークアップから図を生成する方法について紹介します。SD

▼表2　visio-figureディレクティブのオプション

オプション名	概要
page	ページ番号(1、2、3、……)を使って変換対象のシートを指定する
sheet	シート名を使って変換対象のシートを指定する
alt、scaleなど	そのほか、figureディレクティブと同じオプションに対応している

COLUMN

その他のグラフィックツールとSphinx

今回はCacooとVisioについて取り上げましたが、ほかにもSphinxと連携できるツールはいくつかあります(**表A**)。

Sphinx向けに図を作成する際には、自分が使うツールに合わせたSphinx拡張がないか確かめてみると良いでしょう。

拡張によって書式やオプションが異なるため、利用する際は各ツールのREADMEを確認してください。

▼表A　おもな画像系Sphinx拡張

Sphinx拡張の名称	対応するツール／サービス
sphinxcontrib-cacoo	Cacoo
sphinxcontrib-visio	Microsoft Visio
pptshape	Microsoft PowerPoint
sphinxcontrib-astah	astah*
sphinxcontrib-libreoffice	LibreOffice

COLUMN

PyCon JP 2015

Author 清水川 貴之

本連載の執筆陣の1人、清水川です。

2015年10月9～12日に、東京でPyCon JP 2015[注A]が開催されました。筆者はカンファレンスに参加し、Sphinxの発表を行い、Sphinx-users.jpもハンズオン、ポスターセッション、スプリントを開催しました。本コラムでは各活動の様子を紹介します。

■ハンズオン

ハンズオンでは、8名の参加者すべてがSphinxを使い始めたところということでした。Sphinxを使う目的は、Wordの置き換えとしてテキストベースのドキュメンテーションツールを導入したい、という方がほとんどでした。リポジトリで管理しやすい、差分を把握しやすい、プログラムと一緒に管理しやすい、ネットワーク図やシーケンス図を描きやすい、といったあたりがSphinxを使うメリットとして見られているようです。

チュートリアルは、ユーザ会のサイトに掲載している「Sphinxをはじめよう」という記事[注B]をもとに行いました。まずは、sphinx-quickstartから始めて、複数ページの扱い方、記法の練習などを各自で進めてもらい、質問があれば手を挙げてもらって講師が答える、というスタイルです。

チュートリアルの最後には、個別に出た質問と回答をいくつか紹介しました。たとえば、テーブルの作成方法として、Sphinxで使える4つの方法とそれぞれの特徴について紹介しました。テーブルの書き方については、本連載の第4回でも紹介しましたので、そちらもご参照ください。

■プレゼンテーション

筆者は、「Sphinxで作る貢献しやすいドキュメント翻訳の仕組み」というタイトルで、Sphinxのドキュメント翻訳サポート機能(i18n)について紹介し、約100名の方が参加してくれました。発表の最初に参加者に行った質問で、「技術文書の翻訳はほかの技術者の助けになると思いますか？」という問いに、8割近くの参加者が手を挙げました。技術文書の翻訳は、日本では高いニーズがあると言えるでしょう。

筆者の発表では、翻訳者が挫折せずに参加しやすい翻訳のしくみを紹介しました。Sphinxの多言語化機能とTransifex[注C]サービスを組み合わせて使うことで、そのようなしくみを作れます。いつか本連載でも紹介したいと思います。

■ポスターセッション

ポスターセッションでは、Sphinx関連のポスターを掲示し、訪れた参加者にSphinxを紹介したり、質問を受けたり、あるいはドキュメンテーションツールについて議論を交わしたりしました。

■スプリント

スプリントは短期集中型のソフトウェア開発イベントです。筆者はSphinxスプリントのリーダーとして5名の参加者と一緒にSphinxのドキュメント翻訳や開発などを行いました(**写真A**)。

PyCon JPイベント内での開催だったためか、初めて参加した方、遠方から参加した方が半数でした。今回はSphinxドキュメントの翻訳プロジェクトへの参加方法を教え、さっそくいくつかの文章翻訳に協力してもらいました。こういったイベントでは、文字では伝えづらいことを直接伝えられるのがメリットですね。

Sphinx-users.jpでは、このように集まって行うハッカソンイベントと、お茶会イベントをそれぞれ月に1回開催しています[注D]。ハッカソンは休日の日中に開催しており、今回のスプリントと同じように各参加者がそれぞれ題材を持ち寄って、Sphinxやドキュメンテーション、翻訳などについて質問や雑談をしながら各自作業しています。お茶会は平日夜に2時間ほどファミレスで開催しており、雑談や情報交換を中心に行っています。気楽なイベントですので、ぜひご参加ください。

▼写真A　筆者(後列左)とスプリントの参加者

注A) https://pycon.jp/2015/
注B) http://sphinx-users.jp/gettingstarted/
注C) https://www.transifex.com/
注D) http://sphinxjp.connpass.com/

Mackerelではじめるサーバ管理

Writer 田中 慎司(たなか しんじ) ㈱はてな
Twitter @stanaka

第10回 Mackerelの監視ルールをコードで管理しよう

監視の対象・種類、アラートの発動条件などを設定するMackerelの「監視ルール」。今回はその監視ルールを、コマンドラインで取得・設定できるツール「mkr」を紹介します。さらに、そのmkrを使ってGitHubで監視ルールを管理する方法を解説します。サーバをコードで管理する、イマドキの運用にぴったりです。

mkrでInfrastructure as Code

Mackerel[注1]では、サーバ運用をサポートするCLIツール「mkr」を提供しています(本連載の第6回[2015年8月号]で紹介)。その後、Mackerelの監視ルールを取得・更新するためのAPIが公開され、それに合わせてmkrもアップデートされています[注2]。

mkrを利用して監視ルールをAPI経由で扱えるようにすることは、いわゆるInfrastructure as Codeの考え方に合致するものです。今回は監視ルールをmkrで扱う方法と、それを応用して監視設定をGitHubで管理する方法を紹介します。

mkrを導入する

mkrはMackerelの各種設定を取得・更新するためのCLIツールで、ホストのステータスをまとめて変更したり、手順をスクリプトに組み込んで自動化したりすることが可能です。mkrのコードはGitHub[注3]で公開しています。OS Xでbrewを利用してインストールする場合は次のようにします。

```
% brew tap mackerelio/mackerel-agent
% brew install mkr
```

mkrに必要な最低限の設定として、APIキーを環境変数で指定します。

```
% export MACKEREL_APIKEY=<API key>
```

brew以外にもaptやyum、go getによるインストールもできます。詳細は本連載第6回もしくは公式ドキュメント[注4]を参照してください。

mkrで監視ルールを操作する

mkrでは、監視ルールの操作にmonitorsサブコマンドを使います。monitorsサブコマンドでは、次のようなpull/diff/pushの3種類の操作が可能です。

- pull
  Mackerelから監視ルール一覧を取得し、ローカルファイル「monitors.json」に保存
- diff
  Mackerelに設定されている監視ルール一覧と、ローカルファイル「monitors.json」との差分を表示

注1) URL https://mackerel.io
注2) URL http://blog-ja.mackerel.io/entry/2015/08/13/182620
注3) URL https://github.com/mackerelio/mkr
注4) URL http://help-ja.mackerel.io/entry/advanced/cli

・push
ローカルファイル「monitors.json」の監視ルールの設定をMackerelに反映

サブコマンドを指定しない場合、監視ルール一覧が標準出力に表示されます。

監視ルールのフォーマット

個々の監視ルールのJSONオブジェクトは、たとえば**リスト1**のようなものです。監視ルール一覧は、このJSONオブジェクトの配列となります。監視ルールに現れるフィールドとその意味を**表1**に示します。監視ルールの種類ごとに、必要なフィールドは変化します。

監視ルールの識別ロジック

idフィールドはMackerelが監視ルールを識別するためのIDとなります。このidはMackerelが付与しますので、後述のGitHubで管理する際にpushだけでの運用ができるように、nameによる識別方法も用意しています。具体的には次のようなロジックで監視ルールを識別します。

・idを含む……idで識別

▼リスト1　監視ルールの例

```
{
  "type": "host",
  "name": "disk.aa-00.writes.delta",
  "duration": 3,
  "metric": "disk.aa-00.writes.delta",
  "operator": ">",
  "warning": 20000.0,
  "critical": 400000.0,
  "scopes": [
    "Hatena-Blog"
  ],
  "excludeScopes": [
    "Hatena-Bookmark: db-master"
  ]
```

・idを含まず、nameを含む……nameで識別

同じnameを持つ監視ルールが存在した場合、idでのみ識別を行います。そのためすべての監視ルールにidが含まれる必要があり、ない場合はWarningメッセージを出力します。idもnameも含まないルールがある場合はエラーとなります。

mkr monitorsコマンドの実行例

mkr monitorsコマンドの実行例を**図1**と**図2**に示します。**図1**は、ローカルファイルとMackerel間で差分がある場合にdiffを実行した例で、**図2**

▼表1　監視ルールのフィールド

フィールド	説明
id	監視ルールのID
type	監視の種類(host：ホストメトリック監視、service：サービスメトリック監視、external：外形監視、connectivity：サーバの死活監視)
name	監視一覧などで参照できる任意の名称
service	監視対象となるサービス名
duration	指定された間隔(分)の平均値を監視する(有効範囲：1~5分)
metric	監視対象のホストメトリック名。特定の定数文字列を指定することで、割合監視が可能
operator	指定した数値より大きいか小さいかというアラート条件を指定。“>”または“<”(左辺が観測値で右辺が設定値)
warning	warningのアラート発生の閾値
critical	criticalのアラート発生の閾値
scopes	監視対象のサービス名またはロール詳細名
excludeScopes	監視除外対象のサービス名またはロール詳細名
responseTimeWarning	warningのアラート発生の応答時間の閾値(ミリ秒)。service指定が必要
responseTimeCritical	criticalのアラート発生の応答時間の閾値(ミリ秒)。service指定が必要
responseTimeDuration	指定された期間のリクエストの平均値を監視(1~5分)。service指定が必要
maxCheckAttempts	何回連続でwarning/criticalになったらアラートを発生させるか。デフォルトは1回(1~5回)

▼図1　ローカルファイルとMackerel間の差分がある場合のdiff

```
% mkr monitors diff
Summary: 1 modify, 1 append, 1 remove

  {
    "name": "Filesystem %",
    "type": "host",
    "metric": "disk%",
    "operator": ">",
-   "warning": 95.000000,
+   "warning": 96.000000,
    "critical": 99.000000,
    "duration": 3,
    "scopes": [
    ],
    "excludeScopes": [
    ],
  },
- {
-   "id": "<id>",
-   "name": "loadavg5",
-   "type": "host",
-   "metric": "loadavg5",
-   "operator": ">",
-   "warning": 1.000000,
-   "critical": 5.000000,
-   "duration": 3,
-   "scopes": [
-     "My-Service:proxy"
-   ],
-   "excludeScopes": [
-   ],
- },
+ {
+   "name": "loadavg",
+   "type": "host",
+   "metric": "loadavg5",
+   "operator": ">",
+   "warning": 1.000000,
+   "critical": 5.000000,
+   "duration": 5,
+   "scopes": [
+     "My-Service:proxy"
+   ],
+   "excludeScopes": [
+   ],
+ },
```

▼図2　ローカルファイルをMackerelにpush

```
% mkr monitors push --dry-run -F monitors_new.json
      info Create a new rule.
 {
   "name": "loadavg",
   "type": "host",
   "metric": "loadavg5",
   "operator": ">",
   "warning": 1.000000,
   "critical": 5.000000,
   "duration": 5,
   "scopes": [
     "My-Service:proxy"
   ],
   "excludeScopes": [
   ],
 },
      info Delete a rule.
 {
   "id": "<id-1>",
   "name": "loadavg5",
   "type": "host",
   "metric": "loadavg5",
   "operator": ">",
   "warning": 1.000000,
   "critical": 5.000000,
   "duration": 3,
   "scopes": [
     "My-Service:proxy"
   ],
   "excludeScopes": [
   ],
 },
      info Update a rule.
 {
   "id": "<id-2>",
   "name": "Filesystem %",
   "type": "host",
   "metric": "disk%",
   "operator": ">",
   "warning": 96.000000,
   "critical": 99.000000,
   "duration": 3,
   "scopes": [
   ],
   "excludeScopes": [
   ],
 },
```

はローカルファイルをMackerelにpushした例です。

監視ルールをGitHubで管理しよう

これまで紹介したとおり、mkrを使うことで監視ルールをローカルファイルに保存したり(pull)、ローカルファイルとMackerelの設定の差分を確認したり(diff)、ローカルファイルの内容をMackerelの設定に反映させたり(push)することができます。これらを利用してGitHubで監視ルールを管理する方法を紹介します。

mkrによる監視ルールの管理

mkrによる監視ルールの管理方法には、WebUIとローカルファイルの両方で変更を行うか、それともローカルファイルのみで変更するかで、大きく次の2パターンがあります。

・WebUIとローカルファイルの両方で監視ルールを変更する
　→pullとpushを利用する
・WebUIでは変更せず、ローカルファイルのみで監視ルールを変更する
　→pushのみを利用し、pullは利用しない

前述のとおり、mkr monitorsでは監視ルールの識別にidもしくはnameを利用します。前者ではidベースで管理しますので、ローカルファイルのJSONにもidを含める必要があります。後者はnameベースで管理しますので、ローカルファイルのJSONにidを含める必要がありません。ただしnameが重複していないことが必要条件です。

もしローカルファイルのJSONやMackerel側でnameが重複していた場合、mkrはidベースで監視ルールを特定しようとします。nameが重複しているにもかかわらず各監視ルールにidが存在しない場合は、mkrは「不正なJSON」という内容のエラーを出力します。

pullとpushを利用する

idベースで監視ルールを識別します。そのため監視ルールを新規登録したあとで、Mackerel側で付与されたidを取得する手順が必要となります。

●リポジトリを初期化

次の手順に従ってリポジトリを初期化します。

①GitHubにリポジトリを作成

②git clone

```
% git clone <repo-url>
% cd <repo-path>
```

③監視ルールをMackerelから取得

```
% mkr monitors pull
```

④GitHub上のリポジトリにコミット、プッシュ

```
% git add monitors.json
% git commit -m '<commit-msg>'
% git push
```

●リポジトリとMackerelの設定の同期が取れているかを確認

リポジトリとMackerelの設定が一致しているかどうかを確認するには次の手順に従います。

①GitHubから最新のデータを持ってくる

```
% cd <repo-path>
% git pull
```

②Mackerelとの差分を確認

```
% mkr monitors diff
```

③差分がない場合、次のような結果が得られる

```
Summary: 0 modify, 0 append, 0 remove
```

●WebUIで監視ルールを変更

Web側で監視ルールを変更した場合は次の手順でリポジトリに反映します。

①Web側で監視ルールを変更
②変更された監視ルールをMackerelから取得

```
% cd <repo-path>
% mkr monitors pull
```

③GitHub上のリポジトリにコミット、プッシュ

```
% git add monitors.json
% git commit -m '<commit-msg>'
% git push
```

●ローカルファイルから監視ルールを変更

ローカルファイルから監視ルールを変更した場合は次の手順でMackerelおよびリポジトリに反映します。

①GitHub上で変更・レビューなどを行い、masterブランチ(など)に反映
②変更された監視ルールをGitHubから取得

```
% cd <repo-path>
% git pull
```

③Mackerelとの差分を確認

```
% mkr monitors diff
```

④変更された監視ルールをMackerelに反映

```
% mkr monitors push
```

●ローカルファイルで監視ルールを新規追加

ローカルファイルから監視ルールを新規追加するには次の手順に従います。

①JSONを作成後、GitHub上で変更・レビューなどを行い、masterブランチ(など)に反映
(ここでのJSONはidなしのものを作成(JSONフォーマットの詳細は公式サイト[注5]を参照))
②Mackerelとの差分を確認

```
% mkr monitors diff
```

③追加された監視ルールをMackerelに反映

```
% mkr monitors push
```

④追加された監視ルールにMackerelが付与したid付きで取得

```
% mkr monitors pull
```

⑤GitHub上のリポジトリにコミット、プッシュ

```
% git add monitors.json
% git commit -m '<commit-msg>'
% git push
```

pushだけを利用する

nameベースで監視ルールを管理します。各監視ルールでnameが重複しないようにする必要があります。また、WebUI側での監視ルールの変更が非推奨となります。もし変更してしまった場合は手動でJSONを組み立てるか、pullを実行する必要があります。

●リポジトリを初期化

次の手順に従ってリポジトリを初期化します。

①GitHubにリポジトリを作成
②git cloneする

```
% git clone <repo-url>
% cd <repo-path>
```

③監視ルールをMackerelから取得

```
% mkr monitors pull
```

④GitHub上のリポジトリにコミット、プッシュ

```
% git add monitors.json
% git commit -m '<commit-msg>'
% git push
```

●リポジトリとMackerelの設定の同期が取れているかを確認

リポジトリとMackerelの設定が一致しているかどうかを確認するには次の手順に従います。

①GitHubから最新のデータを持ってくる

```
% cd <repo-path>
% git pull
```

②Mackerelとの差分を確認

```
% mkr monitors diff
```

③差分がない場合、次のような結果が得られる

```
Summary: 0 modify, 0 append, 0 remove
```

●ローカルファイルから監視ルールを変更

ローカルファイルから監視ルールを変更した場合は、次の手順でMackerelおよびリポジトリに反映します。

①GitHub上で変更・レビューなどを行い、masterブランチ(など)に反映
②変更された監視ルールをGitHubから取得

```
% cd <repo-path>
% git pull
```

注5) API仕様の「監視設定の登録」 URL http://help-ja.mackerel.io/entry/spec/api/v0#monitor-create

③Mackerelとの差分を確認

```
% mkr monitors diff
```

④変更された監視ルールをMackerelに反映

```
% mkr monitors push
```

●ローカルファイルで監視ルールを新規追加

ローカルファイルから監視ルールを新規追加するには次の手順に従います。

①ローカルファイルに新規監視ルールを追加したあと、GitHub上で変更・レビューなどを行い、masterブランチ(など)に反映
(ここでの監視ルールはidなしのものを作成)

②Mackerelとの差分を確認

```
% mkr monitors diff
```

③追加された監視ルールをMackerelに反映

```
% mkr monitors push
```

Nameベースで監視ルールの同一判定をするため、id反映のためのpullは不要となります。

●Mackerel側のルールを反映

WebUI側で変更してしまった場合は、次のフローでローカルファイルを最新化し、リポジトリに反映させます。

①変更された監視ルールをMackerelから取得

```
% cd <repo-path>
% mkr monitors pull
```

pullすることでid付きのJSONとなる

②JSONからidフィールドを削除
idフィールドが存在しても動作に支障はないので、放置しても実用上は問題なし

③Mackerelとの差分を確認

```
% mkr monitors diff
```

④GitHub上のリポジトリにコミット、プッシュ

```
% git add monitors.json
% git commit -m '<commit-msg>'
% git push
```

CIでテストする

ここまでで、Mackerelの監視ルールをGitHubで管理できるようになりました。ただし、この連携は手動操作が必要ですので、Mackerelの監視ルールとGitHub上のコードがずれる可能性があります。それを防ぐためにCI(継続的インテグレーション)で正しく同期されているか確認するようにしましょう。

mkr monitors diffコマンドには差分があった場合に終了コードを「1」にする--exit-codeオプションがありますので、これを利用します。

たとえば、次のようなRakefile[注6]を用意し、CIでrake[注6]を実行させることで差分が発生したことを通知できます。

```
task :default do
  sh "mkr monitors diff -e"
end
```

まとめ

Infrastructure as Codeのように、インフラ運用のための各種設定をコードで管理する考え方は徐々に広まりつつあります。Mackerelもその流れをふまえ、監視設定をコードで管理するための方法を提供開始しました。今後、そのほかのさまざまな設定もコードで管理できるように拡張していくことを検討しています。まずは監視設定をコードとしてGitHubなどで管理することをぜひお試しください。SD

注6) rakeはRuby製のビルドツールで、Rakefileに定義されたさまざまなタスクを実行させることができます。詳しくはGitHubリポジトリを参照してください。 URL https://github.com/ruby/rake

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17回 Identity Managementを使おう

2011年12月にリリースされたRed Hat Enterprise Linux 6.2から同梱されているIdentity Managementは認証に特化したID管理ソフトウェアです。認証という比較的地味な役割を担うためあまり脚光を浴びることはありませんが、Red Hat Enterprise Linuxにこのような機能も含まれている例として紹介します。

Author レッドハット(株)サービス事業統括本部
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) Twitter@rioriost

無償で利用可能?

Identity Management(IdM)はLDAPサーバとして389 Directory Server、認証やシングルサインオンにKerberosを用い、ユーザインターフェースを含むプロジェクト全体がOSSで開発されているID管理ソフトウェアで、FreeIPA[注1]がアップストリームになります。Red Hat Enterprise Linux(RHEL)に同梱されており、RHELのサブスクリプション契約があればCAL[注2]はもちろんのこと、追加の費用なしに利用することができます。

389 Directory Serverについて少し説明しておきましょう。Red Hatは2004年に米・America Onlineの一部門であった米・Netscape Security SolutionsからNetscape Directory Server(NDS)の知的財産権を買収し、翌2005年[注3]にRed Hat Directory Server(RHDS)およびRed Hat Certificate Systemという名称でOSS化、自社製品化しました。NDSはもとをたどれば米・ミシガン大学のslapdに起源をおくため、旧Sun Microsystemsの JES/SunOne Directory Server(現・Oracle Directory Server)とは兄弟関係にあります。またRHDSは米・HP社のHP-UXにも供給されており、UNIX/Linuxのディレクトリサーバとして広く使われています。さらに2009年にFedoraプロジェクトにおいては、Fedora Directory ServerはLDAPのポート番号・389/udp,tcpにちなんだ名称変更[注4]がなされ現在に至ります。

RHDSとの違い

上述したようにRHDSはRed Hat製品の中でも歴史が長く、現在の最新版であるバージョン10は2015年6月にリリースされ2020年6月までのサポートが決定しています。一方でRHDSのユーザインターフェースは古臭い、あるいは管理手順が現代的でない点は否定できません。LDAPを広範に利用するのであれば致し方ないのですが、認証だけを行う場合、RHDSの管理はLight Weight[注5]なのにちょっと「重すぎる」とも言えます。そこで登場したのがIdMで、Webブラウザによる一元管理を実現しているのはもちろん、インストールや設定もインタラクティブ[注6]に質問に答えていけば一通り行える実装になっています。またOTP(One

注1) IPAは、Identity、Policy、Auditの頭文字(http://freeipa.org/)。
注2) Client Access License
注3) 筆者がRed Hatに入社した2005年に製品化されたため、入社直後に実施されたトレーニングで使われたテキストは"Netscape Directory Server"のままで、ここかしこにNetscapeのロゴが表示されていた。
注4) ググラビリティ(検索のしやすさ)を劇的に低下させたので、筆者や周囲は否定的。
注5) LDAPはLight-weight Directory Access Protocol
注6) answerファイルを用意してバッチ的にインストールすることも可能。

Time Password)にも対応しており、昨今の認証基盤として必要とされる機能を一通り実現しています。

ただしインストールや設定が簡素になっている反面、RHDSとは異なりIdMは汎用のLDAPサーバではありませんし、Windowsの Active Directory(AD)と連携できるもののADの代替製品としてID管理を統合するものでもありません。メリット／デメリットをしっかりと理解して上手に利用しましょう。

Active Directoryとの相互運用

業務システムとしてOSにWindowsを採用している一定以上の規模の企業で、ADを利用していないということは考えにくく、実際、筆者自身も業務の中で「Linuxの認証基盤にもADを利用したい」という要望をいただくことがよくあります。UNIX/Linuxしかなければ今でもNISで認証しているということもあるのですが、「NISが古くなったので移行を検討している」という文脈の中で出てくることがほとんどです。

IdMではADからパスワードの変更を受け取り同期することや、Kerberosを用いたCross Realm Trustチェインによって ADが発行したチケットをもとにIdMでクレデンシャルを提供しLinux(UNIX)/Windowsをまたいだシングルサインオン(SSO)を実現できます。

パッケージをインストールする

IdMを試してみるにはインストールするのが一番です。本稿執筆時点ではRHELのバージョンは6.7あるいは7.1が最新なので7.1のx86_64版での手順を説明します。IdMのサイジングで最も重要なのはメモリ容量で、1万ユーザ・100グループであれば2GB、10万ユーザ・5万グループであれば16GBが必要最小限のメモリ容量となります。試してみる、という目的であれば2GB程度のメモリで十分に動作しますし、仮想化環境でも用意するのは難しくないでしょう。

まずはベースOSとなるRHEL 7.1を最小パッケージ構成でインストールします。IdMのFQDNが正引き・逆引きで問題なく設定されていることを確認します。またNetworkManagerによる動的なネットワーク構成変更がなされないようにsystemctlコマンド、あるいはservice/chkconfigコマンドでNetworkManagerを停止し、networkサービス[注7]が起動するように設定します。subscription-managerで登録後、適切なサブスクリプションとリポジトリ[注8]が紐付けされていれば、次のコマンドだけでIdMに関連するパッケージのインストールが完了します。

```
# yum -y install ipa-server bind bind-dyndb-ldap
```

初期設定はコマンド1発!

図1のようにパッケージインストールの終了

▼図1 IdMのインストール(ipa-server-installコマンドの実行)

```
# ipa-server-install

The log file for this installation can be found in /var/log/ipaserver-install.log
==============================================================================

=======
 This program will set up the IPA Server.

This includes:
  * Configure a stand-alone CA (dogtag) for certificate management
  * Configure the Network Time Daemon (ntpd)
  * Create and configure an instance of Directory Server
  * Create and configure a Kerberos Key Distribution Center (KDC)
  * Configure Apache (httpd)

To accept the default shown in brackets, press the Enter key.

Do you want to configure integrated DNS (BIND)? [no]:
```

注7) サービスといっても、実態としては単なるifup / ifdownコマンド。
注8) subscription-manager repos --enable=rhel-7-server-rpms

後、IdMのインストールはipa-server-install コマンドで開始します。

さらにホスト名やドメイン名[注9]などのネットワーク関連の情報、管理者のパスワードなどをインタラクティブに回答していくと、インストールの最終段で設定項目の確認を求められます。

▼図2　IdMのインストール(設定項目の入力と確認)

```
The IPA Master Server will be configured with:
Hostname:       idm.rio.st
IP address(es): 192.168.1.135
Domain name:    rio.st
Realm name:     RIO.ST

BIND DNS server will be configured to serve IPA domain with:

Forwarders:    192.168.1.2
Reverse zone(s):  1.168.192.in-addr.arpa.

Continue to configure the system with these values? [no]:
```

▼図3　IdMのインストール実行中の画面

```
The following operations may take some minutes to complete.
Please wait until the prompt is returned.

Configuring NTP daemon (ntpd)
  [1/4]: stopping ntpd
  [2/4]: writing configuration
  [3/4]: configuring ntpd to start on boot
  [4/4]: starting ntpd
Done configuring NTP daemon (ntpd).
Configuring directory server (dirsrv): Estimated time 1 minute
  [1/38]: creating directory server user
  [2/38]: creating directory server instance
  [3/38]: adding default schema
  [4/38]: enabling memberof plugin
  [5/38]: enabling winsync plugin
  [6/38]: configuring replication version plugin
 ..........
```

▼図4　IdMのインストールの完了

```
Setup complete

Next steps:
  1. You must make sure these network ports are open:
    TCP Ports:
      * 80, 443: HTTP/HTTPS
      * 389, 636: LDAP/LDAPS
      * 88, 464: kerberos
    UDP Ports:
      * 53: bind
      * 88, 464: kerberos
      * 53: bind
      * 123: ntp

  2. You can now obtain a kerberos ticket using the command:
'kinit admin'
     This ticket will allow you to use the IPA tools (e.g., ipa
user-add)
     and the web user interface.

Be sure to back up the CA certificate stored in /root/cacert.p12
This file is required to create replicas. The password for this
file is the Directory Manager password
```

▼図5　ファイアウォールの設定

```
# firewall-cmd --permanent --zone=public --add-
port={80/tcp,443/tcp,389/tcp,636/tcp,88/tcp,464/tcp,53/tcp,88/udp,464/udp,53/udp,123/udp}

# kinit admin
```

図2の一覧で問題がなければ、"yes"とタイプしEnterキーの入力でインストールが開始されます(図3)。

無事にインストールが完了すると、図4のメッセージが表示されます。

firewall-cmdで必要なポートを解放し、kinitコマンドで"admin"アカウントを初期化します。これで利用開始に必要な作業は完了です(図5)。

ユーザアカウントを追加する

WebブラウザでIdMサーバにアクセスし、ユーザアカウントを追加してみましょう。kinitで設定したadmin/パスワードでログインします(図6)。

ログインするとユーザ管理画面が表示されます。ユーザを追加するには[+Add]ボタンをクリックします(図7)。

ユーザのログインアカウント、氏名、パスワードを入力します(図8)。

"Test"ユーザが追加されました。画面右上の[Administrator]プルダウンからログアウトします(図9)。

ログアウト後、Webブラウザで再度IdMサーバにアクセスし、追加した"Test"ユーザでログインしてみましょう(図10)。

すると「パスワードがすでに失効している」旨の表示がされます(図11)。ユーザアカウント作成時に設定したパスワードは管理者(adminユーザ)が

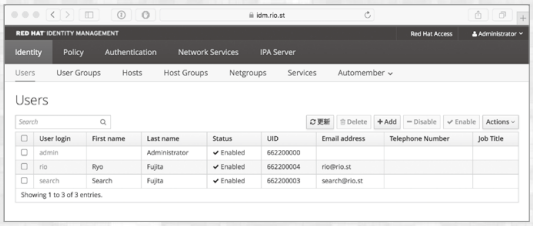
注9) IdMではADで用いるのと似た「ドメイン」という概念を用いる。

知っています。この初期作成パスワードは、ユーザアカウントが自身のパスワードを設定するためにだけ用いられるようになっています。

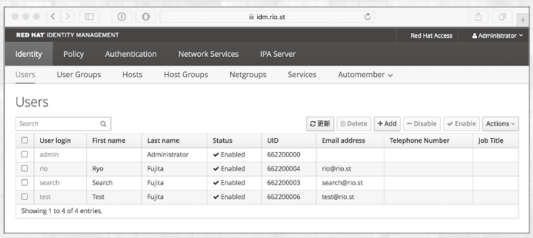
まとめ

前述したように、新規アカウント作成時のパスワードにもセキュリティを考慮した発行フローが採用されていることや、一方向性ハッシュによるパスワードの保存など、IdMはセキュリティ側に倒した実装がなされている点も非常に魅力的です。次回はIdMによるクライアント認証やWebサービスにおけるユーザアカウントの管理、RHEL 7.2におけるIdMとの認証統合の強化について紹介する予定です。SD

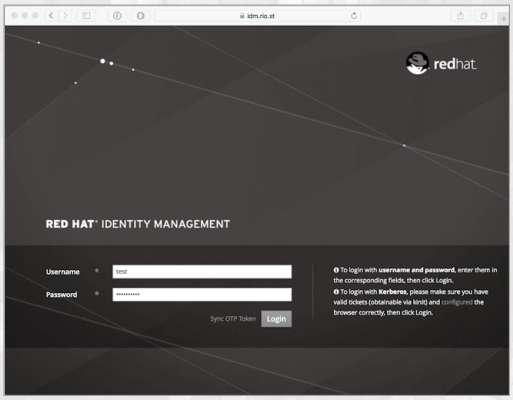
▼図7　IdMサーバのユーザ管理画面



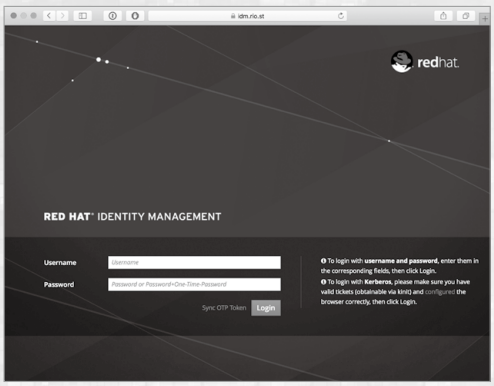
▼図9　Testユーザの追加完了画面



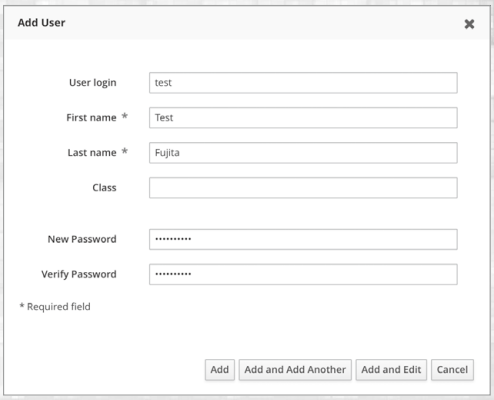
▼図10　Testユーザでログイン



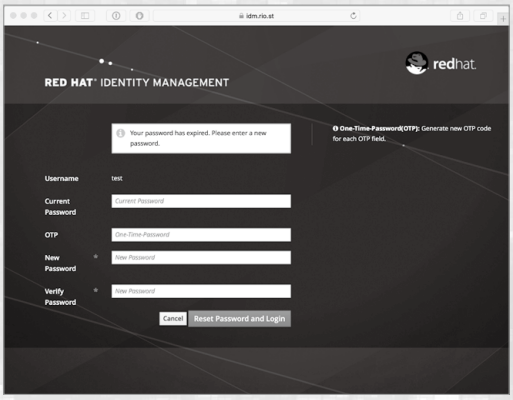
▼図6　IdMサーバにログイン



▼図8　ユーザアカウントの追加設定画面



▼図11　パスワード失効エラー（新規パスワード設定画面）



Be familiar with FreeBSD.

チャーリー・ルートからの手紙

第26回 ❖bhyveでOpenBSDファイアウォール on FreeBSDを構築

大人気のOpenBSDファイアウォール pf(8)

FreeBSDにはipfw(8)、pf(8)、ip lter(8)という3つのファイアウォール機能が用意されています。FreeBSDプロジェクトがネイティブに開発しているファイアウォールはipfw(8)です。この3つの中ではもっとも高速で安定しています。複雑なことを実施しようとするとルールがだいぶ難しくなりますが、性能と安定性、スケーラビリティの高さが魅力です。

pf(8)はOpenBSDプロジェクトで開発されているファイアウォールです。シンタックスがわかりやすく、便利なルール表記がしやすいことから人気があります。FreeBSDはOpenBSD pf(8)をインポートして使っています。ipfw(8)と比べて扱いやすい反面、スケーラビリティや安定性の面でipfw(8)にかなわないところがあります。

ipfilter(8)は現在では開発が停滞していますし、ipfw(8)やpf(8)と比べると存在感の薄いところがあります。今後のメンテナンスなども考えると、そう遠くない将来にはデフォルトの機能からはずれる可能性があります(問題が発生したときや本当にユーザが少なくなったときに、手を挙げる開発者がいなければ消える可能性が高いといえます)。実質的に、FreeBSDのファイアウォールといえばipfw(8)かpf(8)かということになるでしょう。

FreeBSD pf(8)の問題点といえば、常にOpenBSD pf(8)をマージし続けているわけではないということです。シンタックスが変わったときなどに大規模マージを実施して、あとは必要に応じてマージが行われています。つまり、最新のOpenBSD pf(8)と比べると、使える機能やシンタックスが常にちょっと古いということになります。

◉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。

餅は餅屋方式:OpenBSDファイアウォール・オン・FreeBSDハイパーバイザ

餅は餅屋ということになりますが、やはりpf(8)はOpenBSD版を使いたいところです。ただし、OpenBSDはマルチコアへのスケーラビリティがFreeBSDにかなわないところがあります。最近のpf(8)はだいぶ実装がスケールするようになりましたが、OpenBSDはカーネルそのもののスケーラビリティがFreeBSDのスケーラビリティに追いついていません。

現在のマシンはマルチコア／メニーコアが主流になりつつあるため、ハードウェアで直接動作するオペレーティングシステム(OS)には、高いスケーラビリティを発揮するFreeBSDを使いたいところです。ということであれば、ホストにはFreeBSDを使用し、その上で仮想化機能でありハイパーバイザであるbhyve(8)を使い、OpenBSD pf(8)を仮想環境で運用するというのが、1つの方法ということになります。

これはなかなか活用しがいのあるアイデアです。bhyve(8)登場以降は、このように単一のOSに機能をマージするのではなく、複数のOSの良いところを組み合わせたシステム構築が可能になりました。

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。

OpenBSDは興味深いOSですので、使ったことのない方はこの機会に使ってもらえればと思います。

ネットワークのセットアップ

OpenBSD on bhyveを実現するための設定を紹介します。まず、ゲスト向けのネットワーク環境を用意します。**図1**のように1つの物理NICが搭載されたマシンを使っているものとしましょう。em0がそれです。後でこのem0と、ゲスト環境に割り当てるネットワークインターフェースとを接続します。

仮想環境で動作するOpenBSD向けにtap0というソフトウェア的なトンネルネットワークインターフェースを用意します(**図2**)。OpenBSDはこれを経由してネットワークにアクセスできます。tap0をオープンしたときに自動的にステータスがUPに切り替わるように、**図3**のようにsysctl(8)の値を変更しておきます。

em0とtap0を結び付けるために、bridge0というネットワークブリッジデバイスを作成します(**図4**)。次に、em0とtap0を、作成したbridge0に追加します(**図5**)。これでem0とtap0がリンクします。

コマンドで操作した内容はシステムを再起動するとクリアされますので、**リスト1、2**のように/etc/rc.confと/etc/sysctl.confに設定を書いておきます。

▼**図1　マシンのNIC一覧。lo0はループバックインターフェース**

```
# ifconfig | grep mtu ↵
em0: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> metric 0 mtu 1500
lo0: flags=8049<UP,LOOPBACK,RUNNING,MULTICAST> metric 0 mtu 16384
```

▼**図2　ゲストOS向けのネットワークインターフェースtap0を作成**

```
# ifconfig tap0 create ↵
# ifconfig tap0 ↵
tap0: flags=8802<BROADCAST,SIMPLEX,MULTICAST> metric 0 mtu 1500
        options=80000<LINKSTATE>
        ether 00:bd:f9:74:00:00
        nd6 options=29<PERFORMNUD,IFDISABLED,AUTO_LINKLOCAL>
        media: Ethernet autoselect
        status: no carrier
```

▼**図3　tap0オープン時に自動的に状態がUPになるように設定を変更**

```
# sysctl net.link.tap.up_on_open=1 ↵
net.link.tap.up_on_open: 0 -> 1
```

▼**図4　ネットワークブリッジデバイスbridge0を作成**

```
# ifconfig bridge0 create ↵
# ifconfig bridge0 ↵
bridge0: flags=8802<BROADCAST,SIMPLEX,MULTICAST> metric 0 mtu 1500
        ether 02:11:20:51:96:00
        nd6 options=9<PERFORMNUD,IFDISABLED>
        id 00:00:00:00:00:00 priority 32768 hellotime 2 fwddelay 15
        maxage 20 holdcnt 6 proto rstp maxaddr 2000 timeout 1200
        root id 00:00:00:00:00:00 priority 0 ifcost 0 port 0
```

▼**図5　ブリッジにem0とtap0を追加**

```
# ifconfig bridge0 addm em0 addm tap0 up ↵
# ifconfig bridge0 ↵
bridge0: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> metric 0 mtu 1500
        ether 02:11:20:51:96:00
        nd6 options=9<PERFORMNUD,IFDISABLED>
        id 00:00:00:00:00:00 priority 32768 hellotime 2 fwddelay 15
        maxage 20 holdcnt 6 proto rstp maxaddr 2000 timeout 1200
        root id 00:00:00:00:00:00 priority 32768 ifcost 0 port 0
        member: tap0 flags=143<LEARNING,DISCOVER,AUTOEDGE,AUTOPTP>
                ifmaxaddr 0 port 3 priority 128 path cost 2000000
        member: em0 flags=143<LEARNING,DISCOVER,AUTOEDGE,AUTOPTP>
                ifmaxaddr 0 port 1 priority 128 path cost 20000
```

▼**リスト1　/etc/rc.confに追加する設定**

```
cloned_interfaces="bridge0 tap0"
ifconfig_bridge0="addm em0 addm tap0"
```

▼**リスト2　/etc/sysctl.confに追加する設定**

```
net.link.tap.up_on_open=1
```

これでシステムを再起動しても同じ設定になります。

仮想化機能のセットアップ

仮想化の機能を有効にする方法は、vmmカーネルモジュールを読み込むだけです。

```
# kldload vmm
```

この設定も再起動すると無効になりますので、/boot/loader.confに次の設定を追加して、再起動しても自動的にvmmカーネルモジュールが読み込まれるようにしておきます。

```
vmm_load="YES"
```

実際に運用する場合には、この段階で何度か再起動して、再起動後に同じ設定になるか確認しておきましょう。

仮想ディスクのセットアップ

次に、ゲストの仮想ディスクをセットアップします。bhyveの仮想ディスクはZFSの機能を使って用意するのがよいでしょう。使い勝手がよく、性能も期待できるからです。**図6**のようにzfs(1)コマンドを実行すると32GBの仮想ディスクを作成できます。ここではzというプールを使っているので、z/openbsd-5.7という仮想ディスクを作っています。コマンド実行後に/dev/zvol/z/openbsd-5.7というデバイスファイルが生えてきますので、これを仮想ディスクとして使用します。

ファイルを仮想ディスクとして扱うこともできます。その場合には、**図7**のようにして仮想ディスクファイルを作成します。

ただし、ファイルを仮想ディスクとして扱うと、ホスト側のファイルシステムスタックの処理も通ることになるため、zfs(8)でボリュームを作成した場合と比べると処理が遅くなります。イメージとしては眼鏡・オン・眼鏡というか、ファイルシステム・オン・ファイルシステムになるので、無駄な処理が増えるからです。bhyve(8)を使う場合にはZFSの活用を前提にしておきましょう。

カーネルローダのセットアップ

bhyveで仮想環境を起動する手順は次のステップになります。

1　ゲストOSのカーネルを読み込み
2　仮想環境を起動

FreeBSDをゲストOSとして利用する場合、bhyveload(8)というコマンドでFreeBSDカーネルを読み込みます。FreeBSD以外のカーネルを読み込む場合はgrub2-bhyveというコマンドを使います。grub2-bhyveはベースシステムにはマージされていませんので、pkg(8)経由でインストールします(**図8**)。

▼**図6　zfs(8)で仮想ディスクを作成**

```
# zfs create -V 32G -o volmode=dev z/openbsd-5.7 ↵
# ls -l /dev/zvol/z/ ↵
total 0
crw-r-----  1 root  operator  0x7e Oct  5 16:46 openbsd-5.7
```

▼**図7　ファイルを仮想ディスクにする場合**

```
# truncate -s 32G openbsd-5.7.img
```

▼**図8　grub2-bhyveをインストール**

```
# pkg install grub2-bhyve ↵
Updating FreeBSD repository catalogue...
FreeBSD repository is up-to-date.
All repositories are up-to-date.
The following 1 package(s) will be affected (of 0 checked):

New packages to be INSTALLED:
        grub2-bhyve: 0.40

The process will require 1 MiB more space.
408 KiB to be downloaded.

Proceed with this action? [y/N]: y
Fetching grub2-bhyve-0.40.txz: 100%  408 KiB 417.3kB/s    00:01
Checking integrity... done (0 conflicting)
[1/1] Installing grub2-bhyve-0.40...
[1/1] Extracting grub2-bhyve-0.40: 100%
```

grub2-bhyveを使ってカーネルを読み込む場合、仮想ディスクやCD/DVDのISOファイルを設定ファイルで指定しておく必要があります。ここではOpenBSD 5.7をインストールする予定なので、**リスト3**のようにgrub2-bhyveで使用する設定ファイルを作成しておきます。パスはそれぞれの環境に読み替えて変更してください。

▼**リスト3　grub2-bhyveの設定ファイル openbsd-5.7-device.map**

```
(hd0) /dev/zvol/z/openbsd-5.7
(cd0) /bhyve/install57.iso
```

OpenBSD 5.7インストール

執筆現在、OpenBSDの最新リリース版はOpenBSD 5.7です。本誌が出版された頃にはOpenBSD 5.8がリリースされていると思います(OpenBSDのリリースエンジニアリングは厳密で、ほとんど遅延なく計画どおりにリリースされます)。試すときはOpenBSD 5.8以降の最新版を使うとよいと思います。

fetch(1)コマンドなどでインストールISOイメージファイルを取得します(**図9**)。

図10のようにgrub-bhyveコマンドを実行して、OpenBSDカーネルを読み込みます。途中の操作は含まれているファイルを調べているだけで、最後に実行している「kopenbsd -h com0 (cd0)/5.7/amd64/bsd.rd」と「boot」が重要です。

ここではシングルコアを割り当ててOpenBSD 5.7を動作させることを想定しています。**図11**のようにbhyveコマンドを実行すると、OpenBSDインストーラが起動してきます。

OpenBSDのインストールはシンプルです。ときどき入力を求められますが、とくに変更する必要がなければそのままリターンキーを押せばよいでしょう。入力する必要がある部分だけ最低限入力するだけでインストールは完了します。インストール後はshutdown -p nowを実行するなどして、いったん仮想環境を終了してください。

OpenBSD 5.7を運用しよう

先ほどはインストーラを起動するための操作でし

▼**図9　OpenBSD 5.7のインストールISOイメージファイルを取得**

```
# fetch ftp://ftp.kddilabs.jp/OpenBSD/5.7/amd64/install57.iso ↵
install57.iso                                 100% of  218 MB 4525 kBps 00m49s
```

▼**図10　grub2-bhyveを使ったOpenBSDカーネルの読み込み**

```
# grub-bhyve -m openbsd-5.7-device.map -r cd0 -M 2G openbsd-5.7 ↵

                            GNU GRUB  version 2.00

   Minimal BASH-like line editing is supported. For the first word, TAB
   lists possible command completions. Anywhere else TAB lists possible
   device or file completions.

grub> ls (cd0)
Device cd0: Filesystem type iso9660 - Label \\`OpenBSD/amd64   5.7 Install CD' -
Last modification time 2015-03-06 01:10:10 Friday, UUID 2015-03-08-11-10-10-00
- Total size 448444 sectors
grub> ls (cd0)/
5.7/ etc/ TRANS.TBL
grub> ls (cd0)/5.7
amd64/ TRANS.TBL
grub> ls (cd0)/5.7/amd64/
base57.tgz boot.catalog bsd bsd.mp bsd.rd cdboot cdbr comp57.tgz game57.tgz INS
TALL.amd64 man57.tgz SHA256 TRANS.TBL xbase57.tgz xfont57.tgz xserv57.tgz xshar
e57.tgz
grub> kopenbsd -h com0 (cd0)/5.7/amd64/bsd.rd
grub> boot
```

たので、インストールしたOpenBSDを起動する操作はまたちょっと違ってきます。まず、grub2-bhyveの実行は**図12**のように、CD/DVDからではなく仮想ディスクからになります。仮想環境の起動も**図13**のようにちょっと引数が変わります。

grub2-bhyveとbhyve(8)コマンドを毎回このように入力するのは面倒です。シェルスクリプトにすることもできますが、**リスト4**のようにエイリアスを作成するだけでもコマンド一発で起動できるようになります。

典型的な使い方としては、FreeBSDホストにssh(1)を使ってログインし、そこでbhyve経由でOpenBSDを利用するといったことになるでしょう。ここにさらにtmux(1)やscreen(1)といったターミナルマルチプレクサを組み合わせれば、遠隔からコンソール経由でログインできる仮想環境のできあがりということになります。もちろん、コンソールを使わずにssh(1)経由のみのログイン環境を仕立てることもできます。

WindowsやIllumosもサポート

これまでbhyve(8)で動作するOSはFreeBSD、NetBSD、OpenBSD、Linuxあたりでしたが、2015年10月に実施されたコミットでWindows(ヘッドレスモード)やIllumosも動作するようになりました。今後さらに動作するOSの数は増えることになると思います。

これは体感ですが、bhyveはほかの同様の実装と比べて動作が軽快な印象を受けます。最初はコマンドやオプションがややこしく思うかもしれませんが、それぞれが意味するところがわかってくると、これはなかなか扱いやすい機能です。まだ使ったことがないのであれば、一度使ってみてください。

今回はpf(4)まで書ききれませんでした。次回にご期待ください。SD

▼図11　OpenBSD 5.7インストーラの起動

```
# bhyve \
    -H -P -A \
    -W -c 1 \
    -m 2G \
    -l com1,stdio \
    -s 0:0,hostbridge \
    -s 1:0,lpc \
    -s 2:0,virtio-net,tap0 \
    -s 3:0,ahci-cd,/home/bhyve/install57.iso \
    -s 4:0,virtio-blk,/dev/zvol/z/openbsd-5.7 \
    openbsd-5.7 ↵
```

▼図13　OpenBSD 5.7仮想環境を起動

```
# bhyve \
    -H -P -A \
    -W -c 1 \
    -m 2G \
    -l com1,stdio \
    -s 0:0,hostbridge \
    -s 1:0,lpc \
    -s 2:0,virtio-net,tap0 \
    -s 3:0,virtio-blk,/dev/zvol/z/openbsd-5.7 \
    openbsd-5.7 ↵
```

▼図12　仮想ディスクからOpenBSDカーネルを読み込み

```
# grub-bhyve -m openbsd-5.7-device.map -M 2G openbsd-5.7 ↵

                            GNU GRUB  version 2.00

   Minimal BASH-like line editing is supported. For the first word, TAB
   lists possible command completions. Anywhere else TAB lists possible
   device or file completions.

grub> kopenbsd -h com0 -r sd0a (hd0,openbsd1)/bsd
grub> boot
```

▼リスト4　一連の起動作業をエイリアスに設定

```
alias bhyve_openbsd_5.7='sudo bhyvectl --destroy --vm=openbsd-5.7; printf "kopenbsd -h com0 -r sd0a ↵
(hd0,openbsd1)/bsd\nboot\n" | sudo grub-bhyve -m /d/bhyve/openbsd-5.7-device.map -M 2G openbsd-5.7; ↵
sudo bhyve -W -c 1 -m 2G -H -P -A -l com1,stdio -s 0:0,hostbridge -s 1:0,lpc -s 2:0,virtio-net,tap6 ↵
-s 3,virtio-blk,/dev/zvol/z/openbsd-5.7 openbsd-5.7; sudo bhyvectl --destroy --vm=openbsd-5.7'
```

バックナンバーのお知らせ　BACKNUMBER

バックナンバー好評発売中！常備取り扱い店もしくはWebより購入いただけます

本誌バックナンバー（紙版）はお近くの書店に注文されるか、下記のバックナンバー常備取り扱い店にてお求めいただけます。また、「Fujisan.co.jp」（http://www.fujisan.co.jp/sd）や、e-hon（http://www.e-hon.ne.jp）にて、Webから注文し、ご自宅やお近くの書店へお届けするサービスもございます。

2015年11月号

第1特集
すいすいわかるHTTP/2
HTTP/1.1から変わること・変わらないこと

第2特集
攻撃を最前線で防ぐ
ファイアウォールの教科書

特別企画
・SMB実装をめぐる冒険　File System for Windowsの作り方（前編）

定価（本体1,220円+税）

2015年10月号

第1特集
多層防御や感染後対策を汎用サーバに実装
攻撃に強いネットワークの作り方

第2特集
Webメールの教科書
クラウドサービス利用か？　自社で構築か？

特別付録
・創刊300号記念　Vim&Emacsチートシート

定価（本体1,220円+税）

2015年9月号

第1特集
特講
正規表現・SQL・オブジェクト指向
苦手克服のベストプラクティス

第2特集
メールシステムの教科書
日本語もバイナリもちゃんと届くのはなぜか

特別企画
・なぜ俺の提案は通らないのか？

定価（本体1,220円+税）

2015年8月号

第1特集
Lispより始めよ、されば救われん！
なぜ関数型プログラミングは難しいのか？

第2特集
安全な通信を確保する
SSL/TLSの教科書

短期連載
・AWSで始めよう！　モダンなJavaアプリケーション開発

定価（本体1,220円+税）

2015年7月号

第1特集
あなたにもできる！
ログを読む技術［セキュリティ編］

第2特集
黒い画面（tmux）の使い方
プロになるためのターミナル活用術

第3特集
6人の先駆者に訊く
スペシャリストになる方法

定価（本体1,220円+税）

2015年6月号

第1特集
新人さん歓迎特集
Git&GitHub入門

第2特集
OpenLDAPの教科書
ユーザ／ネットワーク管理の基本と活用例

一般記事
・Sambaによる Active Directoryの機能性と移行性を検証する

定価（本体1,220円+税）

Software Design　バックナンバー常備取り扱い店

北海道	札幌市中央区	MARUZEN & ジュンク堂書店　札幌店	011-223-1911	神奈川県	川崎市高津区	文教堂書店　溝の口本店	044-812-0063
	札幌市中央区	紀伊國屋書店　札幌本店	011-231-2131	静岡県	静岡市葵区	戸田書店　静岡本店	054-205-6111
東京都	豊島区	ジュンク堂書店　池袋本店	03-5956-6111	愛知県	名古屋市中区	三洋堂書店　上前津店	052-251-8334
	新宿区	紀伊國屋書店　新宿本店	03-3354-0131	大阪府	大阪市北区	ジュンク堂書店　大阪本店	06-4799-1090
	渋谷区	紀伊國屋書店　新宿南店	03-5361-3315	兵庫県	神戸市中央区	ジュンク堂書店　三宮店	078-392-1001
	千代田区	書泉ブックタワー	03-5296-0051	広島県	広島市南区	ジュンク堂書店　広島駅前店	082-568-3000
	千代田区	丸善　丸の内本店	03-5288-8881		広島市中区	丸善　広島店	082-504-6210
	中央区	八重洲ブックセンター本店	03-3281-1811	福岡県	福岡市中央区	ジュンク堂書店　福岡店	092-738-3322
	渋谷区	MARUZEN & ジュンク堂書店　渋谷店	03-5456-2111				

※店舗によってバックナンバー取り扱い期間などが異なります。在庫などは各書店にご確認ください。

デジタル版のお知らせ　DIGITAL

デジタル版Software Design 好評発売中！紙版で在庫切れのバックナンバーも購入できます

本誌デジタル版は「Fujisan.co.jp」（http://www.fujisan.co.jp/sd）と、「雑誌オンライン.com」（http://www.zasshi-online.com/）で購入できます。最新号、バックナンバーとも1冊のみの購入はもちろん、定期購読も対応。1年間定期購読なら5%強の割引になります。デジタル版はPCのほかにiPad／iPhoneにも対応し、購入するとどちらでも追加料金を払うことなく、読むことができます。

30

Debian Developer　やまねひでき　henrich@debian.org

DebConf15レポート（前編）と最新トピック

Debian Hot Topics

DebConf15開催

Debianの開発者ミーティング「DebConf15」が2015年8月15～22日の間、ドイツ・ハイデルベルクのユースホステルを会場として開催されました。ハイデルベルクはフランクフルト国際空港から電車で約1時間程度の場所にあり、古城が有名なドイツでも十指に入る人気の観光地です（**写真1**）。

今回のDebConfは盛況で、参加者は52ヵ国から過去最大の555名を数えました（**写真2、3**）。特徴的だったのは、この参加者のうちの28名は「kids」、つまり子供だということです。すでに16回を数えるカンファレンスということもあり[注1]、参加者も子持ち世代が増えているという証左ですね。ここから次世代のDebian開発者が生まれてくるかもしれません。

ほとんどのセッションが録画されていますので、興味のある方はビデオアーカイブ[注2]を参照するか、あるいはYouTubeで「DebConf15」で検索してみてください。

今回はそのセッションの中からいくつかを取り上げて、報告してみようと思います。

▼写真1　ハイデルベルクの古城

▼写真2　セッションの様子

▼写真3　多数の参加者でにぎわう会場

注1） 初回のDebconfは「DebConf0」と、0からスタートしています。

注2） URL http://debconf15.debconf.org/videostream.xhtml

PPA "bikeshed"

数年前からDebianにおいても、Launchpad[注3]の「PPA(Personal Package Archive)」のようなしくみについての提案はされていましたが、ここに来て本格的に導入を進めよう、というセッション「PPAs - what's next?」が開かれました。

PPAは、ディストリビューションのリポジトリとは別に、個人／チーム／サードパーティによるパッケージリポジトリを提供する機能です[注4]。公式のリポジトリでは足りない部分を独自に補ったり、実験的なパッケージ群を公式リポジトリに影響を与えることなく提供したりする場合などに使うこともできます。

サードパーティ側が複数のバージョンやアーキテクチャに対してのビルド環境を整えたり、パッケージリポジトリを用意したりするのは、結構骨が折れる作業です。そのため、ディストリビューションがこのような機能を提供してくれるのは、自前のリポジトリを作成して賄(まかな)ってきた開発者やそれを利用したいユーザにとってもありがたいことです。

今回のDebianにおける提案とLaunchpadのPPAとの違いは、Launchpadでは登録ユーザがPPAを持つことができるが、Debianの場合はDebian Developerにのみ提供されるという点です。この違いは次の2点から生じています。

①プロジェクトが提供できるビルドサーバなどのインフラ面でのリソース量に差がある
②個別にリポジトリを用意すれば配布に際して法的なリスクを負うことになるが、Debianではそこまで対応できない

①についてですが、Launchpadの場合は企業がバックについて積極的に支援しているのに比べ、Debianのビルドサーバ群は基本的に寄付／寄贈に頼っていたり、フルタイム勤務のシステム管理者が不在であったりします。この点を改善するのは難しいということがあります。

②について、Debianの公式リポジトリに入るパッケージに関しては、パッケージメンテナとftpmasterと呼ばれるリポジトリ管理者がライセンスの精査を行ってから配布を開始するなどの手順を経ています。しかし、PPAではこのような過程を省略しているため、問題のあるパッケージが「混入してしまう可能性」が発生します。

筆者がLaunchpadで実際に見た例としては、ライセンスに無頓着なユーザが再配布禁止のOracle JDKを配布していたことがありました。このような問題のあるリポジトリは、そのまま放置すると訴訟などの事態になりかねないため、管理者側で該当のファイル削除や、ユーザへの警告やヒアリング、場合によってはアカウント停止措置やその案内などの対応が必要となります。結果として、貴重な人的リソースが消費されてしまいます。この点もDebianプロジェクトとしてはさらなるリソースを追加するのは難しいと思われます。

それでもPPAが利用できるようになれば、たとえばGNOMEなどの巨大なパッケージ群について、更新を安定版に対して提供するなどさまざまな試みが行えるようになるはずで、期待が膨らみます。

なお、セッション中に「このPPAをなんと呼ぶか？」という問いが出たのですが、これに対して「bikeshed」(＝自転車置き場)というジョークが出て、そのまま名付けられてしまいました。「一時的にパッケージを置いておく場所」という意味合いだと思うのですが、bikeshedにはもう1つ「誰もが参加できるようなどうでもいい議論」という意味もあります。そのため、ネーミングについての議論も起きています……が、「これこそまさにbikeshedだ」と思ったのは筆者だけではないはずです。

注3) Canonical社が開発／管理しているホスティングサービス。Ubuntuの開発はLaunchpadを中心に進められている。
注4) openSUSEにおける「OBS(Open Build Service)」やFedoraにおける「Copr」なども、同様のしくみとして挙げられます。

local-apt-repository パッケージ

リポジトリの話が出たので、このあたりの話をもう少し。JenkinsやJenkins-debian-glueを使って継続的デリバリを行っているという「Continuous Delivery of Debian packages」というセッション[注5]の最中に、自前のリポジトリを簡単に作成できるツールとして「local-apt-repository」パッケージが紹介されるとともに、公式パッケージ化の話がでました[注6]。

これは、インストール後に/srv/local-apt-repositoryディレクトリを作って、そこにdebファイルを置くだけでaptからパッケージをインストールできるようになるというツールです。これまでのapt-ftparchivesコマンドやrepreproを使ったリポジトリの作成と比べても容易にリポジトリを作成できます。

Microsoft AzureとDebian

カンファレンスの期間中、「Debian is not welcome on Microsoft Azure」というちょっと刺激的な投稿がdebian-develメーリングリストにありました[注7]。「Microsoft AzureのSLA(Service Level Agreement：サービス品質保証)は、"Non-Endorsed Distribution"(非推奨ディストリビューション)であるDebianには適用されないので、クライアントがAzure上でDebianの利用を許してくれない」という不満です。AzureでSLAの対象になるのは企業体がバックにつき、Microsoft社と協力して問題のエスカレーションが行えるディストリビューションだけとなっており、Debianの場合はこのようなリソースがないことがネックとなっています。

これに対し、今回のカンファレンスのスポンサーの1つであるcredativ社は、AzureでのCentOSイメージにおけるOpenLogic社同様に、credativ社がDebianイメージを提供する企業となって、Debian を Azure の"Endorsed Distribution"(推奨ディストリビューション)にする作業を進めていることを表明しました[注8]。これが進めば、ユーザはAzureのギャラリーからDebianが簡単に利用可能になるのと同時に、前述のSLAの問題も解決します。

また、credativ社に雇用されているDebian開発者らが中心になって、AzureについてのBoFセッションを開き、現状の確認と懸念点などを話しあいました。筆者もこのセッションに参加し、外部へのトラフィックを削減するために、Ubuntu同様にAzure上にリポジトリミラーを置く予定であることを確認しています。今後の動きに注目しましょう。

「公式」イメージにまつわる話

先ほどのAzureなどにも関連してきますが、クラウドになると「クラウドベンダにとっての公式」なのか、それとも「イメージを作成している団体にとっての公式」なのか、それとも「その両方の意味での公式」なのか、何をもって「公式なDebianのイメージと呼ぶのか」という問題が持ち上がります。たとえばDockerのイメージがあったとして、「Docker側が公式サイトで配布する」イメージと「Debianプロジェクトが公式に作成／配布する」イメージはまったくの別物です。ですが、多くの人はその点を省略して「公式イメージ」と呼ぶ傾向にあります。

たいていの人が興味を持ちづらく、技術的にはどうでも良さそうなことですが、法的には商標などのからみもあり、プロジェクトとしてもこのあたりの基準をはっきりとさせておく必要があります。そのため、「What should be allowed to call itself "Debian"?」というセッションが開かれました。結論らしい結論は出なかったものの、この問題に取り組もうという意志は共有できたもの

注5) URL http://annex.debconf.org/debconf-share/debconf15/slides/286-continuous-delivery-of-debian-packages.pdf
注6) URL https://lists.debian.org/debian-devel/2015/08/msg00370.html
注7) URL https://lists.debian.org/debian-devel/2015/08/msg00227.html
注8) URL http://www.credativ.co.uk/credativ-blog/debian-images-microsoft-azure

と思います。

まだまだDebConfについての話題はつきませんが、今回はこれで終わりにします。続いて、カンファレンス以外のネタもピックアップしてみましょう。

ここ最近の話題

GitLabパッケージ

Gitを使うためのホスティングツールというと、GitHubがすぐに思い浮かびます。ただ、企業などでは「このようなツールを完全に組織内部だけでホスティングしたい(オンプレミスで運用したい)」という要求がよく聞かれます。このような利用者からの要望に対して、GitHub社はGitHub Enterpriseという製品を提供していますが、お値段も張るので気軽に導入、とはいきづらいのが実情でしょう。そのような組織が代わりに利用しているのが「GitLab」です。

GitLabはGitHubとは違い[注9]、ソースコードがMITライセンスで提供されているOSSですので、無償で自由に利用できます(GitLabと組織内のディレクトリサービスとの統合など、高度な機能については有償版を利用することになります)。

Debian上で利用するには、すでにGitLab社が提供するCommunity Editionのオムニバスインストーラパッケージがあります[注10]。ただ、GitLabが利用するNginx、PostgreSQL、Redisなどのソフトウェアがすべて一括で入るため300MB程度と比較的大きいサイズです。さらに、頻繁に更新されるのにもかかわらず、リポジトリ側の応答が悪く、筆者の環境でもたまにダウンロードに失敗するなどの事態が起きています。

これに対し、Debianの公式パッケージとして配布ができるようにパッケージングしよう、という動きが出ています[注11]。ただし、依存するパッケージが280個近くあり、さらに、そのうちの20個程度がまだ公式パッケージになっていない状況ですので、道のりは長そうです(公式リポジトリ入りしたとしても、今度は全部入りのオムニバスパッケージでは発生しない、コンポーネントの組み合わせバージョン間での問題が発生する可能性があります[注12]が、それはまた別の問題……)。

Debianプロジェクトにはすでにホスティングサービスとして Aliothがありますが、そのWebインターフェースは使い勝手が良いものとは言えず、単なるリポジトリとして使われています。GitLabのようなモダンなインターフェースを持ったツールがプロジェクトのサービスとして使えるようになると、状況も多少変わってくるかと思いますので、今後に注目です。

GNOME 3.18リリース

9月23日にGNOME 3.18がリリースされ、Debianでの対応も順調に進んでいます。パッケージの進捗状況はWebで確認できます[注13]。

今回から筆者もパッケージチームに加わって作業をしてみていますが、Debian GNOMEチームのリポジトリがSubversionを利用しており、Gitを使っているupstreamとのすり合わせ作業が面倒でたまりません。また、40,000コミット以上もされていてリポジトリ構成も独特なために、簡単にGitへの移行を完了……といかないのが悩ましいところです。SD

注9) よく勘違いされますが、GitHub自体のソースコードは公開されていないプロプライエタリなものです。そのため、GitHub上のグループとしてDebianを作って活用しようという意見が出ても「プロプライエタリサービスに依存するのはどうなのか」という反対意見が出て立ち消えになる、というのがしばしば見られます。

注10) URL https://about.gitlab.com/downloads/

注11) URL http://balasankarc.in/gitlab/

注12) オムニバスパッケージは「すべてのパッケージのバージョンを微調整して1個のパッケージにしている」のに対し、公式パッケージ群は「個々のパッケージが最新版を目指して更新されている」という違いがあります。そのため、ある特定のバージョン間では問題が発生しない場合でも、最新版同士だと問題がある、というような状況が発生する可能性があるのです。

注13) URL https://www.0d.be/debian/debian-gnome-3.18-status.html 「3.18」の部分をほかのバージョンに変更すれば今後も確認が可能です。

UbuntuとSkylake

Ubuntu Japanese Team
㈱創夢　柴田 充也（しばた みつや）　mail：mty.shibata@gmail.com

今年8月に、Intelの新世代CPU「Skylake」の発売が開始されました。そこで今回は、実際にSkylakeでUbuntuを動かしてみた結果や動作確認方法を紹介します。

Intelの新CPU「Skylake」

Skylakeは新しいCPUというだけでなく、ソケット形状が変更になり、チップセットの更新、バス帯域の強化、DDR4メモリのサポートなど、プラットフォーム全体が進化しています。さらに、Windows 10のリリースも重なったこともあって、自作PC向けに多種多様なパーツも一気に発売されました。

せっかく新しいプラットフォームが出たのだから、自作してUbuntuも動かしてみたい、と考えている方もいることでしょう。さらに本誌が発売されるころにはSkylake採用のデスクトップやノートPCも発売開始しているでしょう。でも新しいがゆえに、そもそもUbuntuが動くかどうかわからない、どうやって調べればいいかわからないと不安に感じるかもしれません。

SkylakeでUbuntuは動くのか

今回はテスト用に次のシステムを利用しました[注1]。CPUは「Skylake-K」ではなく通常版です。またマザーボードには、比較的新しいNICとしてIntel I219-Vが載っていることも特徴です。今回は、Ubuntuのインストールに光学ドライブは使用しませんでした。

- **CPU**：Skylake i7-6700
- **メモリー**：Crucial DDR4-2400 8GB×2
- **マザーボード**：ASRock Z170 Extreme4
- **ストレージ**：Crucial MX200 250GB SATA M.2 Type

結論から先に言うと、Ubuntu 14.04.3 LTS、15.04、執筆時点で開発中の15.10のいずれもSkylake上で動作しました。実はNICについては4.1以降のカーネルでないと動かないのですが、Ubuntuの場合15.04カーネルにもサポートコードがバックポートされていたため、リリース間の違いはほとんどありませんでした。ただしそのままでは3Dアクセラレーションが動作せず、後述するようにカーネルの起動オプションを変更する必要がありました。ここからは本誌が発売されるころにはリリースされている15.10を対象に説明します。

新しいプラットフォームでUbuntuが動くかどうかの鍵を握っているのはLinuxカーネルです。Linuxカーネルがサポートしていたらほぼ問題なく動くでしょうし、サポートしていなかったらそもそも動かないか、動いたとしても十分にパフォーマンスを出せません。IntelのCPUの場合は内蔵GPUがあるため、カーネルのサポートに加えて、X.orgのドライバがサポートしているかどうかも重要になります。

注1）残念ながら家族用のPCです。Windowsをインストールする前のハードウェアテスト用に特別にUbuntuをインストールさせてもらいました。

X.orgが十分にサポートしていたら3Dアクセラレーションが有効になり、デスクトップとしての用途も問題ないでしょう。ただし、X.orgのサポートがなくても汎用ドライバで動作する可能性は高いため、「まったく動かない」ということはほぼありません。少なくともXubuntuなどの2Dアクセラレーションが動けばそこそこ高速なデスクトップ環境を使えば、とくに困ることはないはずです。

Ubuntuの場合、6ヵ月の開発期間のおおよそ半ばあたりにリリースされたカーネルを採用します。2014年4月にリリースされた14.04であれば2014年1月にリリースされた3.13カーネルですし、2015年4月リリースの15.04は2015年2月リリースの3.19、2015年10月リリース予定の15.10は2015年8月リリースの4.2カーネルを採用といった具合です。当然のことながら、より新しいUbuntuの方が、新しいデバイスが動く可能性は高くなります。ただしほぼ2ヵ月弱ごとにリリースされるLinuxカーネルからみると、Ubuntuのカーネルは少し古いカーネルです。最新版ではサポート済みのデバイスだけれども、Ubuntuでは動かないこともよくあります。

言い方を変えると、IntelのようなLinuxカーネルへ積極的にコミットしているベンダの新しいプラットフォームについては、それが発売された時期を起点に「次の次」ぐらいのリリースのUbuntuだとまともに動く可能性が高い、ということになります。8月に発売したSkylakeプラットフォームだと「次(15.10)の次」ですので2016年4月リリース予定のUbuntu 16.04 LTSが狙い目ですね。

ちなみにLTSのポイントリリースである14.04.3はカーネルのみ15.04相当、つまり3.19カーネルになります。そのため、14.04.3と15.04では、カーネル(や同様にバージョンが更新されるX.org)に関するハードウェアサポート状況はほとんど変わりありません。また2016年1月ごろにリリースを予定している14.04.4では、15.10の4.2カーネルとなります。

デバイスは認識されるのか

前述のとおり、現在サポートされているバージョンのUbuntuであれば、Skylakeで構築されたシステム上で動作します。では、具体的にどのように動作確認すればよいのでしょうか。ここからは、実際にPCにUbuntuをインストールする流れをもとに、ハードウェアの動作確認を行っていきましょう。

Live USBの起動

最近のPCで、最初につまずく可能性があるポイントがUEFIとセキュアブート周りです。UEFIによって、ブートデバイスの選択画面やBIOS設定UIが非常にリッチになりました。しかし、リッチになったがゆえに製品によっては、USBデバイスから起動する方法がわかりにくくなっていたり、場合によってはUbuntuのLive USBをうまく認識できなかったりします。さらにUEFIでセキュアブートを有効にしていると、OSによっては起動できないこともあります。Ubuntuは署名済みカーネルを同梱しているため、基本的にセキュアブート有効でも起動できるようになっています。ただし、もしうまく認識できないようであれば、UEFIからセキュアブートを無効にしてみてください。それでもダメなら「レガシーBIOS」へ変更するような設定を行う必要があるかもしれません。

メモリのテスト

うまくLive USBを起動できたら、先にメモリテストを走らせておきましょう。PCの電源を入れてから Shift キーを連打すると、Ubuntuを起動する前にGRUBの画面が表示されるはずです。そこで「Memory test」を選択するとメモリのテスト(Memtest 86++)が自動的に開始されます。

画面中ほどにある「Pass」の数が、テストを通過した回数です。一般的には1回では不十分で、連続的に負荷をかけた状況、温度が上がってきたときの動作、PCの外の電源環境の変化を見るためにも、ある程度長時間連続でテストを行うべきです。状況にもよりますが、一晩から一日程度テストを行いつつ放置しておくのがよいでしょう。

ちなみに今回のシステムですと、1回のテストが1時間半ぐらいでした。1回あたりのテスト時間を把握

したうえで、何回目のテスト（いつごろのテスト）でエラーが出たかを覚えておくと、今後の参考になるかもしれません。1回目にエラーが出るようであれば、そもそもメモリに問題があるか、接触不良の可能性が高いので、まずはそこから確認してください。

Live環境での動作確認とインストール

メモリのテストが終わったら、UbuntuのLive環境を起動し、動作確認を行います。この時点ではディスプレイに画面が表示されるか、ネットワークにつながるか、音が鳴るか、マウスやキーボードの挙動に問題がないかといった基本的な機能の確認だけで十分です。ほかの部分はインストール後に確認します。

インストール前のみにできる動作確認としては、ストレージI/Oのテストがあります。読み込みテストは常に実施できるのですが、書き込みテストはマウントしていない状態でしか実施できないのです。Dashから「ディスク」を起動し、ストレージを選択し、右上のハンバーガーアイコンから「ディスクのベンチマーク」を選択してください。

基本的な機能がLive環境でうまく動かない場合、そのUbuntuでは対応できない可能性が高いです。とくにネットワークは、このタイミングで動いていないとインストール後のパッケージのアップデートはおろか、動作するための調査すらままなりません。よってうまく動かないようなら、より新しいUbuntuか、ほかのLinuxディストリビューションを試してみてください。また、ただの接触不良や初期不良という可能性もあります。

なおグラフィックカードによってはこの時点では3Dアクセラレーションが有効にならず、Unityの動作が緩慢になることがあります。この対応はインストール後にした方が楽なので、今は我慢してください。

とくに問題がなければ、Ubuntuをインストールしましょう。

図1　3Dアクセラレーションの有効化の確認

```
$ /usr/lib/nux/unity_support_test -p
OpenGL vendor string:   Intel Open Source Technology Center
OpenGL renderer string: Mesa DRI Intel(R) Skylake DT  GT2
OpenGL version string:  3.0 Mesa 11.0.2

(...中略...)

Unity 3D supported:       yes
```

リスト1　/etc/default/grubの変更点

```
変更前
GRUB_CMDLINE_LINUX_DEFAULT="quiet splash"

変更後
GRUB_CMDLINE_LINUX_DEFAULT="quiet splash i915.preliminary_hw_support=1"
```

図2　GRUB設定を反映する

```
$ sudo update-grub
$ grep "preliminary" /boot/grub/grub.cfg
```

インストール後の動作確認

インストール後は各種デバイスの動作確認を行います。Ubuntuの場合はまず**図1**のように3Dアクセラレーションが有効か確認しましょう。

「Unity 3D supported」が「no」の場合、ソフトウェアレンダリングを行います。しかし、この処理はたいへん重く、ほぼ実用には耐えません。Unityを使い続ける場合は、「追加のドライバー」アプリケーションでグラフィックドライバを追加してください。

Intelの内蔵GPUを使う場合は最初から3Dアクセラレーションが有効になっています。ただしリリース前のデバイスについては特定のモジュールパラメータを指定しないと認識しません。Ubuntu 15.10で使用しているカーネル4.2はまだSkylakeが発売される前に開発していたカーネルなので、このままでは3Dアクセラレーションが有効にならないのです。

モジュールパラメータを追加するために、**リスト1**のようにGRUBの設定ファイル「/etc/default/grub」を編集します。編集したら設定を反映し、再起動してください（**図2**）。**図1**のように最後が「yes」になったら完了です。ちなみに4.2カーネルのSkylake用ドライバはまだ公開前ということもあって、システムログに警告がいろいろと表示されます。このあたりは4.3以降のカーネルを採用するはずのUbuntu 16.04 LTSでは、解消される見込みです。

それ以外にもいくつか動作確認すべき項目があります。まずシステムログ（dmesgやjournalctl）に何らかの警告やエラーが出ていないか確認しておきましょう。「$ sudo lshw」を実行すると、Windowsで言うところのデバイスマネージャーのように認識しているデバイスリストが表示されますので、期待どおりの結果になっているか確認できます。マザーボードのUSBポートにUSBデバイスをつなげて、「$ sudo lsusb」を実行してどのポートもちゃんと動くかどうか確認しておくのもよいでしょう。

最近のUbuntuは、イヤフォンジャックにイヤフォンを接続すると、ボリュームやミュートを自動調整する機能が備わっています。システム設定の「サウンド」を開いてテストしてみましょう。同様に音量や輝度のファンクションキーが動作するかどうかも確認しておきましょう。ノートPCの場合は、サスペンドとレジュームがちゃんと動くかどうかも確認すべき項目の1つです。

パフォーマンスのチェック

少なくともハードウェアが認識されていることを確認したら、あとは「十分なパフォーマンスが出ているか」どうかを確認していきます。確認手順はデバイスごとに異なりますが、ここではより一般的な方法をいくつか紹介しましょう。

起動速度

最近はPCIe接続のSSDからのOSの起動に対応したマザーボードも増えてきました。PCIe接続のSSD自体も値段が下がってきたので、システム領域としてこのデバイスを使うこともこれから増えてくることでしょう。とくにUbuntuの場合はシステム領域が8GBぐらいあれば十分ですので、容量が小さいタイプのSSDでも十分実用に耐えます。そうなると気になるのが、「どれくらい起動時間が速くなるのか」です。

起動時間の計測には「bootchart」というソフトウェアがよく使われます。これは起動時の各プロセスの起動タイミングとCPUやI/Oの消費量をまとめたもので、起動時のボトルネックの調査に非常に有用なツールです。もともと独立したソフトウェアだったのですが、現在はすべてを統べるsystemdに統合されてしまいました。

systemdを使用しない14.04以前はbootchartパッケージをインストールするだけで動作します。インストール後に再起動すると、/var/log以下にsvgファイルが保存されます。

systemdを採用した15.04以降は「/etc/default/grub」を**リスト2**のように編集します。3Dアクセラレーション用にモジュールパラメータを設定しているのであれば、それは残したまま、その後ろに追記してください。編集したら設定を反映し、再起動してください（**図3**）。再起動後は、**図4**のように/run/log/以下にsvgファイルが保存されています。/run以下は再起動すると消えてしまいますので、必要であれば別の場所にファイルを移動してください。

プロセス単位ではなく、systemdの各サービス単位の起動時間を調べたいだけであれば、systemd-analyzeコマンドが便利です。こちらはとくに設定は必要ありません（**図5**）。

ストレージのI/O

Live環境で実施した「ディスクのベンチマーク」はファイルシステムに関係ない生の速度に近い値です。それに対して実生活では、ファイルシステム経由でストレージにアクセスしますので、個々のファイルシステム上のI/Oパフォーマンスも調べておきたいところです。

簡単に計測したいのであればddコマンドが便利ですが、ここではfioを使うことにしましょう。fioは計

リスト2　/etc/default/grubへの変更

```
変更前
GRUB_CMDLINE_LINUX_DEFAULT="quiet splash"
変更後
GRUB_CMDLINE_LINUX_DEFAULT="quiet splash init=/lib/systemd/systemd-bootchart"
```

図3　GRUB設定の反映

```
$ sudo update-grub
$ grep "bootchart" /boot/grub/grub.cfg
```

図4　bootchartの結果

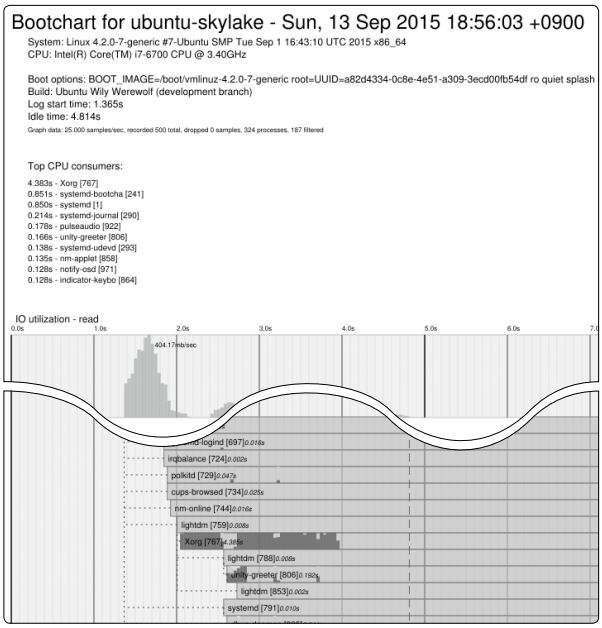


図5　解析結果をSVGファイルとして保存

```
$ systemd-analyze plot > plot.svg
```

リスト3　ジョブファイルcrystaldiskmark.fio

```
[global]
ioengine=libaio
iodepth=1
size=1g
direct=1
runtime=60
filename=fio.
dat

[Seq-Read]
bs=1m
rw=read
stonewall

[Seq-Write]
bs=1m
rw=write
stonewall

[512K-Read]
bs=512k
rw=randread
stonewall

[512K-Write]
bs=512k
rw=randwrite
stonewall

[4K-Read]
bs=4k
rw=randread
stonewall

[4K-Write]
bs=4k
rw=randwrite
stonewall

[4K-QD32-Read]
iodepth=32
bs=4k
rw=randread
stonewall

[4K-QD32-Write]
iodepth=32
bs=4k
rw=randwrite
stonewall
```

図6　ベンチマークの実行

```
$ fio crystaldiskmark.fio
（中略）
Seq-Read: (groupid=0, jobs=1): err= 0: pid=29856: Sun Sep 13 21:10:19 2015
  read : io=1024.0MB, bw=448685KB/s, iops=438, runt=  2337msec
Seq-Write: (groupid=1, jobs=1): err= 0: pid=29891: Sun Sep 13 21:10:19 2015
  write: io=1024.0MB, bw=428165KB/s, iops=418, runt=  2449msec
（中略）
4K-Read: (groupid=4, jobs=1): err= 0: pid=29999: Sun Sep 13 21:10:19 2015
  read : io=1024.0MB, bw=34718KB/s, iops=8679, runt= 30203msec
（中略）
4K-Write: (groupid=5, jobs=1): err= 0: pid=30351: Sun Sep 13 21:10:19 2015
  write: io=1024.0MB, bw=143111KB/s, iops=35777, runt=  7327msec
$ rm fio.dat
```

測ジョブのパラメータをファイルとして渡すことができます。この設定ファイルを調整することで、Windows関連でよく使われているCristalDiskMarkと同じような計測ができるのです。

fioパッケージをapt-getでインストールしたら、**リスト3**のようなジョブファイルを作成してください。

あとはfioコマンドにこのジョブファイルを渡すだけです（**図6**）。実行したディレクトリの直下にfilenameで指定したファイルを作成して計測します。計測サイズを増やしたい場合は、ジョブファイルのsizeを増やしてください。

注目すべきは「bw=」や「iops=」の値です。ちなみにCrystalDiskMarkと似たような計測方法ではありますが、そもそもソフトウェアやOS、ファイルシステムが異なるため、直接比較できる値ではありませんのでご注意ください。

ビルドテスト

CPUの純粋な計算速度だけでなく、統合的な「処理速度」を計測するツールはいくつも存在します。そのうちSysBenchやApache Benchなどは、お手軽なこともあってよく見かけるでしょう。またLinux上でのハードウェア／ソフトウェアスタックのレビューや性能比較を行っているPhoronix[注2]が開発しているPhoronix Test Suiteには、より多くのツールが組み込まれているため、広範囲のテストを一括して実施できます。またOpenBenchmarking.org[注3]にアップロードされたほかの結果との比較も容易です。これらはいずれもubuntu-benchmark-toolsパッケージで導入できます。

統合的なベンチマークツールとは別のアプローチとして、「大きなプロジェクトのビルド時間」でパフォーマンスを計測するという方法もあります。Linuxディストリビューションはたいていの場合、

注2）http://www.phoronix.com/
注3）http://openbenchmarking.org/

ソースパッケージをダウンロードし、パッケージのビルド環境を構築する手順は簡略化されているため、専用のベンチマークツール導入とたいして変わらない手間で実施できるのです。

Ubuntuのパッケージで、ビルドに時間がかかりそうなプロジェクトの1つがLibreOfficeでしょう。ソースパッケージだけでも数百MBのサイズになり、最新のCPUであっても数時間のビルド時間が必要なビッグプロジェクトです[注4]。

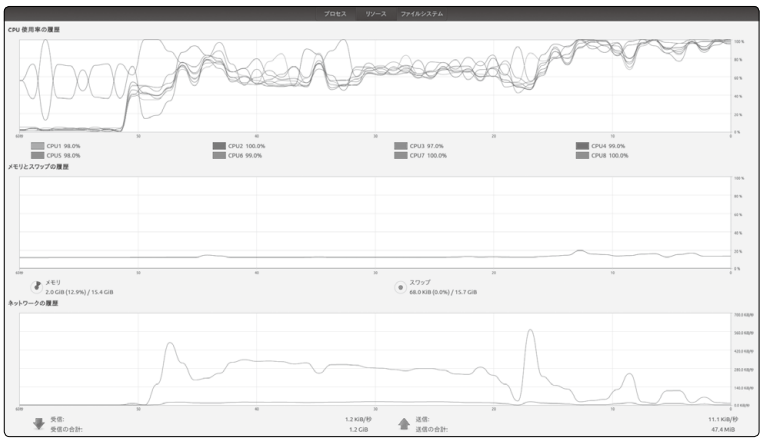
図7はLibreOfficeパッケージのビルド手順です。うまくビルドができたらホームディレクトリのpbuilder/以下にバイナリパッケージとともにビルドログが保存されます。このうちビルド開始時と終了時に「Current time」が記録されますので、その差分を見ればおおよそのビルド時間がわかるというわけです。

図8はビルド中のシステムモニターの様子です。本格的にコンパイルを行いだした後ろ3分の1ぐらいから4コア8スレッドのCPUがすべて100%で張りついているのがわかります。ちなみにSkylakeのCore-i7でも、1時間ぐらいはこの状態が続いていました。

図7　LibreOfficeパッケージのビルド

```
$ sudo apt-get install ubuntu-dev-tools cowbuilder
$ cowbuilder-dist wily create
$ apt-get source libreoffice
$ cowbuilder-dist wily build libreoffice*.dsc
$ grep "^I: Current time" ~/pbuilder/wily_result/libreoffice*.build
I: Current time: Sun Sep 13 21:58:42 JST 2015
I: Current time: Mon Sep 14 00:26:50 JST 2015
```

図8　libreofficeのビルド中のシステムモニター



まとめ

Intelは積極的にLinuxコミュニティへのコミットを行っているため、どのデバイスも比較的簡単に動くという特徴があります。Intel自身、UbuntuやFedora向けに最新グラフィックスタック用インストーラ[注5]を配布しているぐらいです。よって新しいPCでも、半年後ぐらいにリリースされたUbuntuであれば「メインの機能」はほぼ問題なく動きます。

ただPCベンダによってカスタマイズされている部分やIntel以外の部分、とくにNICやグラフィックカードについては、個々のベンダに依存しますので、可能な限りLive環境でテストする前にどこのベンダのなんという型番かは調べておきましょう。「型番 Ubuntu」のように検索すれば、ちゃんと動くかどうかや、運がよければその手順もわかる場合があるからです。

ベンチマークの結果を「正しく」理解するためにはハードウェアとソフトウェアに対する正しい知識が必要です。しかしながら、新しいPCになったのだからどれくらい快適になったか「なんとなく知りたい」程度であれば、そこまで知識は必要ありません。せっかくですので、一度、新旧の環境でベンチマークを動かしてみるのはいかがでしょうか。また、今回は触れませんでしたがビデオデコードやエンコードもパフォーマンステストではよく行われます。VA-APIに対応したツールを使って、試してみるのもよいでしょう。SD

注4）ソースコードのサイズだけで言うと1GBを越えるTeX Liveの方が上ですが、こちらはあまりにも大き過ぎるため、ソースパッケージの時点で複数に分割されています。

注5）https://01.org/linuxgraphics

第45回 Linux 4.1の新機能
ext4の暗号化機能

Text：青田 直大　AOTA Naohiro

10月18日にLinux 4.3-rc6がリリースされています。だいたいrc7かrc8まで出て、リリースとなっているのでこの記事が出ているころにはLinux 4.3もリリースされているのではないかと思います。今月はLinux 4.1からの新機能であるext4の暗号化について見ていきます。

ファイルシステムの暗号化

Ubuntuのインストーラでも、暗号化したファイルシステムの作成がオプションとして提供されているように、ファイルシステムの暗号化には一定のニーズがあります。現状よく使われているのは、dm-crypt、eCryptFS、EncFSの3つでしょうか。

dm-cryptは、デバイスマッパ上に実装され、ブロック単位で暗号化を行い、そのうえにファイルシステム（やその他なんでも）を作ることができます。dm-cryptがデバイスの上に重ねるのに対して、eCryptFSはほかのファイルシステムの上に重ねて暗号化を提供しています。最後のEncFSもeCryptFS同様にほかのファイルシステムの上に重ねるタイプの暗号化ですが、ほかの2つがカーネル内で実装されているのに対して、これはFUSEを用いてユーザランドでの実装を行っています。

ext4の暗号化

こうしたファイルシステムの暗号化の手法がある中、Linux 4.1ではext4にも暗号化機能が実装されました。ファイルシステム自身に暗号化機能が実装されているので、ブロック単位で暗号化を行うdm-cryptや、あるいはほかのファイルシステムの上にスタックするEncFSやeCryptFSなどと比較して、より効率的で柔軟なファイルの暗号化ができることが期待されます。

まずはext4の暗号化を使ってみましょう。この機能を使うには、カーネルだけでなくユーザランド側のツールも必要です。e2fsprogs-1.43_pre20150518.ebuildのようにコピーし、emergeすることでもインストールできます。

まずは、"-O encrypt"でext4の暗号化フラグを立てて暗号化機能が有効になったファイルシステムを作ります（**図1❶**）。ここでは新たなファイルシステムを作っていますが、tune2fsコマンドを使って既存のext4ファイルシステムにencryptフラグを立てることもできます。当然このフラグを立てた場合、以前のカーネルではmountできなくなるので注意してください。

ext4の暗号化はディレクトリツリーの単位で行われます。暗号化を有効にしファイルシステ

ム上にディレクトリfooとbarを作成し、一方は空ディレクトリ時に、もう一方はファイルを作ったあとに暗号化を行うコマンドを実行してみます(**図1❷**)。

暗号化には“e4crypt add_key”を使います。このコマンドは“-S”オプションで暗号のsaltを指定し、パスフレーズを入力することで、指定したディレクトリおよびその下のファイルツリーに暗号化の設定を行います。saltの指定はリストのように“s:”から始めて文字列を指定する方法や、“/”から始まるフルパスあるいは“f:”から始まるファイルパスを指定してファイルの内容で指定する方法、そして“0x”から始めてhex文字列でバイナリデータを指定する方法の3つの方法があります。

パスフレーズを入力すると、暗号鍵がfooに適用されたとのメッセージが出ます。ここでfoo、bar下にファイルを作り、さらにfoo/1に“foo”と書いておきます。その後、先ほどと同様に暗号化コマンドを“bar”に対して使ってみます。パスフレーズは聞かれていますが、ディレクトリが空ではないので、“Error [Directory not empty]setting policy.”とエラーが出ています。このように暗号化設定は空ディレクトリのときに行う必要があります。

“e4crypt get_policy”を使うと、ディレクトリに指定されている暗号鍵のIDを見ることができます。“e4crypt add_key”で、適用されたIDが出力されているのを見ることもできます。

暗号化されたファイルが、暗号化した本人以外からどのように見えるのかを見てみましょう。ファイルシステムを一度mountしなおして、rootとしてfoo、barを見てみましょう。暗号化していないbarの中は普通に元のファイル名が見えています。しかし、暗号化されているfooの中はファイル名がランダムのような文字列に変わっています(**図2❶**)。

では、ファイルへのアクセスについてはどうでしょうか。ファイルサイズ(とinode番号)から“foo/1”に対応しているとわかる“foo/hn+EPsLo2RbnSVQWvlMsWA”を読み込もうとしても、鍵がないとのエラーが出て読み込むことはでき

▼図1　ファイルの暗号化

```
$ truncat -s 30G ext4.img
$ /sbin/mkfs.ext4 -O encrypt ext4.img                    # ❶暗号化機能を有効にしたFSの作成
mke2fs 1.43-WIP (18-May-2015)
Discarding device blocks: done
Creating filesystem with 7864320 4k blocks and
$ /usr/sbin/e4crypt get_policy foo
foo: c75cff06381143e0
$ touch {foo,bar}/{1,2,3}; echo foo > foo/1
$ ls -li foo/* bar/*
655362 -rw-r--r-- 1 naota naota 0 Oct 26 20:54 bar/1
655363 -rw-r--r-- 1 naota naota 0 Oct 26 20:54 bar/2
655364 -rw-r--r-- 1 naota naota 0 Oct 26 20:54 bar/3
393218 -rw-r--r-- 1 naota naota 4 Oct 26 20:55 foo/1
393219 -rw-r--r-- 1 naota naota 0 Oct 26 20:54 foo/2
393220 -rw-r--r-- 1 naota naota 0 Oct 26 20:54 foo/3
$ /usr/sbin/e4crypt add_key -S s:salt2 bar               # ❷barの暗号化指定(失敗)
Enter passphrase (echo disabled):
Added key with descriptor [56786db4865cc0ae]
Added key with descriptor [b1f33fc98fc26cec]
Error [Directory not empty] setting policy.
The key descriptor [b1f33fc98fc26cec] may not match the existing encryption context for
directory [bar].
$ /usr/sbin/e4crypt get_policy foo bar
foo: c75cff06381143e0
Error getting policy for bar: No such file or directory
```

ません(**図2❷**)。もちろんディレクトリの外にリンクしてみても同様に読めませんし、rootであっても鍵を知らないので書き込むことができません。また、暗号化されたディレクトリの中に新しいファイルを作ることもできません。ただし、ファイルを削除する権限があればそのファイルを削除することはできます(**図2❸**)。

このようにext4の暗号化では、鍵がなくてもファイルの存在やinode番号、タイムスタンプという情報は取得できます。ただし、ファイル名は一定の方法でハッシュ化され、元のファイル名を知ることはできません。

ここまでのコマンドで追加した鍵は、基本的には再起動するまでは、鍵を追加したユーザと結びつけられています。すなわちファイルシステムをumountしても鍵の再設定は必要ありません。しかし、再起動後や「鍵セッション」の変更後は、再度鍵を追加する必要があります。鍵の追加は暗号化を設定したときと同じコマンドを繰り返すだけです(**図3**)。

inodeごとの暗号鍵

もう少し深くext4の暗号化を見ていきましょう。同じ暗号鍵を使って2つの暗号化ディレクトリに、同一の名前のファイルを作ったら、その2つのファイルの鍵がないときに表示されるファイル名は一致するのでしょうか。あるいは同じ暗号鍵を使って、2つの同一の内容のファイルを作ったらそのディスク上のデータは一致するのでしょうか。

図4のように同じ暗号鍵を使って同一ファイル名かつ同一コンテンツのファイルfoo0/1とfoo1/1を作ります。まずはファイル名を見てみると、ちゃんとディレクトリごとに表示されるファイル名が変わっているのがわかります(**図4❶**)。

同様にディスク上のデータについても見てみましょう。こちらはdebugfsコマンドで見ていきます。まず、"blocks"コマンドをファイルのinode番号に対して使って、ファイルデータを

▼図2　暗号化されたファイルへのアクセス

```
$ cd; sudo umount /mnt/tmp
$ sudo -i
# mount ~naota/ext4.img /mnt/tmp
# cd /mnt/tmp; ls -li foo/* bar/*                    # ❶ファイル名が暗号化されている
655362 -rw-r--r-- 1 naota naota 0 Oct 26 20:54 bar/1
655363 -rw-r--r-- 1 naota naota 0 Oct 26 20:54 bar/2
655364 -rw-r--r-- 1 naota naota 0 Oct 26 20:54 bar/3
393220 -rw-r--r-- 1 naota naota 0 Oct 26 20:54 foo/FTZnujPSL2Y5gTbFtqOXUD
393218 -rw-r--r-- 1 naota naota 4 Oct 26 20:55 foo/hn+EPsLo2RbnSVQWvlMsWA
393219 -rw-r--r-- 1 naota naota 0 Oct 26 20:54 foo/JDRQqMbrYMYe+JDe7h,36A
# cat foo/hn+EPsLo2RbnSVQWvlMsWA                    # ❷暗号化されたファイルの読み込み
cat: foo/hn+EPsLo2RbnSVQWvlMsWA: Required key not available
# ln foo/hn+EPsLo2RbnSVQWvlMsWA baz
# cat baz
cat: baz: Required key not available
# echo > foo/hn+EPsLo2RbnSVQWvlMsWA
-bash: foo/hn+EPsLo2RbnSVQWvlMsWA: Required key not available
# touch foo/aaa
touch: cannot touch 'foo/aaa': No such file or directory
# rm foo/hn+EPsLo2RbnSVQWvlMsWA                     # ❸暗号化されたファイルの削除は可能
# exit
# ls -li foo/* bar/*                                 # foo/1 が消えている
655362 -rw-r--r-- 1 naota naota 0 Oct 26 20:54 bar/1
655363 -rw-r--r-- 1 naota naota 0 Oct 26 20:54 bar/2
655364 -rw-r--r-- 1 naota naota 0 Oct 26 20:54 bar/3
393219 -rw-r--r-- 1 naota naota 0 Oct 26 20:54 foo/2
393220 -rw-r--r-- 1 naota naota 0 Oct 26 20:54 foo/3
```

▼図3　暗号鍵の再設定

```
(reboot ...)
$ sudo mount ext4.img /mnt/tmp
$ cd /mnt/tmp; ls -li foo/*
$ ls -li foo/*                                    # 鍵がないので読めない
393220 -rw-r--r-- 1 naota naota 0 Oct 26 20:54 foo/FTZnujPSL2Y5gTbFtqOXUD
393219 -rw-r--r-- 1 naota naota 0 Oct 26 20:54 foo/JDRQqMbrYMYe+JDe7h,36A
$ /usr/sbin/e4crypt add_key -S s:somesalt foo     # 鍵の追加
Enter passphrase (echo disabled):
Added key with descriptor [19cb4cd62773ca20]
Added key with descriptor [c75cff06381143e0]
Key with descriptor [c75cff06381143e0] applied to foo.
$ ls -li foo/*                                    # ファイルが読めるようになった
393219 -rw-r--r-- 1 naota naota 0 Oct 26 20:54 foo/2
393220 -rw-r--r-- 1 naota naota 0 Oct 26 20:54 foo/3
```

▼図4　暗号化された同一のファイルを比較する

```
$ /usr/sbin/e4crypt add_key -S s:somesalt foo{0,1}
Enter passphrase (echo disabled):
Key with descriptor [19cb4cd62773ca20] already exists
Key with descriptor [c75cff06381143e0] already exists
Key with descriptor [c75cff06381143e0] applied to foo0.
Key with descriptor [c75cff06381143e0] applied to foo1.
$ echo foo > foo0/1
$ echo foo > foo1/1
(reboot ...)
$ ls -li foo0/* foo1/*
1441794 -rw-r--r-- 1 naota naota 4 Oct 27 01:07 foo0/FzAk81TGszPbrEhct2wg6C
 393218 -rw-r--r-- 1 naota naota 4 Oct 27 01:07 foo1/paj3RWcqpwvhKph0YClCKA
$ /sbin/debugfs ext4.img -R 'blocks <1441794>'      # inode 1441794 の file block を取得
debugfs 1.43-WIP (18-May-2015)
5799936
$ /sbin/debugfs ext4.img -R 'blocks <393218>'
debugfs 1.43-WIP (18-May-2015)
1606661
$ /sbin/debugfs ext4.img -R 'bd 5799936'|head
debugfs 1.43-WIP (18-May-2015)
0000  75f6 f9b7 adc6 01db f8a0 b1f3 754b e096  u...........uK..
0020  df3b 1e06 079a d2c6 0a17 dfe0 b186 72e6  .;............r.
0040  e025 8d23 e2f0 a98f 953c 93c4 27bb 05e3  .%.#.....<..'...
0060  423f a990 d210 8ac6 2e03 306d 267f a061  B?........0m&..a
0100  3cd7 dce7 bdb9 9e11 12db fde3 df6c b86d  <............l.m
0120  2637 31a4 33a3 7b97 6ebe c15a dc4a fef4  &71.3.{.n..Z.J..
0140  a7c1 af18 f857 628c 9d17 1715 f033 a2d7  .....Wb......3..
0160  35e2 ef3c 090b 530c 86c3 f744 fe62 69fe  5..<..S....D.bi.
0200  8d5f b0cf 6081 662e eb5b 67eb da06 4436  ._..`.f..[g...D6
0220  73ad 5959 dcd8 85db 3ff3 8157 e869 f291  s.YY....?..W.i..
$ /sbin/debugfs ext4.img -R 'bd 1606661'|head
debugfs 1.43-WIP (18-May-2015)
0000  b5ec 359b 8be7 86ff 73e4 8a0d bd72 158a  ..5.....s....r..
0020  e23e 3e41 391d 2e09 e151 7836 866b be6f  .>>A9....Qx6.k.o
0040  7573 9a71 5fa3 9646 c1c0 399f ee77 d834  us.q_..F..9..w.4
0060  0edd 4d5b b05c 6d95 f865 6397 ec57 04fb  ..M[.?m..ec..W..
0100  6b0d 169a 8434 81cb 67dc b89b 3d3f a436  k....4..g...=?.6
0120  4c06 2d77 efe3 331e 6697 4e02 647c 475c  L.-w..3.f.N.d|G?
0140  3887 5c59 d2fa 0917 3c8c 7cf5 bc1e 5308  8.?Y....<.|...S.
0160  8561 f979 51fb 44ff a507 0ebd 7b0a 2b98  .a.yQ.D.....{.+.
0200  bf36 7572 670a 5893 bc54 5330 080e 4249  .6urg.X..TS0..BI
0220  955d 605b 2231 f853 f1f2 31e5 ca6c d827  .]`["1.S..1..l.'
```

保存しているブロック番号を取得します。さらに"bd"コマンドでそのブロックデータをダンプします。もともとは同一コンテンツであるのに、比べてみるとまったく異なっているということが確認できます。

この結果から確認できるように、ext4ではinodeごとにランダムなnonceを作成し、指定した暗号鍵とnonceとを組みわせてファイル名およびデータの暗号化を行っています。この情報はinodeのextended attributeとして保存されています。この値は通常は見ることができませんが、debugfsの"ea_list"コマンドでダンプできます(**図5**)。データは、構造体ext4_encryption_contextになっており、最初の4byteがフォーマット番号、ファイルデータの暗号化形式(ここではAES_256_XTS)、ファイル名の暗号化形式(AES_256_CTS)、フラグを示しています。次のc7から始まる8byteが暗号鍵の"c75cff06381143e0"となっており、e4cryptが出力しているものと一致していることが確認できます。そのあとの16byteがinodeごとのnonceとなっていて、この部分は各inodeで違っていることが確認できます。

カーネルによる鍵の管理

さて、これまで"e4cyrpt add_key"によって「鍵が追加」されると書いてきましたが、その鍵とはどこにどんな形式で保存され、誰が管理しているのでしょうか。実はLinuxカーネルは鍵を管理し、必要に応じて提供する機能を持っています。この鍵管理システムを見ていくことで、ext4の暗号化の裏側を探ることができます。

まずはカーネルがどのような「鍵」を持っているのかを見てみましょう。カーネルの「生の」インターフェースとしては/proc/keysと/proc/key-usersとがあります(**図6**)。

前者は自分の鍵のリストを、左からシリアル番号、鍵の状態を示すフラグ、使用数、鍵の期限、パーミッション、UID、GID、タイプ、説明文といった形式で表示します。後者はシステム全体での鍵の使用量を示すファイルです。左からUID、参照カウント、キーの数(有効数/全体)、キー数のquota、キーサイズのquotaを示しています。また、/proc/keysをわかりやすく表示するものとして"keyctl show"コマンドを使うこと

▼図5　inodeごとの暗号設定のダンプ

```
$ /sbin/debugfs ext4.img
debugfs 1.43-WIP (18-May-2015)
debugfs:  ea_list foo0
Extended attributes:
  c = "01 01 04 00 c7 5c ff 06 38 11 43 e0 e5 51 36 34 ff d4 ec 57 94 a7 ec bd d9 11 df 1c " (28)
debugfs:  ea_list foo1
Extended attributes:
  c = "01 01 04 00 c7 5c ff 06 38 11 43 e0 ee 71 12 9d 71 d6 25 36 0c e6 f7 df 6c 46 f8 f3 " (28)
```

▼図6　鍵情報の表示

```
$ cat /proc/keys
38c2cfaf I--Q---     1 perm 1f3f0000  1000 65534 keyring   _uid_ses.1000: 1
3cd234e5 I--Q---     2 perm 1f3f0000  1000 65534 keyring   _uid.1000: empty
$ cat /proc/key-users
    0:     6 5/5 2/1000000 22/25000000
 1000:     2 2/2 2/200 28/20000
$ keyctl show -x
Session Keyring
0x38c2cfaf --alswrv   1000 65534  keyring: _uid_ses.1000
0x3cd234e5 --alswrv   1000 65534   ?_ keyring: _uid.1000
```

ができます。

では、暗号化を設定すると鍵はどうなるでしょうか。先ほどと同様に、e4cyrptを使って暗号化を設定し、鍵を見てみます(**図7**)。

すると、"ext4:19cb4cd62773ca20"と"ext4:4628fd0bbe3215f8"という鍵が追加されていることが確認できます。これらの鍵のタイプは"logon"となっており、このタイプの鍵はカーネルからは参照できますがユーザからは読めないというものです。

これらの鍵は、ユーザごとの"デフォルトセッション"のキーリングに追加されており、そのためログアウトしても鍵情報は残っています。では、一時的に暗号化された部分が見えないプロセスを作るにはどうしたらいいでしょうか。そのためには、新しい鍵セッションを作成する"keyctl session"(またはe4crypt new_session)を使います。これを使うと新しいセッションが開始され、鍵が見えなくなります(**図8**)。実際一度ディレクトリキャッシュをクリアしてから、ファイル名を表示してみると暗号化されているのがわかります。

他ファイルシステムとの共通化

ファイルシステム自身での暗号化機能は、実はext4以外に、すでにF2FSにも実装されています。どちらのファイルシステムもファイルごとの暗号化設定の読み書き、およびsaltを取得するためのioctlを実装しています。

ext4のメンテナであるTed Ts'oはF2FSのメンテナと、2つのインターフェースを共通化する議論を行っているようです。将来的にはこれらのioctlが一般化され、さまざまなファイルシステムで暗号化機能がシームレスに使えるようになるのかもしれません。SD

▼図7　鍵のタイプを表示する

```
$ /usr/sbin/e4crypt add_key -S s:somesalt foo
Enter passphrase (echo disabled):
Added key with descriptor [19cb4cd62773ca20]
Added key with descriptor [4628fd0bbe3215f8]
Key with descriptor [4628fd0bbe3215f8] applied to foo.
$ cat /proc/keys
03bdccb1 I--Q---     1 perm 3d010000  1000  1000 logon     ext4:19cb4cd62773ca20: 72
36c1e96c I--Q---     2 perm 3d010000  1000  1000 logon     ext4:4628fd0bbe3215f8: 72
38c2cfaf I--Q---     1 perm 1f3f0000  1000 65534 keyring   _uid_ses.1000: 5
3cd234e5 I--Q---     2 perm 1f3f0000  1000 65534 keyring   _uid.1000: empty
$ keyctl show -x
Session Keyring
0x38c2cfaf --alswrv   1000 65534  keyring: _uid_ses.1000
0x3cd234e5 --alswrv   1000 65534   ?_ keyring: _uid.1000
0x36c1e96c --alsw-v   1000  1000   ?_ logon: ext4:4628fd0bbe3215f8
0x03bdccb1 --alsw-v   1000  1000   ?_ logon: ext4:19cb4cd62773ca20
```

▼図8　暗号化されたものを非表示にする

```
$ touch foo/1
$ keyctl session (または /usr/sbin/e4crypt new_session)
Joined session keyring: 322164927
$ keyctl show
Session Keyring
 322164927 --alswrv   1000  1000  keyring: _ses
$ cd ..; echo 3 | sudo tee /proc/sys/vm/drop_caches; cd -
$ ls -l foo/*
-rw-r--r-- 1 naota naota 0 Oct 27 11:26 foo/oGy4BHHDKOJVDVzPm9E,ZA
```

December 2015

NO.50

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

リングに再び帰るLLイベント

今回は、13年目を迎えたLightweight Languageイベント（LLイベント）について報告します。今年は2006年のLL Ringで使用したプロレス専用会場・新木場1stRingを再びお借りし、リングに再び帰るという意味を込めてLL Ring Recursiveと題して開催しました（**写真1**）。参加者は166人でした。以下、実施したプログラムを紹介します。

Lightweight Language Ring Recursive

■Lightweight Language Ring Recursive
【日時】2015年9月5日（土）10:30～20:00
【会場】新木場1stRING

■Language Update

- 出演：JavaScript：竹迫良範
  Perl：徳永広夢（subtech）
  PHP：hnw（KLab㈱）
  Ruby：成瀬ゆい
- 司会：高橋征義（㈱達人出版会）

写真1　LL Ring Recursiveスタッフ一同（リング上にて）

各言語の近況を伝えるセッションです。今回は初めての試みとして、司会からお題を提示し、それに各出演者が回答するという方式を採用しました。最新版で変わったところ、すでに古くなってしまった仕様、ほかの言語との比較、個人的偏愛点などについて話をうかがいましたが、言語に精通した方々による内容の濃い回答のおかげで、情報量の多いセッションになりました。

■IoT時代のLLスタック

- 出演：青木俊介（ユカイ工学㈱）、若狭正生、岡島康憲（岩淵技術商事㈱）、椎野孝弘（ヤフー㈱）
- 司会：森藤大地（ニフティ㈱）

IoTデバイスをLLで操作するために必要な技術や知識を紹介するセッションです。IoTを取り巻く状況として注目のベンダやデバイスの紹介、なぜ組み込み言語ではなくLLでデバイスを扱うのか、LLでプログラムを書く側から見たデバイスの制約や制御の難しさ／楽しさ、IoTの延長上に何を目指すのか、といった話題について討論しました。IoTデバイスは小型の物が多いため、PCの内蔵カメラでデバイスを撮影してスクリーンに投影していました。

■パネルディスカッション：エンジニア不足はいつまで続くのか

- 出演：伊藤健吾（㈱キャリアデザインセンター）、清水俊博（㈱ドワンゴ）、吉田浩一郎（㈱クラウドワークス）、吉田真吾（フリーランス）
- 司会：小山哲志（合同会社ほげ技研）

近年、各企業でエンジニアが不足している状況について討論するセッションです。IT系の求人動向の変化、採用する側／される側の考え、副業の是非、エンジニア人口を増やすにはどうすれば良いかなど、多面的な議論が交わされました。「Web系エンジニアは、新卒で入った世代がまだ定年を迎えていないためキャリアパスが見えにくい」「安定は会社ではなく自分に求めるもの」など、印象に残るコメントがいくつも聞かれました。

■LLが支えるデータサイエンスの世界

- 出演：佐藤建太（東京大学/JuliaTokyo）、大野健太（㈱Preferred Networks）
- 司会：村田賢太

機械学習やデータ分析の分野におけるLLの利用について話をうかがうセッションです。大野さんにはChainerというディープラーニングのフレームワークを紹介していただきました。ニューラルネットの理論に始まり、Chainerの環境構築、Pythonで書かれた学習プログラムの解説などがありました。佐藤さんからは科学技術計算向けのLLであるJuliaの紹介がありました。Juliaの特徴である動的なプログラミング、簡潔な言語仕様、高速な処理、科学技術計算向けの豊富なライブラリなどを、実際にJuliaを動かしながら説明されました。

■懇親会とLightning Talks

今回は会場内で飲食できることを利用して、本編に含める形で懇親会とライトニングトークを行いました。ライトニングトークは持ち時間3分で、当日会場にて発表者を募る方式を採用しましたが、総勢12名の方が応募し盛り上げてくださいました。タイトルと発表者は**表1**のとおりです。

■物販＆見本誌展示

今年もIT系出版社による書籍の即売と見本誌展示を行いました。参加してくださった出版社は、㈱オライリー・ジャパン、㈱オーム社、㈱達人出版会、USP出版、アスキードワンゴ、㈱インプレス、㈱マイナビ、㈱技術評論社、㈱日経BP、SBクリエイティブ㈱です。

■終わりに

9年前に同会場で開催したときはあっという間に満席になってしまったのですが、今年は当日券でも入場できるぐらいの参加者数で、かと言って空席が目立つほどでもなく、適正な規模で実施することができました。プログラムも充実した内容だったと思います。言語系のイベントや勉強会が増えた今、このイベントの位置付けを再考しつつ、またおもしろいものを企画できればと考えています。

LL Ring Recursiveの発表資料、写真、映像などはWebサイト[注1]に置いてあります。こちらもぜひご参照ください。SD

表1　ライトニングトークの発表内容

LTタイトル	発表者
IoTなんかシェルで十分だろが	上田隆一（USP友の会／千葉工業大学）
安全に「危険シェル芸」ができるスタートアップスクリプトのご案内	横田真俊（さくらインターネット㈱）
とある正規表現を高速化した話	西山和広
Lua言語	上野豊（産業技術総合研究所）
問題提起：「日々これ修業」の代わりは？	齊藤明紀（鳥取環境大学）
福利厚生の話	HaiTo
そろそろ焼きそばについて一言いっておくか	@kwappa（㈱ドワンゴ）
俺に焼きそばを焼けって言われても	kuzuha
言語をディスるのはやめろ	dark（㈱セプテーニ・オリジナル）
プロジェクト℃ ～夢見者たち～ 舞台裏	海老原寛之（㈱サイタスマネジメント）
PHPでRubyを攻略する	マスクドPHP
新言語XYを作ってみた	竹迫良範（㈱リクルートマーケティングパートナーズ）

（司会：法林浩之（日本UNIXユーザ会））

注1）URL http://ll.jus.or.jp/2015/

Hack For Japan
エンジニアだからこそできる復興への一歩

●Hack For Japanスタッフ
及川 卓也　OIKAWA Takuya
Twitter @takoratta
清水 俊之介　SHIMIZU Shunnosuke
Twitter @donuzium

第48回 Connect 2015 in Koriyama, with UDC

8月29日から2日間にわたって、福島県郡山市で「Connect 2015 in Koriyama, with UDC（アーバンデータチャレンジ）」が行われました。地域の課題をオープンデータを用いて解決するという目的のために、郡山市に集まった方々が泊まり込みで行ったハッカソンの模様を今回はお伝えします。

キックオフ

初日、参加者は郡山駅や郡山市役所などからバスを利用したり、また自家用車などで直接、会場の福島県郡山自然の家に集まりました。この会場は地元の方にとっては有名な施設だそうで、入浴施設、さらには食堂も完備されていて、まさにハッカソンのような開発合宿にはぴったりの会場でした。

まず、主催者などからの説明でイベントは開始されました。

今回のイベントの主催である、郡山地域ニューメディア・コミュニティ事業推進協議会からは、UDCの説明がされました。全国各地に拠点がある中、東北からは一関と会津若松に加えて郡山が選ばれたこと、「Connect」に込めた「笑顔でつないでいく」と「世界につないでいく」という想いが紹介されました。

郡山市政策開発部ソーシャルメディア推進課から郡山市のICTの取り組みが紹介された後、運営協力しているCode for KORIYAMAおよびエフスタ!!の代表の大久保仁さんから、郡山で活動しているITコミュニティ代表の立場からこのイベントへの期待が述べられました。大久保さんは、「まだまだ地方ではITの勉強会やハッカソンなども会社側からは遊びとして見られてしまうことが多いが、このイベントで理解が進むようになるようにしたい。2日間楽しんでほしい」と参加者に呼びかけました。また、地域課題を理解するため、郡山市政策開発部政策開発課から郡山市の課題と取り組みが紹介されました。郡山市在住の人であっても、結構自分の住んでいる自治体のことは知らないものなので、大変参考になったように見受けられました（**写真1**）。

アイデアソン

初日の午後はアイデアソンが行われました。ファシリテーターはNPO法人森とIT理事長の國枝裕介さんです。アイデアソンは次のようなステップで進められました。

◉個人ワーク

2色の付箋紙に、1つは「誰に？」に対して、もう1つは「どんな」価値を届けるかを、3つ以上書き出す

⬇

◉チーム内発表

チームを組み、付箋紙に書きだしたアイデアを1人ずつ発表。チームの他の人はそのアイデアをひたすら褒める

⬇

◉アイデアの分析／まとめ

付箋紙に書きだしたアイデアを模造紙に貼り付け、

◆写真1　郡山市のイメージキャラクター がくとくんも参加

誰に何を提供したいかにおいてチーム内で傾向があったかを話し合う。その後、改めてアイデアを整理する

⬇

◉ペアブレスト

ペアで輪になり、5分間ペアでブレインストーミングを行う。気づかされたことを1分間メモを取り、その後、1人移動し、別のペアとなり、同じことを繰り返す。これを5回行う

⬇

◉アイデアスケッチ

ここまでで温められてきたアイデアを、キャッチコピーをタイトルとした形できれいにまとめる。テキストだけでなく、図や絵などを折り込み、アイデアをよりわかりやすくする

⬇

◉上位案抽出

テーブルにアイデアスケッチを並べ、全員で回り、良いと思ったものに☆をつける

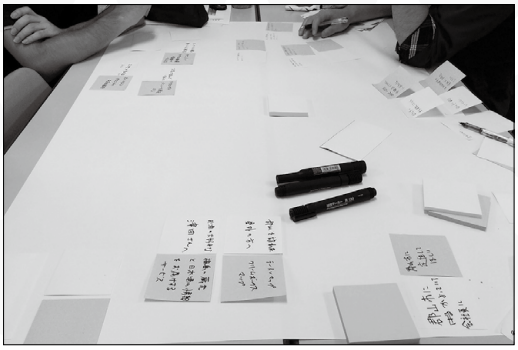
以上のアイデアソンの結果、ハッカソンでの開発の元ネタが固まっていきました(**写真2**)。

ハッカソン2日目

2日目は小雨の降る中、早朝のラジオ体操からスタート。筆者(清水)は初日からハッカソンに参加しました。

朝食を食べると、午後の成果発表に向けてハッカソン再開です。ハッカソンには珍しくチームを超えて助けあう参加者が多く見受けられたのは、日頃から熱心にコミュニティ活動をしている方々が多いからでしょうか。市民活動をする人をつなぎ、情報をオープンデータ化する「まちコミ!!」を作った「チームよし坊」を率いる大久保さんは、まさにそのコミュニティ活動の中心にいます。市民の声を可視化するための「にこにこイライラマップ(仮)」を制作した「チームFIC」を含めた2チームは、大久保さんの会社の後輩を中心に構成されていました。

◆写真2　アイデアソンの模様



雨のBBQ

スケジュールを見て楽しみにしていたBBQは雨で開催が危ぶまれたものの、屋根付きのスペースで無事決行されました。ただでさえ不慣れなうえに雨で湿気ってしまった薪に火をつけるのは、2日間で最難関のハックでしたが、それでも自分たちで火を起こして作ったご飯からはどこか懐かしい味が。途中煙がこもってしまい、屋根の縁の下に人が密集するイベントではありましたが、滅多に味わえないハッカソン中のBBQを皆さん満喫していました(**写真3**)。

BBQ会場は同じ施設ではあるものの施設自体が広いため、ハッカソン会場から歩いて2分ほどのところにありました。雨で足元がぬかるむ中、近くにトイレがあるのかわからず、とりあえず会場を往復しましたが、「チームエフコム」の作った「トイレマップ」があれば一番近いトイレを探すことができたかもしれません。「CSHと愉快な仲間」が作った「エンジョイ！Now！」は、まさに大きな1つの会場

◆写真3　みんなで作るBBQは最高！

で利用するためのアプリで、iBeaconにより詳細な位置情報と連動した情報を提供するため、今回のような会場で大活躍しそうです。

午後からは最後の発表に向けてラストスパートです。どのチームも気合が入ってきます。「チームこじん」のアプリ「アキカツ」は、スペースを提供する側、使いたい側が利用するもので、空きスペースの予約が簡単にできるアプリを目指して、チーム内での議論が白熱していました。

結果発表

ポンプン

長い審査を終えて、まず3位に選ばれたのが「WiZ Graduate」の皆さんが作ったアプリ「ポンプン」。

ポンプンは地域防災力の向上を目的とした消防団員の活動を支援します。消防署に連絡が入ると、すぐさま消防団員のインストールしているアプリに通知が届き、火災現場の情報を示すマップが表示できるというものです。また火災現場の位置情報だけではなく、消火栓や防火水槽の位置も把握することができ、自分たちの良く知っているエリア外であっても、それら消火活動に必要な情報を支援するためのアプリを目指したいということでした。

ポンプンでは通知することに終わらず、各隊員間でもそのアプリ内でコミュニケーションがとれるようなアイデアが盛り込まれています。現在は通知された後の連絡はそれぞれの電話やメールで行われていることが多いそうですが、通知から出動、現場での情報収集までカバーできます。すべてを完結できるこのようなアプリはいろいろな分野でも必要なものだと思います。

オストメイトトイレ

2位に選ばれたのは、会津大生とWiZの学生で構成されたチームの「SHED（データ部門）」の「オストメイトマップ」。ちなみにSHEDというのはアプリ開発合宿で一緒になったメンバーで作ったサークルの名前だそうで、今回は2つのチームに分かれて参加していました[注1]。

SHED（データ部門）のメンバーは、全国におおよそ10万人いるとされる人工肛門や人工膀胱保有者の方々（オストメイト）を支援するため、それらの情報を整理しLinkDataで公開し、サイトの機能を使ってマップ上に公開しました。LinkDataではExcelのようなデータ形式をRDF[注2]に変換してくれるため、よりWebやアプリ開発にフレンドリーなデータ形式となっています。

現在郡山市ではオストメイトトイレの情報をPDFファイルで公開している[注3]ため、このようなオープンデータを整備していくことも、シビックテックを支えるためには大切なことだと思います。

MAP＋DEPLOY＝MAPLOY

ここまで読んで気づかれた方も多いと思いますが、発表されたものの多くが、位置情報や地図を扱うものです。オープンデータを利用したアプリの中に、地図にデータをマッピングする機能が入ったものが多いのは今回に限った話ではなく、筆者（清水）が参加したハッカソンでも同じような機能を実装することが繰り返しありました。

浅井渉さんと結成したチーム「BAKAVIRUS_A」で制作した「.MAPLOY」は、地図を利用したアプリの開発やオープンデータの公開を支援するため、Excelファイルをドラッグ＆ドロップするだけで、データを地図上に可視化できるWebサービスです。さらにはAPIとしてもデータを提供することができます。運良く最優秀賞に選んでいただきました。

ワンアクションで、地図上に吹き出しの出るマーカーや「データの重み」を表現する円を配置することができます。住所だけのデータでも自動で緯度経度

注1　惜しくも受賞はなりませんでしたが、もう一方のSHED（アプリ部門）チームは「About Route Guide」という、キャラクターとの対話形式で希望の場所とそこまでのルートを提案してくれるアプリを発表しました。
注2　Resource Description Framework
注3　https://www.city.koriyama.fukushima.jp/212000/fukushi/documents/2381_osutomeito2207.pdf

をつける機能があるため、難しいことをあまり意識せずに、オープンデータを利用してシビックテックに参加できるようになればという想いもあります。

来年に向けて

来年開催されるUDC 2015本番に向けての、キックオフイベントでもあった今回の「Connect 2015 in Koriyama, with UDC」。年末に再度開催される審査会のために、アイデアをブラッシュアップしたり、開発を進めたりと今後も続いていきます。皆さん、また郡山でお会いしましょう！ SD

Column 情報支援レスキュー隊の一般社団法人化

この連載でも何回かお知らせしている、災害発生時に被災地から正しい情報を迅速に発信することで災害復旧・復興支援を行う情報支援レスキュー隊[注1]（英語名IT DART）が一般社団法人となりました。このコラムでは、8月8日に行われた創立総会の模様と現在の活動状況、今後の予定をお伝えします。

8月8日にスマートニュース社のイベントスペース（東京都渋谷区）で行われた創立総会では、まず一般社団法人としての創立総会・理事会が開催され、初年度の活動体制と計画が承認されました。体制としては、正会員（個人）および賛助会員（法人）からなる会員総会により会の運営方針が決定され、それを代表する立場として理事会および運営委員会が組織されるという体制となっています。一方、実際の活動の主体は隊員となり、これは会員総会とは別に組織されます（**図A**）。

正会員の年会費は1万円であり、賛助会員は同じく年会費10万円となっています。一方、隊員からは会費は徴収しません。正会員や賛助会員もそうですが、とくに隊員は発災時に被災地にすばやく赴くことを考えているので、全国各地から募集したいと考えています。すでに、正会員、賛助会員、隊員の募集と寄附の受け付けを開始しています[注2]。

8月8日の創立総会では、設立にあたっての挨拶を陸前高田市、多賀城市、茨城県の来賓3名の方からいただきました。また、50名弱の参加者とともに、初年度の活動内容をテーマに記念ワークショップが開催されました。

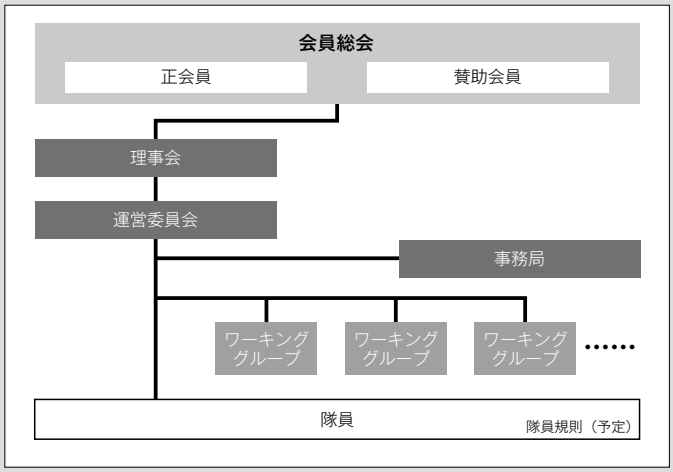
現在はまだ団体としての体制を固めている最中です。たとえば、隊員募集は開始しているものの、本原稿執筆段階では隊員規則もまだ作成中です[注3]。隊員の要件やトレーニング内容などを検討し、それを隊員規則に反映させる予定です。また、運営委員会の中にワーキンググループ（WG）を設立し、会の活動や平時の準備を進めています。現在立ち上がったWGは、情報システムWG、提携・協定促進WG、隊員活動WG、情報分析・発信WGとなります。たとえば、情報システムWGでは、会員や隊員間、さらには提携団体との間のコミュニケーションや情報共有手段の整備を進めています。また、発災時に用いる地理情報システム（GIS）や簡易Webサイト作成サービス[注4]の調査なども行っています。

このように、本格的な活動に向けての準備を進めるのと同時に、災害時の活動も開始しています。9月に北関東から東北にかけての豪雨で茨城県常総市など多くの地域が甚大な被害を被りました。この災害に対して、IT DARTも2回の被災地入りをするなど、「情報支援」の立場で活動を行っています[注5]。

まだ団体としての活動は開始されたばかりですが、技術の力での支援をぜひとも本格化させていきたいと考えています。ご興味のある方は会員や隊員としての参加をご検討ください。

注1 **http://www.itdart.org/**
注2 **http://itdart.org/wanted/**
注3 そのため、隊員募集は暫定的なものとなっております。
注4 JimdoやWix、Weebly、Strikinglyなど。
注5 **http://itdart.org/content/2015911-heavyrain/**

◆**図A IT DART体制図**



温故知新
IT むかしばなし

第49回

シリアル通信 ～高速モデムとホストマシンとの接続～

速水 祐(HAYAMI You)　http://zob.club/　Twitter : @yyhayami

はじめに

1980年代の中ごろ、筆者の地元で活動していた小田原マイコンクラブの例会では、遠く離れた場所の公衆電話機とパソコンとの間でのデータ通信ができたという話題で盛り上がっていました。そこで使用されていた通信機器は、音響カプラ[注1](**写真1**)と呼ばれる転送速度が300ボー[注2]で、1秒間にわずか約30byteのものです。通信ホストは、富士通8bitパソコンFM-7でした。「ピーガラガラ」という音声でのデータ通信でしたので、この程度のマシンの処理速度でも十分だったのです。数年後の1992年3月に筆者のクラブでもパソコン通信ホスト局を開設することになり、使用する通信機器とホスト／パソコンとをつなげるシリアル通信で苦労しました。今回は、そのシリアル通信についてお話します。

注1) スピーカーとマイクが電話機の受話器の位置にあり、その上に受話器を直接置いて、データ通信を行う通信機器。

注2) 80年代初めのパソコンは、プログラムやデータを音声データで転送していました。その通信速度はNECや日立のパソコンでは600ボー(baud)でした。このボーは1秒間の変調／復調回数を表す単位で、600ボーは1秒間に600回変復調が行われます。当時のデータレコーダは、1回の復調で1bitのデータに変換していました(ほかの機器では1回の復調が必ずしも1bitのデータになるものではない)。また、転送速度の単位bps(Bits Per Second)は1秒間に転送されるbit数を示します。データレコーダの転送速度のボーとbpsが一致することになり、この2つの単位が混乱して使用されているときもありました。本稿では厳密さを求めないので、bpsとして説明します。

パソコン通信ホストシステムの構築

パソコンで通信を行うためにアナログモデム(以下、モデム)を使っていました。80年代は、音響カプラと同じ300bpsのモデムから1,200bps程度のモデムが流通しており、通話用の電話網でデータを送受信していました。受信しているテキストを画面に表示すると、リアルタイムで読める程度の速度でした。

ホスト局を開設しようした1992年ごろは、草の根BBS[注3]としては後発でしたので、機器も進歩しており、高速な通信と通信品質の高さを実現するために、次のような環境を構築しました。

- 9,600bpsのモデムを使用
- モデムとパソコンとの転送スピード(DTEスピード[注4])をモデム間スピードの2倍以上に設定
- ISDN 2回線の高品質回線を利用

注3) 個人やグループが 運営していた小規模なパソコン通信ホスト局　http://zob.club/zobst/intro/bbshist.htm

当時のモデムは、2,400bpsのものが広く使われていたのですが、高速化を目指して、発売されたばかりの高価な9,600bpsの高速モデムを設置しました。当初の300bpsの32倍になりましたが、問題も出てきました。

RS-232Cの転送速度

現在のシリアル通信の規格はI^2CやSPI、Serial-ATAなど多種多様ですが、当時のパソコンにおけるシリアル通信といえばRS-232Cを指しました。

9,600bpsモデムの外部との転送速度の上限は9,600bpsですが、パソコン間のDTEスピードは、それ以上を要求しました。当時の最新モデムでは、MNP5/V.42bisと呼ばれる通信プロトコルが利用でき、データを圧縮

注4) DTE(Data Terminal Equipment)。実際に通信を行う機器、外部ネットワークなどと接続するモデムなどの装置はDCE。

して通信ができました。とくにテキストの送受信は、圧縮率が高くなるため、高速なDTEスピードが要求されたのです。

ホストで使用したパソコンは、NECのPC-9801RL（80386 DX 20MHz）で当時の国産パソコンとしては最高速に分類されるものでしたが、通信速度の上限は9,600bps（調歩同期式[注5]）だったため、そのままでは、それを超える速度を実現できません。また、PC-8001のころから継続してIntel 8251互換IC（以下 i8251）が搭載されていたため、高速化の処理に問題がありました。i8251は、与えるクロックによって高速な通信ができるのですが、その処理は1byteごとで、1byte分のbitデータが届いた時点で割り込み処理をするか、監視ループでデータ処理をするしか方法がありませんでした。この方法ではCPUに大きな負担がかかり、短時間に連続した多量のデータが来た場合、処理できなくなります。そこでRS-232Cのハードウェアフロー制御によってモデムに転送を待ってもらうことになります。そのため、シリアル通信だけなら2本程度（Rx/Tx）の信号線で済むはずなのに、ハードウェアフロー制御用の信号線（RTS/CTS、DTR/DSR）も重要だったのです。

拡張RS-232Cボード

2回線のホストの計画でしたが、ホストマシンには、RS-232Cポートが1つしかないので、拡張RS-232Cが必要です。

そこでPC-98の拡張スロットに2つの接続端子を持つRS-232Cボードを取り付け運用を始めました。用意したRS-232Cボードは、9,600bpsを超える設定が可能だったのでDTEスピードを19,200bpsにしました。開設当初は、接続する相手側も9,600bpsモデムを持つメンバが少なかったので、ほとんど問題はなかったのですが、会員数が増え、高速なモデムの接続が多くなると、フロー制御が頻繁に行われる状況になりました。

この問題が出てきたときタイミングよく、シリアル通信を高速に処理できるNS16550A[注6]を使ったRS-232Cボードが発売されました。NS16550Aは、16 byte FIFO[注7]を持ち、割り込み頻度をi8251に対して1/16にできたので、ホストの負荷が大きく低減されました。

開設から1年後の1993年には会員の増加に合わせてISDNをさらに1本導入して、全4回線にしました。モデムもV.32bis規格の14,400bpsに新調し、それに併せてNS16550AのRS-232Cボードを使用することでDTEスピード 38,400bpsを実現しました。

その後のシリアル通信

その後1994年に、8回線に、28,800bpsのモデムを導入し、DTEスピードを57,600bpsに、95年には、高速で安定動作を行うことができるUSRobotics COURIER V.34に変更し、DTEスピードは遂に通常のRS-232C最高速度と考えられている115.2Kbpsにできました。このスピードは、目指していた最高速度でしたので、達成できたときは、多くのパソコン通信メンバと大規模な記念オフ会を開いたことが思い出されます。

しかしその時期、パソコンで一般的に使えるシリアル通信は、RS-232CからEthernetへ移っていっていたのです。10BASE/Ethernetによるデータ転送速度は、標準で10Mbpsであり、苦労して達成した115.2Kbpsとは約100倍違うのです。Netware Lite[注8]でパソコン通信のホストパソコンと接続したところ、巨大なサイズと思っていた通信ログがあっという間に転送でき、大きな技術の飛躍を感じました。1995年にはWindows 95も発売され、パソコン通信もインターネットの時代へと変わっていったのです。SD

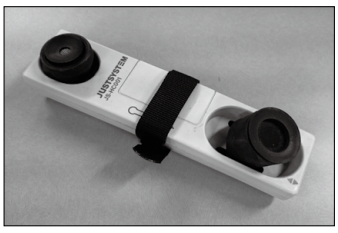
注5） データそのものに同期用信号を追加したもの。＝非同期式。

注6） NSはNational Semiconductor。

注7） First In, First Out。先に入れたものを先に取り出す、格納データの処理方法。

注8） NetWare Lite。ノベル社が開発・販売した、MS-DOS上で動くピア・ツー・ピア型のネットワークシステム。

▼写真1　音響カプラ



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。

開発のボトルネックはどこだ?

迷えるマネージャのための プロジェクト管理ツール再入門

第10回 テストにもっと光を! 言うは易く行うは難し。テストを改善しよう!

Author リックソフト㈱ 廣田 隆之(ひろた たかゆき)、網野 勉(あみの つとむ)、大塚 和彦(おおつか かずひこ)

みなさんはソフトウェア開発においてどのようなテストを実施されているでしょうか。単体テスト、結合テスト、総合テストなど、さまざまなテストを実施されていることと思います。では、これらのテストをどのように計画・実行し、管理されているでしょうか。

ソフトウェアの品質を向上・維持していくうえで、テストは非常に重要です。今回は、その重要性が指摘されながら、あまり取り上げられることがなく、開発の花形である設計やプログラミングに比べて少々地味な役回りのテストについてお話したいと思います。

現状の課題

昨今のソフトウェア開発環境はめまぐるしく進化を続けており、開発サイクルはますます早くなっています。便利で使いやすいライブラリやフレームワーク、ビルドを自動化する継続的インテグレーション(CI)など、有償・無償を問わず、さまざまなツールやサービスが提供され、開発者にとってたいへん便利な時代になりました。

ソフトウェアの開発サイクルが短くなる中で、テストの位置付けはどうでしょうか。テスト駆動開発(TDD)やCIを活用したテストの自動化など、高品質なソフトウェアを作るしくみは整備されてきました。テストの実行はすべてCIに任せているよ、という方々もいらっしゃると思います。しかし、納期とコストに追われた開発現場において、テストのすべてを自動化するのはとても困難であり、日々悩みを抱えている開発リーダーも多いのではないでしょうか。

TDDやCIが開発現場で広く受け入れられているのは、高品質なソフトウェアを作りたいというニーズの高まりであり、テストの重要性が認知されつつあるのは喜ばしいことです。しかしながら、単体テストはxUnitで自動化、それ以降のテストは手動で行う、といった現場が多いことも事実です。

課題は手動のテストがどのように実施されているかということです。おそらく一番多い手法は、Excelなどの表計算ソフトを使う方法だと思います。テスト項目表に従ってテストを実施し、不具合が見つかったら障害票を起票し、開発者による修正を経て、再度テストを実施するという流れです。障害票の起票には、Excelによる課題管理表、またはJIRAやRedmineなどの課題管理システムがよく使われます。

図1は、Excelによるテスト項目表の例です。テスト手順、確認項目、結果、日付、担当者などの項目が並んでいます。プロジェクトによっ

▼図1 Excelによるテスト項目表の例

番号	大分類	中分類	テスト手順	確認項目	結果	日付	担当者	備考
1	ログ	起動時	アプリケーションログを確認する。	起動時のログが出力されている。	×	2015/9/30	Aさん	SYS-12
2	コンテナ設定	文字コード	設定ファイルを確認する。	文字コードがUTF-8になっている。	○	2015/9/30	Aさん	
3		バッファサイズ	設定ファイルを確認する。	バッファサイズが512MBに設定されている。	○	2015/9/30	Aさん	
4	Java設定	ヒープサイズ	起動後にpsコマンドを確認する。	ヒープサイズが2048MBに設定されていること。	×	2015/10/2	Bさん	SYS-15
5		システムプロパティ	psコマンドを確認する。	仕様書通りにプロパティが設定されていること。	○	2015/10/2	Bさん	

て多少の違いはあるものの、おそらく似たような形式になるのではないでしょうか。

Excelによるテスト項目表は長年利用されており、メリットも多くあります。たとえば、学習コストの低さです。特別なトレーニングを受けなくても、テスト項目の作成や実施ができるようになるはずです。自由度の高さもExcelならではでしょう。列を追加したり、シートをコピー&ペーストで増やしたり、プロジェクトのニーズに合わせて自由に拡張できます。

しかし、この方法によるテストでは次のような課題・問題点があるように思います。

・テスト結果の集計が難しい

筆者の実体験では、Excelシートにマクロを埋め込み、件数をバッチで集計するプロジェクトもありましたが、テストの進捗をリアルタイムで計るのはなかなか骨の折れる作業です。件数を手でカウントしているような場合は、常に最新の進捗状態を把握するのは至難の業です。

・テスト項目と障害票の紐付けが手間

テストをすれば何らかの不具合が見つかります。そのとき、テスト項目表に障害票の番号を手動で記入していませんか？

・テスト項目とエビデンスの紐付けがめんどう

テストの証跡として、または不具合発生時の再現テスト用に画面ショット、ログファイル、設定ファイルなどをファイルサーバにコピーする光景をよく目にします。フォルダ管理をきっちりしないと、ファイルが迷子になったり、テスト項目との紐付けで悩むことが多くあります。

・どのバージョンでテストしたのかわからなくなることがある

テスト対象のシステムやプログラムのバージョン管理は労力を要します。テスト担当者に話を聞いてみたら、最新版ではないバージョンでテストを実施していた、なんてことも。

・テスト担当者、開発者、管理者のコミュニケーションが取りづらい

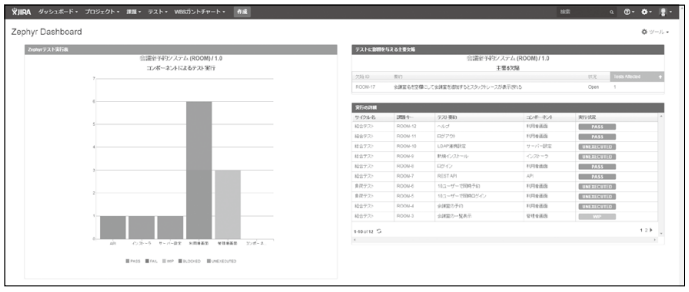
テストで何らかの問題が発生し、開発者による修正が入る場合、テスト→修正→テスト→修正→……といったサイクルをうまく回すには、Excelによる管理では困難を伴います。不具合が増え、納期に追われるとテストの工程はだんだんと厳しくつらいものになっていきます。

こうしてみれば?

これまでの連載でも、Excelを使ったプロジェクト管理をJIRA(Atlassian社の課題管理システム)へ置き換える提言をしてきました。テストの管理においてもテスト管理ツールの活用を考えてみてはいかがでしょうか。

今回ご紹介するツールは、Zephyr for - JIRA Test Management(以下、Zephyr)です。Zephyrは米国のZephyr社が開発しているJIRA向けのアドオンです。このツールを導入すると、テスト項目の作成やテスト結果の記入をWebブラウザ上でできるようになります。JIRAはサーバで動作する

▼図2　テストの進捗状況が把握できるダッシュボード



Webアプリケーションなので、複数人の同時編集も可能になりますし、テスト内容の検索やレポート出力も簡単です。

テストの実施状況(何%完了して、不具合が何件発生しているのか)といったリアルタイムな情報も、常に最新の状態をJIRAのダッシュボードに表示しておくことができます(**図2**)。

Zephyr流のテスト

では、Zephyrを使ったテストのやり方を俯瞰してみましょう。まずは、テスト項目を作成します。これはJIRAにチケットを起票するのとほぼ同じ方法です(**図3**)。テストに記述する内容は、Excelシートを使ったテストと大きく変わりません。テスト項目、テストデータ、期待する結果を書いていきます。

テスト項目を作成したら、次はテストの実施です。Zephyrはテストの規模に応じてさまざまなプロジェクトで活用できるように設計されています。

Zephyrにはテストサイクルという考え方があります(**図4**)。テストサイクルはいくつかのテストをグループ分けできるものです。結合テストや総合テストといった単位でテストサイクルを定義するのが一般的な使い方です。

テストを実施するときは、Zephyrのテスト項目を順番にテストしていき、成功すればPASS、失敗すればFAILにステータスを変えていきます(**図5**)。

テスト担当者やテスト実行日時といった情報は自動的に登録されていきます。テスト件数の集計は自動で行われるため、テスト担当者はテストの実施に集中でき、テスト管理者はZephyrの画面を定期的にチェックすることで、いつでも最新のテスト実施状況を確認できます。

不具合を見つけたときは、テスト項目と同じ画面からJIRAのチケットを起票すれば、テスト項目と不具合チケットの関連付けも自動的に行われます(**図6**)。

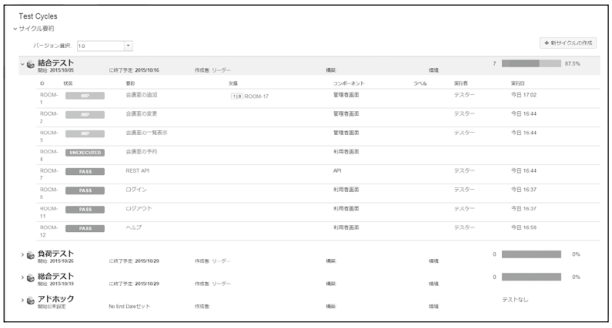
画面ショットやログファイルの添付が必要なときは、JIRAの標準機能でファイルを添付します。テスト担当者はテストの実施、不具合チケットの起票、再テストといった作業をすべてJIRA上ででき、追跡も簡単ですので、ストレスフリーな楽しい(?)テスト生活を送ることができます。

開発者にとっては、どのようなテスト手順によってバグが発生したのかがわかるので、バグを再現するためにテスト担当者に何度もヒアリングすることがなくなると期待できます。JIRAのファミリー製品であるBitbucket Server(Gitリポジトリ)やBamboo(CIサーバ)を併用していれば、開発→ビルド→テストの追跡が一気通貫で可能になるので、管理者やリーダーにとって何よりの安心感を得られるはずです。

その他の便利な機能として、既存のExcelファイルをインポートしてテストケースを作成したり、テストケースをエクスポートしてExcelで

▼図3　テスト項目の作成

▼図4　Zephyrのテストサイクル



開いたりすることも可能です。また、テストサイクルを複製すれば、これまで作成・実施したテストケースを簡単に再利用できます。

Zephyrのテストケースは JIRAのチケットそのものなので、JIRAでできること(検索、印刷、コメントなど)は基本的に何でも可能です。アイデア次第でさまざまな活用方法が考えられると思います。

現場の声

Zephyrは、ユーザから次の点が評価されています。

・結果の履歴が管理できる
・テストサイクルを利用して、グループ的に管理できる
・インポートツールによりExcelファイルからテストデータを登録できる
・1つのテストケース(JIRAのチケット)を複数のテストサイクルに利用でき、Excelのようにファイルが増えない
・テスト中に不具合が見つかったときは、Zephyrの画面からJIRAの不具合チケットを簡単に起票でき、自動的にリンクも設定される
・テストの進捗状況をリアルタイムで確認できる

テストに多くのメリットをもたらしてくれるZephyrですが、欠点や課題もあります。1つは学習コストです。直感的に使えるツールですが、操作性などが洗練されていくのはこれからかもしれません。今後に期待したいところです。2つめは価格です。JIRAからそろえる場合、必要なユーザ数分のライセンスとなると、それなりの費用になります。とはいえ、小規模なチーム向けには安価なライセンスも用意されており、評価版も利用できるので、興味を持たれた方はまず試していただきたいと思います。

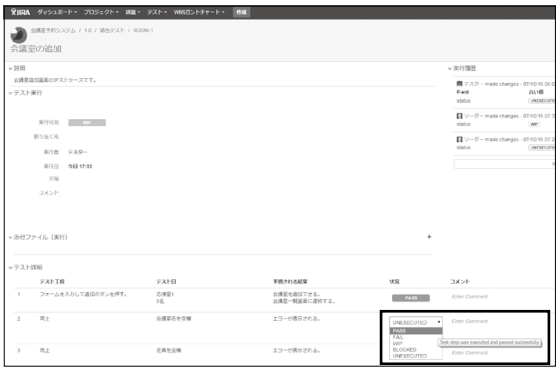
その他、ユーザからは次のような意見も寄せられています。

・Webアプリケーションのため、ネットワークにつながらない環境などで運用が難しい場合がある
・日本語訳がおかしいところがある

おわりに

いかがでしたでしょうか。テスト管理ツールはこれまであまり重要視されてこなかったように感じます。しかし、品質の良いソフトウェアを作るためには重要なツールであり、取り組む価値のあるしくみではないでしょうか。今回の記事が、テスト工程について何らかの課題を抱えている読者のみなさんにとって少しでもヒントになれば光栄です。SD

▼図5　テストを実施し、テストのステータスを変えていく



▼図6　テスト項目と不具合チケットの関連付けは自動的に行われる

Hardware

F5ネットワークス、アプリケーションデリバリコントローラ製品の最新バージョン「BIG-IP 12.0」を発表

F5ネットワークスジャパン合同会社は、アプリケーションデリバリコントローラ（ADC）のアプライアンス製品である「BIG-IP」製品群の最新メジャーリリースとなる「BIG-IP 12.0」を発表、国内提供を開始した。

BIG-IPは、ロードバランサやファイアウォールなど、アプリのスピードの速さ、高いセキュリティと可用性を保証するために必要なさまざまな機能を提供する。最新版の特徴は、「クラウド対応の強化」「セキュリティの強化」「HTTP/2への正式対応」の3点だ。

・クラウド対応の強化

これまではCisco ACI、VMware NSX、OpenStack、VMware vCloud Air、Amazon Web Servicesに対応してきたが、新たにMicrosoft Azureにも対応。

・セキュリティの強化

SSO（シングルサインオン）、DDoS攻撃防御がそれぞれ強化された。

・HTTP/2の正式サポート

2014年8月より限定リリースの形で提供を開始していたが、今回HTTP/2対応機能を正式にリリースした。

CONTACT
F5ネットワークスジャパン合同会社
URL https://f5.com/jp

Report

U-22プログラミングコンテスト、最終審査発表回

10月4日、秋葉原UDX（東京都千代田区）にて、U-22プログラミングコンテスト最終審査発表会が行われた。

U-22プログラミングコンテストでは、「プロをうならせるアイデアと技術」をテーマに、22歳以下の若者が開発したオリジナルのコンピュータプログラミング作品が募集された。7月から作品の募集が始まり、最終審査会では、事前審査・1次審査を通過した約20作品がそろい、開発者自らが作品のプレゼンテーションを行った。

「経済産業大臣賞」に輝いたのは次の4作品。

『allergy』

中馬 慎之祐さん（成蹊小学校）

“外食中のアレルギー食品の誤食”を防ぐことを目的としたアプリ。開発者の中馬さんも卵アレルギーに悩む1人で、言葉の通じない海外での食事において、自分のアレルギーをどう伝えればいいか、という問題提起から開発が始まった。しくみとしては、スマートフォンから言語と（7ヵ国語）自分のアレルゲン（9つ）を選択すると、「自分のアレルギーは○○です、この料理は食べられますか？」というメッセージを選択した言語で表示させる、というもの。まだ小学校高学年の中馬さんだが、本アプリはSwiftを使って開発したそうだ。

▲プレゼン中の中馬 慎之祐さん

『Streeem』

清水 大輝さん（国立米子工業高等専門学校）

「キーワード」に着目したニュースキュレーションアプリ。話題になっているキーワードを使って検索すると、それがどのメディアで話題になっているのかを知ることができる。本アプリは、電子書籍やメールで気になったワードも、ウィジェットから検索できる。

『すまっとシューター』

佐伯 星哉さん、眞鍋 孝明さん、板本 佑磨さん、山田 航己さん（河原電子ビジネス専門学校）

複数人でプレイできる、Webアプリのシューティングゲーム。プレーヤーたちはスマートフォンをコントローラにして、PCからアクセスしたURLの1つの画面を見ながら操作する。スマホゲームでありながら、複数人で場を共有する楽しさを目指した作品である。

『Recture～復習しやすい授業記録アプリ～』

藤坂 祐史さん（筑波大学）

授業でのノート取りを補助する録音アプリ。授業を録音している間、気になった個所にタグやメモを付けることができる。録音した音声は、付与したタグの位置から頭出しで再生できる。

CONTACT
U-22プログラミングコンテスト
URL http://www.u22procon.com

Software

グレープシティ、1言語クロスプラットフォーム開発対応型統合開発環境「Xojo」の国内販売を開始

グレープシティは10月6日、米Xojo社が開発・販売する統合開発環境「Xojo（ゾージョー）」の提供を開始した。

Xojoは「課題解決のためのアプリケーションをすばやく誰でも簡単に開発できること」を目的としており、OSもフレームワークも超えて統一された言語体系とプログラミングインターフェースを持つ。WindowsやMac、iOS、Linuxといった異なるOSのネイティブアプリに加え、Webアプリまで1つの言語で開発できる。どの環境でも同じ言語体系・プログラミングインターフェースなので、OSやプラットフォームをまたいだプロジェクトファイルを共有できる。

言語としては、可読性に優れ、初心者でも理解しやすいBasicを採用、Visual Basicの経験者であればすぐに使いこなせる。

開発したアプリは、それぞれのOSの機械語コードにコンパイルされる。ネイティブで実行されるため、高速で堅牢かつUIの自然さを兼ね備えたアプリを実現できる。

アプリの設計からデバッグまではすべて無料だが、作成したアプリを配布するためにビルドしたい場合や、開発中に日本語技術サポートを受けたい場合にはライセンスが必要となる。

CONTACT
グレープシティ㈱ URL http://www.grapecity.com

Event

「第10回　日本OSS貢献者賞・日本OSS奨励賞」受賞式

10月24日、「OSC 2015 Tokyo/Fall」にて、「日本OSS貢献者賞」「日本OSS奨励賞」の授賞式が行われた。

今年の日本OSS貢献者賞は次の4名（敬称略）。

- 奥 一穂：世界最速のHTTP/2サーバ「H2O」を開発
- 亀澤 寛之：仮想コンテナ技術の核「cgroups」を開発
- 古橋 貞之：「MessagePack」「fluentd」を開発
- 岡島 順治郎：レイヤファイルシステム「aufs」を開発

日本OSS奨励賞は、次の9名と1団体。

- 鯵坂 明：「Apache Hadoop」のコミッタ
- 江木 聡志：プログラミング言語「Egison」を開発
- 榎 真治：「LibreOffice」の普及に貢献
- 奥村 隆一：「YUI」「YUIDoc」「FormatJS」の開発に寄与
- 猿田 浩輔：「Apache Spark」の品質向上・機能追加
- 末永 恭正：「HeapStats」を発者
- 細田 真道：楽譜作成プログラム「LilyPond」のコミッタ
- 宮下 剛輔：「Serverspec」を開発
- 吉田 真也：REPLツール「Kulla」のコミッタとして活躍
- 国土地理院 情報普及課：「地理院タイル」「地理院地図」をそれぞれオープンソースで公開

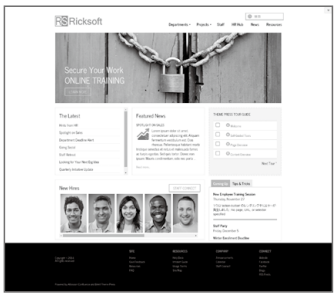
CONTACT
オープンソースカンファレンス 2015 TOKYO/Fall
URL http://www.ospn.jp/osc2015-fall

Service

リックソフト、「Confluence」と「ThemePress」を組み合わせたエンゲージメント情報ポータルを提供開始

リックソフトは10月20日、顧客エンゲージメント[注1]、従業員エンゲージメントの向上を支援するための、アトラシアンのエンタープライズ向け情報ナレッジ共有ツール「Confluence」と、そのリデザインツールである「ThemePress」を組み合わせたエンゲージメント情報ポータルの提供を開始した。

Confluenceはヤフー㈱や㈱インターネットイニシアティブなど、大手企業に導入実績のある強力な情報ナレッジ共有ツール。また、ThemePressはデザインテンプレートを適用するだけで簡単にWebサイトをリデザインできるツール。この2つを組み合わせることで、エンゲージメント向上目的のWebサイトを簡単に作成できる。同社は、本情報ポータルを、初年度で10社以上に提供することを目標としている。



▲理念・方針・活動が伝わるデザイン例

CONTACT
リックソフト㈱ URL http://www.ricksoft.jp

注1）組織に対する愛着心、思い入れ。

Event

翔泳社、「ITエンジニアに読んでほしい！技術書・ビジネス書大賞 2016」募集開始

「ITエンジニアに読んでほしい！技術書・ビジネス書大賞」は、一般投票・審査員投票によってITエンジニア向けの技術書・ビジネス書大賞を選ぶイベント。2014年、2015年に続いて、2016年が第3回目の開催となる。授賞式は、2016年2月に行われる「Developers Summit 2016」にて行われる。

本大賞の対象書籍は、技術書、ビジネス書全般（刊行年は問わない）。過去の大賞『GitHub実践入門』『「納品」をなくせばうまくいく』（2015年）、『リーダブルコード』『小さなチーム、大きな仕事 完全版』（2014年）の4冊は殿堂入りとし、選考から除外される。選考方法として、一次投票は一般読者からのWeb投票（**http://www.shoeisha.co.jp/campaign/award**）で、最終投票は、特別ゲスト3名＋観覧席の参加者による投票となる。投票の多かった計6冊の書籍の著者、または編集者が書籍紹介のプレゼンを行ったうえで、各賞が選ばれる。

●スケジュール

2015年11月9日～2016年1月12日	サイトオープン、一次投票受付
2016年1月18日（予定）	技術書・ビジネス書ベスト10（一次投票結果）発表
2016年2月18日（木）	プレゼン大会、最終投票＆表彰イベント開催
2016年3月～	書店でのフェア展開

CONTACT
㈱翔泳社　URL http://www.shoeisha.co.jp

Service

スイッチサイエンス、基板製造サービス「スイッチサイエンスPCB」を提供開始

㈱スイッチサイエンスは10月30日、個人向けにプリント基板を製造するサービス「スイッチサイエンスPCB」の提供を正式に開始した。

スイッチサイエンスPCBは、プリント基板を安価に製造する個人向けのサービス。注文は同社のWebページ（**https://www.switch-science.com/pcborder**）から行う。基板はSeeed Technology Limited.が委託している中国の工場で製造され、入金確認後14日～21日で自宅へ発送される。窓口はスイッチサイエンスが担当するので、ユーザは注文、支払い、問い合わせなどをすべて日本語で行える。プリント基板は緑色の2層基板の場合、1,389円／10枚（送料別）から注文できる。4層基板や基板色の変更も可能。

●注文例

基盤サイズ	5cm × 5cm	10cm × 10cm
表面処理	半田レベラー	半田レベラー
レジスト色	緑	緑
面付け枚数	1	1
銅箔厚	1 oz.	1 oz.
層数	2	2
枚数	10	10
基板の厚み	1.6mm	1.6mm
通常価格（送料込）	2,469円（税込）	4,154円（税込）

CONTACT
㈱スイッチサイエンス　URL https://www.switch-science.com

Service

PSソリューションズ、農業IoTソリューション「e-kakashi」を販売開始

PSソリューションズ㈱は10月14日、可視化した農業データを活用して栽培手法や知見を共有する農業IoTソリューション「e-kakashi」を販売開始した（サービス提供開始は12月下旬）。おもな販売対象は、国内の営農支援を行う自治体・農業協同組合・企業など。

e-kakashiでは、圃場（ほじょう）の温湿度や日射量、土壌内の温度や水分量、CO^2などを計測できる各種センサーを搭載する子機からデータを収集し、通信モジュールを内蔵した親機を経由して、クラウド上で収集データを管理できる。ユーザは、PCやタブレット、スマートフォンなどから栽培時に必要となるさまざまなデータを参照できるほか、収集データは栽培指導や農作業の品質管理・効率化に役立てることができる。子機ー親機間はアドホックな通信、親機ークラウド間は3G/LTEで通信する。

参考価格は、1親機1台、子機1台の場合、【機器代金：最低価格749,600 円～】＋【月額利用料：最低価格7,980円～】（税抜）。

▲センサなどが搭載された子機

CONTACT
PSソリューションズ㈱　URL https://www.pssol.co.jp

ひみつのLinux通信

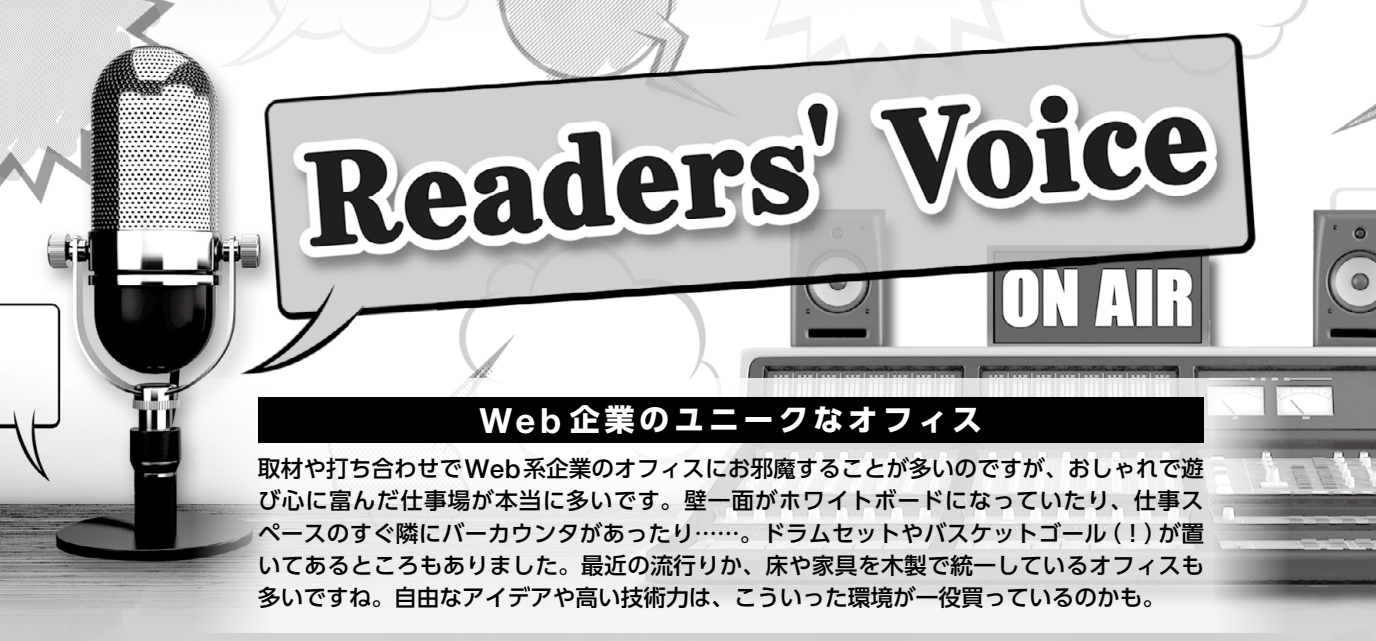
作)くつなりょうすけ
@ryosuke927

第23回 数字が見える！

売れる本と売れない本の違いが見えると豪語する某編集長ですが、彼も死神さんと契約したんでしょうかね（笑。

to be Continued

永久機関は存在しませんので、モノを使ってるといつか訪れるソレ。初期不良、有効期限、欠品、経年劣化、インクの乾燥、製造終了、サポートの終了……。愛用してたそれと無情にも引き離される、突然振りかかるメンテナンス……まさにショッギョ・ムッジョ！　有効期限とサポート終了は調べればわかることなので対策は可能ですが、故障は対策しにくいので結局サポートを契約することになったり。この社会は誰かのメンテナンスで成り立っていると思うしかない。S.M.A.R.Tで見てても壊れるときは突然やって来ますしね。そんなわけで、悪魔がくれる特殊能力みたいなのっていつまで経っても憧れますねぇ。（永遠の厨二病）



Web企業のユニークなオフィス

取材や打ち合わせでWeb系企業のオフィスにお邪魔することが多いのですが、おしゃれで遊び心に富んだ仕事場が本当に多いです。壁一面がホワイトボードになっていたり、仕事スペースのすぐ隣にバーカウンタがあったり……。ドラムセットやバスケットゴール(!)が置いてあるところもありました。最近の流行りか、床や家具を木製で統一しているオフィスも多いですね。自由なアイデアや高い技術力は、こういった環境が一役買っているのかも。

2015年10月号について、たくさんの声が届きました。

第1特集　攻撃に強いネットワークの作り方

サイバー攻撃の検出・防御を、汎用のサーバ＋OSSで実現する方法について解説した特集。「Interop Tokyo」の「Show Net」プロジェクトにかかわったメンバが、そこで培われたセキュリティ技術を存分に披露しました。

セキュリティは、アプリケーションや人任せなので勉強になりました。
食欲より睡眠欲が強しさん／埼玉県

復習にとても良い教材です。
とーふやさん／神奈川県

具体的な対策や高度な内容もあり、たいへん参考になった。　psiさん／東京都

ネットワークインシデントと攻撃の防御が役に立った。　raiさん／東京都

本格的なセキュリティ対策には、やはりアプライアンス製品を導入するのがベターですが、OSSを使って手元でセキュリティ対策を行うことで、どのようなしくみで攻撃を防いでいるのか、実際に目で見ながら学ぶことができます。本特集には、たいへん勉強になったという声を多くいただきました。

第2特集　Webメールの教科書

本誌9月号の「メールシステムの教科書」に続いて、今回はブラウザから利用できるWebメールに関する特集でした。クラウドサービス(Yahoo!メール)と自社構築(Roundcube)、両方の利点を挙げながら、Webメールについて再考しました。

Webメール特集は身近な話題でおもしろかったです。何気なく使っていましたが、動作環境のバリエーションが多いにもかかわらず、それらに対応していることに感心しました。
ReiLLさん／東京都

Webメールの環境作りにチャレンジしたい。　よっきーさん／大阪府

RoundcubeというWebメールサーバはなかなか良さげな感じだったので、導入してみたい。　大沼さん／群馬県

サーバの環境構築に関して冪(べき)とう性の話題が出ていた。自社の運用でも取り入れる必要性を強く感じた。メールの社会インフラ化はまったく同感。使えなくなると何もできなくなってしまう。障害対応の判断、対応は相当たいへんだと推察します。　隼さん／岩手県

もはや社会インフラとも言えるメールサービス。当たり前のように使われている裏では、並々ならぬエンジニアの苦労、工夫がありました。特集で紹介したRoundcubeに興味を示された読者も多いようです。

特別付録　Vim & Emacsチートシート

本誌発行300号を記念して制作したVimとEmacsのチートシート。mattnさん&るびきちさんという、それぞれのエディタの超ベテランユーザが監修した渾身の1枚です。

使用頻度が低いものは「あれ、なんだったっけ？」と検索していましたが、チートシートのおかげでほぼパッと見つかるようになりました！
ewiad420さん／神奈川県

A4両面にまとまっていて、とても使いやすいです。たまにしかエディタを使わない人にも布教しました。
齋藤さん／神奈川県

チートシートの電子版は本誌のサポートページからダウンロードで

10月号のプレゼント当選者は、次の皆さまです

①「Raspberry Pi 2 Model B」&「IR Camera module」セット
春田隆佑様（埼玉県）

②ウイルスバスタークラウド 10
坂口美郁様（熊本県）

③Fedora Tシャツ
キャボさん様（埼玉県）、角田学様（東京都）、羽根田歩夢様（滋賀県）、inu様（岡山県）、青木克憲様（愛知県）、石澤景子様（埼玉県）、田代勝久様（埼玉県）、山田慶行様（埼玉県）、澤下夏実様（大阪府）、檜垣賢様（大阪府）

④『The Art of Computer Programming Volume 2』
西一美様（東京都）、大平圭佑様（東京都）

⑤『ヘルシープログラマ』
松山高明様（大阪府）、田代海霞様（福岡県）

⑥『ルーティング＆スイッチング標準ハンドブック』
金谷直樹様（埼玉県）、vi使って25年様（京都府）

⑦『ビッグデータ活用の常識は今すぐ捨てなさい』
浜田佳明様（兵庫県）、小林淳一様（東京都）

※当選しているにもかかわらず、本誌発売日から1ヵ月経ってもプレゼントが届かない場合は、編集部（sd@gihyo.co.jp）までご連絡ください。アカウント登録の不備（本名が記入されていない場合、プレゼント応募後に住所が変更されている場合など）によって、お届けできないことがあります。2ヵ月以上、連絡がとれない場合は、再抽選させていただくことがあります。

きますので、まだの方はぜひご活用ください！

短期連載　OS Xクライアント管理ツール「munki」【前編】

Windowsに比べると、これぞといった手段がないMacのクライアント管理。そんな中「munki」は"手軽な導入"と"Googleによる開発"で注目を集めるOS Xクライアント管理ツールです。本短期連載では前後編で「munki」の導入・運用方法を解説します。

タブレットも多く利用しているため、iOSの管理ツールも知りたいです。
橿山さん／埼玉県

Macの管理ツールがあったことを知らなかったので、使える場面が来たら使います。　クラウドさん／京都府

Macの管理って面倒なところがあるのですが、良い情報をいただきました。
鈴木さん／熊本県

「これはいいツールを見つけた」という声が多く寄せられました。IBMが企業向けのMac導入支援を始めたそうですが、小・中規模の組織ですとこの「munki」を使ったほうが、コストも低く済み、良いかもしれません。

短期連載　モダンなJavaアプリケーション開発【3】

「コードの修正のたびに、自動的にテストを実行してアプリの品質を保つ」。Javaを使って、そんな"イマドキ"の運用を実現する短期連載です。第3回では、継続的インテグレーションを実現する「Travis CI」を取り上げました。

継続的インテグレーションTravis CIは知らなかった。進化している。
林さん／愛知県

継続的インテグレーションについて詳しく知れて良かった。　村橋さん／北海道

"継続的"に管理していくことのたいへんさは痛感しています。CIサービスを検討する時期に来ているのかもしれません。　NGC2068さん／愛知県

新しい開発手法の導入は、ただツールを入れればいいというわけではなく、開発メンバ、あるいは組織全体で同意を得ないといけません。この手法にはこういう利点がある、と的確に説明できる知識が必要ですね。

短期連載　Jamesのセキュリティレッスン【5】

新しいファイル形式「pcap-ng」も使えるようになったパケットキャプチャのツール「Wireshark」の使い方を紹介する短期連載です。今回はパケットの絞り込みに便利な「ディスプレイフィルタ」の使い方を詳しく見ていきました。

内容が住境に入ってきておもしろかったです。　オミオさん／宮城県

Wiresharkを普段使用しているので参考になりました。
サファイアさん／茨城県

GUIで手軽にパケットキャプチャができるWireshark、普段から使っている読者の方が多いようです。膨大なパケットから目的のものを探すという、トラブルシューティングに便利な手法を学ぶことができました。

コメントを掲載させていただいた読者の方には、1,000円分のQUOカードをお送りしております。コメントは、本誌サイト**http://sd.gihyo.jp/**の「読者アンケートと資料請求」からアクセスできるアンケートにてご投稿ください。みなさまのご意見・ご感想をお待ちしています。

次号予告

Software Design
January 2016

2016年1月号
定価（本体1,220円+税）
192ページ

12月18日発売

［第1特集］チャットを使いこなしていますか?

事例にみるChatOps成功のパターン
――効率アップの秘密教えます!

数年前からWeb系企業をはじめとして浸透してきた「ChatOps」。本特集は、「ChatOps」を実践し実績を上げているさまざまな先端企業の方々に導入前と導入後の違いを解説いただき、SlackやHipChatなどのチャットツールの導入、そしてHubotなどのbotとの組み合わせ、さらにはGitHubとの連携など、具体的な方法論を紹介します!

［第2特集］プロビジョニングはお任せ!

Ansibleではじめるサーバ構成管理の省力化
新定番! 構成管理ツール最新解説

Red Hatが買収したことで、さらに注目を浴びているAnsibleを徹底解説します!

［新連載］「Android界隈 ここだけの話」

※特集・記事内容は、予告なく変更される場合があります。あらかじめご容赦ください。

お詫びと訂正

以下の記事に誤りがございました。読者のみなさま、および関係者の方々にご迷惑をおかけしたことをお詫び申し上げます。

■2015年11月号 連載「Linuxカーネル観光ガイド」
- p.179、右段、10行目
  ［誤］その後、0x400byte目まで ［正］その後、0x200byte目まで
- p.179、右段、11行目
  ［誤］0x400から0x600がentryのヘッダとなっており ［正］0x200から0x400がentryのヘッダとなっており

SD Staff Room

●現在、まとめ本の『インフラエンジニア教本2』を製作中。本誌の過去記事のうちインフラ構築にかかわるものを集めて再編集するというものだ。昨年はネットワーク周りの記事を集めて作ったが、今回はサーバ周りの話をおもにまとめてみた。発売は11月末の予定。また分厚い本になりそうです。（本）

●プログラマの使うエディタと編集者の使うそれは、必要な機能が大きく違う。エディタ環境が良ければ作業も遅滞なく進むしストレスも少ない。機械的にできる作業は自動化して、推敲に時間をかけるのが良い（＋ケアレスミスも減らしたい）。ということで自前のRubyスクリプトを秀丸に移植する日々であります。（幕）

●本誌で連載していた『Android Wearアプリ開発入門』が内容を充実させて書籍になりました。11月17日発売なので書店に並んでいるはずです。魅力的なAndroid腕時計の新商品が年末商戦に出てきますし、ハードの購入にあわせて、スマホ&腕時計のアプリ開発もはじめてみては?（キ）

●東京モーターショー2015では、自動運転技術を搭載した試作車がお披露目されたようですね。個人的に自動運転にはすごく期待しています。最近の自動車の暴走事故のニュースを聞くたびに、「もう人が運転するのはやめて、全車両を自動運転にしたほうが安全なのではなかろうか」と思います。（よし）

●最近の大発見は、会社の屋上からスカイツリーが見えること! 新宿区にある弊社オフィスですが、そこから墨田区にある建物が見えるというのは驚きですね。機会があれば、スカイツリーの高度450m地点『天望回廊』まで登りたいのですが、合計で4,000円近くかかるので足踏みしています。（な）

●久しぶりの入稿作業やそれに纏わるあれこれに戸惑いつつも、なんとかこなす日々です。止まっていた時間が動き出すように、なぜか仕事に関係ない方とも会う機会が増えています。おもに趣味方面の方や懐かしい友人ですが、多くの刺激を受けながら過ごせる日々に感謝。作品づくりに活かせますように。（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

［E-mail］
sd@gihyo.co.jp

［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。

Software Design
2015年12月号

発行日
2015年12月18日

●発行人
片岡 巌

●編集人
池本公平

●編集
金田富士男
菊池 猛
吉岡高弘
中田瑛人

●編集アシスタント
根岸真理子

●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。