special feature 1 Wi-Fi構築のすべて

special feature 2 Hadoopの本質に迫る

Software Design

2015 3

2015年3月18日発行
毎月1回18日発行
通巻359号
（発刊293号）
1985年7月1日
第三種郵便物認可
ISSN 0916-6297
定価 本体 1,220円 +税

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

カンファレンスネットワークの作り方

第一特集

YAPC, LL Diverの縁の下の力持ち!

special feature 1

ネットワーク技術者集団CONBUの無線LAN構築術

第二特集

special feature 2

いまからでも遅くない!

Hadoop超²入門

今年こそ
分散コンピューティングを
極めたい

一般記事

extra feature

第1弾
手軽に仮想化技術
Cisco VIRLで
ネットワークの
シミュレーション

第2弾
難しい技術を簡単に!
IBM Bluemix
「PaaSでIoT入門」

第3弾
コンテナ技術導入!
Snappy Ubuntu Core

how to build conference network

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

第75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。

Android Studio

Android アプリ開発の新スタンダード

「Android Studio」は、GoogleのAndroidチームが公式に提供しているAndroidアプリ開発のための統合開発環境です。2013年5月のGoogle I/Oにおいて発表され、2014年12月に初の正式版（ver. 1.0）がリリースされました。従来のAndroidアプリ開発は、EclipseにADT（Android Developer Tool）と呼ばれるプラグインを追加した環境で行うのが一般的でした。それに対してAndroid Studioは、JetBrains社による「IntelliJ IDEA」のオープンソース版をベースに開発されています。

IntelliJ IDEAはJava開発の現場ではNetBeansやEclipseと並んで高いシェアを持つ統合開発環境です。その強みはクオリティの高い開発補助機能の数々で、とくに直感的な操作性、強力なコード補完、充実したデバッグやリファクタリング機能などが高く評価されています。この操作性や開発支援機能を引き継いでいることがAndroid Studioの最大の強みということになります。

Android Studioの主な機能

Android Studioで提供される主な機能としては、次のようなものが挙げられます。

◉強力なコードエディタ

コード補完、リファクタリング、コード解析など、強力なコーディング支援機能を持った優れたコードエディタが付属しています。ソースコードや設定ファイル中の各種リソースのプレビューや設定値をエディタ上で見ることができたり、レイアウトのXMLの編集がリアルタイムにプレビューできるといった機能も備えています。

◉ Gradleベースのビルドシステム

Android StudioではGradleをベースとしたビルドシステムが採用されています。Gradleの特徴としては、GroovyベースのDSLでビルドルールを記述でき、高い柔軟性と拡張性を持っているということが挙げられます。Android Studioのビルドシステムでは、1つのプロジェクトからデバッグ版とリリース版、無料版と有料版などといった具合に複数の種類のAPKを生成できます。また、サードパーティ製のライブラリの依存関係を一貫して管理できる点も大きな強みとなります。

◉豊富なコードテンプレート

あらかじめ用意されたさまざまなコードテンプレートから、目的に応じたものを選択して利用することができます。さらに、GitHub上で提供されているサンプルコードを取り込んで利用することも可能です。

◉マルチスクリーン開発のサポート

画面サイズや画面の形状、言語、APIバージョンが異なる端末別に製作した複数のデザインのプレビューを、同じ画面内で一覧で確認することができます。また、仮想デバイスマネージャにはあらゆる画面サイズ向けのプロファイルがあらかじめ用意されています。

◉品質向上を手助けする分析機能

さまざまな分析機能によってアプリケーションの品質向上をサポートしている点もAndroid Studioの特徴の1つです。ソースコードを解析して修正すべき点やその修正方法を提示してくれるインスペクション機能、モジュールの依存性の分析、アプリ内のデータフローの分析機能、メモリの使用状況を時系列で確認することができるメモリモニターなどが提供されます。

◉ Google Cloud Platformとの連携

Google Cloud Platformとの連携をサポートするツールを備えており、Google App EngineやGoogle Cloud Endpoints、Google Cloud Messagingなどの機能を自前のアプリに簡単に組み込むことができます。

◆ ◆ ◆

Android Studioのバージョンはまだ1.0ですが、ベースとなったIntelliJ IDEAやAndroidプラグインにはすでに豊富な実績があり、そのポテンシャルは間違いのないものと言えるでしょう。晴れて正式版がリリースされたことで、Androidアプリ開発者は、早々にAndroid Studioに乗り換えるべきか、それともまだしばらくはEclipse＋ADTを使い続けるべきかという選択を迫られています。SD

Android Studio
https://developer.android.com/sdk/

【図解】コレ1枚でわかる最新ITトレンド

斎藤昌義
大越章司
渋屋隆一　著

ISBN978-4-7741-7179-1
A5判／224ページ
定価（本体1580円+税）

「これからはクラウドだ！」「我が社もビッグデータの活用を！」などと、言葉はよく聞くけど、なぜ注目されているのかわからなかったり、実態はどういうものか理解できてなかったりしませんか？
本書は、ネットをながめているだけではなかなか見えてこないITトレンドの全体像を、約100点もの図解とともにわかりやすく解説。図表はPowerPointデータとしてダウンロード可能、ロイヤリティ・フリーなので勉強会の資料や提案書の素材としてご活用いただけます。IT業界に携わる方必携の1冊！

失敗から学ぶユーザインタフェース

世界はBADUI（バッド・ユーアイ）であふれている

中村聡史　著

ISBN978-4-7741-7064-0
B5変形判／256ページ
定価（本体2980円+税）

世の中には多くの人が間違ってしまう、操作に悩んでしまうBADUI（バッドユーアイ）が溢れています。会社内で使用するエクセルのフォームからECサイトのボタンまで、すべての人がBADUIの作り手になりえます。本書では、たくさんのBADUI事例を紹介して「使いにくいことの原因は何なのか」を考察する中で、ユーザインターフェースについて興味を持ってもらい、ユーザインタフェースのトレーニングをすることで、悩んだり、困ってしまう人を少なくすることを目的としています。

直感でわかるロジカルシンキング

伊藤達夫　著

ISBN978-4-7741-7130-2
B5変形判／112ページ
定価（本体1580円+税）

『自分のアタマで考えろ！』って言われても……、どうすればいいの？
そんな悩みに、フルカラーのビジュアルとだれでも感覚的にわかる言葉で答える、まったく新しいロジカルシンキングの入門書。「なぜ、世界中でなんでも売りまくってきたのに、日本の大企業でうまく営業できないのか？」「なぜ、可愛い子なのに彼氏ができないのか？」「なぜ、TOEIC900点のイケメン学生は就活がうまくいかないのか？」といった身近な例で、論理的にわかる・考える・伝える力がだれでも身につく！

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

Contents

Software Design Mar. 2015

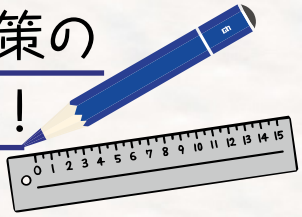
［第1特集］

CONBUの無線LAN構築術

カンファレンスネットワークの作り方

あなたを
合格へと導く
一冊があります！

効率よく学習できる
試験対策の
大定番！



山平耕作　著
A5判／736ページ
定価（本体2980円+税）
ISBN978-4-7741-6767-1

エディフィストラーニング株式会社　著
B5判／496ページ
定価（本体3200円+税）
ISBN978-4-7741-6752-7

大滝みや子、岡嶋裕史　著
A5判／736ページ
定価（本体2980円+税）
ISBN978-4-7741-6924-8

加藤昭、芦屋広太ほか　著
B5判／464ページ
定価（本体1680円+税）
ISBN978-4-7741-6925-5

大滝みや子　著
B6判／384ページ
定価（本体1480円+税）
ISBN978-4-7741-6710-7

大滝みや子　著
A5判／448ページ
定価（本体2280円+税）
ISBN978-4-7741-6112-9

岡嶋裕史　著
A5判／656ページ
定価（本体2880円+税）
ISBN978-4-7741-6937-8

エディフィストラーニング株式会社　著
B5判／392ページ
定価（本体2980円+税）
ISBN978-4-7741-6938-5

内田保男ほか　著
A5判／512ページ
定価（本体3200円+税）
ISBN978-4-7741-6101-3

大滝みや子　著
A5判／608ページ
定価（本体2980円+税）
ISBN978-4-7741-5940-9

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

Contents

OSとネットワーク、
IT環境を支えるエンジニアの総合誌

Software Design

毎月18日発売

PDF電子版
Gihyo Digital Publishingにて
販売開始

年間定期購読と電子版販売のご案内

1年購読(12回)

14,880円(税込み、送料無料)1冊あたり1,240円(6%割引)

PDF電子版の購入については

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

をご覧ください。
PDF電子版の年間定期購読も受け付けております。

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

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

Fujisan.co.jpからのお申し込み方法

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

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

Contents

Software Design Mar. 2015

3

Column

Development

OS/Network

Logo Design ロゴデザイン	デザイン集合ゼブラ+坂井 哲也
Cover Design 表紙デザイン	Re:D
Cover Photo 表紙写真	PK-Photos /gettyimages
Illustration イラスト	フクモトミホ、高野 涼香
Page Design 本文デザイン	岩井 栄子、 ごぼうデザイン事務所、 近藤 しのぶ、 SeaGrape、 安達 恵美子
	[トップスタジオデザイン室] 轟木 亜紀子、阿保 裕美、佐藤 みどり
	[BUCH+] 伊勢 歩、横山 慎昌、 森井 一三、 Re:D、 [マップス] 石田 昌治

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

技術評論社の本が電子版で読める！

電子版の最新リストは

Gihyo Digital Publishingの

サイトにて確認できます。

https://gihyo.jp/dp

GIHYO Digital Publishing

amazon.co.jp

Google play

楽天 Rakuten KC　kobo

法人などまとめてのご購入については
別途お問い合わせください。

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

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

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

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

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

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

Software Design plus

Software Design plusシリーズは、OSとネットワーク、IT環境を支えるエンジニアの総合誌『Software Design』編集部が自信を持ってお届けする書籍シリーズです。

PHPライブラリ&サンプル実践活用
WINGSプロジェクト 著
定価 2,480円+税　ISBN 978-4-7741-6566-0

アドテクノロジープロフェッショナル養成読本
養成読本編集部 編
定価 1,980円+税　ISBN 978-4-7741-6429-8

[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

GPU並列図形処理入門
乾 正知 著
定価 3,200円+税　ISBN 978-4-7741-6304-8

Zabbix統合監視徹底活用
TIS㈱ 池田 大輔 著
定価 3,500円+税　ISBN 978-4-7741-6288-1

過負荷に耐えるWebの作り方
㈱バイブドビッツ 著
定価 2,480円+税　ISBN 978-4-7741-6205-8

HTML5ハイブリッドアプリ開発[実践]入門
久保田 光則、アシアル㈱ 著
定価 2,880円+税　ISBN 978-4-7741-6211-9

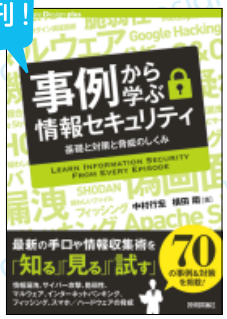
Androidライブラリ実践活用
菊田 剛 著
定価 2,480円+税　ISBN 978-4-7741-6128-0

[改訂新版]Apache Solr入門
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
定価 3,600円+税　ISBN 978-4-7741-6163-1

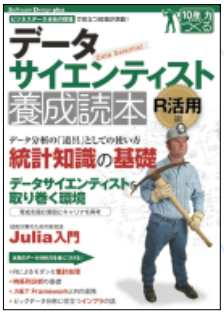
レベルアップObjective-C
沼田 哲史 著
定価 3,200円+税　ISBN 978-4-7741-6076-4

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

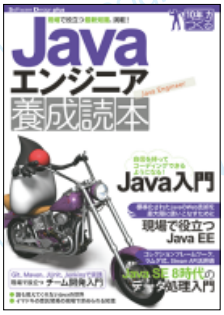
最新刊！



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



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



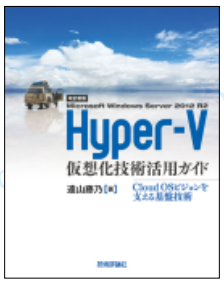
きしだ なおき、のざき ひろふみ、吉田 真也、菊田 洋一、渡辺 修司、伊賀 敏樹 著
B5判・168ページ
定価 1,980円(本体)+税
ISBN 978-4-7741-6931-6

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

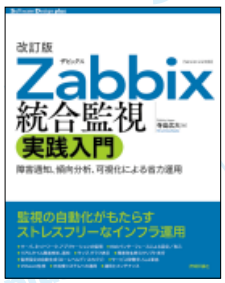
勝俣 智成、佐伯 昌樹、原田 登志 著
A5判・288ページ
定価 3,300円(本体)+税
ISBN 978-4-7741-6709-1

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

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・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・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・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

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

DIGITAL GADGET

Volume 195

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

Interaction Award 2015に見る、新しいガジェットの潮流

インタラクションの世界的アワード

インタラクションアワード（Interaction Award）は、米国IxDA（Interaction Design Association）が主催するアワードです。インタラクションとは、相互の（inter）＋効力（reaction）という意味で、一方通行の操作ではなく、“何か操作をしたら反応する”といった相互作用のことを示します。

インタラクションアワードは2012年から始まった新しいもので、世界各国から質の高い作品や製品の応募があります。とくに最近の傾向は、見た目的に優れているだけでなく、課題を明確に解決するための製品やツールが増えてきています。応募の中から優秀なものが各部門ごとに5作品ずつ最終候補（Finalists）として選出され、その中から、一般投票の結果を加味したうえで、世界各国から集まった実績のある7名の審査員による審査が行われます。

本稿執筆時点では、Webからの一般投票を受け付けている最中です。

Interaction Award 2015公式サイト
http://awards.ixda.org/2015-interaction-awards/

一般投票の受付中（2月7日まで）
http://awards.ixda.org/entries/

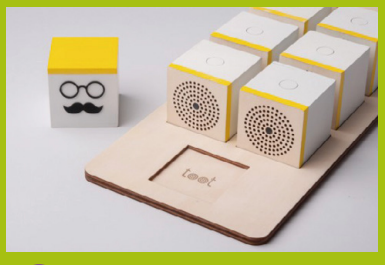
インタラクションアワードでは、その名のとおり人とモノの相互関係や、動作や操作に視点が置かれています。人の行動や行為をどれだけ誘発することができ、新しい体験や課題解決の手法が提供できたのかどうかが受賞の観点になっています。各賞は6つのカテゴリに分けられ、複数部門に入選している作品もあります。

- **Connecting（つながり）**：人と人や、人とコミュニティ間のコミュニケーションを手助けするしくみ
- **Disrupting（再構築）**：当たり前となっている既存のサービスや事柄を壊して再構築し、新しい価値を生み出す
- **Empowering（助力）**：人々が限界を超え、今までできなかったような事柄をできるように仕向ける
- **Engaging（魅了）**：日々の出来事に喜びや注目を集め、その事柄に意味をあたえる
- **Expressing（表現）**：自己表現や創造性を手助けするしくみ
- **Optimizing（最適化）**：日々の行動をより効率的に行う方法

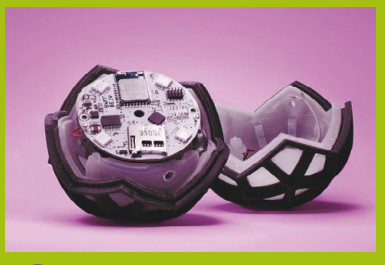
今年の傾向と作品の評価

今年のインタラクションアワードの傾向は、製品やツールなど、モノとしての単独の存在だけではなく、ネットワークやそれを使う人々の振る舞いも考慮した、人を巻き込んだデザインだということです。スマートフォンやタブレット端末が一般化し、IoT（Internet of Things：モノのインターネット）が注

Tink
子供向けIoTデバイス。温度、振動、光、動き、音などをセンサーとして使える



Toot
幼児向けデジタル楽器。
サンプリング録音した音で楽しむこともできる



Hackaball
光るボールを使ったゲームをスマホで作れるデバイス

※本記事で紹介しているものは国内未発表・未発売のものを含んでおります。

目される今、「モノと人とのインタラクティブ性は何だろう」、「今かかえている課題を解決する方法は何だろう」と、地に足がついた作品が印象に残ります。

受賞候補作が素晴らしいのはもちろんのこと、入選候補（shortlists）の作品の中にもまだ広く知られていないながらも輝くアイデアが多数ちりばめられています。ぜひ1つ1つチェックしてみてください。

インタラクションの行方 ～単なるパズルとインタラクションの違い～

相互の（inter）＋効力（reaction）を意味する「インタラクション」。一番思い浮かべやすいのは、車や自転車、バイクの運転でしょう。視覚＋聴覚＋触覚を駆使して乗り物を操作し、時にはエンジンの匂いを感じ取る嗅覚、眠気覚ましのガムの味覚なども加わるかもしれません。また、運転中のさまざまな状況は、似通ってはいるとしても、常に毎回異なる事柄であり、予測可能な要素と予測不可能な要素の集合となっているのがインタラクションのある状況です。

ハンドル操作は直接的にタイヤを操作するのではなく、左右に回す、または左右に傾けることで進行方向を決定します。パワーステアリングによって平易にハンドルを回転できるようになりましたが、それでもハンドルの重さ／軽さ、抵抗感といった細かなフィードバックで、操作範囲を調整しています。

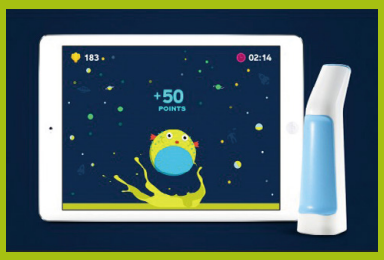
直接操作か、間接操作かということもインタラクションの重要な要素です。マウスカーソルをマウスで操作するのは間接操作で、スマートフォンで撮影した写真を指2本で拡大／縮小するのは直接操作です。どちらも自分の意思が明確に反映した、身体（この場合は指）の延長として扱えることが重要です。運転に慣れた車であれば、その車体の大きさが身体の延長として感じられることも同じです。

インタラクションのある／なしを把握する方法として、ゲームとパズルが例にあげられます。たとえば、人と対戦する将棋はインタラクションがあり、パズル的な要素を持つ詰め将棋にはインタラクションはありません。インタラクションのある道具やオモチャは何度でも楽しむことができ、そこに社会性や人間関係をも持ち込むことができる場合もあります。クロスワードパズルなどの答えがあるパズルは、一度解かれてしまうとその意味を失います。一方、ルービックキューブのようなパズルは何度も楽しめますが、それらの一見同じように見えるパズルは、毎回違う課題と違う解答方法があるわけです。

インタラクションの要素を刺激するには五感が重要です。まれに特定の感覚が鋭い人や、共感覚と呼ばれる感覚同士が協調する人がいますが、たいていの場合人の感覚は、視覚が8割を占め、聴覚が1割程度、続いてそのほかの嗅覚、触覚、味覚となります。触覚は全体の割合は少ないながらも大切な感覚で、なくなると自分という存在がよくわからなくなります。また、肌で感じる触覚にもさまざまな要素があり、形状、温度、質感、特性、硬度などを認識することができます。

インタラクションにおいて、複数の感覚に影響を及ぼすことで効果を強調する手法があります。たとえばタッチパネルのキーボード操作とともにキー

Water Watcher
水道利用量の節約のためのデバイス



Alvio
呼吸器系の病気の子供達用。
ゲームで楽しみながら吸入器を使うしくみ

Blindmaps.org
視覚障害の人が持つ杖をデジタル化し、
道案内をするしくみ

HearMe
視覚障害者向けの音や音楽の
ビートを見て理解するためのしくみ

Oyasami
睡眠のサイクルを記憶し、最適な時間に
寝るようにお勧めしてくれる逆目覚まし

Remind
アルツハイマー患者向けの、
音で記憶を呼び起こさせるデバイス

タッチ音が出たり、振動したりするという感じです。これは物理的キーを押し下げた感覚がないため、代替的な感覚としてほかの感覚に影響を与えるように考えられているのです。ハリウッドの派手なアクション映画も、もし音や音楽が貧弱であれば、その臨場感も半減してしまうでしょう。

インタラクションを評価するには、複数の感覚要素が的確に扱われているかどうかのほかに、それぞれの感覚のスピード、解像度、再現性、遅延、フィードバックなどが重要な要素です。人間はどのような感覚にもある程度慣れること、習熟することができますが、その基本は「思ったとおりに動く」という自身の身体の延長として操れるかどうかにかかっているのです。

また、センサーとしての感覚器官だけでなく、そこから受け取った情報を脳がどのように解釈し、扱うかも重要な要素です。テクノロジが進歩しても、人間の身体、器官はそうそう変わらないものです。そう考えると、より適切なインタラクションを追い求めることは、人間をよく知り、普遍的な事柄を見つけていくことなのかもしれません。SD

Career Quest
学生が仕事や会社のことを知るためのカードゲーム

Stop-Motion Studio
スマートフォンでストップモーションアニメーションが作れるツール

GADGET 1

Grip Hint

http://www.griphint.com/

ペンの持ち方を指示するデジタルペン

ペンの正しい持ち方をすると、ペンに内蔵されたLEDが光るデジタルペン。無理な力が入らない正しい持ち方を自然に学ぶことができます。学習障害や自閉症の子供達に対して、文字や絵を書く手助けができるそうです。残念ならがまだコンセプトの段階で実際の製品にはなっていません。ペンの形を動物やぬいぐるみの人形に見立てたような形も考えているようです。何かの持ち方や使い方が正しいことが客観的にわかる技術は、さまざまな道具やスポーツ用品においても役立ちそうです。

GADGET 2

LEGO Fusion

http://www.lego.com/ja-jp/fusion/apps

ブロックとAR（拡張現実）の共演

LEGO Fusionは、LEGOブロックで組み立てた形状をiPhone/Androidスマートフォンや、iPad/Androidタブレットで認識し、ブロックで作られた街を歩き回ったり、ブロックの塔を守るために戦ったり、自分だけのリゾートを設計したりすることができるしくみです。LEGOブロックで組み立てたのは一部の建物だけでも、デジタルデバイスの中では無限の世界と、そこに歩き回るLEGOの人々を見て想像を膨らませることができます。組み合わせで補い合うオモチャです。

GADGET 3

Signet

http://www.ziba.com/work/signet

運送スタンプ用、未来のデジタル印鑑

米国を中心とする著名なデザインコンサルタント「ZIBA」が提案するのは、物の運送の宛名を独自のIDで刻印する、デジタルスタンプとの連携を進めた先進的な「印鑑（スタンプ）」のしくみです。一見しただけでは、誰から誰宛に送られたものなのかわからないようになっています。未来の個人的な手紙や配送業のことを考えたしくみです。配送料を分割したり、受け取りたい日付で配送してもらえるよう調整したり、柔軟性のあるしくみが考えられています。

GADGET 4

TetraBIN

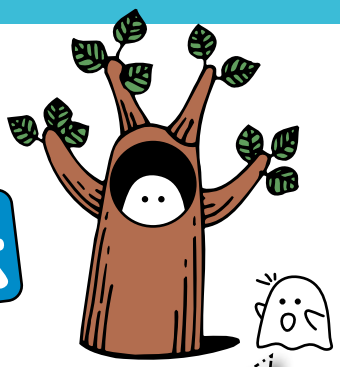
http://www.vividsydney.com/events/tetrabin

デジタルゴミ箱

TetraBINはシドニーの街中に設置された、ゲーム付きのデジタルゴミ箱です。ゴミを入れるとその瞬間電飾が輝き始めます。シドニー大学のプロジェクトで、街の生活に介入することと、持続可能な生活を導き出すために試験的に作られたものです。単に「道にゴミを捨てるな！」ではなく、楽しさとやりがいを提示することでポイ捨て問題をなくし、楽しんで街を綺麗にしてもらうアプローチです。このゴミ箱で楽しんだ人は、ほかの場所でもちゃんとゴミ箱を使うようになることでしょう。

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

結城浩の再発見の発想法 22



Modal Dialog

Modal Dialog——モーダルダイアログ

モーダルダイアログとは

コンピュータを使っていると**モーダルダイアログ**(Modal Dialog)がときどき表示されます。典型的なモーダルダイアログは、「保存しますか？」というものでしょう(**図1**)。ユーザが保存せずにアプリケーション(以下、アプリ)を終了しようとしたときに表示されるダイアログですね。「保存しますか？」のモーダルダイアログが表示されている間、ユーザは非常に限定された操作しか行えません。たとえば「変更を元に戻す」「キャンセル」「保存」という三者択一の選択をするまでは、ほかの操作は何もできません。

モーダルダイアログの**モーダル**(modal)という単語は、モード(mode)の形容詞形です。モード(mode)はムード(mood)に関連した単語で、人間が特定の気分やムードにひたるのと同じように、プログラムが現在入り込んでいる特定の状態のことを意味します。

先ほどの「保存しますか？」の例で言えば、このモーダルダイアログが表示されているとき、アプリは「保存するかどうかをユーザに確認する」という特定のモード(状態)に入り込んでいることになります。アプリが特定のモードにあるとはそういう意味です。

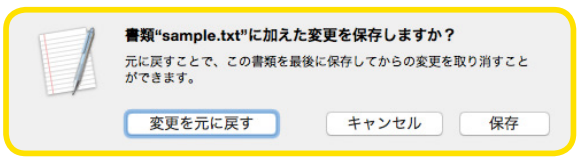
スマートフォンでもモーダルダイアログが表示されます。たとえばアプリをインストールするとき、パスワード入力画面になりますが、あれはモーダルダイアログです(**図2**)。また、アプリを使っていると表示される「このアプリを評価してください」もモーダルダイアログです。アプリの通常の操作は一時的に中断され、「いま評価する」「あとで評価する」「もう評価しない」という判断をユーザに行わせるモードに入っていますね。

モーダルダイアログの特徴

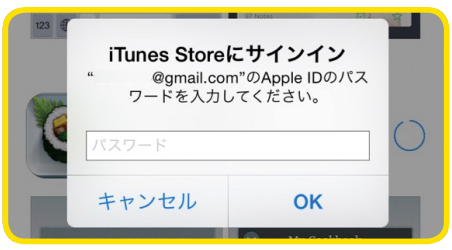
モーダルダイアログの特徴は、

- 普段の操作が中断される
- そのモードでできる操作は非常に限られている

▼図1 「保存しますか?」のモーダルダイアログ



▼図2 「パスワード入力」のモーダルダイアログ



- 普段の操作をしたかったら、進むかキャンセルかを行って、そのモードから抜ける必要がある

というものです。

普段の操作が中断され、できる操作が限られるというのですから、ひとことで言えばモーダルダイアログは「ユーザの自由を奪い、判断を強制する」ことになります。しかし、保存するかどうかを尋ねるモーダルダイアログのように、ユーザにしっかり判断してもらわないと困る状況では重要な役割を果たします。つまりそこでは「ユーザの自由を奪い、判断を強制する」ことで「重要な判断に集中させる」効果が期待できるのです。

モーダルダイアログの使い過ぎ

当然のことながら、モーダルダイアログを使い過ぎてはいけません。ほんとうに必要な場面でのみ使うべきです。

一連の情報をユーザに入力させるとき、開発者はモーダルダイアログを次々に表示させて入力させたくなることがあります。氏名を入力するダイアログを最初に表示して、メールアドレスを入力するダイアログを次に表示して、マシンのシリアル番号を入力するダイアログを次に表示して……のようにするということです。このようなアプリを作るのは簡単ですが、ユーザは不便を感じるでしょう。とくに「アプリのほかのページを開いて調べないと情報を入力できない」という状況のときには、モーダルダイアログの連鎖はいらいらするものとなります。

アプリのどこでモーダルダイアログを使うかは、設計者がよく考えなければなりません。

ときどき「いらいらするモーダルダイアログ」のジョークを目にします。たとえば、「ディスクの初期化」のような致命的なことを始めるのにOKボタンしかなく、キャンセルできないというもの。あるいはまた、1つのダイアログに選択肢となるボタンがずらりと10個も表示されて、どれを押したらいいかわからないというもの。これらのジョークは、自由を奪われていらいらするユーザの気持ちを代弁しているのかもしれません。

日常生活とモーダルダイアログ

モーダルダイアログはコンピュータの中での話ですから、日常生活でモーダルダイアログがポンと表示されることはありません。しかし、モーダルダイアログと同じ不便さを感じることはよくあるでしょう。

たとえば、会社で仕事をしていて、現在の作業を急に中断させられ、「こちらの作業Aをしてください」と依頼されたとします。わけもわからず作業Aを終えると「それでは次に作業Bをしてください」という依頼が来る。このように、「次の一手」しか示されない状態で作業を続けるのは、ストレスも掛かりますし、効率も悪くなるでしょう。

かといって、作業者の裁量にすべてを任せて自由に進めた場合、作業の遅れに気づくのが遅れる可能性もあります。普段は作業者の裁量に任せて作業を進めていいけれど、重要なチェックポイントではしっかりと確認するといった段取りが大事になるでしょう。

それはちょうど「保存しますか？」のモーダルダイアログに似ています。普段はアプリを自由に操作できるけれど、重要なチェックポイントではユーザの行動を制限して判断を強制する。それはつまり、モードにしっかりとメリハリを付けるということなのかもしれません。

設計者がアプリのユーザインターフェースを設計するように、作業の段取りにも設計者が必要なのでしょうね。

◆ ◆ ◆

あなたの周りを見回して、行動を必要以上に制限してモーダルダイアログのような不便さが隠れているものがないかを探してみましょう。

あるいは逆に、集中してことに当たらねばならない重要ポイントで、自由度を高くし過ぎていることがないか、考えてみてください。SD

耽溺せよ電子工作

おとなラズパイリレー

第5回

「Raspberry Piをメディアサーバ／プレーヤにしよう（前編）」

増井 俊之

おとなラズパイリレーは、Raspberry Piを文字どおり「リレー」し、好奇心旺盛なITエンジニアが電子工作をするという企画です。前編で構想を練り、後編で実装します。1年を通してどんなデバイスができあがるのか？……今回は、ユーザインターフェース研究の第一人者である増井先生によるRaspberry Pi B+でGear実装の構想編です。

Writer 増井 俊之（ますい としゆき） 慶應義塾大学 環境情報学部教授

ハルロックに負けられない!

Arduinoや Raspberry Piのようなワンボードコンピュータが流行しています。工作が大好きな女子大生がこういうガジェットを使って変な工作をする『ハルロック』という漫画が週刊モーニング誌に連載される（**図1**）など、ついにワンボードコンピュータの時代がキタか！という気がします。

筆者が高校生だったころ（1970年代後半）は、Intelの8008のようなCPUを買ってコンピュータを自作していたものですが、こういう工作趣味の世界がふたたび流行してきたのはたいへん喜ばしいことです。筆者も『ハルロック』の主人公「はるちゃん」に負けてはいられない！——ということで工作机を復活させたりしているのですが、老眼気味で小さい部品のハンダづけなどに苦労しています。

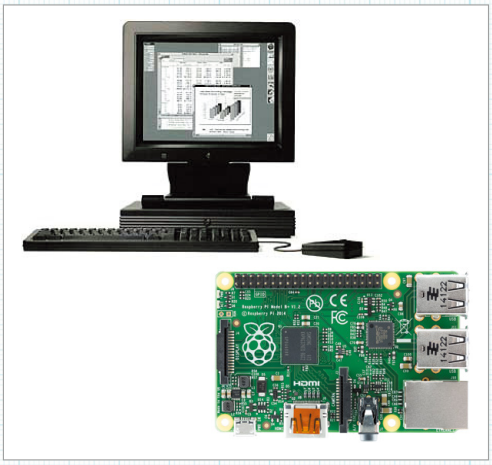
◀ **図1** 『ハルロック第3巻』西 餅（著）、2015年1月23日発売 講談社

NeXTStationとRaspberry Pi

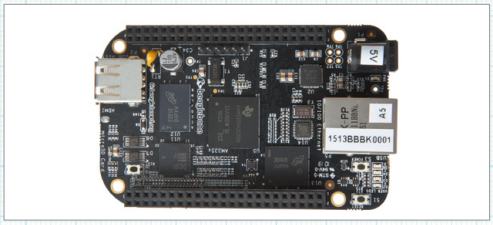
1990年代のはじめ、まだWebが存在しなかったころ、筆者は今のMac OSの前身であるNeXT Workstationの上で、今と同じようにEmacsでプログラムを書いたりTeXで論文を書いたりしていました。NeXTStationは15MIPS程度でしたがRaspberry Piは数百MIPSの性能を持っていますから、値段は1/100になったのに性能は100倍になったといえます（**写真1**）。

最近はIntel EdisonやBeagleBone（**写真2**）、Banana Pi（**写真3**）などさまざまな小型ワンボードコンピュータが利用可能になっており、この範疇（はんちゅう）の製品のなかでRaspberry Piが最も高性

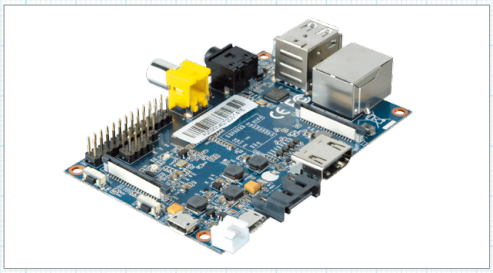
▼ **写真1** NeXTStationとRaspberry Pi



▼写真2　BeagleBoardのBeagleBone（http://beagleboard.org/bone）



▼写真3　Banana Pi（http://www.bananapi.org/）



能というわけではありませんが、現時点ではRaspberry Piは最もポピュラーですので安心して利用できると言えるでしょう。

Raspberry Piをメディアプレーヤとして使う

汎用コンピュータとして使えるRaspberry Piですが、普通のパソコンやタブレットなどに比べると圧倒的にパワー不足ですし、ブラウザが弱力だったりコンパイルに時間がかかったりするので、Web閲覧やプログラム開発といったパソコン的な用途にはあまりお勧めできません。

筆者の研究室では、ネット経由でドアを開閉したり、研究室内のセンサ情報を集めて外部からアクセス可能にしたり、さまざまな「実世界コンピューティング」のためにRaspberry Piを活用しています。ハード／ソフトウェアのトラブルがあったとしても、それも練習だと鷹揚（おうよう）に構えて、気軽に修理したり再インストールしたりしながら利用しています。

最近のテレビはたいていインターネット接続機能を持っていますし、テレビをメディアプレーヤとして使うためのさまざまな装置が販売されています[注1]。

これらの装置をテレビに挿しておけば、リモコンなどからコントロールすることによってネットやハードディスクのコンテンツを楽しむことができるので便利です。

普通のテレビでネット上の大量のコンテンツを楽しめるようになるのはたいへん便利なのですが、従来の家電のような「リモコンによる制御」感覚満載なのが嫌なところです。筆者はユーザインターフェースの研究者ですので、もっと自由に楽にコンテンツを選ぶ方法があるだろうにと思ってしまいます。前述のような専用機器の代わりにRaspberry Piを使えば、安価に動画などを再生できる装置となるうえに、動作を自由に制御できるというメリットがあります。

実はRaspberry Piは、CPU性能はBeagleBoneなどと比べて劣っているのですが、動画再生機能だけはほかのワンボードコンピュータより優れているので、Raspberry Piをメディアプレーヤとして利用することは理にかなっていると言えるでしょう。実際、Raspberry Piをメディアプレーヤとして使うための「RaspBMC[注2]」というAV再生専用のディストリビューションも公開されています。しかしRaspBMCはRaspbianのような標準指向のOSではありませんし、ブラウザを簡単に表示することもできないので自由度が十分とはいえません。Raspberry Pi用のOSとして最も一般的なRaspbianに対してさまざまなコントロールを行うことによって動画／音楽／写真／Webなどを自由に表示させることにすれば、かなり便利なシステムを作れそうです。

注1）AppleTV（https://www.apple.com/jp/appletv/）、Chromecast（www.google.com/chromecast）、Android TV（www.android.com/tv/）、dstick（www.docomo-dstick.jp/）、smart TV Stick（http://www.au.kddi.com/mobile/service/smart-tv-stick）、Fire TV Stick（http://www.amazon.com/dp/B00GDQ0RMG/）

注2）http://www.raspbmc.com/

おとなラズパイリレー

Gearのインタラクション

テレビやビデオのリモコンには何十個ものボタンがついているのが普通で、使いにくいものの代表のように言われていますが、AppleTVのような最近の機器では数個のボタンでさまざまな操作を行えるようになっています。iRiverのU10という音楽／動画プレーヤでは筐体の上下左右のエッジがボタンになっており、これらを押すことによってコンテンツを選択できるのが便利でした(**図2**)。

パソコンや携帯機器のキーやボタンを利用して階層構造データのナビゲーションを行う場合、階層を上下に移動したり項目のリスト内を移動したりすることによって目的の情報を捜すようになっているのが普通です。たとえば上下矢印キー(▼▲)を使ってファイルやフォルダを選択したり、左右矢印キー(◀▶)を使って階層を移動したりすることによって目的のファイルに到達できます。

キーの数を極限まで減らす!

しかしキーは本当に4個必要なのでしょうか？　2個のスイッチだけで階層情報のナビゲーションを実行できれば、より単純な装置を使って階層情報のナビゲーションが可能になり、いつでもどこでも誰でも簡単にデータを検索できます。筆者は、2個のキーだけを使って階層的なコンテンツをナビゲーションできるようにする「Gear」というインタラクション手法を提唱しています。

Gearでは、次の方法を用いることによって2個のキー(▼と▲)だけによる階層情報のナビゲーションを可能にしています。

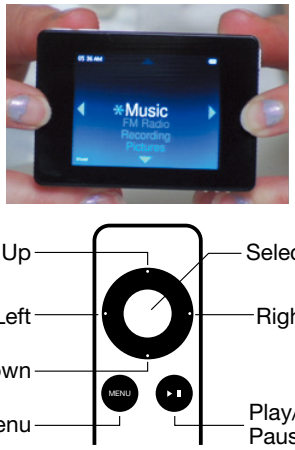
- 選択中の項目に下位層が存在するとき、キー入力を行わずに待つと下位層が自動的に展開され、下位層の最初の項目が選択される
- 項目リストの端を選択しているとき、さらにを押すと下位層は閉じられて1つ上の層の項目が選択される

図3のような構成のショッピングモールの店をGearでナビゲーションしてみましょう(**図4**〜**図9**)。

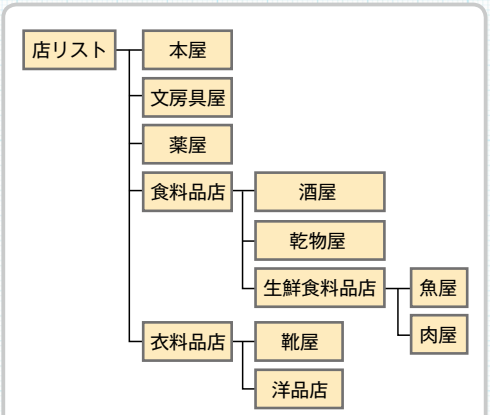
2個のスイッチだけを使うことには、次のような大きなメリットがあります。

- 普通のボタンやスイッチ以外の装置を利用できる
- 操作を迷うことがほとんどなくなる
- さまざまな行動を入力操作として利用できる

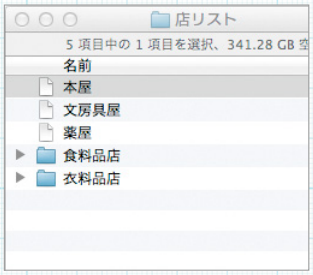
▼**図2**　携帯プレーヤU10(上)とAppleTVのAppleRemote(下)の4方向ボタン



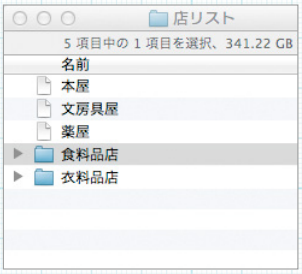
▼**図3**　ショッピングモールのツリー構造例



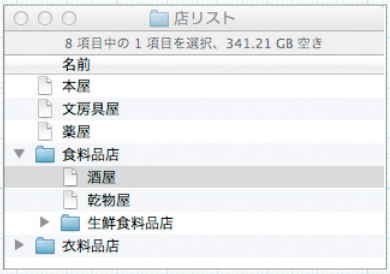
▼図4　①初期状態



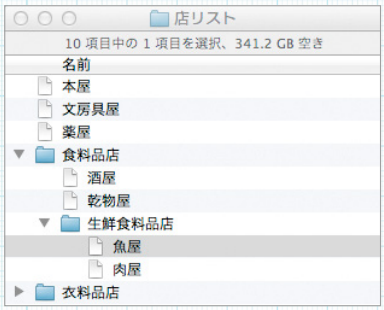
▼図5　②「食料品店」を選択



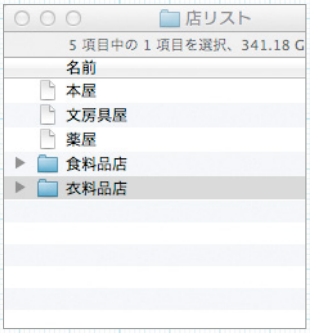
▼図6　③「食料品店」の下位階層を自動展開



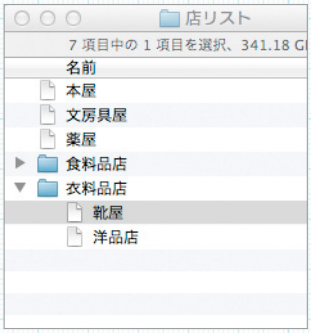
▼図7　④「生鮮食料品店」の下位階層を自動展開



▼図8　⑤「衣料品店」を選択



▼図9　⑥「靴屋」を選択



▼図10　Mac上でのGear実装

次号予告

Raspberry Piの場合、マウスやキーボードのような入力装置、ネット経由の通信、GPIO（General Purpose Input/Output：汎用入出力）端子などを利用すれば、さまざまな方法であらゆるコンテンツを簡単にナビゲーションするというのが可能になります（図10）。次回はいろいろな入力装置を使って実際にRaspberry Piをメディアプレーヤとして活用する方法を紹介したいと思います。SD

軽酔対談

かまぷの部屋

第8献 ゲスト：谷嶋 佑里恵さん

㈲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

谷嶋 佑里恵（やじま ゆりえ）さん

デジタルハリウッド大学卒。㈱サイバーエージェント勤務。アメーバ事業本部で、月間利用者数が約1,000万人を誇る女性向け掲示板サイト『GIRL'S TALK』サービス事業のフロントエンドデベロッパとして事業に携わる。入社2年目にして事業部MVP賞を受賞。サービスシステム移行プロジェクトを成功させた功績を持つ。
Twitter：@sery_dsm（ゆりえってぃ）

サイバーエージェント玄関にて。

USP研究所の鎌田広子（かまぷ）です。今回のゲストは㈱サイバーエージェント（以下CA）の期待の若手“ゆりえってぃ”さんこと谷嶋佑里恵さんです。

（鎌田）谷嶋さんはCAの『GIRL'S TALK』というサイトのエンジニアとのことですが、どういったことをされているのですか？

（谷嶋）『GIRL'S TALK』は、女性限定で完全匿名の掲示板システム（男性は閲覧のみ可）です。悩み相談や女性ならではの話で盛り上がっています。私はまだ入社2年目ですが、担当しているのはフロントエンド周りで、ユーザにWebサイトの利便性を提供するのが役割です。ブラウザとシステムの間を取り持つ部分なので、マークアップやプログラミング、デザインなど幅広い知識が必要です。会員数を伸ばすために、日々切磋琢磨しています。私自身もこのサイトをよく利用してるんですよ。

ご出身はどちらですか？　また、学生時代のエピソードがあれば教えてください。

納豆で有名な水戸出身です。自然が残っている環境で育ち、高校までは文系でした。子供のころから星を見るのが好きで、天体望遠鏡も持っています。小学生のころ、学校のコンピュータで「さめがめ」というゲームばかりしていました。中学生のころには、家でネットサーフィンをするようになり、個人サイトなどを見ているうちに、自分でもWebサイトを作りたいと思うようになりました。

中学生のころには、すでにパソコンを使いこなしていたのですか？

ネットサーフィンをしているうちに、自然に覚えてしまったようです。学校ではオタクと思われたくなくて、そういった話題はもちろん、キーボードも指一本で打っていました。

恥ずかしがり屋なんですね。子供のころからパソコンやWebに興味を持っていたのですね。Webサイトの作成は、誰かに教えてもらったのですか？

いえ、独学です。Webサイトを作ろうと思ったとき、まず書店に行き、ホームページビルダーの本を買ったのですが、役に立ちませんでした。ほかの本を買って勉強することも考えたのですが、学生の身には本が高価だったこともあり、ネットの情報でHTMLの書き方を勉強しました。そんなこともあって、大学進学のときに思い切って、当時秋葉原にあったデジタルハリウッド大学に進みました。水戸から秋葉原まで電車で片道2時間、1時間は睡眠で、起きたら読書や勉強をするといった感じで、移動時間を有効活用していました。

勉強家ですね。どんなWebサイトを作っていたのですか？

ちょうどそのころ、「同盟」というネット上のサークルのようなものが流行っていて、友人たちと同盟のサイトを作っていました。そのサイト上に掲示板を設置し、いろいろな人と交流しました。当時は友人が増えるのがとても楽しみでした。

大学では情報処理を勉強されたのですか？

大学ではおもにWebデザインを勉強しました。今の仕事の知識は入社してから身に付けたことが多いです。ですが、大学で学んだデザインの知識はチームとのコミュニケーションなどで役に立っています。いろいろなことを勉強しておくことは大切ですね。入社してすぐ、担当だったトレーナーが異動してしまい、




一人で猛勉強せざるを得ませんでした。その甲斐もあって、システムの移行の成功が認められて、事業部のMVPをもらうことができました。これはずいぶんと励みになっています。

入社して大活躍されているようですが、どのような苦労があったのか教えてください。

所属するチームは少数精鋭であるうえ、私は立場的にリードしなくてはならない状況でした。ちょっとしたフロントエンドの変更でJavaScriptを書き換えなければならない場合でも、ソースコードをすべて読んで把握しないといけないのでたいへんです。当時は新人だったので、ビジネスでの経験はないですし、どこに手をつけたらいいのか、どう書き換えたらいいのかで悪戦苦闘しました。

そのとき、どうやって乗り越えたのですか？

技術雑誌や書籍、ネットの情報を読み漁り、また社内で公開されているソースコードをお手本に、Backbone.jsの勉強をしました。技術的に理解するまでがたいへんでした。ある日、「ストン」とコツがわかるようになりました。フレームワークを導入し、構造をMVCモデルに書き換えました。すでにリリースまでのスケジュールが押していたので、自分の開発スピードを上げることに注力しました。間に合ったのはチームの皆の協力があってこそです。

やり遂げること自体もそうですが、若手の女性社員にそこまで任せる社風もすごいと思います。上司などには女性も多いのですか？

CAでは8名の取締役が2年ごとに原則2名入れ替わる「CA8」という独自の取締役交代制度があります。そして、8人の取締役に加えて、次世代経営者育成の観点で10名を選抜する「CA18」があり、昨年10月には、初めての女性の執行役員が誕生しました。私のチームも女性が多いです。仕事を頑張れる人、体力のある人が多いですね。でも渋谷にオフィスがあるからか、みんなとてもお洒落ですよ。

役員が定期的に入れ替わるのっておもしろいですね。テレビで御社を見かけますが、明るい雰囲気で、社員同士も仲良さそうですね。社内で飲みに行く機会も多いですか？

自由な社風で、だからこそ勉強も積極的になれます。私は残業としてではなく夜、オフィスで勉強することも多いです。社員同士の仲もいいですよ。ランチタイムはコミュニケーションするためのよい機会です。チーム内外問わず積極的に誘っています。みなさん嫌な顔1つせず、付き合ってくれます。また月に一度、チームで飲む機会があります。お酒は大好きなので、毎月楽しみです。

この取材を受けてくれた時点でお酒はお好きだと思っていました（笑）。

私は好きなのですが、両親がほとんど飲めなくて、お正月に実家用に日本酒を買って帰ったのですが、「あなたそんなに飲むの？」と驚かれました。しかたがないので1人で寂しく日本酒を嗜みました。

お酒を呑めると人生楽しいですよね！

本当にそう、ゼッタイそうですよ（笑）。お酒飲んで損したことないですよ。

同感です！　ところで休日はどんなことをして過ごしているんですか？

Twitterのプロフィールにある『宇宙推し』って何でしょうか？

先ほど、子供のころに天体好きという話をしましたが、それが高じて宇宙萌えなんです。キャッチコピーは『宇宙推しわくわくりえいたーゆりえってぃだよー(ﾉ◕ヮ◕)ﾉ』です。最近は、星空がきれいに見える場所で有名な長野の野辺山に、宇宙好きな仲間としょっちゅう行ってます。

星空好き、行動派のロマンチストなんですね。今日は楽しいお話、どうもありがとうございました。 SD




はんだづけカフェなう

6LoWPANしてみよう（前編）

text：坪井 義浩　TSUBOI Yoshihiro　ytsuboi@gmail.com　@ytsuboi
協力：㈱スイッチサイエンス　http://www.switch-science.com/

6LoWPANとは

IPv6 over Low power Wireless Personal Area Networksの略語です。6LoWPANは、IP（インターネットプロトコル）をセンサネットワークのノード（端末）といった小型のデバイスにもしゃべらせるなど、省電力や近距離の無線ネットワークでIPv6を使うための技術です（図1）。

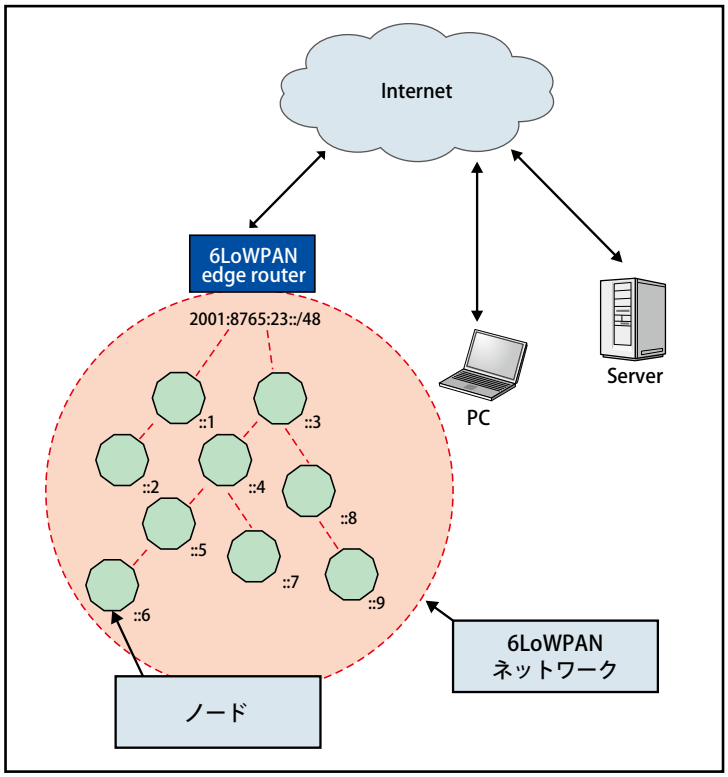
近距離無線プロトコルといえば、ZigBee（ジグビー）[注1]が有名です。みなさんも、この名前を一度は耳にしたことがあるのではないでしょうか。このZigBeeは、IEEE 802.15.4という標準規格上で動作するプロトコルです。IEEE 802.15.4は、物理層やMAC（Media Access Control）層などを規定しています。

一方でZigBeeは、先述のとおり、IEEE 802.15.4の上のレイヤであるネットワーク層のプロトコルです。IEEE 802.15.4は、スマートメーター[注2]で採用されるなど、広く使われている技術です。日本でも、IEEE 802.15.4gをベースにしたWi-SUNという規格がスマートメーターの標準規格として採用されています。

IEEE 802.15.4は低レイヤのプロトコルですので、上位レイヤにはZigBee以外にも、MiWi、Wireless HARTといった具合にさまざまなプロトコルが定義されています。しかし、たとえば、ZigBee技術を使用した製品を開発、販売する場合には、このZigBee Allianceへの入会が必要です。こういったプロプライエタリなプロトコルを使う

注1）　http://zigbee.org

▼図1　6LoWPANの概念図



注2）　次世代電力量計。通信機能を備え、電力消費量を測定する電力メーター。

のではなく、すでに広まっているIPをIEEE 802.15.4の上で使おうという技術が6LoWPANです（図2）。

MTU問題

話は変わりますが、MTU（Maximum Transmission Unit）を本誌の読者の方ならご存じだと思います。MTUは、パケット通信時に機器が送信することのできるフレーム（パケット）の最大サイズです。たとえばEthernetのMTUは1,518byteです。この、Ethernet MTUを拡張するとき、私たちはジャンボフレームを使って9KBといった具合に大きくし、ファイル転送の効率化を図ったりしています。

IEEE 802.15.4にもMTUがあります。IEEE 802.15.4のMTUは、127byteとEthernetと比べて1/10以下のサイズです。一方で、IPv6のヘッダは40byteあります。ここにMACの9byte、UDPの8byteを足すと57byteをヘッダだけで占めることになり、データ転送に使うことができるペイロードは70byte以下となってしまいます（図3）。

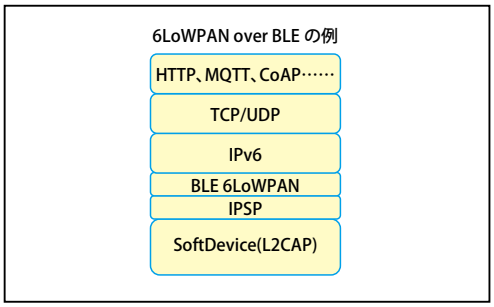
そこで、6LoWPANでは、HC1（RFC 4944）とIPHC（RFC 6282）を使ってIPヘッダの圧縮をします。また、IPv6にもMTUはあり、最小で1,280byteと定められています。6LoWPANでは、ネットワーク層よりも低レイヤでパケットのフラグメンテーションを行います（図4）。

6LoWPAN over BLE

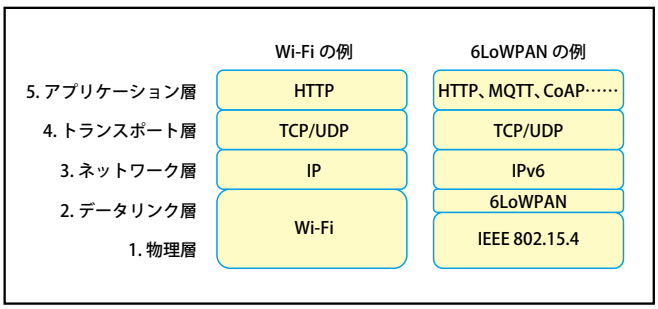
ここまではIEEE 802.15.4という無線プロトコルを中心に記してきましたが、Bluetooth Smart（BLEのブランド名）にもIPSP（Internet Protocol Support Profile）という6LoWPANをサポートするプロファイルが登場しました（図5）。

日本の工事設計認証を得たもので、先述のIEEE 802.15.4の無線を積んだ入手性のよいマイコン内蔵モジュールはあまり見かけません。しかし、BLEのモジュールならありふれています。また、今どきのスマートフォンにはBluetoothが標準で搭載されています。6LoWPAN over BLEという技術が普及していけば、スマートフォンなどの機器がルータの役割を果たし、アプリケーションレベルでの変換を要さず

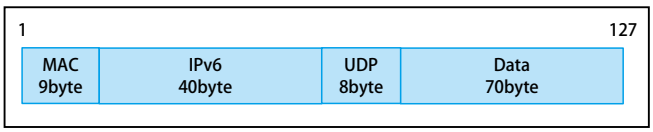
▼図5　IPSPのレイヤ



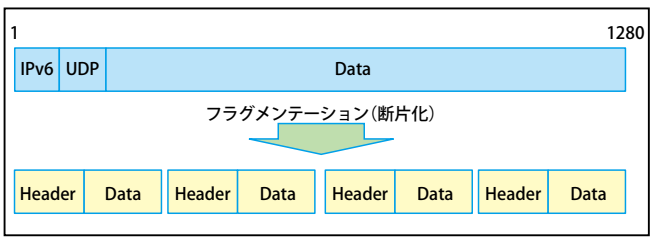
▼図2　6LoWPANの概念



▼図3　データ構造の例



▼図4　パケットの断片化



にインターネットにつながるかもしれません。そういった点で、6LoWPAN over BLEへの筆者の期待感は大きいです。

昨年末、Nordic Semiconductorが、nRF51シリーズSoC向けにIPv6 over Bluetooth Smartプロトコルスタックを「nRF51 IoTソフトウェア開発キット[注3]」としてリリースしています。nRF51シリーズのマイコン用のサンプルコードが入っていますので、ビルドするだけで動かしてみることができそうです。nRF51シリーズは、本連載2014年8月号の第46回でも紹介したHRM1017に搭載されているSoCです。筆者の手元には評価ボードがいくつかありますので、さっそく動かしてみました。

nRF51 IoT SDKの開発環境

nRF51 IoT SDKを動かすために筆者が用意したものは次のとおりです。

- Raspberry Pi Type B+[注4]
- micro SDカード(16GB)
- Planex BT-Micro4
- HRM1017評価キット[注5]

nRF51 IoT SDK

筆者はコンパイラにMDK-ARMという有償の製品を使いましたが、gcc-arm-embedded 4.7 2013q1[注6]もサポートされているコンパイラとしてリリースノートに記されています。実はnRF51822には、メモリが16KBのチップと、32KBのチップの2種類があります。32KBのチップは比較的最近リリースされたもので、筆者の手元にはありません。このため、Nordicが開発した16KBのメモリでも動作するIPスタックを使ったサンプルを使いました。このスタックはどうやらUDPのみをサポートしており、TCPの実装がないようです。そこで、今回はexamplesの中にあるnrf_udp_serverをビルドしてみました。ボードは、HRM1017と同じマイコンが搭載されているPCA10001を選択します。

ビルドをするためには、NordicのnRF51 SDKが必要です。HRM1017評価キットを購入するとプロダクトキーが記されていますので、このキーをNordicのWebサイトで登録することで入手できます。また、ビルドしてできたHEXファイルをHRM1017に書き込むために、nRFgo Studio[注7]もダウンロードしておきましょう。

ビルドが完了するとHEXファイルができあがります。この6LoWPANやIPv6のスタックを含んだマイコンのアプリケーションと、IoT SDKに入っているSoftDeviceのHEXファイルを、それぞれnRFgo StudioでHRM1017に書き込みます。nRFgo Studioは新しいバージョンでなければIoT SDKに同梱されているプロトタイプのSoftDeviceを書き込むことができませんでした。

Raspberry Piのセットアップ

NordicのIoT SDKには、BLEとEthernetの間のルータとして、Raspberry PiにBLEアダプタを取り付けたものを使う例が掲載されています。どうやら、Linux kernel 3.17以降であれば6LoWPANをサポートしているようですので、適当なLinux boxをルータにしてもよいです。ビルド済みkernelをNordicが配布していますので、今回はそれを利用することにします。

このビルド済みkernelも、IoT SDKと同じページで配布されていました。例ではRaspberry PiでRaspbianというディストリビューションを使っていました。これも、“December

注3) http://www.nordicsemi.com/eng/Products/Bluetooth-Smart-Bluetooth-low-energy/nRF51-IoT-SDK
注4) http://ssci.to/1826/
注5) http://ssci.to/1808/
注6) https://launchpad.net/gcc-arm-embedded/4.7/4.7-2013-q1-update
注7) https://www.nordicsemi.com/eng/Products/2.4GHz-RF/nRFgo-Studio

2014”という記事執筆時に最新だったものを使いました。PCで、ダウンロードしたOSのディスクイメージをddを使ってmicro SDに書き込みます。あとは、nRF51 IoT SDKのZIPアーカイブの中に入っている、Documen tation/index.htmlを見て、User Guides→ Setting up Raspberry Pi as routerの項にあるとおりにセットアップします。ここで行うのは、6LoWPANのkernel moduleのあるkernel 3.17以降、BluetoothプロトコルスタックのBlueZ、libpcap、ルータアドバタイズデーモンであるradvdのインストールでした。

つないでみる

セットアップを終えたら、debugfsを有効にしたうえで、6LoWPANのカーネルモジュールを読み込みます。また、PSM（Protocol Service Multiplexer）ナンバーをISPS（Internet Protocol Support Profile）を示す0x23、つまり35にセットします。

```
# mount -t debugfs none /sys/kernel/debug
# modprobe bluetooth_6lowpan
# echo 35 > /sys/kernel/debug/bluetooth/6lowpan_psm
```

Bluetoothアダプタのデバイス名を確認し、インターフェースをリセットします。

```
# hciconfig
# hciconfig hci0 reset
```

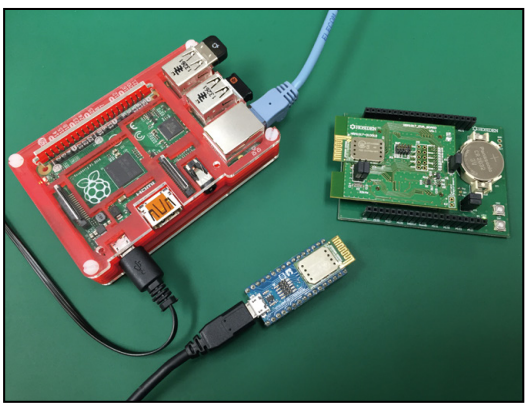
次に、ノードのBDA（Bluetooth Device Address）をスキャンして調べます。ここで調べたBDAを使って、6LoWPAN接続を確立します。

```
# hcitool lescan
# echo "connect 00:27:80:9B:6E:18 1" > /sys/kernel/debug/bluetooth/6lowpan_control
```

ifconfigすると、bt0というインターフェースが追加されていることが確認できるはずです。nRF51822には、BDAに応じてfe80::から始まるリンクローカルIPv6アドレスが割り当てられています。ドキュメントに読み替えの方法が記されていますが、ここではIPv6の全ノードアドレスにpingを打ってノードを見つけてみました（図6）。

1ms以下で応答が返ってきているものがbt0で、他の2つがnRF51822です。マイコンのプログラムのビルド環境づくりは大変ですが、このように、とても手軽に手元で6LoWPAN over BLEを試してみることができました。今回はICMPしか試していませんが、とりあえずはBLEを使ってIPv6で通信できることが確認できました（写真1）。今後は、実装されているUDPの実験、そして最終的にはIPv6のend-to-endでマイコンにつないだセンサの値をサーバに送信するところまで、進めていきたいと思います。SD

▼写真1　実験してみているところ



▼図6　全ノードアドレスにpingを打ってみる

```
# ping6 -I bt0 ff02::1
PING ff02::1(ff02::1) from fe80::21b:dcff:fe06:188a bt0: 56 data bytes
64 bytes from fe80::21b:dcff:fe06:188a: icmp_seq=1 ttl=64 time=0.259 ms
64 bytes from fe80::2b6:83ff:fe62:49bd: icmp_seq=1 ttl=64 time=98.0 ms (DUP!)
64 bytes from fe80::227:80ff:fe9b:6e18: icmp_seq=1 ttl=64 time=374 ms (DUP!)
```

PRESENT 読者プレゼントのお知らせ

『Software Design』をご愛読いただきありがとうございます。本誌サイト http://sd.gihyo.jp/ の「読者アンケートと資料請求」からアクセスし、アンケートにご協力ください。ご希望のプレゼント番号をご入力いただいた方には抽選でプレゼントを差し上げます。締め切りは2015年3月17日です。プレゼントの発送まで日数がかかる場合がありますが、ご了承ください。

ご記入いただいた個人情報は、プレゼントの抽選および発送以外の目的で使用することはありません。アンケートのご回答については誌面作りのために集計いたしますが、それ以外の目的ではいっさい使用いたしません。ご入力いただいた個人情報は、作業終了後に当社が責任を持って破棄いたします。なお掲載したプレゼントはモニター製品として提供になる場合があり、当選者には簡単なレポートをお願いしております（詳細は当選者に直接ご連絡いたします）。

01 テレビ用高音質スピーカー「Olasonic TW-D6TV」

1名

東和電子のオーディオブランド「Olasonic」の新製品。テレビ音声を、全帯域に渡り忠実に再現できる卵型の高音質スピーカーです。ユーザの意見を取り入れ、前機種から、低域の迫力や広がり感のある音の再現などを中心に強化されています。テレビ以外にもイヤホンジャック搭載の製品であれば接続して使うことができます。周波数帯域は80Hz～20kHzです。

提供元 Olasonic／東和電子 **URL** http://www.olasonic.jp

02 モバイルバッテリーチャージャ「ZM-MB350-W」

1名

JVCケンウッド開発のスマートフォン用、モバイルバッテリーチャージャです。付属のUSBケーブルでスマートフォンと接続し充電できます。本体への給電もパソコンなどからUSB経由で行います。容量は3,500mAhで、電池の残量が光ってわかる表示機能付きです。

提供元 シュナイダーエレクトリック **URL** http://www.schneider-electric.com

03 SoftLayer ノベルティ手帳

3名

本誌2014年9～12月号でも記事として取り上げた、IBMが提供するクラウドサービス「SoftLayer」。そのロゴが入った、バンドつきノベルティ手帳です。中身は5mmの方眼紙となっており、さまざまな用途に使えます。

提供元 日本アイ・ビー・エム **URL** http://www.ibm.com/jp/ja

04 詳解 Swift

2名

荻原 剛志 著／
B5変形判、408ページ／
ISBN = 978-4-7973-8049-1

Objective-Cの解説書で知られる萩原氏によるSwift本。言語の基本からObjective-Cとの連携まで、詳細に解説しています。本書はSwift1.1、Xcode6.1以降に対応しています。

提供元 SBクリエイティブ **URL** http://www.sbcr.jp

05 Webエンジニアが知っておきたいインフラの基本

2名

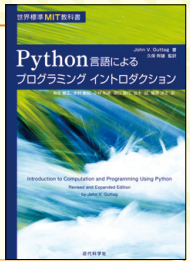
馬場 俊彰 著／
A5判、312ページ／
ISBN = 978-4-8399-5355-3

Webアプリ・フロントエンドエンジニア向けのインフラ解説書です。インフラエンジニアとの連携をスムーズにしたい、インフラへの負担を配慮したアプリを作りたいといった人にお勧めです。

提供元 マイナビ **URL** http://www.mynavi.jp

06 Python言語によるプログラミング イントロダクション

2名



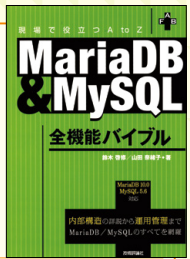
John V.Guttag 著／久保 幹雄 監訳／
麻生 敏正ほか 訳／
B5判、328ページ／
ISBN = 978-4-7649-0469-9

マサチューセッツ工科大学で常にトップクラスの人気を誇る計算科学の講義、その内容をまとめたテキストの翻訳本です。Pythonの言語仕様を追いながら、プログラミングの手法一般を学べます。

提供元 近代科学社 **URL** http://www.kindaikagaku.co.jp

07 MariaDB&MySQL全機能バイブル

2名



鈴木 啓修、山田 奈緒子 著／
A5判、576ページ／
ISBN = 978-4-7741-7020-6

MariaDB 10.0、5.5／MySQL 5.6、5.5の基本的な機能を網羅したリファレンス形式の本。インストール方法から、SQLの文法、運用手法、データベース内部構造の解説まで扱っています。

提供元 技術評論社 **URL** http://gihyo.jp

第1特集

CONBUの無線LAN構築術

カンファレンスネットワークの作り方

IT系カンファレンス（勉強会や交流会など各種イベントを含む）では、無線LANによるインターネット接続環境の提供が普通になってきました。皆さんよくご存じのLL DiverやYAPCでこのネットワークを設営していた技術者集団がCONBU（Conference Network Builders）です。

職場の異なるネットワーク技術者の有志が集まって知恵を出し合い、「いかに無線LANの電波状況を良くするか」、「いかに短時間で構築／撤去できるようにするか」、「いかに運用時のリスクを下げられるか」などを考え、より良い環境を参加者に提供するために改善を繰り返しています。

カンファレンスを裏で支えるCONBUのテクニックには、無線／有線ネットワークを構築する人にとっても、使う人にとっても役立つヒントが満載です！

CONTENTS

イラスト●高野 涼香

第1章

会場でネットがつながりにくくなる理由

参加者自身の行動も原因の一端に!

Writer 田島 弘隆(たじま ひろたか) Genie Networks／CONBU Mail tajima@hirochan.org

IT系のカンファレンスに足を運んだことがある方であれば、その会場でインターネットへの接続をしたことがあるでしょう。もし会場で無線LAN接続が提供されていたら、ぜひそれを利用してください。自前のWi-Fiルータやスマホのテザリングで十分？いえいえ、それこそがネットにつながりにくくなる原因かもしれないのです。

カンファレンスネットワークとは

昨今はITに関連した勉強会がたくさん開催されるようになりました。勉強会の多くは参加者が数十～100名程度ですが、人気のあるものは大きなイベント会場を1日～数日間使用するITイベントとして開催されます。たとえばYAPCやLL Diverなどの言語系やWeb開発者が多数参加するカンファレンスの場合、その参加者は数百～1,000人以上になることがあります。カンファレンス参加者はセッションに参加しながら、手元のPCやスマートフォンなどからSNSや調べものをするためインターネットを使用します。しかしながら、このような大規模カンファレンスの場合、イベント会場にはインターネットアクセスができる無線LAN設備があるにもかかわらず、インターネットにつながりにくかったり、あるいはまったくつながらないケースがよくあります。

本特集では、このような大規模カンファレンスにおけるネットワークについての考え方やノウハウなどについて記述していきます。カンファレンスネットワークの解説を通して、オフィス環境などでのネットワーク作りの参考にしていただけたら幸いです(以降、ネットワークをNWと省略する場合があります)。

“つながらない”のはなぜ?

大規模カンファレンスが行われるような会場には備え付けの無線LANネットワーク設備があることが多いですが、カンファレンス会期中につながりくくなることはしばしばです。この原因の多くは、既設の設備で想定していた以上の無線LAN接続が発生してしまっていることにあります。

とくに昨今はスマートフォンやタブレット端末の普及により1人で複数の端末を使用するケースが多くなり、会場内には来場者数以上の無線LAN端末が存在することが多くなっています

▼写真1　参加者の多くはPCやモバイル端末を使用している

(**写真1**)。また、モバイルルータなどを持つ人も多く、これが無線環境を悪化させる要因の1つにもなっています。

たとえば、筆者は日常的にPC、スマートフォン、タブレット端末およびモバイルルータを持ち歩いています。これらがそのままイベント会場に持ち込まれることになるため、無線LAN端末は来場者数が100人なら端末数は200以上といったように倍以上になることが珍しくありません。

“つながらない”最大の原因は会場内に多数の無線端末が稼働していることです。多数の無線端末は次のような副次的な要因を引き起こします(**図1**)。

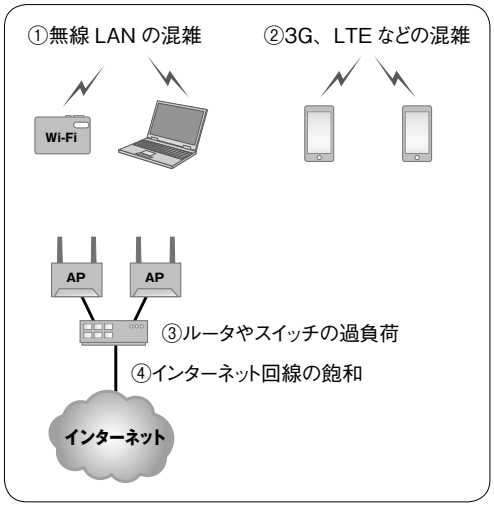
①無線LAN(Wi-Fi)の混雑(802.11b/gの2.4GHz帯、802.11aの5GHz帯)
②スマートフォンやモバイルルータが使用する3GやLTEなどの混雑
③インターネット接続に使用する通信機器(ルータなど)が通信量に耐え切れない
④インターネット回線の飽和

①と②は無線環境による要因、③と④は有線NWによる要因です。とくに悩ましいのが①で、カンファレンスが開催される多くの会場では、会場内や近隣にもともと無線LAN設備が多数設置されていて、電波環境が大変混雑していることがあります(これを“電波が汚い”と表現したりします)。その環境にさらにカンファレンス専用の無線を設置することで無線LANがさらに混雑することになります。③や④はルータを持ち込んだり新規回線を敷設することで対応できますが、会場によってはそのような手段を講じることができない場合もあります。さまざまな制約のなかで「使える」カンファレンスNWをいかに構築するかを本稿で触れていきます。

家庭の無線LANと何が違うのか

みなさんの家庭にも無線LAN AP(アクセスポイント)やブロードバンドルータが設置されていることと思います。無線AP機能とルータ機能が一体型になったブロードバンドルータが量販店では一般的ですし、通信販売でも気軽に購入できます。スマートフォンを契約するとモバイル事業者から無料で提供されることも多く、テレビを置くのと同じ感覚で設置することもあります。さらに、NINTENDO 3DSやPlayStation Portable/Vitaといった携帯型のゲーム機や、WiiやPlayStationなどの据え置き型ゲーム機、加えてテレビ、ビデオ、プリンタなども無線LANを使用してインターネットに接続しているでしょう。

▼図1　つながらない理由

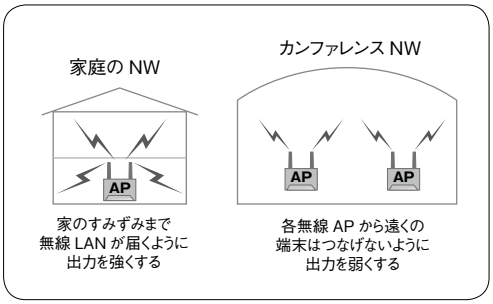


カンファレンスNWといえども、基本的な構成は家庭のNWと同じです。ただ、そのNWの設計思想は対極にあるともいえ、制限事項も多くあります。まとめると**表1**のようになります。カンファレンスNWにとくに特徴的なことは無線APの台数、無線出力強度、ルータでしょう。無線APの台数は家庭であれば(よほどの広い家でなければ)1、2個あれば十分でしょう。対してカンファレンスNWの場合、クライアント端末数が大変多いため、1台のAPにそれらをすべて収容することは不可能であり、多数のAPが必要になります。

▼表1　家庭のネットワークとカンファレンスネットワーク

	家庭	カンファレンスNW
クライアント数	数個～十数個程度	数百個～1,000個以上
無線AP台数	1～2台	10台以上
無線APの無線出力	家全体に届くように出力を強くする	狭い範囲に限定するため弱くする
無線APの通信速度	とくに制限はしない	低速(数Mbps)の通信は拒否する
無線の混雑状況	空いている～やや混雑	大変混雑している
電源コンセント個所	多くある	特定個所に制限される
LAN配線の距離	数m～数十m	数百m以上
インターネット回線	自由に選択できる	かなり制限される(例：既存回線のみ、新規でも特定方式のみ)
ルータ	いわゆるブロードバンドルータ	SOHO向けやソフトウェアルータ
設営に要する時間	あまり制限はない	数時間

▼図2　電波の強弱



無線APをたくさん設置すればいい?

ここで無線APをたくさん並べるだけでよいかというと、そう単純にはいきません。電波出力の強度調整が重要です。家庭向けの場合は家の中のどこからでも通信ができるように、できるだけ強い電波を出力することが多く、家電量販店では高出力を売りにした無線LAN一体型ブロードバンドルータも多数見かけます。ところが、カンファレンスNWでは逆にできるだけ出力を弱くします。これは狭い空間に多数の無線LANクライアントが密集するため、無線AP1台あたりの収容クライアントをAP周辺だけに限定することで高速な通信を確保するためです。遠くの無線APにつながると通信が低速になりがちなため、そのような無線APにはつながらないようにしているといえます(**図2**)。詳しくは第2章で触れます。

カンファレンスNWでも家庭NWと同じくNAPT(Network Address Port Translation)を利用しますので、ルータでNAPT処理を行います。ただ、カンファレンスNWではクライアント端末数が1,000台以上になることもあるため、それだけの端末数を処理するためのNAPTセッション数を十分確保するなどが重要です。また、ブロードバンドルータで行っているDHCP処理やDNS処理についても家庭用とは異なるケアが必要になります。このあたりは第3章で触れます。

ネットワークの稼働状況

ネットワークを流れている通信を可視化することは大変重要です。カンファレンスNWは構築にかけられる時間が短いため、構築後の確認作業に十分な時間はかけられません。会場には多数の無線APを設置しますが、クライアント端末が特定の無線APに偏りなく収容されているかを確認することが重要です。無線LANの電波は人間に吸収されてしまう特性があるため、会場に大勢の人が入ると無線の伝搬に偏りができ、事前の想定とは異なる無線APに多数の端末が収容されてしまうことがあります。言い換えると、実際に大勢の参加者が会場に入り、かつ通信が発生してはじめて確認できることになります。

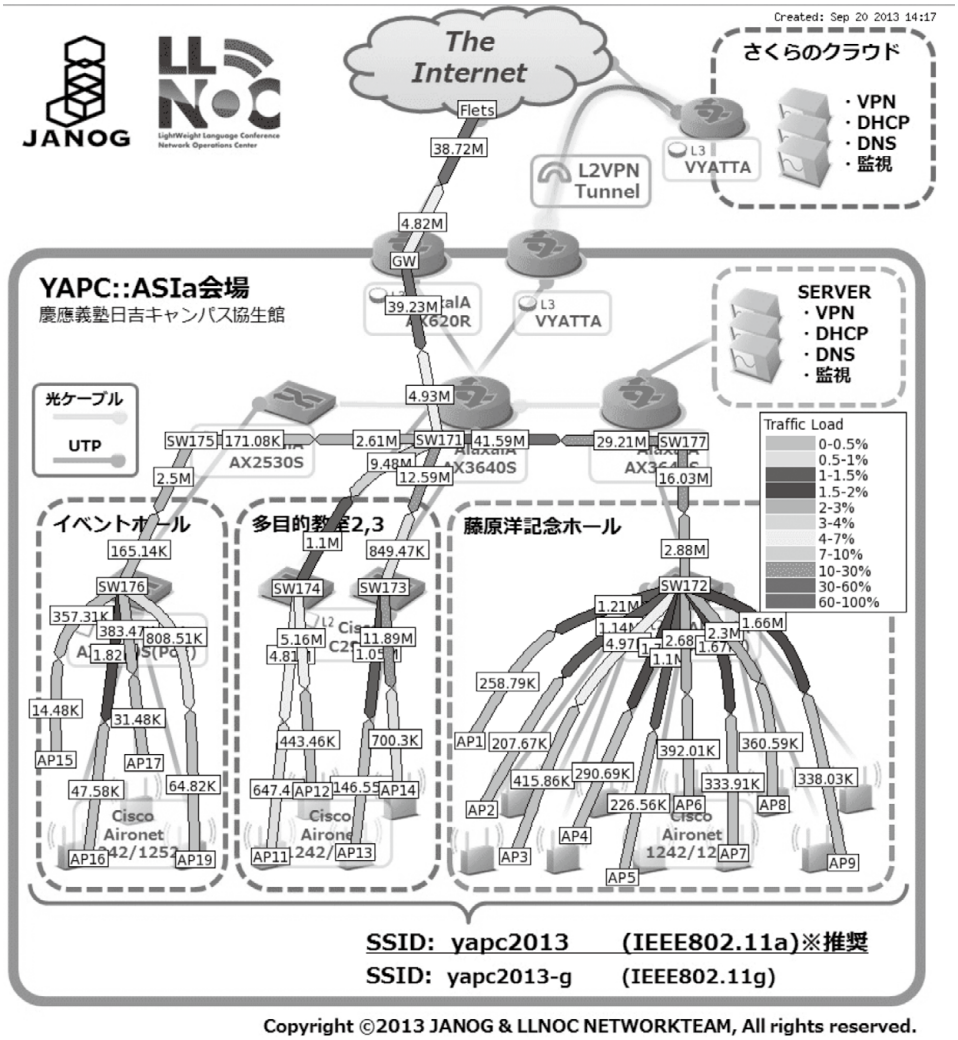
そこで、会場の物理的な位置関係と通信状況を対比しながら確認できると、もし通信状況に偏りが発生していても迅速な発見と修正が可能になります。**図3**は「Weathermap」と呼ばれるツールによりトラヒックを会場図に重ねて表示した模様です。Weathermapによるトラヒックマップの作り方はJANOG32のページ[注1]が便利です。

注1）http://www.janog.gr.jp/meeting/janog32/weathermap.html

トラブルは発生する前提で対策する

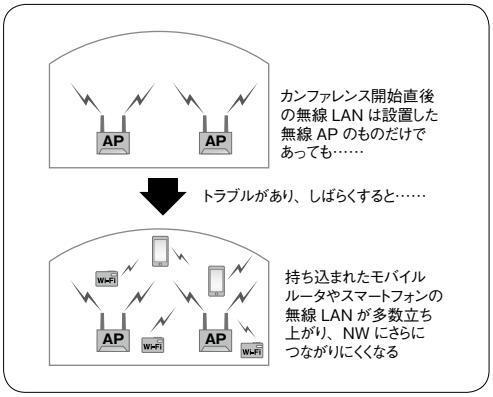
大規模なカンファレンスは大勢の参加者が来場するので、トラブルは必ず起こります。会場に多種多様なPCやスマートフォンが持ち込まれるため、事前の準備段階では洗い出しきることはできません。よくあるトラブル例として「VPNがつながらない」「DHCPアドレスが付与されない」「DNSの名前解決ができない」などがあります。

▼図3　YAPC::Asia Tokyo 2013でのWeathermap



これらのトラブルに早く気づくことが重要です。トラブルに遭遇した参加者がそれをNW担当者に伝えてくれることは期待できません。トラブルが少数であっても、その対象者が会場のNWを使わずに持ち込んだモバイルルータを立ち上げたり、スマートフォンでテザリングをはじめると、会場の無線環境が汚染されてほかの参加者にも影響が出はじめます。すると、ほかの参加者も会場NWにつながりにくくなり、その人たちもモバイルルータを立ち上げます。このようにしていつの間にか多数のモバイルルータやスマートフォンによる独自のWi-Fiが立つようになると、会場の無線LAN環境が極端に汚染されてしまい、ついにはNWが破綻します(**図4**)。

▼図4　モバイルルータにより会場無線が破綻する様子



このような状況になる前に、まずトラブルに気づくことが重要です。トラブルに気づかなければトラブル対応も始められません。ZabbixやNagiosなどのNW監視ツールによる監視は当然として、TwitterなどのSNSでの情報収集は非常に有効です。トラブルに遭遇した人がそれをNW担当者に申告してくれるとは限りませんが、それでも、何かがおかしいとSNSでコメントすることがあります。たとえばTwitterであれば、イベント名やイベントのハッシュタグで検索してトラブルの声を拾いあげましょう。

LAN配線は撤去を想定して敷設する

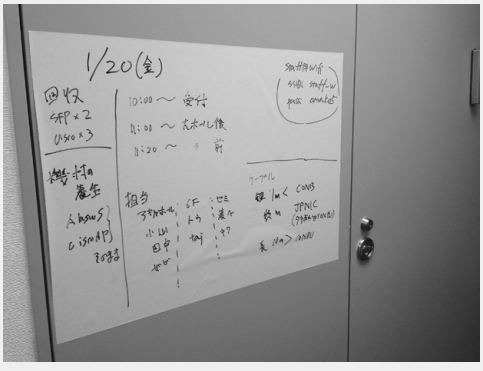
会場にNWを敷設する場合、イーサネットケーブルによるLAN配線は必ず発生します。会場が大きかったり部屋が多数あると配線の本数は100本近く、長さは数百mに及ぶこともあります。さらに、カンファレンス終了後にはこれらの敷設したケーブルは必ず撤収する必要があります。

ケーブルは人が通るところでは足に引っかからないように養生テープなどで養生しますが、養生をやり過ぎると撤収に時間がかかりすぎてしまいます。よって、人が通らないところはでき

Column チーム内の情報共有に有効なホワイトボード

トラブルに限らず、運用していると監視情報やtipsなど、チーム内に伝えておきたい事柄が多数発生します。チーム内での情報共有は非常に重要です。情報共有の手段は多数ありますが、短期決戦のカンファレンスNWの場合に最も有効なのがホワイトボードです。気づいたことはともかくなんでもホワイトボードに書き込み、メンバーは常にチェックする運用にしましょう。ホワイトボードがない場合はシート状のホワイトボードシート注2(**写真A**)を持ち込むと便利です。

▼写真A　情報共有用のホワイトボード。シート状になっていて壁や窓などに貼れて便利



注2) セーラーどこでもシート：http://sailorshop.jp/SHOP/31-3800-000.html?gclid=CNe914DzjsMCFYiVvQodyHIAeQ

るだけ簡易な養生にしたり、会場の既設配線を利用するなどそもそも配線をしない工夫が必要です。**写真2**は面ファスナーによるケーブル養生で、敷設や撤収作業が非常に早く行えて便利です。

カンファレンスネットワーク原則

カンファレンスNWを手がけるうえで、念頭においておくと良いと思う原則をまとめてみました。

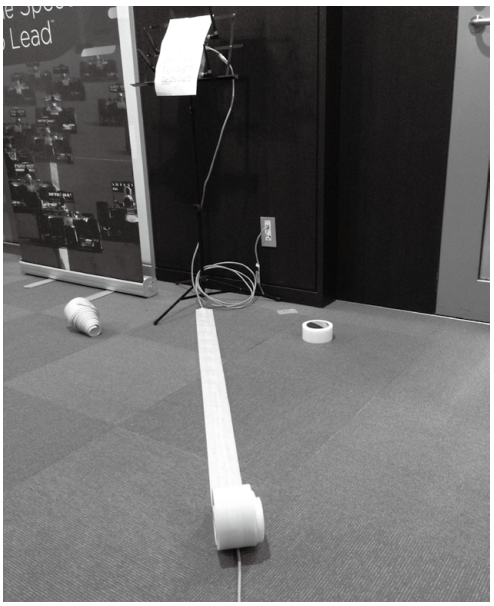
1. ネットワークとは人のネットワーク。機械をつなぐだけではない。人間関係を何よりも大切にしよう
2. 当日に「手を抜く」ための準備をしよう
3. 設営作業は撤収作業の第一歩。撤収を想定して設営しよう
4. 稼働状況は会場に人が入ってはじめてわかる。状況を拾うアンテナをたくさん持とう
5. 想定どおりにはまずいかない。第2、第3のプランを持っておこう

CONBUの紹介をすこしだけ

冒頭でも触れたように、最近のIT系カンファレンスでは会場NWが必須になってきています。カンファレンスも多数開催されるようになりましたが、会場NWの構築運営を行える人は十分とは言えません。とくにプログラム言語系やWebアプリ開発などのカンファレンスの場合、NWを作れる人はかなり限定されています。また、個別のカンファレンスのそれぞれがゼロからNWを作るのではたいへん非効率です。そこで、カンファレンスNW作りを支援していこうとNW技術者のコミュニティであるJANOGやLightweight Language ConferenceのNWチーム(LLNOC)の有志メンバーに呼びかけ、CONBU[注3](Conference Network Builders)が

注3) http://conbu.net/

▼写真2　面ファスナーのケーブルガード



結成されました。

CONBUのメンバーはおもにNW技術者です。CONBUの目的はIT系コミュニティのエンジニア同士がお互いに関心を持ち合うことで、コミュニティの垣根を越えて交流していくことです。インターネットではたくさんのプレイヤーが活躍していますが、Webアプリ開発などのコミュニティとネットワークを手がけるコミュニティの交流は十分とは言えないと思います。お互いに関心を持っていくことで、インターネットはより発展するのではと考えています[注4]。

結び

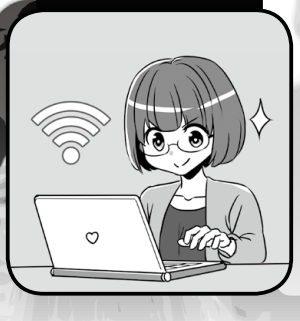
カンファレンスNWは短時間で構築して、多数の端末が期間限定で接続され、短時間で撤去するというものです。本特集の内容はイベント以外にも、引っ越しが多かったり短期間で移転することの多い企業や組織などのオフィスNWにも応用できるかもしれません。SD

注4) 基本的にはCONBUが協力する対象は非営利のIT系カンファレンスです。ビジネスカンファレンスなどは専門の事業者に依頼されることをお勧めします。

第2章

カンファレンス向け高密度無線LANの作り方

成功のカギは電波特性を活かすこと



Writer 熊谷 暁(くまがい あきら) CONBU Twitter @tinbotu

今や、日常の無線LAN利用においては、電波のことなど意識せずとも簡単に使える時代です。しかし、カンファレンスネットワークでは、その特性上、電波のことを理解していないと、快適な無線LAN環境は構築できません。本章では、電波について少しだけ知ることによって、より良い無線LAN環境を作れるようになることをめざします。

無線LANが遅くなるのはどうして?

無線LAN(IEEE802.11)は、快適なときは有線接続と比較しても遜色なく通信できますが、遅くなったり、通信が切断されたりすることもしばしばあります。有線は非常に安定しているのに、どうして無線LANは不安定になるのでしょうか。原因として次の2点が考えられます。

①電波の強度と通信速度
無線LANの親機であるアクセスポイント(AP)が遠くて、電波が弱い。ノイズが多い

②無線LANの混雑
多くの端末が使っていて、無線が混雑している

これらは、互いにあまり関係なく独立して起こる事象です。ほかにユーザはいないけれどAPが遠い。APは近くにあるけれど混雑している。APが遠くにあって、なおかつ混雑している。一般家庭で問題になるのは①のケースが多く、クライアントが高密度なカンファレンスネットワークでは②が問題になります。この2点についてそれぞれ考えてみましょう。

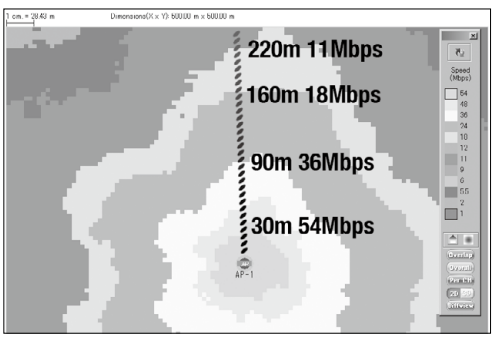
パソコンやスマートフォンなどの端末では、画面上のアイコンなどによってAPの電波強度を大まかに知ることができます。混雑しているかどうかを調べるにはアイコンだけではわからないため、若干難しくなります。

①電波の強度と通信速度

有線LANでは、ケーブルの長さが規格内であればリンク速度は一定で実用速度もほぼ一定です。これに対し、無線LANは電波強度(≒距離)やノイズによって端末ごとにデータレート(伝送速度)をリアルタイムに変更するしくみになっています。これは、データレートが遅ければ遅いほど通信状況の悪化に対して耐性があるためです。周囲の状況に応じて最適な速度に調整されます(**図1**)。

意外なことに、周囲の電波状況が理想的であることは、まれです。問題なく使えていても、多くの場合、何かしら悪い状況があるものです。

▼図1 理想的な空間での距離とデータレートのシミュレーション



無線LANは通信状況の悪化を前提とした規格であり、それをリカバリできるようなしくみを持っています。これは、伝送媒体が空間そのものであることに起因します。空間に何が存在するかわかりませんし、不要なものもどんどん入ってきてしまいます。ケーブルという伝送媒体の品質が一定水準以上であることを前提としているイーサネットとは、大きな違いがありますね。

電波強度が非常に強ければ、多少のノイズは無視できます。無線LANの干渉源として有名なものに電子レンジがありますが、APと端末をものすごく近くすることで、相対的に電子レンジをないものとして考えることができます(現実的な解決法ではありませんが)。また、干渉源を減らすことで、無線LANの電波強度を上げたのと同じ効果があります。

OS XではOptionキーを押しながらWi-Fiアイコンをクリックすると、現在のデータレートを確認できます(**図2**)。押すごとに目まぐるしく変化しているのが確認できる場合も、あるかもしれませんね。

▼図2　OS Xにおける Wi-Fi のデータレート確認画面

②無線LANの混雑

無線通信は、ある瞬間の空間を占有して行います。複数の端末が同時に通信しているように見えますが、実際には、さまざまな方法でリソースを分割して通信を実現しています。有線LANであるイーサネット(IEEE 802.3)も同様ですが、無線LANには無線ならではのリソース共有の難しさがあります(**表1**)。

イーサネットでは、たとえばアップリンク1Gbpsのスイッチに収容できる端末の合計スループットは、余程のことがない限り大まかに1Gbpsであろうと予測できます。しかし、無線LANの場合は電波状況が理想的ではないことが多く、性能が予測しづらくなります。これはおもに、空間の状況が多様であることと、半二重であることによります。

ここでは、非常に単純化したモデルとして、1つの無線LANチャネルを5台の端末で共有して利用することを考えます。時間で区切って、それぞれの端末が順にデータを転送しますが、ある一瞬に注目すれば特定の端末とAPが1対1でチャネルを占有しています(**図3**)。

実際には、占有時間は通信内容によってまちまちです(**図4**)。無線LAN独自の制御フレームのため、転送すべきデータがなくとも通信が発生することもあります。

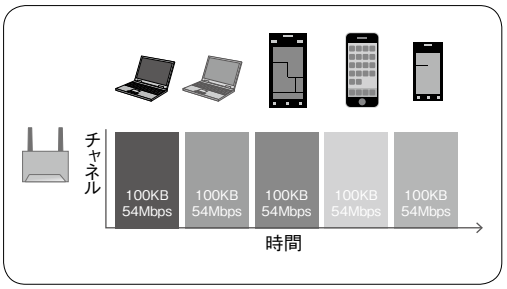
細かい手続きはいったん無視し、仮に**図3**の

▼表1　イーサネットと無線LANの違い

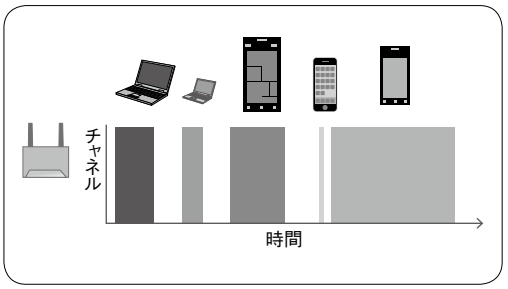
	イーサネット	無線LAN
全二重[注1]	できる	できない
スイッチング	できる	できない
衝突検知	できる	できない
リンク速度	ほとんど皆同じ	端末および状況によってまちまち
端末同士の存在検知	全端末同士が見える	電波強度によって互いに検知できない場合あり
チャネル	線を増やせばほぼ無限	法律で決められた数～十数チャネル

注1) 誰かが送信中でもほかの端末が送信可能であること。

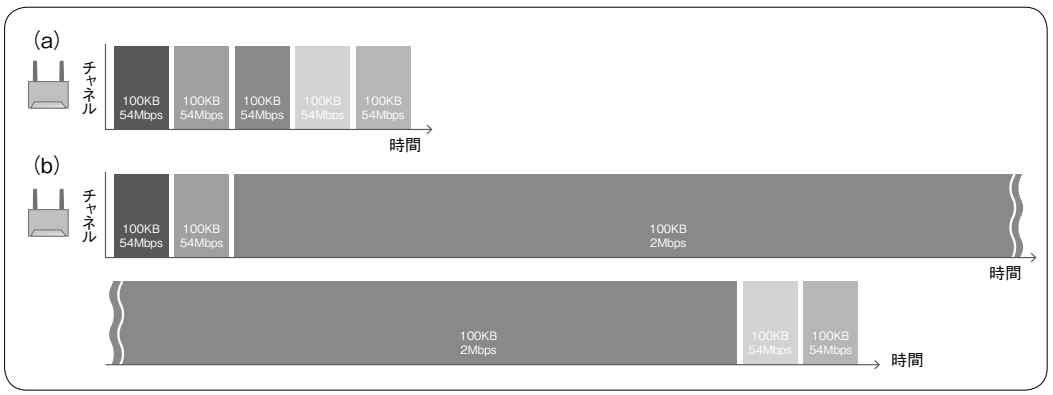
▼図3　1つのチャネルを複数台の端末で共有（5台の端末に100KBずつ転送すると仮定）注2



▼図4　占有時間は通信内容によってまちまち。送信されない時間もある



▼図5　全端末が54Mbps変調の場合（a）と、うち1台が2Mbps変調の場合（b）にかかる時間の比較（概念）



ように5台の端末が100KBずつ同じ54Mbpsのデータレートで計500KB転送することを考えます。どの端末も同じ時間で転送が終了しました(めでたしめでたし)。ところが、次にまた同じデータを転送しようとしたら、特定の端末が遠くへ行ってしまったために通信状況が悪化し、データレートが2Mbpsに下がりました。同じく100KBずつ転送を行います(**図5**)。

5台の端末が同じ100KBを転送したのですが、1台の端末のデータレートが遅いだけで全体の時間リソースが5倍ほど無駄になってしまいました。

図5での時間は、当然ながら有限です。データレートの遅い端末が1台あるだけで、そのチャネルに収容できる端末数が大幅に低下します。

通信速度の遅い端末が全体の能力を低下させる

遅い端末の存在は、空間全体の足を引っ張ることになります。無線LANでは制御も同一空間で行いますので、ここが輻輳（ふくそう）してしまうと制御も正常に行えなくなり、不安定になったり切断されたりすることになります。共有された空間において遅いデータレートは悪、とも言えます。

APによっては、遅いデータレートを禁止する機能を持ったものがあります。これを利用すれば利用効率の向上を図れます。

しかし、通信速度は電波状況によってリアルタイムに変化しています。速いデータレートほど通信状況の悪化に対して耐性が弱いため、遅いデータレートを禁止すると今度は少しの電波状況の悪化でも頻繁に切断されてしまうことになります。

APをたくさん設置すれば良いというものではない

APをたくさん設置して電波状況が悪化しないようにすれば良い、確かにそのとおりです。

注2）分割された通信のあいだに少し隙間が空いていますが、ここは誰も送信していないことを確認する時間です（CSMA/CA方式）。

Column 身近な無線LANの電波を見てみよう

端末で無線LANの接続先を選択するときに、近くにあるAPのSSIDと電波強度のアイコンが一覧表示されますが、もう少し詳しく見る方法があります。

Mac OS Xではairportコマンドで周囲の無線LANをスキャンできます。ターミナルで次のように入力すると、**図A**のように表示されます。

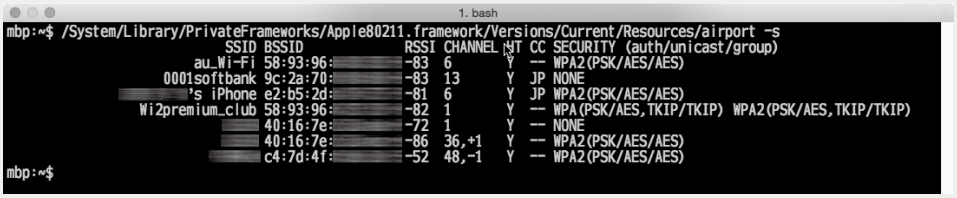
```
$ /System/Library/PrivateFrameworks/Apple80211.framework/Versions/Current/Resources/airport -s
```

スキャン結果を可視化してくれるユーティリティもあります(**図B、C**)。

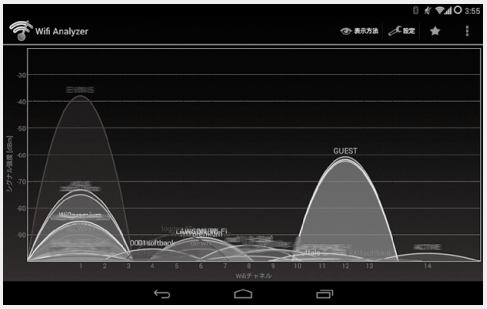
これらは、周波数(チャネル)ごとに受信された電波の強度を表示しています。ここからわかるのはAPの存在です。つまり、たくさんのSSIDが検出されているからといって、そのチャネルが混雑しているとは限りません。強力なAPが複数存在していても、利用率が低い場合にはほとんど問題になりません。逆にAPが1つだけでも、多数のクライアントが存在していたり、常時大量のトラヒックが流れていたりするようなケースでは、干渉源として問題になり得ると言えます。

混雑状況を正確に把握するには、専用のハードウェアを持った測定器が必要になりますので、あまり手軽ではありません。しかし、多くのAPが存在しているチャネルは、多くのトラヒックが流れている可能性も高いと言えるでしょう。

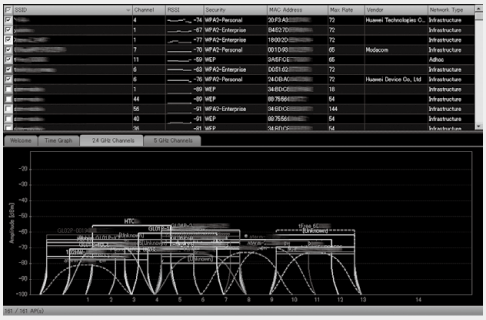
▼図A　airportコマンドの実行結果



▼図B　Wifi Analyzer(Android)



▼図C　MetaGeek inSSIDer(Windows、Mac OS X)



しかし次に、同じチャネルを使っているAP同士が干渉するという問題が出てきます。結局、「チャネル×空間」を共有している限り、干渉してしまうのです。

先ほどから述べている「時間と空間の共有」という考え方については、まだ直感的にピンときていないのではないでしょうか。次節にて電波の特性を整理しながら、もう少し詳しく解説していきます。

そもそも電波って何

電波は目に見えませんし、耳にも聞こえません。光よりも周波数の低い電磁波であり、広い意味では光の仲間です(**図6**)。人間の可聴域から超音波に近いような非常に低い周波数の電波まで広く利用されていますが、電磁波と音の弾性波は別のものですので、人間が直接聞くこと

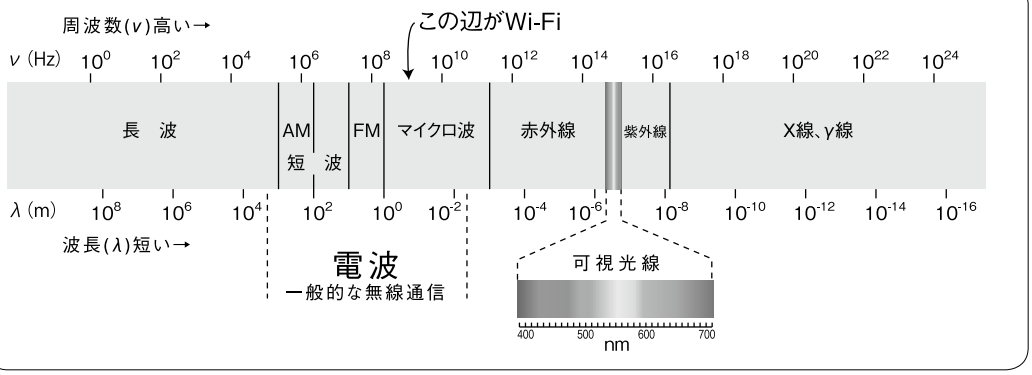
はできません。

電波は周波数によって特性が変わります。周波数の高低どちらかが高性能である、ということはなく、それぞれに特徴があります(**図7**)。光に近い高い周波数では光と似たような伝搬特性となり、逆に低い周波数では、ものを貫通したり、地面を這ったり、電離層反射など複雑な伝搬をします。

無線LANで利用されている周波数帯は比較的高いため、どちらかと言えば光に近い伝搬特性であると言えます。光と同様に、反射、屈折、回折することもあります。

電波特有の性質としては、導体や水分をほぼ貫通しない、石やコンクリートも貫通しにくい、ガラスなどは貫通する、導体が近くに存在すると相互作用を起こしてしまう、などがあります。人体は水分をかなり含んでいるため、誰もいないホールと満席のホールとでは状況がかなり変

▼図6　電磁波の周波数帯域(電磁スペクトル：Electromagnetic spectrum)



▼図7　周波数と性質

周波数	低 → 高
波長	長い → 短い
伝送容量	狭小 → 広大
飛び方	いろいろある → ほぼ直進
	障害物を貫通 → 反射／吸収
	減衰しにくい → 減衰大
	導体を貫通しない
	水分を貫通しない
指向性	作りにくい → 作りやすい
アンテナ	大型 → 小型

無線LANはこのあたり

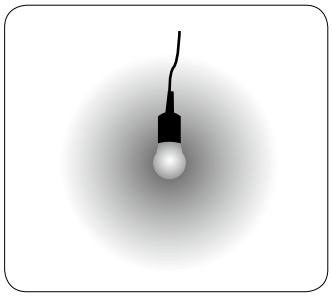
Column フレネルゾーン

無線LANの電波はコンクリートの壁や金属のドアを貫通できないことを説明しました。でも、壁にガラス窓があれば窓から貫通できますね。窓が小さかったら、どうでしょう？　すごく小さい窓だったら？　針穴くらいだったら？　APと端末同士が可視でありさえすれば通信できるのでしょうか？

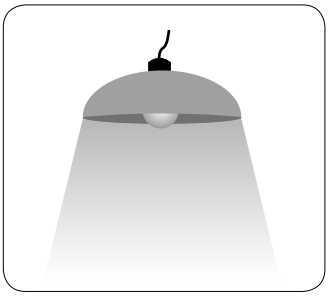
実際には電波は線状に進むのではなく、あるサイズの回転楕円形に広がって伝搬します。これをフレネルゾーンと呼び、この範囲に障害物があれば電波が届きにくくなります。

オフィスビルなどの無線LAN密集地帯では、金属羽根のブラインドを下げることで外からのノイズを減らせることがあります。ブラインドの羽根を開けても、ノイズは減ったままです。ブラインドの隙間くらいでは、無線LANの電波はなかなか通れません。

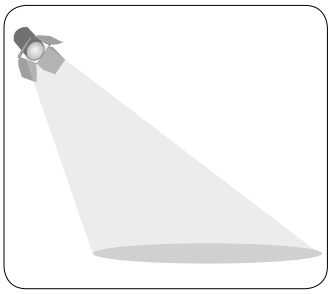
▼図8　完全無指向性のアンテナは、裸電球のようなもの



▼図9　指向性のアンテナはカサ付きの電球のようなもの



▼図10　パラボラアンテナはスポットライトのようなもの



わることがあります。カンファレンスネットワークにおいては、無線LANの状況は本番になってみないとわからないことがしばしばあります。

光に見立てて考えてみる

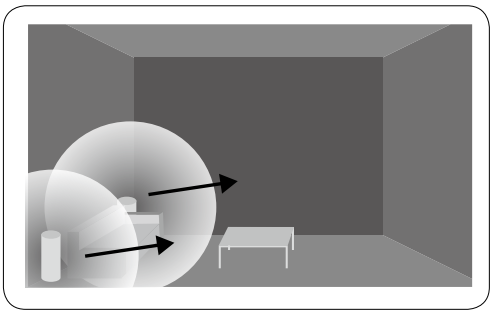
無線LANの周波数は比較的高いため、伝搬特性は光に似てきます。とはいえ、およそ光であると言えるほど似ているわけではありません。これは見えない電波をイメージするための喩(たと)えです。

APからの電波は裸電球だとします。電波は四方八方へ、球状に発射されていきます。無線LANで言えば、完全無指向性のアンテナです(**図8**)。

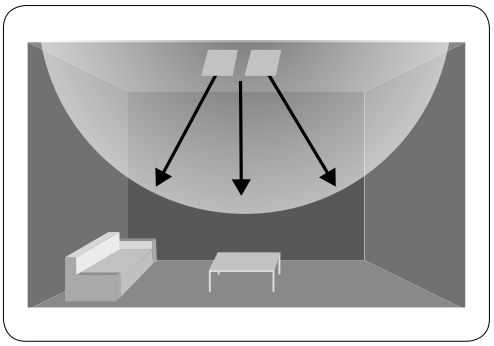
電波の効率を考えると、カサがあったほうが良いでしょう。利用したい方向へ反射させて強めることができます。不要な方向へ発射している電波は、ほかの無線の干渉源になりますから、やはりカサで遮ると良いでしょう。無線LANで言えば、このカサは指向性を持ったアンテナです(**図9**)。天井に取り付けるタイプのAPに内蔵されたアンテナでは、このような特性を持っているものが多くみられます。

使用したい場所が決まっているのなら、スポットにするのが一番です。電波を非常に強くできますし、干渉も大幅に改善できます。無線LANで言えば、非常に強い指向性を持った八木アンテナまたはパラボラアンテナです(**図10**)。ビル間接続や、非常に高密度かつ小さなセル分割が要求されるスタジアムなどで利用されます。無線LANではあまり一般的なアンテナではありません。

▼図11　床に照明を置く



▼図12　天井にシーリングライトを設置する



アンテナを床などの低いところに設置してしまうと物陰ができ、通信が不安定な個所が多く発生します(**図11**)。

やはり、シーリングライトのように天井にAPを設置するのが理想的です(**図12**)。

電球のワット数が送信電力だとすれば、電球の周りにあるカサやスポット装置はアンテナです。カサやアンテナ自体には、電球自体を明るくする効果はありません。できることは、四方

八方に散らばってしまうパワーを一定の方向へまとめることだけです。狭い範囲へまとめるほど、結果として強力な電波になり、受信感度も高くなります。

複数の光源が混ざってしまう

図13のように、同じ色の照明が複数ある場合を考えてみます。これは困りました。それぞれ違う色であれば良かったのですが、同じ色では区別できなくなります。無線LANでも、1つのチャネルに複数のAPがあると干渉が起きます。光や電波が届かないほど遠いところへ行ってしまえば、または周波数(チャネルまたは光の色)が違えば、この干渉は起きません。電磁波の周波数と、それが到達できる空間の大きさが、端末を収容できる空間の単位(セル)となります。

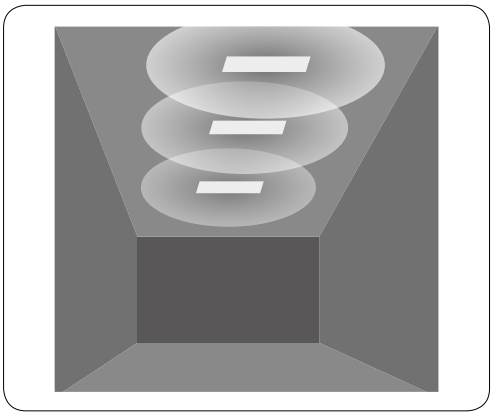
「時間と空間の共有」とは

このセル内を何らかの電波が常時送信されている状態を利用率(デューティー比)100%とし、実運用では、これが100%に達することのないようにします。

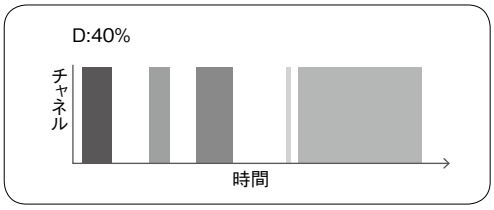
①チャネル×空間あたりの時間は、図14のイメージのようにチャネルに存在する全端末で共有されている
②何らかの電波が送信されている時間(デューティー比)が図15のように100%に近づくにつれ衝突などでエラーが増えてくる
③図16のように通信状況の悪化のためデータレートを下げる端末が増えてきて、利用効率が急激に悪化
④通信状態が悪化するとエラーのため再送が起こり、さらにデューティー比が悪化する→破滅!

つまり、1つのチャネルを共有している限り、同じところにAPをいくら増設しても改善されません。ボトルネックは伝送媒体である空間そのものであるためです。高密度な無線LANではこの「チャネル×空間」をいかに分割していくかが重要になります。

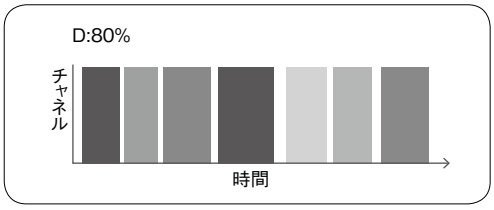
▼図13　広い場所の天井に複数のライトを設置する



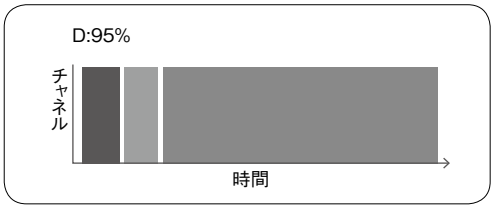
▼図14　チャネル×空間あたりの時間(デューティー比)は全端末で共有する



▼図15　デューティー比が高くなってきた場合



▼図16　特定端末の通信状況悪化でデューティー比が極限まで高まった場合



カンファレンスネットワークでの構築ポイント

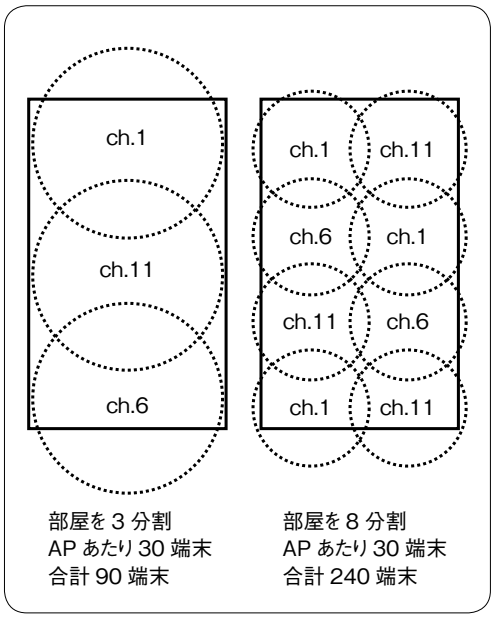
空間の分割

1つのチャネルやAPでカバーしている範囲に多くの端末が存在してしまうと、必然的に輻輳が起こります。チャネルあたりに収容できる合計スループットの上限はデータレートで決まります。事前に高密度になることがわかっているエリアは、カバーする範囲を狭くして、遠くまで電波が届かないようにして、収容する端末が多くなり過ぎないようにします。狭くなったカバレッジはAPを増やすことで解決します。

図17では、同じ部屋をそれぞれ3個、8個に分割する案です。仮に、1つのチャネルで端末を30台収容できると仮定すると、それぞれ90台、240台の端末が収容できることになります。

ここで問題になるのが、電波の飛び過ぎです。5GHz帯はチャネル数に余裕があるため多少飛び過ぎたとしても大きな問題にはなりませんが、

▼図17　空間の分割例



Column 電波の強さと距離

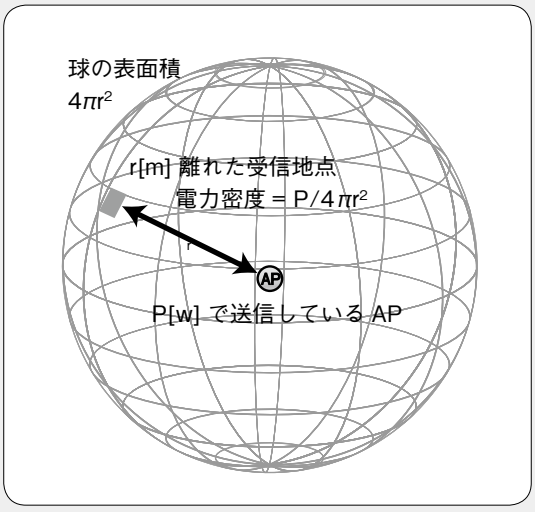
「電界強度は距離の2乗に反比例する」と耳にしたことがある方もいらっしゃるかもしれません。無線APの出力を10mWから20mWにしても2倍遠くまで届くわけではないのです。なんだか面倒ですね。なぜ2乗なのでしょうか。

電波を四方八方へ球状に送信する点状の送信アンテナを想定してみますと、ある受信地点での電力は、送信アンテナから受信地点の距離をを半径rとした球面上のある一部の面積となります。球の表面積を$4\pi r^2$とすると、送信アンテナの電力Pと、ある受信地点の電力密度PDは、$P/(4\pi r^2)$となり、距離の2乗に反比例することがわかります(**図D**)。

電波の送信出力を高めることは通信品質を上げる1つの手段ですが、干渉源を減らすことも同じくらい効果のある手段です。不要な方向へ飛び過ぎている電波は、ほかの無線設備のノイズとなります。

状況が許されるのであれば、指向性のあるアンテナを使うのが、もっとも効果のある手段です。指向性は受信されてくる干渉源をカットするのにも効くのですから。

▼図D　電波の強さと距離の関係



2.4GHz帯には同時に利用できるのは3チャネルしかありません。1つの会場で同じチャネルを重複して使わざるを得ないことがあります。セルを小さく分割するには電波を遠くへ飛ばないようにする必要があります。スポットライトのように指向性の鋭いアンテナを高いところに設置し、細かく空間を分けていくのが理想ですが、一時的なカンファレンス無線ではそのような設置はとても望めませんし、アンテナも高価です。さて、どうすれば良いでしょうか。可能な範囲でできることを検討してみましょう。

一般家庭での無線LANでは遠くまで飛ぶのは良いことですが、カンファレンスネットワークでは逆にもっとも厄介な問題となります。

▼表2　電波の飛びをコントロールする方法

	遠くまで飛ぶ	遠くまで飛ばない
APの設置位置	見通しの良い高い位置	低い位置
データレート	遅い	速い
送信出力	大きい	小さい

電波の飛びをコントロールするには、**表2**のような手法があります。

この中で、電波の飛び自体をもっともコントロールしやすいのはAPの設置位置です。低くすると、人体や各設備その他のさまざまなものに遮られ、遠くまで飛びにくくなり、遠くのノイズも拾いにくくなります。

エンタープライズ向けAPでは、データレートを制限できる機種があります。前述のとおり、データレートが遅いほど、弱い電波でも通信可能になります。これを逆手に取って、遅いデータレートを拒否することによって、弱い電波での通信をできないようにします。

送信出力はもっともわかりやすい手法ですが、ほかの2つと比較してコントロールしづらい印象があります。近距離の条件まで悪化しやすいのと、受信感度はそのままという問題があります[注3]。

注3) 受信感度を低下させないのは衝突を防ぐ意味で有用なケースもあるため、一概にデメリットとも言い切れない。

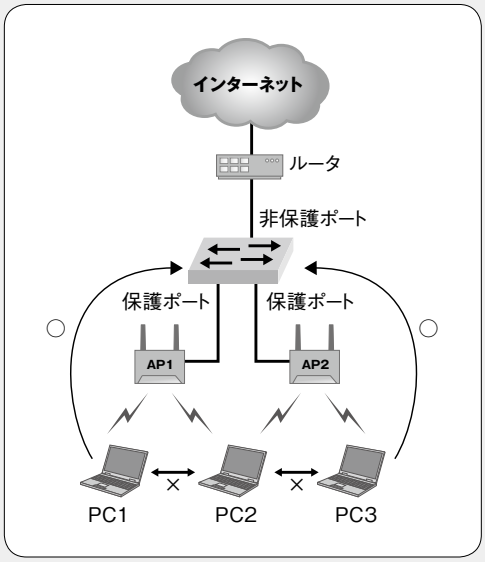
Column 保護ポート

保護ポートとは、端末同士の通信を禁止する機能です。来場者同士でエクスプローラやFinderの共有ディレクトリが見えてしまうと、セキュリティ上の問題もありそうです。保護ポートでは、野良DHCPサーバの問題もDHCPスヌーピングを使用せず解決できますし、LAN経由で感染を広げるマルウェアへの対策にもなります。

ただし、ハッカソンなどでは端末同士が見えないと作業を困難にするケースがありますので、必ず保護ポートに設定するとは限りません。

APの設定では、「Public Secure Packet Forwarding(PSPF)」、「P2P Blocking Action」、「bridge-group n port-protected」などという名前になっています(ベンダ、OS、UIによる)。APが1台だけならこの設定で完了です。しかし、APが複数台ある場合は、上位のスイッチ経由で端末同士が通信できてしまうので、APが接続されているスイッチのポートもすべて保護ポートに設定(switchport protectedなど)してください(**図E**)。

▼図E　APが複数台ある場合の保護ポート設定



譜面台の利用

CONBUでは、次のような理由から、APを設置するのに譜面台を利用しています(**写真1**)。

・高さ、角度などを簡単に調整できる
・会場に備品として用意されていることが多い(購入しても安価)
・椅子などと違い、来場者に間違えられて座られる恐れがない
・SSIDなどの来場者向け情報を紙で貼り付けやすい

カンファレンスネットワークでは、事前に念入りな調査や無線設計ができることはほとんどないため、本番時に高さなどを簡単に調整できるものだと便利です。

ローミング

1つのSSIDで複数のAPを設置すると、端末はもっとも近くて電波状況の良さそうなAPを自動的に選んで接続します。端末が物理的に移動するなど、電波状況の変化により別のAPのほうが良くなると、切断されることなく自動的に接続先を変えます。端末からは1つのSSIDに見えるため、端末利用者が意識する必要はありません。

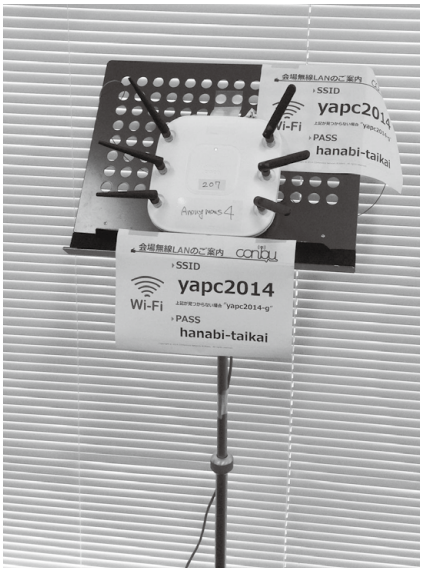
データレート制限

無線LANは電波状況に応じてリアルタイムにデータレートを変化させていますが、これを制限できるAPもあります(**図18**)。データレートが遅いほど、弱い電波でも通信可能になります(データレートが半減するたびに信頼性が約30%向上する)。これを逆手に取り、遅いデータレートを拒否することによって、弱い電波での通信をできないようにさせ、早めにローミングを行わせます。チャネルの利用効率の観点からも、遅いデータレートの端末が存在することは良いことではないため、この設定はたいへん重要です。

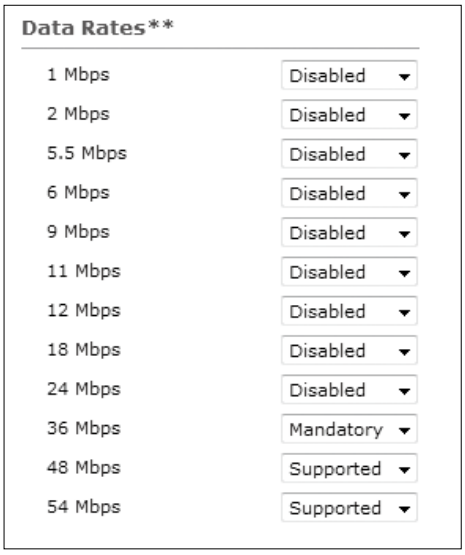
APが定期的にSSIDなどの情報をのせて放送しているビーコンは、最も遅いMandatory(Required)データレートで送信されます。ビーコンを速いレートで送信すると遠くの端末はAPの存在を検知できなくなり、遅めで送信すると遠くの端末からも見つかるようになります。

これを利用すると、メインホールなどの高密度な場所では速いレート、ホワイエなど来場者がゆっ

▼写真1　譜面台を利用してAPを設置



▼図18　データレート制限の設定例



たりと歓談する場所では遅いレートで送信することで、メインホールのカバレッジを狭く、ホワイエのカバレッジを広くコントロールできます。もちろん、この場合はホワイエが満員電車のように高密度になるとつながらなくなってしまいます。ですので、そこはカンファレンスやパーティの参加者になったつもりで、「ここではちょっと休憩するのに良いな」とか、「ここは知人と待ち合わせするのに良さそうだな」とか、そのようなことを想像しながらAPの配置と台数、データレートや送信出力を決めていきます。慣れれば、予想を大きく外さなくなってくると思います。自身もまた、カンファレンスの来場者なのですから。

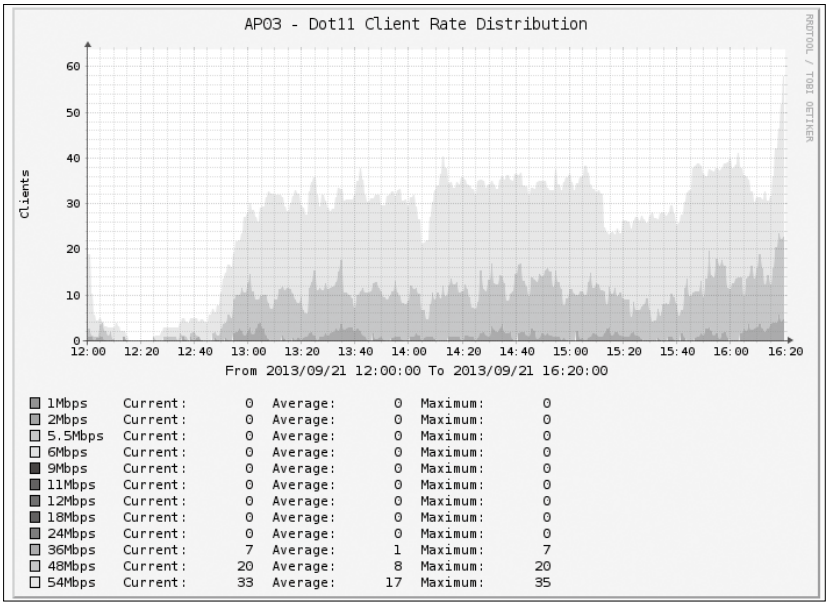
カンファレンスネットワークの実際

図19は、あるAPに接続されたクライアントとそのデータレートの推移です。**図18**で設定しているように、24Mbps以下での接続を拒否しています。わずかに36Mbpsに下がっている端末もあります。

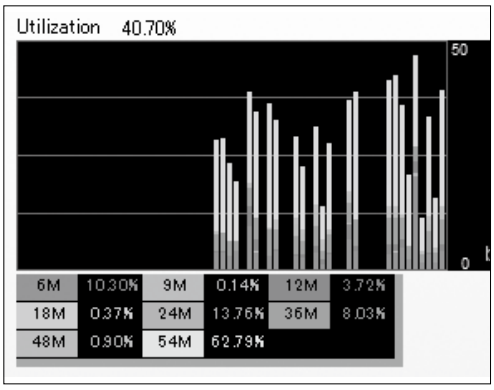
このときは約40台の端末が接続されている状況でしたが、AirMagnet[注4]で測定してみると、チャネル全体の約62%がデータレート54Mbpsで転送されており、全体のチャネル利用率(≒デューティー比)も40%程度と正常です(**図20**)。この瞬間のチャネル全体のスループットは約17Mbpsで、カンファレンスネットワークにおいては十

注4) 無線LANの測定／分析を行うツール。

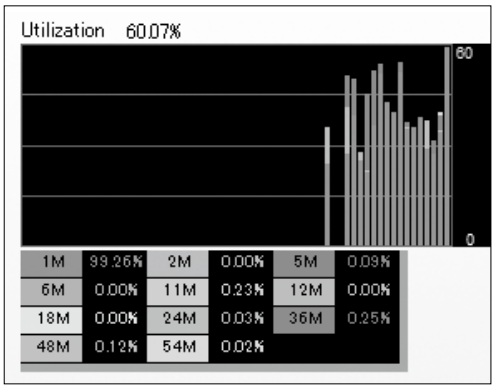
▼図19　データレート推移の例



▼図20　AirMagnetによる測定結果(良いケース)



▼図21　AirMagnetによる測定結果(悪いケース)



分といって良いと思います。

もし、遅いデータレートの通信がこのチャネルに存在するとどうなるでしょうか。利用率を圧迫し、チャネル全体の状況が悪化する可能性があります。**図21**のような悪いケースでは、トラヒックの99%以上がデータレート1Mbpsになっており、チャネル利用率が60%を超えているにもかかわらず、チャネル全体のスループットは0.7Mbps程度しか出せていません。チャネル利用率が高まると衝突などでエラーが起きやすくなり、エラーは再送やデータレートの低下を招き、さらなるチャネル利用率の悪化となります。結果として、つながらない無線LANになってしまいます。

設置するAPにて遅いデータレートを禁止できたとしても、来場者が持ち込むポータブル無線LANルータなどは遅いデータレートでの通信を行って悪影響を及ぼすことがあります。そのため、電源を切っていただくなどの来場者の協力が重要になります。

ポータブル無線LANルータは通信を行っていなくても、SSIDなどの情報を遅いデータレートで常時送信していることも多く、台数が多いと問題になることがあります。また、来場者が会場間を移動するなどしてカンファレンス無線が圏外になると、意識せずとも端末がポータブル無線LANルータに接続されます。そのままの状態でカンファレンス会場に移動してきてしまうケースもあります。

APの配置

APは、無線的に必ずしも理想的な配置にできないことも多々あります。来場者の近くに設置することが多いため、倒れたりひっかけたりするなどの事故が起こらない配置が求められます。来場者が横切るようなケーブルの敷設は可能な限り避け、やむを得ず行う場合は十分な養生が必要です。ケーブルの養生は時間のかかる工程ですが、超短時間で構築を完了させるカンファレンス無線では、養生の工数を考慮してAPの配置を変えることもあります。

イーサネットケーブルを通じて電源を供給するPower Over Ethernet(PoE)も多く利用しています。スイッチからAPに1本のケーブルで、ネットワークと電源を同時に供給できるため、APを設置する個所には電源がなくても良くなり、APの配置の自由度を高められます。また、機材の総数が減らせるのも大きなメリットです。APごとに電源アダプタやACケーブルやタップなどを用意していると、構築時間もさること

Column カンファレンス以外のイベント無線LAN

これは筆者が個人的にかかわった事例になりますが、ネットレーベル[注A]が主催するクラブイベント(パーティ)でも、無線LANを提供しています。開催場所が地下など、電波が入りにくい場所が多いことに加え、ネットレーベルならではの試みのために、フリー無線LANはなくてはならない存在になっています。

国内最大手のネットレーベルの1つであるMaltine Recordsのイベントでは、来場者のスマートフォンからVJ(Video Jockey)や照明を直接コントロールできるインタラクティブ要素や、アーティストにiBeaconを持ち歩いてもらい、来場者は会場のどこかにいるそのアーティストに接近するとボーナストラックをダウンロードできる企画なども行いました。もちろん、ネット発のレーベルであることもあり、会場内でもインターネット上でのコミュニケーションが活発です。

カンファレンスネットワークでは、来場者は基本的に着席していますが、クラブイベントネットワークは立っていることが多いため、さらに高密度になること、来場者は暗いところを動き回るため安全にAPを設置できる場所が限られること、大音響でハードディスクの緊急モーションセンサが頻繁に作動してしまうなど、カンファレンスネットワークとはまた別の難しさやおもしろさがあります。

注A) おもにインターネット上で運営されているインディペンデント・レコードレーベル。

ながら機材管理の工数が大きくなります。

家庭用APでも構築できる?

エンタープライズ向けの機材を使わなくても、どこでも安価に手に入る機材で、誰もが簡単にカンファレンスネットワーク向けの高密度無線LANを構築できるようにするのが、CONBUの目標の1つです。

エンタープライズ向けAPにある管理機能やVLAN対応なども重要ですが、もっとも必要なのはデータレート制限です。便利な管理機能などなくても動作はしますが、遅いデータレートの拒否ができなければ、高密度な無線LANは構築不可能です。逆に言えば、家庭用APでも遅いデータレートを拒否できる機種であれば活用できるかもしれません。

そのような機種は、筆者の知る限り国内では市販されていません。もしご存じの方がいらっしゃいましたら、ぜひとも教えていただきたく思います。

まとめ

本章で述べてきたカンファレンスネットワークを構築する際のポイントをまとめます。

- 無線LANの電波は光に似た伝搬特性がある
- 空間が常時、電波でいっぱいにならないようにする
- データレート制限が必要不可欠
- 安全第一、構築時間優先
- 譜面台は便利

もし勉強会やイベントなどで自分たちで無線ネットワークを構築するようなことがあれば、これらの点を参考に実施してみてください。SD

Column LANと電源を1本のケーブルで供給

Power Over Ethernet(PoE)とは、通信に利用する既存のLANケーブルに電源も同時にのせて供給する技術です。ケーブル1本で済むため、機器の設置場所で電源をとる必要がなくなり、見た目もすっきりします。USBがデータと電源供給を1本のケーブルで実現できているのと似ていますね。

USBは規格策定時より電源供給が考慮されていますが、イーサネットはそうではなかったため、当初、PoEはベンダ間の互換性がありませんでした。ツイストペアケーブルには電源用の線はありませんから、既存のケーブルをそのまま利用しつつ安全に電源供給するのは簡単ではありません。2003年にIEEE 802.3afとして標準化され、これに準拠した機器は異なるベンダ間でも電源供給が可能です。

PoEは、給電/受電の両方に対応機器が必要です。給電にはPoEスイッチと呼ばれるもので行うことが多いでしょう。給電非対応スイッチにPoE機器を接続した場合は、電源が入らないだけですが、受電非対応機器(イーサネットに限らず同じRJ45コネクタを持った機器)に電源を供給してしまうと、機器の損傷、火災などにつながる恐れがあります。そのため給電側は、受電機器の存在を検出するまで給電を開始しないようになっています。

APは10W以上の電力を必要とするものが多いですが、1台のPoEスイッチに多数のAPを接続すると、PoEスイッチの電源装置の能力を超えてしまうことがあります。APより低消費電力のIP電話機などでは多数接続しても問題ないこともあります。そのため、PoE受電機器は必要な電力の要求を給電側とネゴシエーションを行い、可能であれば要求された範囲で給電するようになっています。もし給電側の能力を超える場合、設定した優先順位でほかのポートを落として給電できるものもあります。多くのPoEスイッチでは、ポートごとに供給電力などの状況をモニタできます。

以上のように、電源供給を想定していなかったケーブルへ安全に電源をのせるために、意外と(?)賢いことをしているのです。IEEE802.3af/IEEE802.3atに正しく準拠した機器同士では事故が起こる可能性はとても低いと言えますが、まれに手順を守らないPoEスイッチやパワーインジェクタが格安で売られていると聞きます。筆者は興味本位でも怖くて手が出せません。

第3章

構築・運用時のトラブルリスクを下げるクラウド活用

機材搬入～配線～撤去までを配慮するからわかること

Writer 高橋 祐也(たかはし ゆうや) CONBU Twitter @albee824
Writer 岡田 雅之(おかだ まさゆき) CONBU Twitter @smadako

前章では無線LANの接続品質を高める取り組みを述べましたが、本章では、無線APからインターネットへつながる有線ネットワークの設計と構築について紹介します。限られた時間のなかで最高のパフォーマンスをあげるために試行錯誤してきた経験を、これまで設営してきたネットワークの変遷を通してお伝えします。

カンファレンスネットワークで考慮すべきこと

一般的なカンファレンスネットワークで提供されている機能は、一般家庭のホームネットワークと大きく変わりません。一番大きな違いはネットワークに接続される端末の数や同時に利用する人数です。一般家庭で同時に接続する端末の数は、多くて数台から10台程度になるかと思います。しかし、IT系のイベントであれば数百人単位の端末がネットワークに接続されることになります。そのように規模が大きい場合、一般家庭用と同じ装置や構成でネットワークを提供することは非常に難しいです。

難しい理由として、一般家庭向きのブロードバンドルータでは性能や機能の限界値に達してしまうためです。具体的に性能が不足しがちな機能としては、1つのIPアドレスを複数のユーザで共有をして通信を行うNAPT(Network Address Port Translation：コラム参照)というしくみです。一般家庭で利用されるようなブロードバンドルータであれば、NAPT可能なセッション数が数千という装置が多いです。NAPTのセッションは、ユーザが1つのWebページを開いただけでも数十セッション利用されることもあります。ほかにもさまざまなアプリケーションが通信を行っています。とくにIT系のイベントに参加する層はコアなユーザが多いため、一人あたり利用するセッション数が多くなりがちです。そのため、NAPTのリソースは容易に枯渇してしまいます。このNAPTのリソースがなくなると、ユーザが新しい通信を始めることができなくなります。ですので、機器選定の際にはNAPT可能なセッション数を確認することが多いです。

ほかにも、ネットワークに接続をしてきた端末にIPがアドレスを払い出すしくみのDHCPも、規模が大きくなるとルータに内蔵されている払い出し機能では対応できなくなる場合があります。すると、ネットワークに端末をつなぐことはできるがIPアドレスを取得できずに通信ができない状態になります。しかし、ルータなどで動作するDHCPサーバ機能の詳細な性能が表示されていることは少ないです。そのため、CONBUが構築するカンファレンスネットワークでは、別にサーバ機器を用意してUnix/Linux系OSの上でDHCPサーバを動作させている場合が多いです。

また、今日のインターネットを利用するうえで欠かせないDNSによる名前解決も、ルータのNAPTのセッションを枯渇させる要因の1つです。一般的にルータがフルリゾルバ(p.40のコラム参照)となることはなく、ISPのフルリゾルバに転送をしています。その際にもルータ

のリソースを消費してしまいます。そのため、フルリゾルバはNAPTが必要ないネットワーク上に設置して、名前解決が行えるようにすることが望ましいです。CONBUが構築するカン

Column NAPTのしくみ再入門

1対1で通信を行うときは、重複しないユニークなIPアドレスを用いる必要があります。しかし、家庭にあるPCやスマートフォンすべてに対してインターネット上でユニークなIPアドレスを割り当てるだけのリソースがIPv4には存在しません。そのため、インターネット上でユニークな1つのIPアドレスを家庭内や企業内で共有するしくみとしてNetwork Address Port Translation(NAPT)があります。

家庭内や企業内のネットワークはRFC 1918で定義されているプライベートアドレスを用いて構築し、インターネットへの通信はNAPTをしているケースが多いです。

PC(192.168.0.2)からインターネット上のサーバ(198.51.100.3)に対して通信する場合の動作の例を**図A**に示します。

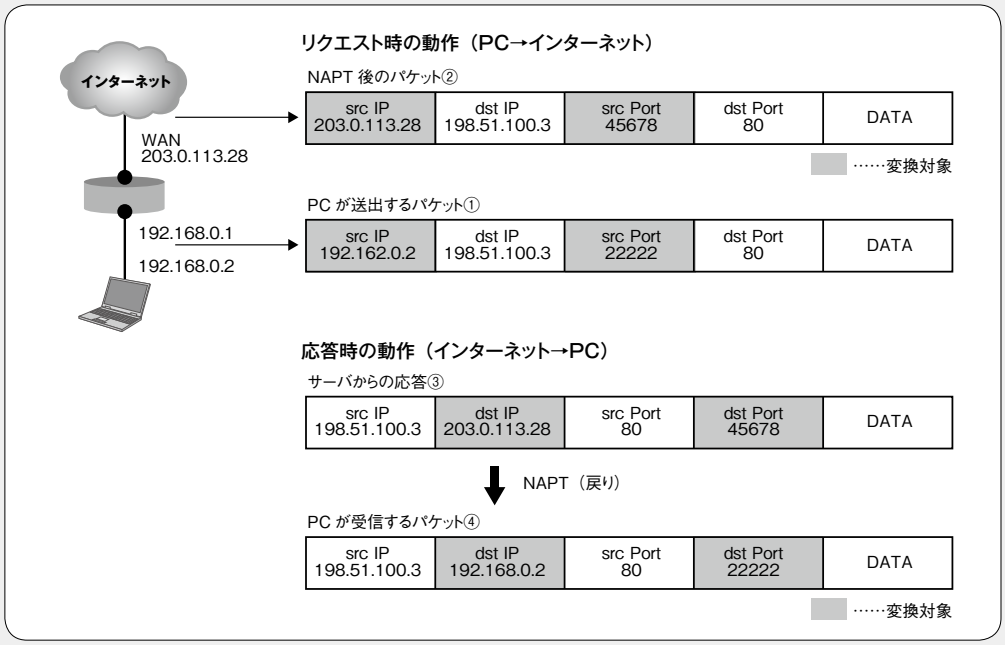
まず、PCが送出するパケットは①です。このときはPCに割り当てられているプライベートアドレスである「192.168.0.2」がソースアドレス(src IP)となります。その後PCに設定されているデフォルトゲートウェイである192.168.0.1に転送されます。

そのときにルータでNAPTを行います。PCから送出されたパケットの送信元IPアドレス(src IP)、送信先IPアドレス(dst IP)、送信元ポート(src Port)と、NAPT後に利用する送信元アドレス、送信元ポートを対応させる表に追加します。実装によっては利用するパラメータが少ない場合もあります。

変換した後のパケットが②です。このパケットがサーバまで到達します。②のパケットを受信したサーバは応答を返します。サーバからはルータのインターネット側のIPアドレス(203.0.113.28)を宛先IPアドレスとした③を転送します。③のパケットを受信したルータは①を②に変換する際に記録したデータを元にパケットを変換し、PCに④のパケットを転送します。

上記のような動作をすることで1つのIPアドレスを複数の端末で共有することができます。しかし、変換前と変換後のパラメータを保存しておくためにルータのリソースを消費します。

▼図A　通信時のNAPTの動作



ファレンスネットワークではフルリゾルバも、DHCPと同様にUnix/Linux系OSの上でサービスを動かして実行する場合が多いです。

CONBUのネットワークの進化：一期網

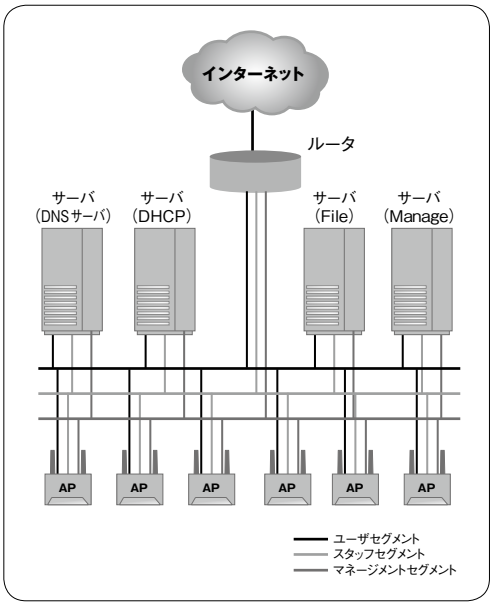
CONBUの前身のLLNOCとして、2013年のLLまつりの際に構築したネットワークを紹介します。このネットワークは**図1**のようにシンプルな構成です。本特集内では、このネットワークを一期網と呼ぶことにします。一期網では、参加者が接続するネットワークとスタッフが接続するネットワーク、そして管理用のネットワークの3面のみのネットワークです。

一期網では、先述のカンファレンスネットワークで気をつけるべきポイントを押さえた装置の選定と設定を行っています。NAPTのセッション数は60,000セッション以上保持できるように設定しています。しかし、NAPTを60,000セッション以上保持できるようにしても、500人ほどの参加者が見込まれるイベントではリソースが枯渇してしまうことも十分にあります。そのため、NAPTセッションのタイムアウトを短く設定する必要がありました。

NAPTのセッションタイムアウトの時間を短くすることで、リソースが枯渇してしまうことを避けることができます。しかし、さまざまな通信やアプリケーションに影響を及ぼします。たとえば、スマートフォンのIP電話アプリやIMAP IDLEでメールサーバと接続しているメーラーなどです。NAPTする装置がパケットの内部まで識別するようなファイアウォール製品でもない限り、セッションが異常終了して残留しているのか、通常状態でセッションを維持しているのか判別することができないからです。

この一期網のNAPTの動作はTCP/UDP/ICMPすべて30秒でタイムアウトという比較的短い時間の設定をしています。それでも、ピーク時にはNAPTの限界値の約半分である30,000セッションを越えるような状態を観測しています。様子を見てNAPTのエントリのタイムアウトを変更するようなことも考えていましたが、とくに参加者からの申告などがなかったため、すべてのタイムアウトを30秒のままで運用を終えています。

▼図1　一期網（LLまつり）



一期網でのトラブルや課題

イベント本番に向けて、事前にホットステージと呼ばれる仮組みをして、装置類の設定や動作確認をする期間を設けます。ホットステージで装置を設定済みの状態にして、会場に持ち込んで設置をするだけでネットワークを提供できるようにしていました。しかし、装置を会場で動作させてみるとサーバが動作しないという問題が発生しました。このままではIPアドレスを参加者の端末に割り当てられずにインターネットへアクセスができない状態になってまうため、急遽、現地でほかのサーバに構築をしなければなりませんでした。

ビジネスで利用されるようなネットワークであれば冗長構成を組み、片系の障害であれば回避できるように設計をすることが多いと思います。しかし、カンファレンスネットワークを運

用する期間はイベント開催中の数日のみです。さらに、装置類をホットステージ会場からイベント会場まで輸送しなければなりません。運搬コストや現場での設営の時間を考えると、装置の台数や複雑なネットワークは無駄になってしまいます。そのため、複数台で冗長構成にすることがすべて正しいとは言えません。

また、イベント開催日の直前まで設定などを見直したいときもあります。しかし、仮組みしたネットワークはイベント本番に向けて一度解体し、運送の準備をしてしまいます。そのため、短期間のホットステージ中にすべての設定を完了させなければならないことが、我々にとってとても負荷が高いことでした。

CONBUのネットワークの進化：二期網

二期網では**図2**のように、一部機能をクラウドサービス上の仮想サーバ（以下、リモートサイト）で動作させる構成にしています。理由は、一期網の設営時に発生したような問題を避ける方法を検討していたからです。しかし、リモー

Column DNSのフルリゾルバとNAPT

CONBUの一期網では、DNSのフルリゾルバ[注1]がNAPT配下にいました。この構成は好ましいものではありません。NAPTのリソースを多量に消費するだけではありません。業界一般的に“毒入れ”と呼ばれる正しい応答に偽装したDNSのパケットを受け入れてしまう確率が増えます。DNSの毒入れは、正しい応答がサーバからフルリゾルバに到達する前に、正しい応答に偽装した偽装パケットが受け入れられてしまうことによって発生します。

通常、偽の応答を受け付けないようにするためのしくみがあります。**図B**のようにsrc Portとtransaction IDで応答を認証しています。どちらも16bitのため65535^2通り[注2]の組み合わせになります。しかし、ネットワークの大容量化や高速化によって偽の応答をつかまされやすくなっています。

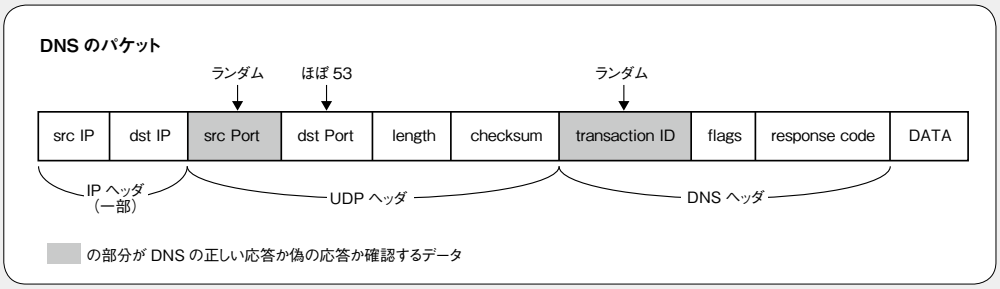
NAPTの配下にDNSのフルリゾルバを置くと、DNSのクエリもNAPTされます。NAPTをすることによって、**図B**中の網かけ部分のsrc IPアドレスとsrc Portが変換されます。このときNAPTの実装によって変換のパターンが異なります。一部の実装ではsrc Portをランダムに選択せず、パターンが存在するものもあります。NAPTのsrc Port選択に一定のパターンが存在すると、DNSのフルリゾルバがsrc Portをランダマイズして送出した効果が薄れてしまいます。NAPT後のsrc Portが推測可能だとすると、TransactionIDだけの認証となってしまうため、毒入れ攻撃を受ける確率が上がります。

DNSの毒入れを防ぐために、UDPの代わりにTCPを利用する方法やDNSSECで認証を行う方法がありますが、利用されるケースはまだまだ少ないです。ですのでDNSのキャッシュに毒を入れられにくくするためにも、NAPT配下のネットワークにDNSのフルリゾルバは置かない構成にするべきです。

注1） フルサービスリゾルバ、あるいはDNSキャッシュサーバとも呼ばれます。名前解決を行うためにドメインツリーをたどりIPアドレスのレコードを見つけ、要求元にレコードを返す動作を行います。

注2） 厳密には使えないポート番号などもあるため、65535^2よりも少なくなります。

▼図B　偽装パケットをチェックするしくみ



トサイトだけに機能を持たせることに不安があったため、会場内とクラウドサービスの仮想サーバの両方に同じような機能を持たせています。一期網には存在しなかったリモートサイトができたため、会場内からリモートサイト内のサーバにアクセスできるようにする必要があります。二期網では、簡単な設定でサイト間のVPNを構築できるOpenVPNを利用して接続をしています。

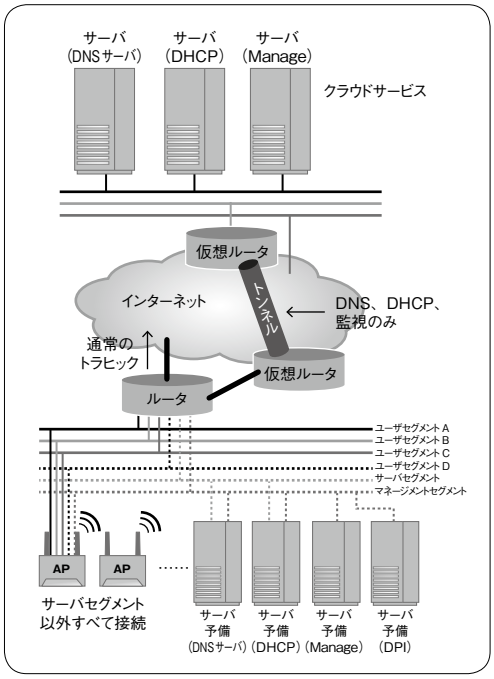
それ以外の違いとしては、一期網に比べて参加者が接続されるセグメントが増えています。また、サーバ類がユーザセグメントに直接足を出していた構成をやめて、サービス提供用のサーバ専用セグメントにしています。これは、リモートサイトと会場とで切り替えを簡易化するためにこのような構成となっています。

ほかは一期網とほぼ同じ構成で二期網を構築しています。DNSのフルリゾルバをNAPTしているネットワーク内に設置せずに済んだことが一番効果がありました。フルリゾルバは名前を解決するために複数のサーバにクエリが投げられます。CNAMEやNSを外部名で指定しているドメインの名前解決は通常よりも多くのクエリが投げられるため厄介です。そのような通信がNAPTのセッションを消費しなくなるのでリソースに余裕が生まれます。また、DNSを利用した名前解決のリスクを軽減することにつながりました。

二期網でのトラブルや課題

この二期網では、大きな問題は発生せずにカンファレンスネットワークを提供することに成功しています。しかし二期網の課題として、リモートサイトと会場の2つのサーバを構築する必要があり手間がかかります。また、持ち込み機材は一期網と変わらない状態です。さらに、OpenVPNを利用しているため会場内のサーバ内で仮想ルータを設定する必要がありました。その結果、複雑になり、運用する人間が苦労することになりました。

▼図2　二期網（YAPC::Asia 2013）



一期網から二期網と構成が変わったことで結果は良くなりましたが、我々の負担も同様に増加してしまいました。このときから、省力化を目指して次のネットワークを検討するようになります。

CONBUのネットワークの進化：三期網

三期網のネットワークでは過去の2つの設計から大きく変更しています(**図3**)。前回の二期網の課題だった、構築にかかる手間を最小限にすることを目指してネットワークを設計しています。また、今まではイベントごとに構築を行っていましたが、三期網の構成はほかのカンファレンスネットワークでも再利用できるようにしています。

手間を最小限にして再利用することにした理由は、CONBUという団体として活動をするようになったためです。イベントごとにネットワークを設計して構築し、検証を行い設営をするのはとても大変なため、少しでも楽にできれば良

いと考えています。

そのための解決策としてすべての機能をリモートサイト側で行うことによって、会場側のネットワークがシンプルになるように設計しています。結果として、会場内にはリモートサイトと会場をつなぐための「VPNルータ」と「スイッチ」と「無線アクセスポイント(AP)」のみとなり、サーバ類を持ち込む必要がなくなります。

サーバ類を持ち込まないことによって、運搬する装置も多少減らすことが達成できています。それ以外にも、NAPTを行うためのソースアドレスを複数個用意することでスケールをさせることが可能になります。二期網で改善されていたフルリゾルバに、グローバルアドレスを直接割り当てることが可能なことも引き継がれています。

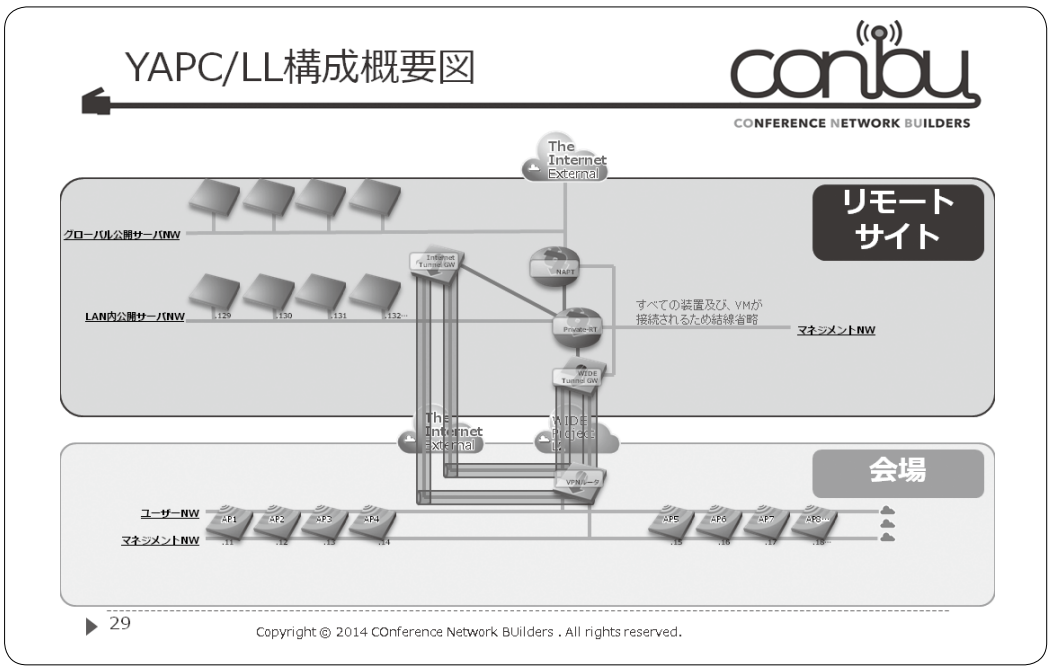
リモートにネットワークを構成する機能の大半を置くことによって、ホットステージも遠隔で効率的に行えるようになります。また、設定後に検証などを行う時間も以前に比べて多く使うことができるようになります。

三期網の詳細

会場側のネットワークは一期網や二期網に比べて簡易になりました。しかし、リモートサイトの構成は**図3**を見ると複雑になっているように見えます。1台のルータにまとめることも可能でしたが、あえて役割を分割することによって後の構成変更に備えた結果です。NAPTをしてインターネット接続を提供している層と、プライベートアドレス空間のルーティングをしている層、会場とのVPNを終端している層の三層構造になっています。

NAPTをしてインターネット接続をしている層はボトルネックになりやすいため、スケールしやすいように分離をしています。たとえば秒間の新規セッション開始数がルータの性能上限を上回ってしまった場合は、他の実装に変更したり新しく追加しなければなりません。そのときに複数台のNAPTする装置にトラヒックを分散させることが可能になります。

▼図3　三期網(LL Diver／YAPC::Asia 2014のネットワーク)



▼表1　トンネリングに用いられるプロトコル比較

	実装(ルータ、Unix/Linux)	NAPT越え	暗号化
OpenVPN	× (ルータでの実装：少)	○	○
GRE	○	×	×
IPsec	○	○	○

また、CONBUのリモートサイトに接続できるように設計しているため、リモートサイトに接続する方法を用意しています。LL DiverやYAPC::Asia 2014のときは、インターネット経由や研究用の網などから会場とリモートサイト間の接続が可能でした。そこで接続方式や網の違いを吸収するために、会場とのVPNを終端している層を分けています。

サイト間接続で用いられている技術

この三期網で会場とリモートサイト間を接続しているのは、IPsecを用いたトンネルです。二期網のOpenVPNから変更しています。よくトンネルに用いられるプロトコルを比較した**表1**を見ながら、その理由を説明します。

二期網で利用していたOpenVPNは、Unix/LinuxやWindowsなどのOSで動作させるのは簡単にできます。しかし、ルータという装置での実装を見たときに利用できるケースはごく稀(まれ)です。そのため、OpenVPNのためにサーバを動かさなければなりません。三期網は先述のとおり、可能な限り運搬する装置を減らすことを目指しているため条件に合わないと考えました。

GREは実装としては多くありますが、IPのプロトコル番号として47を利用するためNAPTを配下で利用することができない環境があります。

IPsecは、ルータやUnix/Linux系OSにも実装があります。また、NAT Traversal(NAT越え)機能を用いることでNAPT配下にいる場合でもトンネルを確立することが可能です。

このようにして適切なプロトコルを選んだ結果、会場とリモートサイト間はIPsecでトンネルを張ることにしました。安全に暗号化用の鍵を交換するためにIPsecで用いられるIKE(Internet Key Exchange)には、IPアドレスも認証に含めるMainモードと、IPアドレスは認証に含めないAggressiveモードがあります。CONBUではAggressiveモードを採用しています。会場によって左右されるIPアドレスに依存しない設計にするためにも選択せざるを得ませんでした。また、NAPT配下のネットワークしかないような環境でもカンファレンスネットワークの提供を可能にするために、NAT Traversalも有効にしています。

三期網の設営準備と結果

カンファレンスネットワークの設営は通常、事前に設営を完全に行うことは難しく、何かしらの不確定要素を抱えたまま当日の設営にのぞむことになります。CONBUでは一期網以後、不確定要素を徐々に減らしていきました。一期網では、すべてのサーバ、ネットワーク機器を現地に持ち込んでいたところを、三期網では基本的に無線APとVPNルータを接続するだけの構成になりました。

ここまで不確定要素を削った三期網ですが、どうしても事前にはっきりとしない部分が残りました。カンファレンスネットワークではカンファレンスだけのために存在する建物に構築することはほとんどなく、何らかの組織のネットワークの一部を借用して会場の隅々までネットワークを提供します。ここで難しいのが、対外ネットワークと会場、サーバ設置場所で一意に設計しなければならない要素の扱いです。IPアドレスだけではなく、VLAN番号について

もケアが必要になります。付加的に会場のネットワークを借用する場合、そのポートがいわゆるTag VLAN[注3]となっているのか、それともPort VLAN[注4]で、後で何らかのVLAN番号が付加されるのか、など会場ごとに調査・調整が必要になってきます。とくに、ある建物とある建物を接続する場合、組織ごとにVLAN番号を重複しないようにケアをして、全体としてVLAN番号の重複などが発生しないようにしなければなりません。

ここまでVLANに関する準備をしても、ネットワークの疎通がなかなかうまくいかなかったというのが三期網の当初の設営結果となりました。少しぼやかした言い方になりますが、三期網の提供開始時に時間を要したのは、このような低いレイヤでのトラブルが複数発生したため解析を行っていたからです。とはいえ、三期網の構成はVPNルータを接続して無線APを予定の設置場所へ設置するのみで設営が完了するため、運搬時の故障・紛失などの大きなリスクを回避することができ、短時間の設営が可能であることが実際にわかりました。

三期網のデメリット

三期網の構成にすることによって、設営や設計の時間を大幅に減らすことができました。しかし、メリットばかりではなくデメリットもあります。

暗号化や復号化を経た後にリモートサイトを経由してインターネットに接続しているため、直接ISPなどを通してインターネットをするよりも通信遅延が発生します。そのため、TCPを用いた通信であればウィンドウサイズの制限により1セッションでの通信速度の低下が考えられます。これはデメリットでもありメリットであるとも言えます。カンファレンスネットワークでは、1人のユーザに帯域を独占されるようなことを防ぐことになります。よって、ほかの参加者への影響を軽減することにつながっています。

ほかにも、リモートサイトにすべての機能を置いているため、会場とリモートサイト間を接続できなければすべての機能が止まってしまうという問題があります。そのため、事前の接続検証を確実に行う必要がありました。LL DiverやYAPC::Asia 2014ではこの部分に起因したトラブルは発生しませんでした。しかし、その後の他イベントで別の装置を使ってIKEをフィルタしていたことが原因で、通信できない問題が発生して肝を冷やしたこともあります。

ルータにとって暗号化を行うのは高負荷な処理です。そのため装置の性能も必要になります。この点に関しては、VPNの性能を高めた小型のルータなども比較的手に入れやすくなっているため大きな課題とはなりませんでした。

取得するべきログ

カンファレンスネットワークの運用でも、ログは確実に取得するようにしています。参加者がネットワークに接続するPCがウィルスなどに感染して、不正アクセスの踏み台などになっている可能性もゼロではありません。

そのような不正アクセスが行われた場合は、発信元IPアドレスによって発信者を特定することになります。しかし、NAPTをしているため発信したユーザの情報は攻撃された被害者が知ることはできずに、CONBUが利用しているIPアドレスから攻撃を受けていることまでしかわかりません。そのため会場内とインターネットの通信記録を保存しておく必要があります。

たとえば、会場内で利用しているIPアドレスとグローバルIPアドレスを変換しているNAPTのログの取得は必須です。このログによって、どのIPアドレスから不正アクセスが行わ

注3）ネットワークを流れるフレームのヘッダに識別番号を付けることでグループを識別する仮想LANの方式の1つ。
注4）ポートを単位に物理的な回線でグループを構成する仮想LANの方式の1つ。

れているのかを確認することができます。その中でも必要な情報としてセッションのオープンとクローズ、そして宛先とポート番号を取得することによって、通信していた時間を特定することが可能です。

NAPTのログだけではIPアドレスしかわからず個人の端末にかかわる情報を知ることはできません。DHCPでIPアドレスを払い出したときのログと突き合わせを行う必要があります。そのためにも、端末固有のMACアドレスと払い出したIPアドレスを記録しています。このログによって、NAPTのログの送信元IPアドレスからMACアドレスを特定することができます。MACアドレスは端末固有となっているため、それを元に調査を行うことができます。

さらにスイッチでDHCP Snoopingを利用することで、DHCPで割り当てられたIPアドレス以外からの通信を規制することによって、確実な発信元特定を可能にできます。そのためには装置やサーバの時刻同期が行われていることが前提です。もしもNAPTのログとDHCPの払い出しのログの時刻がずれていると、ログの突き合わせに時間がかかり、発信元の特定が難しくなります。トラブルなどが発生した場合も、時系列がわからないと原因を特定するのに時間を要することになります。ですので装置やサーバの時刻を常に同期させておくことはとても重要です。時刻同期もすべての装置から外部のNTPサーバを指定するよりも、NTPサーバを網内に用意して時刻マスターとしたほう良いです。世界の時刻と正確に合わせることも大切ですが、同じポリシーで運用している網内で時刻を統一するほうが重要度が高いです。

ユーザの通信ログ以外にもルータやサーバのエラーなども同様に取得しており、トラブルを解析するための材料にしています。ほかにはDNSのクエリログも取得しています。不信な通信を行う端末がいた場合、その端末の前後の

Column MACアドレスからメーカーを調べる方法

NIC(Network Interface Card)はそれぞれに固有のMACアドレスが付けられています。MACアドレスは48ビットで構成されており、8ビットごとに区切って表記します(xx:xx:xx:xx:xx:xx)。先頭の24ビット(3オクテット)は製造元を示しています(OUI：Organizationally Unique Identifier)。そのため製造元が示されているリストと突き合わせることによって、NICの製造元を知ることができます。このリストはIEEEのWebページ[注5]で検索／取得することができます。

最近利用されることが多くなってきた仮想マシンを利用する場合も、決まった範囲から自動的にMACアドレスが割り当てられることが多いです。しかし、起こらないはずのMACアドレスの重複が発生することが稀にあります。そのため、独自にOUIを取得してクラウドサービスで用いている事業者もあるようです。

運用中に観測したMACアドレスから製造者を調べてみました(**表A**)。Apple製品の利用者が相当数いるようです。YAPC::Asia 2014で観測したMACアドレスを、YAPC::Asia 2013のときに取得したOUIのリストで突き合わせをしてみました。すると318個のアドレスが未登録でした。最新のOUIリストを取得して再度突き合わせをした結果、未登録だった318個のアドレスのうち307個のアドレスがAppleによって作られた物でした。

▼表A　来場者の端末MACアドレスから調べたメーカーの内訳

YAPC::Asia 2014		LL Diver	
Apple	1151	Apple	222
Intel	83	Intel	28
Sony	54	ASUSTek	19
LG	46	Sony	12
ASUSTek	31	LG	12
Murata	22	Murata	3
HTC	16	HTC	1
Other	125	Other	71

注5) http://standards.ieee.org/develop/regauth/oui/public.html

▼表2　イベントでの統計情報

	参加者数	接続端末数	秒間最大DHCPリクエスト数	秒間最大DNSクエリ数
LL Diver	約400人	368	38req/sec	117req/sec
YAPC::Asia 2014	1,361人	1,530	15req/sec	200req/sec

※接続端末数はDHCPのログで観測されたMACアドレスの数を数えています。

動きを追うことが可能です。また、統計として活用することもできます。たとえば、GitHubへのアクセスが多いという傾向や、Google、Akamaiなどのハイパージャイアンツへの通信が多い傾向といったインターネットの利用のされ方などを知る材料にもなります。

ユーザの行動を記録する以外にも、全体のトラヒックを監視して装置の限界を超えないか確認しています。今後の装置の増強などの目安としています。

取得ログとIPv6

LLとYAPCのカンファレンスネットワークではIPv6に対応しませんでしたが、カンファレンスネットワークをIPv6に対応させるための課題はまだまだ多いです。前述のログ取得はIPv4での話を前提で書いています。IPv6ではしくみが異なるのでログの取得方法にも違いが出てきます。とくに、端末とIPアドレスを結びつけるしくみと、NAPTをしなくても通信できることが課題となります。

IPv4ではDHCPサーバがステートフルに端末とIPアドレスを管理していたのに対して、IPv6ではRA(Router Advertisement)によってステートレスにIPアドレスが設定されるしくみが用いられることが多いです。そのため、端末のMACアドレスとIPv6アドレスのマッチングはIPv4と比べると複雑で管理しづらいです。DHCPv6によってステートフルにIPv6アドレスを設定することも可能ですが、RA + DHCPv6に比べると装置の対応や端末の実装がこなれていない部分も多いです。

またNAPTをしないため、IPv6を利用した通信をログに記録することが簡単にはできなくなります。ログを取得するためにセキュリティゲートウェイなどを挟む必要があり手間になってしまいます。

カンファレンスネットワークにおいてIPv6の対応はまだまだ課題が多い状況です。今後IPv6が利用されていく中で改善されていくことに期待をしたいです。

機器間接続は基本1本

一般的にアクセスポイントには1本のUTP(Unshielded Twisted Pair：非シールドより対線。平たく言えばイーサネットケーブル)しか接続できません。また、カンファレンスネットワークでは、ケーブルの敷設にあまり手間をかけていられないため、スイッチ間の接続もUTP1本の場合が多いです。しかし、ネットワークは管理用やスタッフ用、参加者用を分ける必要があるため、VLANを利用しています。VLANは難しい技術ではなく最近では低価格のスイッチでも利用できるようになってきています。

運用データの解析

カンファレンスネットワークを設計する場合に、どれくらいの負荷に耐えられる必要があるのかを知る必要があります。あくまでLL DiverやYAPC::Asia 2014での例ですが、秒間の最大瞬間リクエスト数を出してみます(**表2**)。

この結果から、LLやYAPCのスタイルのイベントでは余裕をもたせた設計で、来場者数の1.5倍〜2倍の収容能力にするべきだと言えます。SD

Column 養生の目的

みなさんは“養生”という言葉を聞いてどんな行為を思い浮かべるでしょうか。一般的に、養生とは何らかの壊れやすい物、傷つきやすい商品などをそれらのリスクから守るために行う包装や梱包と考えられるのではないでしょうか。一方で、カンファレンスネットワークを構築する立場からすると、真っ先に思い浮かぶのが、ケーブル配線に関する養生になります。ケーブル配線の養生とは、カンファレンス会場のホールや廊下に敷設したケーブルを保護用の粘着テープで覆うことが一般的です(保護用の粘着テープは、通常マスキングテープや養生テープなどと呼ばれます)。

養生はなぜ行われるのでしょうか。理由の1つは、配線の見た目です。カンファレンスは、華やかであったり、質実剛健な雰囲気を持つ会場で行われることがほとんどです。ですが、その会場をUTPケーブルや光ケーブルがくねくねはっていたら参加者はどのように感じるでしょうか。カンファレンス参加者の運営側への印象も考慮し、配線や機器配置の見た目は軽視されがちですが重要な要素です。

美観も重要ですが、養生を行う重要な理由がさらに存在します。その理由は会場の安全確保にあります。カンファレンスは最低でも数百人、状況によっては数千人の参加者が会場にひしめき合います。ただでさえ込み合っている状況で、廊下や会場内をケーブルが無造作に配線されてしまっていたら、いったいどんな状況になってしまうでしょうか。おそらく、ケーブルに足を引っかけて転んで怪我をしてしまったり、場合によってはケーブルの断線につながり、会場運営者と参加者の両者にとって不幸な結果となってしまう恐れがあります。さらに、何らかの災害時の避難を考慮すると、円滑な避難が可能なように参加者に露出してしまう配線についてはすべて養生し、主要な導線については安全確保が確実に行われるように養生部材などを工夫しなければなりません。

筆者はとある100名規模の会議の運営中に東日本大震災に遭遇しました。そのときは短時間の会議であり、機材配置や人員配置をあまり重要視していなかったのですが、その人数でも導線確保や養生が不足していた配線の問題があったことを覚えています。「なぁ～に。今回はそんな問題にはならないよ」と安全確保については軽視しがちですが、カンファレンスネットワーク構築集団として、常に配線の養生を念頭に置いて配線設計をしています。

養生部材の基本としては、養生テープと呼ばれる“貼る・剥がす”を接着面を破壊せずに行うテープを用います。一見、ガムテープやダクトテープに類似していますがまったく違うものです。なお、養生テープの代わりにガムテープを使用してしまうと、接着面を破壊して塗装や意匠を損ない、会場運営側に多大なる損害を与えることになりますので絶対に避けましょう。この養生テープを床や座席の後ろの隙間にはわせて、引っかかることなく、多少踏んでもケーブルが破壊されないように養生します。

また、中心的な導線を横切るケーブルについては、消耗品の養生テープではなく、マジックテープのように面ファスナーがついたナイロン製の養生部材を使うことがあります(**第1章の写真2**)。こうすることで、養生テープでは破損してしまうような多人数が行き交う場所を強固に養生することが可能になります。また、付加的に“本格的感”があり、見た目にも良好な意匠となります。

このような養生ですが、会場からも養生について指導される場合があります。多くの会場では壁面への養生テープ貼り付けは禁止されているため必然的に床を配線することになりますが、トイレの前や受付場所等、人の行き来が多い場所の導線では、会場側において、実は想定配線経路があったりする場合も多く、カンファレンスNWの経験者として、事前に養生はどこまで行いますか？どういった配線経路がよいでしょうか？などを聞き出すことが、会場との良好な関係を保つ秘訣でもあります。

一方、養生は撤収も考慮しなければなりません。強固に養生テープにて保護した個所は、配線を撤収する際にケーブルを巻き取るだけではなく、養生テープを剥がすといった行為が想像以上に手間取ることに気づくでしょう。そういった点でもナイロン製養生部材は剥がしやすくまとめやすいため重宝しています。養生する際には撤収するときのことも考えましょう。

長々と養生について書き連ねましたが、最後に、ネットワーク配線を行う際には、“いかに養生する必要性を減らすか？”も考慮して全体を考えています。配線はなるべく見えないところ、人が通らないところを配線作業量を考慮して決定し、養生の工数が少なくなるようにしています。次にCONBUが関係するカンファレンスへご参加の際には、少しだけ“養生”についても思いをめぐらせいただければと思います。

第4章

人と人をつなぐカンファレンスを支える
ネットワーク構築のウラガワ

Writer 東松 裕道(とうまつ ひろみち)DMM.comラボ／CONBU Mail tomatsu@tomaz.org
Writer (コラム共著)森久 和昭(もりひさ かずあき)CONBU Mail morihisa.sec@gmail.com

短い構築時間、地理的・時間的にすぐには集まれないメンバー……、本章では実際のネットワーク構築にあたって直面した課題をCONBUがどのように解決してきたかを紹介します。また、カンファレンス参加者との接点を作り、CONBUとの交流を深めてもらうための新たな試みもお伝えします。

会場ネットワーク構築の実際

CONBUは、前身であるLLNOCを含めると、本当にさまざまな会場でのネットワーク構築に挑みました。その中で遭遇した共通的な課題と解決方法を紹介します。

開場まで1時間!?

カンファレンスの開催期間は、カンファレンスの規模に比例して長くなる傾向があります。大規模なカンファレンスであるほど会場ネットワーク構築の難易度が高くなると思われがちですが、実際には小規模なカンファレンスのほうが難易度が高いです。なぜならば、大規模なカンファレンスは開催前日に会場設営のための時間が設けられていることが多く、現場でのトラブル対応やテストを行う時間の余裕がありますが、小規模なカンファレンスではそういった時間が確保されていないからです。前日の設営時間など存在せず、スタッフの会場入りからカンファレンスの開会時間まで1時間程度しかなく、設営できる時間が45分と非常に短いこともありました(**図1**)。

さすがに設営時間が短すぎるため、主催者さんから「お昼ごろに提供できれば良いですよ……」とご配慮いただけたものの、そこは何とかして、カンファレンスの開会宣言までには会場ネットワークを提供したくなるのがCONBUです。そこで会場ネットワークの設営工程と費やした時間を分析したところ、「開梱／梱包作業」と「トラブル対応」の2つがとくに多くの時間を占めていることが判明しました(**図2**)。

開梱／梱包作業

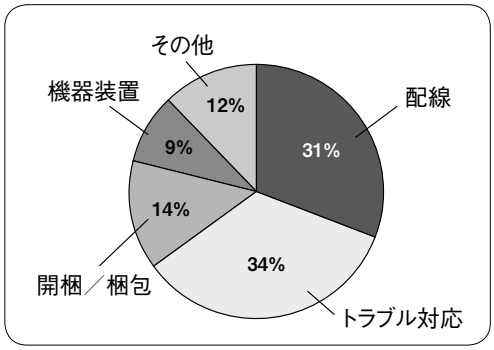
会場ネットワークを構成する機材の中で、もっ

▼図1　設営時間はわずか45分……

開設前の設営時間は実質ここだけ！

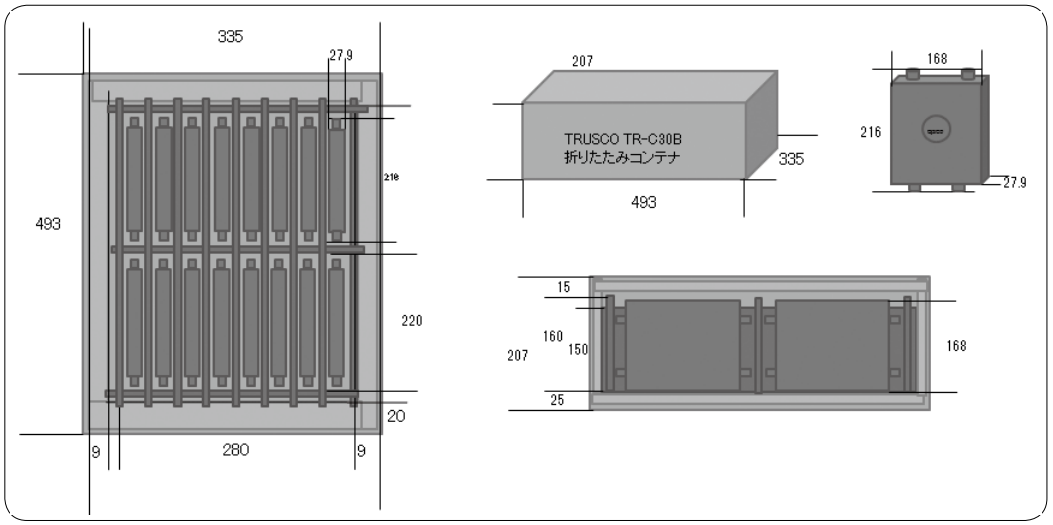
		共有スペースのみ入れる					
	8:00	8:15	8:25	8:40	8:50	9:00	9:10
TeamA	開梱作業	朝礼、及び伝達	開梱作業			IX設置し、メディコンと接続	
TeamB	開梱作業	朝礼、及び伝達	開梱作業			IPSecが通ったら、無線周りに集中	
TeamC	開梱作業	朝礼、及び伝達	SW2の設定変更(AP15接続ポート)	SW2を設置し電源を入れる。SW2とAP15を接続する	ケーブル#1-3を接続し、ホールのドアの前まで引く	ホールに入れるようになったらAP9を本設置する	AP3を設置。SW4からケーブル#3-4を引き、AP3に接続

▼図2　ネットワーク設営の作業時間比率



▼写真1　完成した無線AP収納箱

▼図3　無線AP収納箱の設計図



とも数が多い機材は無線AP（アクセスポイント）です。1カンファレンスあたり、予備機材も含めると約30台の無線APを会場に持ち込むため、30回の開梱作業と梱包作業が発生します。とくに無線APは精密機器となりますので、これでもかというくらいに緩衝材に包まれており、開梱／梱包作業にかなりの時間と労力を消費します。これを改善するには、複数の無線APが収納可能で、1台1台を緩衝材に包まなくても衝撃から保護され、蓋を閉めるだけで発送が可能な無線AP収納箱が必要となります。残念ながらそのような都合の良い製品は世の中になかったので、自分たちで設計して作ることにしました（**図3**）。

作成には、折りたたみコンテナ、プラスチックダンボール、緩衝材のクッションを用い、すべて手作業で作りました。上下方向の緩衝にはクッションを、水平方向の緩衝にはクッションと仕切りのプラスチックダンボールそのものが緩衝材となるしくみです（**写真1**）。実際に使用してみたところ、20分ほどかかっていた開梱作業が1分程度で終えられるほどの効果を発揮しました。また、1箱あたりの無線AP収容数は18台と台数が決まっているため、一目で機材の数を確認できます。最近では会場ネットワーク設計の議論を行う際、「無線APは何箱必要かな？」という感じで浸透しつつあります。これにより、開梱／梱包作業にかかる時間の短縮

に一歩踏み出すことができました。

トラブル対応

会場ネットワークの設営時に発生するトラブルは本当に困ります。ただでさえ配線や機器設置に追われている状況なのに、さらにトラブルシューティングまで加わると心が折れてしまいそうになります。しかし、大半のトラブルは事前の準備不足に起因するもので、準備を十分に行っておけば回避できるものばかりです。そうとわかっていても、事前準備を十分に行うことにもまた困難が伴います。会場ネットワークの事前構築やテストなどの準備は1週間ほどの時間を必要としますが、CONBUに所属するメンバーは会社や勤務地、そして勤務時間もバラバラなのです(**図4**)。

このため、事前構築やテストに何日も同じ場所へ集合するのはとても難しく、有休を使ってまで事前準備をしたり、準備不足のまま本番に突入してしまうことも珍しくありませんでした。

そこで考えたのが「オンライン事前構築」です。

会場ネットワークに必要な機材をインターネット回線がある場所へ集め、物理的な配線とリモートログインのみできるような最低限の設定だけを済ませ、本格的な設定やテストはインターネット越しに行うという方法です。リモート設定ですので、フィルタやインターフェースの設定を間違えないよう細心の注意を払う必要がありますが、空いた時間にどこからでも作業を進められるのはとてもすばらしいことです。結果的に設定やテストにかける時間を多く確保できるので、事前準備を満足にできるようになりました。

このようにとても地味ではありますが、CONBUはひとつひとつの小さな課題を解決することで、よりコストパフォーマンスの高い会場ネットワーク構築手法の確立を目指しています。

来場者との接点を増やすために

CONBUの理念として、「ネットワークの提供を通してコミュニティを超えた人の交流を大切にする」が挙げられます。これは、カンファレンスの運営者や来場者と交流することで、他業界の情報を得たり、人とのつながりを増やすことで視野を広げたいという思いからきています。しかし現状は、カンファレンスの運営スタッフさんには仲良くしていただいているものの、来場者との交流はあまりできていません。これ

▼図4　CONBUメンバの分散っぷり(地図データ©Google)

はCONBUが会場のインフラ担当に納まっているためで、インターネットという土管しか提供していないのが原因だと捉えています。この状況を打開するため、CONBUから何らかのコンテンツを発信することにより、もっと来場者のみなさんと交流することができないかと考えています。

CONBUが提供できるコンテンツの1つとして、会場ネットワークの情報をAPIにより提供する計画があります。具体的には次の情報提供を考えています。

- 装置固有ID(MACアドレスは特定できない形に加工)
- 装置固有IDの電波強度
- APごと、またはスイッチごとのトラヒックレート
- APごと、または全体の端末接続数
- アクセス先(ドメインレベル)

これらを提供することで、次に挙げる例1、2のようなコンテンツの提供が可能となるかもしれません。

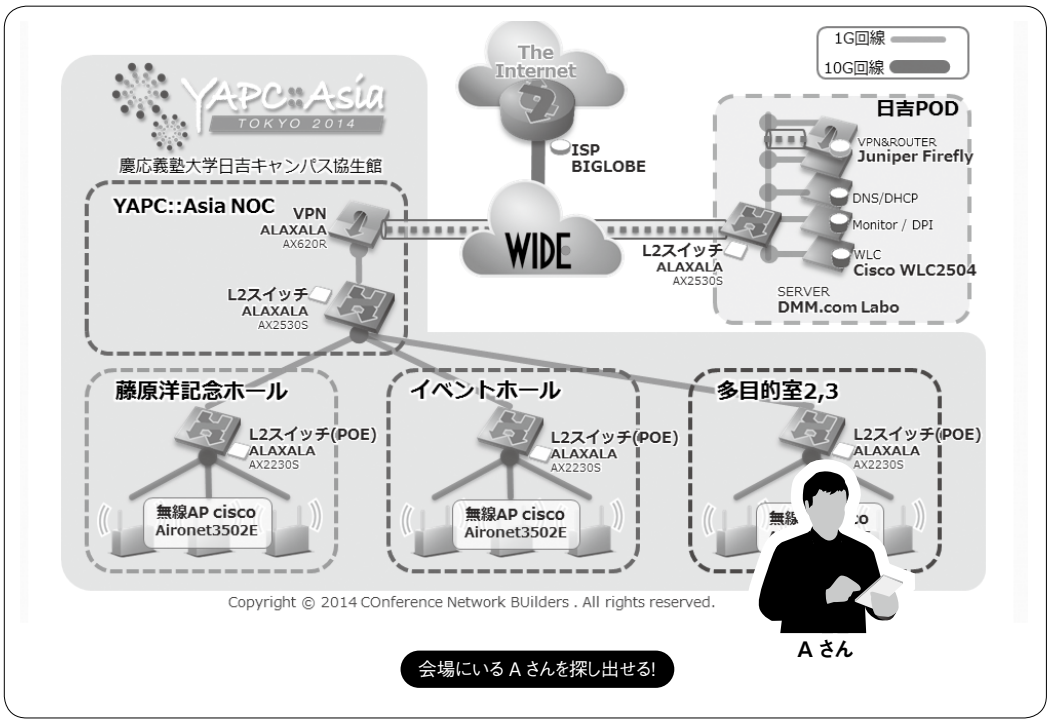
例1:あの人は今どこ?

会場地図と無線APの位置をマッピングし、どの端末がどの無線APに接続しているかをリアルタイムに取得することで、誰がどこにいるかを把握できます(**図5**)。もちろんプライバシーに配慮し、位置情報の発信希望者のみに適用できるようOAuthなどと組み合わせます。これにより目的の人がどこにいるかを知ることができます。

例2:お友達かも?

複数の発表会場にわたるマルチトラック方式のカンファレンスに適用できるコンテンツです。前述の「あの人は今どこ?」と同じように会場の地図と無線APの位置をマッピングし、来場者

▼図5 「あの人は今どこ?」イメージ図



の端末がどの無線APに接続しているかをリアルタイムで取得します。これにより時刻ごとに来場者がどの部屋にいるかを把握できるので、聴講している発表プログラムを割り出すことができます。ここから同じプログラムを聞いている人のクラスタ化を行い、「一定回数同じプログラムを聞いている≒同じ興味を持っている」と捉え、興味が似ている来場者同士をあとから提示することで、懇親会などで盛り上がってもらうというものです(**図6**)。

◆ ◆ ◆

これらは構想や試作段階です。ネットワーク情報を取得できるAPIを提供することで、例1、2のようなコンテンツの作成が可能だと考えています。また本誌を読んでいるみなさんは、我々が思いつかないすばらしいアイデアをお持ちだと思っています。もし、みなさんのお知恵を拝借できるなら「こんな情報をAPIで取得できると、こんなコンテンツが作れるよ！」ということを教えていただけると助かります。

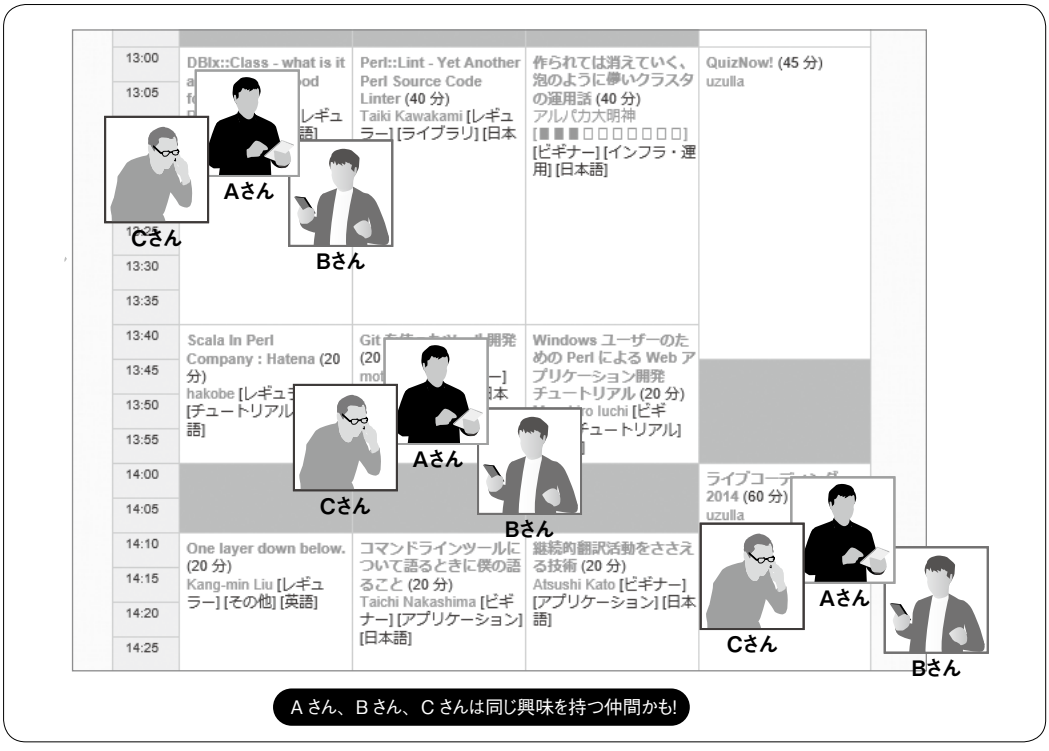
まとめ

CONBUは2014年の結成より1年が経過しようとしています。この間、さまざまな団体様、企業様、そしてCONBUに携わってみたいと思ってくれた人々から多大なご支援、ご理解をいただき今日まで歩んでくることができました。この場をお借りして感謝いたします。今後もCONBUは、「ネットワークの提供を通してコミュニティを超えた人の交流を大切にする」を軸に、カンファレンスネットワークの構築を継続していきます。SD

新しいチームメンバの募集について

CONBUの一員、プロジェクトチームメンバーとなって、会場ネットワークを構築してみたい熱い方を募集しています(会場ネットワークを作ってみたい若手の方、現場を離れているものの久しぶりに手を動かしてみたい方……)。詳しくはhttp://conbu.net/をご覧ください。

▼図6 「お友達かも?」イメージ図



Column CONBUの会場ネットワークセキュリティ

本コラムでは、CONBUのネットワークセキュリティに対する取り組みの一部を紹介させていただきます。CONBUはイベントに参加する来場者が安全な会場ネットワークを利用できるようにさまざまな取り組みをしています。一般的に、来場者の誰もが接続できるオープンなネットワークには、次のようなセキュリティリスクが存在します。

- **SSIDを模倣した、偽の無線アクセスポイントによる通信の盗聴**
- **弱い暗号化方式の脆弱性を狙った攻撃**
- **中間者攻撃による通信内容の改ざん**
- **マルウェア感染した端末からの感染拡大を狙った通信**

基本的にCONBUが作る会場ネットワークは、セキュリティ上の観点から利用者同士の通信を制限しています。しかし、諸事情によりその制限を解除することもあるため、万が一ウィルスやマルウェアなどに感染した端末が会場ネットワークへ接続されてしまうと、利用者全体に影響が出る可能性があります。せっかく勉強やディスカッションをしに来たのに、会場ネットワークに接続してウィルスやマルウェアに感染してしまっては有意義な時間が台無しになってしまいます。

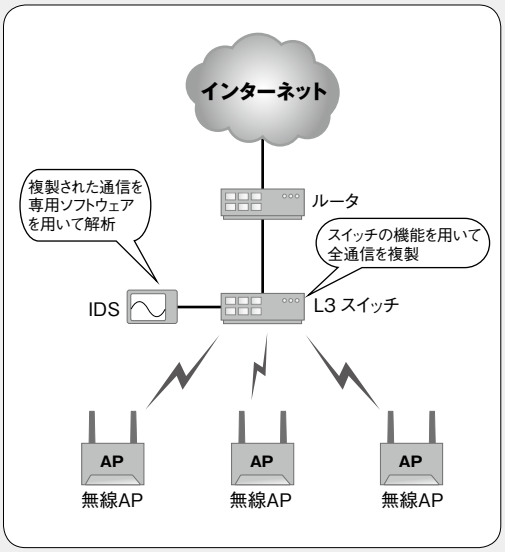
CONBUはこの問題への取り組みとして、カンファレンス開催中は会場ネットワークのすべての通信を自動的に解析し、異常な通信や不審な通信が発生した場合はアラートとして検知するIDS(Intrusion Detection System：侵入検知システム)を導入しています。このアラートを契機にCONBUのセキュリティ担当者が手動でその内容を確認し、問題がある場合は会場内にアナウンスしたり、最終手段として一時的にネットワークから隔離したりするなどの対応をします。

図7はCONBUが用いるIDSの構成です。前記のとおり、IDSの解析対象は会場ネットワークの全通信となります。つまり、全通信が集中する回線に何らかの方法で通信内容を複製して、IDSに送り込む必要があります。CONBUでは、全通信が集中するアップリンク回線にネットワークTAP[注1]や光スプリッタを接続したり、コアスイッチがポートミラーリング機能[注2]をサポートしている場合はこれを用いて全通信を複製することでIDSによる解析を実現しています。

肝心なIDSの種別やカスタマイズ、そしてオリジナルシグネチャの内容を紹介したいところですが、実はこれを明かしてしまうと手の内を晒すこととなり、セキュリティ上の脅威となる可能性もあるので詳細はヒミツです。

このようにCONBUが作る会場ネットワークでは、万が一ウィルスやマルウェアに感染した端末が会場ネットワークに接続されても、その影響範囲を最小限に留める努力をしています。

▼図7　IDSの構成



注1) ネットワークTAPとは、信号レベルで通信を複製できる特別な機器です。おもに通信内容の解析やトラブルシューティングなどに用いられます。

注2) スイッチの機種によっては、性能や仕様の制限によりすべての通信をミラーリングできない場合があります。マニュアルや仕様だけではなく、事前に確認することをお勧めします。

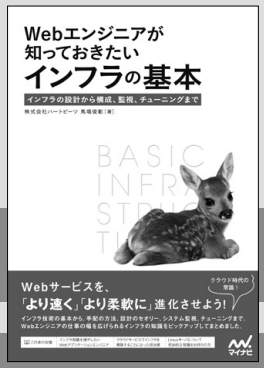
BOOK no.1 詳解Swift

荻原 剛志【著】
B5変形判、408ページ／価格＝2,980円＋税／発行＝SBクリエイティブ
ISBN＝978-4-7973-8049-1

Objective-C本の著者として知られる荻原氏によるSwiftの解説書。C/Objective-Cプログラミングの経験がある人を読者として想定しており、Swiftの言語仕様を網羅的に記している。言語仕様だけではなく、Swiftの中心的な機能である「クロージャ」や「ジェネリック」については、概念・記法・使い方の順に丁寧に解説されており、Swiftのプログラムを「Swiftらしく書く」ための豊富な知識が紹介されている。

実際のアプリ開発のためのTipsとしては、iOSおよびMac OS XのAPIを利用するときの注意点（データの互換性など）、Xcodeにおいて Objective-CとSwiftのプログラムを混在させて使う方法を説明している。

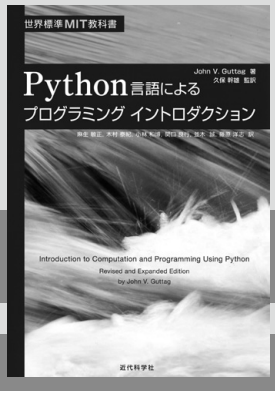
BOOK no.2 Webエンジニアが知っておきたいインフラの基本
インフラの設計から構成、監視、チューニングまで



馬場 俊彰【著】
A5判、312ページ／価格＝2,680円＋税／発行＝マイナビ
ISBN＝978-4-8399-5355-3

インターネットのしくみやサーバの役割／構成といったWebシステムの基礎知識については、対象読者にとって「これだけは」というものに絞り込んであるため全体像が短時間でつかめる。一方で物足りなさを感じるかもしれないが、Webエンジニアがインフラを「うまく使う」ために必要なポイントを実践的に解説している点が魅力とも言える。インフラを監視し、得られた情報からボトルネックを探しだし、パフォーマンスチューニングでそれを改善する。著者の豊富な経験に基づいて選ぶ有用なツールやLinuxコマンドを使った解説により、インフラを活用するための基本が学べる。アプリとインフラの双方のエンジニアにとって待望の一冊だ。

BOOK no.3 世界標準 MIT 教科書
Python言語によるプログラミングイントロダクション

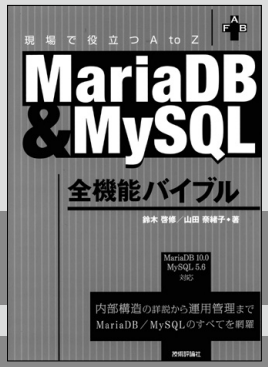


John V. Guttag【著】／久保 幹雄【監訳】／麻生 敏正ほか【訳】
B5判、328ページ／価格＝3,800円＋税／発行＝近代科学社
ISBN＝978-4-7649-0469-9

本書は、マサチューセッツ工科大学の計算機科学の講義内容をベースに書かれたもの。プログラミングで問題を解決するための方法論を解説する。書名に「Python言語による」とあるとおり、プログラミングの手法やアルゴリズムの考え方を紹介するための手段としてPythonを使っている。ただし、Pythonの文法説明は最低限で、主な内容は計算効率の良いプログラムを書くための理論を説明すること。そのために、具体的なコード、数式、グラフが多数用いられる。

本書はプログラミング入門者向けに書かれたものだが、少々難解かもしれない。むしろ、「処理の速いプログラムを書くのに苦労している」という経験者にこそ、読んでもらいたい。

BOOK no.4 MariaDB&MySQL全機能バイブル



鈴木 啓修、山田 奈緒子【著】
A5判、576ページ／価格＝3,500円＋税／発行＝技術評論社
ISBN＝978-4-7741-7020-6

MariaDBとMySQLは現時点では、SQL文においてはほぼ違いがない。だが、「Thread pool」が標準で利用できるか否かなど、機能については少々の違いがある。本書では、冒頭でそういったMariaDBとMySQLの違いや内部構造について説明し、中盤以降では両者がサポートするSQL文・データ型・演算子・関数を網羅的に解説している。DBサーバの運用管理の章では各種機能の設定を、膨大なシステム変数を掲載しながら、参照しやすいようにまとめている。

対応しているバージョンはMariaDB 10.0/5.5、MySQL 5.6/5.5である。本書によると、開発の事情からMariaDB 10.0以降はMySQLとの内部構造の違いが大きくなってくるとのことだ。

第２特集

今年こそ並列分散処理を極めたい

いまからでも遅くない！ Hadoop超^{2}入門

Javaで書かれた検索エンジンであるApache Luceneをベースに誕生したHadoop（ハドゥープ）は、並列分散処理を可能にするOSSです。すでに利用されている方も多くなってきましたが、このしくみをちゃんと理解している方、使いこなしている方はまだまだ限定的です。本特集の第1章では、Hadoopがどのようなソフトウェアで、どう使えるのかを改めて紹介します。そして第2章では、Hadoopのインストールを解説します。さらにHadoopで分散処理されたデータに対してSQL的なクエリが可能なApache Hiveと、従来のMapReduceの代わりにTezと呼ばれるフレームワークを組み合わせ、実際に分散処理を体験していただきます。

Hadoopの基本的なアイデアと構成

Writer 濱野 賢一朗(はまの けんいちろう)　日本Hadoopユーザー会

Hadoop(ハドゥープ)という名前はあちこちで見かけるようになりました。しかし、Hadoopをちゃんと理解できていると自信を持って言える方、使いこなしている方はまだ限定的なようです。本特集では、Hadoopがどのようなソフトウェアで、どう使えるのかを紹介します。

大容量データを高スループットに読み出すしくみ

Hadoopを一言で表現するならば「並列分散処理を実現するミドルウェア」です。よく見かける説明は「ビッグデータ活用を実現する……」といった切り口になっていますが、最初に理解しておきたいのは並列分散処理のためのソフトウェアである点です。

Hadoopでいう並列分散処理は、複数のサーバをずらっと並べて、それを1つのコンピュータのように見立てて使う技術ということです。ハイパーバイザのような仮想マシン技術が、1つのコンピュータを仮想的に複数のコンピュータに分割して利用するために使われていますが、その逆と言えます。複数のサーバを束ねた仮想的なコンピュータで、データを蓄積したり、そのデータの処理を実現します。もちろん、実際には個々のコンピュータでデータの蓄積や処理を行いますから、内部的には「並列に」「分散されて」いるわけです。

並列分散処理の技術にもさまざまなものがありますが、Hadoopでは大容量(もしくは大件数)のデータを蓄積・処理することにフォーカスしています。大容量のデータを処理したいときに、ネックになるのがI/O(スループット)性能です。計算能力がある程度高くても、処理対象のデータを読み出すのが課題になります。

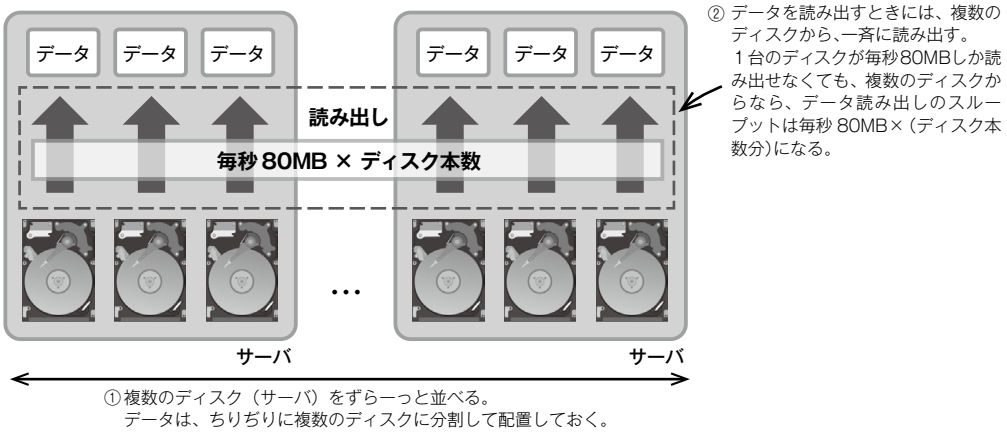
一般的なSATAのハードディスクの場合、平均すれば、よくて毎秒80MB程度しか読み込めません。SSDでも毎秒200MBとか300MB程度です。現在市販されるディスクには容量が8TBのものがありますが、先頭から末尾まで読み込むには、8TB(＝約8,000 × 1,000MB)÷毎秒80MB＝約10万秒＝約27.8時間かかるということです。1台のディスクのデータを読み出すのに1日では終わらないのです。テラバイト級やペタバイト級のデータを扱うには、このデータの読み込みをいかに高スループットで実現するかが課題になります。これを並列分散処理を使って解決しているのがHadoopです。

アイデアは、シンプルです。1台のディスクで80MBしか読み込めないのであれば、複数のディスクをずらっと並べ、そこにデータを分割して配置しておくのです。そのうえで、データの読み込みは複数のディスクから同時に読み込みます(**図1**)。読み出したいデータの、ある部分はあるディスクから、ある部分は違うディスクからと、データの部分ごとに複数のディスクから一斉に読み込むわけです。そうすると、たとえばディスクが1,000台あれば、毎秒80MB × 1,000台＝毎秒80GBのデータを読み込むことができます。8TBのデータも8TB(＝約8,000GB)÷毎秒80GB＝約100秒で読み込むことができます。1台のサーバに搭載できるディスクの台数には限界がある(たとえ搭載できてもバス幅が足りなくなる)ため、実際には、複数のサーバを用意し、各サーバに数台ずつディスクがつながっている構成になります。

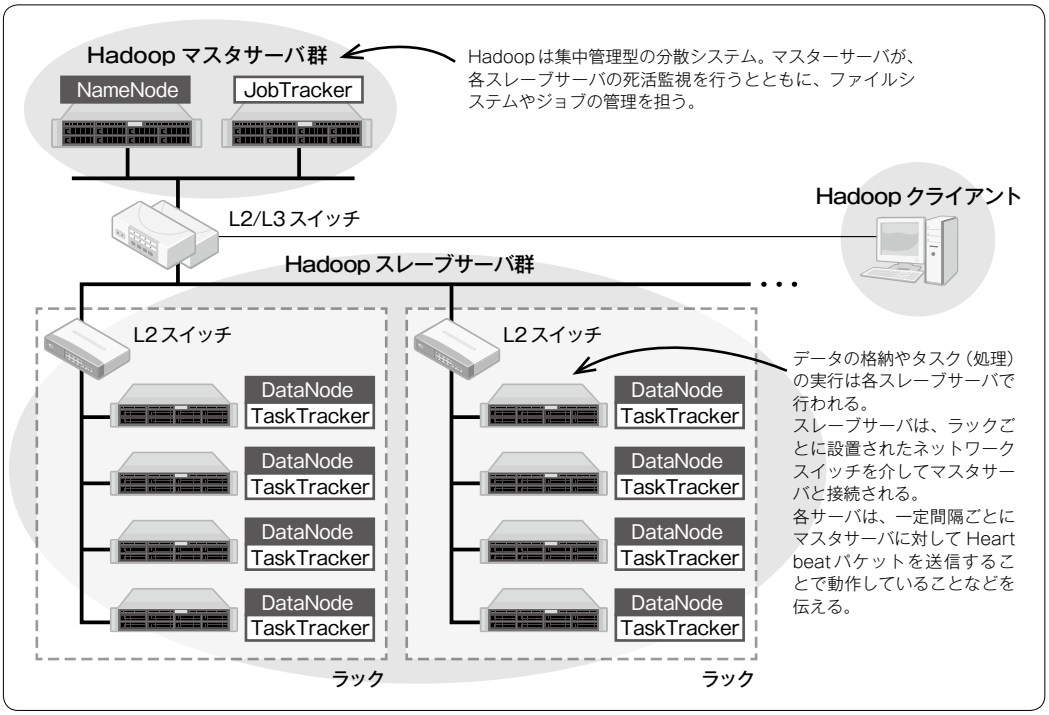
Hadoopの構成

Hadoopを構成するコンポーネントは、バージョンによって異なりますが、まずはバージョ

▼図1　複数のディスク・サーバから同時にデータを読み出すことで、高スループットなデータ読み込みを実現する



▼図2　Hadoopクラスタのよくある物理構成



ン1を前提に紹介します。バージョン2（YARN）の話はのちほど紹介します。

Hadoopは大きく2つのコンポーネントで構成されています。分散ファイルシステムHDFS（Hadoop Distributed File System）と並列分散処理フレームワークMapReduce Frameworkです。前述のアイデアを実現するのがHDFSです。

物理構成は**図2**のようになっています。集中管理型の分散システムになっています。マスタサーバ群が全体を管理して、実際のデータ格納やデータ処理はスレーブサーバ群が担当します。Hadoopを構成するマシン群は総称してHadoopクラスタと呼ばれます。

HDFSのマスタサーバをNameNode、スレー

ブサーバをDataNodeといいます。DataNodeはデータの格納を担当します。NameNodeは、各データのDataNodeへの配置状況、ファイルのメタ情報(アクセス権など)を管理したり、DataNodeの死活を把握したりしています。

MapReduce Frameworkのマスタサーバは JobTracker、スレーブサーバはTaskTrackerといいます。のちほど紹介しますが、Hadoopの世界では、処理をジョブという単位で管理します。ジョブは複数のサーバで並列分散処理されます。この分割された単位をタスクといいます。各TaskTrackerではタスクの実行が行われ、JobTrackerはジョブの実行状況やTaskTrackerの死活を管理しています。

並列分散処理を実現するMapReduceのアイデア

複数のディスクから並列にどーんと(＝高スループットで)データを読み込めたとしても、このデータをネットワークなどを介して別の処理システムに移動しようとすると、せっかく実現した高いスループットが失われてしまいます。データを転送するネットワークやチャネルの性能は、そこまで高くない(もしくは非常に高価である)ためです。

スループットを維持したまま、データを処理するには、できるだけ、データが配置されていたサーバ内でそのデータを処理するのが効率的です(**図3**)。そうすればディスクが読み出したデータをネットワークで転送する必要がありません。

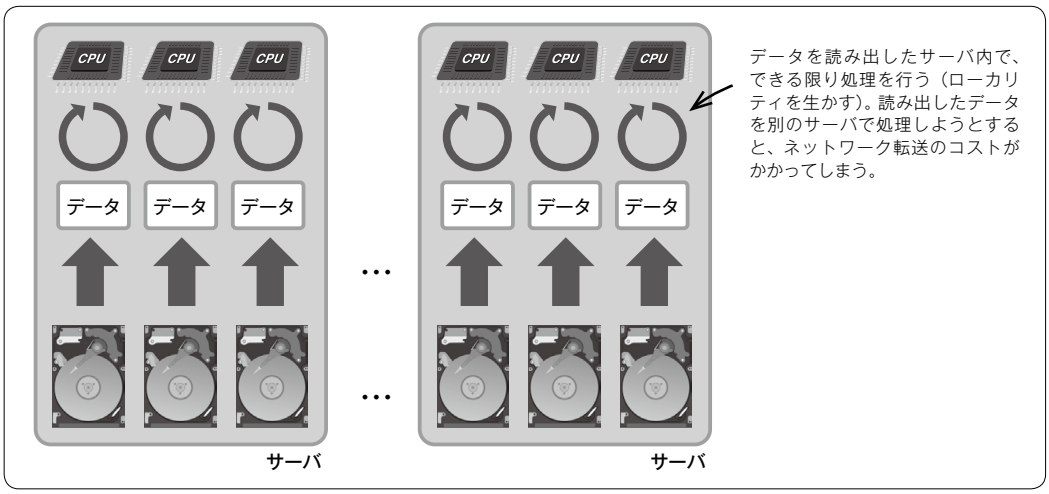
そのためには、データ処理を複数のサーバで並列に分散して実行できるしくみが必要です。これを実現するのがMapReduceです。MapReduceはアルゴリズムの名前で、MapReduceを実現するミドルウェアがMapReduce Frameworkですが、両方とも単にMapReduceと呼ばれることが多いです。ちょっとややこしいですね。

MapReduceのアイデアは、複数人で作業するときに自然にやっている分担のしくみを、汎化(はんか)してソフトウェアの世界に持ち込んだものです。たとえば、選挙の開票作業を思い浮かべてみましょう。投票が終わり、投票用紙が大量に集められてきたあとに、記入された名前(候補者)ごとに枚数を数えるという作業です。

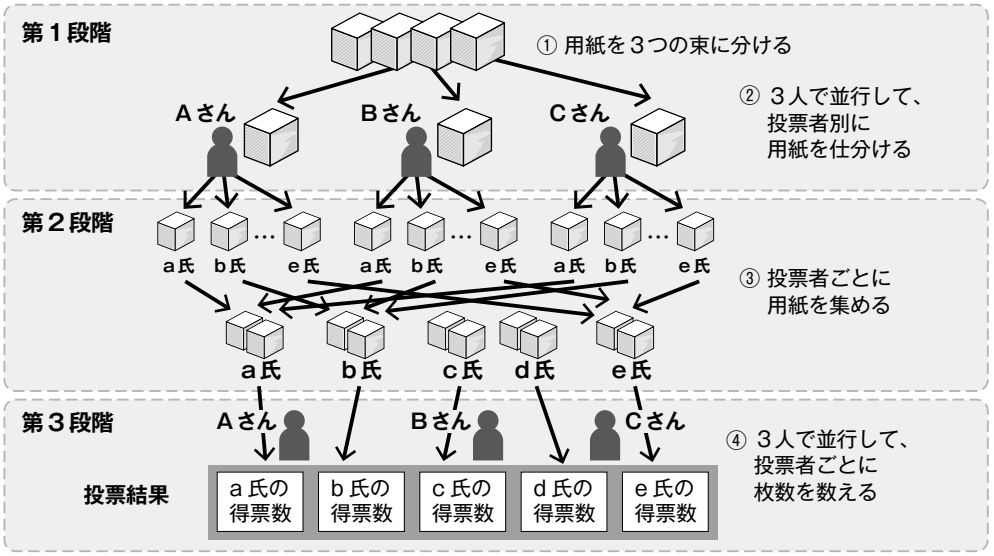
たとえば、3人でこの作業をする場合、次のようにするのではないでしょうか。まずは、投票用紙をおおよそ3つの束に分割します。そのあと、それぞれの担当者は、記入された候補者ごとに投票用紙を仕分けします。候補者がa氏、b氏、……、e氏の5名の場合、5つの投票用紙の束ができあがるわけです(**図4：第1段階**)。

次に、3名が仕分けした5つずつの束(全15つの束)を候補者ごとにまとめていきます。こ

▼図3　読み出したデータをできるだけそのサーバ内で処理する



▼図4　開票作業における作業分担



れにより、a氏、b氏、……、e氏と書かれた投票用紙がそれぞれ1つの束(全5つの束)にまとまります(**図4：第2段階**)。

最後に、投票者ごと(束ごと)の枚数を数えます。たとえば、ある人はa氏とb氏の分を、ある人はc氏の分を、ある人はd氏とe氏の分を数えます(**図4：第3段階**)。これにより、候補者ごとの投票数が数えられます。

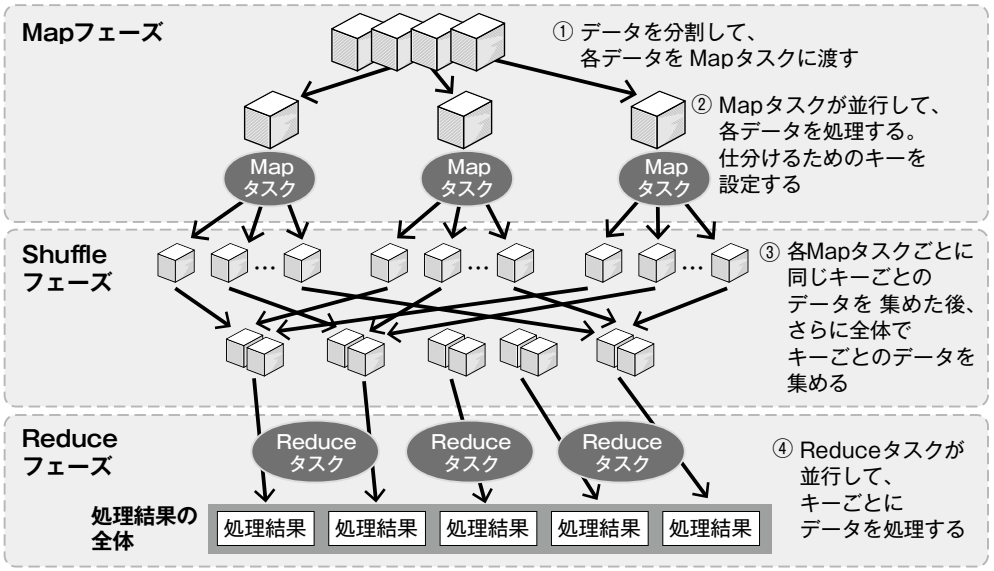
このような分担は特別なことではなく、普段の生活の中でも自然にやっていることではないでしょうか。重要なことは、1名でやるよりも3名でやる方が短時間に処理できる点です。なぜならば、第1段階や第3段階での作業は、ほかの人の作業に影響を受けずに(待たされたりすることなく)、各メンバは黙々と進めることができるからです。

さらに重要なことはメンバを増やせば、さらに処理速度が上がることです。第1段階では、N人でやれば、1人でやる場合に比べてN倍のスピードで処理できます。また、第3段階でも、N人でやれば「おおよそ」N倍のスピードで処理ができます。「おおよそ」というのは、特定の候補者に投票用紙が偏っている場合には、その枚数を数える作業が極端に遅くなってしまう場合があるからです。そのようなケースでは、もう少し工夫が必要です。枚数が多い候補者の投票用紙をさらに複数の束に分けて、複数人でそれぞれを数え、最後にその枚数を合算するといった工夫が必要です。各メンバに割り当てられる作業量(投票者ごとの枚数)に大きな偏りができることだけ気をつけていれば、おおよそN倍のスピードが実現できるということです。

このような、複数人での作業分担を汎化したものがMapReduceです。MapReduceでは、第1段階のことをMapフェーズ、第2段階のことをShuffleフェーズ、第3段階のことをReduceフェーズといいます。Mapフェーズでは、データを分類・仕分けします。Shuffleフェーズでは、同じ分類とされたデータを1ヵ所に集めます。Reduceフェーズでは、分類ごとに集められたデータに対して処理を行います(**図5**)。

開票作業の例はかなりシンプルなものですが、MapフェーズとReduceフェーズでの処理を工夫して定義することで、複雑な業務処理でもかなりの割合がMapReduceで実現できることがわかっています。

▼図5　MapReduceによる並列分散処理



Hadoopでは、HDFSから読み出したデータに対して、MapReduceを用いて並列分散処理することで、高スループットなデータ処理を実現します。1台のサーバでは難しかったり、時間がかかっていた処理を現実的なものとします。

HDFSやMapReduceでは、構成するサーバ台数を増やすと、格納できるデータ容量や並列に処理できる計算性能が向上できるという高いスケーラビリティが得られます。これはスゴイことです。

HDFSとMapReduceの特徴

Hadoopが実現する基本的なアイデアを紹介しましたが、より具体的な工夫や特徴のうち、ぜひ知っておくとよいトピックに限定して説明します。

大容量のデータをブロックに分割して配置する（HDFS）

HDFSは、ファイルの中身がどのような形式になっているかは関知しません。画像や動画、任意のバイナリ形式でも格納できます。MapReduce Frameworkがデータを読み込むときに、解釈（意味づけ）することになっています。従来のRDBMSなどはデータ格納時にスキーマを適用する「Schema on Load」と分類されますが、Hadoopではデータ読み込み時にスキーマを適用する「Schema on Read」といえます。これは、生データをそのまま格納しておくのが容易だったり、処理ごとにデータの意味づけを変更するなどの柔軟性の確保に役立ちます。この特徴をもって、Hadoopは「非構造データ、準構造データの格納、処理に向いている」と説明されることもあります。

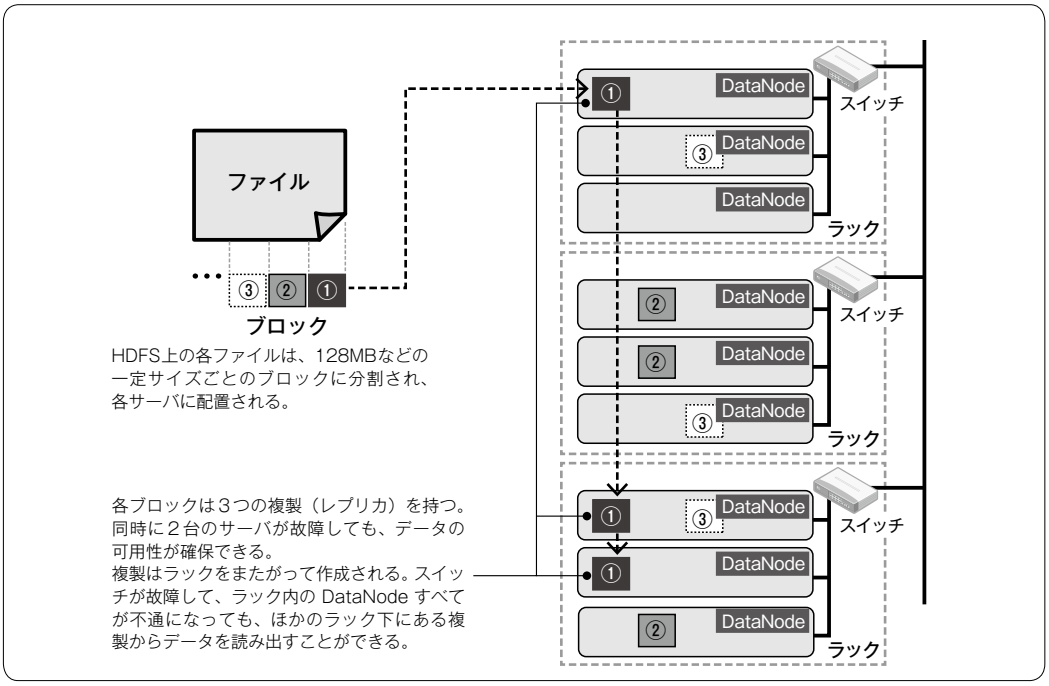
一般的には、HDFSに格納されるデータは、1件が1行で表現されることが多いです。CSV形式のようなカンマ区切りだったり、タブ区切りになったものが一般的です。

HDFSへのデータのロードは、

```
$ hdfs dfs -put /tmp/dataset1 /foo/bar
```

などとして実現できます。/tmp/dataset1はローカルファイルシステムのパスで、/foo/barはHDFS上のパスになっています。データをロー

▼図6　HDFS：ブロックに分割されたデータが3つの複製で配置される



ドすることで、複数のサーバに分割して配置されます。HDFSでは、分割された単位を「ブロック」と呼んでいます。通常、ブロックは64MBや128MBといったサイズになります。ファイルの先頭128MB分があるサーバに、次の128MB分が別のサーバに、次の128MB分が……といった具合に配置されます(**図6**)。これにより、複数のサーバで一斉に読み込めるようになるのです。

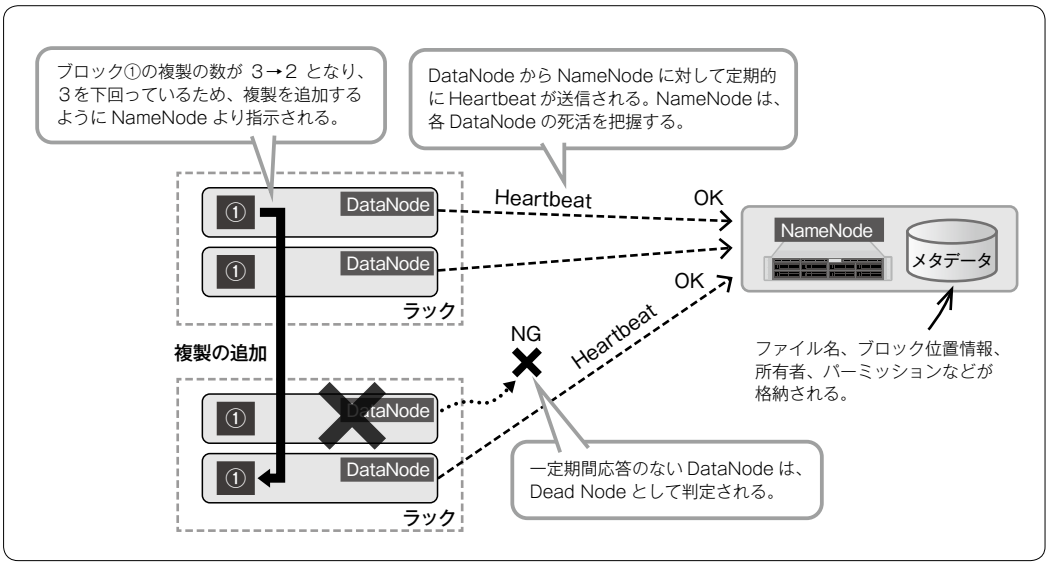
数KBや数MBといった小さいサイズのファイルをHDFSに配置しても、分割されません。あくまでHDFSは数100MB以上の大きいサイズのデータの格納・読み込みに適したファイルシステムといえます。HDFSに配置されるファイルの大半が小容量のデータになるケースでは、複数のファイルを1つにまとめるなどの工夫が必要です。

各ブロックの複製により可用性を高める(HDFS)

分散システムを構成、運用するうえで気をつけるべき点の1つに「全体の数%程度のノードが故障しても可用性が失われないように」工夫するということが挙げられます。多数のサーバを運用していると、どれか1台以上が故障している時間は長くなります。すべてのサーバが正常に動作していないとサービスが継続できないしくみだと、サービスを安定的に提供できません。そこで全体の数%のサーバが故障していても、全体としてはサービスが提供できるような工夫が必要です。

HDFSの標準的な構成では、各ブロックを3台の異なるサーバに配置します。Hadoopクラスタ全体では、同じブロックのデータを3つ持っていることになります。これにより、ブロックが配置されているサーバが2台故障していても、データにアクセスできます。NameNodeは、DataNodeの死活を把握するとともに、各ブロックの複製(レプリカ)が3個あることを保証するように調整します(**図7**)。たとえば、あるDataNodeが故障して、あるブロックの複製(レプリカ)数が3を下回った場合、そのブロックを別のサーバにコピーして、複製数を3に回復させます。

▼図7　HDFS：NameNodeの役割、レプリカの維持



複製の配置先にも工夫があります。Hadoopのスレーブサーバは、**図6**のように、複数のラックにまたがって配置され、各ラックごとのネットワークスイッチに接続されます。もしブロックの複製がすべて同一のラック内に配置された場合、スイッチが故障すると、すべての複製にアクセスできなくなってしまいます。そこで、複製はラックをまたがるように配置され、スイッチが故障してもデータへのアクセスができるようになっています。

ジョブはMapタスクとReduceタスクとして実行される（MapReduce）

MapReduceは、HDFS上のデータを並列分散処理できるフレームワークです。ユーザは、処理をジョブという単位で指示します。アプリケーション開発の観点からみると、MapReduceジョブは、Map関数とReduce関数の2つを定義するものです。ジョブを実行すると、MapReduce Frameworkにより複数のタスクに分割され、各タスクはTaskTrackerで実行されます。たとえば、1つのジョブは、20個のMapタスクと15個のReduceタスクに分割されたりします。開票作業の例でMapReduceのイメージを紹介しましたが、より一般的に説明すると、次のようになります。

Mapタスクでは、HDFS上のブロックのデータを読み込み、各データ（一般的には1行ごと）から、ユーザが指定したMap関数により、KeyとValueを指定します。Keyは、後続のReduce処理の単位となります。Keyの指定はデータを分類しているイメージです。開票作業の例では、候補者名がKeyに相当します。Valueには、後続のReduceで必要なデータを指定します（開票作業の例でいえば、便宜的にValueを1としているイメージです）。このMapタスクは複数起動され、並列に実行されます。

Mapタスクの数は、原則として、対象データのブロック数になります。各ブロックごとにMapタスクが実行されます。各Mapタスクは、可能な限り、処理対象のブロックが配置されているサーバ上で実行されます。これによりネットワークを介さずにデータを読み込むことができます。HDFSでは、64MBや128MBなどのブロックに分割されて配置されますが、その区切りが必ずしもデータの境界に対応しているとは限りません。1件のデータが複数のブロックにまたがってしまう場合もあります。このようなケースでは、必要な分のデータがネットワークを介して転送されます。

Shuffleフェーズは、MapReduce Frameworkにより自動で実行されます。Map処理で指定したKeyごとに、Valueの値が束ねられます。このとき、Valueがソートされます。Shuffleでは、ネットワークを介してデータが転送されます。処理対象データが大きい場合、Shuffleでのデータ転送がボトルネックになりやすいので、Map処理時にデータ量を絞っておくなどの工夫が必要な場合があります。

各Reduceタスクでは、KeyごとのValueの値を受け取り、新しいKeyとValueを出力します。この処理を指定するのがReduce関数です。開票作業の場合は、Valueの数を単に足し合わせる処理といえます。結果は、HDFS上のファイルとして出力されます。Reduceタスクの数は、ジョブの実行時に指定できます。Keyの値のバリエーション、TaskTrackerのサーバ数が十分にあるのなら、Reduceタスクの数を増やせば、並列度を上げることができます。

開票作業の例をもとに、各処理を表現すると**図8**のようになります。Map処理以降のデータは<Key, Value>の形式で表現しています。

スロットによって同時実行されるタスク数が制限される（MapReduce）

MapタスクやReduceタスクは並列に実行されますが、あまりに多数のタスクが同時に実行されてしまうと、サーバのリソースが不足して、処理全体が円滑に実行できない事態になりかねません。そこで、あらかじめMapスロット、Reduceスロットとよばれる実行枠を設定しておきます（**図9**）。MapタスクはMapスロット数を超えない範囲で同時に実行され、足りない場合は先行するMapタスクが完了するのを待って実行されます。

投機的実行（MapReduce）

HDFSと同様、サーバの故障が直ちにジョブの失敗につながらないように工夫されています。JobTrackerは、各TaskTrackerでのタスクの実行状況を把握しており、タスクが中断してしまった場合、別のサーバで実行されるようになっています。

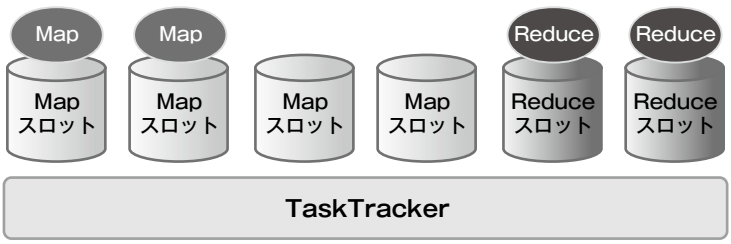
さらに「投機的実行」と呼ばれるしくみも実現されています。これは、同じMapタスクやReduceタスクを複数実行します。空スロットがある場合に、同じタスクが実行されるもので、残り数タスクでジョブが完了する……といった場合に、早く終わったほうを採用するためのものです。

◆◆◆

▼図8　開票作業の場合の各処理

```
投票用紙の元データ
    ↓  Map処理
<a氏,1>, <c氏,1>, <a氏,1> , ……
    ↓  Shuffle
<a氏,[1,1,1, ……]>, <b氏,[1,1, ……]>, ... , <e氏,[1,1, ……]>
    ↓  Reduce処理
<a氏,123456>, <b氏,765432>, …… , <e氏, 97531>
```

▼図9　スロットによって同時実行されるタスク数が制限される



Map、Reduceごとにスロット数を設定。最大並列数はスロット数で制限される。

このほかにもHadoopにはさまざまなアイデアや工夫が実現されています。「処理をどう分割するか」「分割して実行した処理結果をどうまとめるか」「処理中に起きた障害などをどうリカバリするか」といった並列分散処理を実現しようとすると面倒だった部分をミドルウェアとして解決してくれるようになっています。

よくあるHadoopの誤解

基本的なHadoopのアイデアや特徴はご理解いただけたと思います。ここでは、よく聞かれるHadoopに関する誤解のうち、いくつかを紹介します。

Hadoopは高速なデータベース?

高速にデータ処理を実現できるという観点で、リレーショナルデータベースやデータウェア製品などと比べられることが多いため、データベースのプロダクトと誤解されるケースが多いのです。しかし、Hadoopは、ファイル単位でデータを管理しており、1件ごとのデータを管理するしくみはありません。その意味で、データベースとはいえません。あくまで、分散ファイルシステムと並列分散処理フレームワークの組み合わせです。

また、Hadoopは大容量(多件数)のデータを高スループットで処理することに力点を置いています。したがって、少なくとも数秒、長ければ数時間におよぶバッチ処理を実行することになります。リレーショナルデータベースで求められるような少量のデータを数ミリ秒で取得するような低レイテンシな処理には向きません。

Hadoopは検索エンジン?

Hadoopのアイデアは、もともとGoogleの検索システムを支えるものとして登場しました。インターネット上のデータを収集して蓄積し、そのデータから検索インデックスを生成する処理には、従来のミドルウェアやアイデアでは限界があったため、分散ファイルシステムGFS(Google File System)やMapReduceを考案したと言われています。これらのアイデアは2003～2004年に論文として公開されました。このアイデアをもとにオープンソースソフトウェアとして実装されたのがHadoopの始まりです。現在はApache Software Foundationのプロジェクトとして開発が進められています。

Googleの検索システムのアイデアがもとになっているせいか、検索エンジンと誤解されているケースがあります。あるイベントで、クイズで「Hadoopは検索エンジンのソフトである。○か×か」と出題したことがありますが、参加者の5%くらいは○を選んでいました。

もちろん、Hadoopは検索エンジンではありませんが、検索インデックスを生成するためにはよく利用されています。検索対象のデータをHDFSに格納しておき、MapReduceで検索のためのインデックスデータを生成するといった使い方です。たとえば、オープンソースの検索ソフトウェアSolr向けのインデックスの作成はよくある使い方の1つです。

検索のたびに対象データを走査していてはすばやく(低レイテンシで)応答するのが難しいので、前処理として検索インデックスを作成しておくわけですが、このような前処理にHadoopを使うというのは珍しくありません。Hadoopに生データを格納しておき、クレンジング処理や突き合わせなどの前処理を行い、その結果をリレーショナルデータベースやデータウェアハウス製品などに格納するといった使い方です。大量のデータを高スループットでバッチ処理できるHadoopの特性と、低レイテンシに少量の結果を取得できるデータベース系プロダクトのよいところ取りと言えるでしょう。

HDFSには単一障害点がある?

よく競合プロダクトによるHadoopとの比較で「HDFSには単一障害点があるので、可用性が担保できない」という記述があります。

スレーブサーバであるDataNodeやTaskTrackerは、もともと、全体の数%が故障しても、全体の可用性に影響がでないように設計されて

います。マスタサーバであるNameNodeやJobTrackerはHA(高可用性)構成を組むことで単一障害点にならないようにします。昔のHadoopにはHA構成を実現するしくみが内在しておらず、PacemakerなどのHAソフトと組み合わせる必要がありました。その意味で「HDFSに単一障害点があった時代があった」のは事実ですが、現在ではNameNodeやJobTrackerをHA構成にするしくみがあらかじめ備わっています。

Hadoopは何に利用されているか

Hadoopを使いこなしている企業は着実に増えてきていますが、どのようなシーンで活用されているのでしょうか。Hadoopは汎用的なミドルウェアであるため一言で表現するのは難しいですが、大きく2つの領域で利用されていると整理できます。

全件データ処理の実現

Hadoopのデータ読み込みのスループットを最大化する特徴を活かして実現されているのが全件データを対象とした処理です。

従来のデータ分析などでは、すべてのユーザの、すべてのデータを処理しようとすると時間もコストもかかるため、統計的に有意な範囲でサンプリングを行い、大局的な動向などを把握するといったパターンが主流でした。

しかし、Hadoopにより全件データの処理も十分なスループットで、しかも現実的なコストで行えるようになったのです。全件データを処理することで、大局的な特徴だけでなく、たとえばユーザごとの個別の特徴も把握できるようになります。いわゆるロングテールの把握と言えるでしょう。ユーザごとの過去の行動ログを処理することで実現されているサービスには、レコメンデーションやソーシャルサービスにおけるパーソナライズが挙げられるでしょう。

サービスやアプリケーションから取得できる行動ログ、センサーや機器から収集したログデータなどを全件みることによって、同業他社にはないサービスを提供したり、従来より高度なサービスを提供するといった使い方はHadoopの王道と言えます。並列分散処理を使わないと現実的には難しく、いわゆるビッグデータ活用と言える領域です。

データ件数増大への対応

Hadoopは大容量データの処理を得意としていますが、大容量ではなくても「多件数の」データ処理にも向いています。MapReduceによって並列分散処理を実現できるからです。同じ処理でも、対象データ量が増加して、処理時間が延びてしまっているケースがあります。最近では、国内企業が海外でのビジネス／サービスを開始した結果(対象となる人口が一気に増えることで)対象データ件数が急激に増大して、処理時間が延び延びになっている話を耳にします。このようなデータ件数の増大に、並列分散処理の導入が有効になっています。

Hadoop活用シーンの例

より具体的にHadoop活用シーンの例を挙げると、次のような処理に採用されています。

- 過去のアクセス履歴を格納・処理して、ユーザごとの嗜好(特徴量)を抽出する、コンテンツ最適化やレコメンド
- オンラインゲームなどのサービスにおいて、ユーザ行動を分析することによる、解約低減やその効果の評価
- 金融商品の現在価値計算（中間データが肥大化するシミュレーション)、利用者ごとのリスク計算
- N:Nのデータの突き合わせが必要な名寄せ系処理
- PL/SQLなどで多件数の小容量データを繰り返し取得・処理していたケースの高速化（並列処理に）
- タービン、橋梁、自動車、航空機につけたセンサーからのデータを格納・処理すること

による故障検知、利用の効率化

知っておきたいHadoopのトピック

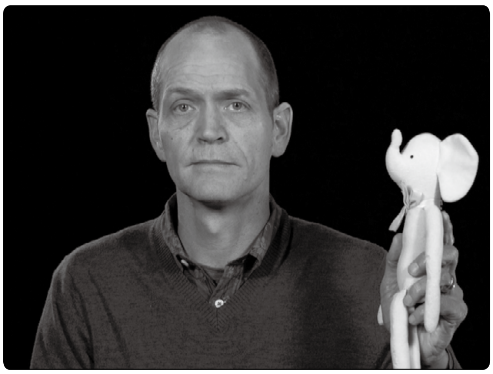
Hadoopという名称

一般的な辞書で「Hadoop」を調べても、その項目はありません。もともとあった意味のある単語ではなく、このソフトウェアのために作られた造語だからです。Hadoopの語源？は、生みの親Doug Cutting氏のお子さんの人形にあるそうです。お子さんが持っていた黄色いゾウさんの人形(**写真1**)のことを「はどぅーぷ」と呼んでいたため、ソフトウェアの名称をHadoopとして、ロゴを黄色いゾウさんにしたそうです(**図10**)。

Hadoopディストリビューション

Hadoopはhadoop.apache.orgからダウンロードしてインストールできます。しかし、現状では、Hadoopディストリビューションと呼ばれるパッケージを使うのが一般的です。

▼写真1　HadoopのぬいぐるみとDoug Cutting氏



▼図10　Hadoopのロゴ(黄色いゾウさん)

Hadoopには、エコシステムと呼ばれる周辺ソフトウェアが多数あります。SQLライクな言語で処理を記述できるHive、データ処理フローを独自言語で記述できるPig、カラム指向のキーバリューストアHBaseなど多数挙げることができます。これらのエコシステムの組み合わせをパッケージ化しているのが、Hadoopディストリビューションです。エコシステム間の依存関係などを意識することなく、容易にインストールして利用できます。

オープンソースソフトウェアだけでなく、商用プロダクトを組み合わせたものも含めれば、Hadoopディストリビューションには、

- CDH(Cloudera's Distribution including Apache Hadoop)
- HDP(Hortonworks Data Platform)
- MapR
- IBM BigInsights
- Pivotal HD

などがあります。また、最近ではFedoraなどのLinuxディストリビューションにもHadoopパッケージが同梱されるようになっています。

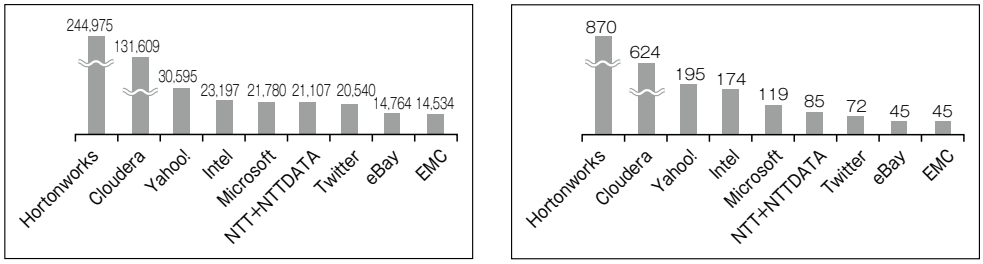
Hadoopは誰が作っているのか

Hadoopの開発はYahoo!のラボから始まっています。現在では、Apache Software Foundationのプロジェクトとして、多くのエンジニアが参加して開発が進められています。

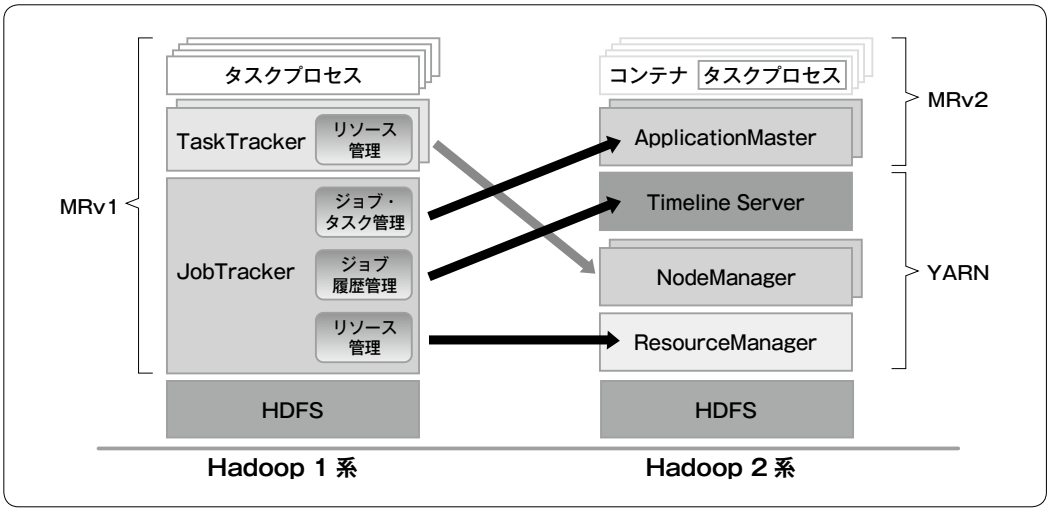
筆者らが2014年にHadoopに行われた機能追加や修正を所属する企業ごとに集計したところ、**図11**のようになりました。Hadoop専業であるHortonworksやClouderaが圧倒的な割合を占めていますが、それ以外からも多数の貢献があります。これまでに約3000名の開発者からのパッチや報告が取り込まれています。

また2014年12月には、NTTの小沢健史氏とNTTデータの鯵坂明氏の2名がHadoopコミッタに就任しており、両社の貢献(2014年)がコード量で第6位、件数では第4位になっている点も注目です。

▼図11　Hadoop開発貢献のランキング（2014年）



▼図12　MapReduceバージョン1とバージョン2の対応



Hadoopの今とこれから

ここまで、Hadoopのバージョン1を中心に説明してきました。現在、Hadoopはアーキテクチャや利用においても大きな変貌（へんぼう）を遂げようとしています。ここからは、Hadoopバージョン2を中心に起きている変化について紹介します。

YARNの導入

MapReduceによる並列分散処理は、利用者にとって大きな威力をもたらしました（**図12**）。研究レベルでは古くからあっても、一部のシステムでしか活用されていなかった並列分散処理がHadoopによって、一気に身近なものとなり浸透を始めました。

一方で、Hadoopを使いこなし始めた利用者にとっては、MapReduceだけだと、うまくはまらない処理も顕在化してきて、より柔軟に並列分散処理を活用したいというニーズが生まれてきています。MapReduceとほかの並列分散処理システムを併用したいのです。

しかし、扱うデータ量が大きいため、Hadoopクラスタとは別に、ほかの並列分散処理システムを用意して、その間でデータを移動するのは非現実的です。したがって、同じデータ、すなわちHDFS上のデータをMapReduceだけでなく、ほかの並列分散処理システムからも利用できるようにしたいというニーズが上がってきました。

そこで開発されたのがYARN（Yet Another

Resource Negotiator)です。従来のMapReduce FrameworkではリソースとジョブをYARNと呼ばれるリソース管理機構として提供するようにしました(**図12**)。これにより、MapReduce以外の並列分散処理システムからYARNのリソース管理機能を利用できるようになり、HDFS/YARN上に複数の並列分散処理システムを動作できるようにしました。

現在、YARN上では、MapReduceバージョン2のほか、DAG型分散処理を実現するApache Tez、インメモリ分散処理を実現するApache Spark、ストリーミング処理を実現するApache Stormなどが動作します(**図13**)。

MapReduceバージョン1では、スロット数によってMapタスクやReduceタスクの同時実行数を制御してきましたが、YARNではコンテナ技術によって仮想コア数やメモリ量をもとにリソース制御を実現します。

Apache Tezへの移行

MapReduceでは、Mapフェーズと Reduceフェーズを直列につなげてジョブを実行します。このしくみだけでも、さまざまな処理を並列分散処理でき、その恩恵を受けることができます。しかし、1つの業務処理を実現しようとすると、MapReduceジョブを多段に実行しないといけない場合が登場します。たとえば、Map→Reduce→Reduceという順序で処理する必要がある場合、MapReduceジョブを2段階に組み合わせる必要が生じます(**図14**)。

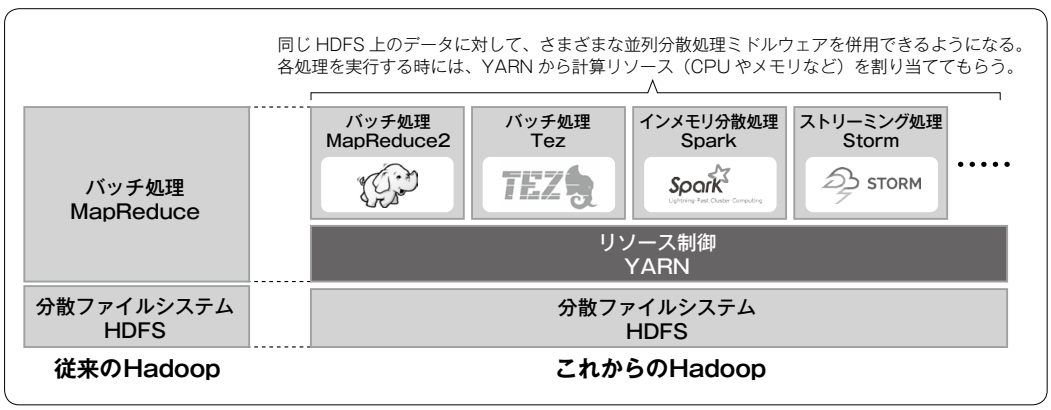
Apache Tezは、MapReduceの後継として最有力の並列分散処理フレームワークです。DAG(非循環有向グラフ)型の処理系で、MapReduceよりもより柔軟なデータ処理パターンを実現できます。MapやReduceの処理パターンも自在な組み合わせで実行できます。そのため、TezではMap→Reduce→Reduceという順序を1つのジョブで処理できます。

同様の処理をMapReduce Frameworkで実行する場合、**図14**のようにMapReduceジョブを2回実行することになります。何もしないMapタスクが起動してしまう、YARNのApplication Masterが2回起動する、といったオーバーヘッドが発生します。TezではApplicationMasterを起動し続けることで、2度目以降のジョブ実行でJVMの起動コストを削減する、という工夫もなされています。ほかにも、Reduceタスク数を動的に制御するなど、さまざまな最適化がなされています。

HiveやPigもTezに対応しています。従来、HiveやPigの処理はMapReduceジョブに展開されて実行されていましたが、現在ではTez上のジョブに展開して実行できます。

Tezの利用にはYARNの利用が不可欠です。まだYARNの導入・移行は始まったばかりですので、Tezの浸透もこれからが本番という状

▼図13　YARNによる並列分散処理ミドルウェアの併用



況です。しかし、アプリケーション視点では、移行障壁も低いため、着実に浸透していくものと思われます。

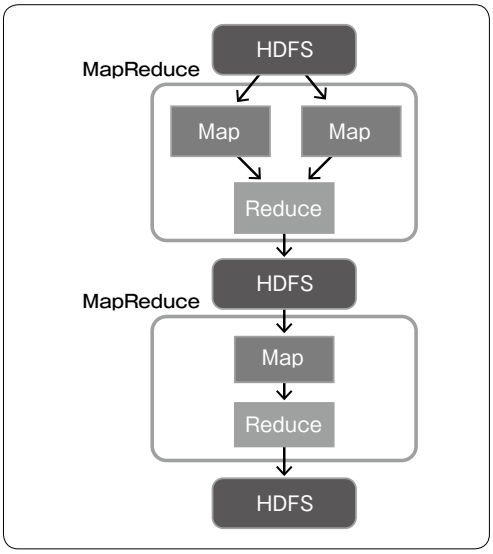
SQL on Hadoop

Hadoop上で実行したい処理をSQLで記述する取り組みは、これまでにもHiveなどで行われてきました。従来のHiveでは，SQLライクなクエリをMapReduceとして実行でき、多くのHadoopユーザに利用されてきました。イベント「Hadoop Conference Japan 2014」の来場者アンケートで行った調査でも、Hadoopユーザの71.3%がHiveを利用していると回答していました。しかし、従来のHiveでは、SQL標準との互換性や性能の面でも課題があるものでした。そのため、JDBCやODBCを利用した接続や各種ツールとの連携には大きな制約がありました。

現在、Hiveはもちろん、それ以外にも処理をSQLで記述する取り組みが進められています。支えるソフトウェアとしては、Apache Tez、Apache Spark、Presto、Cloudera Impala、Apache Drillなどを挙げることができます。これらの取り組みや成果物はSQL on Hadoopと総称されています。SQL標準にこれまで以上に対応し、高スループットと低レイテンシの両面を追求しようとしています。従来のMapReduceにとどまらず、DAG型やMPP型の処理系も活用しようとしています。

SQL on HadoopはHadoop活用の起爆剤になりそうです。これまでのリレーショナルデータベースでは十分な性能が出なかった処理への適用や、BIや可視化ツールからの利用などに活用できそうだからです。強力なツールになりそうですが、一方で注意も必要です。Hadoopはリレーショナルデータベースと違うしくみで実現されている点をよく理解して使いこなす必要があります。同様のトランザクション機能を期待するのは困難ですし、処理特性が大きく違います。とくに、BIツール、可視化ツール、O/Rマッパが自動生成するクエリは、リレーショナルデータベースのオプティマイザの賢さに頼っている部分が多いので注意が必要です。SQL on Hadoopにおけるクエリの実行最適化もしだいに高度化するでしょうが、現状では進化の過程にあるものと理解して、活用することが重要です。並列分散処理の特性に合わせてうまくクエリを記述すれば、強力に処理を行えるでしょうが、向いていないクエリ、たとえば、ノード間で大量のデータをネットワーク転送するような処理を動かすのは避けるべきです。「過度に期待せずに、使える部分から、適宜クエリを見直しながら移行していく」くらいの感覚がよいでしょう。

▼図14　MapReduce Frameworkで2段のジョブになる例



まとめ

Hadoopのベースとなっている技術的なアイデアや構成、活用シーン、動向などについて紹介してきました。Hadoopは、これまで広くは普及・浸透していなかった並列分散処理を身近なものとしてくれた新しいミドルウェアです。ぜひ、深く理解して、うまく使いこなしていただきたいと思います。

次章では、Hadoop 2.6をインストールして、Hiveを通じてMapReduceジョブやTezのジョブによる並列分散処理を実際に体感してみましょう。SD

第2章

Hadoopで並列分散処理を体験！
――Hadoop 2.6＋Tez＋Hiveの実行例

Writer 鯵坂 明(あじさか あきら)日本Hadoopユーザー会(Hadoopコミッタ)／
濱野 賢一朗(はまの けんいちろう)日本Hadoopユーザー会

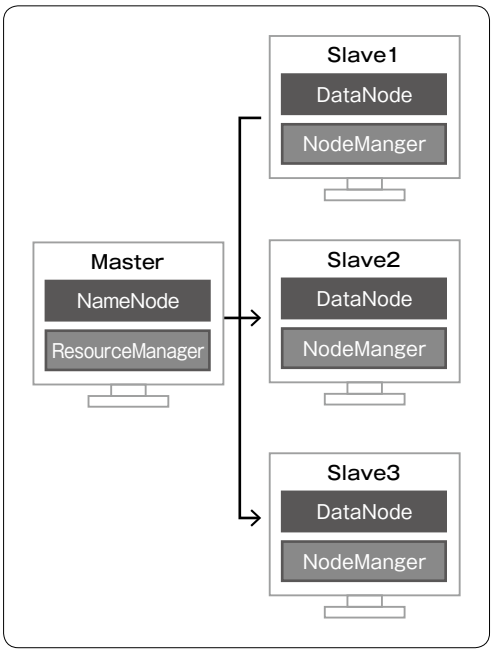
Hadoop 2系では、YARNによってMapReduce以外の分散処理もHadoop上で実行できるようになりました。SQLライクに分散処理を実現するHiveについても、その下回りの処理基盤として、従来のMapReduce以外にTezと呼ばれるフレームワーク上で実行できるようになりました。本記事では、HiveをTez上で動作させる方法(Hive on Tez)について、Hadoopをインストールするところから順番に紹介します。

Linuxの設定

図1のように実行環境として64bit Linuxををインストールした環境を4台用意します。1台はマスタ兼クライアント、3台はスレーブとして利用します。マスタはHA(High Availability：高可用性)構成を組むことが可能ですが、説明を簡単にするため本記事ではHA構成を組まないことにします。

HadoopはWindows OSにも対応していますが、作業手順が簡単なLinuxをお勧めします。Hadoopをインストールする前に少し設定が必要です。以降では、CentOS 6で作業ユーザがec2-userの場合のコマンド例を紹介します。

▼図1　今回の記事における構成図



・SELinuxを無効にする

```
$ sudo setenforce 0
```

・ファイアウォールを無効にする

```
$ sudo service iptables stop
$ sudo service ip6tables stop
$ sudo chkconfig iptables off
$ sudo chkconfig ip6tables off
```

Javaのインストール

Hadoopを動作させるJavaとしては、Oracle JDKが推奨されています。Oracle社のWebサイト[注1]からJDK7の最新版をダウンロードして、インストールします。

```
$ sudo rpm -ivh jdk-7u71-linux-x64.rpm
```

リスト1のように**~/.bashrc**に環境変数を設定して、jps、jstack、jstatなどJavaまわりのコマンドが実行できるようにしておきます。

▼リスト1　~/.bashrcの例(Javaの実行環境設定追加)

```
…… (略) ……
export JAVA_HOME=/usr/java/latest
export PATH=$PATH:$JAVA_HOME/bin
…… (略) ……
```

注1) http://www.oracle.com/technetwork/java/javase/downloads/jdk7-downloads-1880260.html

Hadoopのインストール

最新版のHadoop 2.6.0を、Apache Hadoopのミラーサイトからダウンロードして、/usr/local ディレクトリ配下に解凍します。

```
$ wget http://ftp.riken.jp/net/apache/hadoop/common/hadoop-2.6.0/hadoop-2.6.0.tar.gz
$ tar xf hadoop-2.6.0.tar.gz
$ sudo mv hadoop-2.6.0 /usr/local
$ sudo chown -R ec2-user:ec2-user /usr/local/hadoop-2.6.0
```

hadoopコマンドを実行させるため、**リスト2**のように**~/.bashrc**の環境変数を設定します。

リスト2では、$HADOOP_HOME/etc/hadoop 配下に設定ファイルが配置されています。一通りの動作確認をするためには、最低限でも**リスト3〜5**のように、ファイルを編集する必要があります。

注意する点としては、ここまでの設定作業を**図1**で示したすべてのノードで実施する必要があることです。

これらの設定作業が終わったところで各種デーモンを起動しますが、NameNodeを起動する前にHDFSのフォーマットが必要です。**図2**は、HDFS上のファイル情報を管理するメタデータ(fsimage)を初期化しています。

各種デーモンを実行するためのスクリプトは、**$HADOOP_HOME/sbin**に配置されています。NameNode、DataNode、ResourceManager、NodeManagerの順に起動します(**図3**)。

各種デーモンが起動していることをWeb UIから確認します。デフォルトのポート(NameNodeは50070番ポート、ResourceManagerは8088番ポート)にブラウザからアクセスするとWeb UIを視覚的に確認できます(**図4**)。

▼リスト2　~/.bashrcの例(Hadoopの実行環境設定追加)

```
…… (略) ……
export HADOOP_HOME=/usr/local/hadoop-2.6.0
export PATH=$PATH:$HADOOP_HOME/bin
…… (略) ……
```

▼リスト3　core-site.xml

```
<configuration>
  <!-- NameNodeのホスト名およびRPCポートを記述する。 -->
  <property>
    <name>fs.defaultFS</name>
    <value>hdfs://master:9000</value>
  </property>
</configuration>
```

▼リスト4　mapred-site.xml

```
<configuration>
  <!-- MapReduceをYARN上で実行するための設定。デフォルトでは、ローカルで実行する。 -->
  <property>
    <name>mapreduce.framework.name</name>
    <value>yarn</value>
  </property>
</configuration>
```

▼リスト5　yarn-site.xml

```
<configuration>
  <!-- ResourceManagerのホスト名を記述する -->
  <property>
    <name>yarn.resourcemanager.hostname</name>
    <value>master</value>
  </property>

  <!-- MapReduceをYARN上で実行するための設定その2。 -->
  <property>
    <name>yarn.nodemanager.aux-services</name>
    <value>mapreduce_shuffle</value>
  </property>
</configuration>
```

▼図2　HDFSのフォーマットの様子（masterサーバで実行）

```
(master)$ hdfs namenode -format
…… (略) ……
15/01/12 03:36:23 INFO common.Storage: Storage directory /tmp/hadoop-ec2-user/dfs/name has been
successfully formatted.
15/01/12 03:36:23 INFO namenode.NNStorageRetentionManager: Going to retain 1 images with txid >= 0
15/01/12 03:36:23 INFO util.ExitUtil: Exiting with status 0
15/01/12 03:36:23 INFO namenode.NameNode: SHUTDOWN_MSG:
/************************************************************
SHUTDOWN_MSG: Shutting down NameNode at master/10.188.149.180
************************************************************/
```

▼図3　各種デーモンの開始（プロンプト先頭のMaster、Slaveはサーバを示す）

```
(master)$ $HADOOP_HOME/sbin/hadoop-daemon.sh start namenode
(slave)$ $HADOOP_HOME/sbin/hadoop-daemon.sh start datanode
(master)$ $HADOOP_HOME/sbin/yarn-daemon.sh start resourcemanager
(slave)$ $HADOOP_HOME/sbin/yarn-daemon.sh start nodemanager
```

DataNodeタブをクリックすると、起動しているDataNodeの一覧が表示されます（**図5**）。それぞれResourceManagerのWeb UIで確認できます（**図6**）。

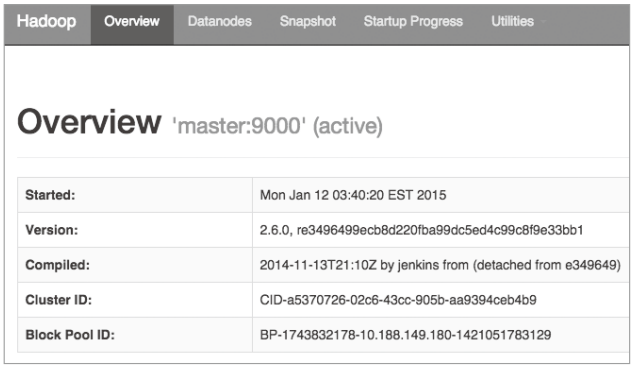
左側のNodesをクリックすると、起動しているNodeManagerの一覧が表示されます（**図7**）。

次に、MapReduceジョブが動作することを確認します。まず、HDFS上にジョブを実行するために必要なディレクトリを次のように作成します。

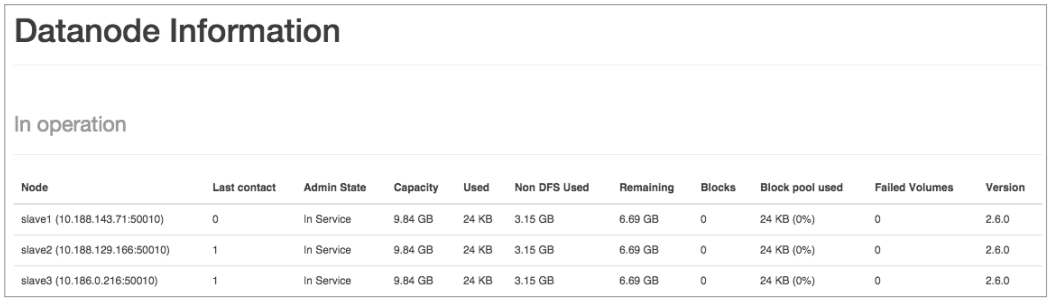
```
(master) $ hdfs dfs -mkdir -p /user/ec2-user
```

このコマンドを実行すると次のようなWARNメッセージが出力されます。HadoopはJavaで実装されていますが、速度を重視する場面ではC言語で書かれたライブラリを利用します。本手順ではそのライブラリをビルドしていないために次ページのようなメッセージが出力されています。存在しない場合はそれに対応するJavaのクラスが実行されるため、無視しても問題ありません。

▼図4　NameNodeのWeb UI



▼図5　起動しているDataNode一覧



```
15/01/12 03:57:25 WARN util.NativeCodeLoader:
Unable to load native-hadoop library for your
platform... using builtin-java classes where
applicable
```

ここまで正しく設定できたかどうか確認するため、サンプルのMapReduceジョブを実行します(**図8**)。

図8のジョブは、モンテカルロ法により円周率の近似値を計算するものです。ジョブ起動中のスレーブの情報を確認してみましょう。それぞれSlave1からSlave3まで確認します。

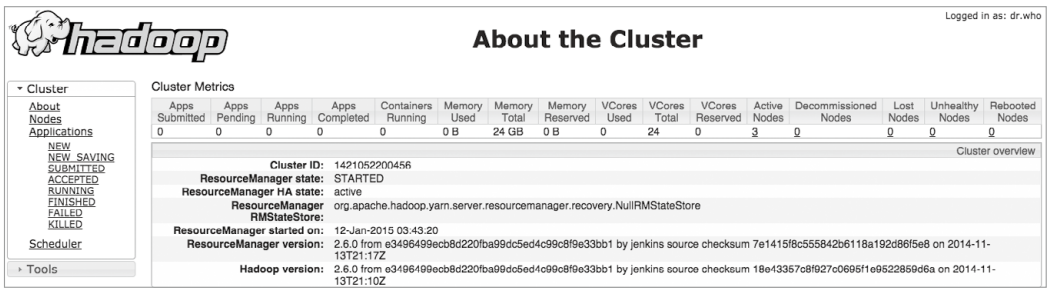
```
(slave1)$ jps
550 YarnChild
24672 NodeManager
2417 DataNode
906 YarnChild
1054 Jps
29073 MRAppMaster
741 YarnChild
```

```
(slave2)$ jps
24261 DataNode
6965 YarnChild
6792 YarnChild
6588 YarnChild
7034 Jps
24638 NodeManager
```

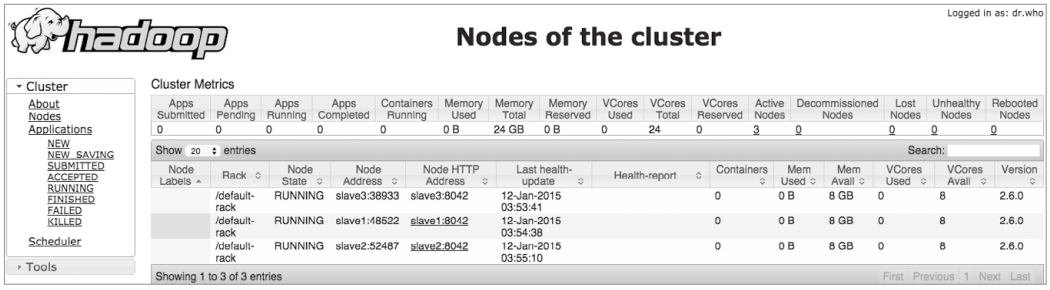
```
(slave3)$ jps
24347 DataNode
24580 NodeManager
9551 YarnChild
9260 YarnChild
9653 Jps
```

YarnChildが各スレーブサーバで複数起動しています。これが実際にデータ処理を実行しているプロセスで、MapもしくはReduceタスクが動作しています。MRAppMasterは、このMapReduceジョブのスケジューリングを担当し、MapReduceジョブ1つにつき1つ起動しま

▼図6　ResourceManagerのWeb UI



▼図7　起動しているNodeManager一覧



▼図8　サンプルのMapReduceジョブの実行（masterサーバで実行）

```
(master) $ yarn jar $HADOOP_HOME/share/hadoop/mapreduce/hadoop-mapreduce-examples-2.6.0.jar pi 10 10000
Number of Maps  = 10
Samples per Map = 10000
…… (略) ……
Job Finished in 43.363 seconds
Estimated value of Pi is 3.14120000000000000000
```

す。この場合、MRAppMasterはslave1で起動していることがわかります。

Hiveのインストール

最新版のHive 0.14.0を、Apache Hiveのミラーサイトからダウンロードします(**図9**)。hiveコマンドが実行できるように~/.bashrcの環境変数を設定します(**リスト6**)。これらを設定すると、Hiveが動作するようになります。

Hive on MapReduceの実行

本記事では、Wikipediaのページビューに関する統計データ"Page view statistics for Wikimedia projects(http://dumps.wikimedia.org/other/pagecounts-raw/)"を利用してクエリを動作させてみます。

統計データは**図10**のように、"言語およびプロジェクトの種類","タイトル名","リクエスト数","転送したデータサイズ"が、スペース区切りで配置されています。

図10のようなデータから、2015年1月1日のデータを取得してHDFS上に配置します(**図11**)。1時間あたり1ファイルなので、全部で24ファイルがHDFS上に配置されます。HDFSに配置されたファイルは、各DataNodeのディスク上に配置されています。HDFSのブロックの複製数が3で、DataNodeの台数も3なので、DataNode3台に同一のデータが配置されていることになります。

次に、**図12**のようにクエリを実行して、HDFS上に配置したデータをHiveのテーブルにロードします。

データロードはすぐに終わります。なぜなら、実際にはHDFS上のデータをHive用領域(デフォルトでは、**/user/hive/warehouse**)に移動しているだけだからです(**図13**)。

次に、集計用のクエリを実行します(**図14**)。

このクエリは**図15**のように、2段のMapReduceで実行されます。1段目のMapReduceではGROUP BYの処理、2段目のMapReduceではORDER BYの処理が実行されています。**図16**からも、ApplicationMasterが2回起動した、すなわちMapReduceジョブが2回実行されたことが確認できます。

▼リスト6　~/.bashrcの例(Hiveの実行環境設定追加)

```
……(略)……
export HIVE_HOME=/usr/local/apache-hive-0.14.0-bin
export PATH=$PATH:$HIVE_HOME/bin
```

▼図9　Hiveのインストール(MasterサーバでHiveをダウンロードし解凍、/usr/localにコピー)

```
(master)$ wget http://ftp.riken.jp/net/apache/hive/hive-0.14.0/apache-hive-0.14.0-bin.tar.gz
(master)$ tar xf apache-hive-0.14.0-bin.tar.gz
(master)$ sudo mv apache-hive-0.14.0-bin /usr/local
```

▼図10　Wikipediaのページビューに関する統計データ(例)

```
fr.b Special:Recherche/Achille_Baraguey_d%5C%27Hilliers 1 624
fr.b Special:Recherche/Acteurs_et_actrices_N 1 739
fr.b Special:Recherche/Agrippa_d/%27Aubign%C3%A9 1 743
fr.b Special:Recherche/All_Mixed_Up 1 730
fr.b Special:Recherche/Andr%C3%A9_Gazut.html 1 737
```

▼図11　DataNodeへHDFSで配置(Masterサーバで実行)

```
(master)$ wget http://dumps.wikimedia.org/other/pagecounts-raw/2015/2015-01/pagecounts-20150101-000000.gz
……(略)……
(master)$ wget http://dumps.wikimedia.org/other/pagecounts-raw/2015/2015-01/pagecounts-20150101-230000.gz
(master)$ hdfs dfs -put pagecounts-*.gz .
```

▼図12　Hiveのクエリを実行し、テーブルにロードする

```
prepare.hql
CREATE TABLE pagecounts (project STRING, title STRING, view INT, size BIGINT)
  ROW FORMAT DELIMITED FIELDS TERMINATED BY ' ';
LOAD DATA INPATH 'pagecounts-20150101-000000.gz' INTO TABLE pagecounts;
 ……(略)……
LOAD DATA INPATH 'pagecounts-20150101-230000.gz' INTO TABLE pagecounts;
 hive -fオプションで、ファイル名を指定して実行
(master)$ hive -f prepare.hql
 ……(略)……
Loading data to table default.pagecounts
Table default.pagecounts stats: [numFiles=1, totalSize=80094002]
OK
Time taken: 0.452 seconds
Loading data to table default.pagecounts
Table default.pagecounts stats: [numFiles=2, totalSize=170997787]
OK
 ……(略)……
Time taken: 0.212 seconds
Loading data to table default.pagecounts
Table default.pagecounts stats: [numFiles=24, totalSize=2284122018]
OK
Time taken: 0.175 seconds
```

▼図13　HDFS上のデータの本体の移動

```
(master)$ hdfs dfs -ls /user/hive/warehouse/pagecounts
15/01/12 05:23:47 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform...
using builtin-java classes where applicable
Found 24 items
-rw-r--r--   3 ec2-user supergroup   80094002 2015-01-12 05:16 /user/hive/warehouse/pagecounts/
pagecounts-20150101-000000.gz
 ……(略)……
-rw-r--r--   3 ec2-user supergroup  101452277 2015-01-12 05:02 /user/hive/warehouse/pagecounts/
pagecounts-20150101-230000.gz
```

▼図14　集計用クエリ（ranking.hql）の実行

```
ranking.hql
 プロジェクトごとにビュー数を集計して、大きい順番に表示する
SELECT project, SUM(view) AS s FROM pagecounts
  GROUP BY project
  ORDER BY s DESC;
$ hive -f ranking.hql
 ……(略)……
Total jobs = 2
 ……(略)……
Launching Job 1 out of 2
 ……(略)……
Starting Job = job_1421052200456_0005, Tracking URL = http://master:8088/proxy/
application_1421052200456_0005/
Kill Command = /usr/local/hadoop-2.6.0/bin/hadoop job  -kill job_1421052200456_0005
Hadoop job information for Stage-1: number of mappers: 24; number of reducers: 9
2015-01-12 05:31:34,534 Stage-1 map = 0%,  reduce = 0%
 ……(略)……
2015-01-12 05:33:55,011 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 491.19 sec
MapReduce Total cumulative CPU time: 8 minutes 11 seconds 190 msec
Ended Job = job_1421052200456_0005
Launching Job 2 out of 2
 ……(略)……
```

次ページに続く

前ページより

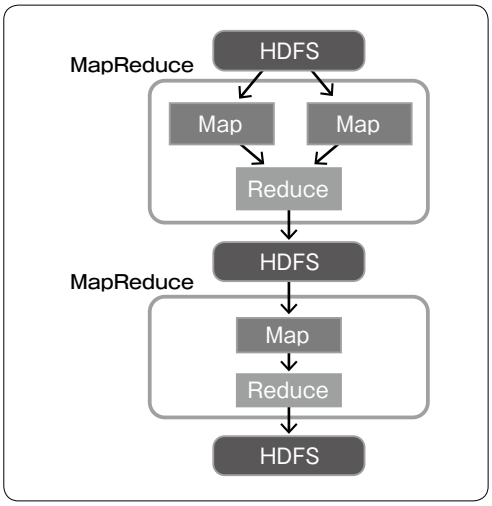
```
Starting Job = job_1421052200456_0006, Tracking URL = http://master:8088/proxy/
application_1421052200456_0006/
Kill Command = /usr/local/hadoop-2.6.0/bin/hadoop job  -kill job_1421052200456_0006
Hadoop job information for Stage-2: number of mappers: 1; number of reducers: 1
2015-01-12 05:34:03,892 Stage-2 map = 0%,  reduce = 0%
2015-01-12 05:34:12,135 Stage-2 map = 100%,  reduce = 0%, Cumulative CPU 2.07 sec
2015-01-12 05:34:20,366 Stage-2 map = 100%,  reduce = 100%, Cumulative CPU 4.03 sec
MapReduce Total cumulative CPU time: 4 seconds 30 msec
Ended Job = job_1421052200456_0006
MapReduce Jobs Launched:
Stage-Stage-1: Map: 24  Reduce: 9   Cumulative CPU: 491.19 sec   HDFS Read: 2284127634 HDFS Write:
35417 SUCCESS
Stage-Stage-2: Map: 1  Reduce: 1   Cumulative CPU: 4.03 sec   HDFS Read: 37875 HDFS Write: 13839 SUCCESS
Total MapReduce CPU Time Spent: 8 minutes 15 seconds 220 msec
OK
↓クエリの実行結果が表示されている
meta.mw 213186062 ←メタ情報(モバイル)
meta.m  197948682 ←メタ情報(Wikimedia)
en  153295080 ←英語
en.mw   134486420 ←英語(モバイル)
ru  25701272 ←ロシア語
ja.mw   23854627 ←日本語(モバイル)
de  21748744 ←ドイツ語
ja  19470520 ←日本語
fr  16845470 ←フランス語
de.mw   16816816 ←ドイツ語(モバイル)
zh  16590020 ←中国語
……(略)……
Time taken: 179.243 seconds, Fetched: 1370 row(s)
```

Tezのインストール

Tezにはバイナリが用意されていないため、最新リリースバージョンであるTez 0.5.3をビルドします。ビルドするためには、Apache MavenとProtocolBuffersが必要なのでインストールする必要があります。ここで、Mavenのバージョンは3以降、ProtocolBuffersのバージョンは2.5.0を利用してください。

Tezのソースコードをチェックアウト(**図17**)した後にpom.xmlを修正して、hadoop.versionプロパティを2.6.0に変更します。

▼図15　Hive(MapReduce)の処理の流れ



▼図16　Hive(MapReduce)実行時のResourceManager Web UI

Show 20 entries　　Search:

ID	User	Name	Application Type	Queue	StartTime	FinishTime	State	FinalStatus	Progress	Tracking UI
application_1421052200456_0006	ec2-user	SELECT project, SUM(view) AS s FROM p...DESC(Stage-2)	MAPREDUCE	default	Mon, 12 Jan 2015 10:33:56 GMT	Mon, 12 Jan 2015 10:34:19 GMT	FINISHED	SUCCEEDED		History
application_1421052200456_0005	ec2-user	SELECT project, SUM(view) AS s FROM p...DESC(Stage-1)	MAPREDUCE	default	Mon, 12 Jan 2015 10:31:26 GMT	Mon, 12 Jan 2015 10:33:54 GMT	FINISHED	SUCCEEDED		History

▼図17　TezのソースコードをGitレポジトリからチェックアウト（Masterサーバ）

```
(master)$ git clone https://github.com/apache/tez.git
(master)$ cd tez
(master)$ git checkout -b release-0.5.3 release-0.5.3
```

▼リスト7　tez-site.xml

```
<configuration>
  <property>
    <name>tez.lib.uris</name>
    <value>${fs.defaultFS}/apps/tez/tez-0.5.3.tar.gz</value>
  </property>
</configuration>
```

▼図18　マスタサーバからクライアントへのjarファイル配置

```
(master)$ sudo mkdir -p /usr/local/tez-jars/0.5.3
(master)$ sudo chown -R ec2-user:ec2-user /usr/local/tez-jars
(master)$ export TEZ_JARS=/usr/local/tez-jars/0.5.3
(master)$ tar xvzf tez-dist/target/tez-0.5.3-minimal.tar.gz -C $TEZ_JARS
```

▼リスト8　hadoop-env.sh

```
for f in $HADOOP_HOME/contrib/capacity-scheduler/*.jar; do
  if [ "$HADOOP_CLASSPATH" ]; then
    export HADOOP_CLASSPATH=$HADOOP_CLASSPATH:$f
  else
    export HADOOP_CLASSPATH=$f
  fi
done
+
+ # tez-site.xmlを配置したディレクトリを指定する
+ export TEZ_CONF_DIR=$HADOOP_HOME/etc/hadoop
+ export TEZ_JARS=/usr/local/tez-jars/0.5.3
+ export HADOOP_CLASSPATH=${HADOOP_CLASSPATH}:${TEZ_CONF_DIR}:${TEZ_JARS}/*:${TEZ_JARS}/lib/*
```

Tezをビルドするためのコマンドは、次のとおりです。

```
(master)$ mvn clean package -DskipTests=true ➥
-Dmaven.javadoc.skip=true
```

Tezを実行するためには、先ほどビルドしたTezのライブラリをHDFSに配置する必要があります。

```
(master)$ hdfs dfs -mkdir -p /apps/tez
(master)$ hdfs dfs -put tez-dist/target/➥
tez-0.5.3.tar.gz /apps/tez
```

マスタの$HADOOP_HOME/etc/hadoop配下（先ほど編集した各種設定ファイルと同じディレクトリ）に、Tez用の設定ファイル（tez-site.xml）を用意して、tez.lib.urisプロパティに先ほどライブラリを配置したHDFS上のパスを指定します（**リスト7**）。

また、ジョブを実行するクライアントにもjarを配置する必要があります（**図18**）。

次に、$HADOOP_HOME/etc/hadoop配下にあるhadoop-env.shを編集して、javaのクラスパスにこれらのjarを追加します（**リスト8**）。

Hive on Tezの実行

Hive側でTezを利用する設定は、この1行だけです。

```
set hive.execution.engine=tez;
```

この行をsample.hqlの1行目に追加して実行すると、Hive on Tezが動作します（**図19**）。

▼図19　Hive on Tezの実行

```
$ hive -f ranking.hql
……(略)……
Query ID = ec2-user_20150112061717_dc9057e6-b70d-4048-805f-1440ed54893e
Total jobs = 1
Launching Job 1 out of 1

Status: Running (Executing on YARN cluster with App id application_1421052200456_0007)

--------------------------------------------------------------------------------
        VERTICES      STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
--------------------------------------------------------------------------------
Map 1 ..........   SUCCEEDED     24         24        0        0       0       0
Reducer 2 ......   SUCCEEDED      9          9        0        0       0       0
Reducer 3 ......   SUCCEEDED      1          1        0        0       0       0
--------------------------------------------------------------------------------
VERTICES: 03/03  [==========================>>] 100%  ELAPSED TIME: 107.98 s
--------------------------------------------------------------------------------
OK
meta.mw 213186062
meta.m  197948682
en  153295080
en.mw   134486420
ru  25701272
ja.mw   23854627
de  21748744
ja  19470520
fr  16845470
de.mw   16816816
zh  16590020
……(略)……
Time taken: 118.755 seconds, Fetched: 1370 row(s)
```

MapReduceの場合とは違い、Progress barが用意され、リアルタイムでジョブの進行状況が更新されるようになりました。この機能はHive 0.14で追加され、ジョブ全体の進捗を把握することが非常に簡単になりました(**図20**)。

ここで**図20**を見ると、Map → Reduce → Reduceの順番で処理が実行されていることがわかります。MapReduceでは、**図15**のように2回MapReduceジョブを実行する必要がありましたが、Tezでは余計なMapタスクを起動することなく処理していることがわかります。

また、**図21**を見るとApplication TypeがTEZになっており、確かにTez上で実行されたということがわかります。また、ApplicationMasterの起動も1回になっていることがわかります。

以上で説明した効率化によって、クエリの実行時間もMapReduceの179秒と比較して、Tezは118秒とより高速になりました。SD

▼図20　ジョブ実行中のProgress bar

```
        VERTICES      STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
--------------------------------------------------------------------------------
Map 1 .....          RUNNING     24         12       12        0       0       0
Reducer 2            RUNNING      9          0        4        5       0       0
Reducer 3             INITED      1          0        0        1       0       0
--------------------------------------------------------------------------------
VERTICES: 00/03  [==========>>----------] 35%   ELAPSED TIME: 95.43 s
```

▼図21　Hive(Tez)実行後のResourceManager Web UI

Show 20 entries　　Search:

ID	User	Name	Application Type	Queue	StartTime	FinishTime	State	FinalStatus	Progress	Tracking UI
application_1421052200456_0007	ec2-user	HIVE-75a4d8db-ac2a-49aa-85f5-603ab354cb0f	TEZ	default	Mon, 12 Jan 2015 11:17:17 GMT	Mon, 12 Jan 2015 11:19:17 GMT	FINISHED	SUCCEEDED		History

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2月号

第1特集
Linux systemd入門
あなたの知らない実践技

第2特集
そろそろ、やめませんか？
なぜ「運用でカバー」がダメなのか？

一般記事
・Intel DPDK技術詳解
・これはなんて読む？　UNIX用語読み方指南
・Googleベンチャーズが提唱するデザインスプリントとはほか

定価（本体1,220円＋税）

2015年1月号

第1特集
プログラマ・インフラエンジニア・文章書きの心得
Vim使い事始め

第2特集
SI崩壊を乗りきる3つの方法
ソフトウェア開発の未来

一般記事
・「ひみつのLinux通信」年末年始スペシャル　ITエンジニア出世双六
・Jamesのセキュリティレッスン［最終回］

定価（本体1,220円＋税）

2014年12月号

第1特集
急速に普及するコンテナ型仮想環境
Dockerを導入する理由

第2特集
基礎の基礎から押さえる必須技術
やさしくわかるVPNの教科書

一般記事
・bashの脆弱性“Shellshock”その影響と対策
・SoftLayerを使ってみませんか？［最終回］
・Jamesのセキュリティレッスン［2］

定価（本体1,220円＋税）

2014年11月号

第1特集
Docker・Ansible・シェルスクリプト
無理なくはじめる Infrastructure as Code

第2特集
オンプレミスもクラウドも縦横無尽
サーバの目利きになる方法［後編］

一般記事
・8086時代から今を俯瞰する　CPU温故知新
・はてな謹製、サーバ管理ツール　Mackerel入門

定価（本体1,220円＋税）

2014年10月号

第1特集
今ふたたびのJava
言語仕様・開発環境・デバッグ機能

第2特集
オンプレミスを制するものはクラウドを制する
サーバの目利きになる方法［前編］

一般記事
・オーケストレーションツールSerf・Consul入門［Consul編］
・SoftLayerを使ってみませんか？［2］　ほか

定価（本体1,220円＋税）

2014年9月号

第1特集
この夏に克服したい2つの壁
C言語のポインタとオブジェクト指向

第2特集
止まらないサービスを支えるシステム構築の基礎
クラスタリングの教科書

一般記事
・SoftLayerを使ってみませんか？
・NICをまとめて高速通信！（前編）
・Serf・Consul入門　特別定価（本体1,300円＋税）

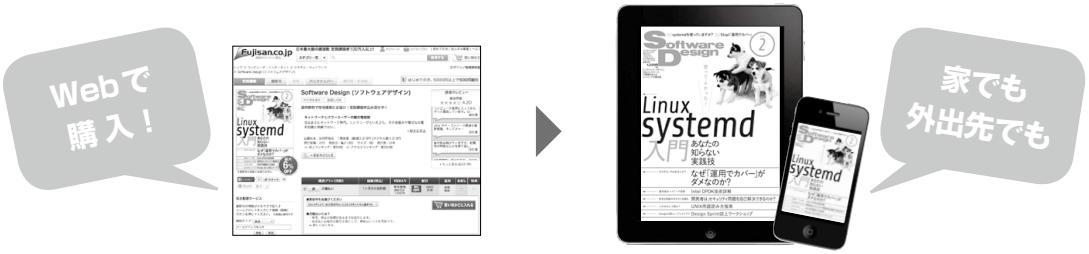
Software Design　バックナンバー常備取り扱い店							
北海道	札幌市中央区	MARUZEN＆ジュンク堂書店　札幌店	011-223-1911	神奈川県	川崎市高津区	文教堂書店　溝の口本店	044-812-0063
	札幌市中央区	紀伊國屋書店　札幌本店	011-231-2131	静岡県	静岡市葵区	戸田書店　静岡本店	054-205-6111
東京都	豊島区	ジュンク堂書店　池袋本店	03-5956-6111	愛知県	名古屋市中区	三洋堂書店　上前津店	052-251-8334
	新宿区	紀伊國屋書店　新宿本店	03-3354-0131	大阪府	大阪市北区	ジュンク堂書店　大阪本店	06-4799-1090
	渋谷区	紀伊國屋書店　新宿南店	03-5361-3315	兵庫県	神戸市中央区	ジュンク堂書店　三宮店	078-392-1001
	千代田区	書泉ブックタワー	03-5296-0051	広島県	広島市南区	ジュンク堂書店　広島駅前店	082-568-3000
	千代田区	丸善　丸の内本店	03-5288-8881		広島市中区	丸善　広島店	082-504-6210
	中央区	八重洲ブックセンター本店	03-3281-1811	福岡県	福岡市中央区	ジュンク堂書店　福岡店	092-738-3322
	渋谷区	MARUZEN＆ジュンク堂書店　渋谷店	03-5456-2111				

※店舗によってバックナンバー取り扱い期間などが異なります。在庫などは各書店にご確認ください。

デジタル版のお知らせ　DIGITAL

デジタル版Software Design 好評発売中！紙版で在庫切れのバックナンバーも購入できます

本誌デジタル版は「Fujisan.co.jp」（http://www.fujisan.co.jp/sd）と、「雑誌オンライン.com」（http://www.zasshi-online.com/）で購入できます。最新号、バックナンバーとも1冊のみの購入はもちろん、定期購読も対応。1年間定期購読なら5%強の割引になります。デジタル版はPCのほかにiPad／iPhoneにも対応し、購入するとどちらでも追加料金を払うことなく、読むことができます。



特別企画

手軽に仮想化技術を応用

Cisco VIRLでネットワークのシミュレーション（前編）

――自在に設計・試行・動作確認が可能な最新ツール

Writer 山下 薫（やました かおる） kaoru@cisco.com

Cisco VIRLは、コンピュータネットワークを手軽に仮想的に構築できるソフトウェアです。実際のネットワーク機器のOSと同じソースコードを用いたシミュレーションが可能で、現在の仮想化ネットワークの隆盛にマッチした豊富な機能を備えています。前後編でその可能性と面白さを解説します。

ネットワーク機器の現状とシミュレーションの必要性

ソフトウェアの力でネットワークを変えていこう、という動きが活発です。たとえばOpenStack Neutronなどで用いられているOpen vSwitch（OvS）のように、LANスイッチなどの従来のネットワーク機器の形態と実装を見直して、ソフトウェアベースのものに置き換えていこう、というアプローチがあります。また、PuppetやChefのようなサーバの設定を自動化するしくみを、ネットワーク機器にも適用できるようにする、すなわちネットワークをよりサーバやアプリケーションに近い形で管理運用できるように変えていこう、という取り組みも盛んです。

本稿では、従来のネットワーク機器を出発点にして、OpenStackなどの新しいソフトウェア技術を適用することで、ネットワークの構築、運用管理、勉強、トラブル対応などを楽にするアプローチの、Cisco VIRL（Virtual Internet Routing Lab）というソフトウェアを紹介します。

VIRL≒フライトシミュレータ

まず、Cisco VIRL（以下「VIRL」と表記）とはざっくり何なのかを、旅客機とフライトシミュレータに例えて説明します。従来のネットワーク機器（実機）を旅客機[注1]だと思ってください。旅客機は厳しい訓練を受けたパイロットしか操作することはありません。パイロットの訓練には、多くの場合フライトシミュレータが使われます。シミュレータですので、誤って操作しても実害はありませんし、何より実際の旅客機で訓練するよりコストが安く済みます。

VIRLは、まさにフライトシミュレータに相当します。筆者は、パイロットの訓練に使われている本物のフライトシミュレータに乗ったことはありませんが、普通に入手できるゲームのフライトシミュレータとはまったく別次元[注2]のものだそうです。

VIRLは、実際のネットワーク機器のOSのソースコードの多くの部分を利用しているので、使い方によっては実機と区別が付かないこともあります。

注1）タイトルバックの絵は戦闘機っぽいですが気にしないでください（笑）。

注2）http://ascii.jp/elem/000/000/898/898978/index-4.html

それでは、まずVIRLの出発点である従来のネットワーク機器について復習し、VIRLとは具体的に何なのか、何ができるのか、どうやって動かすのかを解説していきます。

実機検証を困難にする地味な課題

ネットワークの構築例

図1に、単純なネットワークの例を示します。このネットワークを実機で組んで試験する場合と、VIRLを用いた場合を比較しながら、解説を進めます。

図1の構成には、3台のネットワーク機器が含まれています。具体的には、IOSが動作する比較的小型のルータ、IOS XRを用いる大型のルータ、さらにNX-OSを採用しているスイッチの3台です。

実際のネットワークでは、Cisco製品ではない機器やロードバランサ、ファイアウォール、無線LAN関連機器、さらにはサーバとクライアントなどが含まれることが一般的ですが、解説のため単純化しています。

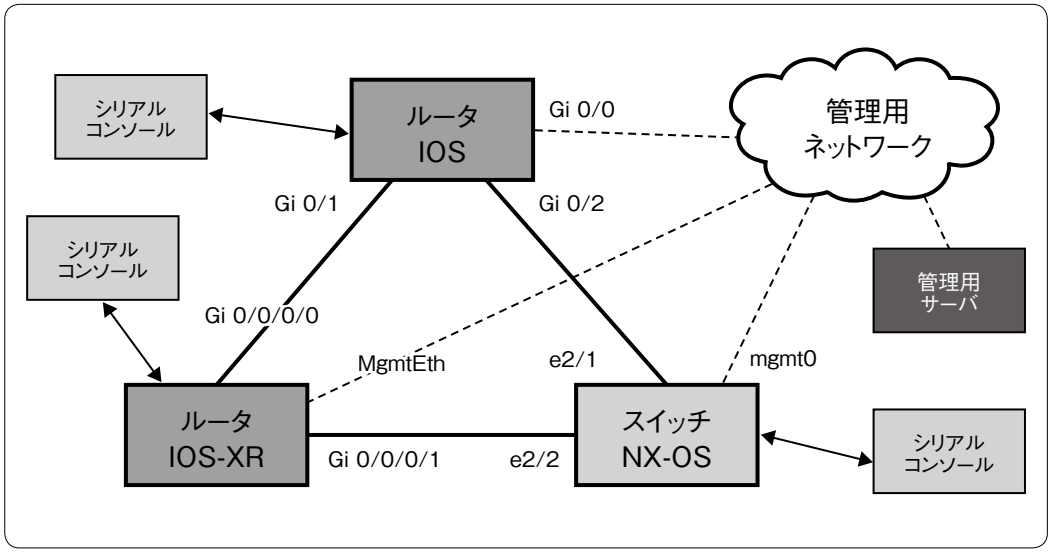
3台のルータとスイッチはそれぞれ直結され、三角形のトポロジを組んでいます。また、初期設定のためシリアルコンソールを接続しています。

比較的新しい機器では、管理のための専用のEthernetインターフェースを持っており、IOS XRでは「MgmtEth」、NX-OSでは「mgmt0」という名前が付いています。IOSルータにはこれに相当するものがないので、代わりにGi 0/0を管理用ネットワークに接続しています。初期設定が完了すれば、**図1**右上の管理用ネットワークを経由して、3台のルータとスイッチにアクセスできるようになります。また、OSのファイルのコピーやそのほかの管理のために、**図1**のように管理用ネットワークにサーバを接続することが普通です。

具体的に生じる問題は何か？

では、**図1**のネットワークを実際に「組む」には、どの程度の手間と時間が必要になるでしょうか。これは筆者のざっくりした感覚ですが、すべての機器とケーブル、設置場所と電源などがすべて確保できていたとして、仮に1人で作業すると、順調に進んでも数時間はかかりそうです。大型の機器は1人ではラックマウントできなかったり、慣れない機種(筆者の場合は

▼図1　単純なネットワークの例



IOS XRルータ)について調べたり同僚に聞いたりすると、さらに時間がかかります。また、必ずしも各機器にインストールされているOSのバージョンが、想定どおりとは限らないので、管理用サーバとの通信が可能になってから、順次OSのファイル(イメージ)をコピーしてバージョンを合わせます。

さらに、**図1**のネットワークはとても単純なトポロジですが、実際のネットワークは機器の台数やリンクが多く、はるかに複雑です。ケーブルが多くなってくると、かき集めてきたケーブルが途中で切れていたり、コネクタが壊れていてつながらないという経験をされた方も多いと思います。

もっと本質的な問題は、設計からの変更に伴う手戻りです。組み上げたあとで要求仕様が変更される場合、配線のやり直しや設定(コンフィグ)の修正などが必要になります。また、当初の設計が機器の仕様と合わないことが、実際に組んで試験を始めてから判明することもあります。たとえば、サポートされていると思っていた機能やコマンドが、実機では動作しなかったり投入できないケースです。

こうなると、ユーザやアプリケーション、サーバ側などの要求と、ネットワーク側で実現可能なラインをすり合わせ、落としどころを見つける作業が必要になり、これはかなり難易度が高く時間を要します。

ネットワーク機器のアーキテクチャ——専用ハードウェアと汎用CPU

前述のようにネットワーク機器の実機を旅客機、VIRLをフライトシミュレータに例えました。ジェット旅客機は、およそ50年ほど前から現在のような形になっています。巡航速度も、マッハ0.8から0.9程度で変わっていないそうです[注3]。ネットワーク機器はそれより歴史ははるかに短いですが、LANスイッチの場合、今からおよそ20年前からの約10年間にアーキテクチャが急激に進化し、「ワイヤレート(ワイヤスピード)」の性能が普通に達成できるようになりました。大型のルータも、個別には実装が異なる点がありますが、大まかなアーキテクチャはほぼ同じです。

ネットワーク機器のしくみ

それでは、VIRLの解説に進む前に、現在のネットワーク機器のアーキテクチャを紹介します。**図2**が、典型的なネットワーク機器(スイッチやルータなど)の内部構造です。

物理インターフェースがネットワーク機器の外部にあるコネクタに相当し、ほかのネットワーク機器やサーバなどとのあいだでパケットを送受信します。大型の機器では数百の物理インターフェースを持っています。現在では、物理インターフェースはほとんどがEthernetで、お馴染みのRJ45コネクタや、距離が長い場合は光ファイバを使用します。

バックプレーンは、物理インターフェースから別の物理インターフェースへ転送されるパケットを、機器の内部で運びます。高速のインターコネクトと、実際にパケットを仕分けるためのクロスバースイッチや共有メモリを用いて実装されています。

データプレーンは、パケットのヘッダを読み取り、ルーティングテーブルやフィルタリングの設定などに従って、そのパケットをどのように転送するかを決定します。専用ハードウェアに実装されていることが多く、すべての物理インターフェースに最大の負荷がかかり、ルーティングテーブルなどの各種情報が複雑になっても、パケットをとりこぼすことなく転送します[注4]。

コントロールプレーンは、ネットワーク機器全体を制御します。ルーティングテーブルや、そのほかの情報を絶えず更新して最新の状態に

注3) アビオニクス(Avionics:航空機に搭載される電子機器の総称)や燃費など、現在までに大きく進化している部分もあります。

注4) データプレーンが持っている各種ハードウェアテーブルの容量の範囲に収まっている場合。

保ち、それをデータプレーンに書き込み続けます。コントロールプレーンは汎用のCPUと専用のOSで構成されています。OSは、機種に依存しないPlatform Independent(PI)と、デバイスドライバに相当するPlatform Dependent(PD)に分けられます。Cisco社のネットワーク機器では、目的別に複数のOSが用いられており、ルータとスイッチではIOS、IOS XE、IOS XR、NX-OSなどが使われます。

ネットワーク専用機器のシミュレーションの有用性

ネットワークに発生する問題とは何か？

現在、**図2**のような内部構造を持つネットワーク機器を、よりサーバやアプリケーションに近い形で管理運用できるように変えていこうという動きや、ソフトウェアベースのものに置き換えるというアプローチが注目を集めています。このため、将来はネットワーク機器の位置づけが大きく変わるかもしれません。

しかし、もしそうであったとしても、移行期には従来のネットワーク機器との併用と相互接続が必要になり、エンジニアは従来のネットワーク機器と新しい技術の両方に対処しなければなりません。

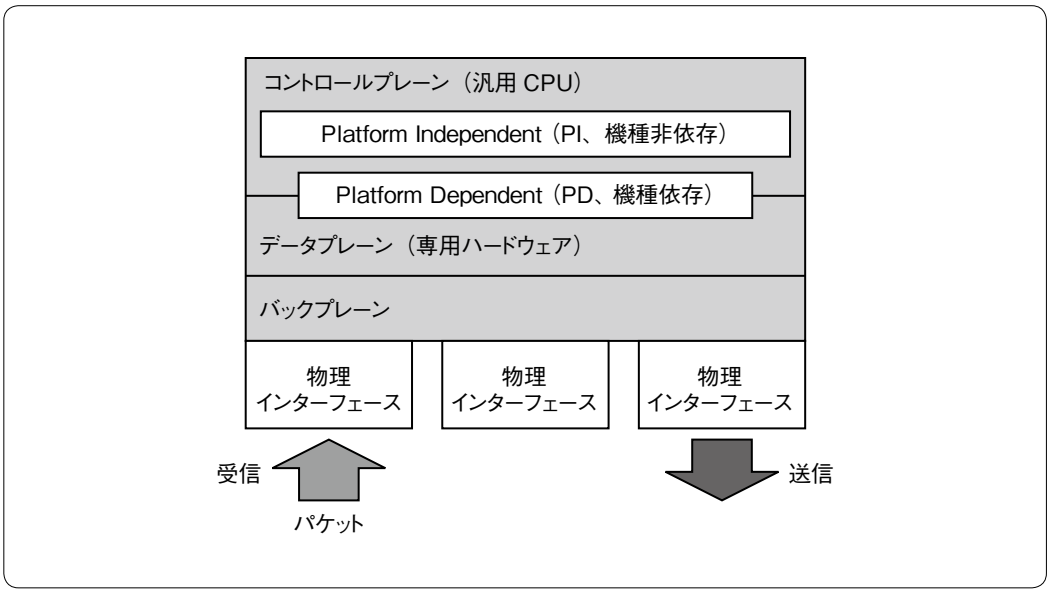
逆に、新しいソフトウェア中心のネットワークのための技術を、従来のネットワーク機器そのものにも適用できれば、移行のためのハードルを下げるための手段の1つになり得ます。それが、ネットワーク機器のOSを実機から取り出し、x86サーバで稼動させてネットワークをシミュレートするという手法です。

この手法は、従来のネットワーク機器で構成されている既存のネットワークの運用管理を楽にするためにも、当然応用できます。先に解説した、実機での検証作業のための工数を劇的に少なくできるのはもちろん、シミュレーションが実機に近ければ近いほど、実機を使わずに設計の確認や、トラブルシュートができる可能性が高まります。

実機で問題が起こる前に、原因の解析ができる可能性

先に、実機での検証の「あるある」を解説しま

▼図2　ネットワーク機器の内部構造

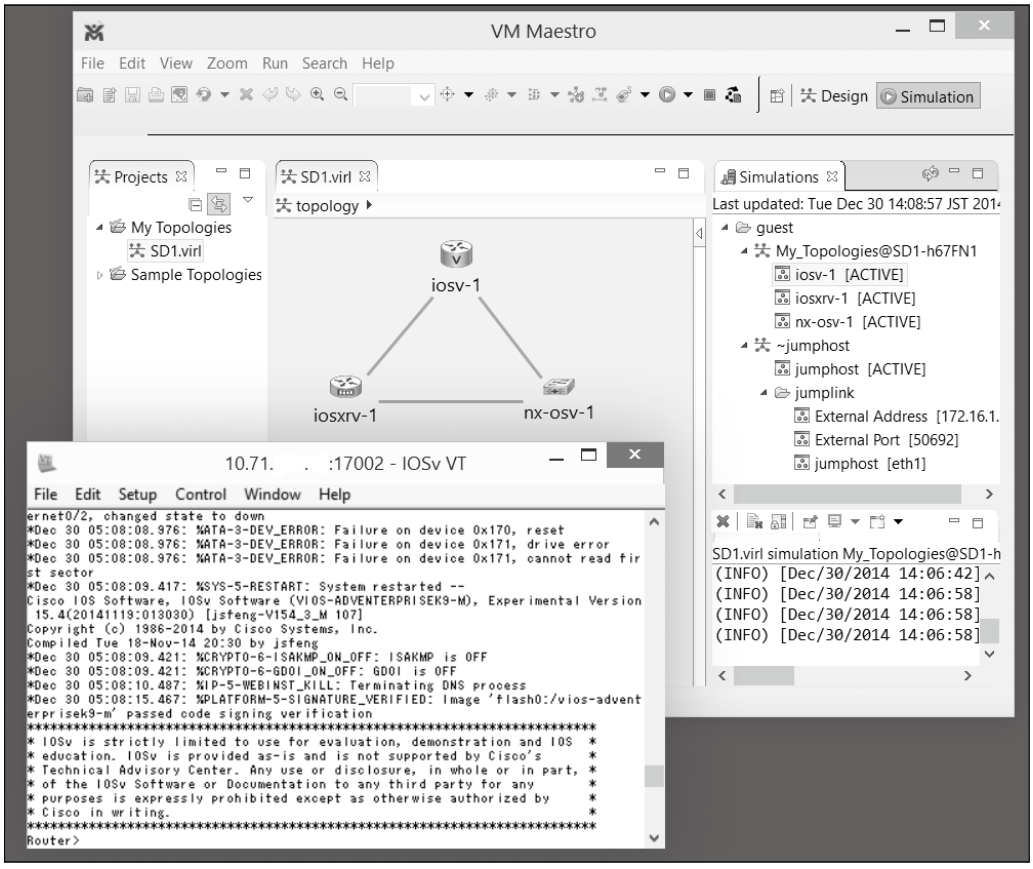


した。そこでは実機は確保できている前提でしたが、実はこれは簡単ではありません。旅客機ほど巨大で高価ではありませんが、とくに大型のネットワーク機器の場合、実機を一定期間確保するのはたいへんです。また、旅客機のような高度で定期的な整備が必要ではないとはいえ、ネットワーク機器でも、本当にその機器が正しく動いているのかを確実に判定するのは容易なことではありません。自己診断機能の充実や、Ethernetの相互接続性の問題が少なくなってきたために、「動かない」「つながらない」で検証が止まってしまうことは珍しくなりましたが、ハードウェアに起因する問題で想定どおりに動かないと、原因の切り分けや解決はやはり難しいものがあります。VIRLではネットワーク機器のハードウェアが仮想化されているので、もし実機でうまく疎通しない場合に、その構成をVIRL上で再現して、同じ問題が発生すればハードウェアが原因ではないと判断できる可能性があります。

VIRLは問題を解決する糸口を作りだす

また、現時点のソフトウェアベースのルータやスイッチは、Intel DPDK(本誌2015年2月号参照)などの技術によって、それなりの性能を達成しています。しかし、筆者が調べた限りでは、100ギガビットEthernet(100GbE)を収容するには、専用ハードウェア(データプレーン)を持つネットワーク機器が必要です。また、40ギガ(40GbE)や10ギガ(10GbE)であって

▼図3　VIRLによるシミュレーションの実行例



も、パケット長が短かい(1秒あたりに処理すべきパケットが多い)、収容するインターフェースの数が多い、さらにフィルタリングやQoSなどのルールが複雑な場合は、ソフトウェアベースではすべてのパケットを転送できないケースがあると思われます。何より、x86プロセッサとDPDKに従来のLANスイッチとまったく同じ仕事しかさせないということは、費用対効果の面で見て「もったいない」のが現状です。仮に性能を確保するために従来のネットワーク機器を使ったとしても、アプリケーションやサービスの拡張や変更のためにネットワーク側で対応が必要な場合などに、VIRLでコントロールプレーンやプロトコルの動作を確認してからその結果を実機に適用すれば、トータルの工数や所要時間を短くすることが可能になります。

そもそもVIRLとは何か

本物と同じソースコードを用いたシミュレーションの威力

VIRLは、x86サーバやPCで動くソフトウェアパッケージです。Cisco社の社外向けイベントなどで以前から紹介されてきましたが、個人ユーザ向けの「Personal Edition」の販売が、2014年12月1日から始まりました。**図3**が、実際にVIRLを用いて**図1**と同じネットワークトポロジのシミュレーションを実行している具体例です。3つのノードを接続し、上側のIOSにアクセスしています。

なお、「VIRL」の発音ですが、英語圏の人間は「バイラルメディア」と同じ「バイラル」、または「バイル」と発音しています。日本では、「バールのようなもの」の「バール」と同じように、平板な読み方をすることが多いようです。

VIRLの中で動作しているノードのうち、Cisco製のネットワーク機器に対応するものは、実機で稼動しているOSと同じソースコードを用いてビルドされています。たとえば、Nexus 7000用のNX-OSは、当初からx86 CPUで稼動していたため、実機用のバイナリをビルドすると、VIRLのノードとして動作するバイナリもビルドされるようになっています。

シミュレーションゆえの限界と制約

VIRLのノードには、実機の物理インターフェース(ラインカードなど)と同じハードウェアはありません。実機には、40GbEや100GbEに対応する物理インターフェースを持つ機種もあり、パケットを高速で処理するためのデータプレーンとセットで動作します。このため、実機用のOSには、物理インターフェースやデータプレーンに対するドライバが含まれています。

一方、現時点のVIRLのノードのインターフェースは、すべて仮想的なギガビットEthernet(1GbE)であり、実機のデータプレーンに対応するものはありません。Cisco製のネットワーク機器では、先に解説したハードウェアに依存しないコードに対応するバイナリ(PI)と、ハードウェアのドライバに対応するバイナリ(PD)が、1つのファイルとして提供されます。このため、実機用のバイナリとVIRL用のバイナリの間には、互換性がありません。

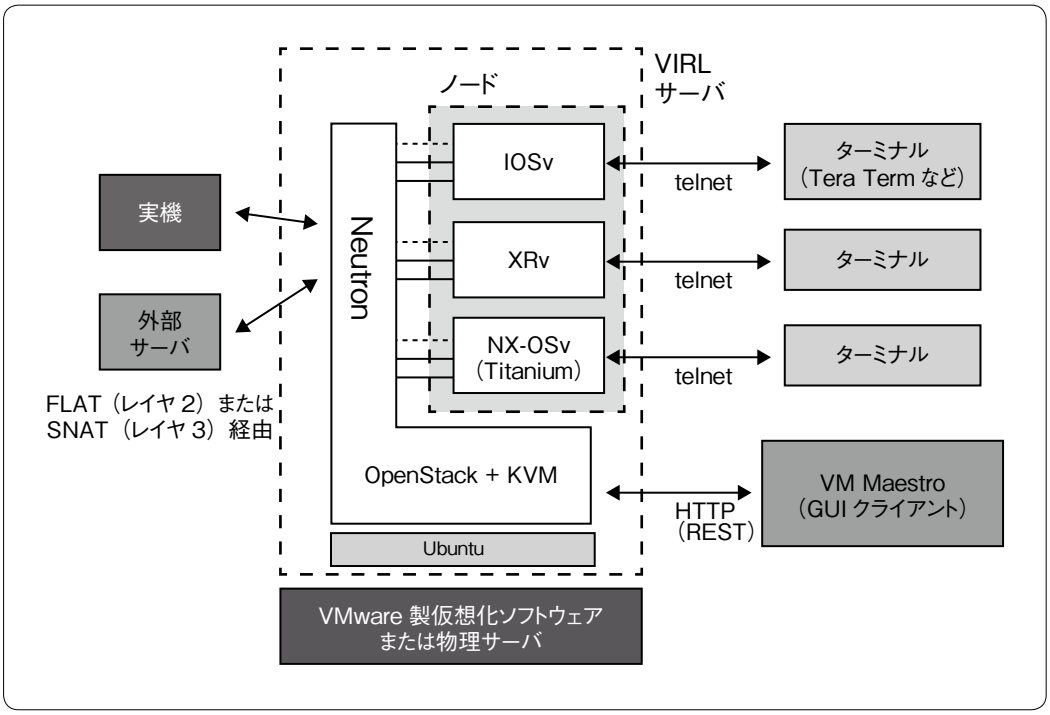
VIRLでシミュレーションを実行すると、ルーティングプロトコルなどは、実機とまったく同様に動作します。これは、ルーティングプロトコルの処理がハードウェアに依存せず、PIに閉じているためです。同じソースコードが用いられていることにより、実機に忠実なシミュレーションが可能になっています。

一方、ハードウェアに依存する機能は、実機と動作が異なることがあります。これは、実機のPDにあたるコードがVIRLのノードには含まれていなかったり、実機とは異なるコードで代替しているためです。

VIRLのアーキテクチャ

図4が、VIRLの内部アーキテクチャを簡略化したものです。VIRLは大きく3つのコンポーネントに分けられます。主役は、ネットワーク

▼図4　VIRLのアーキテクチャ概要



▼表1　VIRLのノードの名称（※がついたノードは単体でも販売しています）

実機の例	実機のOS	VIRLのノード
Ciscoルータ	IOS	IOSv
Catalystスイッチ	IOS	IOSvL2（提供予定）
ASR 9000	IOS XR	XRv※
ASR 1000	IOS XE	CSR1000v※
Nexus 7000	NX-OS	NX-OSv（Titanium）

機器のOSに対応するノードです。VIRLの中で動作しているノードは、先に解説したように実機とまったく同じものではありません。そこで、区別のために**表1**のような名称を用いています。

図4では、**図1**と同じトポロジをシミュレートしており、IOSv、XRv、NX-OSvの3種類のノードがそれぞれ動作しています。それぞれのノードは、外部からtelnetでアクセスすることで、実機のシリアルコンソールと同様に操作できます。

ノードの起動／停止や、ノード間の接続をシミュレートするのが、VIRLサーバです。OpenStackとKVM、Ubuntuを用いて実装されています。VIRLサーバの詳細は、次号の後編で解説しますが、VMware製の仮想化ソフトウェアの上でVMとして動かすか、物理サーバに直接インストールして使用します。VIRLサーバは仮想化にKVMを使用しているので、プリインストールされている以外のVMのイメージファイルを追加し、ノードとして利用することができます。

3番目のコンポーネントが、「VM Maestro」です。**図3**で先にスクリーンショットが出てきていましたが、各ノードをGUIを用いて配置し、結線できます。VIRLサーバに対してノードの

起動／停止を指示したり、各ノードの設定ファイルの管理も行います。VM Maestroは、VIRLサーバとは別のコンピュータ(Windows PC、Mac)か、VIRLサーバのグラフィカルデスクトップ(Ubuntu Unity)上で動作します。

VIRLサーバは、FLATとSNATという名前のしくみによって、外部のネットワーク機器(実機)やサーバなどと接続できます。FLATはレイヤ2、SNATはレイヤ3(NAT：Network Address Translation)で動作します。VIRLのノードは実機とまったく同じではありませんが、FLATやSNATを用いて実機と組み合わせると、よりリアルな動作検証が可能になります。

ネットワーク界の「まかない飯」

VIRLのノードは、Ciscoが仮想ラボで提供しているトレーニングや認定試験と基本的に同じソフトウェアをベースにしています。単体の製品としても販売しているCSR1000vとXRvは、製品としての出荷前テストが行われていますが、NX-OSvの完成度は現時点ではそれほど高くはありません。また、Catalystスイッチに相当するレイヤ2機能を持ったIOSvL2は、Cisco社内でのトライアルが続いていますが、まだVIRLのノードとしては提供されていません。

ですので、筆者はVIRLというソフトウェアを、ネットワーク界の「まかない飯」だととらえています。お客様に公式に出すメニュー(＝実機)ほどはきっちりしてはいないものの、安価で美味しいこともある、といったところです。もともとの目的がトレーニングや認定試験ですので、実機の機能が必ずしもすべて動作するわけではありません。現時点では、日本語の情報提供がほとんどなく、困ったときのサポートも、英語のコミュニティだけです(ただし、VIRLのコミュニティは非常に活発です)。

あくまでもVIRLは、実網(Production Network)をシミュレートし試験などに用いるためのソフトウェアであり、実網に組み込んで実機を置き換えることは想定していません。この記事の冒頭で、VIRLをフライトシミュレータ、実機を旅客機に例えましたが、位置づけは同じで、フライトシミュレータをそのまま使って旅客機を操縦することはありません[注5]。

さらにVIRLでは、実機とは障害発生時の挙動が違う場合があります。たとえば、VIRLノードはほかのノードと論理的にはポイントツーポイントで直結していますが、実際にはOpenStack Neutronが介在しており、レイヤ1的には直結していません。このため、直結している実機同士では、片側の物理インターフェースがダウンすると、即座に対向もダウンしますが、VIRLではダウンしません。また、実機ではコントロールプレーンの二重化が可能ですが、VIRLのノードではまだサポートされていません。

次号では

VIRLを動かすためのポイントを解説します。個人ユーザ向けのVIRL Personal Editionのライセンス[注6]を購入したあと、実際にインストールしてシミュレーションを動作させるまでの過程を、筆者の実体験をもとに解説していきます。VIRLを稼働させる際のPC環境の留意点や、インストールの勘所、VIRLでスムーズに本格的なシミュレーションを実行する方法を、手順ごとに説明します。ゴールとしては、より本格的な利用方法の提案や、“VIRL Professional Edition”とでも言うべきCML(Cisco Modeling Labs)との違いを解説する予定です。もっと先に予習したい方は、Webページ(http://www.lansw-book.net/VIRL/)も参照ください。SD

注5) UAV(Unmanned Air Vehicle、無人航空機)では似たような遠隔操縦が使われています。

注6) 本稿執筆時点では、VIRL Personal Editionのライセンスは1年間で$199.99で、同時にCisco製のOSを15ノードまで動作させることができます。

短期集中連載

Bluemixでためしてみる IoT入門

実践、モノのインターネット

Writer 宮田 裕樹(みやた ゆうき) 日本アイ・ビー・エム㈱

前編 BluemixでIoTアプリを作ってみよう

BluemixはIBMが提供する、高速・手軽にアプリを開発・デプロイできるPaaSサービス。本短期連載ではそのBluemixを使って昨今話題の「IoT」アプリを開発する手法を紹介します。前編では、センサーから気温の情報を受け取って通知を行うアプリを作りながら、Bluemixの基本的な使い方を学びます。

クラウド、モバイル、ソーシャル、ビッグデータに続くITトレンドとして、IoTが注目を集めています。IoTは「Internet of Things」、つまり「モノのインターネット」のことで、家電や車などのさまざまな「モノ」をインターネットにつなぐことによって、人の役に立つ便利なシステムを作るための技術です。本連載で紹介するBluemixを利用すると、このIoTによるシステム開発が簡単に行えることを体感できます。本連載では、Bluemixを使ってどのようにIoTアプリを開発していくのか、その手順を説明していきます。

Bluemixとは

みなさんが何かプログラムを作って動かそうとするとき、準備することは何でしょうか。当然、プログラムを動かすためのハードウェアとその上で稼働するOSを導入・構成することが必要となります。IaaSのようなクラウド・サービスを利用すれば、準備された環境を利用できますが、それでも、JavaやRubyなどのランタイム環境の導入・構成が必要となります。さらにアプリによっては、データベースやメッセージングなどのミドルウェア、分析や運用監視ツールの導入・構成が必要になってくるかもしれません。そのような環境をわずか1分足らずで準備して提供してくれるプラットフォーム、それがBluemixです。

アプリを魅力的にするためには、常にユーザからのフィードバックを反映し、リリースを繰り返すことによって鮮度と品質を向上していく必要があります。またそのリリース・サイクルを短かくすることにより、鮮度の高い状態を維持していくことが重要となってきます。必要なときに必要なサービスをつないで、すぐ動かしてみることができるBluemixは、そのようなアプリを提供するためにとても適した環境であると言えます。

BluemixはCloud FoundryベースのオープンスタンダードなPaaSサービスです。ランタイムとしては、Java、JavaScript、Go、PHP、Python、Ruby on Rails、Ruby Sinatraといった言語環境をサポートしており、新たに追加することもできます。そこから呼び出せるサービスとしては、PostgreSQL、MongoDBといったデータベースをはじめ、Webアプリ関連、モバイル関連、ビジネス分析関連など実に60種類以上のメニューが用意されています(**図1**)。ユニークなものとしては、アメリカのクイズ番

組で最も多くの賞金を獲得したことで一躍有名になった人工知能「Watson」もサービスとして呼び出せるようになっています。

今回紹介するIoTもBluemixの1サービスとして提供されており、これを利用することで簡単かつスピーディーにIoTアプリが開発できるようになっています。

Bluemixを支える通信プロトコル

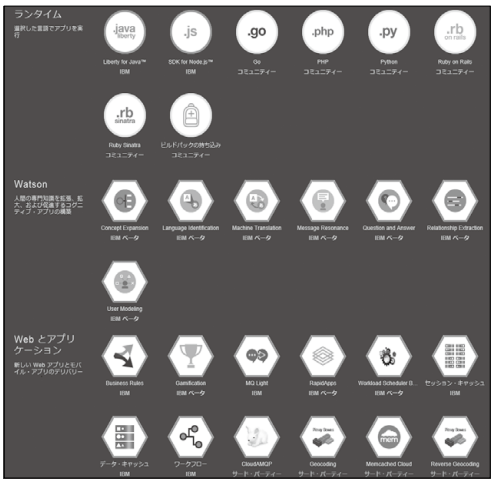
MQTT

IoTでは、さまざまなデバイスが発するデータをリアルタイムで受信し、利用者に逐次送信する必要があります。不安定なネットワーク環境にある膨大な数のデバイスとも通信できるような軽量なプロトコルが必要となります。このような要件を満たし、IoTに最適化されるように開発された軽量通信プロトコルがMQTT（Message Queuing Telemetry Transport）です。

MQTTは、OASIS[注1]で標準化が進められているオープンな通信プロトコルです。メッセージ・ヘッダが最小化されているため（最低限必要なヘッダは2バイト）、HTTPと比べ10倍のスループットを実現し、バッテリー消費量を1/10以下に抑えられると言われています。このことから、リソースの少ないリモートのセンサーやデバイスでも効率的に通信を行うことができるのです。

またこのプロトコルは、パブリッシュ／サブスクライブ型のメッセージング・モデルで、MQTTサーバ上のトピックというオブジェクトを介して通信を行います。送信側（パブリッシャー）は送信相手を意識することなくデータ送信を行い、受信側（サブスクライバー）は受信したいトピックを購読（サブスクライブ）登録することにより、必要なデータを受信できるようになっています。

▼図1　Bluemixのカタログ画面（一部）



IoT Foundation

IoT Foundationは、MQTTサーバの機能をクラウド上で提供するサービスで、Bluemixから簡単に利用できるようになっています。IoT Foundationでは、Raspberry Pi、TI SensorTag/BeagleBone、ARM mbedなどさまざまなデバイスとの接続方法とサンプルプログラムをレシピとして提供しています。また、デバイスをシミュレートするIoTセンサーも提供しており、実際のデバイスがなくてもIoTアプリの動作を試してみることができます（**図2**）。

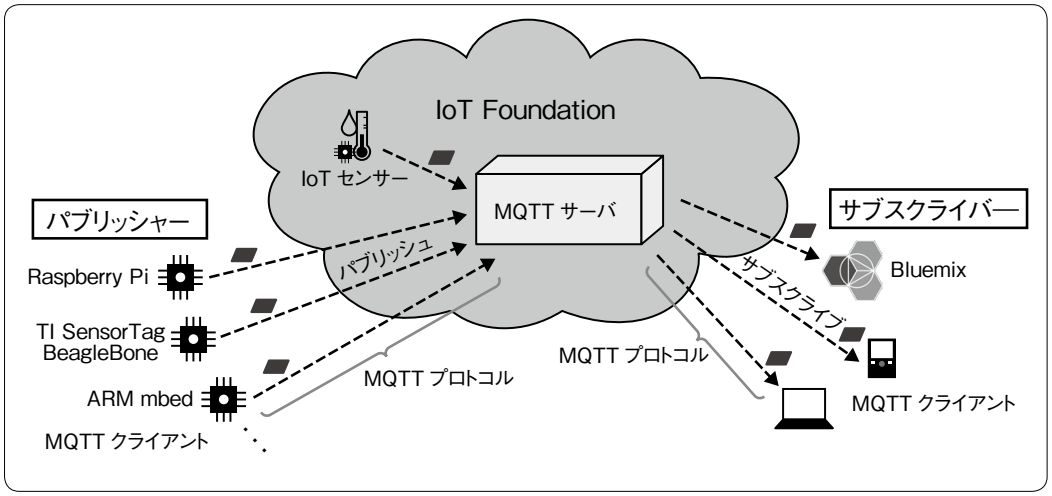
今回はまずIoTセンサーを使って、IoTアプリ開発を体感してみましょう。なお、実際のデバイスへの接続については、後編で紹介する予定です。

Bluemixを始める

では、さっそくBluemixを始めてみましょう。Bluemixを試しに使う方法として、30日間無償のフリートライアルがあります。まずは次の手順にしたがってログイン用のIBM IDを登録してみましょう。

注1） XMLに関連する技術の普及・促進活動を行う非営利団体。

▼図2　IoT Foundation



▼図3　Bluemix初期画面

①BluemixのURL[注2]にアクセスし、右上の「登録」ボタンをクリック(**図3**)

②次の登録画面で必要事項を入力(ここでのメールアドレスが「IBM ID」となる)。入力し終わったら「Submit」ボタンをクリック、画面に「Registration Complete」と表示されれば申請完了

③Bluemixから登録アドレスにメールが送られてくる。そのメールの「Click here to complete your registration」というリンクをクリックし、登録完了

注2)　http://www.bluemix.net

IoTアプリを作ってみる

Bluemixでのアプリ作成は、言語環境であるランタイムを選び、利用するサービスを選んでバインドするというのが基本的な流れになります。ここでは、より簡単に開発を行うためにボイラープレートを利用することにします。ボイラープレートは英語で「鋳型(いがた)」を意味しており、ある目的のアプリを作成する際に必要なランタイムとサービスをあらかじめ鋳型のように組み合わせて提供しています(**図4**)。

IoT用にもボイラープレートが用意されており、ランタイムとしてNode.js、サービスとしてデータ保管用のCloudantがあらかじめ組み合わされています。Bluemixのカタログから「Internet of Thingsのボイラープレート」[注3]を選択し作成を指示すると、1分あまりで完了します。このNode.js上にはビジュアルなIoTフロー・エディタであるNode-REDが配置され動いています。

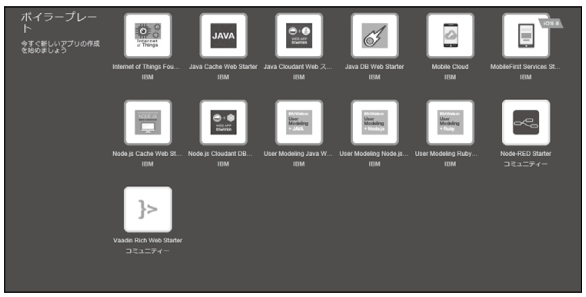
注3)　原稿執筆時点(2015年1月)、Bluemixは地域によって提供しているサービスに差があり、IoTボイラープレートは「米国南部」でのみ提供されています。地域が「英国」で登録されてしまった場合には、まず画面右上の地域表示において地域を追加し、「米国南部」に切り替える必要があります。

Bluemixダッシュボード上に作成されたIoTアプリのアイコン(**図5**)の「URLを開く」ボタンをクリックすると、Node-REDの初期画面が開きます。以降このNode-REDを使って、IoTアプリを作成する手順を説明していきます。

Node-REDでビジュアルに開発

Node-REDはパレットの部品をキャンバスにドラッグ&ドロップで配置し、部品同士をワイヤーでつなぐことでアプリを組み立てることのできるビジュアルなフローエディタです(**図6**)。Bluemixで提供されるNode-REDはIoT用にカスタマイズされており、IoT Foundationと簡単に連携ができるようになっています。

▼図4　Bluemixのボイラープレート



では、順を追ってIoTアプリを組み立てていきましょう。

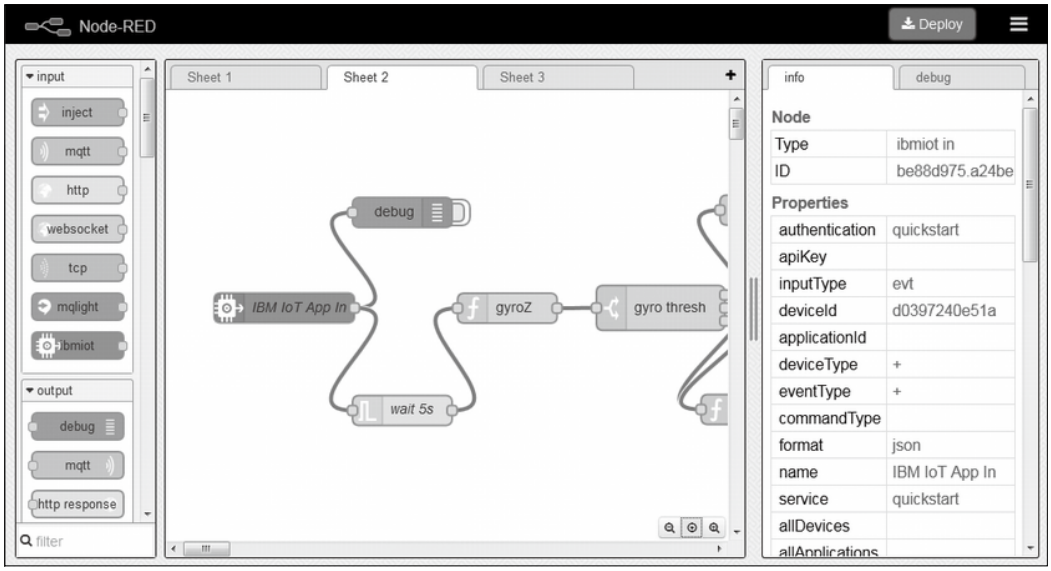
❶IoTセンサーにアクセス

今回はIoTのデバイスとして、温湿度センサーをシミュレートするIoTセンサーを利用します。https://quickstart.internetofthings.ibmcloud.com/iotsensor/にアクセスしてみてください。

IoTセンサーは、気温・湿度・物体温度のセンサーをシミュレートし、データをIoT Foundationにパブリッシュしています(**図7**)。

▼図5　ボイラープレートから作成したIoTアプリのアイコン

▼図6　Node-REDの画面



❷IoTセンサーからデータを取得

最初にNode-REDにアクセスすると、キャンバス内にサンプルのフローが表示されますが、これは削除して1から作成してみましょう(部品を選択しBackSpaceで削除できます)。

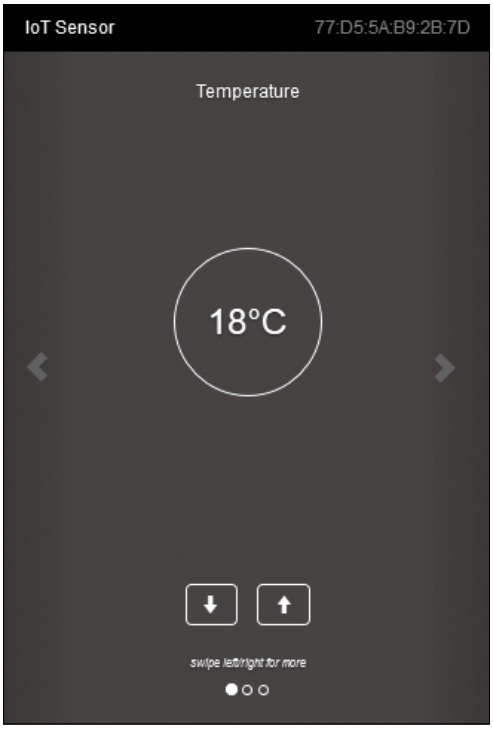
IoT Foundationからの入力はibmiotコンポーネントによって実行します。Node-REDの画面、左のパレットのinputカテゴリからibmiotを選択し、キャンバスにドラッグ&ドロップします。これをダブルクリックし、**図8**のように入力します。「Device Id」には、IoTセンサーの右上に表示されているMACアドレスから、コロンを抜いて英字を小文字に置き換えた文字列です。MACアドレスが77:D5:5A:B9:2B:7Dの場合は、77d55ab92b7dというふうになります。

❸IoTセンサーからのデータを確認

IoTセンサーからのデータを正しくサブスクライブできているかどうか確認してみましょう。これには、debugコンポーネントを利用します。パレットのoutputカテゴリからdebugをドラッグ&ドロップし、❷で作成した「IoTセンサー」ノードからワイヤリング(ドラッグ&ドロップでノードの端にあるポッチ同士をつなぐ)します(**図9**)。右上のDeployボタンをクリックするとBluemixにデプロイできます。右側フレームのdebugタブ内にデータが表示されれば成功です。また、キャンバス上のdebugノードの右のレバーをクリックするとログの有効/無効をスイッチできます。

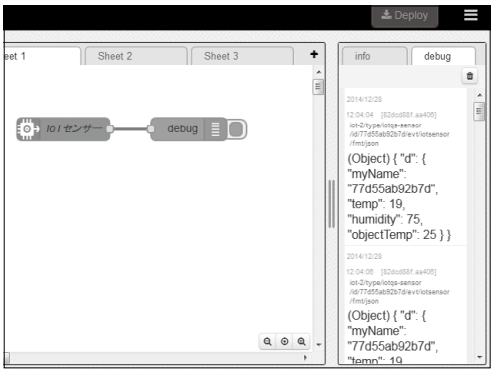
debugタブ内には**リスト1**のようなデータがログされているはずです。IoTセンサーの気温、湿度、物体温度はそれぞれ「temp」「humidity」「objectTemp」という変数名で渡されていることがわかります。ここで、IoTセンサーの気温(Temperature)画面で、温度を矢印ボタンで変化させると、ログ上のtempの値もそれに応じて変化することが確認できます。

▼図7　IoTセンサー



▼図8　「IoTセンサー」ノードの編集

▼図9　IoTセンサーの確認



▼リスト1　debugタブ内のデータ

```
iot-2/type/iotqs-sensor/id/77d55ab92b7d/evt/iotsensor/fmt/json
(Object) { "d": { "myName": "77d55ab92b7d", "temp": 19, "humidity": 75, "objectTemp": 25 }
```

❹データをサンプリング

IoTセンサーからのデータは約1秒に1回の割合で送信されますが、それだと少し多いのでここでは5秒に1回に制限することにします。

パレットのfunctionカテゴリのdelayをドラッグ＆ドロップし、「IoTセンサー」ノードからワイヤリング、キャンバス上のdelayノードをダブルクリックし、**図10**のように編集します。

❺気温データを取得

IoTセンサーからのデータは「d」というオブジェクトで渡されてきますので、ここから気温データ「temp」を取り出します。

パレットのfunctionカテゴリからfunctionをドラッグ＆ドロップし、「サンプリング」ノードからワイヤリングします。キャンバス上のfunctionノードをダブルクリックし、**図11・リスト2**のように編集します。

▼リスト2　「Function」欄の入力（気温データ取得）

```
return {payload:msg.payload.d.temp};
```

❻気温の条件分け

次に、取得した気温の条件分けを行います。ここでは、30℃を気温の条件分岐点として定義するものとします。

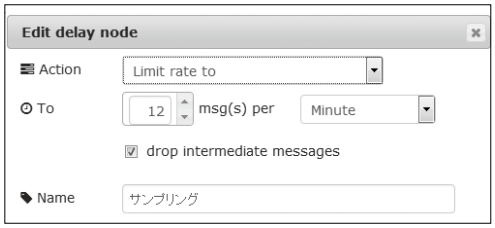
パレットのfunctionカテゴリのswitchをドラッグ＆ドロップし、❺で作成した「気温の取得」ノードからワイヤリングします。switchノードをダブルクリックし、**図12**のように編集します（条件の追加は左下の「+Add」ボタンでできます）。

❼高温の場合の処理

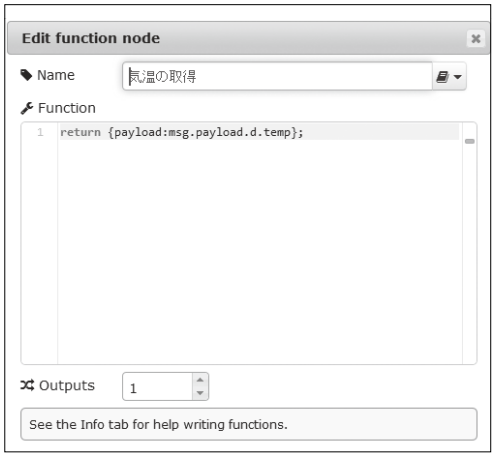
高温（30℃以上）の場合のメッセージ出力を定義します。

パレットのfunctionカテゴリからfunctionをドラッグ＆ドロップし、❻で作成した「気温の

▼図10　「サンプリング」ノードの編集



▼図11　「気温の取得」ノードの編集



▼図12　「気温の条件分け」ノードの編集

条件分け」ノード右端の出力の上側からワイヤリングします。functionノードをダブルクリックし、**図13・リスト3**のように編集します。

❽適温の場合の処理

同様にして、適温(30℃未満)の場合のメッセージ出力を定義します。

パレットのfunctionカテゴリから、functionをドラッグ&ドロップし、「気温の条件分け」ノードの出力の下側からワイヤリングします。functionノードをダブルクリックし、**図14・リスト4**のように編集します。

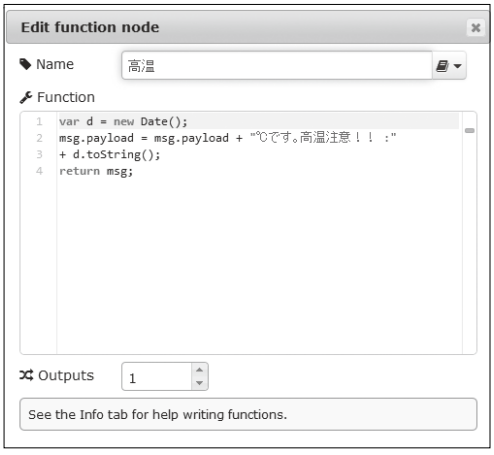
❾メッセージをdebugに出力する

最後にdebugノードを配置し、出力を確認してみましょう。

パレットのoutputカテゴリからdebugをドラッグ&ドロップし、「高温」ノードと「適温」ノードの両方からワイヤリングします。これで、フローは完成したので右上のdeployボタンをクリックしてBluemixにデプロイします。

IoTセンサーの気温を矢印で変化させると、30℃未満では「適温です。」、30℃以上では「高温注意！！」というメッセージが、約5秒おきにdebugタブに出力されることが確認できるはずです(**図15**)。

▼図13　「高温」ノードの編集



▼リスト3　「Function」欄の入力(高温の処理)

```
var d = new Date();
msg.payload = msg.payload + "℃です。高温注意！！ :"
+ d.toString();
return msg;
```

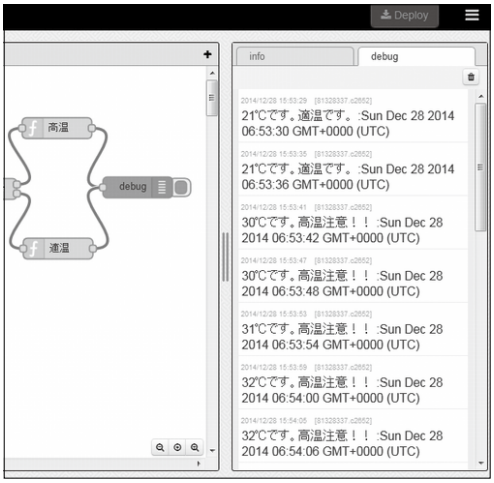
▼リスト4　「Function」欄の入力(適温の処理)

```
var d = new Date();
msg.payload = msg.payload + "℃です。適温です。 :"
+ d.toString();
return msg;
```

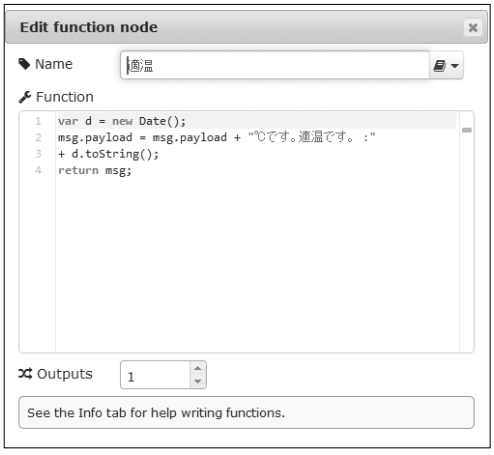
Twitterでつぶやいてみる

ここまでで、IoTセンサーの気温に基づいた

▼図15　debugへのメッセージ出力



▼図14　「適温」ノードの編集



メッセージをdebugに出力できるようになりました。ただ、これだと温度の変化を知るのに常にdebugメッセージをチェックしている必要があります。もっと確実に人に伝わるように工夫してみましょう。ここでは、高温になったときだけTwitterでつぶやくように拡張してみることにします。Node-REDでは、このような拡張も簡単に行うことができます。

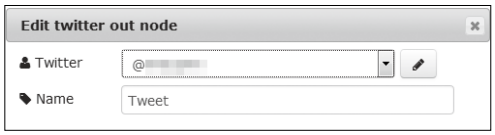
パレットのSocialカテゴリからtwitter(2つの内の入力がある方)をドラッグ&ドロップし、「高温」ノードからワイヤリングします。「Tweet」ノードをダブルクリックし、自分のTwitterアカウントを追加します(**図16**)。Twitter側から「Node-REDがあなたのアカウントを利用することを許可しますか？」と聞かれるので、ユーザ名とパスワードを入力し「連携アプリを認証」ボタンをクリックすれば認証完了です。

これで、フローは完成したので(**図17**)、ふたたびDeployボタンをクリックし、Bluemixにデプロイします。

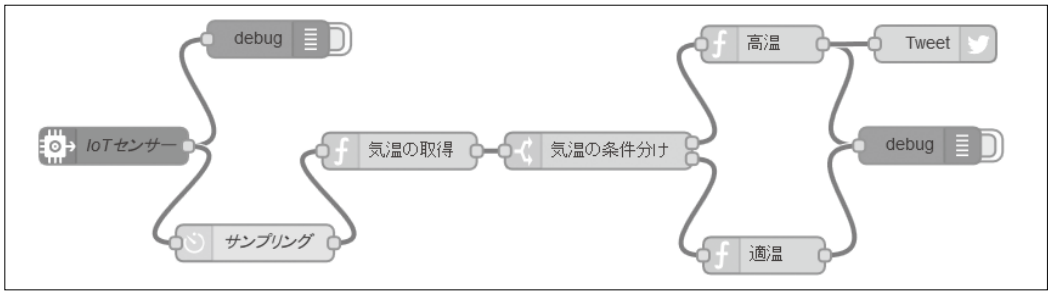
IoTセンサーの気温を変化させて、30℃以上にしたときにTwitterでつぶやくことを確認してみてください(**図18**)。

Node-REDはTwitterのほかに、E-mailやIRC(チャットシステム)へのインターフェースも持っています。また、Twilioというサービスを利用すると、ケータイにSMSを送信することもできますし、ibmpushというサービスを利用すると、スマホにプッシュ通知することもできます。ぜひいろいろなインターフェースを試してみてください。

▼図16　「Tweet」ノードの編集



▼図17　完成したIoTフロー



終わりに

今回は、IoTセンサーを使って、BluemixのIoTアプリ開発を体感してみました。Bluemixを利用すると、思った以上に簡単に、IoTアプリを開発できることが実感できたのではないでしょうか。

次回はIoT Foundationのレシピを使って、実際のデバイスにアクセスする方法を紹介したいと思います。SD

▼図18　Twitterへのつぶやき

コンテナ時代のUbuntu

Snappy Ubuntu Core

Ubuntu Japanese Team
柴田 充也(しばた みつや)

コンテナ時代のUbuntu：Snappy Ubuntu Core

2014年の12月にUbuntuの新しい機能として「Snappy Ubuntu Core」が発表されました[注1]。

この「Snappy Ubuntu Core(以下Snappy)」はクラウドとコンテナ型仮想化時代を見据えた、軽量かつ汎用的な新しいインストール済みイメージファイルです。標準でアップデート、ロールバック機能や個々のアプリケーションの隔離機能を備えているため、Webデベロッパがより簡単かつ安全に各種Webサービスをデプロイ、リリースできるようになります。

平たく言うとCoreOSやRed HatのAtomic HostのUbuntu版です。Snappyがこれらと異なるのは、必ずしもDocker/Rocketに依存しているわけではないということです。もちろんDockerコンテナのホストとして使うこともできますが、SnappyならRailsやNode.jsアプリを直接隔離環境にデプロイできますし、従来のUbuntuと同じように複数のサービスを1つのマシンで提供したり、ほかの構成管理ツールを用いた構築も行えます。もう1つ重要な点は「Snappyパッケージ」という新しいパッケージングシステムを採用していることです。従来のapt-getと同じような感覚で、Webサービスやコマンドなどをインストールできることを目指しています。

注1) http://www.markshuttleworth.com/archives/1434

なぜ新しいシステムが必要なのか

2014年はコンテナ型の仮想化が大きく注目された年でした。その中でもコンテナ技術を利用したDockerは1.0のリリースやRed Hatによるサポート、Microsoftとの提携など着実に実績を積み重ね、本誌2014年の11、12月号でも取り上げられるなど、今、最も熱いトピックの1つといっても過言ではないでしょう。

コンテナ型の仮想化は従来の「仮想マシン」とはしくみからして大きく異なります。Dockerはさらに「1コンテナ1プロセス」のように運用方法も特殊です。これに応じてDockerのコンテナイメージやホストOSとしてUbuntuを使う際に求められる機能も変わってきました。Snappyは、Dockerをはじめとするコンテナ型仮想化技術を利用したサービスを構築するうえで、より適切なLinuxディストリビューションの在り方を提供するために作られたのです。

図1はSnappyの模式図です。これを参考に、コンテナ型仮想化の時代に必要になる機能とSnappyによってどのように解決できるかを見ていきましょう。

1. よりシンプルなホストOS

Dockerはコンテナごとに環境を構築します。ですからホストOSそのものは、Dockerが動くだけの必要最低限の機能さえそろっていれば問題ありません。これを突き詰めて作られたのが

CoreOSです。従来のUbuntu Serverもサーバ OSとしてはそれなりにシンプルな部類ではありましたが、それでもDockerを動かすにあたって不要なサービスがいくつも動いている状態でした。そこで、より「Core」な部分だけを抜き出して再構築したのが「Snappy Ubuntu Core」です。

起動直後のイメージサイズは300MBと、CoreOSよりははるかに大きいものの、一般的なUbuntu Serverと比べると4割程度です。プロセスもsystemd[注2]とdbus、CPU周波数スケーリング、設定によってはsshdぐらいしか動いていません。

またDockerは、構築時にベースイメージからDockerfileに基づいて必要なソフトウェアや設定を追加していきます。そのため使われるベースイメージのサイズが小さいに越したことはありません。

2. 読み込み専用のファイルシステム

「Immutable Infrastructure」も、Dockerと併せて2014年に流行った単語です。サーバを構築後は不変(immutable)とし、変更が必要な場合は新規に構築して古いほうを廃棄(dispose)することで、常に最新の状態を反映した構築手順書を維持するという考え方です。DockerはDockerfileによってアプリケーションコンテナのデプロイを自動化できますし、ホストとなるUbuntuはcloud-initによってDockerの導入を自動化できます。このため従来から「新規に構築する」ということが簡単に実現できるようになっていました。

Snappyではこれをさらに一歩進めて、ファイルシステムのほとんどを読み込み専用にしています。**図1**でいうところのsystem-a、system-bは書き込み不可です。アプリをインストールするwritableパーティションのみ書き込み可能で、/varや/etcの一部もここにマウントされます。これによりユーザによる不用意な設定変更を阻止し、よりimmutableなシステムになりました。同じ読み込み専用のファイルシステムを採用しているCoreOSは、ディレクトリツリーを見直すことで/etc以下は書き込み可能になっているのに対して、Snappyは今のところ/etc以下のほとんども書き込み不可なため、いくつか不便な部分も存在します。

従来のパッケージ管理システムを使えないという大きなデメリットはありますが、後述のClick/Snappyパッケージを使うことでカバーしていく予定です。

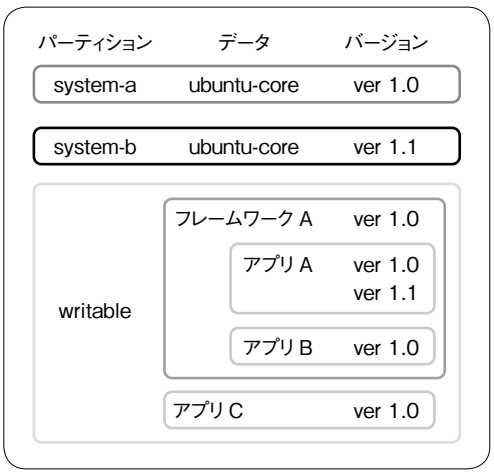
3. ロールバック可能なアップグレード機能

Snappyのイメージは最初から「2つのシステムパーティション」を持っています。インストール直後はsystem-aにのみイメージがインストールされていますが、システムアップグレード時はsystem-bの方に新しいイメージを展開し、system-bから再起動することでアップグレード完了になるというしくみです。

ある種の「ブルーグリーンデプロイメント」です。system-aには古いバージョンが残っていますので、簡単にシステムを差し戻すことができます。ただしシステムの切り替えにはブートローダーを利用する必要があるので切り替えロスは大きめです。その代わり従来のUbuntuでは手間

注2) Snappyは最初からsystemdを採用しています。

▼図1 Snappyの模式図



がかかった「アップグレード後のロールバック」が簡単に実現できるようになったことは大きな意味を持つでしょう。

同様にアプリ、フレームワークについても複数のバージョンを残しておくことができます。

4. コンテナ向けのパッケージングシステム

昨今隆盛を極めるWebフレームワークやその言語は、バージョンアップの頻度が高く、独自のパッケージングシステムを備えていることも特徴の1つです。単一の言語のエコシステムの中で完結するのであれば、言語そのものや必要なライブラリのビルドも含めて、言語独自のパッケージングシステムだけを使った方が便利なケースも増えてきました。

そこでSnappyでは、Ubuntu Touch用に開発していたClickパッケージを拡張して、アプリ同士の独立性を重視したパッケージングシステムを開発することになりました。Snappyアプリは**図1**でいうところの「アプリ」の中にインストールされます。アプリ同士の依存関係は存在しないので、Coreシステムにインストールされていないものはすべてパッケージの中に含める必要があります。このような制約を設けることで、パッケージの作成は2、3個のファイルを数行編集するだけという、非常にシンプルな手順になっています。

Click/Snappyアプリのもう1つの特徴は、AppArmorを用いた権限管理をアプリ単位で行えることです。具体的にはスマートフォンアプリの「パーミッション」をイメージしてください。あのような権限ルールをアプリのメタデータとして持つことができるのです。これにより、このサービスはネットワーク通信だけを許可する、このサービスはDockerの利用も許可するといった管理がより簡単に可能になります。

5. メタパッケージを拡張したフレームワーク

Click/Snappyアプリでは、必要なライブラリを全て同梱する必要があると述べました。しかし、そのままではDockerを使うアプリは個別にDockerをビルドする必要が出てきて手間がかかります。そこでSnappyでは、アプリレイヤーと読み込み専用のOSレイヤーの間に「フレームワーク」と呼ばれるレイヤーを用意しています。これは複数のアプリで共通して使われるコンポーネントを1つにまとめた特殊なアプリとして実装しています。

Snappy発表と同時に提供されたDockerフレームワークの場合、最新のDockerとその依存パッケージからなります。このDockerフレームワークをインストールすることで、各種アプリからDockerを使えるようになる、というわけです。Dockerのコンテナ内であれば従来のaptもそのまま使えるため、Debian由来の豊富なパッケージ資産も活用できます。さらに既存のDockerfileをアプリの起動スクリプトとしても利用できるのです。

そのほかにもサーバ管理者向けのCLIツール一式を用意したComfyフレームワークなども開発中です。

Snappyを体験しよう

2015年1月時点で、次のようにローカルシステム上だけでなく複数のクラウドサービス上でもSnappyのテスト版を実行できるようになっています。

- QEMU
- Vagrant/VirtualBox
- OVFに対応した仮想アプライアンス
- Microsoft Azure
- Google Compute Engine（GCE）
- Amazon Web Services（AWS）

いずれもCanonical／クラウドベンダが公開している公式イメージを使います。ツールごとのSnappyのインストール方法は、公式ドキュメント[注3]を参照してください。Snappyをコントロールするクライアント側マシンは、上記のサービ

注3）http://www.ubuntu.com/cloud/tools/snappy

スが使えるのであればUbuntu以外でもかまいません。

たとえば**図2**は、Vagrantを使ってVirtualBox上にSnappyインスタンスを立ち上げる手順です。

AWSでSnappyインスタンス起動

クラウドサービスを使う方法として、AWS上にSnappyインスタンスを立ち上げてみましょう。あらかじめAWSのアカウントを作成しアクセスキーなどを取得していること[注4]が前提となります。

公式ドキュメントではJava版ツールを使っていますので、ここではPython版awscliの手順を説明します。**図3**ではawscliをインストールしています。またSnappyではSSHアクセスをするためsshの鍵を作成し、awscliから参照できるようにしておきます。既存の鍵を流用してもかまいません。

SnappyのSSH機能を有効化するためcloud.cfgという名前で**リスト1**の内容のファイルを作っておきます。

最後にsnappyインスタンスを起動し、アドレスを取得したうえでsshでアクセスします(**図4**)。あとでownCloudを使うために、443番ポートにアクセスできるセキュリティグループも指定しておくと便利です。

Snappyの基本の“キ”

Snappyの基本コマンドを見ていきましょう。**図5**ではインストール直後の状態を表示しています。

2つ目のversionsにはubuntu-coreのAvailableに新しいバージョンが表示されています。そこで**図6**のようにubuntu-coreをアップデートしましょう。新しいイメージが、もう1つのシステムパーティションにインストールされ、再起動が要求

注4) http://docs.aws.amazon.com/cli/latest/userguide/cli-chap-getting-set-up.html

▼図2　VagrantによるSnappyの起動

```
$ mkdir snappy && cd $_
$ sudo apt-get install vagrant virtualbox
$ vagrant box add snappy http://cloud-images.ubuntu.com/
snappy/devel/core/current/devel-core-amd64-vagrant.box
$ vagrant init snappy
$ vagrant up
$ vagrant ssh
```

▼図3　awscliの設定

```
$ sudo apt-get install python-pip
$ sudo pip install awscli
$ aws configure
AWS Access Key ID [None]: (access key)
AWS Secret Access Key [None]: (secret access key)
Default region name [None]: ap-northeast-1
Default output format [None]: json
$ ssh-keygen -t rsa -f ~/.ssh/snappy-rsa
$ aws ec2 import-key-pair --key-name snappy-key
--public-key-material file://~/.ssh/snappy-rsa.pub
```

▼図4　インスタンスの起動

```
$ aws ec2 run-instances --image-id ami-84555c85 --key-name
snappy-key --instance-type m3.medium --user-data file://cloud.cfg
(出力されるInstanceIdを保存)
$ aws ec2 describe-instances --instance-ids InstanceId
(出力されるPublicDnsNameを保存)
$ ssh -i ~/.ssh/snappy-rsa ubuntu@PublicDnsName
```

▼リスト1　cloud.cfg

```
#cloud-config
    snappy:
        ssh_enabled: True
```

▼図5　Snappyの情報

・インストールされているコンポーネント一覧

```
$ snappy info
release: ubuntu-core/devel
frameworks:
apps:
```

・各コンポーネントのバージョンリスト

```
$ snappy versions
Part         Tag   Installed  Available  Fingerprint     Active
ubuntu-core  edge  141        142        7f068cb4fa876c  *
```

されます。再起動後にふたたびversionsを表示すると、更新されていることがわかります。なおsudo権限で「-a」付きで実行した場合、過去のバージョンも表示されます。

次にロールバックを試してみましょう(**図7**)。こちらもロールバック後に再起動を行う必要があります。アップデート、ロールバックは、起動パーティションを切り替えているだけです。よって起動前後でdfコマンドを実行してみると、それぞれ「/」にマウントしているデバイスが変わっていることがわかります。

もう1つの基本操作はフレームワークやアプリのインストールです(**図8**)。ちなみにsearchによる検索結果はインストール済みのフレームワークに依存します。dockerフレームワークをインストールするまでは、owncloudアプリは検索結果に表示されません。なおフレームワークやアプリはすべて、書き込み可能な/apps以下に保存されます。

このowncloudアプリは単純にdocker runを実行しているだけです。よって10分ほどするとdocker psに状態が表示されるようになります(**図9**)。無事にデプロイが完了したら、Webブラウザから「https://Snappyインスタンスのアドレス」にアクセスしてみましょう。ownCloudの初期画面が表示されることでしょう。docker runで指定しているイメージは、Docker Hubにあるkickinz1/big-owncloud[注5]です。またSSL証明書などはrun時に生成しています。

▼図6　アップデート

```
$ sudo snappy update ubuntu-core
(中略)
Reboot to use the new ubuntu-core.
$ sudo reboot
(再起動後)
$ sudo snappy versions -a
Part         Tag   Installed  Available  Fingerprint     Active
ubuntu-core  edge  142        -          18d8361edb919f  *
ubuntu-core  edge  141        -          7f068cb4fa876c  -
```

▼図7　ロールバック

```
$ sudo snappy rollback ubuntu-core
Rolling back ubuntu-core: (edge 142 18d8361edb919f -> ➡
edge 141 7f068cb4fa876c)
Reboot to use the new ubuntu-core.
$ sudo reboot
(再起動後)
$ sudo snappy versions -a
Part         Tag   Installed  Available  Fingerprint     Active
ubuntu-core  edge  141        142        7f068cb4fa876c  *
ubuntu-core  edge  142        -          18d8361edb919f  R
Reboot to use the new ubuntu-core.
```

▼図8　フレームワークやアプリの操作

・フレームワークとアプリの検索

```
$ snappy search docker
Part    Version    Description
docker  1.3.3.001  The docker app deployment mechanism
```

・フレームワークとアプリのインストール

```
$ sudo snappy install docker
Part          Tag         Installed        Available        Fingerprint       Active
docker        edge        1.3.2.004        -                788b0787b18b1c    *
$ docker --version
Docker version 1.3.3, build be8f9a2-dirty
$ sudo snappy install owncloud
owncloud  edge  7.0.3.008  -          81ebbbea41f48e  *
$ snappy info
release: ubuntu-core/devel
frameworks: docker
apps: owncloud
```

注5）https://registry.hub.docker.com/u/kickinz1/big-owncloud/

Snappyアプリの作り方

Snappyコマンドでインストールできるアプリは「snapp」とも呼ばれ、Ubuntu上で簡単に作成できます[注6]。そこでここではowncloudアプリのコードを紐解きつつ、dockerフレームワークを利用したSnappyアプリの作り方について紹介しましょう。

Snappyアプリの中身

あらかじめ**図10**のようにowncloudアプリのソースコードをダウンロードしておいてください。ディレクトリ構成は**図11**のようになっています。このうち必要なのはpackage-dirのほうだけです。

Snappyでは任意のディレクトリ名(今回の例ではpackage-dir)の中にmetaディレクトリを作成し、package.yamlとreadme.mdを作成します。最低でも必要なのファイルはこの2つです。

それに加えてowncloud.apparmorはAppArmorによるポリシー管理を、owncloud.svgではアプリのアイコンを設定します。これらはともにpackage.yamlから参照することになります。

metaディレクトリの外については、任意のファイルを置けます。これらは一括して「/apps/パッケージ名/バージョン/」以下に「そのまま」インストールされますので、パッケージ作成者にとってわかりやすい配置になっていればよいでしょう。

readme.mdの内容は**リスト2**に掲載します。1行目の先頭にはパッケージ名を記載します。2行目以降はパッケージの概要説明です。これらは将来的にSnappyパッケージの情報表示(apt-cache show相当の機能)に使われます。

リスト3はpackage.yamlの内容です。nameからvendorまではその名のとおりの内容で、nameとversionだけが必須です。frameworksは依存するフレームワークをカンマ区切りで指定します。owncloudアプリはdockerコマンドを使うため、dockerフレームワークに依存しているというわけです。portsでは使用するポート番号を指定できます。これにより使用済のポートと衝突

注6) http://www.ubuntu.com/cloud/tools/snappy#snappy-apps

▼図9　owncloud環境の確認

```
$ docker images
REPOSITORY              TAG                 IMAGE ID            CREATED             ➥
VIRTUAL SIZE
kickinz1/big-owncloud   latest              44d3ad1f46e2        8 weeks ago         1.019 GB
$ docker ps
CONTAINER ID        IMAGE                              COMMAND             ➥
CREATED              STATUS              PORTS                    NAMES
d989b28e81e4        kickinz1/big-owncloud:latest   "/sbin/my_init"     About a minute ➥
ago   Up About a minute   0.0.0.0:443->443/tcp   owncloud-demo
```

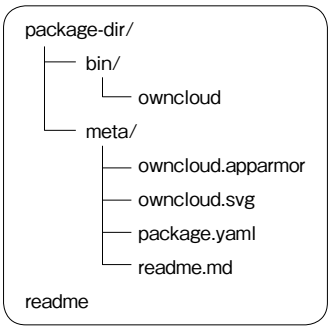
▼図10　ソースコードの取得

```
$ sudo apt-get install bzr
$ bzr branch lp:~snappy-dev/snappy-hub/owncloud
```

▼リスト2　readme.md

```
Owncloud 7.0.3.001
This is a owncloud snappy package.
Requires a docker framework at least version 1.3
Can take some time to show up as there is many layers to ➥
download
```

▼図11　ソースコードのディレクトリ構成



するパッケージはインストールできないようになっています。

servicesがこのメタデータの肝となります。Snappyは最初からsystemdを採用しています。Snappyの中でサービスを立ち上げるためには、systemdのサービスファイルを作成する必要があります。このservicesで、サービス名、start時に実行するファイル名などを設定することで、アプリインストール時にSnappyが自動的にサービスファイルを生成してくれるのです。今回は起動時に「bin/owncloud」を実行するだけのサービスファイルが作られます。start以外にもstopやstop-timeoutによって停止時に実行するコマンドも指定できます。

この例には存在しませんが、servicesと対をなす存在としてbinariesがあります。これはそのアプリがサービスではなくコマンドを提供する場合に、インストールするべきコマンド名を指定するデータとなります。

integrationではAppArmorの設定ファイルを指定します。SnappyはアプリをシステムからAppArmorを利用しています。ここではそのAppArmorのプロファイルを生成するためのマニフェストファイル(meta/owncloud.apparmor)を指定しているのです。詳しい実装はWiki[注7]を参照してください。ネットワーク通信を行うだけのアプリであれば最初から許可されているため、integration以降の3行とapparmorファイルは必要ありません。owncloudアプリはDockerを使うためにこのような設定になっています。同様にDockerを使うアプリを作る場合は、マニフェストの中身は同じでアプリ名(owncloud)の部分だけ差し替えるとよいでしょう。

もう1つ「bin/owncloud」については内容が長いため省きます。簡単に言うと「docker ps -a」を行いコンテナが存在しなければ「docker run」を、存在すれば「docker start」を行うスクリプトです。

Snappyアプリのビルド

このソースコードをSnappyアプリ形式にビルドします。あらかじめクライアント側に**図12**のようにツールをインストールしておいてください。パッケージのビルド方法とインストール方法は**図13**です。パッケージの作成手順がとても

注7) https://Wiki.ubuntu.com/SecurityTeam/Specifications/SnappyConfinement

▼リスト3　package.yaml

```
name: owncloud
version: 7.0.3.008
icon: meta/owncloud.svg
architecture: amd64
vendor: Pierre-Andre MOREY <pierre-andre.morey@canonical.com>
frameworks: docker-1.3
ports:
  required: 443
services:
  - name: owncloud
    description: owncloud for snappy's docker framework
    start: bin/owncloud
integration:
  owncloud:
    apparmor: meta/owncloud.apparmor
```

▼図12　ビルドツールのインストール

```
$ sudo add-apt-repository ppa:snappy-dev/beta
$ sudo apt-get update
$ sudo apt-get install snappy-tools
```

▼図13　ビルドとインストール

```
$ snappy build package-dir
$ snappy-remote --url=ssh://IPアドレス:ポート
install ./owncloud_7.0.3.008_amd64.snap
```

シンプルだということがわかるでしょう。

ここではSnappyアプリを直接Snappyが動いているサーバにインストールしています。作成したパッケージファイルは、Ubuntu TouchのClickアプリと同様に、アプリストア[注8]にアップロードできます。アップロード時に必要なのはUbuntu Oneアカウントだけです。アプリストアに登録することで、「snappysearch」コマンドでも自作のアプリが表示されるようになります。

Snappyアプリ管理UI：WebDM

2015年の1月にSnappyアプリを管理するためのWeb UIである「WebDM」がリリースされました(**図14**)。これはSnappyがARMボードにも対応した際に、作成されたフレームワークで、Go言語で作られていることもあってか、ARMボード上でもそこそこ軽快に動作します。

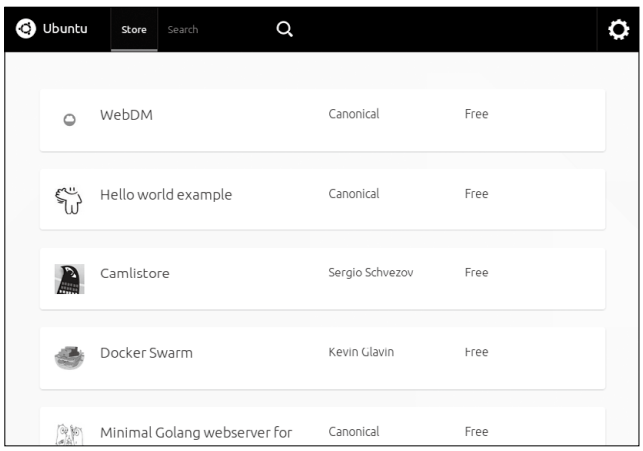
WebDMではsnappyコマンドでいうところの検索やインストール・アンインストールに対応しています。よってWebDMさえ入っていればアプリの追加や削除はWebブラウザから行えるということになります。

WebDMは次のコマンドで導入できます。

```
$ sudo snappy install webdm
```

注8) https://myapps.developer.ubuntu.com/dev/apps/new/

▼図14　WebDMのインターフェース



あとはWebブラウザから次のアドレスにアクセスしてください。

```
http://Snappyのホスト名:4200/
```

AWS EC2などを使っている場合は、セキュリティグループのポリシーで4200番ポートを空けておく必要があります。インスタンス作成時に**リスト4**のようなcloud.cfgファイルを用意しておけば、最初からWebDMサービスが立ち上がった状態となりますので、ログインすることなくアプリをインストールできます。

アプリケーションのインストールは、画面上部の「Store」から必要なパッケージを選択し、「Install」を選ぶだけです。通常のsnappyコマンドと同様にdockerフレームワークをインストールすれば、Storeに表示されるアプリも増えます。

まとめ

Snappyは昨年12月に公開されたばかりの、まだアルファ段階のプロジェクトです。既存のコンテナ向けツールに比べたら完成度はまだまだではあるものの、AppArmorやClickといったUbuntuならではのツールを駆使することで「Ubuntuならでは」の使いやすさを実現させようという意気込みは感じられるのではないでしょうか。

今後、2016年にリリース予定の次のLTSである16.04に向けて、機能の追加・安定化がどんどん行われていく予定です。作りがとてもシンプルな今のうちにUbuntuの新しい仲間をぜひ、体験してみてください。SD

▼リスト4　WebDMをインストールする

```
#cloud-config
  snappy:
    ssh_enabled: True
    packages:
      - webdm
```

新連載

Android Wear アプリ開発入門

～より生活に密着するスマートデバイスの世界～

第1回 Android Wearアプリ開発を初体験！

神原 健一（かんばら けんいち） Web http://blog.iplatform.org Twitter @korodroid

iplatform.orgにて情報発信するかたわら、「セカイフォン」などを開発。Droidconなどでのカンファレンス講演、MWC/CES/IFAでのプロダクト展示、執筆などの活動も実施。NTTソフトウェア株式会社テクニカルプロフェッショナル。現在はAndroid以外のモバイルOSにも取り組み、公私にわたってモバイルアプリの世界に没頭中。

環境が整ってきたAndroid Wear

Android Wearとは、スマートウォッチなどのウェアラブルデバイスを対象としたAndroidプラットフォームです。Google I/O 2014（Googleの開発者向けカンファレンス）にて、Android Wear（以降「Wear」と表記）搭載のスマートウォッチ実機、および、アプリケーション（以降「アプリ」と表記）を開発するためのSDK（Software Development Kit）が発表されてから、8ヵ月あまりが経過しました。同発表以降、新しい実機の発売（**図1**）、OSのバージョンアップ（**表1**）が行われるとともに、Wearに対応したアプリの数も急速に増えてきました。

本連載では、Wear向けアプリ開発入門をテーマに、Wearの各種機能紹介だけでなく、実際のコード例も解説していきます。連載を通じてWearアプリ開発のポイントについて、ひと通り習得できるはずです。ぜひ実際に手を動かしながら毎回読み進めていただけると幸いです。初回となる今回は、Wearアプリの開発環境構築から簡単なアプリの開発・実行までをご説明します。

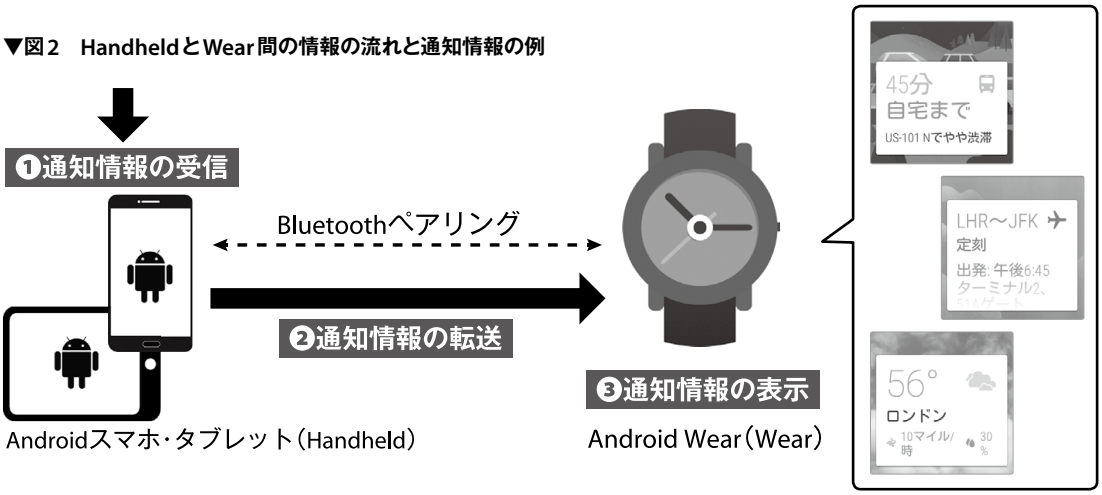
アプリ開発に入る前に、Wearの特徴を復習しておきましょう。なかでも最も代表的なのは「通知」です。自分が持っているスマホやタブレット（以降「Handheld」と総称します）とBluetoothペアリングしておけば、Handheldをポケットやカバンから取り出すことなく、交通情報やメールの受信などの通知情報を確認できます（**図2**）。このようにWearアプリを考える際には、**ユーザが必要とする情報を適切なタ**

▼図1 Android Wear搭載スマートウォッチ

▼表1 Android WearのOSの変遷

バージョン	主な特徴	リリース時期
Android 4.4W	Android Wearリリース時の最初のバージョン	2014年 6月
Android 4.4W.2	Wear単体での音楽再生、時間表示の改善、バッテリー省電力化ほか	2014年10月
Android 5.0	Lollipopベースへの移行、映画館・太陽光モードのサポート、Watch Face APIの公式提供ほか	2014年12月

▼図2　HandheldとWear間の情報の流れと通知情報の例



イミングで自動的に提供することが重要です。そのほかにWear上での音声入力や、WearとHandheld間でのデータ送受信といったことも可能です。Handheldアプリとは考慮すべきポイントが異なるので、Wearならではの注意点についても適宜紹介していきます。

Android Wearアプリの開発環境構築

アプリ開発に必要なもの

それでは開発環境の構築を行いましょう。Wearアプリ開発には、**表2**に示すものが必要となります。

JDKは、Androidに限らずJavaを用いたアプリ開発に必要です。Android Studio（以降「Studio」と表記）はIntelliJ IDEAをベースとして作られた、Androidアプリの統合開発環境です。ビルドツールとしてGradle[注1]がサポートされていたり、コーディングを楽にしてくれる便利な機能が数多く提供されているなど、開発者にたいへん人気があります。ver.1.0がリリースされ、Wearのみならず Handheld向けアプリ開発を含め、Eclipseに代わって、Androidの公式開発環境となっています。Android SDKは、Androidアプリ開発に必須のツールです。Wearエミュレータもしくは実機は、開発したWearアプリの動作確認に用います。Wearと連携させるためのHandheld実機も準備しておきましょう。

続いて、それぞれのセットアップ手順を紹介します。

▶JDKの導入

【Windowsの場合】
http://www.oracle.com/technetwork/java/javase/downloads/

【Macの場合】
http://support.apple.com/kb/DL1572

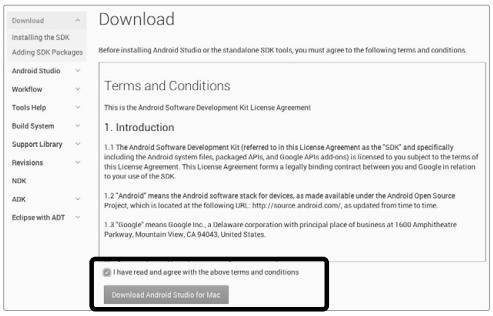
上記Webサイトからインストール用パッケージをダウンロードします。同ファイルをダブルクリックするなどしてインストールしてください。

▼表2　Wearアプリ開発で必要なもの

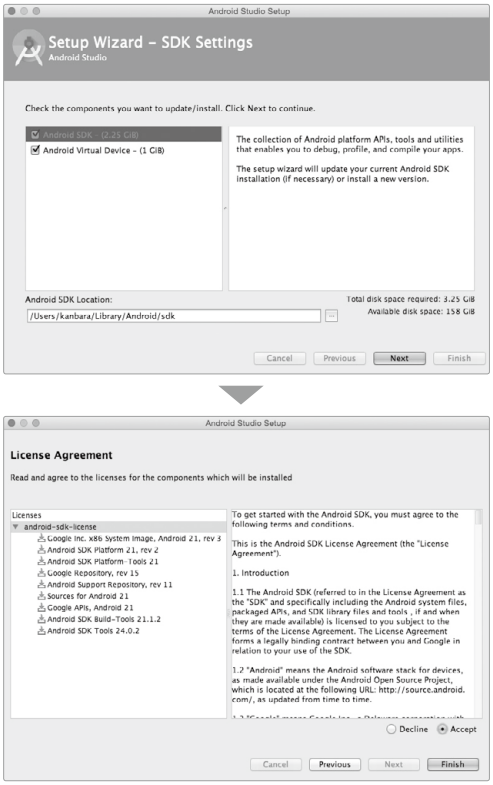
	名称	概要
1	JDK	Javaの開発キット
2	Android Studio	Android向け統合開発環境
3	Android SDK	Androidの開発キット
4	Wearエミュレータ	Wear用System Image
5	［オプション］ Wear実機	Wear搭載スマートウォッチ
6	Handheld実機	Androidスマホ・タブレット （Android 4.3以上）

注1）拡張可能なドメイン特化言語（DSL）でビルドスクリプトを記述できる柔軟性の高いビルドシステム。

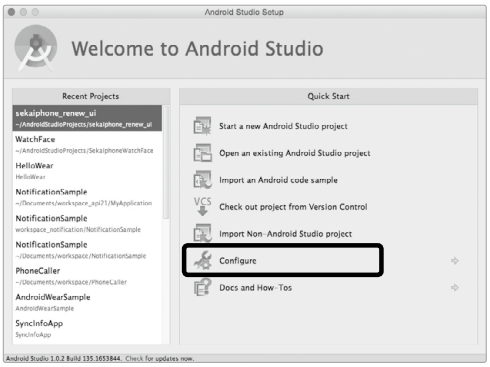
▼図3　Studioのダウンロード



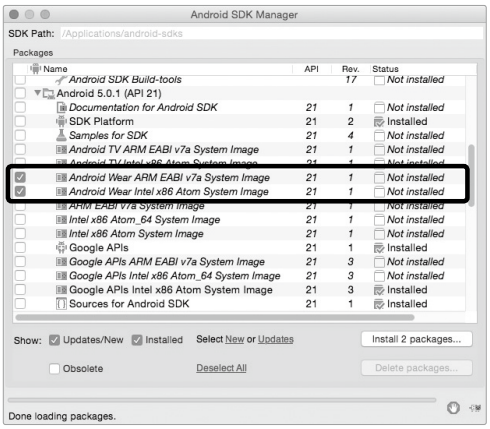
▼図4　SDKやツールのセットアップウィザード



▼図5　Studioのトップ画面からSDK Managerを起動



▼図6　Wear用System Imageのインストール



▶Android Studio&Android SDKの導入

http://developer.android.com/sdk/

上記Webサイトにアクセスして Studio を入手します。Downloadボタンを押すと利用規約同意画面が表示されます(**図3**)。同意する場合はチェックボックスにチェックをつけ、ふたたびDownloadボタンをクリックすると、インストール用パッケージのダウンロードが行われます。インストールと起動はOS別に次のように行ってください。

【Windowsの場合】
ダウンロードしたファイルをダブルクリックしてインストールを開始。ウィザードに沿って進めて完了させた後、Studio を起動

【Macの場合】
ダウンロードしたファイルをダブルクリック後、「Android Studio」を「Applications」にドラッグ&ドロップ。その後、Android Studio.app を起動

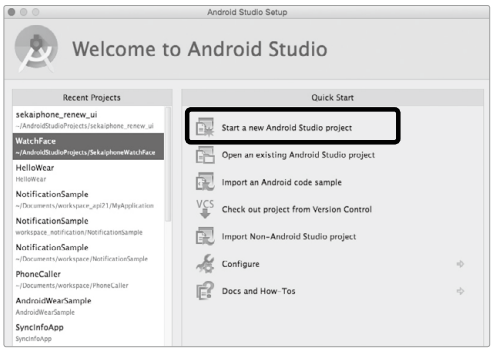
起動すると、必要最低限のSDKやツールをインストールするためのウィザードが表示されます。**図4**を参考に、セットアップを進めてください。

Wearアプリを開発するには、Wearエミュレータを動作させるためのSystem Image も必

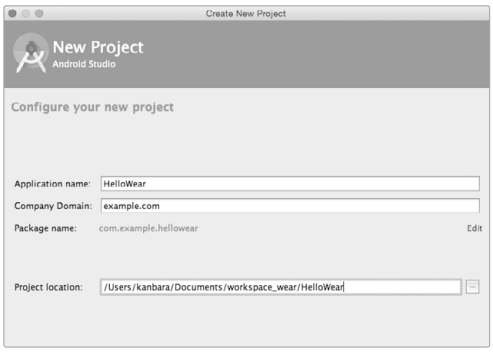
要です。Studioのトップ画面で[Configure]→[SDK Manager]を選択します(**図5**)。SDK Managerが起動しますので、Wear用System Image(今回は5.0系ARM版、Intel版の計2つ)を選択し、[Install]ボタンを押します(**図6**)。次の画面で規約に同意し、セットアップを進めます。以上でインストール作業は完了です。

アプリのプロジェクト作成

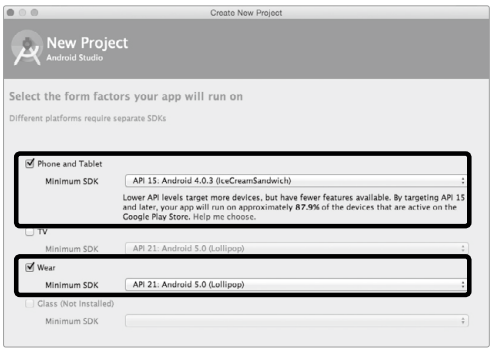
Studioを起動し、次の画面のように「Start a new Android Studio project」を選択します。



続いて、アプリ名、パッケージ名などを次の画面のように入力します。

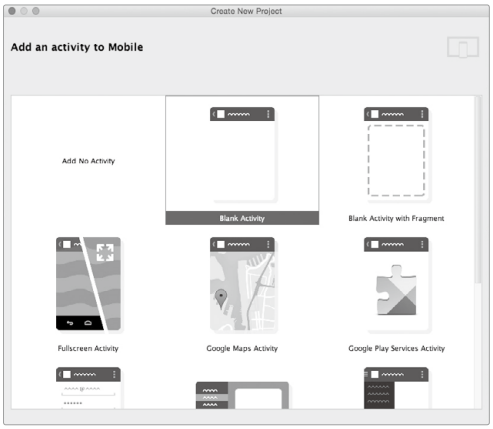


次に、アプリがサポートするプラットフォームを選択します。今回はHandheldとWearを対象とするため、次の画面のように「Phone and Tablet」と「Wear」にチェックを付けます。なお、これらはStudio上でモジュールという単位で管理されます。

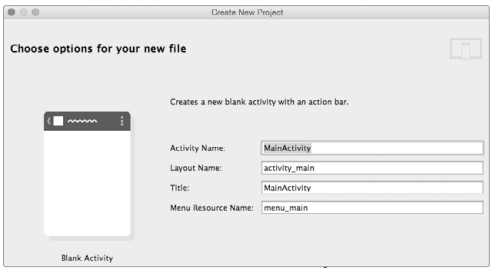


Handheld用モジュールの設定

Handheld向けに追加するActivityを選択します。今回は空のアクティビティを指定するため、次の画面のように「Blank Activity」を選択します。



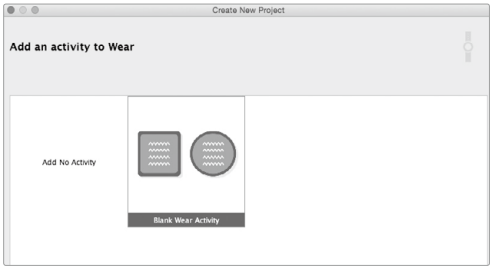
続いて、次の画面のようにアクティビティ名やレイアウトXMLファイル名などを設定します。



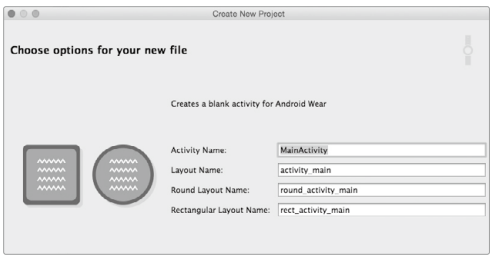
Wear用モジュールの設定

Wear向けに追加するActivityを選択します。

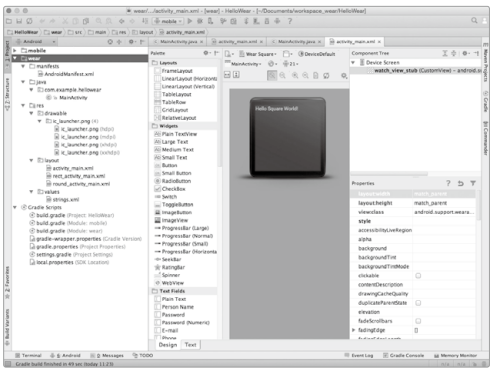
今回は空のアクティビティを指定するため次の画面のように「Blank Wear Activity」を選択します。



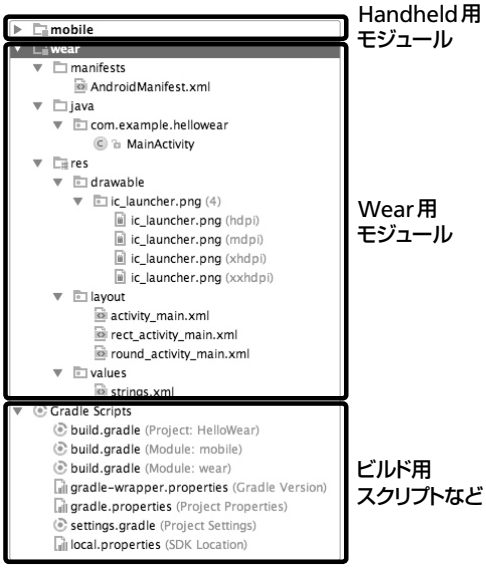
続いて、次の画面のようにアクティビティ名やレイアウトXMLファイル名などを設定します。



しばらくするとプロジェクト作成が完了し、次のような画面が表示されるはずです。この画面で、ソースコードやレイアウト編集などを行うことができます。

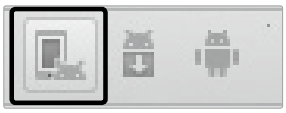


今回作成したプロジェクトは、次の画面のとおりHandheld用モジュール＋Wear用モジュール＋ビルド用スクリプトなどから構成されます。Wear用モジュール内のファイルを編集することで、Wearアプリのビジネスロジックや画面を改造することができます。



仮想デバイスの作成

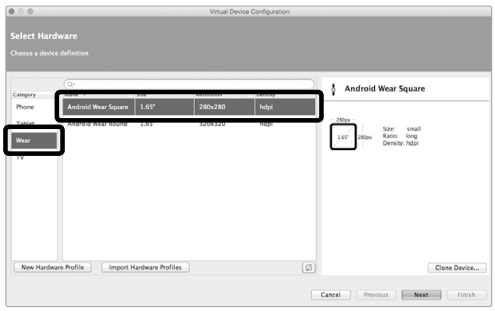
Wearアプリでも Handheldアプリの場合と同様に、エミュレータで動作確認を行うことができます。そのためには仮想デバイス(AVD)をあらかじめ作成する必要があります。Studioのツールバーから次の画面アイコンを選択してください。



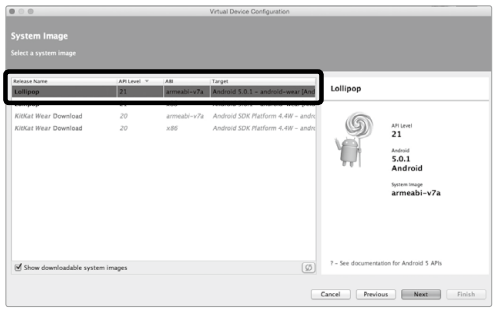
すると、仮想デバイスの作成画面が次のように表示されます。

中央のボタンを押した後、次の画面で「Android Wear Square」を選択し、[Next]ボタンを押します[注2]。これは矩形の画面を意味しています。

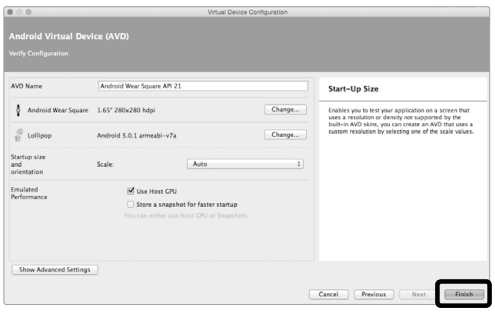
注2)「Android Wear Round」を選択すれば、円形の画面を持つAVDを作成できます。



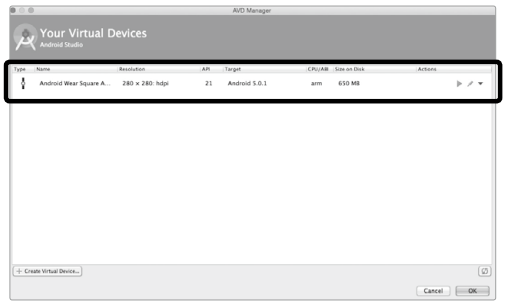
次の画面で「Lollipop/armeabi-v7a」を選択し、[Next]ボタンを押します[注3]。



最後に、次の画面のようにオプション設定を行った後、[Finish]ボタンを押せば完了です。



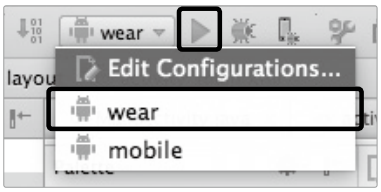
次の画面のように、仮想デバイス一覧に、AVDが1つ追加されているはずです。



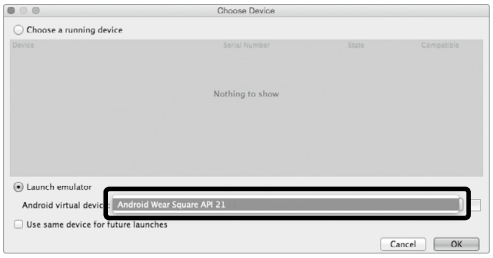
注3）ほかのバージョンやCPUを選択することも可能です。

アプリの実行

作成したアプリを実行してみましょう。今回はWear用モジュールをビルドするため、次の画面のようにStudioのツールバーの中央付近のリストボックスで「Wear」を選択し、緑色の[実行]ボタンを押してください。



どのデバイスで実行するかを聞かれますので、次の画面に示すとおり「Launch emulator」から「Android Wear Square...」を選択した状態で、[OK]ボタンを押せば完了です。



しばらくするとエミュレータが起動し、次の画面のように「Hello Square World!」が表示されればOKです。

おわりに

今回はWearの概要に加え、開発環境の構築、簡単なアプリの開発、エミュレータでの実行までを解説しました。次回からは、より実践的なテーマを対象として、サンプルコードを交えて説明していく予定です。お楽しみに！SD

新連載

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1回 Mackerel事始め

「Mackerel」はSaaSのサーバ管理ツール。自分で監視サーバを立てることなくすぐに運用を開始できる、さまざまなツールと連携し運用の手間を減らせるといった特長があります。本連載はMackerelの使い方を紹介しながら、サーバ管理のノウハウもお伝えしていきます。

Mackerel誕生の背景

Mackerel[注1]は㈱はてなが提供するサーバ管理のためのSaaSです(**図1**)。本連載では、Mackerelの基本的な使い方から、より実践的な活用方法までを紹介していきます。第1回目は、Mackerelの概要から基本的な使い方までを紹介します。

はてなでは、はてなブックマークやはてなブログなどさまざまなWebサービスを運営しており、そのために1,000台以上のサーバを管理しています。これら多数のサーバを管理するための社内向けツールをサービスとして作りなおしたのがMackerelで、2014年5月にベータリリース、2014年9月に正式リリースしており、現在も活発に機能追加を続けています。

Mackerelとは「鯖」を表す英単語なのですが、「サーバ」から「鯖」という連想で名前を付けています。

Mackerelの概要

Webサービスを正しく動作させ続けるためには、そのサービスを動かしているサーバ群の管理を行うことが必要です。これまでのサーバ管理は、ZabbixやNagiosといったオープンソースソフトウェアがよく利用されていますが、徐々にMackerelのような、SaaS型のサーバ管理サービスが使われ出しています。

Mackerelを利用する際、次のようなメリット・デメリットがあります。

▼図1 Mackerelトップページ

①セットアップが容易で、利用開始が簡単
②監視サーバが不要で、運用コストが下げられる
③UIが使いやすく、外部サービス連携も容易
④オープンソースソフトウェアに比べてカスタマイズできる範囲が狭い

サーバ管理方法を選定すると

注1） https://mackerel.io

きには、メリットである①～③とデメリットの④、サービスの利用コストを天秤にかけることになるかと思います。

Webサービスの開発現場では、GitHubやSlackなどのSaaSの採用が進んでおり、サーバ管理もSaaSを利用することが一般的になると考えています。

Mackerelのアーキテクチャ

サーバ管理サービスのアーキテクチャは、大きくPush型とPull型の2種類があります。

- Pull型：管理サーバから各サーバへメトリックを取得しにいく
- Push型：各サーバから管理サーバにメトリックを送信する

両方式の違いは管理サーバと各サーバのどちらが主体となってメトリック(個々のサーバに関連する時系列データ)を得るかにあります。

Pull型アーキテクチャの場合、必要なメトリックをすべて外部から取得できる必要があります。外部からメトリック値を取得するためにSNMP(Simple Network Management Protocol)を利用することが多いです。CentOSやDebianが標準で持つSNMPエージェントを利用することでCPU使用率やメモリ使用率などの基本的なメトリックは取得できるようになります。ミドルウェアのメトリックを収集するには、SNMPエージェントの設定追加など一手間かかります。

Push型アーキテクチャの場合、各サーバに自律的に動作するエージェントをセットアップする必要があります。このエージェントが定期的にメトリックを収集し中央サーバに送信します。エージェントは各サーバの内部で動作しますので、さまざまなミドルウェアへの対応が容易です。

◆◆◆

NagiosはPull型を基本にPush型も使うことができ、ZabbixはPush型を基本にPull型も使うことができます。両形式ともメリット・デメリットがありますので、両方使えるのが理想です。

MackerelはPush型を基本としており、エージェントを各サーバで動作させる必要があります。

Mackerelを使ってみる

ではMackerelを使ってみましょう。ユーザ登録から基本的な監視・通知まで設定してみます。ここでは、VPS1台を対象とします。DebianもしくはUbuntuを想定し、スクリーンショットなどは執筆時点(2015年1月)のものを使っています。

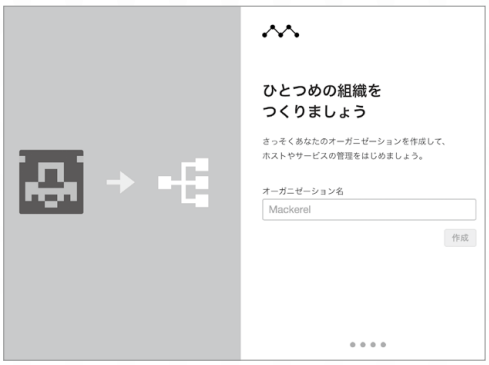
ユーザ登録

初めにトップページからサインアップページ(**図2**)に遷移し、ユーザ登録を行います。ここにメールアドレスを入れると仮登録されます。仮ユーザ登録を行うと簡単なチュートリアルが始まります(**図3**)。

▼図2　サインアップページ

▼図3　チュートリアルページ

▼図4　オーガニゼーションの作成



▼図5　登録直後の画面

チュートリアルの最後にオーガニゼーションを作成します(**図4**)。オーガニゼーションは、Mackerelのサーバを管理するための一番大きな単位となります。このオーガニゼーションには複数のユーザが所属できます。また、1人のユーザが複数のオーガニゼーションに所属することもできますので、たとえば会社用のオーガニゼーションと個人用のオーガニゼーションを同じユーザアカウントで切り替えながら使うことができます。登録が終わると**図5**のような画面になります。

mackerel-agentの概要

Mackerelは前述のとおりPush型アーキテクチャなので、各サーバ上でエージェントとなる**mackerel-agent**を起動する必要があります。mackerel-agentは次のような動作を行います。

①サーバのスペック情報を定期的に送信する
②メトリック情報を定期的(1分ごと)に収集する
③プラグインが指定されている場合、プラグインによりメトリック情報を収集する
④②と③で収集したメトリックをまとめて送信する

ちなみに、mackerel-agentがメトリックを送信する際に利用するAPIは公開されており、標準で提供されているmackerel-agent以外の手段でメトリックを送信することもできます。

mackerel-agentのインストール

次に、VPSにmackerel-agentをセットアップします。次の3ステップで実行できます。

❶リポジトリの設定

```
# curl -fsSL https://mackerel.io/assets/files/scripts/setup-apt.sh | sh
# apt-get install mackerel-agent
```

❷APIキーをmackerel-agentの設定ファイルに追記

```
# sudo sh << SCRIPT
cat >>/etc/mackerel-agent/mackerel-agent.conf <<'EOF';
apikey = "<API KEY>"
EOF
SCRIPT
```

ここでの<API KEY>は初期画面中サイドバーの「Dashboard」から遷移できるページの「Detail」ボタンから取得できます。

❸mackerel-agentを起動

```
# sudo /etc/init.d/mackerel-agent start
```

mackerel-agentを起動するとMackerelのサーバが登録されます。登録されたサーバの一覧は、サイドバーの「Hosts」から遷移できます。サーバ一覧から登録したサーバをクリックすると、ホスト詳細画面に遷移します。

ホスト詳細画面では、そのサーバのLoad Average、CPU使用率、メモリ使用状況、ディスクI/O、ネットワークトラフィック、ディスク容量のグラフが確認できます。エージェント起動後、10分ほど待つとメトリックグラフが描画されるようになります。

登録直後のサーバは「Standby」状態となっています。これはサーバが本番投入前の準備状態であることを示しています。後述のアラート通知が実行されるには「Working」状態に変更する必要があります。これはサーバが本番投入されていることを示します。

監視ルールの設定

次に収集したメトリックに対して監視ルールを設定してみましょう。適切に監視ルールを設定することで、サーバの負荷が異常に高まった際にアラートを発生させることができます。Mackerelの監視ルールは、次の5つの項目を設定します。

- 監視対象メトリック
- 監視対象のサービスとロール
- WarningとCriticalの閾値
- 閾値による判定をするための期間
- 監視ルール名

監視ルールを新しく作成するにはサイドバーの「Monitors」から遷移できる監視ルール一覧画面の右上にある「監視ルールを追加」ボタンをクリックします。ボタンをクリックすると**図6**のダイアログが表示されます。

たとえば、Load Averageが3分間、1を越えたらWarning、さらに5を越えたらCriticalとしたい場合、「監視対象のメトリック」に“loadavg5”、「Warning条件」に“1”、「Critical条件」に“5”を入力します。

「条件の持続時間」で指定された期間の平均値が閾値を越えるとアラートを発生させます。デフォルトでは3分となっていますので、このままにしておきます。また監視ルール名は自動生成されますが、任意に変更することもできます。最後に「作成」をクリックすると監視ルールが作成されます。

▼図6　監視ルールの設定

アラートの確認

設定した監視ルールの下、メトリックが閾値を越えたと判断されるとアラートが生成されます。生成されたアラートはサイドバーの「Alerts」から遷移できる「アラート一覧画面」で確認できます。また、それぞれのアラートをクリックすることで各アラートの個別画面(**図7**)が表示されます。

アラート個別画面では、アラート発生前後の時間帯のメトリックグラフ、該当した監視ルール、発生したサーバと現在の状況を確認できます。またこの画面から手動でアラートを閉じることもできます。

▼図7　アラート個別画面

通知の設定

Mackerelでは発生したアラートを、メールだけ

ではなくさまざまなチャットサービスへ簡単に通知できます。Mackerelではアラートの通知先のことをチャンネルと呼んでいます。チャンネルの設定を行うことで、アラート発生時に指定したチャンネルに通知が流れます。

執筆時点では、メール、Webhook、Slack[注2]、HipChat[注3]、PagerDuty[注4]、TypeTalk[注5]、im.kayac.com[注6]に対応しています。

メールではそのオーガニゼーションに属しているメンバの登録メールアドレスにメールが送信されます。Webhookは、アラートが発生した際に指定したURLにアラートの内容をPOSTします。Slack、HipChat、PagerDuty、TypeTalk、im.kayac.comは、それぞれのサービスにアラート内容が通知されます。

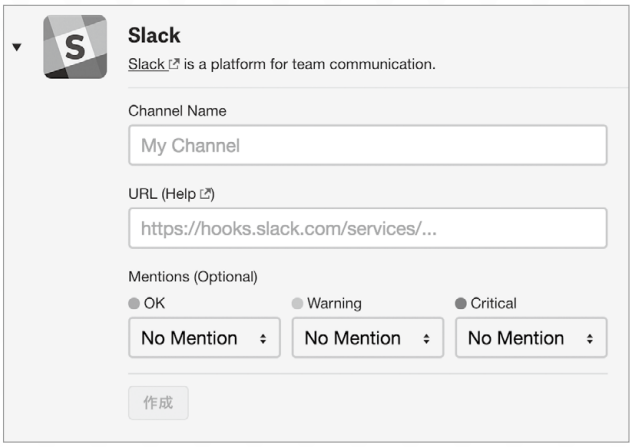
たとえば、Slackへの通知を設定するには監視ルール一覧画面の「Channels Setting」からSlackを追加し、チャンネル名と通知先URLを指定します(**図8**)。通知先URLはSlackのIncoming Webhookの設定(**図9**)のWebhook URLを指定します。

チャンネル設定ができたら、チャンネル一覧画面から通知テストを行うことができます。通知テストを行い、チャンネル設定が正しくできていると、Slack上に**図10**のように通知が表示されます。

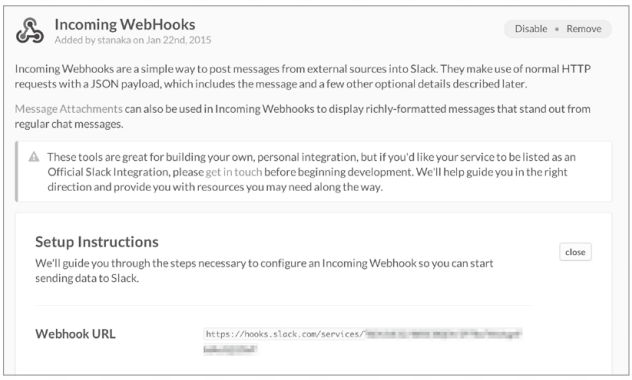
また通知が行われるにはMackerel上のサーバの状態を「Working」にする必要がありますので、注意してください。

ここまででサーバの基本的な管理ができるようになります。

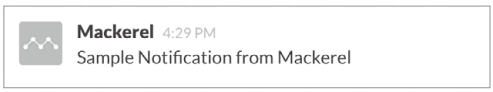
▼図8　Slackのチャンネル設定



▼図9　SlackのIncoming Webhook



▼図10　Slackへの通知テスト



管理対象のサーバ台数が増えてくると、複数のサーバをまとめて扱うことで管理効率を上げる必要が出てきます。

サービスとロール

Mackerelでは「サービス」と「ロール」の2つ

注2）https://slack.com
注3）https://www.hipchat.com
注4）http://www.pagerduty.com
注5）http://www.typetalk.in
注6）http://im.kayac.com

の概念でサーバを分類します。

サービスとは、たとえば「はてなブログ」を提供するためのサーバというように、提供するサービス単位でサーバを分類します。

ロールとは、「アプリケーションサーバ」「データベースサーバ」のように役割によってサーバを分類します。

こうすることでサーバが「はてなブログ」の「アプリケーションサーバ」、「はてなブックマーク」の「データベースサーバ」というように分類されます。

数十台程度にサーバ台数が増えてくると、個々のサーバをひとつひとつメンテナンスするのではなく、複数のサーバをロール単位でひとまとめにして扱うことで、より効率的にサーバ群を扱うことができます。

たとえば、サーバA、B、Cに「はてなブログ」の「アプリケーションサーバ」というロールが割り当てられていた場合に、個々のサーバの名前を意識してひとつひとつ処理するのではなく、サーバ群としてまとめて扱えるということです。

とくにAWSのようなクラウドサービス上でオートスケールさせている場合は、時間とともに負荷によってサーバ台数が増減することになります。このような場合には、ひとつひとつのホストを意識する意味はほとんどなく、まとめて扱うことが一般的です。

サイドバーの「Services」からサービス一覧画面に遷移でき、ここから新しくサービスを作成できます。またサービスを作成すると、そのサービスに割り当てられたサーバ群の詳細が表示される画面(サービス詳細画面)を見ることができるようになります。この画面から新規ロールを作成できるようになります。

サービスとロールの指定

サービス・ロールの指定は、Web UIによる方法とagentの設定ファイルによる方法の2種類があります。

Web UIにより指定する場合、ホスト詳細画面から**図11**のように指定できます。各サーバには複数のロールを紐づけることができます。

mackerel-agentの設定ファイルで指定する場合は次のような設定を設定ファイルに追加します。設定ファイルはデフォルトでは**/etc/mackerel-agent/mackerel-agent.conf**に配置されています。

```
# /etc/mackerel-agent/mackerel-agent.conf
roles = [ "My-Service:app", "Another-Service:db" ]
```

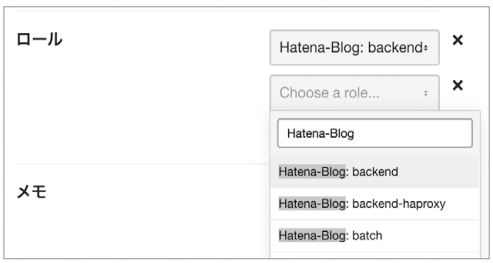
設定ファイルに書いておくことでchefやpuppetなどのプロビジョニングツールを使ってサーバ設定を行う場合もMackerelへの登録を自動化できます。

おわりに

本稿ではMackerelの基本的な使い方を紹介しました。本連載では今後、データベースやアプリケーションサーバ、リバースプロキシなどの各種ミドルウェアの管理方法や外部ツールとの連携方法など、さまざまな活用方法を紹介します。

みなさんもSaaSによるサーバ管理のお手軽さをMackerelで体験してみてはいかがでしょうか。SD

▼図11　サービスとロールの指定



書いて覚えるSwift入門

第3回 演算子

Writer 小飼 弾（こがい だん）　twitter @dankogai

演算子＝へんな関数

連載第4回目の今回は、前回予告どおりSwiftにおける演算子（operators）を詳しくみていきます。

その前に、演算子とは一体なんでしょうか？

なんだか難しそうな定義がありそうな気がしますが、Perlの父、Larry Wallが単純にして明快な定義を示してくれています。

> *operators are just funny looking function calls*[注1]

「ただの変な見た目の関数呼び出し」。たしかに言われてみると、演算子がなくとも一人前な言語はいくらでもあります。とくによく知られているのはLispでしょう。`1 + 1`ではなく`(+ 1 1)`と書けば事足りますし、`3 + 2 * 4`を計算したければ必ず`(+ 3 (* 2 4))`と書くので優先順位の問題もありません。

実際、「へんな関数」がないおかげでLispの実装というのはずいぶんと楽で、`Lisp in ほげほげ`でほげほげ言語を検索すると、誰かが手なぐさみに実装したLispがいくつも見つかりますし、簡単なものは大学の学部生の課題として取り上げられたりします（私はSchemeでSchemeを書かされました。もう20年前）。Swiftの母体ともなったLLVMがChris Lattnerの博士号の課題だったのと比べるとその難易度の差も感じられようというものです。

しかし、演算子がない言語は実装しやすい代わりに使いにくい。`E = m * c ** 2`と書ける言語と、`(setq E (* m (** c))`[注2]と書かねばならない言語では、やはり前者の方が人気者。CもC++もJavaもJavaScriptもPerlもPHPもPythonも、本誌が好んで取り上げる言語はほとんど演算子を持っています。

しかし演算子を再定義したり、ましてや新演算子を定義できたりする言語となると、格段に減ります。Swiftはその数少ない例外。C++のような演算子オーバーロードはもちろん、Haskellのような優先順位定義までサポートしています。これはやるっきゃないでしょう。

二項演算子

前口上はこれくらいにして、実際に使ってみましょう。前回は、

```
struct Point {
    var x:Int
    var y:Int
}
```

を`Printable`にしたところで終わっていました。これを`Equatable`にしてみます。equatableということは、等号があるということ。つまり`==`を実装すればいいわけです。こういうふうに。

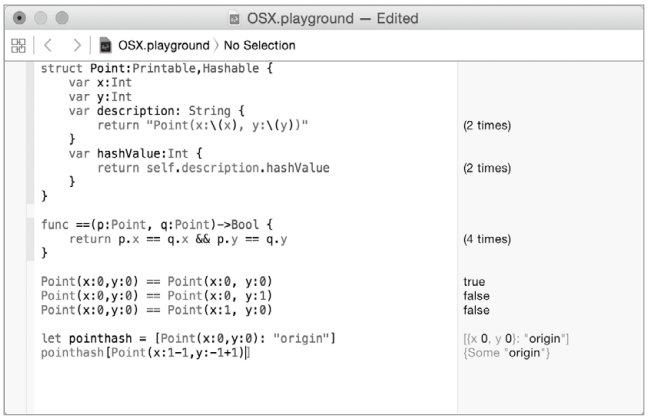
```
func ==(p:Point, q:Point)->Bool {
  return p.x == q.x && p.y == q.y
}
```

こんな簡単でいいのでしょうか？

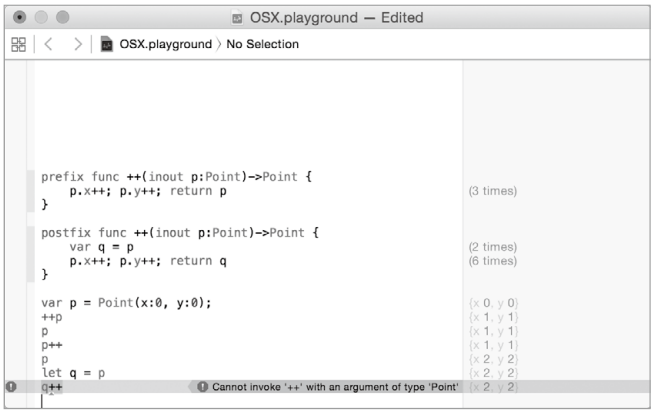
注1） Apocalypse 3: Operators（http://perl6.org/archive/doc/design/apo/A03.html）

注2） ふつうのLispでは`**`は定義されていません。Schemeであれば`(define (** x) (* x x))`といったところでしょうか。

▼図1　二項演算子の実行結果



▼図2　前置と後置の実行結果



いいみたいです(**図1**)。

二項演算子(binary operators)の場合、普通の関数と同じように定義できてしまいます。関数名は、演算子そのもの。簡単ですね。

前置と後置の場合は？

二項演算子の場合、位置に曖昧さはないのでこれで十分なのですが、単項演算子(unary operators)の場合、前置(prefix)と後置(postfix)の2つが考えられます。たとえば++は前置と後置両方ありますよね。

前述のPointを++したいケースというのはあまり考えられないのですが、ここでは便宜上xとyそれぞれ++するということにして、前置と後置を書き分けられるかを試してみることにしましょう。

前置のほうは簡単です。引数を上書きするので、inoutがついている点に注意してください。

```
prefix func ++(inout p:Point)->Point{
    p.x++; p.y++; return p
}
```

後置のほうはちょっと工夫がいります。返り値は++する前の値なので、前の値を一時変数に格納したうえでそれを返しています。

```
postfix func ++(inout p:Point)->Point {
    var q = p
    p.x++; p.y++; return q
}
```

それではうまくいくでしょうか。うまくいったみたいです(**図2**)。引数がvarではなくletで宣言された定数の場合、きちんとエラーになっているところも含めて。

オレオレ演算子の書き方

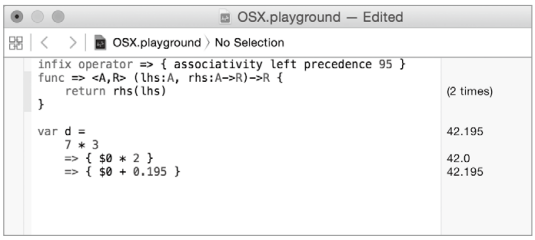
ここまでは、既存の演算子をユーザが定義した型に適用していました。いわゆる演算子オーバーロードで、ここまでであればできる言語は少なくありません。しかしここから先はSwiftのほかにはHaskellぐらいしかできない領域、オレオレ演算子の世界です。

たとえばこんなのはいかがでしょう？

```
infix operator => { associativity left precedence 95 }
func => <A,R> (lhs:A, rhs:A->R)->R {
    return rhs(lhs)
}
```

f(a)の代わりに、a => fと書けるようになります。

▼図3　オレオレ演算子の実行結果



▼図4　公式ドキュメントでのU+の羅列?

AMMAR OF OPERATORS

erator → *operator-head operator-characters*$_{opt}$
erator → *dot-operator-head dot-operator-characters*$_{opt}$

erator-head → / | = | - | + | ! | * | % | < | > | & |
^ | ~ | ?
erator-head → U+00A1–U+00A7
erator-head → U+00A9 or U+00AB
erator-head → U+00AC or U+00AE
erator-head → U+00B0–U+00B1, U+00B6, U+00BB, U+00BF,
U+00D7, or U+00F7
erator-head → U+2016–U+2017 or U+2020–U+2027
erator-head → U+2030–U+203E
erator-head → U+2041–U+2053
erator-head → U+2055–U+205E
erator-head → U+2190–U+23FF
erator-head → U+2500–U+2775
erator-head → U+2794–U+2BFF
erator-head → U+2E00–U+2E7F
erator-head → U+3001–U+3003
erator-head → U+3008–U+3030

erator-character → *operator-head*
erator-character → U+0300–U+036F
erator-character → U+1DC0–U+1DFF
erator-character → U+20D0–U+20FF
erator-character → U+FE00–U+FE0F
erator-character → U+FE20–U+FE2F
erator-character → U+E0100–U+E01EF
erator-characters → *operator-character operator-characters*$_{opt}$

t-operator-head → ..
t-operator-character → . | *operator-character*
t-operator-characters → *dot-operator-character dot-operator-characters*

nary-operator → *operator*
efix-operator → *operator*
stfix-operator → *operator*

これの何がうれしいかというと、たとえばprintfデバッグならぬprintlnデバッグの時。Swiftにはplaygroundがあるのでprintlnデバッグしなければならないケースは他より減るとはいえ、コマンドラインでのテストなどにやはり重宝します。こういう時にわざわざprintln(と書いて)で閉じるより、そのままうしろに=> println と付け加えた方が楽ですし直感的でもあります。

見てのとおり、ユーザが演算子を定義する際には、(pre|in|post)fix operatorで演算子を定義しておく必要があります。続く{}の中で、結合の方向と優先順位を定義します。ここでは左結合、優先順位95なので、=>の重ね打ちも**図3**のような感じでできたりします。

演算子に関しては、「こんなのどうよ」というのをGitHub[注3]にまとめてあるのでよろしければご確認を。

The (Un)?documented Feature

それではどのような記号が演算子として使えるのか。公式ドキュメントにはこう書いてあります(**図4**)。

U+の羅列、なんのこっちゃという感じですが、これ、Unicodeの記号文字です。ということは……、**リスト1**のコードが動くということではありませんか？　playgroundで実行してみてください(**図5**)。

1つ注意したいのは、使えるのはあくまで記号であって、ギリシャ文字などは記号ではなく普通の文字として扱われること。**図5**のΣはU+3A3(GREEK CAPITAL LETTER SIGMA)ではなくU+2211(N-ARY SUMMATION)、∏もU+3A0(GREEK CAPITAL LETTER PI)ではなくU+220F(N-ARY PRODUCT)です。

記号が使えることはかつては公然の秘密でしたが、Swift 1.1現在では公式化されています。

さらに次のコードを追記してみてください。

```
for n in 1...16 {
    let e_1 = 1...n .  {1/(1...Int($0) ∏  {$0})}
    println( exp(1) - e_1 )
}
```

どんどん1に近づいています。どこかで見たことがありませんか？　そう、ネイピア数、eの定義です。

注3）https://github.com/dankogai/swift-operators/wiki

指数関数exp(x)はテイラー展開で

$$exp\ x = \sum_{n=0}^{\infty} \frac{x^n}{n!}$$

と書けますが、1...n ・ {1/(1...Int($0) ∏ {$0})}はexp(1) − 1に相当するわけです。なぜexp(1)そのものではないかというと、階乗計算に使っている∏は、上の定義では0!が扱えないから。第0項がない分、1少ないわけです。

ここまでくるといっそ後置の!を定義してn!とかやりたい誘惑に駆られますが、残念ながらn!はImplicitly Unwrapped Optionalsのために予約されていて再定義不可能です。

まとめ

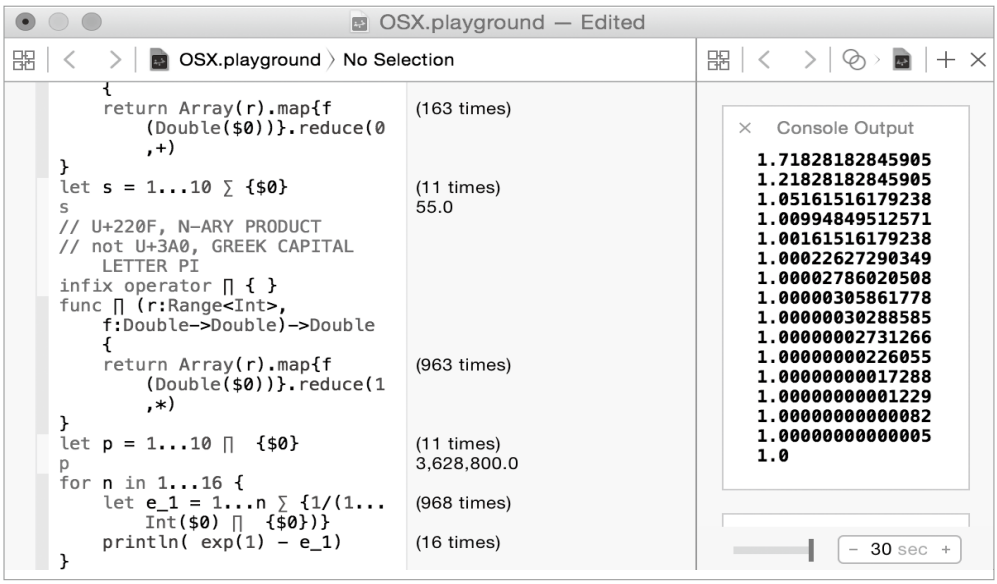
Lispを見ればわかるとおり、演算子というのはプログラミング言語に必須の機能ではありません。が、やはりLispを見ればわかるとおり、演算子というのは一定以上の人気を得るのに欠かせない機能でもあるようです。この点において、Swiftに比肩するのはHaskellぐらいしか見当たりません。

そして一定以上の人気を得るのに欠かせないいまひとつの機能が、過去の遺産の継承。次回はSwiftからCやObjective-Cの遺産を活用します。SD

▼リスト1　公然の秘密のサンプルコード

```
import Foundation
prefix operator √ {}
prefix func √(x:Double)->Double {
    return sqrt(x)
}
√2
// U+2211, N-ARY SUMMATION
// not U+3A3, GREEK CAPITAL LETTER SIGMA
infix operator ∑ { }
func ∑ (r:Range<Int>, f:Double->Double)->Double {
    return Array(r).map{f(Double($0))}.reduce(0,+)
}
let s = 1...10 ∑ {$0}
s
// U+220F, N-ARY PRODUCT
// not U+3A0, GREEK CAPITAL LETTER PI
infix operator ∏ { }
func ∏ (r:Range<Int>, f:Double->Double)->Double {
    return Array(r).map{f(Double($0))}.reduce(1,*)
}
let p = 1...10 ∏ {$0}
p
```

▼図5　リスト1の実行



目指せ！定時帰りのエンジニア！

Hinemosで学ぶジョブ管理超入門

㈱NTTデータ 基盤システム事業本部　山本 未希（やまもと みき）

新人SE藤井君の奮闘記第6回。今回は、Hinemosを使ったシステムのメンテナンスについて学びます。Hinemosクライアントからの履歴管理、メンテナンススクリプトによるデータベースの管理・バックアップの方法など、運用においては外せない知識を紹介します。　イラスト（高野 涼香）

第6回　さて運用開始だ！ でも始まってみると新たな課題がいっぱい!?

登場人物紹介

藤井　今年SI企業に入社した新人SE。運用自動化のために日々奮闘中。

上司　軽いノリで仕事を依頼してくるが、藤井君の成長を考えている。

定時先輩　藤井君の先輩社員。華麗に仕事をこなし定時に颯爽と帰っていく優秀な女性。

前回までのあらすじ

今年SI企業に入社した新人SEの藤井君は、社内の「勤怠管理システムの運用」を上司から任されました。試行錯誤の末、ようやくオープンソースソフトウェア運用管理ツール「Hinemos」を使うことに。ジョブの設定やスケジュール設定を済ませ、勤怠管理システムの運用を開始したのでした。

定時「お疲れさま。システムの運用はうまくいってるの？」

藤井「はい、おかげさまで！　Hinemosを使い始めてからは、実行時間の変更があっても設定画面からすぐに変更できるし、結果も同じ画面で確認できるので、あっちこっち見ずに済んで運用もスムーズです！」

定時「それはよかった。最近は藤井君も定時に帰れてるみたいだし、本当に順調なようね」

ジョブの再実行をするには？

定時「そういえば、今はジョブの結果をどうやって確認しているの？」

藤井「普段は、ジョブ管理パースペクティブのジョブ［履歴］ビューとかジョブ［ジョブ履歴］ビューで実行がどこまで進んでいるか、無事に「正常」で終了したかを確認してます。前回学んだ通知の機能で「メール通知」も設定しているので、ジョブが「警告」とか「異常」で終了したら、僕にメールが届くようになっているんです」

定時「なるほど。それなら画面をずっと見てなくても異常時にすぐ気づけるわね」

藤井「そうなんです！　実はこの間、ジョブが異常終了してメールが届いたんですが、どうやらネットワークの疎通の問題で一時的に実行できないタイミングがあったみたいで……。でもHinemosだと、ジョブネットの途中で失敗したジョブがあっても、そのジョブから手動で再実行がかけられたので助かりました」

Hinemosでは、ジョブの結果を確認できるジョブ［ジョブ詳細］ビュー（**図1**）から、特定のジョブを再実行できます。また、実行したジョブに後続のジョブがあった場合は、その後続ジョブも続いて実行されます。たとえばジョブネットの途中で特定のジョブが止まってしまい、それ以降の処理も含めて再実行をかけたいときなどのリカバリ処理も簡単に行うことができます。

ちなみに、前回も登場したHinemosジョブマップオプションというオプション製品を導入すると、ジョブの実行状況がHinemosクライアントの画面にマップ形式で表示され、実行状況（待機中、実行中、終了）やジョブのつながりが一目でわかります。どのジョブが異常終了したのか、再実行をかけたらそれ以降どのように処理の流れが進んでいくのかを把握したいときにとても便利です。

メンテナンスもしっかりやろう！

ジョブの実行履歴ってどう管理するの？

定時「そうそう、あとはジョブの実行履歴ってちゃんとメンテナンスしてる？」

藤井「えっ実行履歴のメンテナンスですか？　いや、そういえばあんまり気にしたことがなかったような……」

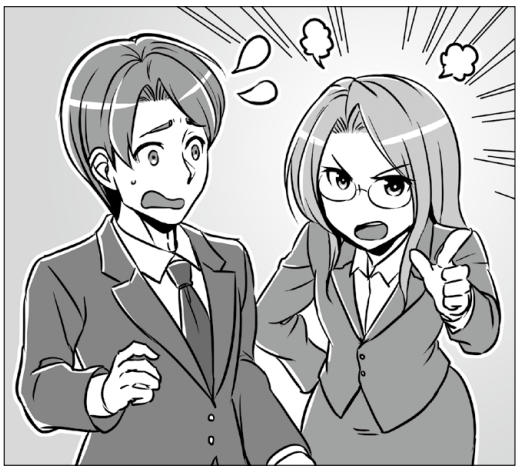
定時「こら！　ジョブ管理では、実行履歴をどのくらいの期間保存しておくか定義して、しっかり管理することが大事なのよ。必要なものだけ残して、それ以前のものは削除するっていうのが基本的な考え方ね。これをちゃんと実施してないと、不要な履歴データでデータベースの容量は取られるし、性能面でのデメリットも発生するかもしれないのよ！」

藤井「う……な、なるほど。これはちゃんと管理しないといけないですね。でもメンテナンスって、またシェルスクリプトを書いたりとか、何か難しいことが必要なんですか？」

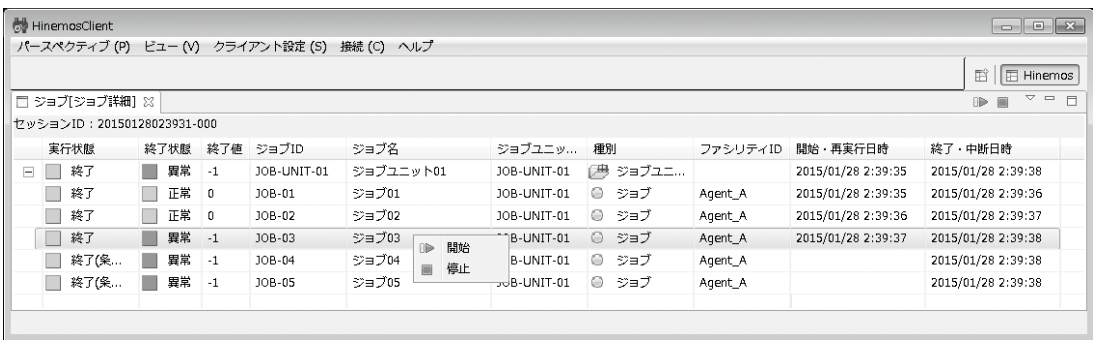
定時「その点は大丈夫よ。Hinemosにはメンテナンス機能っていうのが備わってるから、それを使えばいいわ」

藤井「ちゃんと機能として用意されてるんですね！」

定時「そうよ。ジョブの設定と同じように、Hinemosクライアントの画面から履歴削除のスケジュール設定ができるの」



▼図1　ジョブ［ジョブ詳細］ビュー



▼図2　メンテナンス［履歴情報削除ビュー］

ジョブの実行履歴に限らず、Hinemosではメンテナンス機能を使うことで、データベース内の履歴情報を定期的に削除できます。

メンテナンス機能はメンテナンスパースペクティブのメンテナンス［履歴情報削除ビュー］（**図2**）から実行します。画面右上の作成ボタンをクリックすると履歴削除［作成／変更］ダイアログ（**図3**）が表示されるので、そこから削除したい履歴情報やスケジュールを設定します。削除できる履歴情報としては次のものがあります。

- 監視（イベント）履歴：監視［イベント］ビューに表示される情報
- ジョブ実行履歴：ジョブ［履歴］ビューに表示される情報
- 一括制御履歴：一括制御［履歴］ビューに表示される情報
- 性能実績：性能［一覧］ビューに表示される情報

また、あらかじめ各履歴情報ごとにデフォルトの設定も用意されているので、こちらを活用するのも良いでしょう。

藤井「デフォルトの設定だと1ヵ月に1回履歴を削除する設定になっているんですね」

定時「そうね。このまま使ってもいいんだけど……、勤怠管理システムはジョブの実行数も多いからもう少し短い周期で削除したほうが良いかもしれないわね」

藤井「なるほど、そうですね。えーっと、新たに作成するには右上の作成ボタンをクリックしてっと……このメンテナンス種別というところでどの履歴情報を削除するか選ぶんですね」

定時「そうね。今回は『ジョブ実行　履歴削除』を選べばいいわね。保存期間は履歴を過去どの範囲まで保存するか、スケジュールは削除するタイミングを指定すればOKよ」

▼図3　履歴削除［作成／変更］ダイアログ

藤井「じゃあ……、保存期間はとりあえず14日にして、スケジュールは0:00に設定しておきます！　これで毎日0:00になったら2週間より古いものを削除してくれるんですよね」

定時「そうなるわね。あとは、ジョブの実行数がさらに増えてきたときに、また保存期間やスケジュールを見直すとよさそうね」

履歴情報以外のメンテナンス

定時「これで実行履歴のメンテナンスもバッチリね。そういえば管理者ガイドっていうマニュアルにも書いてあったけど、Hinemosクライアントから設定するメンテナンス機能とは別に、メンテナンススクリプトっていうシェルスクリプトが用意されているみたいよ。これは履歴の削除だけじゃなくて、Hinemosマネージャのログ情報

▼図4　メンテナンススクリプトでバックアップ

```
$ cd /opt/hinemos/sbin/mng        //メンテナンススクリプトを配置しているディレクトリ
$ ls
hinemos_backup.sh         hinemos_cluster_db.sh       hinemos_reset_scheduler.sh
hinemos_clear_notify.sh   hinemos_delete.sh           hinemos_restore.sh
hinemos_clear_tmp.sh      hinemos_manager_summary.sh
$ ./hinemos_backup.sh       //バックアップの実行
input a password of Hinemos RDBM Server (default 'hinemos') :
dumping data stored in Hinemos RDBMS Server (PostgreSQL)...
   output file : /opt/hinemos/sbin/mng/hinemos_pgdump. 2015-01-28_033212
successful in dumping data.
$ ls        //バックアップファイルの確認
hinemos_backup.sh         hinemos_manager_summary.sh
hinemos_clear_notify.sh   hinemos_pgdump.2015-01-28_033212
hinemos_clear_tmp.sh      hinemos_reset_scheduler.sh
hinemos_cluster_db.sh     hinemos_restore.sh
hinemos_delete.sh
```

の削除とかデータベース自体のメンテナンスもできるみたいだから覚えておくとよさそうね」

藤井「そうか……、長く運用していくと履歴の削除だけじゃなくていろいろと気にしなきゃいけないことが出てくるんですね。さっそく管理者ガイドのマニュアルを確認してみます！」

Hinemosを長期間使用するには、各環境の使用状況に合わせた、定期的なメンテンス作業が必要となります。Hinemosでは、Hinemosマネージャのメンテナンス作業に必要なシェルスクリプトをあらかじめ提供しているので、このスクリプトをうまく活用して運用していきましょう。ここではメンテナンススクリプトの一部を紹介します。

- hinemos_delete.sh：データベース内から不要なデータ（保持する必要のなくなったログ情報など）を削除する
- hinemos_cluster_db.sh：データベースを再構成して不要領域（使用されていない確保領域）のシステムによる再利用を可能にする
- hinemos_clear_tmp.sh：一時キューに格納されている情報（テンポラリ情報）を消去する

ジョブの設定をバックアップするには

藤井「そういえばさっき出てきたメンテナンススクリプトの一覧に、設定データのバックアップとかリストアっていう文字があったんですが……」

定時「おっ、良いところに気づいたわね。そうなの、Hinemosのメンテナンススクリプトには、リポジトリとかジョブといったHinemosの設定をバックアップ・リストアするためのシェルスクリプトが用意されているのよ」

藤井「そうだったんですね！　じゃあこのシェルスクリプトを設定の変更後とか定期的に実行しておけば、万が一のときも安心ですね」

定時「そうね。せっかくだから今の設定もバックアップを取っておいたらどうかしら？　バックアップは『hinemos_backup.sh』を実行すればいいみたいよ」

藤井「やってみます！　え〜っと、どれどれ……メンテナンススクリプトは全部『/opt/hinemos/sbin/mng』ディレクトリにあるみたいだから移動してっと。おっ、ほかのメンテナンススクリプトもちゃんと配置されてますね。ではさっそく実行してみます！（**図4**）」

定時「……うん、ちゃんとバックアップがとれたみたいね！　作成したファイルに日付も入っていてわかりやすいわね。あと、設定を戻すときは『hinemos_restore.sh』を使うみたいね」

藤井「なるほど。せっかくなのでリストアも試してみます！」

定時「あ、でもリストアするときはHinemosマネー

ジャを止める必要があるみたいよ？　これは検証環境で試したほうがよさそうね」

藤井「バックアップはHinemosが起動したままでも取れるけど、設定を戻すときはHinemosマネージャを止める必要があるんですね。では検証環境で……えーっと、まずはHinemosマネージャを止めて、そのあと内部データベースだけ動かしておいて、最後に『hinemos_restore.sh』を実行するときにさっき作成したバックアップファイル『hinemos_pgdump.2015- 01-28_033212』を指定してっと……やった！　これもできました！（**図5**）メンテナンススクリプトを使うと簡単に設定のバックアップもリストアも実行できますね。これでますます運用に自信がつきました！」

定時「これなら何かあっても安心ね。ちなみにオプション製品のHinemos Utilityオプションを使うと、データベースをまるごとバックアップするだけじゃなくて、リポジトリとかジョブの機能ごとに設定をエクスポート、インポートすることができるみたいよ。あとエクスポートした設定を専用のExcelファイルで編集してから、インポートするっていうのもできるみたいね」

藤井「へぇ～！　じゃあ、たとえばジョブの設定だけエクスポートして、ちょっと設定を書きなおして、またHinemosマネージャに設定を戻すっていうこともできるんですね」

定時「そのとおり！　設定の移行とか今後の運用を考えるとHinemos Utilityオプションはあったほうが便利そうね。そういえば、さっきリストアしたけれど、手順の最後にある『Hinemosマネージャの起動』も忘れないようにね！」

藤井「は、はい！（ひえ～忘れてたっ）」

今月の時短ポイント

ついにHinemosを使った運用を始めた藤井君。日頃からのメンテナンス作業は面倒に思われがちですが、通常運用時のメンテナンスをしっかり実施することで故障や性能問題の発生を回避し、長い目で見ると不要な作業時間を短縮できるのです。

次回予告

上司「定時君と一緒に運用のノウハウも学んでいるようだな。感心感心！　っと、そうだった。順調なところ悪いんだが、実はほかの部署からまたジョブの設定を変えたいという依頼が来ているんだ。頼まれてくれるか？」

藤井「はい、任せてください！　もうHinemosのジョブの設定はバッチリです！」

上司「はっはっは、ではお願いしようか。依頼内容としては、特定のジョブだけ飛ばして後続のジョブはそのまま実行させたり、ジョブネットの中で特定のジョブだけいったん実行を止めたいということらしいんだが」

藤井「えっ飛ばしたり、いったん止めたり……ですか？　えーっと、うーん……」

上司「どうやらまだまだHinemosのことを調べる余地はありそうだな」

藤井「は、はい！　もうちょっとジョブの設定方法を調べてみます！（ぴゅーんっ）」

ジョブの自動化が一段落して安心していた藤井君でしたが、運用を始めたことで定期的なメンテナンスや設定のバックアップといった運用中の作業も重要だということを学びました。次回はジョブの、さらに詳細な設定方法を学んでいくようです。

次回「飛ばして、留めて。自在にジョブを扱おう」SD

To Be Continued...

▼図5　メンテナンススクリプトでリストア

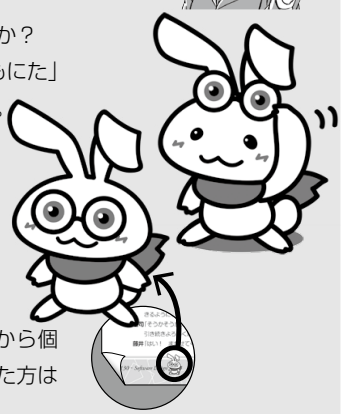
```
$ pwd
/opt/hinemos/sbin/mng
$ /opt/hinemos/bin/hinemos_stop.sh      //Hinemosマネージャを停止
waiting for Hinemos Manager to stop...

waiting for Java Virtual Machine shutdown...
Thread Dump 1
Thread Dump 2
Thread Dump 3
......done
Java Virtual Machine stopped

waiting for PostgreSQL shutdown...
PostgreSQL stopped (shutdown mode : fast)

Hinemos Manager stopped
$  /opt/hinemos/bin/pg_start.sh     //内部データベース（PostgreSQL）の起動
waiting for PostgreSQL startup...
PostgreSQL started
$ ./hinemos_restore.sh hinemos_pgdump.2015-01-28_033212       //リストアの実行
input a password of Hinemos RDBM Server (default 'hinemos') :
dropping hinemos database...
restoring Hinemos RDBMS Server from file (PostgreSQL)...
   input file : hinemos_pgdump.2015-01-28_033212
successful in restoring Hinemos RDBMS Server (PostgreSQL).
$ /opt/hinemos/bin/pg_stop.sh          //内部データベース（PostgreSQL）の停止
waiting for PostgreSQL shutdown...
PostgreSQL stopped (shutdown mode : fast)
$ /opt/hinemos/bin/hinemos_start.sh      //Hinemosマネージャを起動
waiting for Hinemos Manager to start...

waiting for PostgreSQL startup...
PostgreSQL started

waiting for Java Virtual Machine startup.....................done
Java Virtual Machine started
Hinemos Manager started
```

Column

Hinemos公認キャラクタ「もにた」って？

本連載の第4回から登場しているキャラクタについてお気づきでしょうか？ 実は、このキャラクタはHinemosの公認キャラクタなんです！　名前は「もにた」といいます。今回は、この「もにた」のプロフィールを少しだけ紹介します。

名　前：もにた
性　格：運用管理に困っている人を見るとほおっておけない
語　尾：「～もに！」
その他：普段は上げているメガネを下ろすとある能力が……



こんな所にも！

ちなみにこの「もにた」のキャラクタデータはHinemosポータルサイトから個人利用向けに自由にダウンロードいただけますので、気に入っていただけた方はぜひダウンロードして、デスクトップの壁紙などにお使いください。

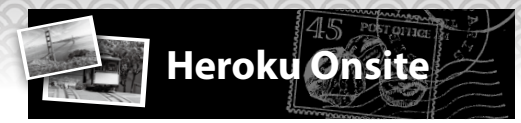
サポートエンジニアのクラウドワークスタイル

Heroku女子の開発日記

最終回 知恵者からの教え
Herokuスペシャリストに聞く開発Tips

PaaS「Heroku」の使い方をお伝えする連載も今月で最終回です。今回はHerokuでアプリを開発する際にユーザをサポートするメンバから、開発言語ごとのヒントや、アプリ運用においての秘訣を伺いました。「サンフランシスコだより」では、アメリカの冬のイベントを紹介。日本との違いに注目です。

Heroku　織田 敬子（おだ けいこ）



1月19日から23日までの1週間、サンフランシスコ本社でHerokuサポートチームのオンサイト（チーム合宿のようなもの）が行われました。現在サポートチームにはメンバが10人以上いますが、マネージャを除くと筆者以外は全員リモート勤務です。そんないつもは離れ離れなチームだからこそ、こうして集まる機会はとても貴重です！　また、同時期に別のチームのオンサイトも行われ世界中から従業員が集まっていたので、今回はこのサンフランシスコに集まったメンバにHerokuを使う上での勘所などをインタビュー形式で聞いていきたいと思います（以下、Keikoは筆者）。

Twitter @HunterLoftis

Keiko：Heroku上にもNode.jsを使ったアプリがどんどん増えていますね。Herokuでよく使われている、お勧めのフレームワークはありますか？

Hunter：Node.jsのフレームワークというと、2種類に分けられます。Express[注1]または、MEANやSailsといったExpressベースのもの、もしくはhapi[注2]です。どちらもHeroku上でよく動きます。たいていのプロジェクトでは、Expressのほうが小さくシンプルで柔軟性があるため向いているでしょう。一方hapiは大きなプロジェクト向けにデザインされています。

Keiko：Node.jsをHerokuで使う上で気をつけておきたいことはなんですか？

Hunter：npmのデフォルトは、実はベストプラクティスじゃない場合が多いのです。というのも、npmはけっこう古い部分があるからです。たとえば、`npm install`を走らせるとデフォルトではpackage.jsonファイルが変更されることはなく、ただローカルにパッケージがインストールされてしまいます。たとえばRubyだと、`bundle install`を走らせるとGemfile.lockが更新されますよね。ですので、常に`npm install`には`--save`オプションを付けましょう。たとえば、`npm install express --save`というようにです。また、アプリが大規模になってきたり、プロダクションとしていったんローンチしてしまったあとにはshrinkwrap[注3]を使うのも1つのテクニックです。

Keiko：パフォーマンス系のNode.jsのチケットですと、cluster[注4]がよく出てきます。cluster関連でのアドバイスはありますか？

Hunter：そうですね、常にclusterを使うようにしましょう。とくにHerokuでPX dynoを使っている場合は必ず使うようにしましょう。clusterを使うことにより、マルチコアを活かすことができます。自分で実装するのもいいですが、僕の作ったthrong[注5]を使うと、

注1）URL http://expressjs.com
注2）URL http://hapijs.com
注3）URL https://docs.npmjs.com/cli/shrinkwrap
注4）URL http://nodejs.org/api/cluster.html

より手軽にclusterが導入できます。workersの数は、アプリを再デプロイすることなく環境変数の設定で変更し、適切なworker数をメモリ使用量と見比べながら見極めるようにするといいでしょう。1X、2X dynoですとworkerは1〜2個から試してみてください。

Terence Lee / Ruby team

Twitter @hone02

Keiko：HerokuのRubyチームって、どんなことをしているのでしょう？

Terence：HerokuのRubyチームの役割は、お客様にHeroku上でのすばらしいRuby体験をしてもらうことです。そのために、お客様が何か煩わしいことをすることなく、Rubyアプリがなるべくシームレスにherokuにデプロイできて、それがうまく走るように日々取り組んでいます。たとえば、もしRubyをHerokuプラットフォーム上で動かすときに問題があった場合、コミュニティと積極的に協力してうまく動くように変更を加えていきます。たとえば、Rails issue team(Richard)やRuby securityとcoreチーム(Terence)にチームメンバがいます。また、day zeroリリースはHerokuのRubyチームの特徴とも言えるでしょう。Rubyに精通したメンバがいるからこそ、Rubyがリリースされたその日にHeroku上でも最新版が使えるようになっています。

Keiko：サポートチームで働いていて、Rubyチームに投げたお客様からのフィードバックがRubyやRailsに適応されているシーンを何度も見てきました。とてもいいことですよね。最近で言うとどんなことをしましたか？

Terence：最近の例だと、古いバージョンのRuby(1.8.7と1.9.2)へセキュリティーフィックスをバックポートしました。これは、Herokuでこれらのバージョンを使っているお客様に対して短期間での移行を強制することを避け、8ヵ月という十分な移行猶予期間を提供するためでした。

Joe Kutner / Java (JVM) team

Twitter @codefinger

Keiko：Heroku上でJavaはどんなふうに使われていますか？

Joe：ClojureやScalaといった新しい言語や、それを使用したPlayといったようなフレームワークと組み合わせて使われることが多いです。Javaはとても古く歴史のある言語で、長らくの間その古さから脱却できずにいました。しかし、約3年半前にHerokuがJavaをサポートし始めてからの頃と比べると、Javaという言語はより進化して現代的になり、このような新しい言語やフレームワークは現在Herokuととても相性がいいです。

Keiko：Heroku上でJavaユーザはどのようなアプリケーションサーバを使用していますか？

Joe：1番はTomcatで、2番はJettyでしょう。これらは非常に安定しており、Heroku上でも問題なく動きます。新しいものが好きな人は、Playフレームワークに標準で組み込まれてい

▼ 写真1　Terenceとのインタビュー風景

注5）URL https://github.com/hunterloftis/throng

るNettyをそのまま使うのがいいでしょう。

Joe はThe Healthy Programmer[注6]という本の著者でもあります。この本は、長時間椅子に座りっぱなしの我々プログラマが抱えがちな健康問題にアプローチする本です。残念ながらまだ日本語訳は出ていないですが、Joe曰く、出るはずとのことです。

Andy Macdonald / TAM

Twitter @theandym

Keiko：Andy はSan Francisco在住で毎日顔を合わせていますが、TAM(Technical Account Manager、Herokuコンサルのようなもの)の仕事はいいネタになると思うのでぜひ聞かせてください。TAMの仕事としては大規模アプリを運用するお客様のお手伝いをすることが多いですが、よくお客様にアドバイスをすることはなんでしょう？

Andy：大きく6つあると思います。

①Twelve factor[注7]にのっとった開発をすること
②CIツールなどを使用しテストを自動化、デプロイ前には必ずテストを行うこと
③本番環境と開発環境をできるだけ近づけること
④ステージング環境を活用すること、またデータはできる限り本番に近いものを使用すること
⑤アドオンなどを活用しHeroku上でのメモリやCPU使用量などをしっかりとモニタリングをすること
⑥負荷テストを確実に行い、ローンチ前にアプリがスケールすることを十分に確認すること

Keiko：何かこのアドバイスをうまく適用したお客様の例はありますか？

Andy：あるお客様のアプリがアメリカのShark Tank(マネーの虎に似た番組)という番組で取り上げられることになりました。収録からTV放映まで数ヵ月の準備期間があり、その期間をTAMとしてサポートすることになりました。まずは想定されるアクセス数での負荷テストをし、そのテスト結果からアプリのどの部分が悪いのか、ボトルネックは何かというのをお客様に提案していき、それを元にお客様が変更を加え……という良いサイクルができました。これにより、収録時点の構成では到底捌(さば)けなかった莫大な量のアクセスを放映時に見事に捌き切ることができました。

サンフランシスコだより

年末年始は日本に帰り、3週間ほどを実家のある金沢で過ごしました。今年には北陸新幹線も完成するということで金沢駅前は賑わいを増していました。ぜひみなさんも、機会があったら北陸新幹線で金沢に！

また、1月13日には東京にてHeroku Meetupを行いました。今回のMeetup[注8]はHerokuの新機能紹介、事例紹介など充実した内容だったと思います。Herokuでは数ヵ月に1回Meetupを行っているので、興味のある方はぜひいらしてください。Meetupの情報はTwitterアカウント@herokujpで流れます。

Herokuのサポートエンジニアは時間・場所にとらわれずに働けるので、日本滞在中の3週間もすべてお休みをとったわけではなく、日本からリモートで働いていました。こういった働き方ができるのは、Herokuのサポートチームならではだと思います。

さて、少し時期がずれますが今回はサンフランシスコの年末の過ごし方について紹介したいと思います。

サンクスギビング

サンクスギビングとは、11月の第4木曜日にあるアメリカの休日です。家族で集まってターキー(七面鳥)を始めとした豪華な料理を、お腹

注6）URL http://healthyprog.com
注7）URL http://12factor.net/
注8）URL http://herokujp.doorkeeper.jp/events/18242

がはち切れるまで食べるのがならわしとなっています。これは、日本で言うとお盆のようなものでしょうか。この日の前後は交通機関が混雑しますが、当日はお店が午前中で閉まったりなど、自宅で家族みんなと祝う人が多いです。

筆者のまわりでは、去年はたいていのアメリカ人の同僚はそれぞれの実家に帰り家族とサンクスギビングを過ごしていました。前後も含めて1週間ほど休暇をとっている人も珍しくなかったです。筆者は今年はサンフランシスコで過ごし、友達とターキーを焼いて食べました。

このサンクスギビングの翌日の金曜日はブラックフライデーと呼ばれ、クリスマスに向けて小売業界が一斉にセールをしかけ始めます。人気店では深夜から開店に向けて並んでいる人も少なくありません。日本で言う初売りのようなものですね。

ホリデーパーティー

サンクスギビングが終わると、12月はホリデーシーズンとなります。みんながあまり働かなくなり、サポートチーム的に言うとチケットの数も減ってきます。ホリデーシーズンにはよくホリデーパーティーが開かれます。Herokuでも毎年「Matsuri」(祭)という名前でパーティーを行っています。Matsuriは毎年違ったテーマで行われ、優秀なVibeチーム(雰囲気チーム、総務のようなもの)が本当にいいパーティーに仕上げてくれます。今回はオフィスで開催され、マジックショーやフェイス&ボディーペイント、もちろん美味しい料理にお酒、などなど……とても素敵な会でした。普段カジュアルな格好ばかりしている同僚のフォーマルな姿を見られる数少ない機会で、そこも非常に楽しめます。これはとくにHerokuだけで行われているわけではなく、規模は違えどサンフランシスコではたいていどの会社でもやっています。

▼写真2　Heroku holiday partyでの1コマ(女子っぽい?!)。1番左が著者です

クリスマス

同じ「クリスマス」という名前のイベントですが、日本とアメリカではとらえ方が大きく違います。アメリカのクリスマスは、家族と過ごし家族のためにプレゼントを用意するものであり(もちろんプレゼントを用意して恋人と過ごすこともできますが……)、日本で言うとお正月とお年玉といった感覚です。また、アメリカではとくにフライドチキンもクリスマスケーキも食べません。日本人はクリスマスにケンタッキー・フライドチキンを食べると言うと、かなりの確率でアメリカ人にウケます。クリスマスイブとクリスマスはアメリカの会社はたいていお休みで、その前後を含めてたっぷり休暇をとる人が多いです。その間、彼らはまた実家に帰ります。「この間もサンクスギビングで帰ったよね?」とつっこみたくなるのですが、そう考えると筆者は日本の盆と正月のシステムのほうが好きです。アメリカ人の同僚や上司などがいる方は、12月のレスポンスの悪さに苦労したこともあるかと思います。

今回でこの連載は終了です。最後までお付き合いいただき、ありがとうございました。最後まで「女子」的なところがあまり出ませんでしたが……少しでもこの連載がみなさんにとって実りのあるものであったことを祈って。SD

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11回 Emacsに革命を起こすパッケージ「helm」（前編）

helmはファイル検索を始めとしたEmacsの各種機能に対して、まったく新しい操作を提供するパッケージです。helmにより、操作対象の絞り込み→選択→アクションの適用という流れで機能を利用できます。前編となる今回はhelmの概要・歴史、helm-miniを使っての実習を扱います。

Emacsを支配するもの

ども、るびきちです。本連載も来月で1周年となります。1年の締めくくりとして今月と来月の2回に分けてhelmという超絶便利なパッケージを紹介します。

helmというパッケージとは何かを言葉で説明するのはとても難しいことです。ですが、一度使ってみればその他のEmacsのコマンドとは異次元のものを感じることは間違いありません。なぜなら、その他のEmacsのコマンドとは考え方そのものが根底から異なるからです。

helmとはヘルメットではなくて「舵（かじ）」とか「支配」という意味ですので、「Emacsの舵」とか「Emacsの支配者」ということになりますが、実際その名に相応しい活躍をします。

新しいパラダイムへ

プログラミングの世界では幾度かパラダイムシフトが起こりました。手続き型からオブジェクト指向へと移行したのは特筆すべきことです。たとえば文字列や配列の長さを求める関数は、手続型の場合だと型に応じて別な関数名を用意する必要がありますが、オブジェクト指向ならば同じメソッド名でよくなります。string_length(a_string)、array_length(an_array)が、a_string.length()、an_array.length()と書けます。より人間にわかりやすいコードになりました。それをさらに推し進めると配列に似たオブジェクト「リスト」も定義できます。リストに配列と同じメソッドを用意してあげれば、人間にとっては「リストは配列と同じように扱えて、覚えるのが簡単」なものとなります。書き方も手続型では関数を先に指定するのに対し、オブジェクト指向ではオブジェクトを先に指定します。

これと同じことがhelmによりEmacsの世界で起こるようになります。

通常のEmacsは手続型の世界です。なぜなら、ファイルを開くことに関しても「ファイル名を指定してファイルを開く」「最近開いたファイルを開く」「ファイルキャッシュから開く」「locateの出力結果から開く」などが別々のコマンドとして存在しているからです。しかし、どれも「ファイルを開く」ということには変わりありません。ファイルを指定する前に特定のコマンドを指定する意味でも手続型です。

これらを手続型の疑似コードで書くと「open_from_recent_file(a_recent_file)」「open_from_file_cache(a_file_in_file_cache)」などになります。どう見ても同じ働きをする型違いの関数ですよね。

オブジェクト指向になると「a_recent_file.open()」「a_file_in_file_cache.open()」となります。a_recent_fileもa_file_in_file_cacheもファイル名ですので、ファイルを開くopenメソッドは共

通のものが使えることになります。

つまり、何らかの手段で最近開いたファイルやファイルキャッシュから1つのファイルを選び出すことができれば、Emacsでもオブジェクト指向を実現できることになります。これを行うのがhelmです。これでもなお、helmのたった一面しか説明できていませんが……。

「最近開いたファイルの集合」「ファイルキャッシュ全体」などはオブジェクト指向におけるクラスに相当しますが、helmでは「情報源（source）」といいます。

多くのアクション

ファイルをどのように開くかということに関しても、いろいろな方法があります。普通に「現在のウィンドウで開く」以外にも「別ウィンドウで開く」や「別フレームで開く」などがあります。最近開いたファイルにおける疑似コードで「open_from_recent_file_other_window(a_recent_file)」や「open_from_recent_file_other_frame(a_recent_file)」などと用意するのではきりがありません。そうするくらいならば「a_recent_file.open_other_window()」などのように新しい「メソッド」を定義するでしょう。

ファイルに関するメソッドにはファイルを開くだけでなく、ファイルの内容をバッファに挿入したり、ファイルを削除したりなど、たくさんあります。この、オブジェクト指向におけるメソッドをhelmでは「アクション」といいます。

各情報源には複数のアクションが定義されていることが多いのですが、一番よく使われるであろうアクションが優先度最上位のデフォルトアクションになります。バッファやファイルでは、開くにあたります。

絞り込み検索

つまり、用意された情報源から1つの候補、アクションリストから1つのアクションをすばやく選択することさえできれば、Emacsのオブジェクト指向化は成されることになります。オブジェクト指向化が成されれば「最近開いたファイルを別ウィンドウで開く」などEmacs標準では用意されていないことが簡単にできるようになります。なぜなら「『最近開いたファイル』から1つのファイルを選択」し「『別ウィンドウで開く』というアクションを選択」すれば良いのですから。

そのときに超重要になるのが候補選択です。helmの使い勝手は候補選択1つに大きく左右されます。これは、情報源においてだけでなく、アクションリストにおいても使われるからです。

候補選択においてはスペースで区切った正規表現を複数個指定して絞り込み検索ができます。たとえば「org el」でorgとel双方にマッチする候補がリストアップされます。一方「!」を指定すればマッチしない正規表現になります。「org el !dblock」ですと、orgとelにマッチするがdblockにマッチしないものが対象になります。

デフォルトアクションは候補選択後すぐにRETで実行できます。よって、バッファを開くのであればバッファの候補選択をしてRETと、操作感覚がido（2014年1月号で紹介）とあまり変わりません。

この絞り込み検索機能のおかげでhelmは世界中から熱狂的な支持を得ています。しかしこれでも、まだhelmの半分しか説明できていないのが恐ろしいところです。

ひとまとめにされた情報源

helmが「Emacsの舵」と言えるのは、複数個の情報源を串刺し検索できてしまうところにあります。情報源は何の集合であってもかまいません。バッファ名とファイル名が混在していてもかまいません。たとえば、開いているバッファと最近開いたファイルから「helm」が含まれているものをまとめて検索できます。

このおかげで、バッファを切り替えるコマンドと最近開いたファイルを開くコマンドを別個に割り当てる必要はなく、1つのhelmコマンドでまかなえてしまいます。最近開いたバッファを間違えて削除してしまった場合にも、今度は

自動的に最近開いたファイルから拾い出せます。

つまり、複数の情報源から目的のオブジェクトとやりたいアクションを絞り込み検索するところにhelmの凄味があるのです。現代の潤沢なマシンパワーによる強引な力業です。

anythingからhelmへ

helmの前身はEmacs界に新たな流れを呼び込んだanythingでした。anything時代から前節で説明したことはすでに達成されていました。今日ではanythingに関する情報もたくさん見つかるのでhelmを理解するにあたっては大いに役立ちます。また、anythingとhelmは共存させられます。それでは、anythingからの変化を見ていきましょう。

anything時代

anythingは2007年にTamas Patrovics氏が開発したものです。筆者はこれにものすごく惚れ込み、2008年にはTamas氏より開発を引き継ぎました。そして、絞り込み検索機能(anything-match-plugin)、Migemo対応(anything-migemo)、候補バッファの導入(candidates-in-buffer)、チラ見機能(persistent-action)、独立したanythingコマンド(M-x anything-miniなど)など、今のhelmの核となる機能を実装しました。anythingは2008年の時点で、すでに機能として完成されていました。

2008年の2月には本誌のEmacs特集でanythingの解説も書きましたし、2010年の「Emacsテクニックバイブル〜作業効率をカイゼンする200の技〜」、2011年の「Emacs Lispテクニックバイブル〜真髄を知るLispの掟〜」では独立した章を設けてanythingを解説しました。

anythingの情報源がすべて入っているanything-config.elは膨大な量となりました。候補選択といえばanything、という流れになり、たくさんの人によって数えきれないほどの情報源・anythingコマンドが書かれました。

フォークからリネーム！

anythingが活発だった時代、開発は筆者とThierry Volpiatto氏の2人で行っていましたが、2人の意見は正反対でした。そのころにはanythingは十分成熟していたと筆者は感じていたので、筆者は互換性を最優先し、変化に対しては極めて慎重派でした。対してThierry氏はanythingを抜本から変えたいという急進派で、名前をanythingからhelmという名前に変更して開発を続けると言い出しました。

彼の熱意で始まったhelmはどんどんと抜本的なコード改造がなされ、基本的にはanythingでありながらも、さまざまな機能が追加されるようになりました。helmインターフェースも数多くMELPAに登録されました。時代はanythingからhelmに取って代わったのです。

彼は熱狂的な開発者で、ものすごい勢いでコードを書いています。しかし、突然操作法を変更することを平気でやってしまいます。そのため、anythingから移行する際、あるいはhelmを更新する際に戸惑う可能性があります。helmの内部は複雑であり、解説も英語であるため、Emacs熟練者でなければ問題解決できないこともあります。その点については気づいた限り筆者のサイトでお伝えすることにしています。

一方、筆者は今なおanythingも使いメンテナンスしていますが、ここ数年は大きな変更はありません。今後のEmacsでも動作させるようにはします。細かい点がいろいろと変更されたhelmよりも昔ながらのanythingが好きと言ってくれる人がいてくれるのはありがたいことです。

新旧入れ交じった様相はかつての「Ruby 1.8 vs. Ruby 1.9」、今の「Perl 5 vs. Perl 6」「Python 2 vs. Python 3」を思わせます。

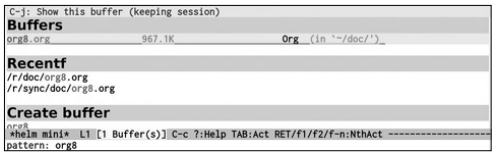
再構成されたファイル群

anything時代は本体(anything.el)と設定集(anything-config.el)の2つのファイルで構成されていました。しかし、anything-config.elが肥

▼リスト1　パッケージを使うための初期設定

```
(package-initialize)
(add-to-list 'package-archives '("marmalade" . "http://marmalade-repo.org/packages/"))
(add-to-list 'package-archives '("melpa" . "http://melpa.org/packages/") t)
```

▼図1　M-x helm-mini でorg8と入力したところ



大化していったため、見通しが悪くなってきました。そこでhelmでは目的別に多数のファイルに分割されました。

- helm.el：本体
- helm-config.el：キーの設定、その他
- helm-mode.el：helmを用いた補完
- helm-match-plugin.el：絞り込み検索
- helm-help.el：ヘルプ機能
- helm-source.el：情報源を定義するときに使う
- helm-utils.el：ユーティリティ関数
- その他：各パッケージに関する情報源を定義

helmはとても活発に開発されています。helmパッケージには数多くのhelmコマンドが定義されているので、あなたがほしい情報源・helmコマンドはすでに存在するかもしれません。興味ある方はお宝探索気分でM-x grepなどでhelmのソースコードを検索してみるといいでしょう。

helm-miniに見るhelmの力

helmをインストールしたら真っ先に使っていただきたいコマンドがM-x helm-miniです。名前のとおり最小構成のhelmコマンドですが、これをくわしく知るだけでもhelmの威力をまざまざと感じられます。

helmのインストール

まずはhelmをインストールしてみましょう。helmはMELPAに登録されています。**リスト1**で設定し、

```
M-x package-refresh-contents
M-x package-install helm
```

でhelmをインストールしましょう。

情報源はバッファとrecentfだけ！

M-x helm-miniの情報源は「開いているバッファ」「最近開いたファイル」「バッファ新規作成」の3つだけです（**図1**）。そもそもファイル以外のバッファを作成すること自体が少ないので、実質前者の2つのみです。

バッファの切り替えを効率的にすることは、Emacsライフにおいて大きな意味を持ちます。なるべく少ない打鍵数で目的のバッファに切り替えられるとストレスが少なくなります。そこでhelmの絞り込み検索がとても役立ちます。同種の目的を持つものにidoがありますが、絞り込み検索ができる点、そしてバッファに対する多数のアクションが用意されている点でhelmが勝っています。

一方、最近開いたファイルはrecentfパッケージとなっています。使ってみると、過去に開いたファイルを簡単に開けます。新規ファイル以外では過去に開いたファイルを開くことがとても多いので有用性は実感できるはずです。

バッファを削除してしまった場合でも、今度は最近開いたファイルから見つけ出してくれます。よって、recentfはバッファ検索で見つからないときの保険になってくれます。

37のアクション

helmはオブジェクトのタイプによって多くのアクションを持っていることが特徴です。M-x

helm-miniはバッファとファイルを扱うので、それらのアクションを実行できます。

執筆時点で、バッファのアクションは17個、ファイルのアクションは20個用意してあるので、M-x helm-miniは実に37個の機能を持っていることになります。しかも開いているバッファと最近開いたファイルは同時に検索できるので単に37個のコマンドがバラバラに用意されているのに比べて圧倒的に便利です。

バッファのアクション

M-x helm-miniを起動すると、即座にバッファのリストが見えます。そこで Tab を押すと**リスト2**のようにアクションが羅列します。

アクションのうち一番上にあるのがデフォルトアクション(この場合Switch to buffer)です。アクションリストを開かずとも候補をRETで選択すると実行されます。

ほかにも目立つのがファンクションキーとアクションに書いてあるキーバインドです。たとえば、アクション「[f3] Switch to buffer other window `C-c o'」はアクションリストを開いても開かなくても F3 かC-c oで実行できます。

このことは、デフォルトアクション以外を実行する場合でも、キーバインドさえ覚えていればアクションリストを開かなくてもすぐに実行できることを意味します。helmを使い始めたばかりのころではアクションリストを開いて選択していたけど、慣れてくると候補選択後に特定のキーを押せばそのアクションが実行できます。

各情報源によって使えるキーバインドは異なりますので、C-c ?で見られるヘルプを参照してください。

アクションをざっと見てみると、開き方だけでも多種多様であることがわかります。[f2]のpopup windowとはpopwin.elがインストールされているときに使えます。[f5]のElscreenとはelscreen.elがインストールされているときに新しいウィンドウ構成としてそのバッファを開きます。[f8]ではview-modeで開き、[f9]ではバッファを選択しないで表示だけします。

外部ライブラリに依存するアクションは、ライブラリがインストールされていない場合には表示されません。そのため、環境によってはファンクションキーがずれることがあります。

「(s)」と末尾に付くアクションは複数形も受け付けるようになっています。これは、候補をC-SPCでマークすることで、マークしたバッファすべてにおいて実行するアクションです。diredでファイル操作するとき、マークなしのときはカーソル位置、マーク有りのときはマークされたファイルが対象になるのと同じです。たとえば一連のバッファを削除したい場合は、削除したいバッファをマークしてM-Dを押せばいいのです。

ほかにも置換(Query replace)や検索(Grep、Multi occur)、diffが取れたりもします。マークすれば複数のバッファに渡る置換や検索ができます。

ediffはEmacs屈指の人気機能でdiffをわかりやすく示してくれます。2つのバッファをマークしたらそれらの比較をし、マークがない場合はカレントバッファとの比較になります。ediff mergeは、2つのバッファをマージしたバッファを作成します。ediffはそのうち記事にする予定です。

このようにアクションは本当にたくさん用意されていますが、わざわざ覚える必要はありません。そういえばこんなことができたなと頭の片隅に置いてさえすれば、 Tab でアクションリストを開き、該当アクションを選択すれば良いだけです。

ファイルのアクション

ファイルを選択して Tab を押すと、ファイルのアクションが現れます(**リスト3**)。

ファイルを開くアクションにはいろいろあることがわかります。一般ユーザでは読み書き不能なファイルはroot権限で開けます。別ウィン

▼リスト2　バッファのアクション

```
[f1]  Switch to buffer；バッファを開く
[f2]  Switch to buffer in popup window；ポップアップウィンドウで開く
[f3]  Switch to buffer other window `C-c o'；別ウィンドウで開く
[f4]  Switch to buffer other frame `C-c C-o' ；別フレームで開く
[f5]  Display buffer in Elscreen ；新しいelscreenを作成する
[f6]  Query replace regexp `C-M-%'；正規表現置換（複数可）
[f7]  Query replace `M-%' ；置換（複数可）
[f8]  View buffer；view-modeで開く
[f9]  Display buffer ；表示するだけでウィンドウを選択しない
[f10] Grep buffers `M-g s' (C-u grep all buffers) ；grepする（複数可）
[f11] Multi occur buffer(s) `C-s'  ；multi-occur（elisp版grep）する（複数可
[f12] Revert buffer(s) `M-U'  ；再読み込み（複数可）
      Insert buffer；バッファを挿入する
      Kill buffer(s) `M-D'  ；バッファを削除する（複数可）
      Diff with file `C-='  ；バッファと保存されているファイルをdiffする（未保存時のみ）
      Ediff Marked buffers `C-c ='；2つのバッファのediffを取る
      Ediff Merge marked buffers `M-='；2つのバッファをマージしたバッファを作成する
```

ドウ・別フレームで開いたり、そのファイルが収められているディレクトリをdiredで開いたり、view-modeで開いたりできます。最後のhex dumpというのはバイナリファイルを16進ダンプで開くhexl-find-fileを起動します。

バッファ同様、ファイルに対してもediffが使えます。org-modeのリンクを挿入するアクションは、org-modeファンにとってうれしい限りです。

C-zでチラ見

helmでC-zを押すとpersistent-actionを実行します。これはいわゆるチラ見機能で、helmを終了することなくそのオブジェクトの内容を見られます。ファイルやバッファならばそれを開き、関数や変数ならばその説明を表示します。絞り込んだオブジェクトの内容をひとつひとつ見たい場合に便利です。

スクロールはあたかも別ウィンドウを操作するようにC-M-vとC-M-S-vでできます。

チラ見するだけでhelmを終了するには、C-gを押します。ウィンドウ構成がhelm起動時のものに戻ります。

まとめ

今回は最小構成のM-x helm-miniを通してhelmの概念を大まかに解説しました。helmコマンドはそれこそ無数に定義されていますが、ファイルやバッファを開くのであればM-x helm-miniで8割方間に合うでしょう。このコマンドで使われているバッファの絞り込みは独特のものになっているのですが、残念ながら誌面の都合上、来月に回すことにいたしました。来月はほかのhelmコマンドの解説もしていきますので楽しみにしててください。

筆者のサイトhttp://rubikitch.com/はEmacs総合サイトを目指して日々更新しています。helmについての記事も多数書いてあります。SD

▼リスト3　ファイルのアクション

```
[f1]  Find file
[f2]  Find file as root
[f3]  Find file other window
[f4]  Find file other frame
[f5]  Open dired in file's directory
[f6]  Grep File(s) `C-u recurse'
[f7]  Zgrep File(s) `C-u Recurse'
[f8]  Pdfgrep File(s)
[f9]  Insert as org link
[f10] Checksum File
[f11] Ediff File
[f12] Ediff Merge File
      Etags `M-., C-u tap, C-u C-u ➥
      reload tag file'
      View file
      Insert file
      Add marked files to file-cache
      Delete file(s)
      Open file externally (C-u to choose)
      Open file with default tool
      Find file in hex dump
```

シェルスクリプトではじめるAWS入門

――AWS APIの活用と実践

第10回　AWS APIでのデジタル署名の全体像を明らかにする④

Writer　波田野 裕一（はたの ひろかず）／運用設計ラボ　operation@office.operation-lab.co.jp

トピック「AWS CLIに新サービス登場」

新サービス登場

皆さんもご存じとは思いますが、2014年11月11日から14日にラスベガスでAWS re:Inventというで最大規模のイベントが開催されました。このイベントの場で、AWS Key Management Service（KMS）、AWS CodeDeploy、AWS Lambdaなど9つの新サービスが公開されました。このうち、いくつかはプレビュー版のものや、2015年リリースのものがありましたが、同時期にリリースされたAWS CLI 1.6.0と1.6.1で、次のサービスがさっそく使えるようになっていました。

・kmsコマンド：AWS Key Management Service
・deployコマンド：AWS CodeDeploy
・configserviceコマンド：AWS Config（プレビュー版）
・lambdaコマンド：AWS Lambda

さらに、年明けすぐの2015年1月9日にリリースされたAWS CLI 1.7.0では次のサービスに対応しました。

・cloudhsmコマンド：AWS CloudHSM[注1]
・ecsコマンド：Amazon EC2 Container Service
・glacierコマンド：Amazon Glacier

これにより、AWS CLIは40近いサービスに対応することとなり、むしろ対応していないサービスを探すほうが難しくなってきました。個人的には、CloudFrontへの正式対応を引き続き心待ちにしているところです。

JAWS DAYS 2015

すでに登録受付を開始していますが、一昨年、昨年に引き続きJAWS DAYS 2015が開催[注2]されることになりました。このイベントは、全国レベルでのAWSユーザの交流を目的としてJAWS-UG（AWS User Group – Japan）が主催するもので、今年が3回目になります。次の7トラックが企画されており、終日AWS三昧の1日となる予定です。筆者も初心者ハンズオンのお手伝いをする予定です。ぜひ参加ください。

・ビッグトラック
・AWS Technical Deep Dive & ACEに聞け！
・テクニカルトラック（事例紹介）
・DevOpsトラック
・モバイル ビッグデータ IoT
・HackDay
・初心者ハンズオン

注1）残念ながら東京リージョンではサービスが提供されていません。

注2）URL http://jawsdays2015.jaws-ug.jp

今回の流れ

連載第7回から、AWS APIを直接操作するために必要な次の3種類のデジタル署名について解説しています。

・Signature Version 2（第9回、今回）
・Signature Version 3（第7回、第8回）
・Signature Version 4

前回は、Signature Version 2の概要について解説しました。今回は、Signature Version 2の手順をシェルスクリプト化したときのサンプルとその実行例について解説します。

Signature Version 2

シェルスクリプト（Signature Version 2）

Signagure Version 2の署名付きリクエストデータの作成手順をシェルスクリプトにすると**リスト1**のようになります。

このスクリプトでも、Signature Version 3のときと同様に、認証情報についてデフォルトで~/.aws/default.rcを参照しますが、cオプションで別の認証ファイルを指定できるようになっています。

▼リスト1　aws-v2-get.sh

```
#!/bin/sh

# get auth config (-c)
while getopts c: option
do
  case $option in
    c)
      FILE_AUTH=${OPTARG}
  esac
done
shift `expr ${OPTIND} - 1`

# read auth file
if [ "${FILE_AUTH}x" = "x" ];then
  . ${HOME}/.aws/default.rc
else
  . ${FILE_AUTH}
fi

# define files
FILE_TMP_REQ=${HOME}/tmp/aws-v2-request.tmp
FILE_TMP_S2S=${HOME}/tmp/aws-v2-s2s.tmp
FILE_OUTPUT=${HOME}/tmp/aws-v2-output.txt

# env-v2.txt
SIGNATURE_METHOD='HmacSHA256'
SIGNATURE_VERSION='2'
API_VERSION='2013-10-15'
TIMESTAMP=`date -u +%Y-%m-%dT%H:%M:%SZ`

# 1. make request data
ENDPOINT=$1
FILE_REQ=$2

if [ "${FILE_REQ}x" = 'x' ]; then
  echo "Usage: $0 endpoint file-request"
  exit 2
fi

cp ${FILE_REQ} ${FILE_TMP_REQ}
{
  echo "AWSAccessKeyId=${aws_access_key_id}"
  echo "SignatureMethod=${SIGNATURE_METHOD}"
  echo "SignatureVersion=${SIGNATURE_VERSION}"
  echo "Timestamp=${TIMESTAMP}"
  echo "Version=${API_VERSION}"
} >>${FILE_TMP_REQ}

# 2. create the String to Sign.
MSG=`
  cat ${FILE_TMP_REQ} |\
  sort                |\
  sed -e 's| |%20|g'  |\
  sed -e 's|(|%28|g'  |\
  sed -e 's|)|%29|g'  |\
  sed -e 's|\*|%2A|g' |\
  sed -e 's|+|%2B|g'  |\
  sed -e 's|/|%2F|g'  |\
  sed -e 's|:|%3A|g'  |\
  sed -e 's| |+|g'    |\
  sed 's/^/\&/'       |\
  sed '1s/^\&//'      |\
  tr -d '\n'`

{
  echo 'GET'
  echo ${ENDPOINT}
  echo '/'
} >${FILE_TMP_S2S}

if [ "${OSTYPE}x" = 'darwinx' ];then
  echo "${MSG}¥c" >>${FILE_TMP_S2S}
```

※次ページに続く

▼リスト1　aws-v2-get.sh(続き)

```
else
  echo -n "${MSG}" >>${FILE_TMP_S2S}
fi

# 3. Calculate the Signature.
SIGNATURE=`openssl dgst -binary -hmac \
  ${aws_secret_access_key} -sha256 ➥
${FILE_TMP_S2S} |\
  base64 |\
  sed -e 's|+|%2B|g' | sed -e 's|/|%2F➥
|g' | sed -e 's|=|%3D|g'`

# 4. output request data

# 4.1. for browser
RESOURCE="?${MSG}&Signature=${SIGNATURE}"
echo "\nhttp://${ENDPOINT}/${RESOURCE}\n"

# 4.2. for OpenSSL sclient
{
  echo "GET /${RESOURCE} HTTP/1.1"
  echo "Host: ${ENDPOINT}"
} >${FILE_OUTPUT}
cat ${FILE_OUTPUT}
```

VPCの一覧を取得してみる

前回の手順1で作成したVPCの一覧を取得するリクエストデータ(**リスト2**：vpc-describe-vpcs-query.txt)を**リスト1**(aws-v2-get.sh)で署名付きリクエストデータに変換してみましょう。

リージョンは前回と同様にus-east-1を指定します(EC2のAPIエンドポイントはec2.us-east-1.amazonaws.comになります)。コマンドは次のように入力します。

```
$ EC2_ENDPOINT='ec2.us-east-1.amazonaws.➥
com'
$ ./aws-v2-get.sh ${EC2_ENDPOINT} vpc-➥
describe-vpcs-query.txt
```

Signature Version 2の署名付きのリクエストデータが出力されます(**リスト3**)。

リスト3の出力結果のうち、http://ではじまる1行のデータはブラウザ用のリクエストデータです。GETメソッドの場合は、ブラウザのURL欄に貼り付けて実行できます。

後半2行のデータは生のHTTPリクエストデータです。opensslコマンドを立ち上げて貼り付けてみましょう。コマンドは次のように入力します。

```
$ openssl s_client -connect ${EC2_➥
ENDPOINT}:443

  (略)
    Timeout   : 300 (sec)
    Verify return code: 20 (unable to ➥
get local issuer certificate)
  入力待ち(ここに署名付きリクエストデータの後半2行を貼り付けて、
  2回[Enter]キーを押してください)
```

いずれの実行方法でも、**リスト4**のようなXMLが表示されるはずです。この例では、us-east-1にはVPCが1つだけ存在していることがわかります。

EC2インスタンスを起動／停止してみる

もう少し結果がわかりやすい例として、EC2インスタンスの起動／停止をしてみましょう[注3]。

注3) 1年間の無料利用枠の範囲での作業ですが、環境によっては課金が発生する場合があります。利用しているアカウントの無料利用枠について必ず確認しておいてください。

▼リスト2　vpc-describe-vpcs-query.txt

```
Action=DescribeVpcs
```

▼リスト3　リクエストデータ(サンプル)

```
http://ec2.us-east-1.amazonaws.com/?AWSAccessKeyId=AKIDEXAMPLE&Action=DescribeVpcs&Signature➥
Method=HmacSHA256&SignatureVersion=2&Timestamp=2015-02-03T01%3A12%3A32Z&Version=2013-10-➥
15&Signature=BqzAw0wmERkrt%2BNPR2aECaU20T81rwnI5nNd3r76ryo%3D

GET /?AWSAccessKeyId=AKIDEXAMPLE&Action=DescribeVpcs&SignatureMethod=HmacSHA256&Signature➥
Version=2&Timestamp=2015-02-03T01%3A12%3A32Z&Version=2013-10-15&Signature=BqzAw0wmERkrt%2BNP➥
R2aECaU20T81rwnI5nNd3r76ryo%3D HTTP/1.1
Host: ec2.us-east-1.amazonaws.com
```

▼リスト4　実行結果(例)

```
<DescribeVpcsResponse xmlns="http://ec2.amazonaws.com/doc/2013-10-15/">
    <requestId>f90e7a79-80a4-4409-a85b-f0e2a168067c</requestId>
    <vpcSet>
        <item>
            <vpcId>vpc-0a77f66f</vpcId>
            <state>available</state>
            <cidrBlock>172.31.0.0/16</cidrBlock>
            <dhcpOptionsId>dopt-f4726c86</dhcpOptionsId>
            <instanceTenancy>default</instanceTenancy>
            <isDefault>true</isDefault>
        </item>
    </vpcSet>
</DescribeVpcsResponse>
```

誌面の都合もありますので、今回はEC2インスタンスの起動と停止だけを行います。EC2インスタンスにSSHログインする必要がある場合は、あらかじめ以下の作業をしたうえでEC2インスタンス起動時にキーペア、VPCID、セキュリティグループIDを指定してください。

①EC2のキーペア作成
②VPCのセキュリティグループ作成(SSH通信を許可)

手順①「EC2インスタンスの起動」

最初にEC2インスタンスの起動に必要なリクエストデータを作成します。EC2インスタンスを起動するには、"RunInstances"アクションを使います。

RunInstancesアクションのパラメータ情報はAWSのドキュメントページ[注4]に記載されています。RunInstancesアクションでは3つの必須パラメータ(ImageId、MaxCount、MinCount)を指定する必要があります。このうちImageIdは、EC2インスタンスの起動に使用するAMI(Amazon MachineImage)のImage IDを指定します。今回は、起動するインスタンスタイプにt2.microを利用するため、HVMタイプのAMIを指定する必要があります。また、同じAMIでもImage IDはリージョンごとに異なることに注意してください。

今回は、t2.microタイプのインスタンスで利用できるAmazon Linuxの最新版AMI(2015年1月時点は'2014.09.1')を利用します。

AWS CLIでは、次コマンドの入力でImage IDを取得できます。

```
$ AWS_REGION='us-east-1'
$ EC2_AMI_VERSION='2014.09.1'
$ EC2_AMI_ID=`aws ec2 describe-images
--region ${AWS_REGION} --owners amazon
--filters Name=description,Values
="Amazon Linux AMI ${EC2_AMI_VERSION}
x86_64 HVM EBS" --query 'Images[].
ImageId' --output text`; echo ${EC2_
AMI_ID}
```

必須のパラメータではありませんが、想定外の課金を避けるためにInstanceTypeも必ず指定するようにしましょう。

EC2インスタンスを起動するために必要なデータは**リスト5**のような内容になります。

作成した**リスト5**を、aws-v2-get.shで署名付きのリクエストデータに変換します。実行コマンド例は次のようになります。

```
$ ./aws-v2-get.sh ${EC2_ENDPOINT} ec2-
run-instances.txt
```

▼リスト5　ec2-run-instances.txt

```
Action=RunInstances
ImageId=ami-b66ed3de
MaxCount=1
MinCount=1
InstanceType=t2.micro
```

注4) URL http://docs.aws.amazon.com/ja_jp/AWSEC2/latest/APIReference/API_RunInstances.html

出力された結果のうち、'http://'で始まるブラウザ用のリクエストデータをブラウザのURL欄に貼り付けて、実行してみましょう。

実行結果で表示されるXMLの6行目付近に、起動したEC2インスタンスのInstance IDが表示されているはずです。これをコピーして保存しておきます。例としてXMLに下記の表示があるとします。

```
<instanceId>i-096b1f26</instanceId>
```

EC2マネジメントコンソールにアクセスして、EC2インスタンスが起動していることを確認してみましょう(**図1**)。Instance ID欄に先ほどメモしたInstance IDと同じものが表示されていることも併せて確認しておきましょう。

手順②「EC2インスタンスの廃棄」

次に、EC2インスタンスを停止するリクエストデータを作成します。EC2インスタンスを停止するだけでなく解約するには、"Terminate Instances"アクションを使います。

このアクションでは、InstanceIdだけが必須パラメータとなります。 このInstanceIdには、先ほどメモしておいたInstance IDを記入します。

EC2インスタンスの停止と解約をするために必要なデータは次のような内容になります。

```
Action=TerminateInstances
InstanceId=i-096b1f26
```

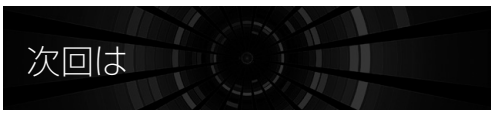
作成したリクエストデータを、aws-v2-get.shで署名付きのリクエストデータに変換します。例として次のようにコマンド入力します。

```
$ ./aws-v2-get.sh ${EC2_ENDPOINT} ec2-terminate-instances.txt
```

出力された結果のうち、後ろの2行をコピーし、openssl コマンドを立ち上げて貼り付けてみましょう。

```
$ openssl s_client -connect ${EC2_ENDPOINT}:443
…(略)…
    Timeout   : 300 (sec)
    Verify return code: 20 (unable to get local issuer certificate)
  入力待ち(ここに署名付きリクエストデータの後半2行を貼り付けて、2回 Enter キーを押してください)
```

EC2マネジメントコンソール上で、EC2インスタンスのステータスが"Terminated"になっていることを確認しましょう。実行結果は**図2**のようになります。ブラウザで確認した画面は**図3**です。



AWSの3種類のデジタル署名のうち、最後のSignature Version 4の概要と実例について解説予定です。SD

▼図1　EC2インスタンスの起動確認

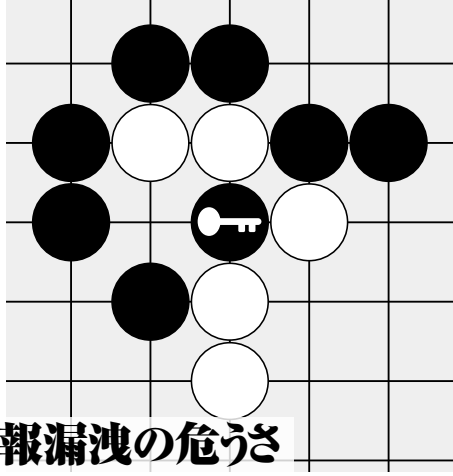
▼図3　EC2インスタンスの終了確認

▼図2　実行結果例(TerminateInstances)

```
HTTP/1.1 200 OK
Content-Type: text/xml;charset=UTF-8
Transfer-Encoding: chunked
Vary: Accept-Encoding
Date: Fri, 16 Jan 2015 08:29:41 GMT
Server: AmazonEC2

248
   <?xml version="1.0" encoding="UTF-8"?>
<TerminateInstancesResponse xmlns="http://ec2.amazonaws.com/doc/2013-10-15/">
    <requestId>72a42232-f804-4107-9015-3df3a83d0da0</requestId>
    <instancesSet>
        <item>
            <instanceId>i-096b1f26</instanceId>
            <currentState>
                <code>32</code>
                <name>shutting-down</name>
            </currentState>
            <previousState>
                <code>16</code>
                <name>running</name>
            </previousState>
        </item>
    </instancesSet>
</TerminateInstancesResponse>
0
```

セキュリティ実践の基本定石

すずきひろのぶ
suzuki.hironobu@gmail.com



みんなでもう一度見つめなおそう

【第十八回】家電化した情報機器が持つ情報漏洩の危うさ

最近では、あらゆる機器がインターネットに接続できるようになってきています。その中には、十分なセキュリティ対策が施されないまま外部に公開されているものもあります。今やそのような機器を見つけ出すのは簡単です。その象徴的な存在が「SHODAN」というWebサービスです。SHODANを紹介することで、いかにインターネットに接続された機器が無防備であるかを説明していきます。

はじめに

首都大学東京南大沢キャンパスの教務課事務室内に設置されたネットワーク接続で使われるストレージ・NAS（Network Attached Storage）が外部から匿名ftpでアクセス可能になっており、延べ人数で約5万1千人分の個人情報データが5ヵ月間外部にさらされていたという事故がありました[注1]。これらの問題をIoT（Internet of Things）のセキュリティという観点から考えていきたいと思います。

情報機器の普及の影に潜む問題

最近では、家庭でNASが使われるケースも珍しくありません。パソコンの外部ストレージとして使うだけではなく、ネットワーク・オーディオやHDDレコーダーといったデジタル家電のための外部ストレージとしても利用されています。

NASを提供しているベンダーのWebサイトにあるカタログを見てみると、普及品レベルのものでも、最小で1TBの容量からラインナップがそろっています。つい20年ほど前まで、1TBの容量を持つネットワーク接続された外部記憶システムは、大学や研究所の計算機センターにしかなく、極めて高価で特殊な、そして専門のオペレーターが管理するような機材でした。それが今日では、一般のユーザが購入し、ネットワークに接続して簡単に利用できる時代になっています。

首都大学東京の事件の場合、外部ネットワークからアクセス可能であったことが、最大の問題です。しかし、もう1つの大きな問題として、同大学のシステム管理者の認識の甘さがあったのではないでしょうか。認証を必要としない誰でもアクセスできる匿名ftp（anonymous ftp）アカウント[注2]を危険だとは思わず、「手間のかからない便利なアクセス方法だ」ぐらいに認識していたのではないかと筆者は危惧します。

同校のNASには学内／学外の学業成績などを含む大量の情報が置かれていました。インターネット時代における個人情報云々と議論する以前に、成績などはどんな時代でも扱いに慎重を期するべき性質のものです。紙の時代であれば、5万1千人分の成績や住所録が流出することは、物理的な意味でも、手続き的な運用の意味でも大量過ぎて極めて難しいことだったと思われます。それが今では、ちょっとした間違い、勘違い、あるいは「とりあえず便利なNASだから使ってみた」レベルのことで、全部が流

注1）「首都大学東京における個人情報を含むNASに対する外部からのアクセスについて<お詫び>」
http://www.tmu.ac.jp/news/topics/8448.html?d=assets/files/download/news/press_150119.pdf

注2）通常、工場出荷時にanonymous ftpは有効になっていません。今回の事案を受けて、工場出荷時にanonymous ftpが有効になっている機種があるかどうかを調べてみたのですが、筆者が調べることのできた範囲では見つけることができませんでした。

出する危険に晒されます。これでは、さすがにたまったものではありません。

この問題は、情報自体の取り扱いのポリシーや情報システムの取り扱いのポリシーの欠如が引き起こした結果とも言えるわけですが、一方で、技術的な面から考えてみると、IoT時代のセキュリティの問題が徐々に現れてきた面もあるのではないか、と筆者は考えています。

IoT時代のセキュリティ問題

内部ネットワークからのアクセスしか意識していなかったNASが、外部からアクセスされたという部分にフォーカスして考えてみます。

現状でも、ネットワークサーバ機能を持つHDDレコーダーやデジタル音楽機器、ネットワーク経由で接続できるビル内監視カメラ、内線向けIP電話、複写機やプリンタでも、同様の問題が発生するでしょう。

さらに時代が進めば、スマートフォン経由でコントロールできるエアコンなどといった、これまでの情報家電ではなく、白物家電の性格を持ったものもネットワーク化、情報化が進むことが予想されます。そういう時代になれば、問題はどんどん拡大していくかもしれません。その可能性は極めて高いと思います。

みなさんの多くは、「悪意のある第三者がなぜ、インターネットという広大なネットワーク空間の中から、使っている本人ですら気づかない情報家電のような隠れた機器を見つけられるのか」と不思議に思うかもしれません。

そこで今回は、IoT時代のセキュリティの基礎知識として、ぜひ知っておきたいWebサイト「SHODAN」を紹介します。

SHODANとは

SHODANはインターネットに接続しているサーバやクライアントといったコンピュータだけではなく、ルータなどのネットワーク機材、IoT機器と言

◉どんなセキュリティの問題が予想されるのか

■プリンタ、スキャナー、FAXなどの複合機

現在では、これらはいったん、データの形で機材の中に取り込まれ、保存され、必要に応じて、出力をしたり、あるいはファイルを転送したりします。つまり、単なる印刷機械ではなく、ファイルサーバと一体化している情報機器だと言えます。

外部からアクセス可能であれば、機密情報が外部に漏れ出してしまう可能性があります。しかも、通常のファイルサーバやデータベースサーバとは違い、この手の機材では（印刷機器と認識されていますから）、セキュリティのための監査ログは不十分であり、外部から不正にアクセスされ情報が漏洩した場合、何が起こっていたのかを、あとで調べるのはたいへんだと思われます。

■ネットワーク接続監視カメラ

インターネット経由で監視映像を送ることができるカメラで、Webインターフェースで設定を変更するようなものも、今や特別ではありません。Webインターフェースで操作や設定をする際には、勝手に第三者に悪用されないように、当然ながら認証を用意しています。

過去の例では、認証回避の脆弱性が存在し、第三者が任意の操作ができるというケースもありました。そうなれば、監視をするという本来の重要な役割が果たせなくなります。

■プロキシ機能が搭載されている機材

情報家電などで設定やユーザインターフェースのために、httpサーバやそれに関連するhttpプロキシ機能を搭載しているものがあります。

第三者から無制限に匿名プロキシを受け付けるような設定であった場合、この匿名プロキシを踏み台にして、外部のWebサーバにアクセスするなど悪用されてしまいます。

■ネットワーク同期時計

ntpdを搭載したネットワーク時間同期対応時計が存在していて、monlist機能を使ったDDoS攻撃の踏み台となったならば……。

われる情報家電や組込み機材、あるいは制御系システムなど、(意図的であるか否かを問わず)外部に公開されている接続をスキャンしデータベース化して、それらの情報をユーザに提供しているサイトです(**図1**)。

Googleはインターネット上のコンテンツを走査し、データベース化し、ユーザからの検索というリクエストに対し、検索結果を提供します。今や私たちはGoogleのような検索サイトがないと、インターネットのどこにどんな情報があるのかがわからず、右往左往してしまいます。

SHODANはインターネットに接続している機器やサービスを探す検索サイトと言えるでしょう。別の言い方をすると、脆弱性を持つサイトやサービスすらも検索できる便利なサイトと言えます。

SHODANをもうちょっと具体的に説明しましょう。SHODANのボットはインターネットという広大な空間に接続されて、公開されている通信ポートに対しリクエストを送ります。通常、サービスを提供するサーバプログラムは規約にそった情報を返します。それをSHODANではバナーと呼んでいます。

SHODANは延々とその情報を蓄えていきます。かくして、膨大なデータが記録されます。その膨大な情報から必要に応じたフィルタをかけたうえで検索すると、インターネット上にどんな機材が稼動しているのか、あるいはサービスが提供されているのかがわかります。

簡単に得られるサーバなどの情報

まずは、SHODANでバナーと呼んでいるものが、どんなものかを見てみましょう。ここではhttp、ftp、telnetのサーバのレスポンスを観察してみたいと思います。

VirtualBoxでDebian GNU/Linux 7(Wheezy)をローカルに動かし、その上にhttpd、ftpd、telnetdを稼動させました。現在では、ftpもtelnetも、sshやsftpに代替され一般に使われるようなサービスではありませんが、あえて実験のためにインストールしています。

◆**図1　SHODANのWebサイト(http://www.shodanhq.com/)**

なお、補足すると、ftpdとtelnetdは、inetdから起動され、その過程でtcpdによるアクセス制御が可能となっていますが、このディストリビューションのデフォルトではアクセス制御の設定がなされていません。ですので、もし自分でインストールする場合には、十分にアクセス制御に注意してください。

httpとftpのレスポンスを見るために、コマンドwgetを使い、サーバレスポンスを取得する“-S”を指定します。telnetは、コマンドtelnetを使い、ログインおよびパスワードのプロンプトのレスポンスを確認しています。

http

まずは、一番わかりやすいであろうWebサーバのレスポンスを観察してみます。

図2のサーバレスポンスから、Webサーバのプログラムの種類とバージョン、オペレーティングシステム、およびディストリビューション種類などの情報がわかります。もしPHPが使われていれば、次のようにPHPのバージョンもわかります。

```
X-Powered-By: PHP/5.3.3
```

筆者が使っているレーザープリンタは、かなりの年代ものですが、ステータスをhttpで問い合わせることができる機能を持っています。実際に問い合わせてみました(**図3**)。

図3にあるJC-HTTPDというサーバについて、Googleで検索をかけると複数のメーカーのプリンタ名が現れます。どうやらプリンタの組込み向けhttpサーバで、OEMにより複数のメーカーで採用されているようです。

ftpd

今度はftpを試してみます。**図4**のように利用しているftpのバージョンが取得できます。

◆**図2　Webサーバのレスポンスを確認してみる**

```
$ wget -S 192.168.100.50
--2015-01-20 01:35:28--  http://192.168.100.50/
Connecting to 192.168.100.50:80... connected.
HTTP request sent, awaiting response...
  HTTP/1.1 200 OK
  Date: Mon, 19 Jan 2015 16:35:26 GMT
  Server: Apache/2.2.22 (Debian)
  Last-Modified: Sat, 11 Oct 2014 04:12:14 GMT
  ETag: "2058d-b1-5051ddeda36e3"
  Accept-Ranges: bytes
  Content-Length: 177
  Vary: Accept-Encoding
  Keep-Alive: timeout=5, max=100
  Connection: Keep-Alive
  Content-Type: text/html
Length: 177 [text/html]
```

◆**図3　レーザープリンタに問い合わせてみる**

```
$ wget -S 192.168.100.110
--2015-01-20 02:51:07--  http://192.168.100.110/
Connecting to 192.168.100.110:80... connected.
HTTP request sent, awaiting response...
  HTTP/1.1 200 OK
  MIME-Version: 1.0
  Server: JC-HTTPD/1.12.16
  Pragma: no-cache
  Cache-control: no-cache
  Connection: close
  Content-Type: text/html
  Content-Length: 491
  Accept-Ranges: none
Length: 491 [text/html]
```

◆**図4　ftpのレスポンスを確認してみる**

```
$ wget -S ftp://192.168.100.50
--2015-01-20 03:00:40--  ftp://192.168.100.50/
           => '.listing'
Connecting to 192.168.100.50:21... connected.
Logging in as anonymous ...
220 d7.h2np FTP server (Version 6.4/OpenBSD/Linux-ftpd-0.17) ready.
--> USER anonymous

331 Guest login ok, send your complete e-mail address as password.
--> PASS Turtle Power!

530 Login incorrect.

Login incorrect.
```

telnet

telnetはコマンドtelnetをバッチ的に使い、ログインとパスワードのプロンプトが現れるかどうかを確認できます（**図5**）。

◆図5　コマンドtelnetをバッチ的に使ってレスポンスを確認してみる

```
$ ( sleep 1; echo ; sleep 1 ; echo sleep 5) | telnet 192.168.100.50
Trying 192.168.100.50...
Connected to 192.168.100.50.
Escape character is '^]'.
Debian GNU/Linux 7
d7 login:
Password: Connection closed by foreign host.
```

巨大なデータベースを保持するSHODAN

ここまで、既存のUNIXコマンドを使い、httpd、ftpd、telnetdといった典型的なサーバからどのようなレスポンスが取得できるのかを、実験し、観察してみました。たいへん簡単に取得できることがわかったと思います。

基本的にはこのような考え方で、レスポンスを取得する専用のプログラムを作成し、インターネット規模でスキャンをし、それをデータベース化したのがSHODANです。

もちろん、これらは単純にレスポンスを得るだけですので、機器やサービスが外部に公開されている限り、日本も含めてインターネットを自由に使える国々では、違法性はありません。

SHODANは、基本的に有料サイトです。ログインアカウントを取得し、1クレジットあたり5ドルを支払います。各種のサービスによって必要なクレジット数は変わります。50サーチまでは無料で使えますが、ちょっと試してみると、あっというまに使い切ってしまう感じです（**図6**）。

SHODANは、極めて強力なサーチエンジンであり、また検索時に絞り込みしやすいフィルタリングを用意しています。フィルタリングには、国、地域、ポート番号（プロトコル名）、OS種別、ネットワークレンジ（CIDR）、ホスト名などを指定できます。絞り込みに必要なものは、ほぼそろっていると言っていいと思います。

もし、ネットワーク接続している情報機器が不用意に、あるいはミスで外部からアクセスできる状態になっていれば、SHODANに登録されていることでしょう。

もちろんSHODANはセキュリティ監査として使えば、非常に有効なことは言うまでもありません。IPA（独立行政法人情報処理推進機構）は、SHODANを利用したインターネット接続機器のセキュリティ検査の方法を紹介する冊子を提供しています[注3]。

外部からアクセス可能なNASを公開していたならば、ホスト名やドメイン名などが付いているかどうかに関係なく、このSHODANにIPアドレスとバナーが記録されていることでしょう。

NASなどが存在していなくても、たとえば、JC-HTTPDのようなプリンタに組み込まれているhttpdサーバが発見できたならば、そのドメインは、ネットワークの正しいアクセス制御ができていない可能性が高いというヒントを与えてしまいます。

多数存在する無防備な機器たち

次のような興味深い2つの資料を見つけました。

①「九州大学の学内LANにおけるウェブサーバの分布と傾向について」[注4]

②「Shodan Computer Search Engine: 2013 Edition」[注5]

①は、2003年に九州大学の学内LANをスキャンして、どのようなWebサーバがあるのかを調査したものです。学内で反応した825台のうち、Apache

注3）IPAテクニカルウォッチ「増加するインターネット接続機器の不適切な情報公開とその対策」の公開
http://www.ipa.go.jp/about/technicalwatch/20140227.html
注4）「九州大学の学内LANにおけるウェブサーバの分布と傾向について」
http://catalog.lib.kyushu-u.ac.jp/handle/2324/1470659/p027.pdf
注5）"Shodan Computer Search Engine: 2013 Edition"
http://www.slideshare.net/Shakacon/dan-tentler

やMicrosoft IISといったサーバが6～7割程度あり、httpプロトコルを使っている組込み機器が3割強あるということがわかっています。筆者と同じタイプのCanon系レーザープリンタ組込みhttpdであるJC-HTTPDが40台、NECやXerox系で使われているSpyglass Microserverが31台などです。

同じ学内キャンパスネットワークといえども、本来は別セグメントのネットワークですので、これらのプリンタに別セグメントからアクセスできるのはネットワークの設定として正しい設定とは言えません。また、①では、ネットワークを経由して管理者権限でコントロールできるネットワーク機器も見つけています。資料の本文中では、組込み機器もPCやサーバと同様に、セキュリティに関して注意を払うべきであると提言しています。

組込み系の機器がネットワーク的にルーズなのは、2003年あたりには認識されていて、それが今でも同じく繰り返されていると言えるでしょう。

②は、SHODANを使った大規模な調査結果で、社会インフラに使われている制御システムがインターネット側からアクセスできる、しかも、そのようなものが大量に存在しているという可能性を示しており、極めて憂慮する内容です。社会インフラのシステムは、私たちが思うよりも、ずっと脆弱であることを示唆しています。

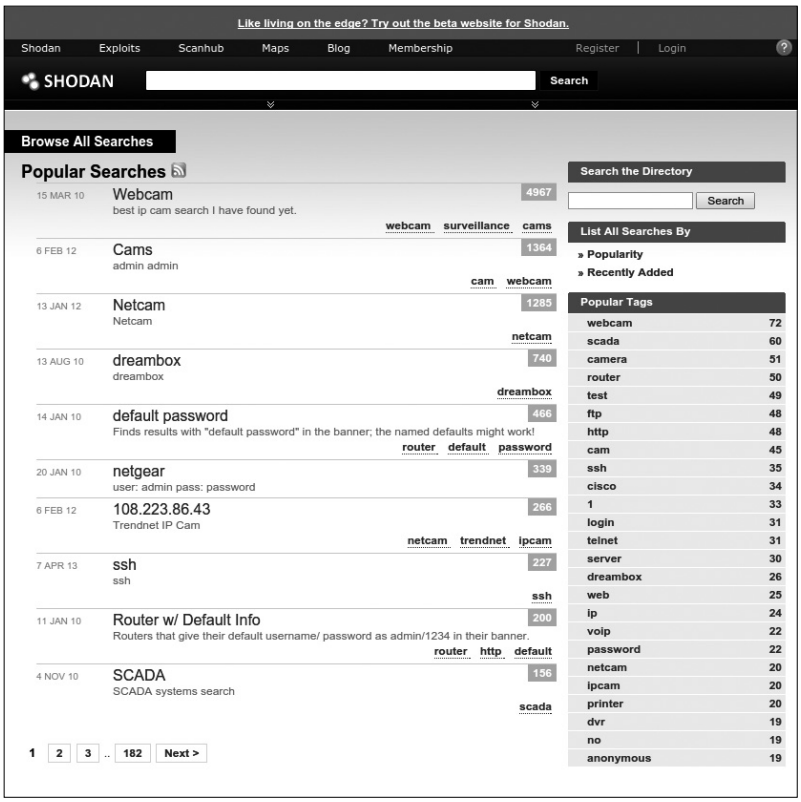
外部からのアクセス許可は慎重に

「インターネット側からアクセスできるなら便利」と多くの方は思っているでしょうし、またそのような機材がどんどん増えています。しかしながら、十分な安全設計をしないまま、便利というだけで入れた機能が、第三者から悪用されてしまった場合、問題はたいへん複雑になります。また、情報漏洩の場合、デジタル情報ですから、それがどのように広まり、どこに存在しているのかは、確かめようもありません。

今回の首都大学東京の公開NASの問題は、今日的な問題だと言えるでしょう。しかし振り返ると、組込み的に使われる機材の本質的な問題は、あまり目立たなかっただけで、以前より存在していたことがよくわります。

また、現在では、不用意に外部のネットワークに接続しているならば、SHODANのようなサービスが確実にキャッチし、白日のもとに晒してしまうことを覚悟しておかなければなりません。SD

◆図6　SHODANのサーチ結果（http://www.shodanhq.com/browse）



単純にブラウズするだけならば、無料で利用できる。この状況を見ても、いかにネットワーク上に無防備に晒されている機器があるかがうかがえる。

Red Hat Enterprise Linuxを極める・使いこなすヒント

.SPECs ドット・スペックス

第11回 1年ぶりの新バージョン・Fedora 21登場!

Fedora21が久々にリリースされました。Fedoraと言えば早いリリースが身上だったのですが、今回はひと味違うものになっています。Fedora.nextの錦の旗のもと、軸足を定め続々と発表されるさまざまな新技術に対応していく、というのが特徴です。

Writer 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)

Fedora.next下での最初のリリース

2013年8月に開催されたFlock[注1]で提出されたさまざまなアイデアをベースに、次の10年間に向けてFedoraを刷新していこうというプロジェクトがFedora.nextです。刷新の理由についてはFedora Magazine[注2]に詳細が綴られているのですが、かいつまむとFedoraを含むLinuxディストリビューションの開発は、より上位レイヤのOSSの開発と比較して「退屈」[注3]で開発者が集まりにくくなっていることや、クラウドを意識したアプリケーションの隆盛によって従来のFedoraディストリビューションが時代にそぐわなくなってきたこと、などが理由です。

Fedora.nextでは**図1**のように5つのワーキンググループを設けてディストリビューションであるFedoraをモジュール化し、**図1**の項目c、d、eを成果物として配布する形態を採ることになりました。

これらのいわば「ディストリビューションのリファクタリング」が行われたため、Fedora 20のリリースから1年を要して21がリリースされた、ということになります[注4]。

▼図1 Fedora.nextのワーキンググループ

a. Base Design
b. Environments and Stacks
c. Fedora Server
d. Fedora Workstation
e. Fedora Cloud

Fedora 21の配布形態

Fedora 20では次のような配布形態でしたが、

- CPUアーキテクチャごとの汎用ディストリビューション
- AWSで利用可能なAMI(Amazon Machine Image)
- ウインドウマネージャやゲームに特化した「スピン」

注1) Fedoraのユーザ・開発者が集まるFUDConに代わり、2013年からはFlockという名称でカンファレンスが開催されるようになった。

注2) http://fedoramagazine.org/fedora-present-and-future-a-fedora-next-2014-update-part-i-why/

注3) Fedora Magazineには次の一節が追加されており、Fedora Magazineで先に触れられているSlackware, openSUSE, Ubuntuだけでなく Gentooですら「退屈」とのことなので深刻な事態。"Update I forgot to mention Gentoo in the talk. Don't feel left out, Gentoo friends. You are also no longer cool."

注4) リリースノートを見るとFedora 21の新機能はそれほど多くなく、Fedora.nextに合わせたエンジニアリングが相当の工数を割いて行われたことがわかる。

Fedora 21では、次のように細分化されました。

・i386/x86_64/armv7hl 用 の“Fedora Workstation”
[ウインドウマネージャやゲームに特化した「スピン」]
・i386/x86_64/armv7hl用の“Fedora Server”
・“Fedora Cloud”
[i386/x86_64/armv7hl用ゲストイメージ(Base)]
[x86_64用Fedora Atomic Hostイメージ(Atomic)]
[AWSで利用可能なAMI]

これら各配布物の容量を小さくするとともに、用途に応じたISOファイルやイメージをダウンロードすることが可能になりました。たとえば、Fedora 20のx86_64用ISOファイルは約4.6GBの容量がありましたが、Fedora 21のx86_64用のServerは約2.1GB、Workstationは Live版ということもあり約1.5GB、さらにCloudの汎用イメージは約100MBに収まっています。

Fedora 21の「ダイエット」

配布物としての容量を小さくするために、Base Designワーキンググループでは、RPMパッケージ間の依存性の見直し[注5]や、配布形態に合わせてパッケージを選択しやすくするための分割などを行いました。

一例を挙げると、kernelパッケージをメタパッケージとし、カーネル本体はkernel-coreパッケージに、カーネルモジュール(ドライバ)はkernel-modulesパッケージに分割しました(**図2**)。これにより、通常は約40MBの容量があるkernel関連のパッケージは、Fedora Cloudでは約19MBのkernel-coreパッケージのみで利用可能になりました。クラウドコンピューティング=仮想化ではないのですが、おおむね世の中のクラウドコンピューティングの実装は仮想化技術を利用しているため、従来インストールされてきたデバイスドライバの多くが仮想化環境では不要である[注6]ことを利用した一種のハックといえます。

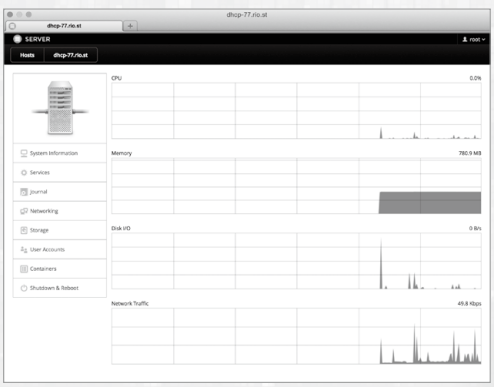
▼図2 Fedora 21のkernel関連パッケージ

```
# ls -1sh kernel-*
 42K kernel-3.17.4-301.fc21.x86_64.rpm
 19M kernel-core-3.17.4-301.fc21.x86_64.rpm
9.1M kernel-devel-3.17.4-301.fc21.x86_64.rpm
945K kernel-headers-3.17.4-301.fc21.x86_64.rpm
 18M kernel-modules-3.17.4-301.fc21.x86_64.rpm
2.2M kernel-modules-extra-3.17.4-301.fc21.x86_64.rpm
 35K kernel-rpm-macros-26-1.fc21.noarch.rpm
```

Fedora 21 Serverの目玉・Cockpit

Serverの新機能として最も注目されるのはCockpitでしょう(**図3**)。Cockpitはブラウザでサーバ管理を可能にする管理インターフェースで、サーバ管理の初心者だけでなく、上級者にとっても新たな管理手法の選択肢を提供するものといえます。

▼図3 Cockpitの画面



注5) RPMパッケージの定義ファイルであるSPECファイルでRPMパッケージの依存性を記述するが、往々にして「余計な」依存性が記述されているために不要なRPMパッケージがインストールされ容量が増大する現象が起きる。
注6) 残念なことにRHEL 7.0でも「サーバ」にWi-Fiチップ用のファームウェアなどがインストールされる。

ServerではCockpitはデフォルトでインストールされるので、firewalldの設定を変更しサービスを起動すればアクセス可能[注7]です。

```
# firewall-cmd --add-service cockpit
# systemctl start cockpit
```

Cockpitではサービスやジャーナル(ログ)、基本的なネットワークの設定、LVMを含むストレージの設定に加えDockerによるコンテナの管理も視野に入れており、今後注目される機能の1つといえます。

サーバロールの導入

仮想アプライアンスの普及に伴い、WebサーバやDBサーバとして迅速にデプロイできる機能がFedora/RHELにも求められており、Fedora 21にその実装がrolekitとして追加されました。Fedora 21ではドメインコントローラとDBサーバの2つのロールが含まれる予定でしたが、GA版では前者だけが提供されています。設定は非常に簡単で、管理者のパスワードを含むJSON形式のファイルを用意して、rolectlコマンドを発行するだけでドメインコントローラが構築できます。

```
# cat /etc/dc.settings.json
{
    "admin_password": "password"
}
# rolectl deploy --settings-file=/etc/dc.
settings.json [--name=instancename]
domaincontroller
```

今後、ロールが追加されていくに従い、サーバ構築が簡単に実施できるようになることが期待されます。

さまざまな分散処理ツールの追加

Fedora Serverでは多くのApache Foundationのプロジェクトの成果物、とくにHadoopに関連するものが追加されています。NoSQLデータベースのApache Accumulo、Apache HBase、DWHであるApache Hive、Hadoopの管理ツールであるApache Ambari、Hadoopのジョブ管理スケジューラのApache Oozieなどです。

ほかにも、Dockerとも組み合わせて利用されるクラスタ管理ツールのApache Mesos、大規模なデータをHadoopよりも簡単に分散処理することを目指しているApache Pig、MapReduceのより高速な代替であるApache Sparkなどが追加されており、Fedoraを利用して分散処理を行う敷居が下がったと言えるでしょう。

Fedora Atomic Hostの登場

前回の本連載で紹介したRHEL Atomic Hostのupstreamが、Fedora Atomic Hostです。RHEL Atomic Hostではdockerのバージョンは1.2.0、Kubernetesは0.4ですが、Fedora Atomic Hostではそれぞれ1.4.0、0.7(2015年1月中旬)が利用できるようになっており、春ころには正式版としてリリースされるRHEL Atomic Hostでもより新しいバージョンが提供されることになりそうです。

まとめ

WorkstationではGNOME 3.14が採用されるなど、1年ぶりのリリースということもあり多くの更新が含まれるFedora 21はまさに新しい世代のFedoraとなっており、久しぶりに「いじり回す楽しさ」が復活しています。ぜひダウンロードして触ってみてください。

次回はリリースが目前に迫るRHEL 7.1やAtomic Hostについて紹介する予定です。SD

注7) http://localhost:9090/でアクセス可能。

LibreOffice 4.4の新機能

Ubuntu Japanese Team
あわしろいくや　ikuya@fruitsbasket.info

今回は、大小さまざまな変更点があるLibreOffice 4.4の新機能をお伝えします。今までと使い勝手が変わったところも多数あります。

LibreOffice 4.4の概要

LibreOfficeは、OpenOffice.orgのフォークとして誕生し、リファクタリングやCalcのコアの変更、不要なコードの削除、新旧バグの修正、ドイツ語によるコメントの英語への翻訳など、内部的な変更が継続的に行われています。同時に、魅力的な新機能の追加も行われています。LibreOffice 4.4は2014年に行われたGoogle Summer of Code（GSoC）の成果を多数盛り込んでおり、新機能の追加や機能向上が図られています。ここまではこれまでも見られてきた傾向ですが、4.4からはOpenOffice.orgの使い勝手の見直しも積極的に行われ、「こうなったらもっと使い勝手がよくなるのに」という変更が盛り込まれています。換言すれば、歴史的経緯以外に変更しない積極的な理由が存在しない個所にも手が入っています。

今回はそういった部分を重点的に解説することにしました。とくに具体的な変更点を押さえていると、これまでOpenOffice.org/LibreOfficeをお使いだった方の利便性が格段に向上すると考えたからです。

変更点の傾向としては、ディスプレイサイズの変化によるところが大きいものと考えられます。以前はXGA程度の解像度を念頭に置いているように思えたデザインですが、現在は解像度も上がり、また横長のワイドディスプレイが主流になりました。すなわち縦方向にはスペースの節約を図り、横方向に機能を充実させるという方向にシフトしているように思えるのです。このあたりからも、今後も利便性を向上し、継続して開発していくという強い意志を感じます。

全般

ツールバーアイコンの見直し

ツールバーのアイコンが見直されました。増えたものも減ったものもありますが、全体的な数としては増加しています。増えたものはいくつか後述します。これは前述のとおり、ディスプレイサイズの変更によるものと考えられます。

ウィジェットレイアウトへの移行完了

LibreOffice 4.0から継続して作業していた、ウィジェットレイアウト（Glade[注1]を使用したXMLベースのユーザインターフェース）への移行がついに完了しました。GSoC 2014の成果でもあります。ウィジェットレイアウトと旧来のダイアログの見分け方は簡単で、マウスでサイズの変更ができるかどうかでわかりました。前のバージョンを使用しているのでしたら、一度試してみるとおもしろいかもしれません。

注1）https://glade.gnome.org/

PDFエクスポートのデジタル署名

使っている人はいなさそうな気はするのですが、LibreOfficeにはデジタル署名機能があります。LibreOffice自体には証明書の管理機能はなく、Thunderbirdの機能を使用しているため、別途Thunderbirdのインストールと証明書のインポートが必要です。これまでは[ファイル]-[デジタル署名]しかなかったのですが、PDFエクスポートのオプションに[デジタル署名]というタブが新設され、ここで選択した証明書で署名できるようになりました(**図1**)。

インポートフィルタ

Adobe PageMakerのファイルがインポートできるようになりました。LibreOffice、あるいはThe Document Foundationは開発が終了したアプリケーションのインポートフィルタに力を入れており、ほかにもMac Draw/Mac Draw Ⅱといった、古いMac用アプリケーションのインポートも可能になっています。手元にファイルがある方は読み込ませてみてはいかがでしょうか[注2]。ほかにも既存のインポートフィルタの強化も行われています。

図1　PDFエクスポートに[デジタル署名]が新設

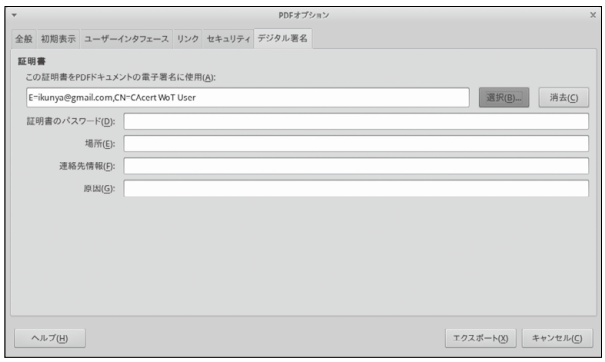


図2　Impress用の新しいテンプレート

Microsoft OneDrive/SharePoint 2013への接続機能

以前のバージョンからSharePoint 2010には接続できましたが、新たに2013もサポートしました。OneDriveの接続機能は4.4の目玉とも呼べる機能……と言いたいところですが、残念ながらオフィシャルの4.4.0 RC3の段階でも有効になっていないバイナリが配布されています。いずれにせよ、UbuntuのパッケージではGoogle Driveと同様接続できるようにはならないものと思われます[注3]。

新しいテンプレートの追加

新しいテンプレートがいくつか追加されました(**図2**)。Impress用テンプレートのうち2つは、日本からの応募(のがたじゅん氏)によるものです。ちなみにコンテストによって選ばれたのですが、コンテストを開催→応募作品が集まる→優秀作はデフォルトでインストールされる→作者には景品プレゼント[注4]、という流れはみんなが得する素晴らしいモデルといえます。ただ、これでも充分とはいえないので、もっとたくさんの選択肢が欲しい場合はUbuntu Weekly Recipe 第279回[注5]を参考にしてみてください。

色の選択

色の選択が新しくなりました(**図3**)。最近使用した色を記憶するようになったほか、カラーパレットの変更もできるようになりました。さらに[色の指定]から任意の色を指定できるようになりました。これまではあらかじめ使用する色を登録する必要があり、使い勝手がいいとはいえませんでした。

注2) 本誌の読者さんなら、普通にいらっしゃる気がしますがいかがでしょう?

注3) ちなみにOneDrive接続機能もGoogle Drive接続機能もlibcmisというライブラリで実現しているのですが、これがバージョンアップされたことによりLinux版のオフィシャルバイナリでもGoogle Driveに接続できるようになるのかなと思ったものの、実際にはなりませんでした。Macでは試していませんが、Windowsでは問題なく接続できてます。

注4) 今回はTシャツでした。

注5) http://gihyo.jp/admin/serial/01/ubuntu-recipe/0279

行間のドロップダウンメニューの追加

おもにWriterで使用しますが、これまではサイドバーにしか行間の設定がありませんでしたが、書式設定ツールバーにアイコンが追加されました。同時にメニューの文言も変更されています。

箇条書きと番号付けのドロップダウンメニューの追加

行間のドロップダウンメニューと同じく、書式設定ツールバーにアイコンが追加されました。また、いちいちメニューを開かなくてもドロップダウンから設定できるようになりました。

サイドバーの仕様変更

これまではサイドバーにある[スタイルと書式設定]と[ギャラリー]は、F11キーを押した際に表示される[スタイルと書式設定]と、[ツール]-[ギャラリー]で表示されるギャラリーとはまったく別に動作していました。しかし、4.4からはF11キーを押すとサイドバーの[スタイルと書式設定]を表示するようになり、[ツール]-[ギャラリー]は削除されました。すなわち、同じ役割を持つものが2つあったものが、4.4からはサイドバーに統合されたということになります。Writerで長文を書こうとすると必ず[スタイルと書式設定]のお世話になるので、これまでのLibreOfficeに慣れている方はちょっと驚くかもしれません。

右クリックメニューの変更

右クリックで表示されるメニューが見直されました。よく使う切り取りや貼付けを上に移動し、いくつか追加された項目もあります。むしろ今までどうしてこうではなかったのかという変更です。

Firefoxテーマの強化

以前よりFirefoxのテーマを使用できていましたが、テーマの場所(URL)を直接入力する必要があるなど、使い勝手がいいとはいえませんでした。しかし、4.4からはテーマの検索ができるようになりました。いくつかプリセットの検索キーワードもあります。検索結果が9個までしか表示されないという制限はありますので、好みの検索結果にならない場合は検索キーワードを追加してみてください(**図4**)。

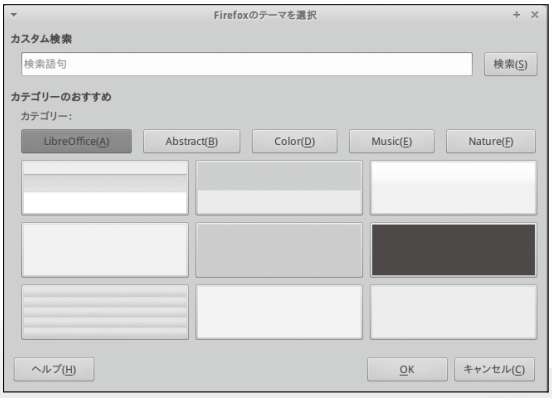
Writer

図形描画ツールバーにあるシェイプで作成した図形にテキストボックスが埋め込める

シェイプを右クリックすると[テキストボックスを追加]というメニューが追加され、これをクリックするとテキストボックスが埋め込めます。このテキストボックスはWriterの段落でできることであれば何でもできます。すなわち、表などを埋め込むことができるようになります。どの程度の実用性があるのかはさておき、表現力が格段に向上しました。

図3　モノクロなのでわかりにくいですが、ようやくまともで今時の色の選択機能になりました

図4　新しくなったFirefoxのテーマ選択



スタイルのドロップダウンにサブメニューを追加

スタイルのドロップダウンに［選択範囲と一致するように更新］と［スタイルの編集］が追加されました。後者はスタイルの編集ウィンドウが開くのでわかりやすいですが、前者は直感的にはわかりくいです。原則としてスタイルは一括で設定するものですが、今編集しているスタイルの書式を全体に反映したいということもあります。その場合、たとえば色を変えるなど直接変更し、その部分を選択して［選択範囲と一致するように更新］をクリックすると、全体のスタイルとして適用されます。Microsoft Wordをお使いの方は、［選択個所と一致するように更新する］と同じ機能と解説するとわかりやすいかと思います。

画像の変更

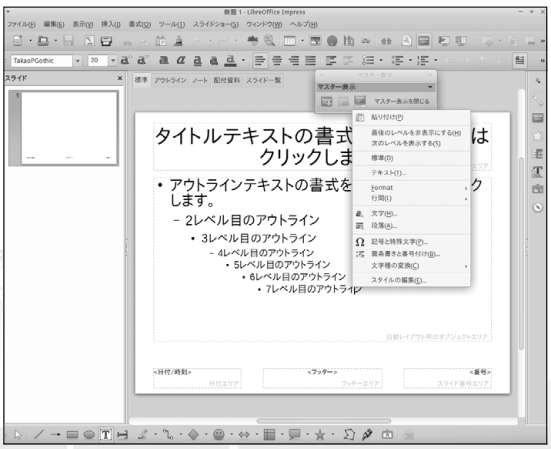
画像を右クリックすると［画像の変更］という項目が表示されるようになりました。誤った画像を貼り付けてしまった場合に使用すると便利です。

Calc

統計の追加

［データ］-［統計］に［z検定］と［カイ二乗検定］が追加されました。

図5　ダミーテキストが編集不可になり、右クリックに2つ項目が追加されました。



数式を数値に

［ツール］-［セルの内容］に［数式を数値に］が追加されました。これまで数式を数値にする場合は一度数式をコピーし、貼付け時に［形式を限定して貼り付け］-［数値］などを行う必要があったので、手間が省けることになります[注6]。

追加された関数

AGGREGATE関数が追加されました。使い方はExcelの同関数と同様で、各種オプションに応じて数値の集計ができます。説明が簡潔なうえにヘルプもないので扱いにくいですが、ソースコードを見る限りでは値の意味はExcelと同じでした[注7]。

タブの変更

これはCalcだけではなくDrawでも同じですが[注8]、タブのデザインが変更されました。これまではスクロールバーとタブが同じ段にあり、タブが増えたり名称が長くなったりするとタブ欄を広げスクロールバーを狭くする必要があり、煩わしく思ったことは1度や2度ではないと思います。しかし、4.4からはタブとスクロールバーが別の段になったことにより、このような煩わしさはなくなりました。同時にタブの形状も変更されており、これまでのような台形[注9]ではなく長方形になっています。

Impress/Draw

マスター表示のプレビューの仕様変更

4.4ではImpressに大きめの変更が入りました（**図5**）。マスター表示は［表示］-［マスター］-［スライドマス

注6） ただし、個人的な経験では数式は数式で、数値は数値であったほうが望ましいため、いちいちコピーする機会のほうが多いように思います。

注7） Excelとの相互運用性向上のために実装されたので当然ではあるのですけど。

注8） ただ、Drawでタブを駆使する、というケースは少数ではないかと思うのでここに分類しました。

注9） 下底が短い台形です。

ター]あるいはサイドバーの[マスターページ]を表示し、[このプレゼンテーションで使用]を右クリックの[マスターを編集]で表示できます。テンプレートを外見のカスタマイズだとすると、マスター表示は内面のカスタマイズで、アウトラインレベルをいくつまで表示するかなどの設定ができます。

タイトルやオブジェクトには「タイトルテキストの書式を編集するにはクリックします」などのダミーテキストがあらかじめ挿入されていますが、4.3まではなぜかこれが変更できました[注10]。しかし、4.4からは変更ができなくなりました。その分アウトラインレベルを増減することもできなくなったので、右クリックに[次のレベルを表示する]と[最後のレベルを非表示にする]が追加されました。

マスター要素の削除が簡単に

マスター表示にはタイトルとオブジェクト(内容)以外にも日付やページ番号用の枠があります。これをマスター要素といいます。これまでマスター要素を削除するためには[表示]-[マスター]-[マスター要素]でダイアログを表示して不要な要素のチェックを外す必要がありましたが、4.4からは不要なマスター要素を選択して Delete キーで削除できるようになりました。より直感的な動作にするための仕様変更といえます。

読み取り専用での保存

ほとんど使っている人はいないと思うのですが、Writer/Calcには保存時に[パスワード付きで保存する]のチェックを入れるとパスワードを設定するダイアログが表示されます。ここに[オプション]があり、[ファイルを読み取り専用で開く]にチェックを入れてパスワードを設定すると、ドキュメントを読み取り専用で開くようにできます。入力したパスワードは編集の際に必要となります。この機能がDraw/Impressでも使用できるようになりました。

ズームとパン機能

[表示]-[ツールバー]-[ズーム]に[ズームとパン]という機能が増えました。この機能を有効にすると、クリックでズームイン(拡大)、Ctrl +クリックでズームアウト(縮小)、Shift +クリックでパン(右隣にある[移動]と同じ機能)ができるようになります。細かい作業をする場合は画面の拡大縮小をよく使いますし、拡大すると目的のところをパンするのも手間になりますが、これで一気に解決するようになります。

翻訳状況

新規の翻訳語数はさほど増えていませんが、ウィジェットレイアウトへの移行を行うと既存の翻訳が使用できなくなり、その分を新規に訳すことになるので[注11]、作業量としてはけっこうなボリュームがありました。

しかし、2015年1月下旬現在で99%の翻訳が完了しています。残りは翻訳に専門的な知識が必要なもの、どこで使われているのかわからないので翻訳のしようがないものばかりが残っています。こればかりはたくさんの目が必要ですので、翻訳にご協力いただけると幸いです。

Ubuntuで4.4を使用するには

いつものようにUbuntu 15.04ではLibreOffice 4.4が使用できるようになる見込みです。またPPA[注12]にもアップロードされるものと思われるので、こちらを利用するのもいいでしょう。これまで見てきたとおり4.4は積極的にバージョンアップするに値する変更がたくさん見られます。SD

注10) 例えばロシア語のLibOで作成されたテンプレートは、ダミーテキストがロシア語になっている、ということもありましたが、これでそういうことがなくなったということです。

注11) もちろん基本的には以前のものを踏襲するのですが、中にはわかりやすく変更してしまうものもあります。

注12) https://launchpad.net/~libreoffice

Be familiar with FreeBSD.

チャーリー・ルートからの手紙

第17回 ❖安定動作につながるディレクトリの知識（その1）

◉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。

ディレクトリあたりの最大ファイル数、ご存じですか?

安定して高速に動作するシステムを組み上げるときに重要になるのが、ファイルシステムとディスクキャッシュです。ファイルシステムが最も性能を発揮できるようにファイルやディレクトリを配置し、可能な限りディスクキャッシュに載ってディスクへのアクセスが発生しないようにすることが大切です。

今回はそうしたスキルの1つとして、1ディレクトリあたりの最大ファイル数について紹介します。運用していたシステムがいきなり遅くなった、といったケースでは、1つのディレクトリに納めるファイルの数が、あるしきい値を超えたことが原因で処理が遅くなるといったことがあります。ファイルシステムの構造を知り、こうした上限を把握することで、突然システムの動作が遅くなるといったことを避けることができます。

変数ファイル（Value File）の活用

著者の会社では主キーをファイル名として値を書き込んだファイルを「変数ファイル（Value File）」と呼んで活用しています（**図1**）。1レコードないしは1フィールドなどを書き込んだファイルのことを言います。ファイルの中身は単一のデータであることもあれば、複数のフィールドであることもあります。

変数ファイルはいわばファイルシステムが提供するハッシュ機能を利用して、主キーを値に変換するしくみです。**図2**のように、特定のディレクトリにたくさんの変数ファイルを作成して利用することになります。**図2**の例では16万個ほどのファイルが1つのディレクトリに作成されています。

変数ファイルを活用するとデータへの高速アクセスが可能になります。リアルタイムのデータ更新およびデータ参照系にも活用できます。アクセスが新しいほどディスクキャッシュに載るため、2度目以降のアクセスはオンメモリで高速になりますし、ファイルシステムに展開されているのでデバッグや

▼**図1 変数ファイルの例**

```
% cat 0000001208
20150118201121 000008 000003 000021 有効 有 有 無 0 0
%
```

▼**図2 変数ファイルを大量に格納したディレクトリ**

```
% ls | head
0000001205
0000001206
0000001207
0000001208
0000001209
0000001210
0000001211
0000001212
0000001213
0000001214
% ls | wc -l
  159446
%
```

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。

データトレース、解析などが容易であるという特徴もあります。

変数ファイルを活用する場合の注意点は、利用するファイルシステムによっては1つのディレクトリに納めて性能を発揮できるファイルの個数に上限があることや、作成できるファイルの数に上限があることです。大規模システムによってはファイルの個数が億単位になることはざらにありますので、この上限を知らないでシステムを構築すると、あるときi-nodeが枯渇してシステムが停止するといった事態が発生したり、いきなりプロセッサリソースが100%消費されてシステムとして使い物にならなくなることがあります。

ディレクトリの構造を知ろう

変数ファイルの高速性は、ファイルシステムの提供するディレクトリのファイル名→データ取得の動きが高速、理想的にはO(1)[注1]であることを期待しています。これが成り立たないと、変数ファイルの利点は半減します。ではディレクトリはそのような構造になっているかといえば、典型的なUFS系のファイルシステムではそのようなしくみにはなっていません。1ディレクトリあたりのファイルが増えるほど、動作は遅くなる構造になっています。

ユーザの視点から見ると、ディレクトリはファイルをツリー構造に整理するためのしくみに見えます。しかし実際には、「ディレクトリ」というのは「配置を整理するために使用が限定されたファイル」のことを言います。ディレクトリも1つのファイルなのです。ディレクトリファイルの中身はファイル名やディレクトリ名とi-nodeとのキーバリューデータになっています。ディレクトリの中身を読むことで、そのディレクトリに置いてあるファイルやディレクトリの一覧を得たり、その実態を納めたデータへのメタデータであるi-nodeにアクセスできるというしくみになっています。こういったしくみを用意することで、あたかも木構造へファイルを整理しているように見せかけています。

ここでは実際にディレクトリからそのディレクトリに配置されたことになっているファイルや、ディレクトリの一覧を取得して出力するソースコードを見ることで、その動きを追ってみます。**リスト1**のソースコードをcat_dir.cとしたとしましょう。このプログラムを実行するとls(1)を引数なしで実行したのと同じような結果が得られます（**図3、4、5**）。

Cに詳しくない方はコメントだけ読んで動作を追ってみてください。このプログラムはカレントディレクトリのファイルやディレクトリを出力します。とくに「readdir(dirp)」という関数に注目してください。この関数でディレクトリの中の次のエン

▼**リスト1　ディレクトリの中身を出力するプログラム（cat_dir.c）**

```
#include <dirent.h>
#include <stdio.h>

int
main(int argc, char *argv[])
{
    DIR *dirp;
    struct dirent *dp;
    char *p;
    /* ディレクトリをオープン */
    dirp = opendir(".");
    /* ディレクトリエントリへのポインタを取得 */
    dp = readdir(dirp);
    while (NULL != dp) {
        /* ディレクトリ名へのポインタを取得 */
        p = dp->d_name;
        /* ディレクトリ名を出力 */
        while (*p != '\0') putchar(*p++);
        putchar('\n');
        /* 次のディレクトリエントリへのポインタを取得 */
        dp = readdir(dirp);
    }
    /* ディレクトリをクローズ */
    closedir(dirp);
    return (0);
}
```

注1　O記法（ランダウ記法）。O(1)は処理データが増えても性能が変わらないことを意味している。

トリを持ってきて、出力して、また次のエントリを持ってきて、という動作を繰り返しています。

ディレクトリは先頭からはじまるリンクドリストと呼ばれるデータ構造になっています。つまり、ディレクトリの一番最後のエントリを取得するには、ディレクトリの最初のエントリから全部たどっていかないといけません。これが「1ディレクトリあたりのファイルが増えるほど、動作は遅くなる構造」という動作の理由です。

ちなみにcat_dirは、ls(1)と比べて高速に動作します。lsとcat_dirの出力結果を見ればわかりますが、cat_dirがたどった結果をそのまま出力している一方で、lsでは結果をいったんソートしてから出力しています。

今回のケースですと**図6、7**に見るように、cat_dirのほうが16倍ほど高速に動作します。このように、高速なシステムを組む、とくにオペレーティングシステム（OS）の提供している機能を100％使い切るには、OSの提供している機能をシンプルに活用した単機能コマンドを開発して利用するというのも効果的な方法です。ほとんどの場合で、専用に開発したコマンドのほうが高速に動作します。

▼図3　cat_dir.cのコンパイル方法

```
% cc -o cat_dir cat_dir.c
```

▼図4　cat_dirの実行例その1

```
% cat_dir | head
.
..
0000084380
0000001264
0000108140
0000327734
0000030940
0000292699
0000407035
0000173676
%
```

▼図5　cat_dirの実行例その2

```
% cat_dir | wc -l
  159448
%
```

こうしたUFSのディレクトリが抱える限界を突破するには2つのアプローチが考えられます。1つはUFSのディレクトリ機能を高速化する機能を実装すること。もう1つはUFSではなく別のファイルシステムを採用することです。

ディレクトリの限界を突破するDIRHASH

前者のアプローチはさまざまなOSで実施されています。FreeBSDの場合はUFSにDIRHASHと呼ばれる機能を追加してディレクトリオペレーションの高速化を実現しています。

DIRHASHは大規模なディレクトリ、これまで紹介してきたような変数ファイルを大量に格納したディレクトリを高速に操作するための機能で、ファイル名とディレクトリエントリへのオフセットのマップデータをキャッシュするというものです。先ほど説明したように、ディレクトリの先頭からアクセスするため動作が遅くなるわけで、ディレクトリエントリの場所まで一発で移動できれば速度は遅くならないというしくみです。DIRHASHではブロックごとのフリースペースについても情報を管理することで処理の高速化を実現しています。

DIRHASHはデフォルトで有効になっています。この機能はsysctl(8)値経由で制御が可能です（**図8**）。DIRHASHの効果を引き上げたいのであれば、メモリが豊富にあるならvfs.ufs.dirhash_maxmemの値を引き上げることで性能の改善が期待できます。また、キャッシュはデフォルトでは60秒しか保持されませんので、vfs.ufs.dirhash_reclaimageの値を引き上げればキャッシュの効果をより長期にわたって適用することができます。

▼図6　ls(1)の実行時間

```
% /usr/bin/time ls > /dev/null
        0.82 real         0.75 user         0.06 sys
%
```

▼図7　cat_dirの実行時間

```
% /usr/bin/time cat_dir > /dev/null
        0.05 real         0.01 user         0.04 sys
%
```

もちろんこの効果が見込めるかどうかはどのようにファイルやディレクトリを配置し、そしてどのようにアクセスがあるかに依存していますので、値を変更する場合には設定変更の前後でパフォーマンスチェックをして比較することを忘れないようにしてください。

◆◆◆

今回はここで誌面がつきてしまいました。次号ではUFSのディレクトリが抱える限界を突破するための、もう1つの方法について紹介します。SD

▼図8　DIRHASH機能を制御するsysctl(8)値

```
% sysctl -a | grep dirhash
vfs.ufs.dirhash_minsize: 2560
vfs.ufs.dirhash_maxmem: 26927104
vfs.ufs.dirhash_mem: 0
vfs.ufs.dirhash_docheck: 0
vfs.ufs.dirhash_lowmemcount: 160
vfs.ufs.dirhash_reclaimage: 60
%
```

column

UFSの詳細情報を知る

UFSはブロックサイズとフラグメントサイズという値を調整できます。一般的に、ブロックサイズを引き上げればアーカイブや動画のようなサイズの大きなファイルが多い場合に性能を発揮でき、小さくすればディスク領域の利用効率を引き上げることができると言われています（もちろんケースバイケースです）。

フラグメントサイズは通常、ブロックサイズの8分の1に設定されています。FreeBSD 10.1-RELEASEであればブロックサイズ（bsize）は32KB、フラグメントサイズ（fsize）は4KBです。この値はなかなかよく考慮されていて、多くのケースで優れた性能を発揮します。

こうした値であったり、i-nodeやブロックの残りの数などはdumpfs(8)というコマンドを使って調べることができます（**図A**）。UFSを使ってシステムを構築する場合にはぜひ知っておきたいコマンドです。

▼図A　UFSの詳細情報を得るdumpfs(8)コマンド

```
% dumpfs / | head -18
magic   19540119 (UFS2) time    Sun Jan 18 15:36:11 2015
superblock location     65536   id      [ 54af88eb c3bab1bb ]
ncg     66      size    10485632        blocks  10153495
bsize   32768   shift   15      mask    0xffff8000
fsize   4096    shift   12      mask    0xfffff000
frag    8       shift   3       fsbtodb 3
minfree 8%      optim   time    symlinklen 120
maxbsize 32768  maxbpg  4096    maxcontig 4     contigsumsize 4
nbfree  1166552 ndir    41606   nifree  5029873 nffree  5635
bpg     20035   fpg     160280  ipg     80256   unrefs  0
nindir  4096    inopb   128     maxfilesize     2252349704110079
sbsize  4096    cgsize  32768   csaddr  5056    cssize  4096
sblkno  24      cblkno  32      iblkno  40      dblkno  5056
cgrotor 0       fmod    0       ronly   0       clean   0
metaspace 6408  avgfpdir 64     avgfilesize 16384
flags   soft-updates+journal
fsmnt   /
volname         swuid   0       providersize    10485632
%
```

24

Debian Developer　やまねひでき　henrich@debian.org

reportbugでバグを報告する方法

Debian Hot Topics

Debian 8 “Jessie” バグ潰しも佳境に

本誌の発売時には、そろそろDebian 8 “Jessie”がリリースされているかいないかの微妙な時期です。本稿執筆時点では、バグ修正作業も佳境に入っています。世界各地でBSP(Bug Squashing Party)と呼ばれるバグ潰し作業の集まりが開かれ、Debian BTSで「Release Critical」(リリースに際して致命的な問題)となっているバグを潰す作業が着々と進んでいます。

さて、前回はそのDebian BTSに、メールでバグ報告をする話をしましたので、今回はバグ報告の専用ツール「reportbug」を使ってみることにしましょう。

reportbugコマンドの設定

reportbugコマンドを初めて実行すると、設定ウィザードが出てきます。次の❶～❿と図1～10を参考に値を入力してください(あとから「reportbug --configure」を実行して再度ウィザードを実行したり、~/.reportbugrcファイルを直接修正したりもできますので気楽にどうぞ)。

❶reportbugを実行すると、動作モードの選択肢が表示されます(図1)。「novice」(初心者)、「standard」(標準)、「advanced」(上級)、「expert」(エキスパート)の4つです。デフォルトの「novice」のまま Enter を押しましょう

❷「直接インターネットへ接続できるか?」と聞かれる(図2)ので、そのまま Enter を押して進みます

❸「本名を入力してください」と出る(図3)ので入力します

❹メールアドレスを入力します(図4)。ここでは「アドレスが公開されるので、スパムメールがたくさん来るよ。スパムフィルタリングできるようなアドレスを入力してね」と注意が出ています。Gmailなどが適当でしょうが、筆者はメールソフト(Sylpheedと迷惑メールフィルタ機能)を使ってローカル環境でフィルタリングするようにしています

❺「インターネットにちゃんとメール送信できるメールサーバ(MTA)を動かしているか?」という質問です(図5)。ローカルにMTAがあっても、きちんとした設定がされていることはあまりないでしょうから、そのまま Enter を押して進みましょう

❻「メールを送るSMTPサーバを指定してください」とあります(図6)。指定しない場合はデフォルトのポート25を使いますが、ISPによってはブロックされていることもあるので注意しましょう[注1]。例ではポート587を使って送るようにしてみました

❼「SMTP送信時のユーザ名を入力してください」とある(図7)ので、ISPから指定されているSMTP接続時のユーザ名を入力します

注1) Outbound Port 25 Blockingと言います。

▼図1　❶reportbugコマンドを実行

```
$ reportbug

Welcome to reportbug! Since it looks like this is the first time you have used reportbug,
we are configuring its behavior. These settings will be saved to the file "/home/
username/.reportbugrc", which you will be free to edit further.
Please choose the default operating mode for reportbug.

1 novice    Offer simple prompts, bypassing technical questions.

2 standard  Offer more extensive prompts, including asking about things that a moderately
sophisticated user would be expected to know about Debian.

3 advanced  Like standard, but assumes you know a bit more about Debian, including
"incoming".

4 expert    Bypass most handholding measures and preliminary triage routines. This mode
should not be used by people unfamiliar with Debian's policies and operating procedures.

Select mode: [novice]↵
```

※ ↵は Enter キーを押下することを表している(以下同)

▼図2　❷インターネット接続の確認

```
Will reportbug often have direct Internet access? (You should answer yes to this question
unless you know what you are doing and plan to check whether duplicate reports have been
filed via some other channel.) [Y|n|q|?]?↵
```

▼図3　❸本名の入力

```
What real name should be used for sending bug reports?
[henrich]> Hideki Yamane↵
```

※ 文字が斜体の部分はユーザの入力例を表している(以下同)

▼図4　❹メールアドレスの入力

```
Which of your email addresses should be used when sending bug reports? (Note that this
address will be visible in the bug tracking system, so you may want to use a webmail
address or another address with good spam filtering capabilities.)
[henrich@linux-machine]> henrich@debian.org↵
```

▼図5　❺MTAに関する質問

```
Do you have a "mail transport agent" (MTA) like Exim, Postfix or SSMTP configured on this
computer to send mail to the Internet? [y|N|q|?]?↵
```

▼図6　❻SMTPサーバの指定

```
Please enter the name of your SMTP host. Usually it's called something like "mail.example.
org" or "smtp.example.org". If you need to use a different port than default, use the
<host>:<port> alternative format. Just press ENTER if you don't have one or don't know,
and so a Debian SMTP host will be used.
> mail.example.com:587↵
```

▼図7　❼SMTP送信時のユーザ名の入力

```
If you need to use a user name to send email via "mbox.example.com:587" on your computer,
please enter that user name. Just press ENTER if you don't need a user name.
> henrich@exmaple.com↵
```

▼図8　❽暗号化の有無

```
Do you want to encrypt the SMTP connection with TLS (only available if the SMTP host
supports it)? [y|N|q|?]? y↵
```

▼図9　❾HTTP Proxyの指定

```
Please enter the name of your proxy server. It should only use this parameter if you are
behind a firewall. The PROXY argument should be formatted as a valid HTTP URL, including (if
necessary) a port number; for example,  http://192.168.1.1:3128/. Just press ENTER if you
don't have one or don't know.
>↵
```

▼図10　❿パッケージ名の指定

```
Default preferences file written. To reconfigure, re-run reportbug with the "--configure" option.
Please enter the name of the package in which you have found a problem, or type 'other' to
report a more general problem. If you don't know what package the bug is in, please contact
debian-user@lists.debian.org for assistance.
>↵

No package specified or we were unable to find it in the apt cache; stopping.
```

❽「TLSを使って暗号化接続をするか？」という質問です(図8)。SMTPサーバがサポートしているなら「y」と答えましょう

❾HTTP Proxyの指定です(図9)。既存のバグレポートを取得するのはHTTP経由ですので、Proxyでブロックされている場合はそちらを指定します。なければ Enter を押しましょう

❿パッケージ名の指定です(図10)が、reportbugの設定が完了していないので、ここではまだ入力しません。 Enter を押して進み、設定を終了します

▼図11　~/.reportbugrcにパスワードを設定する

```
--- .reportbugrc.old    2015-01-14 12:33:42.127869106 +0900
+++ .reportbugrc        2015-01-14 12:34:01.659965960 +0900
@@ -14,7 +14,7 @@
 # Send all outgoing mail via the following host
 smtphost "mail.example.com:587"
 smtpuser "henrich@example.com"
-#smtppasswd "my password here"
+smtppasswd "mypassword"   ←行頭のコメントを外して、パスワードを記述する
# Enable TLS for the SMTP host
 smtptls
 # If nothing else works, remove the # at the beginning
```

ここまでで作成された~/.reportbugrcファイルをエディタで開いて、SMTPパスワード行のコメントを外してパスワードを入力します。設定前後のdiffは図11のようになります。

これでreportbugの設定が完了しました。

reportbugコマンドを使ったバグ報告

reportbugでバグを報告する流れは、次のようになります。

(1) パッケージ名を指定して起動
(2) 既存のバグレポートを確認
　(a) 追加のレポートを行う
　(b) 新規にレポートを行う
(3) バグレポートを埋め、パッチなどを添付
(4) メールで送信

ここでは、dh-makeパッケージについてバグ報告をする例を示します。

reportbug dh-makeと入力し、reportbugを起動します。すると、dh-makeで報告されているバグレポートが重要度と番号でソートされて表示されます(図12)。ここでは数字を入れて既存のレポートを確認するか、 N を押して新規にバグ報告をします。既存レポートの表示は q で抜けられます。

今回は、新規にバグを報告してみます。バグの要約／重要度などを入力する(**図13**)と、エディタが立ち上がります。先ほど入れたバグレポートの要約がSubjectに、そのほかに指定したSeverityなどが入っていることがわかります。あとは本文のテンプレートを埋めていってください(**図14**)。

エディタを終了すると、reportbugが送信するかどうかを尋ねてきます(**図15**)。[y]を押せ

▼図14　新規にバグ報告を行う

```
Subject: dh-make: Please make it i18n available
Package: dh-make
Version: 1.20140617
Severity: normal
(...以下略...)
```

▼図12　パッケージ名を指定してreportbugを起動

```
$ reportbug dh-make   ←dh-makeパッケージのバグ報告を行う
*** Welcome to reportbug.  Use ? for help at prompts. ***
Note: bug reports are publicly archived (including the email address of the submitter).
(...略...)
Querying Debian BTS for reports on dh-make (source)...13 bug reports found:

Bugs with severity important
   1) #773275  dh-make: fails to create native package with --defaultless
  (...略...)
  13) #774545  dh-make: Improve Emacs lisp package support to comply newer
Debian Emacs Policy
(1-13/13) Is the bug you found listed above [y|N|b|m|r|q|s|f|e|?]?↵
```

dh-makeの既存のバグレポートが表示される

←既存レポートを確認するか、新規にバグ報告を行うかを選択する

▼図13　バグの要約／重要度などを入力

```
Briefly describe the problem (max. 100 characters allowed). This will be the bug email
subject, so keep the summary as concise as possible, for example: "fails to send email" or
"does not start with -q option specified" (enter Ctrl+c to exit reportbug without reporting
a bug).
> Please make it i18n available↵   ←ここで問題の要約を記述する
Rewriting subject to 'dh-make: Please make it i18n available'
How would you rate the severity of this problem or report?

1 critical     makes unrelated software on the system (or the whole system) break, or causes
serious data loss, or introduces a security hole on systems where you install the package.
(...略...)
8 wishlist     suggestions and requests for new features.

Please select a severity level: [normal]↵
Do any of the following apply to this report?

1 d-i        This bug is relevant to the development of debian-installer.
(...略...)
8 none

Please select tags: (one at a time) [none]↵
```

←重要度を決める。デフォルトは「normal」。重要度の種類については前号の本連載を参照

タグを付ける。デフォルトは「なし(none)」。
←タグの種類については前号の本連載を参照

▼図15　バグ報告を送信するかどうかの確認

```
Report will be sent to "Debian Bug Tracking System" <submit@bugs.debian.org>
Submit this report on dh-make (e to edit) [y|n|a|c|E|i|l|m|p|q|d|t|s|?]? y↵
```

▼図16　添付ファイルの指定

```
Submit this report on dh-make (e to edit) [y|n|a|c|E|i|l|m|p|q|d|t|s|?]? a↵
Choose a file to attach:
```

▼図17　送信の中止

```
Report will be sent to "Debian Bug Tracking System" <submit@bugs.debian.org>
Submit this report on dh-make (e to edit) [y|n|a|c|E|i|l|m|p|q|d|t|s|?]? q↵
Bug report written as /tmp/reportbug-dh-make-20150114-11570-Mekrfo
```

ばバグ報告が送信されます。

[a]を押すと添付するファイルの指定について尋ねられます(**図16**)。

また、[q]を押せばreportbugを終了します(**図17**)。送信を取りやめた場合でも、ファイルは/tmp/reportbug-<package>-<time>……という形で保存されますので、あとから送りなおすことも可能です。

バグ報告での文章のお作法

バグレポートで、常に念頭に置くべきことは「確認方法と再現性」です。報告者の手元で起こっている事象が、開発者の手元でも確認できなければ、修正のしようがありません。そのためには、次の3点を意識しましょう。

①step by stepで再現までの手順を記述する
②バグが発生する環境の情報をできる限り詳細に記述する
③可能であればログを添付する

①については、reportbugの報告テンプレートにある4つの質問に対して答えましょう。

- What led up to the situation?
  何をしたらその状況になったか？
- What exactly did you do (or not do) that was effective (or ineffective)?
  何をした(あるいはしなかった)ら効果があった(あるいは効果がなかった)か、を正確に
- What was the outcome of this action?
  この作業の結果、何が起こったか
- What outcome did you expect instead?
  期待していたのはどのような結果だったのか

②については、reportbugが環境変数やパッケージバージョンなどの情報を収集してくれるので、それをそのまま報告すればOKです。

③については、Ubuntuの「Apport」やFedoraの「abrt」などのツールはクラッシュ時にログを送付してくれるようになっていますが、このあたりについては、残念ながらDebianは遅れています[注2]。とはいえ、レポートツールのログ収集機能を「動作が重くなる」などと言って、切ってしまわれる方もいるので、なんとも難しい部分もありますね。/var/log以下のログファイルから有用な情報を切り出すことができれば、それを添付してあげましょう。

LTSスポンサーにグリーが参加

以前の記事でも取り上げましたが、Debian 6 "Squeeze"は現在、LTSサポート[注3]の期間中です。このLTSサポートはセキュリティチームの担当ではなく、LTSサポート専任のチームが、LTSを必要とする企業からの資金援助を受けて行っています。現在のところ、いくつかの企業がLTSサポートに参加していますが、今回、グリー㈱様がBronzeスポンサーとして参加されました[注4]。前回のさくらインターネット㈱様のsecurity.debian.orgミラー同様、こちらもアジアの企業としては初のLTSスポンサー参加となります。今回、筆者はコーディネイトをさせていただきましたが、ほかの組織でも興味があるところは同様に相談に乗りますので、ぜひご連絡ください[注5]。SD

注2) 一応Apportは、Debianにも存在してはいますが、experimentalリポジトリに入っているだけの状況です。

注3) Long Term Supportの略。通常のサポート期間と合わせて5年間のセキュリティアップデートサポートを行います。Debianでは、2年間のサポート+次の安定版が出てからの1年間のサポート=合計3年間のサポートが行われています。

注4) URL http://www.freexian.com/services/debian-lts.html参照。記事執筆段階では若干調整中。

注5) Debian JP Project理事会<board@debian.or.jp>宛でお願いします。

ひみつのLinux通信

作)くつなりょうすけ
@ryosuke927

第14回 地球危機一髪

正月太りのため、ビリーズブートキャンプ一式をタンスの奥から出してワークアウトに励む先生に愛のツイートを！

to be Continued

98年頃に宇宙人の映画がありまして、地球侵略を目論むエイリアンから攻撃を受ける地球上の各国政府、反撃を試みるも宇宙船にはバリアが張られていて攻撃が届かない。最後の手段はMacで作成したコンピュータウィルスを母艦に感染させ、バリアを解除させて攻撃・撃沈したとさ。「地球のPCで作ったウィルスって他の星のPCでも動くんだ！　もしかしてJavaで書かれてたのか!?　やっぱりこれからはJavaやな！」と当時ホットなJavaで少しだけ妄想を膨らませました。若いですね。本当にエイリアンが来た時にはまったく参考にならない映画でしたが、元ラッパーの俳優が、砂漠で愚痴を言いながら捕虜の宇宙人を引きずって歩くシーンは大好きです。

第36回

プロセスのメモリ状態を設定する PR_SET_MM_MAP

Text：青田 直大　AOTA Naohiro

今月はプロセスの「メモリ状態」を設定するためのAPIである、PR_SET_MMの新しいバージョンであるPR_SET_MM_MAPについて紹介します。

checkpoint/restoreとは

CRIUは、Checkpoint/Restore In Userspaceの略称です。CRIUプロジェクトでは、動作中のプロセスの状態をディスク上にダンプ(checkpoint)し、その後ディスク上のデータからプロセスを再開する(restore)ことをuserspaceで実現するツールを作成しています。

checkpoint/restoreをuserspaceで実現するにはkernel内の情報を取得・設定するインターフェースが必要になります。CRIUではそのためのカーネルパッチの開発も行っています。今回紹介するPR_SET_MM、PR_SET_MM_MAPもそのためのインターフェースの1つになります。

プロセスのメモリ状態

checkpoint/restoreはさまざまなプロセスの「状態」を保存・復元する作業と言うことができます。プロセスのメモリ状態もそのうちの1つとなります。まずは、このプロセスのメモリ状態について見ていきましょう。

プログラムが実行されると、まずカーネルがプログラムのELFプログラムヘッダの情報を用いて、プログラムのバイナリをメモリに読み込みます。ELFプログラムヘッダはreadelfを使うことで見ることができます(**図1**)。

カーネルはプログラムヘッダのうち、TypeがLOADのものを見て、ファイルからメモリへの割り当てを行います。すなわち**図1**に示した/bin/sleepの場合、0x00400000から0x00406074に読み込み(R)と実行可能(E)な領域と、0x00606e10から0x006073e0に読み書き(RW)可能な領域が作られ、それぞれ/bin/sleepにマップされます。

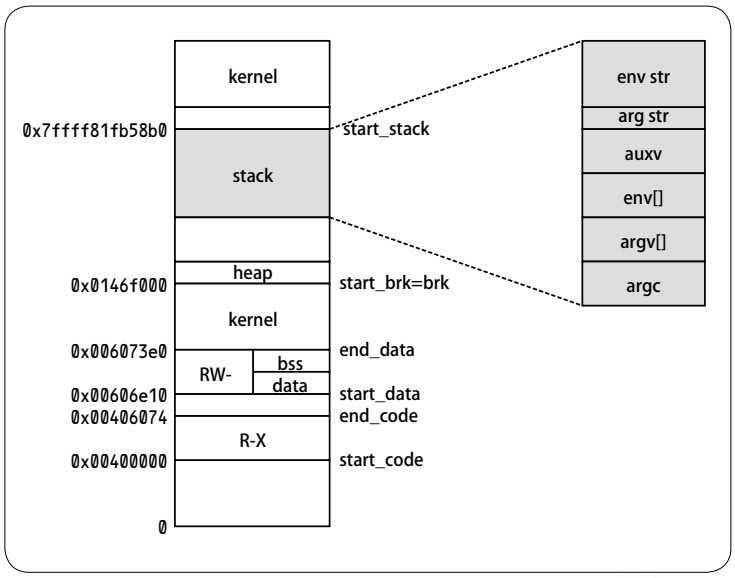
ここで2番めのLOAD(0x00606e10から0x006073e0)のファイルサイズが0x0424、0x05d0であることに注意してください。この2つの差の部分は一般にはBSSセグメントの部分であり、無名ページの割り当て、または0書き込みによって0で初期化された領域となります。また、このマップ作業と同時にメモリ領域の次のような値が計算されます。

・コード領域を示す(start_code、end_code)

- データ領域を示す(start_data、end_data)
- heap領域を示す(start_brk、brk)
- スタック領域の先頭を示す(start_stack)

start_codeはマップされた領域の開始位置のうち一番小さいもの、end_codeは実行可能な領域の終了位置のうち一番大きなもの、start_dataはマップされた領域の開始位置のうち一番大きなもの、end_dataは前述したBSSセグメントを除いた部分の終了位置のうち一番大きなもの、start_brkおよびbrkはBSSセグメントの末尾からランダムなオフセット分ずらしたところ、start_stackはメモリ末尾のカーネルが使う領域を除いた部分からランダムなオフセット分ずらした位置に設定されます。まとめるとカーネルによってプログラム

▼図2　BPFとeBPF



▼図1　/bin/sleepのプログラムヘッダ

```
$ readelf -l /bin/sleep
Elf file type is EXEC (Executable file)
Entry point 0x4017ec
There are 10 program headers, starting at offset 64

Program Headers:
  Type           Offset             VirtAddr           PhysAddr
                 FileSiz            MemSiz              Flags  Align
  PHDR           0x0000000000000040 0x0000000000400040 0x0000000000400040
                 0x0000000000000230 0x0000000000000230  R E    8
  INTERP         0x0000000000000270 0x0000000000400270 0x0000000000400270
                 0x000000000000001c 0x000000000000001c  R      1
      [Requesting program interpreter: /lib64/ld-linux-x86-64.so.2]
  LOAD           0x0000000000000000 0x0000000000400000 0x0000000000400000
                 0x0000000000006074 0x0000000000006074  R E    200000
  LOAD           0x0000000000006e10 0x0000000000606e10 0x0000000000606e10
                 0x0000000000000424 0x00000000000005d0  RW     200000
  DYNAMIC        0x0000000000006e28 0x0000000000606e28 0x0000000000606e28
                 0x00000000000001d0 0x00000000000001d0  RW     8
  NOTE           0x000000000000028c 0x000000000040028c 0x000000000040028c
                 0x0000000000000020 0x0000000000000020  R      4
  GNU_EH_FRAME   0x000000000000528c 0x000000000040528c 0x000000000040528c
                 0x000000000000026c 0x000000000000026c  R      4
  GNU_STACK      0x0000000000000000 0x0000000000000000 0x0000000000000000
                 0x0000000000000000 0x0000000000000000  RW     10
  GNU_RELRO      0x0000000000006e10 0x0000000000606e10 0x0000000000606e10
                 0x00000000000001f0 0x00000000000001f0  R      1
  PAX_FLAGS      0x0000000000000000 0x0000000000000000 0x0000000000000000
                 0x0000000000000000 0x0000000000000000         8
```

が**図2**のように配置されるということになります。

さらにカーネルはスタックに、引数、環境変数、Auxiliary Vectorの情報を書いていきます。Auxiliary Vectorとはプラットフォーム名や、ページサイズ、ユーザIDなどのシステム情報を渡す配列です。**図3**のように環境変数LD_SHOW_AUXVを設定することで見ることができます。これらの位置についても、引数の範囲についてarg_startとartg_end、環境変数の範囲についてenv_startからenv_end、そしてAuxiliary Vectorのデータがsaved_auxvに保存されます。

▼図3　Auxiliary Vectorの表示

```
$ LD_SHOW_AUXV=1 /bin/true
AT_SYSINFO_EHDR: 0x7fff363fe000
AT_HWCAP:        bfebfbff
AT_PAGESZ:       4096
AT_CLKTCK:       100
AT_PHDR:         0x400040
AT_PHENT:        56
AT_PHNUM:        10
AT_BASE:         0x7f6385040000
AT_FLAGS:        0x0
AT_ENTRY:        0x4013b2
AT_UID:          1000
AT_EUID:         1000
AT_GID:          1000
AT_EGID:         1000
AT_SECURE:       0
AT_RANDOM:       0x7fff363c7609
AT_EXECFN:       /bin/true
AT_PLATFORM:     x86_64
```

プロセスのダンプ

CRIUのプロセスダンプの過程では、前項のようなカーネルが設定した情報を、さまざまなインターフェースを通じて読み取り保存することになります。読み取るべき情報には次のものがあります。

- メモリ上のマップ状況
- コード・データ領域の範囲
- スタック・heapの位置
- Auxiliary Vector

まず、メモリ上のマップ状況は"/proc/<pid>/maps"を使ってみることができます(**図4**)。/bin/sleepの0x00400000から0x00406074がページ単位(=0x1000byte)にアライメントされ、メモリ上の0x00400000から0x00407000にそのまま配置されています。

▼図4　メモリマップの表示

```
$ sleep 3 & pid=$!
[1] 26232
$ cat /proc/$pid/maps
00400000-00407000 r-xp 00000000 00:20 9648594                            /bin/sleep
00606000-00607000 r--p 00006000 00:20 9648594                            /bin/sleep
00607000-00608000 rw-p 00007000 00:20 9648594                            /bin/sleep
0146f000-01490000 rw-p 00000000 00:00 0                                  [heap]
7ff6caef1000-7ff6cb4df000 r--p 00000000 00:20 15654918                   /usr/lib64/locale/ロ
locale-archive
7ff6cb4df000-7ff6cb670000 r-xp 00000000 00:20 15654899                   /lib64/libc-2.20.so
7ff6cb670000-7ff6cb870000 ---p 00191000 00:20 15654899                   /lib64/libc-2.20.so
7ff6cb870000-7ff6cb874000 r--p 00191000 00:20 15654899                   /lib64/libc-2.20.so
7ff6cb874000-7ff6cb876000 rw-p 00195000 00:20 15654899                   /lib64/libc-2.20.so
7ff6cb876000-7ff6cb87a000 rw-p 00000000 00:00 0
7ff6cb87a000-7ff6cb89b000 r-xp 00000000 00:20 15654900                   /lib64/ld-2.20.so
7ff6cba48000-7ff6cba4b000 rw-p 00000000 00:00 0
7ff6cba99000-7ff6cba9a000 rw-p 00000000 00:00 0
7ff6cba9a000-7ff6cba9b000 r--p 00020000 00:20 15654900                   /lib64/ld-2.20.so
7ff6cba9b000-7ff6cba9c000 rw-p 00021000 00:20 15654900                   /lib64/ld-2.20.so
7ff6cba9c000-7ff6cba9d000 rw-p 00000000 00:00 0
7fff81f96000-7fff81fb8000 rw-p 00000000 00:00 0                          [stack]
7fff81fe0000-7fff81fe2000 r--p 00000000 00:00 0                          [vvar]
7fff81fe2000-7fff81fe4000 r-xp 00000000 00:00 0                          [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0                  [vsyscall]
```

一方で、2つ目の領域である0x00606e10から0x006073e0は、読み取り専用の0x00606000から0x00607000と 読み書きできる0x00607000から0x00608000とに分けられています。これは動的リンカによるものです。

/bin/sleepをディスアセンブルして、nanosleepなどのglibcの関数の呼び出し位置を捜してみるとわかりますが“callq 401370<nanosleep@plt>”といった呼び出しを行っており、callqされている<nanosleep@plt>の部分をディスアセンブルすると**図5**のようになっています。ここには“0x205d42(%rip)(= 0x6070b8)”に書かれているアドレスにジャンプするコードが書かれています。動的リンクでは実行時にライブ

▼図5　/bin/sleep <nanosleep@plt>部分のディスアセンブル

```
$ objdump -d /bin/sleep | grep -A 5 '<nanosleep@plt>:'
0000000000401370 <nanosleep@plt>:
  401370:       ff 25 42 5d 20 00       jmpq   *0x205d42(%rip)        # 6070b8 <__ctype_b_
loc@plt+0x205b48>
  401376:       68 14 00 00 00          pushq  $0x14
  40137b:       e9 a0 fe ff ff          jmpq   401220 <__uflow@plt-0x10>

0000000000401380 <strrchr@plt>:
```

▼図6　/bin/sleepのセクション情報

```
$ readelf -S /bin/sleep
There are 27 section headers, starting at offset 0x7330:

Section Headers:
  [Nr] Name              Type             Address           Offset
       Size              EntSize          Flags  Link  Info  Align
  [ 0]                   NULL             0000000000000000  00000000
       0000000000000000  0000000000000000           0     0     0
…
  [16] .eh_frame         PROGBITS         00000000004054f8  000054f8
       000000000000b7c   0000000000000000   A       0     0     8
  [17] .init_array       INIT_ARRAY       0000000000606e10  00006e10
       0000000000000008  0000000000000000  WA       0     0     8
  [18] .fini_array       FINI_ARRAY       0000000000606e18  00006e18
       0000000000000008  0000000000000000  WA       0     0     8
  [19] .jcr              PROGBITS         0000000000606e20  00006e20
       0000000000000008  0000000000000000  WA       0     0     8
  [20] .dynamic          DYNAMIC          0000000000606e28  00006e28
       00000000000001d0  0000000000000010  WA       5     0     8
  [21] .got              PROGBITS         0000000000606ff8  00006ff8
       0000000000000008  0000000000000008  WA       0     0     8
  [22] .got.plt          PROGBITS         0000000000607000  00007000
       00000000000001c0  0000000000000008  WA       0     0     8
  [23] .data             PROGBITS         00000000006071c0  000071c0
       0000000000000074  0000000000000000  WA       0     0     32
  [24] .bss              NOBITS           0000000000607240  00007234
       00000000000001a0  0000000000000000  WA       0     0     32
  [25] .gnu_debuglink    PROGBITS         0000000000000000  00007234
       0000000000000010  0000000000000000           0     0     1
  [26] .shstrtab         STRTAB           0000000000000000  00007244
       00000000000000eb  0000000000000000           0     0     1
Key to Flags:
  W (write), A (alloc), X (execute), M (merge), S (strings), l (large)
  I (info), L (link order), G (group), T (TLS), E (exclude), x (unknown)
  O (extra OS processing required) o (OS specific), p (processor specific)
```

ラリがリンクされるので、そのときにならなければライブラリの関数がロードされたアドレスはわかりません。そこで“<nanosleep@plt>”のように間接的に呼び出すコードが使われています。

ここでアドレスを解決する方法を2つ考えることができます。1つはライブラリを読み込んだときにすべての関数のアドレスを解決しておく方法で、もう1つは実際に関数が呼び出されるときまでアドレス解決を遅延する方法です。

前者にはプログラムの実行開始時のオーバーヘッドが大きくなるというデメリットがありますが、セキュリティ上の利点もあります。アドレス解決を遅延する場合、一般にアドレスを保存する場所(nanosleepの場合でいう0x6070b0)は読み書きできるようにしてあります。ということは、この部分を書き換えることで任意の外部関数呼び出しをほかの関数の呼び出しにすりかえてしまうことができるということになります。

この問題は前者の方法を使っていれば、アドレス解決した領域を読み込み専用にしてしまうことで解決できます。 このアドレス解決の保管領域は“.got”セクションと“.got.plt”セクションに置かれ、“.got”セクションの方はライブラリのロード後にアドレス解決が行われて読み込み専用に設定されます。実際、“**readelf -S /bin/sleep**”でセクション位置を見てみる(**図6**)と、“.init_array”から“.got”までが0x00606000から0x00607000までの読み込み専用の範囲にあり、“.got.plt”、“.data”、“.bss”が0x00607000から0x00608000の読み書きできる領域に置かれていることが確認できます。

さらに**/proc/<pid>/stat**を見ることで、start_code、start_dataなどの値も見ることができます。statファイルには多くの数字が並んでいますが、ここまでの解説に関連した部分を抜き出すと**図7**のようになっています。この数字の意味は**表1**のとおりです。

プロセスの復元

最後に復元について見ていきましょう。プロセスを復元するコードのうちメモリ関連のコードはおもにCRIUのコードのうちrestorer.cの**__export_restore_task()**関数で行われています。ここでは、まずmmap()を使ってプロセスメモリ領域上のマッピングを復元しています。そして、start_code、end_code、start_data、end_dataなどの値を復元するために**prctl(PR_SET_MM)**システムコールを使います。たとえば、start_codeを復元するには“**prctl(PR_SET_MM, PR_SET_MM_START_CODE, start_code, 0);**”というコードが実行されます。PR_SET_MM_START_CODEの部分を変更することで、そのプロセスのstart_code、end_code、start_data、end_data、start_stack、start_brk、brk、arg_start、arg_end、env_start、env_endそしてauxvが更新されます。

prctl(PR_SET_MM)を使うことで、プロセスの領域を示すさまざまな値を変更できます。これを使えば、たとえばstart_dataをend_dataよりも後に設定しデータ領域のサイズが負であるかのように見せることができるようになります。すると、1プロセスが確保できるデータ領域のサイズ制限を回避できるようになってしまいます。そのため、プログラムが**prctl(PR_SET_MM)**を使って不正にシステムリソースを使ってしまわ

▼図7　statの表示

```
$ cat /proc/$pid/stat
26232 (sleep) S 8596 26232 8596 34819 26233 1077944320 109 0 0 0 0 0 0 0 25 5 1 0 15232380
10493952 190 18446744073709551615 4194304 4218996
140735374121136 140735374120760 140697950313696 0 0 0 0 18446744071579682587 0 0 17 7 0 0 0
0 0 6319632 6320692 21426176 140735374126721 140735374126729 140735374126729 140735374131177
0
```

ないように、**prctl(PR_SET_MM)**はCAP_SYS_RESOURCE権限がなければ動作しないように作られています。

ところが、このCAP_SYS_RESOURCE権限チェックが、user namespaceの復元時には障害となってしまいます。user namespaceというのは、namespaceの内のユーザID／グループIDを、namespaceの外のユーザID／グループIDにマップする機能です。これを使うことで、たとえばdockerコンテナ内でのroot権限をコンテナの外の一般ユーザにマップすることで、万一コンテナから外に影響することがあってもその影響を抑えることができるようになります。このuser namespaceを復元した場合、CAP_SYS_RESOURCE権限が落ちてしまうことがあります。そうすると、**prctl(PR_SET_MM)**が使えなくなり、プロセスの復元ができなくなってしまいます。この問題に対処するために導入されたのが**prctl(PR_SET_MM_MAP)**です。

PR_SET_MM_MAPの導入

prctl(PR_SET_MM_MAP)は、これまでの**prctl(PR_SET_MM)**とは異なりCAP_SYS_RESOURCE権限を必要としないように作られています。もちろんそれだけでは、先ほど述べたようにシステムリソースを不正に使われてしまいます。**prctl(PR_SET_MM_MAP)**では、start_code、end_codeなどのPR_SET_MMで設定できるすべての値を一度に設定できるようにして、より厳密な値のチェックを行うことで、リソースの不正使用を防止し、CAP_SYS_RESOURCEの排除を実現しています。具体的には次のチェックが行われています。

- code、data、brk、arg、envの範囲を示す変数の順序関係
- data領域よりも後ろにbrk領域があるか
- プロセスごとのデータ領域制限にかかっていないか
- Auxiliary Vectorのサイズが正しいか

こうした厳密なチェック(とくにアドレスの範囲チェック)を行うことで、**prctl(PR_SET_MM_MAP)**は**prctl(PR_SET_MM)**よりも優れており、より安全に**prctl(PR_SET_MM)**のコードを置き換えることができます。そのため、commitlogにも、反対がなければ**prctl(PR_SET_MM)**を削除しようと書かれています。SD

▼表1　statの数字の意味

項目	値	16進表現
pid	26232	-
tcomm	(sleep)	-
state	S	-
…		
start_code	4194304	0x00400000
end_code	4218996	0x00406074
start_stack	140735374121136	0x7fff81fb58b0
…		
start_data	6319632	0x00606e10
end_data	6320692	0x00607234
start_brk	21426176	0x0146f000
arg_start	140735374126721	0x7fff81fb6e81
arg_end	140735374126729	0x7fff81fb6e89
env_start	140735374126729	0x7fff81fb6e89
env_end	140735374131177	0x7fff81fb7fe9

March 2015

NO.41

Monthly News from jus

Japan UNIX Society

広島市立大学大学院情報科学研究科
林 直樹　HAYASHI Naoki

めざせ!　気遣いができるウェアラブルコンピュータ

今回は、2014年11月に広島で行われたインターネットコンファレンス（**写真1**）の模様を報告します。

インターネットコンファレンス

■インターネットコンファレンス2014
【日時】2014年11月4日（火）～11月5日（水）
【場所】アステールプラザ（広島市）

■カンファレンスの概要

同会は1996年からjusを含む複数の組織で共催しているインターネットに関する研究の発表会で、今回で19回目になります。2014年はjusに加え、日本学術振興会産学協力研究委員会インターネット技術第163委員会（ITRC）、日本ソフトウェア科学会インターネットテクノロジー研究会（ITECH）、WIDEプロジェクト（WIDE）の共催により開催され、参加者は2日間で約70名でした。

まず、本会合の中心となる論文発表は4つのセッションで構成され、11件の研究成果が発表されました。内訳は、プロトコルに関する発表が3件、ワイヤレスに関する発表が2件、OS／ミドルウェアに関する発表が3件、セキュリティ／アプリケーションに関する発表が3件で、それぞれ活発な議論が展開されました。

そのほかに、ポスター展示11件、デモンストレーション展示2件も行われ、各ブースでは発表者と聴講者が実際に対話しながら議論を行っていました。

招待講演では、広島市立大学大学院情報科学研究科システム工学専攻の谷口和弘氏による「耳飾り型ウェアラブルコンピュータの開発」、慶應義塾大学理工学部情報工学科の松谷宏紀氏による「ビッグデータ向け計算機アーキテクチャの研究動向と研究事例」、芝浦工業大学工学部通信工学科の森野博章氏による「ガラス球を用いた自由落下型深海探査機『江戸っ子1号』の開発と簡易型海中センシングへの応用に向けた展開」の3つの講演が行われました。

写真1　インターネットコンファレンスの会場の様子

■耳飾り型ウェアラブルコンピュータの開発

谷口和弘氏による招待講演の内容を報告します。

谷口氏は装着者の体温や脈拍、表情などをモニタリングできる外耳装着型センサを内蔵した「耳飾り型ウェアラブルコンピュータ（以降、耳飾り型WPC）」を開発しました。耳飾り型WPCには、口やまぶたの開閉、咀嚼（そしゃく）などの表情変化を検知可能なearable技術[注1]が用いられており、操作に手や視覚を必要としないのが特徴です。ユーザは何か別の作業をしなが

注1） イアラブル技術。URL http://earable.jp/

ら耳飾り型WPCを操作する、という並行作業が可能になる点で、Google Glassに代表されるグラス型ウェアラブルコンピュータとの差別化を図っています。実際に耳飾り型WPCを装着した状態でまぶたの開閉を行うことで、端末と連動して音楽を再生、停止する実験を行い、これに成功しています。

年配者のユーザが耳飾り型WPCを身につけて体温や脈拍、咀嚼などの情報をクラウド上に送信すれば、別の場所から生活リズムをモニタリングすることも可能です。この装置は、2015年度から広島市で掲げられている「広島市高齢者見守りプロジェクト」の要となる装置として期待されており、2016年の販売に向けて開発、調整が進められています。

ところで、ウェアラブルコンピュータを開発するうえで考えなくてはならない要素として、機能はもちろんですが、デザイン性も重要な要素として挙げられます。服飾という行為には防寒などのさまざまな意味がありますが、その根源には着飾った人間の魅力を向上させるという目的もあります。ウェアラブルコンピュータも人間が身につけるものである以上、装飾品としての一面、つまりアクセサリのような高いデザイン性が必要です。耳飾り型WPCでは、三日月と勾玉（まがたま）を模したデザインに生け花のような模様をあしらうなど、デザインについても創意工夫を行っています。

最後に、ウェアラブルコンピュータの理想型として、「気遣いができること」が挙げられました。たとえば、現在のカーナビゲーションは、道案内中にユーザが寄り道を行った場合、案内ルートに戻すよう再三の通知がなされます。しかし、「気遣いができるコンピュータ」はユーザの「寄り道したい」という意思を汲み取り、通知を控えるなどの処理が行えます。耳飾り型WPCは今後、ユーザの心的ストレスの検知が可能になる見込みがあり、この機能の発展によってはさまざまな「気遣い」が可能な装置になり得る、という内容で招待講演を終えられました。

ユーザはシステムに、機能だけではなく、デザイン性やユーザビリティも求めています。研究開発を行ううえでは機能面を重視しがちですが、一度ユーザの目線に立ち、あらゆる角度からユーザのためにシステムをブラッシュアップする必要があることをあらためて感じました。

■表彰論文／発表

本会では優秀な論文／発表を表彰しています。2014年度は次の論文が論文賞を受賞しました。

- 「無線網に対応した実時間映像音声伝送制御の検証のための時間推移ネットワークエミュレーション」
  村本衛一、香林誠、石井秀教、明石邦夫、
  知念賢一、篠田陽一

また、今後の発展が期待される学生の報告には学生奨励賞が授与されます。次の3件の論文が選ばれました。

- 「データセンター内における高速VM間通信用準仮想化ドライバの設計と実装」
  上野幸杜、植原啓介
- 「Android端末省電力化に向けたブロードキャストインテント発行とアプリケーションの因果関係の評価」
  早川愛、半井明大、竹森敬祐、山口実靖、小口正人
- 「データセンター環境におけるショートフロー通信改善手法の一提案」
  藤居翔吾、田崎創、関谷勇司

今回は、活発な議論を生んだポスター発表に対してポスター賞が授与され、次の2件が選ばれました。

- 「OpenFlowによる移動透過通信支援システムの提案」
  藤本大地、大石恭弘、林直樹、前田香織、
  近堂徹、相原玲二
- 「空調効率の改善を目的としたサーバ負荷配置方式」
  千喜良和明、橋本英明、浦田穣司、増尾剛

発表された論文は本会のWebサイト[注2]から参照できます。また、2015年度は大分大学の池部実氏を実行委員長に迎えて開催する予定です。SD

注2）URL http://www.internetconference.org/ic2014/

Hack For Japan
エンジニアだからこそできる復興への一歩

Hack For Japanスタッフ
佐々木 陽　SASAKI Akira
Twitter @gclue_akira
鎌田 篤慎　KAMATA Shigenori
Twitter @4niruddha
小泉 勝志郎　KOIZUMI Katsushiro
Twitter @widesilverz
高橋 憲一　Takahashi Kenichi
Twitter @ken1_taka

第39回 地方のコワーキングスペース

ここ数年コワーキングスペースと呼ばれる場所が増えてきていますが、その勢いは東京のみならず地方にも広がっているようです。そこで今回はHack For Japanと交流のある福島県と宮城県のコワーキングスペースについてご紹介したいと思います。

福島県のコワーキングスペース

学生や熟練エンジニアを巻き込む会津若松市のスペース

Fab蔵(くら)[注1]は、2014年6月に福島県の会津若松市にオープンしたものづくり支援施設です。名前のとおり、"Fabrication(=製造)蔵"です。築120年の蔵に、レーザーカッター、3D Printer、簡易リフロー、電動ミシン、電動工具などを設置し、IoTなものづくりを支援しています。

会津若松市では1960年代から半導体産業を中心に工場の進出が活発でした。しかしながらここ10年で、ピーク時と比較して製造品出荷額、従業員数ともに半減しています。これは半導体関連産業での規模縮小が要因で、多くのエンジニアが職を失ったことになります。

1993年には会津大学が日本初のコンピュータ専門大学として設置され、コンピュータアーキテクチャー全般のユニークな授業が実施されています。

そして2011年3月11日、東日本大震災が発生します。会津若松市は震度5強程度で大きな被害は発生しなかったのですが、100km程度離れた沿岸部の被害は甚大で、原発事故が発生しました。原発事故の放射能の値を計測するためにはガイガーカウンターという計測機が必要だったのですが、当時は買おうとしても在庫がなく、値段も2倍、3倍に跳ね上がっていました。

そんな中、Hack For Japan主催の意見交換会が2011年4月25日に会津大学で開催されました。そこに自作ガイガーカウンターを持参して参加してくれた福島市在住の寺脇さん(株式会社シーエー)と筆者(佐々木)が共同で、Android端末にBluetooth経由でガイガーカウンターを接続し、その値をAndroidの画面に表示させることに成功します。見た目はかっこよくありませんが、自作ガイガーカウンターです。

それがキッカケとなり、Hack For Japanや日本Androidの会のコミュニティやTwitterの力で、日本中の自作ガイガーカウンター愛好家とのやりとりが始まります。その中で四国在住の今岡さん(今岡工学事務所)と河野さん(オープンフォース)が、2011年6月11日に四国から会津若松に駆けつけ、ガイガーカウンターを自作する「福島ガイガーカウンター勉強会」を会津で開催することになります。勉強会はWorkshop形式で実施し、40名参加のもとで、自作したガイガーカウンターを持ち帰ってもらうことに成功しました。これが、会津で開催されたはじめてのIT系ハードウェア勉強会となりました。

このようにガイガーカウンターを通して、緩やかにハードウェアを自作するコミュニティが会津若松を中心に形成されていくことになり、2012年には筆者(佐々木)の会社であるGClueの社内を改造して「Open Hard Cafe」という3D Printerや中国から輸入したレーザーカッターを備えた自作Fabスペースをオープンすることになります。小規模なFab設備ではありましたが、そういった試みの積み重ねを通して2014年6月に会津若松市の支援のもとで、Fab蔵をオープンするに至りました。

注1　http://www.fabkura.org/

●ものづくりイノベーションのハブを目指すFab蔵

Fab蔵での活動は、よりオープンであるべきと考えています。オープンソース・ハードウェア（OSHW）基準書1.0に記載のあるとおり、「オープンソース・ハードウェアは、設計図のオープンな交換による知識の共有と商品化を奨励すると同時に、テクノロジーを統御する自由を人々に与える」べきであると考えています。テクノロジーには、オープンとクローズの適度なバランスが重要で、原発事故と向き合わなければいけない福島県は、この適度なテクノロジーのバランスを、人材育成を通して根付かせる必要があると考えています。また、会津若松という土地柄、規模が小さくなってしまった半導体産業ですが、そこで仕事をしていたエンジニアはたくさんいるはずです。そして会津大学をコアとした学生や、教授陣、地元ベンチャーとのものづくりを介した交流を通して、Fab蔵が新しいものづくりのイノベーションのハブになれないかと考えています。

そんなFab蔵のターゲットは、OSHWを軸にしたIoTイノベーションを会津若松で加速することです。そのためには、最初に一緒に戦える仲間を作る必要があります。そこで、週2回程度のWorkshopと、月2回程度の講演会、月2回程度の60歳以上のエンジニアを講師に招いたWorkshopを実施しています（**写真1**）。WorkshopではOSHWを用いた勉強会を、講演会ではIoTの先駆者たちに会津若松に来ていただいて講演してもらっています。また、60歳以上のエンジニアを講師に招いたWorkshopでは、地域に埋もれているものづくりのノウハウを継承することを目的としています。

◆**写真1　Arduino入門Workshopの様子**

Workshop（週2回）の例

- Arduino互換基板を作ろう（オリジナルArduino互換基板を自作）
- 3D Printerを作ろう（3D Printerをコントロール基板から作成）
- ロボットカーを自作しよう（Fab蔵オリジナルロボットカーを自作）

60歳以上のエンジニアを講師に招いたWorkshop（月2回程度）の例

- ロボットカーをハックしよう
- イヤホンアンプの構造を知ろう

講演会

- Fab蔵 IoT Night

また、現在進行中の試みとしてハードウェアの地産地消のトライアルをしています。Fab蔵のWorkshopで使うハードはすべて、回路設計／基板製造／部品実装をオリジナルで行っています。ハードウェアの地産地消を確立することで、地域に根付いた新しいオープンソースハードウェアの産業を起こせないかと考えています。

ハードウェアの地産地消（Fab蔵）

- Arduino互換ボードをFab蔵講習会では使用、地元の高校に無料配布
- 3D Printerを自作

これらの活動は、すべてFab蔵Docsのページ[注2]にチュートリアルという形で、Creative Commons 2.0のライセンス化で公開しています。Fab蔵では、OSHWを軸にしたIoTイノベーションを、会津若松の保持している地域リソースを最大化することで実現しようと考えています。ぜひ、福島にお越しの際は、Fab蔵へお立ち寄りください！

起業家を育成する郡山市のスペース

福島県は非常に広く、先にご紹介した会津のFab蔵のほかに、中通り、浜通りといった他地域の中に

注2　https://sites.google.com/a/gclue.jp/fab-zang-docs/

もコワーキングスペースが徐々に誕生しつつあります。昨年11月には株式会社ツクルバが全国にフランチャイズしているコワーキングスペース「co-ba」の新しい拠点として、郡山市に「co-ba koriyama[注3]」がオープンしました（**写真2**）。郡山を中心に活動している一般社団法人グロウイングクラウドが運営を担っています。

グロウイングクラウドは、復興・地域活性化に欠かせない要素の1つとして起業家の育成を掲げています。創業を志す人の情報収集・人脈構築・スキルアップ・創業相談・起業練習の場として、郡山市創業支援事業「フロンティア.netこおりやま」の受講者には受講期間中はco-ba koriyamaを無償で提供しているそうです。こうした起業家を支援する土壌は各地にも拡がりつつあります。

また、その利用者は学生から年配の方まで多岐にわたり、co-ba koriyamaの象徴でもある卓球台が、異なる目的を持ってco-baに訪れる人たちのコミュニケーションにも一役買っているとのことでした。

●コワーキングスペースのオープンラッシュか!?

同じく福島県郡山市を中心に活動しているITスキルアップコミュニティ「エフスタ!![注4]」の大久保仁代表によると、本誌2014年10月号で紹介したエフサミが大盛況だったこともあってか、ここのところコワーキングスペースの運営を考えている団体からの問い合わせが増えてきているそうです。

そうしたコワーキングスペースの運営を検討しているNPO法人や団体はさまざまな業種のコミュニティに対し、ニーズをヒアリングしているとのこと。東北だけに留まらず、東京などでも活動しているエフスタ!!のようなコミュニティの存在は、コワーキングスペースの立ち上げを検討している団体からすると、利用者として重要な存在なのかもしれません。そして、コワーキングスペースの認知度が低い状態をいち早く脱するために、クリエイターからの認知獲得の必要性が団体の方とのやりとりを通して伝わってくるそうです。エフスタ!!としてもそ

◆写真2　co-ba koriyamaオープニング時の様子

うした機運を活かし、エンジニアの気持ちにも火をつけ、活動が活発ではない地方という問題に対して、福島の土地から、いかに盛り上げていくかということを考えているとのことでした。

もしかすると、2015年の郡山はコワーキングスペースのオープンラッシュになるかもしれません。

宮城県のコワーキングスペース

バラエティ豊かな仙台市のスペース

仙台にもコワーキングスペースが増えつつあります。仙台駅から歩いて行ける範囲でも「クラウドガーデン」「ソシラボ」が、少し離れて「ごくり」「ノラヤ」「ココリン」「mag」があります。今回は仙台のコワーキングスペースの中でも特徴的な「ごくり」と「mag」を紹介します。

●ごくり

ごくり[注5]はファイブブリッジという地域活性化をはかる拠点に作られたコワーキングスペースで、ワンドリンクで滞在できるユニークな料金体系です。300円のほの香コーヒー、200円の緑茶・紅茶などがあります。

料金体系以上の特徴がごくりにはあります。それは「動画配信するための設備があること」。ここで収録・配信されるUstream番組も多くあります。IT系の番組もあり「東北ITニュース[注6]」という東北のITコミュニティの活動を紹介する番組も月に一度配信

注3　http://tsukuruba.com/co-ba/koriyama/
注4　http://efsta.com/
注5　http://www.five-bridge.jp/gocre/
注6　http://tohokuitnews.info/

されています。

この番組は毎月第3日曜（変動あり）の20:00～22:00に配信されています。このときもドリンク代のみでごくりに訪れての観覧も可能です。予定が合う方は参加してみてはいかがでしょうか。

映像設備レンタルもあり、映像設備は1時間2,000円、オペレーターも1時間2,000円となっています。Ustream配信ありのイベントを行うのに非常に適したスペースです。

●mag

コワーキング・カフェmag[注7]は仙台駅からは離れた南光台というところにあるコワーキングスペースです。コワーキングスペースとしては異色の「マンガ、アニメ、ゲーム」を題材としていて、実は「mag」という名前もこの頭文字をつなげたもので、プラモショップの隣という立地もあってか、プラモとコワーキングスペースの特性を活かした「プラモソン」というイベントも行われています。

また、magは萌えキャラの拠点としての活動もしていて、各種イベントには萌えキャラのショップを出展しています。扱っている萌えキャラには東北ずん子のほかにも、本誌2014年8月号の本連載記事にも載ったハッカソン「島ソン」で生まれた「渚の妖精ぎばさちゃん（TwitterID @gibasachan）」も扱われています。

隣のプラモショップで買ったプラモを作る人や、萌えキャラのグッズの作業をする人たちが多く集まるコワーキングスペースです。同好の士を探すのに訪れてみてはいかがでしょうか。

地元民への教えを歓迎する石巻市のスペース

●伊藤学

石巻市にある「伊藤学[注8]」は、Hack For Japanにとって毎年夏の恒例イベントとなっている石巻ハッカソンの運営母体でもあるイトナブ石巻が開設したコワーキングスペースです（**写真3**）。2014年7月にできたスペースで、石巻ハッカソンや定期的に開催されている東北TECH道場[注9]の石巻道場の場所にもなっています。ほかにもさまざまなイベントの会場となることも多く、1週間に渡って東京のIT企業の合宿が行われたり、2014年12月には情報レスキュー隊であるIT DART[注10]の稼働検証のための訓練もここを本部にして行われました。

このコワーキングスペースの最大の特徴はその料金体系です。1日1,000円、5日分の3,000円のチケット、1ヵ月で6,000円という料金が設定してあるのですが、イトナブに出入りする地元の若者に何か（開発、デザイン、人生など）を教えてくれた場合は利用料金が無料となります。地元の若者が活躍できる環境を作るのが目的でもあるイトナブにとって、外からやってきた人に何か教えてもらえるということは、この上ない良い刺激となるためそのような形式をとっています。

この号が発売されるのは2月中旬、その時期の石巻は雪は少ないもののとても寒いのですが、元気のよい若者と、こうこうと燃えるペレットストーブがこのスペースを暖かい場所にしてくれています。

最後に

ここまでご紹介したように、それぞれの場所で特色のあるコワーキングスペースが誕生してきています。福島や宮城にお出かけの際はぜひ訪れてみてはいかがでしょうか。東京にはない新たな出会い、可能性が見つかるかもしれません。SD

注7　http://office-mag.net/
注8　http://itomanabu.com/
注9　http://www.tohokutechdojo.org/
注10　http://itdart.itxsaigai.org/

◆写真3　イトナブのデザイナーによる季節ごとに変わる看板が皆さんをお迎えします

温故知新
IT むかしばなし

第42回

BASIC

Software Design 編集部

はじめに

今やプログラミング言語のBASICというと「使ったことがない」「知らない」という方が多くなってきていますが、パソコンが発売され始めた時期は、ほとんどのマシンでBASIC[注1]が動いていました。今回は、そのBASICについてお話します。

ROM-BASIC

高校時代、友人の持っていたシャープのポケコンPC-1211を触ったのが、筆者にとって初めてのBASICとの遭遇でした。その後、カシオのFX-702Pを入手し、そのころブームだったバイオリズム[注2]のプログラムを作ったりしていました。

当時の1980年前後といえば、タイトーのアーケードゲーム、スペースインベーダーが大流行した時分で、浪人中の筆者は秋葉原でパソコンが触れるショップに日参していました。当時の秋葉原の聖地と言えば、駅前の秋葉原ラジオ会館の7階で、NECの「Bit-INN」や日立の「GAIN」などのショールームがあり、パソコンが自由に触れたのでパソコン少年で賑わっていました[注3]。

そのころの人気機種は、NECのPC-8001がダントツで、Bit-INNにあったPC-8001には大勢の待ち行列ができるほどの人気でゆっくり触れませんでした。そのため筆者は、GAINで比較的空いていた日立のベーシックマスターレベル2でBASICを触っていました。

当時のパソコンは、電源を入れるとすぐにROM-BASICが起動し、画面にはBASICの起動メッセージとプロンプトが現れ、そこに直接コマンドや行番号を付けた命令を入力するといった、BASICとOSが一緒になったようなものでした。また、入力したプログラムは、(ファックスのような)音として別途用意したカセットデッキで録音して保存し、読み込む際にはカセットテープを再生するというしくみでした。

クリーンコンピュータ

異色だったのはシャープのMZ-80Kで、パソコン本体とディスプレイ、カセットデッキが1つになった筐体でした。起動時にはBIOSモニタと呼ばれる最小限のシステムが起動し、そこからS-BASICをカセットデッキからRAM部分に読み込んで実行するものでした。今では当たり前のことですが、RAM上に言語を読み込んで実行するために、クリーンコンピュータという名称が付けられていました。そのため、ほかの言語なども実行でき、HudsonのHu-BASICなども実行でき、当時のBASICの中では実行速度も高速でした。

各社のBASIC

NECのPC-8001にはN-BASIC、富士通のFM-8にはF-BASICのように、ROM-BASICには各社とも独自のものが搭載されていました(どちらもMicro

注1) Beginner's All-purpose Symbolic Instruction Codeの頭文字をとったもの。1964年に米ダートマス大学で開発された。初期のBASICはほとんどMicrosoft製だった。

注2) 生まれた日を起点に生存日数を計算し、身体・感情・知性を23・28・33日周期として各日の調子を予想するもの。イメージ的には占いに近い。

注3) 現在の秋葉原ラジオ会館は2014年に建て替えられたもので、昔の面影は残っていない。

soft系)。またシャープのMZ-80シリーズはS-BASICとHu-BASICが使えましたが、機種ごとに独自の方言があり、パソコン誌に載っているゲームは、それぞれの機種特有のものでした。

たとえば画面に文字を表示するPRINT命令は、PRINT "TEST"と入力するところを? "TEST"のように省略できたり、LIST命令をL.と略して入力(現在の入力補完の前身)できたりしました。またハードウェアの仕様にも差があり、メモリを直接操作するPEEK、POKE命令や、I/Oポートで直接やりとりするIN、OUT命令などもあり、特定の機種のプログラムを別の機種で動くようにする「移植」という記事がパソコン雑誌によく掲載されていました。

行番号とGOTO文

今ではどの言語も、ブロック単位で分岐命令やループを記述したり、ラベルに対してジャンプを指示したりしますが、初期のBASICでは、各行の先頭に行番号をつけ、その順番が少ないものから順番に実行し、GOTO文で行番号を指示してジャンプを行っていました。GOTO 120(120行にジャンプ)などという、行き先がどういった場所かわからない記述方法であるために、書き間違いやバグによって無限ループに入るプログラムも多かったです。また、GOTO文を多用しているプログラムはスパゲティと揶揄(やゆ)されていました。

DISK-BASIC

その後、パソコンでのプログラムやファイルのやりとりは、最初は8インチ、その後5.25インチのフロッピーディスク(FD)を使って行えるようになりました。320KBのFDが千円以上した時代です。FDが使えるようになったことにより、入出力を高速に行えるようになり、FD対応のDISK-BASICが主流になりました。

1981年にNECから発売されたPC-8801は、N88-BASICを搭載していましたが、その後N88-DISK BASICが発売され標準で搭載するようになりました。当時はパソコンはゲーム機として利用される場合が多く、パソコン少年たちは、雑誌に掲載されたプログラムを手入力で打ち込み、それをFDに保存して遊んでいました。当時はPC-8801と並んで富士通のFM-7、シャープのMZ-700なども人気があり、それぞれDISK-BASIC化されていきました。

MS-DOS時代の到来

1983年にMicrosoftからMS-DOSが発売され、やっとOS上で動くシステムが確立されました。PC-9801に標準で搭載されるようになると、MS-DOS上で動くN88-BASIC(86)や、そのコンパイラも発売されました。これによってパソコン上のBASICで作られたプログラムも高速で動くようになり、またコンパイルして、実行ファイルだけを公開すればよくなったために、ソースリストを見られないで済むといったメリットもありました。しかし、コンパイルされたコードの実行にはN88BASIC.LIBというサイズの大きなランタイムライブラリが必要でした。

1985年には、Microsoftから統合環境としてQuickBASICも発売され、デバッグ環境やコンパイラなどを内蔵し、便利になったのを実感しました。また1992年にはMS-DOS版のVisual BASICも発売されました。

また、このころになるとC言語や、ほかの言語[注4]も使用できるようになり、BASICのシェアが下がり始めたと思います。

Windows時代の到来

Microsoft Windowsが普及し始めたのは1992年に3.1が発売されてからだと思います。1995年にはWindows 95が発売され、Visual BASIC 4.0がリリースされました。またMicrosoft Office 95から、Excelをはじめとして WordやAccessにもVisual BASIC for Applications(VBA)が搭載されるようになりました。その後、1998年には現在も利用者が多いVisual BASIC 6.0がリリースされ、2002年には言語仕様が一新されたVisual BASIC .NETがリリースになり、現在のVisual Studio 2013に続くのはみなさんがご存じのとおりです。SD

注4) ボーランド社のTurbo PascalやTurbo Cなどが流行ってました。

Report

GoAzure 2015、開催

1月16日、ベルサール渋谷ファースト（東京都渋谷区）にて、クラウドサービス「Microsoft Azure」のコミュニティイベント「GoAzure」が催された。GoAzureは、マイクロソフトとJapan Azure User Group（Azureのユーザコミュニティ、通称JAZUG）との共催イベントで、2012年の第1回から約3年ぶりの開催となる。Azureユーザによるセッションやハッカソンが多数行われるユーザ寄りのイベントとなった。本記事では基調講演の模様をレポートする。

Develop your superpower with the Cloud

▲米マイクロソフト社 Scott Hanselman氏

米マイクロソフト社のPrincipal Program Managerを務めるScott Hanselman氏が招かれ、Azureについて、マイクロソフトのオープンソース化を含めた今後についてセッションを行った。

始めにScott氏は、彼の7歳の息子がMinecraft上に作った「Azureのデータセンター」をモニターに映し、「Minecraftという道具を与えたことで息子はこんなにもすばらしいものを完成させ、創造する喜びを感じられたのだ」と語った。彼の息子のようにハードルや制約を感じることなくものを作ることに集中できたことは、クラウドの世界において最も重要だという。クラウドは往々にして人から職を奪うものだと指摘されるが、7歳の息子が大きな力を発揮できたように、新しいワークフローを生むのだと強調した。

Scott氏は趣味で20ほどのWebサービスをAzureで運用している。開発・デプロイがすぐにでき、サービスの人気が上がればすぐにスケールアップ、下がればすぐにスケールダウンと、スケーリングが容易なことがAzureの魅力である。氏は実際に手元のマシンでAzureコマンドラインツールを使い、Node.jsやASPのWebアプリのテンプレートをGitを使ってデプロイしてみせた。次に氏は、運用しているWebサービスの1つである口述筆記アプリ「myEcho」についてデモを行った。氏がiPhoneに向かって英語や日本語で話しかけると、手元のマシンのMicrosoft Wordにテキストとして表示された。人気が出てきたこのアプリをさらにスケールアップしようと考えたとき、Azureの「Traffic Manager」機能を使い、短時間で複数のデータセンターへルーティングできたという。

セッション後半はオープンソース化の話題。マイクロソフトは2014年11月、.NETの一部オープンソース化を発表している。Scott氏は大きな変化に踏み切った理由として「さまざまなデバイスが存在する現状の中で他サービスとのハイブリッドな連動を考えなければならないこと」「すべての人がWindowsを使っているわけではないという前提で、ほかのサービスと同じ条件で戦う必要があること」の2点があると分析している。

オープンソース化によって次期バージョンである.NET 2015は、LinuxやMacでの実行が可能となる。また.NETのすべてをコンポーネント化し、ユーザが必要な部分をパッケージ・マネージャ「Nuget」によって取捨選択できるといった、カスタマブルなフレームワークにしていくとのことだ。

また開発環境の選択をより柔軟にすることも大きな目的の1つとしており、.NETのコンパイラプラットフォーム「Roslym」のオープンソース化や、VimやEmacsなどでVisuai Studioのインテリセンスを利用できる「OmniSharp」へのサポートなどが紹介された。

Enterprise向けクラウドの選択肢

▲日本マイクロソフト㈱　澤円氏

エンタープライズでのクラウド利用について、日本マイクロソフト㈱テクノロジセンターの長である澤円氏がセッションを行った。

セッションの中で一番に強調されたのはマイクロソフトのセキュリティへの取り組みである。氏によると、マイクロソフトは世界で2番めにサーバアタックを多く受けている組織だという（1番はアメリカ国防総省）。不名誉なことではあるが、そのことによってセキュリティに関するさまざまなノウハウの蓄えがある。米国のマイクロソフト本社にあるサイバークライムセンタではそのノウハウを利用しつつ、世界中の脅威の分析、情報の共有をしている。2012年には「Nitol」というボットネットが、中国の市販PCにプリインストールされたプログラムであることを突き止めたという。

セッションの後半で紹介されたのは、Azureの新サービス「ExpressRoute」。これは、Azureデータセンターとユーザのインフラとの間でプライベートな接続を作成する閉域網接続サービス。1月15日より、IIJやエクイニクス・ジャパンなどがパートナーとなってサービス提供を行っている。

CONTACT Japan Azure User Group
URL http://r.jazug.jp

Hardware

ティントリジャパン、スマートストレージの新製品を販売開始

ティントリジャパン合同会社は、データ圧縮と暗号化をサポートし、2倍以上の容量拡張とデータの堅牢性を実現したストレージ製品「Tintri VMstore T800シリーズ」および最新オペレーティング システム「Tintri OS 3.1」を1月15日より販売開始した。

「Tintri VMstore T800シリーズ」は、搭載するハードディスク上で機能するデータ圧縮をサポートし、圧縮前と比較して論理実効容量（ユーザが実際にデータを保存できるストレージ容量）を2倍以上に拡張するとともに、容量当たりの単価を50%に低減した。

ラインナップとしてはT820、T850、T880の3種類のモデルを用意。ハイエンドモデルとなるT880は、論理実効容量で100TBを実現し、従来のT650と比較して同じ筐体サイズでありながら3倍のデータを保存できる。これにより、42Uサイズの1ラックに最大で1PBの容量、ならびに140万IOPSの性能まで拡張でき、3万5千の仮想マシンを最小限の設置面積で稼働させることができる。

また、同時に発表した最新のオペレーティングシステム「Tintri OS 3.1」ではデータの災害対策の自動化や暗号化などを新たにサポートし、エンタープライズデータのセキュリティとビジネス継続性を強化した。

▲Tintri VMstore T800シリーズ

▼Tintri VMstore製品詳細（価格は1ノード、税別）

製品名	参考価格	実効容量	論理実効容量	サポートVM
T820	14,800,000円	10.5TB	23TB	750
T850	29,800,000円	30TB	66TB	2,000
T880	50,600,000円	45TB	100TB	3,500

CONTACT ティントリジャパン合同会社
URL http://tintri.co.jp

Software

グレープシティ、「ComponentOne Studio」シリーズの新バージョン「2014J v3」をリリース

グレープシティ㈱は、業務アプリケーション開発に便利なコンポーネントを数多く収録したスイート製品「ComponentOne Studio」シリーズの新バージョン「2014J v3」を1月15日にリリースした。

「ComponentOne Studio」はWindowsフォーム、ASP.NET、WPF、Silverlight、Windowsストアのアプリケーションを開発できるコンポーネントを数多く収録した製品。「2014J v3」の代表的な新機能として、Microsoft Excelのピボットテーブルおよびピボットグラフに類似したオンライン分析処理コンポーネント「OLAP」を、Windowsフォーム／WPF／Silverlightアプリケーション開発用のエディションに追加した。

OLAPコンポーネントは、クロス集計（多次元データ分析）に最適化したデータグリッド。コントロールにデータバインドするだけで、Excelのようなピボットテーブルをアプリケーション上で簡単に作成できる。ピボットテーブルを使えば、フィールドをドラッグ＆ドロップするだけであらゆる角度からのデータ分析を一瞬で行え、データベース言語の知識がないエンドユーザでも容易に操作できる。たとえば、販売部門向けのアプリケーションの場合、1つの売上データを「年代別」「地域別」「価格帯別」「会計年度別」といった複数の側面から集計することができる。さらに、アプリケーション上では横棒、縦棒、折れ線といった5種以上のチャートが自動的に生成されるので、集計したデータの視覚化も容易。OLAPコンポーネントを使用すれば、日々更新される大量のデータをExcelにエクスポートする手間なく、アプリケーション上で必要なものだけを抽出、集計し、レポートとして可視化できる。そのため、BIなど分析に特化したシステムを簡単に開発できる。

ComponentOne Studioは定額制のサブスクリプション方式で販売しており、初回費用は「ComponentOne Studio Enterprise」の1ユーザライセンスで162,000円(税込)、1年単位の更新費用は初回費用の40%である64,800円(税込)となっている。

そのほか、同製品の新機能は以下の専用サイトで確認できる。

○http://c1.grapecity.com/SuperPages/whatsnew/

CONTACT グレープシティ㈱
URL http://www.grapecity.com

Letters from Readers

次は○○型？　プログラミング言語の移り変わり

手続き型、オブジェクト指向、関数型……プログラミング言語の数は多く（Wikipediaの記事「プログラミング言語の一覧」によると271!）、今も増え続けています。最近は関数型言語「Scala」が人気ですが、次はどんな言語が躍進するのでしょうか。Rubyの開発者まつもとひろゆき氏は現在、新言語「Streem」を開発中で、これはストリーム指向の並行スクリプト言語になる予定だそうです。

2015年1月号について、たくさんのお便りをありがとうございました！

第1特集　「Vim使い」事始め

初心者向け、「Vim」の入門講座です。導入・カスタマイズを扱った基本編、プログラマ・インフラエンジニア・文章作成ごとのTips・対策を紹介した実用編、そしてVimの魅力について語る2つのコラムを掲載。40ページにわたってVimを特集しました。

最近使っていなかったので、参考になりました。

埼玉県／お手軽大好きさん

viは必須なので、役に立ちました。まだまだ知らないことも多いですが。

東京都／psiさん

普段からVim（またはvi）をよく利用しているが、それでも参考になる記事があった。

京都府／芦田さん

長年のVimユーザですが、特集で初めて知った知識も多くてとても役に立つ内容でした。とくに、普通はあまり重視されていない日本語入力を含めた文章作成についてもページが割かれていたのはうれしかったです。

埼玉県／犬棟梁さん

敷居が高いけど、ないと困る。

愛知県／kmさん

Vimは私も一番利用するエディタなので参考になります。

千葉県／Tayuさん

ちょうどVimの勉強をしていたのでタイムリーでした。

埼玉県／櫃山さん

入門記事ながら、すでにVimを使っているという読者からも参考になったという声が多く寄せられました。Vimを深く知る執筆陣の話はどのレベルのユーザにとっても有意義なものになったことでしょう。

第2特集　ソフトウェア開発の未来

厳しいスケジュールや多重下請けなど、ソフトウェアの受託開発にはブラックなイメージがつきまといがちです。本特集では受託開発というビジネスモデルの問題点を洗い出しながら、「価値創造契約」「一人情シス」などのビジネスモデルや働き方を紹介しました。そして、これからのシステムインテグレーションがどうあるべきかを論じました。

興味深かった。SIの今後についてもっと掘り下げてほしい。

埼玉県／fkdaさん

近年の技術者にとってとても大事なことだと思います。

東京都／tekitoizmさん

なんだか憂鬱に……。

大阪府／きよさん

「受託SIはたいへん」という見方もありますが、己のスキルアップには最適な道でもあると思います。仕事の環境面で今後の流れに期待したいです。

愛知県／NGC2068さん

自分も一人情シスなので、湯本さんの記事には勇気づけられました。もし自分が倒れてしまったら、ということを考えるとあまりマニアックな言語は使えないため、Excel・Access VBAくらいしか選択できず、最新言語の技術にまったくキャッチアップできないことや、ソース管理が難しいことに悩んでいます。みなさんどうされてるんでしょうかね?

愛知県／都築さん

いつも自分の感じていることがまとめてあり非常に共感を覚えました。一過性の表面的なコストだけを見ているとユーザ企業も技術者も誰も幸せになれないで

すよね。だけど、誰しもが納得できる解決策がないのがつらいところです。

千葉県／今井さん

考えさせられる記事で良かった。もはや、何でもかんでも「自社専用システム」ではなく、OSSの組み合わせで済ませられる時代が到来したと考えている。そのとき、自分ができる仕事について考えると深い内容の記事だと思った。今後もこのような開発の将来を俯瞰するような記事を期待したい。

神奈川県／東風谷さん

なかなか切れ味鋭かったです。

宮城県／オミオさん

実際にSIに携わる読者から多くの声が寄せられました。普段から業界に対して危機感を持っている方が多いようです。特集で紹介した新しいビジネスモデルが、良い流れの始まりになればいいですね。

一般記事　Jamesのセキュリティレッスン【最終回】

パケットキャプチャ「Wireshark」に導入された「pcap-ng」という新しいファイル形式を紹介する短期連載の最終回です。実際にpcap-ngファイルをバイナリエディタで読み、既存のファイル形式pcapとの違いがどこにあるかを解説しました。

必要になったら再度読むことにしようと思いました。

大阪府／平野さん

ただ今、Wireshark勉強中。

北海道／中谷さん

パケットキャプチャは自分もするので、とても参考になった。

静岡県／ももんがさん

バイナリエディタ、という力技を久々に見ました。

奈良県／組み込み以外知らないさん

キャプチャ情報の種類などたいへんわかりやすかったです。

山口県／A758さん

Wiresharkと組み合わせるようなソフトウェアの紹介もぜひ。

東京都／下平さん

パケットキャプチャは、トラフィックの確認やネットワーク障害発生時の原因調査などに利用できる技術です。本連載は、社内のネットワーク管理に携わる人にとっては実践的な内容であり、そうでない人には具体的なパケットの仕様を見ながらネットワークを学べる実習的な内容であったかと思います。

年末年始スペシャル　ITエンジニア出世双六

「ひみつのLinux通信」でお馴染みの、くつなりょうすけ氏制作の双六。ITエンジニアにとってはあるあるな風景も多いのではないでしょうか。

エンジニアとして将来のキャリアをどうしていくべきか考えさせられました。

東京都／n0tsさん

shufコマンド、ないです。客先の受付嬢されている方はきれいですよね。

東京都／sempreffさん

「俺の名前を言ってみろ!!」「Vagrantのありがたみがドッカーに行ってしまった」が良かったです。

東京都／山下さん

反響が大きかった、お正月の特別企画です。細かいネタの数々が、読者のツボに入ったようです。

連載

「軽酔対談　かまぶの部屋」について、技術書の品揃えが良いのでよくジュンク堂を利用しています。単に商品として並べるだけではなく書店員自らイベントに出展しているのは知らなかったです。商品に関連する接点を増やして、売れそうな商品のニオイを嗅ぎ付ける努力は大切ですね。

岩手県／隼さん

IT業界からのゲストを招く本連載ですが、1月号では珍しく書店業界で働く長田さんからお話を伺いました。コンピュータ書フロアにいるお客さんと実際に話して意見の交換をするというお話は、技術書を作る側の我々にとっても非常にためになるものでした。

1月号のプレゼント当選者は、次の皆さまです

①アークタッチ Bluetooth マウス
大阪府　與喜好様

②裸族の一戸建て SATA6G
埼玉県　石澤景子様

③弥生会計 15 スタンダード
福岡県　石内理絵様
大阪府　澤下夏実様
埼玉県　小堀大介様

④Linuxによる並行プログラミング入門
東京都　向後哲郎様
東京都　中村佳子様

⑤新装改訂版Linuxのブートプロセスをみる
兵庫県　村上真也様
神奈川県　齋藤仁史様

⑤Gitバージョン管理
東京都　山添淳一様
埼玉県　田中正人様

⑥15時間でわかるJava集中講座
滋賀県　田中良明様
東京都　香取謙治様

※「弥生会計 15 スタンダード」につきまして、1月号読者プレゼントページでは1名様への提供となっておりましたが、3名様へ変更となりました。

次号予告

Software Design
April 2015

2015年4月号
定価（本体1,300円+税）
192ページ

3月18日発売

［第1特集］ソフトウェア開発、運用管理、クラウド環境

トラブルシューティングの極意
――達人に訊く問題解決のヒント

思わぬミスでトラブル発生！　安定稼働しているシステムほど要注意です。ソフトウェア開発の鉄人、インフラエンジニアの巨人達、クラウドエンジニアのフロンティア達から問題解決のヒントを学びましょう。

［第2特集］

DNSの教科書
――ネットワークを支える本物のインフラを学ぶ

本特集では、DNSの仕組みを理解し、サーバを構築するまでを解説します。いざというときのエンジニアの底力を身につけよう!

［特別付録］3分間ネットワーク基礎講座

※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2月号　連載「Hosting Department」
●p.H-1、右段、22行目
［誤］SSL 3.0以前が影響を受ける「Bash」の脆弱性や，　［正］SSL 3.0以前が影響を受ける「SSL」の脆弱性や，

休載のお知らせ

「Androidエンジニアからの招待状」（第56回）は都合によりお休みさせていただきます。

SD Staff Room

●密閉された空間の遊具群が未知のウィルスを醸す。ノロわれた胃腸は一切の水分を受けつけずスルーする。子供は1日で治るも、大人はもがき苦しみ厠（かわや）に向かう。発熱でループする悪夢と膝関節の疼痛をくぐり抜け、1週間床に伏したその先にあるのは仕事の山か。次回「年度末」。2月は逃げるでお茶を濁す。（本）

●今年買って驚いたもの。靴の消臭剤「グランズレメディ」。時々中敷きを変えたりしていたが、評判を見て使ってみたら……5kmほど歩いた後で汗かいてるのに靴中は完全に無臭。／個人輸入扱いの「アルギニンカプセル」。朝起きた時の爽快感。ダル重だった身体や頭がスッキリ。すごい。（効能には個人差があるかも幕）

●「ぼっちテント」をノリで購入。2人の子どもが一緒の部屋で勉強すると、お互いが気になり、はかどらなくなりがちでした。でも、なんとこのぼっちテントに1人が入ればそれぞれ集中できるようになったんです！　勉強時間が短くなったと子どもたちにも大好評です!※感想には個人差があります w（キ）

●2月号の記事「UNIX用語読み方指南」がなかなか好評だった様子。弊誌ではOSSの開発者によく原稿を書いてもらいますが、IT用語の読み方というのは、開発者側からはなかなか出てこない企画かもしれません。今回は、編集者がうまくユーザ目線で企画できたのが良かったのかなあ、と思いました。（よし）

●社会人になってから映画館に行く機会が多くなりました。2014年のベストはジャン=ピエール・ジュネ監督の「天才スピヴェット」。単純なストーリーながら、ハートフルときどきブラックな展開で2時間いっぱい目が離せませんでした！　ところで映画館のチケット代もう少し安くなりませんかね。（な）

●旦那の友人が東京ディズニーシーで結婚式をするということで、お祝いがてら見物しに行きました。チャペルからゴンドラ乗り場まで園内を歩いていくので、ちょっとしたショーを見ているみたい。居合わせた人たちからも祝福され、幸せ感満載。お天気もよかったし、いい結婚式をみることができました。（ま）

ご案内
編集部へのニュースリリース、各種案内などがございましたら、下記宛までお送りくださいますようお願いいたします。

Software Design 編集部
ニュース担当係

［FAX］
03-3513-6173
※ FAX番号は変更される可能性もありますので、ご確認のうえご利用ください。

［E-mail］
sd@gihyo.co.jp

Software Design
2015年3月号

発行日
2015年3月18日

●発行人
片岡　巌

●編集人
池本公平

●編集
金田富士男
菊池　猛
吉岡高弘
中田瑛人

●編集アシスタント
松本涼子

●広告
中島亮太
北川香織

●発行所
㈱技術評論社
編集部
TEL：03-3513-6170
販売促進部
TEL：03-3513-6150
広告企画部
TEL：03-3513-6165

●印刷
図書印刷㈱

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

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