special feature 1 Java再発見

special feature 2 PCサーバを見極める

Software Design

2014 10

2014年10月18日発行
毎月1回18日発行
通巻354号
（発刊288号）
1985年7月1日
第三種郵便物認可
ISSN 0916-6297
定価 本体 1,220円 +税

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

» special feature 01

今ふたたびのJava

» 言語仕様 » 開発環境 » デバッグ機能

あなたはどこまで使いこなせてる？

» special feature 02

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]

x86サーバハードウェア入門

» 特別企画

オーケストレーションツールSerf・Consul入門[Consul編]

» 新連載

帰宅が5分早くなり、休出もなくなる!?

Hinemosで学ぶジョブ管理超入門

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

第69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。

OpenSSL とその派生プロジェクト

セキュリティ問題に揺れる「OpenSSL」

OpenSSLは、暗号通信のためのSSLおよびTLSプロトコルを実装したオープンソースソフトウェアです。さまざまなOSやプログラミング言語に対応しており、ソフトウェアに組み込むライブラリとして利用することができるため、幅広い分野で標準的に利用されています。

2014年に入って、このOpenSSLに2つの重大な脆弱性が報告されました。1つはTLSのHeartbeat拡張機能に混入した「Heartbleed」と呼ばれる脆弱性です。Heartbeat拡張とは、ネットワークリソース間で暗号通信のセッション保持時間を延ばすためのしくみで、OpenSSLでは2012年3月にリリースされた1.0.1よりサポートされました。しかしこの実装にメモリの取り扱いに関する深刻なバグがあり、不適切なリクエストを送ることで、本来は参照することができないサーバのメモリの内容を最大で64KBまで読み取ることができてしまうことが判明しました。それがHeartbleedです。この脆弱性を利用することで攻撃者は痕跡を残すことなくメモリの内容を読むことができ、場合によっては秘密鍵などの重大な機密情報が盗まれ、暗号化そのものが無力化する恐れもあります。

Heartbleedの熱が冷めない中、「CCS Injection Vulnerability」と呼ばれる新たな脆弱性が報告されました。これはOpenSSLのハンドシェイク中に不適切な順序でChange CipherSpecメッセージを挿入することによって、強度の弱い暗号通信へ強制変更することができるというものです。この結果、適切な強度の暗号化が行われずに、通信内容や認証情報などの情報を読み取られたり改ざんされる恐れがあります。

派生プロジェクトの登場

OpenSSLのシェアは極めて高いことから、これらの脆弱性（とくにHeartbleed）の発見はIT業界に大きな衝撃を与えました。HTTPSサイトのうちの17.5%が脆弱性を抱えたままHeartbeat拡張を有効にしていたという報告もあり、その影響力がWebの安全性を根底から覆しかねないものであることがわかります。そのため、この衝撃的な発表を受けて、OpenSSLをフォークした新しいプロジェクトも発足しました。

LibreSSL

「LibreSSL」はThe OpenBSD Projectが立ち上げたプロジェクトで、OpenBSDで利用されているOpenSSLライブラリを置き換えることを目的としています。同プロジェクトでは、OpenSSLの問題として古いシステムをサポートするためにソースコードが肥大化・複雑化している点や、mallocをはじめとするカスタムメモリコールにさまざまな問題を抱えている点などを挙げています。またプロジェクトの管理方法にも問題を抱えており、古いバグが解決されないまま放置されているとも指摘しています。

そこでLibreSSLプロジェクトでは、OpenSSLのコード削減、メモリ管理の標準ライブラリへの置き換えやFIPS規格サポートの廃止、古いバグの修正などを行い、よりシンプルで安全性の高いセキュリティ基盤を構築するとのことです。

BoringSSL

「BoringSSL」はGoogleが立ち上げたプロジェクトで、その目的はおもにGoogle内部での利用を想定したコードベースの構築にあります。Googleでは以前からOpenSSLのコードを検証し、独自に多数のパッチを適用して自社のプロダクトに使用してきましたが、その数が増えるにしたがってパッチを充てる労力だけでも大きな負担になっていたとのことです。そこでいったん本家のOpenSSLから離脱して独自のコードベースを築いたうえで、今後OpenSSLに加えられる変更を取り込んでいく、というスタイルを選んだ結果がBoringSSLというわけです。BoringSSLでは、OpenSSLだけでなくLibreSSLに対する変更も取り込んで行く方針を明らかにしています。

こうした新しい活動に注目が集まる一方で、依然として脆弱性のあるOpenSSLを使い続けているWebサイトも多数存在するとのことで、事態が完全に収束するにはもうしばらく時間がかかりそうです。SD

OpenSSL
http://www.openssl.org/

OSとネットワーク、
IT環境を支えるエンジニアの総合誌

Software Design

毎月18日発売

PDF電子版
Gihyo Digital
Publishingにて
販売開始

年間定期購読と電子版販売のご案内

1年購読(12回)

14,880円(税込み、送料無料)1冊あたり1,240円(6%割引)

PDF電子版の購入については

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

をご覧ください。
PDF電子版の年間定期購読も受け付けております。

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

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

Fujisan.co.jpからのお申し込み方法

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

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

第70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。

CoreOS

Docker 専用 OS「CoreOS」

クラウドサービスの普及にともなって、アプリケーションの実行環境を丸ごとコンテナ化して管理することができるコンテナ型仮想化ツールの「Docker注」が人気を集めています。そんななかで、クラウドの専門家として知られるAlex Polviが設立したCoreOS社が、Docker専用の新しいOS「CoreOS」をリリースしました。

CoreOSはLinuxカーネルをベースに作成されたディストリビューションの1つですが、クラウド上で柔軟にスケールし、さらにインフラの構築や運用のプロセスそのものを簡略化することに主眼を置いて設計されており、おもに次のような特徴を持っています。

- 不要な部分を削ぎ落とした小さなコア
- アプリケーションコンテナとしてDockerを使用
- 安全なアップデート方式
- クラスタリング機能と分散システムツールを標準で搭載

コンテナに限定されたアプリケーション実行環境

CoreOSの最大の特徴は、アプリケーションの実行環境をコンテナだけに特化しているという点です。アプリケーションを実行したい場合には、通常のOSのように直接インストールして実行するのではなく、実行環境ごとDockerイメージとしてコンテナ化しておいて、それをDockerエンジンの上で動作させます。Docker専用OSと言われる理由はここにあります。

OSのコア機能は最低限必要なパッケージに限定して軽量化されているため、オーバーヘッドが小さく、少ないリソースで動作させることができます。前述のとおり、この部分にはアプリケーションを追加できないことから、管理の手間やセキュリティリスクを低減できるというメリットもあります。

OS本体が正／副2つのブート可能なファイルシステムに格納されているというのもCoreOSの大きな特徴です。OSをアップデートする際には、まず副のほうのファイルシステムのみ更新します。それが正常に完了したら、更新したほうのファイルシステムを新たな正として再起動します。もしこのときに問題が発生しても、片方は元のシステムのままなので簡単に復旧することができるというわけです。正常に起動できたら、正のほうも更新することでアップデートが完了します。

CoreOS本体にアプリケーションを追加するなどのカスタマイズを行いたい場合には、CoreOS SDKというツールを利用します。CoreOS SDKもオープンソースで提供されており、これを使えばそれぞれの用途や環境に合わせたオリジナルのCoreOSを作成することが可能です。

従来のOSでは、OSのコア部分の管理とアプリケーション実行環境の管理はセットで考える必要がありました。CoreOSの場合は両者が明確に分離されているため、それぞれを個別に管理できる点が大きな強みと言えます。

標準でクラスタリング機能を搭載

CoreOSのもう1つの特徴は標準でクラスタリング機能を持っているということです。CoreOSのクラスタリング機能は「etcd」と呼ばれるツールによって実現されています。etcdはシステムの環境情報を共有・管理するKVS（Key-Value Store）のようなシステムです。分散環境における対故障性に優れており、CoreOSではetcdを使ってコンテナの環境情報を記録することによりクラスタを構築します。クラスタ上ではetcdはマスタとフォロワの2つの役割に分かれており、障害が発生した場合に自動でマスタを入れ替えたり、追加されたピアを簡単に検出したりといった機能を備えています。

クラスタへのデプロイやクラスタ内のコンテナの起動・停止などといった操作には、「fleet」と呼ばれる分散システム管理ツールが用意されています。fleetではDockerと組み合わせて利用することで指定した数のApacheを立ち上げたり、サービスを自動でフェイルオーバーさせたりといった設定を行うこともできるようになっています。

CoreOSは用途が限定された先鋭的なOSですが、Dockerのメリットを最大限に活かしたい場合には極めて有効な選択肢と言えるでしょう。SD

CoreOS
https://coreos.com/

注）Dockerについては本連載第67回（2014年7月号）で解説しています。

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

Contents

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

Contents

Software Design Oct. 2014

2

技術評論社の本が電子版で読める！

電子版の最新リストは

Gihyo Digital Publishingの

サイトにて確認できます。

https://gihyo.jp/dp

GIHYO Digital Publishing

amazon.co.jp

Google play

楽天 Rakuten KC　kobo

法人などまとめてのご購入については
別途お問い合わせください。

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

Contents

Software Design Oct. 2014

Column

Development

OS/Network

Logo Design ロゴデザイン	デザイン集合ゼブラ+坂井 哲也
Cover Design 表紙デザイン	Re:D
Cover Photo 表紙写真	Julia Christe /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

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。

BOOKS INFOMATION

データマイニングの**一番わかりやすい**入門書です！

\手を動かしながら学ぶ/

ビジネスに活かす データマイニング

尾崎隆　著

こんな方におすすめ

- データマイニングの入門者
- 統計分析ツール「R」を使ってみたい方

CONTENTS

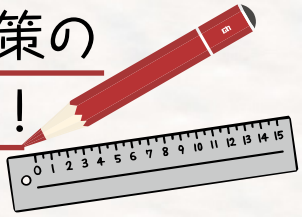
ISBN978-4-7741-6674-2
A5判／224ページ　定価（本体1980円+税）

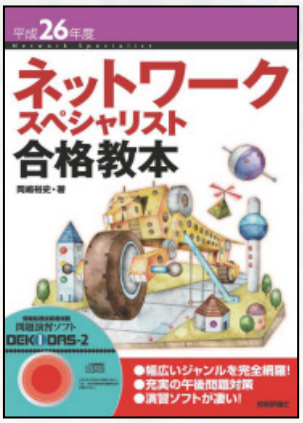
人気ブログ「銀座で働くデータサイエンティストのブログ」を運営する現役データサイエンティスト（Data Scientist）である著者が、Rを使ったデータマイニングの基礎から最新の手法まで、ビジネス現場での具体例を交えながらやさしく解説します。当社のサイトからダウンロードできるサンプルデータを用いて実際に手を動かしながら学習していく方式なので、データもRも「使える」力を確実に身に付けることができます。

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

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！





岡嶋裕史　著
A5判／624ページ
定価（本体2980円+税）
ISBN978-4-7741-6354-3

エディフィストラーニング株式会社　著
B5判／384ページ
定価（本体2980円+税）
ISBN978-4-7741-6355-0

岡嶋裕史　著
A5判／656ページ
定価（本体2880円+税）
ISBN978-4-7741-6099-3

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）
ISBN978-4-7741-6570-7

濱本常義ほか　著
A5判／592ページ
定価（本体3200円+税）
ISBN978-4-7741-6178-5

左門至峰、平田賀一　著
A5判／328ページ
定価（本体2280円+税）
ISBN978-4-7741-6389-5

大滝みや子、岡嶋裕史　著
A5判／736ページ
定価（本体2980円+税）
ISBN978-4-7741-6111-2

加藤昭、芦屋広太ほか　著
B5判／464ページ
定価（本体1680円+税）
ISBN978-4-7741-6556-1

金子則彦　著
A5判／688ページ
定価（本体3300円+税）
ISBN978-4-7741-6177-8

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

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

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

DIGITAL GADGET

Volume 190

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

IDEA 2014 にみるガジェットプロダクト

インダストリアルデザインの世界的アワード

IDEA（International Design Excellence Awards）は、米国工業デザイナー協会による、優れたインダストリアルデザインに与えられる賞です。経済活動や生活の質にかかわるインダストリアルデザインの価値をより広く伝えることを目的に、1980年に設立されたデザイン賞です。

金賞、銀賞、銅賞、学生奨励賞が協会に所属する24名の現役デザイナーの審査員によって選出されます。今年は数多くの製品の中から、全23部門、176作品が受賞しました。サイトには2001年以来の受賞作が紹介されており、時代の変遷を知るのにも役立ちます。

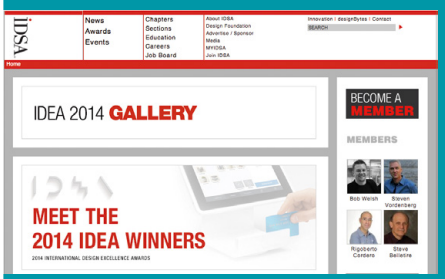
たとえば、2001年の金賞には、クラシックカーのような風貌の車「Chrysler PT Cruiser」や、Virgin Atlantic航空の座席シートなどが受賞しています。さらに最近では、製品そのものとして形があるわけではない、デジタルデザインに関しても受賞する製品が増えてきました。

IDEA 2014公式サイト
http://www.idsa.org/awards

23部門の受賞作一覧
http://www.idsa.org/idea-2014-gallery

賞は一部特殊な医療機器などを除き、米国で一般に販売されている製品から選出され、インダストリアルデザインの価値を一般の消費者とビジネス界にどれだけ伝えられるかが観点となっています。単に製品としての材質、形状や色あいだけではなく、その製品がもたらす体験的な価値も重視されているようです。

カテゴリは、自動車や交通、製造業（おもに道具）、デザイン戦略、環境、キッチン、アウトドア、リサーチ、バスルーム、コミュニケーションツール、デジタルデザイン、庭、医療と科学、パッケージ、サービスデザイン、スポーツ、



A

B


C


D


E


F


G

[A]IDEA 2014公式サイト
[B]Teslaの電気自動車用充電ステーション
[C]消防士用の新型酸素マスク「DELTAIR」
[D]Hamilton Medicalの可搬用人工呼吸器
[E]赤ちゃん向け救急セット「Healthy Baby」
[F]3Dプリンタで作られた外骨格スーツ
[G]雪山登山用のアタッチメント「Maglon」

※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。

子供向け、コンピュータ関連、エンターテインメント、家具、オフィス、アクセサリ、社会的影響のあるデザイン、学生によるデザインに分かれます。複数のカテゴリにノミネートされている製品もあります。たとえば、社会的影響のあるデザインとしては、赤ちゃん用の救急セット、ケニアの子供たちに衛生的な水を提供するプロジェクト、3Dプリンタで作られた医療器具などが受賞しています。

また、リサーチ部門としては、消防士が息をしやすくなるよう工夫された酸素マスク、医療機器の操作改善、可搬式医療機器の検討、末期患者のための病院のデザインが受賞。デザイン戦略部門では、電気自動車の充電ステーション戦略、学校給食のデザイン、企業のブランド戦略、Eコマース戦略が選ばれるなど、多岐に渡っています。

今年の受賞作あれこれ

今年のIDEA賞は、部門にかかわらずスマートフォンアプリが多く受賞していました。

Making of Making Powered by NIKE MSI

http://nikemakers.com/

素材選定を手助けするためのデザイナー向けアプリ。“乾きやすい”など状況に適した素材を選択できる

Spark Camera

http://www.sparkcamera.com/

動画クリップ録画アプリ。動画クリップをつなげて活用できる、動画撮影の概念を覆すツール

Yahoo Weather mobile apps for iPhone

https://mobile.yahoo.com/weather/

美しいお天気アプリ。誰もが使うが、革新のなかった分野に美しさと新体験をもたらしたツール

Yahoo News Digest

https://mobile.yahoo.com/newsdigest/

ニュースアプリ。今日のニュースを見終わったという達成感がある、画面を斜めに切り取った独特のデザインのアプリ

また、スマートフォンを活用した周辺製品も多く受賞していました。

The LeapFrog Creativity Camera Protective Case & App

http://www.leapfrog.com/en-us/store/p/creativity-camera-protective-case-app/_/A-prod19234

スマートフォンにかぶせて子供用カメラに利用するケース。約20ドル。専用アプリあり

WeMo Insight Switch

http://www.belkin.com/us/support-product?pid=01t80000003JS3FAAW

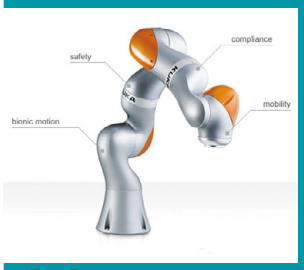
家庭用電源コンセントをネット経由でON/OFFコントロールできる電源端子に。約60ドル

今後もこのように、ネットの世界を具現化したアプリと現実世界のモノをつなぐサービス、すでに存在するさまざまなモノをアプリによってより便利に拡張していくサービスが浸透していくと思われます。

インダストリアルデザインの広がり

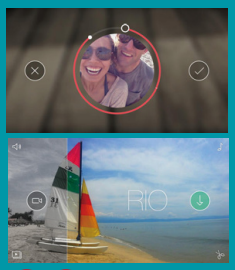
IoT（Internet of Things；モノのインターネット）を始め、今まで単独で存在していたさまざまな家電機器や、道具、家具、製品といったものがネットの世界とつながってきました。それに対し、ネット上のサービスも、現実世界といかに連携したサービスであるかが重視されるようになってきました。

また、単に必要だから購入される製品ばかりではなく、娯楽や楽しみのためのモノも増え、その一方で最先端の

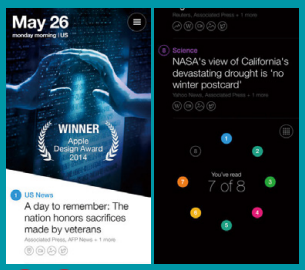


H

I



J



K

L

M

[H]軽量産業用ロボット「LBR iiwa Robot」
[I]素材選定アプリ「Making of Making」
[J]IDEOが開発した動画クリップ作成アプリ
[K]米国Yahoo!の美しいニュースアプリ
[L]子供用スマートフォンケース
[M]スマートフォンでコントロール可能な電源コンセント

テクノロジでなければ解決しない道具や医療機器なども存在します。ユーザインターフェースデザインやサービスデザインも包括しながら、インダストリアルデザインの範囲や意味も大きく広がってきています。

IDEAのサービスデザイン部門では、薬の飲み方に関する新しい提案やパッケージデザインが考えられたり、スポーツ部門でも単体のスポーツグッズだけではなく、スポーツで使う通信機器や、危険回避のためのデジタルツールなど、モノ単体だけではないサービスが増加しています。

マーケティングの専門家、セオドア・レビット博士の書籍で、「ドリルを買った人が欲しかったのはドリルではなく穴である」という言葉が紹介されています。この言葉はさまざまな製品に当てはめることができ、その製品にとっての「穴」が何なのかを考えることで、価値ある体験が提供できるのだと考えます。

また、高性能で安価な3Dプリンタの登場で、試行錯誤のスピードが速くなったと言われていますが、人間が5本指の手で扱う限り、形や操作などに関する知見は、急に変化するものでもありません。モノとして触れる道具などのデザインでは、プログラム次第で何でも作れるデジタルなデザインと異なり、生産可能な形状や生産コスト、材質、質感、耐久性、パッケージング、流通、販売、サポートなど、デジタルにはないさまざまな要素を持ちます。Webデザインなどのデジタルデザインも、本のデザインや印刷、フォント、色使いなど過去の遺産が活かされています。

インダストリアルデザインが持つ、バランスや質感、道具に対する習熟度、動作スピードの大切さといった観点が、デジタルツールにもおおいに活かせるのではないかと考えています。SD

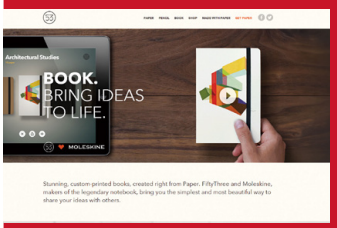
GADGET 1

BOOK by FiftyThree

https://www.fiftythree.com/book

デジタルノートをリアルなノートに

BOOK by FiftyThreeは、手書き風の仕上がりが人気のiPad用デジタルペイントツールで描いた画像データをアップロードすると、画像を表紙にプリントアウトした現物のMoleskineノートを手に入れることができる安価なサービスです。市販のMoleskine風の黒い表紙か、カラフルな印刷でくるんだカスタムカバーか、好きなほうを選べます。デジタル印刷機HP Indigoで折りたたみ式の15ページ分が印刷されます。絵心のある人はプレゼントにも良さそうです。



GADGET 2

Square Stand

https://squareup.com/stand

カード決済端末

Square Standはスマートフォンやタブレット端末で平易にクレジットカード決済できるサービスを提供する専用スタンドです。導入が簡単なタブレット端末で決済が可能だといっても、タブレットだけでは実際の店舗の現場では、狭いレジの領域で混乱を招きます。旧来のレジ端末風のスタンドを用意することで、信頼感や現場での操作感の向上をもたらした、コスト効果の高いアイデア商品と言えます。Square Standは実際にSquare社の社員食堂で使われ、効果が試されているそうです。

GADGET 3

Locale Office System

http://www.hermanmiller.com/products/workspaces/individual-workstations/locale.html

組み合わせ自在のオフィス家具

ハーマンミラー社製のオフィス家具シリーズLocale Office Systemは、グループやチームといった区分、または仕事の状況やプロジェクトの進行に応じて、オフィス家具の配置が変えられる組み合わせ式の家具シリーズです。活発に意見をかわしたい状況や、集中して仕事がしたい状況など、ほどよく開放的で、ほどよく閉鎖的な、微妙な距離感を考慮した家具の配置が可能になっています。また規模や数、形態に応じてさまざまなプランをチョイスでき、理想の仕事場、変化に富んだ職場を素早く構築することができます。

GADGET 4

iPin Laser Presenter

http://ipinlaser.com/ipin/

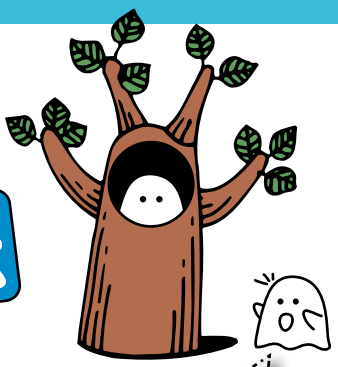
iPhone用レーザーポインタ

iPinはiPhoneのヘッドフォンコネクタに接続して利用するレーザーポインタです。ヘッドフォンコネクタからの給電と、専用アプリによるON/OFFで、プレゼンテーションなどに用いる赤色レーザーポインタとして利用できます。持ち歩きはiPhoneに刺したままで良いですし、バッテリーの心配もいりません。パッケージも、どんな働きをする商品なのか一目でわかり、とても好感度の高いものです。一方、iPinを刺したままでは通話ができませんので、注意が必要です。

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

17

結城浩の再発見の発想法



Template

Template——テンプレート

テンプレートとは

テンプレート(template)とは、**定型的な何かを作り出すための雛形(ひながた)**のことです。

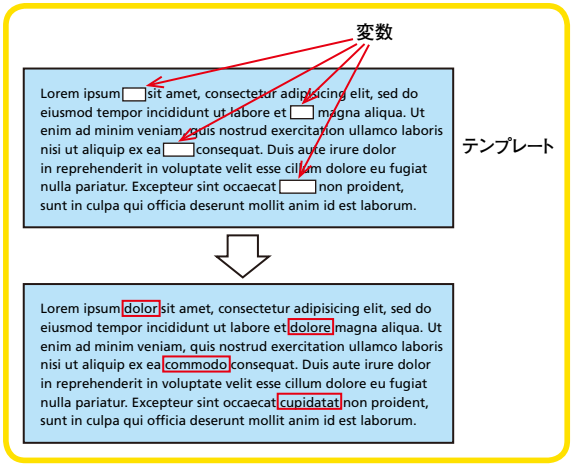
たとえば**製図用具のテンプレート**は、丸や四角などのよく使う図形の部分が穴になった薄い板です。それを使えば同じ大きさ／同じ形の図形を手軽に描くことができます。

Webサービスがユーザに送るメールでは、**定型メールを作るテンプレート**を使うことがあります。ユーザに送る文章のテンプレートをあらかじめ用意しておき、実際にメールを送るときに一部を置き換えるのです。メールに書かれる「日付」や「ユーザの氏名」などが**変数**(パラメータ)になっており、実際にメールを送るときにその変数を今日の日付や実際のユーザの氏名で置き換えます(**図1**)。こうすれば、人間がメールの文章をいちいち書かなくとも、ユーザごとの文章をコンピュータが自動生成できることになります。

Webサイトでは、**テンプレートエンジン**と呼ばれるソフトウェアを使うことがあります。たとえば、商品カタログを表示するWebサイトを考えてみましょう。表示される商品名の価格や写真がページごとに異なっていても、それらを表示する位置や大きさは同じでしょう。そのようなWebサイトでは、ページの共通な構造を定義し「商品名／価格／写真」を変数にしたテンプレートを作っておき、テンプレートエンジンにその変数を実際の中身と置き換えてもらえば、ページ作成が容易になります。1つのWebサイト内では、文章や画像のような「中身」がページごとに異なっていても、「構造」は同じになっていることが多いものですから、テンプレートエンジンが活躍するのです。

プログラミング言語のC++には**関数やクラスを作るテンプレート**の機能があります。この機能を使うと、関数やクラスを書くときに、型の部分を変数(パラメータ)にできるため、字面上は同じでも扱う型が異なるようなコードをうまく扱えます。変数の部分はコンパイル時に実際の型で置き換えられますので、同じようなコードを人間が書く必要がなくなりますし、それに起因するバグ

▼図1　テンプレートと変数



も少なくなります。

以上、いくつかの例で示したように、テンプレートは定型的な何かを作り出すための雛形であり、実際の値で置き換えられる変数(パラメータ、穴)を持っています。この変数によって、定型的ではあっても実際に必要なカスタマイズができるのです。

テンプレートの目的

テンプレートを使う目的は「労力を減らす」ことです。定型的なものを繰り返しゼロから作るのは労力の無駄ですから、テンプレートによって労力を減らそうというのです。ですから、テンプレートを作るには、まず**繰り返しを見つける**ことが大切になってきます。製図でも、定型メールでも、プログラミングでも、「同じことを繰り返している」と人間が気づいてはじめて「テンプレートを使おう」という発想に至るのです。

さらに、テンプレートでは、どの部分を変数にするかという設計判断が重要です。どの部分を変数にするかは、何度も起きる繰り返しの中にある**繰り返していないところを見つける**必要があるからです。

テンプレートには、2つの極端な設計方法があります。1つは、どんな場合にも対処できる万能テンプレートを1つ用意して変数を非常に多くする場合(最大限に汎用化)です。そしてもう1つは、個別のケースすべてにテンプレートをそれぞれ用意して変数を非常に少なくする場合(最大限に特殊化)です。現実のテンプレートは、この2つの間にあるわけですが、唯一の正解はありません。

日常生活とテンプレート

日常生活には定型作業の繰り返しがたくさんあり、テンプレートと見なすことができるものも少なくありません。会社や学校での毎日の活動も、あちこちに繰り返しがありますので、テンプレートを使って定型的な扱いが可能なことも多いでしょう。発生する繰り返しにうまく対処するためのテンプレートがあることは効率化に役立ちます。

たとえば、ファーストフードの店頭対応はテンプレートの一種です。「いらっしゃいませ」から始まって「顧客の注文の復唱」「料金の受け取り」「おつりの支払い」「商品の引き渡し」という一連の流れは、注文内容や料金を変数としたテンプレートと見なすことができるでしょう。テンプレートがうまく設計されていれば、店員の能力に大きく依存せずに、多くの顧客を効率よくさばくことができます。

テンプレートをうまく使えば労力を削減できますので、効率を重視する場面ではとくに役に立ちます。また、良いテンプレートは、活動のパターンやノウハウがその中に込められますので、個人の能力によるバラツキを減らすこともできます。その一方で、テンプレート化を進め過ぎると柔軟性を欠き、味気なくなることもあるでしょう。いわゆる「マニュアル化の弊害」ですね。

たとえば、病院で「問診票に自分の症状などを記入する」のはテンプレートを活用し、効率的に話を進めるために有効です。その一方で、医者が患者の悩みを聞く状況ではテンプレートの活用がそぐわないこともあるでしょう。たとえ、医者の側にはテンプレートがあったとしても、その存在を患者に感じさせるのが不適切な場合もあるということです。

テンプレートの存在を顧客にわざと感じさせることで「私はこの問題を扱い慣れていますよ」とアピールできる場合もありますが、逆に「機械的な対応であり、《私》を特別な存在として扱っていない」という印象を与える場合もあるでしょう。

◆ ◆ ◆

あなたの周りを見回して、テンプレートを探してみてください。それにはどんな変数がありますか。また、テンプレートで効率化できるようなものはありませんか。さらに、テンプレートの存在をわざと見せているサービス、見せないように工夫しているサービスはありますか。

ぜひ考えてみてください。SD

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。

軽酔対談

かまぷの部屋

第3献 ゲスト：シルネン ブヤンジャルガルさん

㈲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

シルネン ブヤンジャルガルさん

国籍はモンゴル。日本や米国に留学し、モノづくりエンジニアが社会を変えると信じて日本に本格渡航し、約10年。日本大手メーカー勤務を経て、現在はUSP研究所で海外におけるユニケージ啓蒙活動に尽力中。同時に母国であるモンゴルで高等専門学校設立事業に参画。シェルを始めとして「C」や「R」などのソフトウェアも得意。

英語も堪能。「本気で国をよくしたい」と語ります。

こんにちは。今回は日本を飛び出して、モンゴルでの対談です。

（鎌田）ブヤンさんは本当に日本語が上手ですよね。今回、モンゴルの地で対談できて、とてもうれしいです。そもそもブヤンさんが日本に興味を持ったきっかけは、なんでしょうか？

（ブヤン）まだ高校生のころモンゴルの先輩が日本に留学していました。ある日、日本で知り合ったケニア人の学友を連れて、僕の地元に遊びに来たんです。カルチャーショックでした。それがきっかけで自分も外国の友達をモンゴルに連れてきたいと憧れを持っていたんです。そして、日本にどうしたら留学できるか学校の先生に相談したところ僕も日本に留学できることがわかりました。僕は田舎育ちですが、モンゴルの都会であるウランバートルの大学に入学し、一年勉強した後、日本に留学することを決めました。日本へ渡り、日本語学校で日本語漬けになったあと、佐世保の工業高等専門学校に進学しました。日本語がある程度わかっていても、先生の授業は100%日本語ですので、ついていくのがたいへんで、最初の3ヵ月は苦労しました。方言もありましたから（笑）。

苦労されたのですね。そもそもモンゴルの日常生活で、日本の文化に触れたことはありましたか？

モンゴルでは、NHKの番組が普通に放送されています。その中で、日本の一日の生活のスタイルを紹介しているので親近感があります。一番大きな影響はNHKで放送していたロボコンの大会でした。その番組ではロボットがバスケットボールをしていて、本当におもしろかった。それが子供のころの憧れで、佐世保の高専時代では、もちろんロボコンに参加しました。夢が実現したんです。それと、ウランバートル市街では、日本からの観光客が多いので、日本語をよく耳にします。

モンゴルと、ほかの国とを比較して感じたことはありますか？

いろんな国に行ってみて、それぞれの国にはいろんな長所があると思いました。当たり前ですが地球というのは1つしかないじゃないですか。僕にとってはクラスメートのようなものだと思うんです。そして自分のクラスメートの特長を活かすべきだと思っています。モンゴルの特徴は人口が少ないですが、土地は広く大自然があります。モンゴル人は少ない資源で人生を豊かにする知恵があります。そしてもっと幸せを感じる国にしたいと思っています。あわただしくはしたくないです。そして世界の一員として何かをするべきだといつも思っています。

視野が広いですね。そのような考えを持った理由はなんですか。

僕がそう思うのは両親の影響が強いと思います。僕の家はおもしろいんです。父はトラックの仕事をしています。僕の子供のころは、まだ道路が整備されていなかった。車も普及していないので、たくさん人が家に出入りしていたんです。いろいろなところへ行き、多くの人と出会うのですが、父は食事を振る舞うのが趣味なんです。僕は当時、それが理解できなかった。「家の食べ物をみんなにあげちゃうの？」と質問しました。父は「人生は長いものだよ。人は




助け合うものなんだよ」と教えてくれました。子供のころは家の物が盗られたみたいで正直よくわかりませんでした。でも今は理解できます。父の行動とその言葉に感謝しています。

しっかりした生き方を持ったお父様だったんですね。モンゴルの方は家族愛が強いですね。

表には出さないけど、皆、家族はものすごく強いです。親の面倒を看るとか、兄弟は助け合うとか。親にお金がないときには、子供がお金をあげます。貸すということではないんです。家族は皆それぞれ自立していますが、心はつながっています。僕もいつかは家族を作りたい。でも今は仕事が忙しくてプライベートの時間が少ないんです。今、仕事以外で大事にしたいことは、結婚を前提におつき合いしている彼女を理解することかもしれない。彼女との時間をもっと作りたいんですね。これも大事な仕事かな(笑)。

彼女とはアメリカで知り合ったのですよね、アメリカの大学ではどんなことを学びましたか。

はい。アメリカで学んだことは、「世界は小さい。地球は大きくない」ですね。アメリカにはいろんな国の人が集まって、みんなが自己実現のためにお互いを尊重して、真剣に勉強しているのを目の当たりにするんです。その姿を見ること自体が最も勉強になりました。僕は電子・機械の専攻だったのですが、分野に限らずアメリカの先生の授業では「なぜですか？」という終わり方をしていました。考えさせる教育をしているんです。また、アメリカは大学と企業が一体になっています。企業で実際に仕事をしている方が教えに来るということがよくありました。話がとてもおもしろかったです。

初めて触ったコンピュータは何でしたか。

Intelの386CPUを積んだパソコンでした。一番初めは……ゲームをしました(笑)。そのあと、「Pascal」を学びました。ウランバートルの大学で「C」を学び、次に「C++」を勉強しました。オブジェクト指向の勉強を多くしましたね。Icalc[注1]のようなプログラムをCで書いたことがあります。実際にパソコンで動いたときには、とてもおもしろいと感じました。これは仕事とは関係なくコンピュータとプログラミングが純粋におもしろいと思ったいい経験でした。

今の仕事(ユニケージと学校設立)は楽しいですか？　日本人や日本の社会に期待することなどはありますか？

日本発であるユニケージ[注2]をもっと海外に展開したいです。今モンゴルで設立している学校では、ユニケージを2年生の後半から教えるプログラムを考案中です。ビジネスも一緒に教えることで、効果が高まると思っています。日本に期待することは、日本人はとても力があるのでその力を信じていろんなことに挑戦してほしいということです。日本人には底力があり、スキルもあると思います。失敗することがあったとしても能力を信じて挑戦してほしい。自分は、失敗は忘れるようにします。そのほうがいい。

モンゴルは国としてまだ成長しますか？

もちろん成長します！　経済発展して多くの国から人が来て、みなさんの夢を実現し、ハッピーな国になります(笑)。今はまだまだ空港で両替ができないなどの社会インフラの課題があります。空港から変わると思いますよ！

それは楽しみですね。モンゴルは相撲文化も盛んですし、日本のIT業界とも、盛んな技術交流でもっと仲良くなるといいですよね。今日はどうもありがとうございました。SD

注1) シェル上で動作する高精度演算コマンド。

注2) コマンドを組み合わせてシェルスクリプトで作る手法。




はんだづけカフェなう

Intel Galileo Gen 2とRaspberry Pi Model B+

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

はじめに

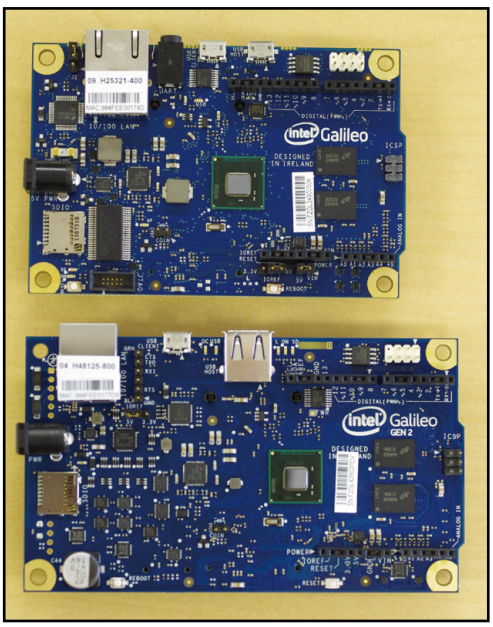
去る8月2日に、IntelのQuark搭載ボード、GalileoのGen 2が発売されました（**写真1**）。従来発売されていた基板と比較すると少しだけ大きくなったのですが、パッと見は違いがわかりづらいです。新しくなったGalileoの何が変わったのか、また、Galileoの魅力について、今回は紹介していきたいと思います。

Galileoとは

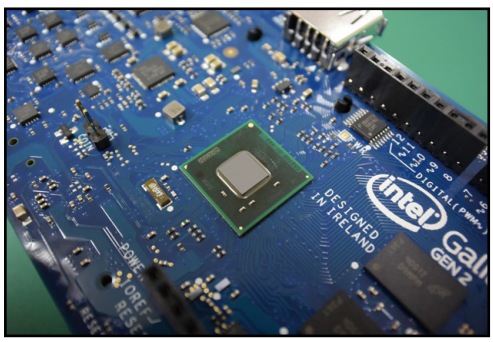
そういえば、Galileoについては、2013年12月号の第38回などで少し触れた程度で、具体的に紹介をしていませんでした。Galileoは、Intelが出すArduino Certifiedなボードです。Arduino Certifiedというのは、Arduinoが提供するプログラムの名前で、Arduinoが出しているIDEでサポートされていなかったCPUを搭載したボード向けに、Arduinoの基本的な機能の互換性があることの認定を提供するものです。つまり、GalileoはArduinoと一定の互換性があることが確認されています。実際、Galileo専用のものですが、Arduino IDEを使ってGalileoで動くスケッチを書き、コンパイルと転送を行うことができます。

Galileoには、Intel Quark X1000という低消費電力プロセッサが搭載されています（**写真2**）。このQuark X1000は、32bitのPentium命令セットという懐かしい仕様ですが、Atomの1/10程度の低消費電力が特徴です。また、SoC（System-on-a-Chip）ということで使うために必要になる周辺回路が少なくなっています。写真を見ると、Pentiumクラスの CPUが載っている基板なのに、真ん中のCPUと、その右のDDR3メモリ、Ethernetコネクタの左のPHY（Ethernetの物理層を担当するチップ）くらい

▼写真1　Galileo（上）とGalileo Gen 2（下）



▼写真2　Quark X1000



しかx86のマザーボードを構成するチップがありません。

Galileoのボード自体はLinuxが走るようになっており、ArduinoのスケッチはLinuxのユーザランドで動くアプリケーションにコンパイルされます。先ほど記したように、Quark X1000はPentium命令セットが採用されていますので、i586用のgccがArduino IDEに同梱されており、このコンパイラでバイナリがビルドされます。

Galileoは従来のArduinoのシールドを搭載できるようにコネクタが配置されています。Arduinoといえばたいてい5Vですが、この連載でも触れているように最近の半導体は3.3Vのものが多くなってきています。Galileoは5Vと3.3Vが切り替えられるようになっており、とても便利そうです。Linuxが動く、電子工作やIoT(Internet of Things:モノのインターネット)用のボードというと、Raspberry Piが有名どころです。実際、Raspberry PiとGalileoの違いについて尋ねられることがよくあります。Raspberry Piは、ARM11ファミリのCPUが搭載されています。このARM11というのは、NASやルータなどの組み込みLinux機器のCPUによく採用されているCPUコアです。一方でGalileoは先述のようにPentium命令セット、つまりx86アーキテクチャのボードです。MicrosoftがWindows Developer Program for IoTというプログラム[注1]で、IoT専用のWindowsをGalileoで実行して、Visual StudioでWin32環境で開発を行えるしくみを提供しています。

Windowsと聞くと、いつも我々が使っているWindows 7や8のようなGUI環境を想像してしまうかもしれません。しかし、"for IoT"とついているように、Galileo用に配布されているWindowsは別の製品です。Windows ServerのServer Coreのように、Windowsのコア部分をGalileoで実行して、GUIはなく、telnetでコマンドラインにアクセスすることが可能なWindowsです。そもそも、Galileoには、Raspberry Piのようにディスプレイを接続する端子はありません。

先述のようにVisual Studio環境を使うことができ、リモートデバッグができますから、デバッグ環境のないArduinoからすると、なかなかおもしろい環境が手に入ります。筆者はあまり馴染みがないのですが、日頃Visual Studioを使って開発している方にはとても親しみやすい開発環境なのだろうと思います。やはり道具は馴染みのあるものがよいですからね。

Gen 2で何が変わった

Gen 2は、基本的には最初のGalileoのイケていなかった点が修正されたような製品です。

最初のGalileoの最もよくなかった点は、GPIO(General Purpose Input/Output:汎用入出力)が低速だということでした。当たり前ですが、GalileoのCPU、Quark X1000はArduinoのCPUであるATmega328Pの100倍くらい速いです。しかし、最初のGalileoのGPIOは、Arduinoより50倍程度遅かったのです。最初のGalileoはGPIOエクスパンダというチップを使ってGPIOを搭載していたため、どうしても処理が低速になってしまっていました。Gen 2では、このGPIOエクスパンダを経由せずにQuark X1000のGPIOを直接使うように仕様変更(図1)がなされたため、このGPIOの速度問題が解決されています[注2]。

Galileoで動いているLinuxにログインするには、telnetなどのネットワーク越しか、シリアルコンソールを使います。このシリアルコンソールのシリアル端子が、最初のGalileoでは、3.5mmミニジャックの形状をしており、なおかつ、信号レベルがRS-232Cという仕様でした。ネットワーク機器であれば、RS-232Cという仕

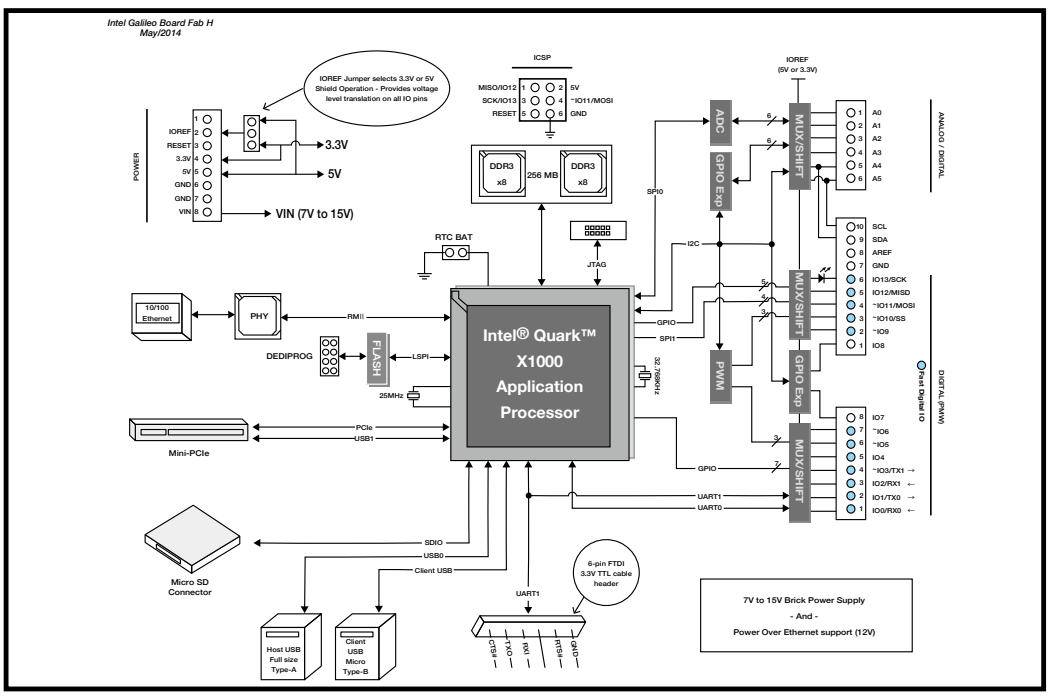
注1) https://dev.windows.com/en-us/featured/Windows-Developer-Program-for-IoT

注2) ただし、D7とD8、A0～A5は、Gen 2でもGPIOエクスパンダ経由です。

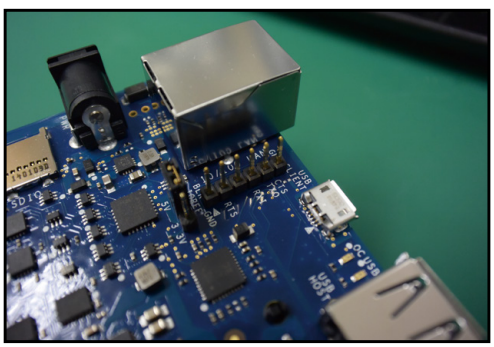
様のものが一般的ですが、マイコンボードでは一般的に5Vや3.3Vの信号電圧のシリアル(UART)でシリアルコンソールにアクセスします。最初のGalileoのこの仕様が、マイコンボードを使っている人たちには不評でした。Gen 2では、この仕様が改められ、FTDI USB-シリアル変換アダプタ互換配列のピンヘッダになりました(**写真3**)。信号レベルは3.3Vですので、マイコンを使う人であればたいていは持っているであろう、シリアル変換アダプタを使ってシリアルコンソールに接続できるようになります。

また、Gen 2では、電源の仕様も大きく変更されました。最初のGalileoでは、5VのACアダプタが添付されており、5V専用でした。これに対して、Gen 2は、7～15Vという仕様に変更されました。添付のACアダプタも12Vになっています。ここまでは大した違いではないのですが、Gen 2は、SilvertelのAg9712-SというPoE(Power Over Ethernet)モジュールを**写真4**の部分に取り付けることで、IEEE 802.3afのPoEに対応して受電をさせることが

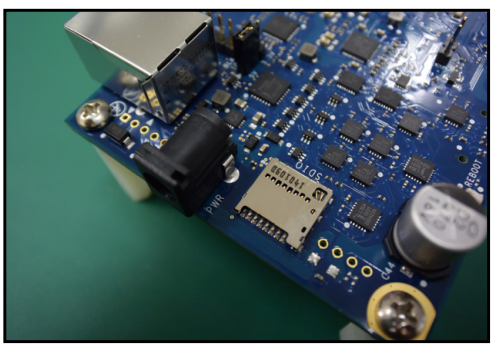
▼図1　Galileo Gen 2のブロック図



▼写真3　Gen2のシリアルコンソール



▼写真4　PoEモジュールを取り付ける場所



できます。とてもIoTボードらしくなりました。

以上の3つからすると、あまり大きな違いではないのですが、地味に便利なのがUSBホスト端子の仕様変更です。最初のGalileoは、この端子がMicro-Aだったため、USB接続の装置を接続しようとするとたいていは変換ケーブルが必要でした。しかし、Gen 2からはスタンダードAですので、たいていの機器をそのままGalileoに接続できます(**写真5**)。

Raspberry Pi Model B+

Galileo Gen 2の発売よりも少し前、Raspberry Piの新型、Model B+も発売されました(**写真6**)。こちらはGalileoのGen 2ほどの大きな違いはありませんが、いくつか改良が施されています。

GPIO端子がModel Bの26ピンから40ピンに大幅に増えました。といっても、Galileoのように構成が変更されたわけではないです。従来の26ピンの部分は変更されていませんので、従来のRaspberry Pi用の小基板はそのまま接続できます。

Model B+では、端子類が整理されました。USBポートが4つになったり、コンポジットビデオ端子がなくなってイヤフォンジャックに統合されました。このUSBポートの数が増えたのと同時に、どうやらType Bの泣き所であった電源の問題も改善されているようです。Type Bでは、電力を要するUSB機器を接続するとRaspberry Piが不安定になったのですが、Model B+ではこの問題が起きなくなっているとよいですね。

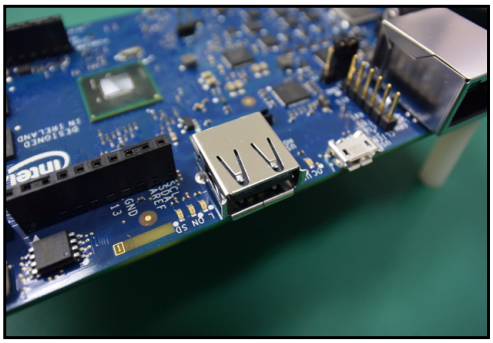
以前のRaspberry Piは基板の4辺からコネクタがせり出していて、なんだか騒がしかったのですが、2辺にまとまってかなりスッキリしました。また、Type Bは、SDカードを挿すようになっていて、これがとても飛び出ていたのですが、Model B+ではmicroSDになり、飛び出る部分もとても少なくなりました。

Raspberry Pi Model B+が発表されたとほぼ同時に、Raspberry Piにアドオンする基板のための「HAT」(Hardware Attached on Top)という規格が発表されました。Raspberry Pi側からどんな基板がつながっているかわかるようにするためのEEPROMの搭載が必須になるなど、よりユーザが使いやすくなる工夫がなされた分、設計や製造には一手間増えるのではないかと思います。Raspberry Piのブログ記事によると、この仕様は開発業者に強制するものではないが、この規格に準拠していないものにHATという名称は使えないということです。

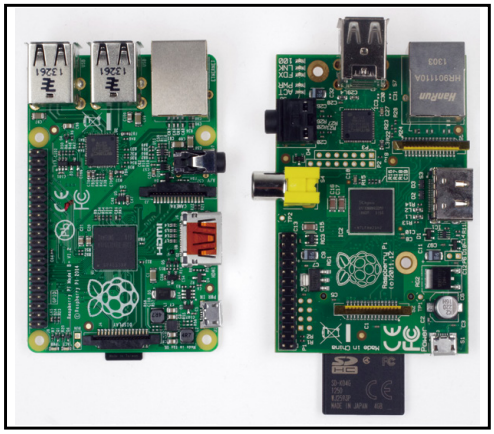
まとめ

Galileoを触るのが忙しくて、Raspberry Piのほうはあまり触っていないので紹介が短くなってしまいすみません。また、この連載でも、GalileoやRaspberry Piを使って作ったものを紹介していきたいと思います。SD

▼写真5　USBレセプタクル



▼写真6　Model B+(左) Type B(右)



PRESENT 読者プレゼントのお知らせ

『Software Design』をご愛読いただきありがとうございます。本誌サイト http://sd.gihyo.jp/ の「読者アンケートと資料請求」からアクセスし、アンケートにご協力ください。ご希望のプレゼント番号をご入力いただいた方には抽選でプレゼントを差し上げます。締め切りは 2014 年 10 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 ドッキングステーション BOOMERANG JUD480

1 名

ブーメラン型のドッキングステーション。USB3.0 でパソコンと接続し、ほかのインターフェースを USB 経由で利用できる。インターフェースは、HDMI ／マイク・スピーカー端子／ VGA ／ Ethernet ／ 4 つの USB3.0 type A（うち 2 つは 1.5A 出力充電可）です。

提供元 加賀ハイテック
URL http://www.j5create.com/jpn/

02 SSD370 64GB モデル

1 名

ノートパソコンに最適な、軽くてコンパクトな SSD。規格は SATA Ⅱ 6Gb/s、サイズは 2.5 インチで 7mm 厚。最大で毎秒 570MB の読み出し／ 470MB の書き込み速度を実現できます。「DevSleep モード」により消費電力を抑え、より短い時間で動作モードに復帰できます。

提供元 トランセンドジャパン **URL** http://jp.transcend-info.com/

03 超ホーダイ 1 年版（パッケージ版）

2 名

120 本以上のソフトウェアを、自由に選んで使えるサービスです。パッケージ版には PIN 番号が封入されており、インターネット経由でソフトウェアをダウンロードできます。1 年間の期限で、新たな製品のダウンロード、インストール済み製品の起動ができます。

提供元 ソースネクスト **URL** http://www.sourcenext.com/

04 Java8 ではじめる「ラムダ式」

清水 美樹 著／
A5 判、192 ページ／
ISBN = 978-4-7775-1841-8

2 名

Java 8 から追加された「ラムダ式」を学ぶ本。Java の従来の記法と、ラムダ式で書かれたコードを比較しながら解説しています。ラムダ式に深く関わる Java 8 からの新 API の紹介もされています。

提供元 工学社 **URL** http://www.kohgakusha.co.jp/

05 エンジニアのためのフィードバック制御入門

Philipp K. Janert 著／
野原 勉 監訳／星 義克、米元 謙介 訳／
A5 判、344 ページ／
ISBN = 978-4-87311-684-6

2 名

ソフトウェアエンジニアに向けて書かれた、「フィードバック制御」の入門書。Python のシミュレーションコードを使って、フィードバック原理を、ソフトウェアシステムに活用する方法を解説しています。

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

06 Vim script テクニックバイブル

Vim script サポーターズ 著／
A5 判、320 ページ／
ISBN = 978-4-7741-6634-6

2 名

Vim をより使いこなしたい人を対象とした、Vim script の入門書です。既存のプラグインをカスタマイズしたり、新たに作ったりして、自分だけの Vim エディタにカスタマイズしましょう。

提供元 技術評論社 **URL** http://gihyo.jp/

07 Hinemos 統合管理［実践］入門

倉田 晃次、澤井 健、幸坂 大輔 著／
B5 変形判、520 ページ／
ISBN = 978-4-7741-6984-2

2 名

オープンソースの統合管理システム「Hinemos」の入門書です。想定される事例に基づいた、実践的な運用方法を学べます。現在 Hinemos の導入を検討されている方にもお勧めです。

提供元 技術評論社 **URL** http://gihyo.jp/

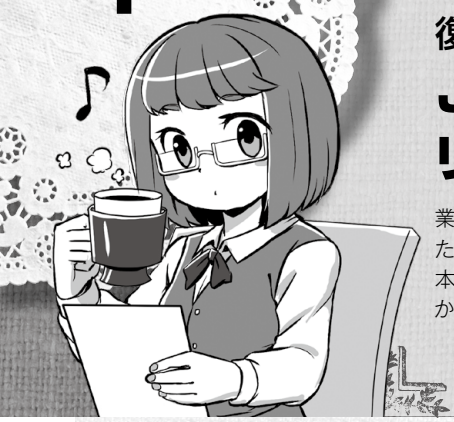
第1特集

言語仕様・開発環境・デバッグ機能
あなたはどこまで使いこなせてる？

今ふたたびのJava

1996年に公開されて以来、Javaは今でも機能が追加／改善されています。2014年3月にはJava SE 8(以下、Java 8)もリリースされました。しかし、ユーザ数の多いプログラミング言語だけあって、基本的な文法だけでコーディングしている初心者や、昔ながらの機能だけでずっと開発しているベテランなど、新しい機能を使いこなせていない人もいるかもしれません。本特集では、プログラミング技法、統合開発環境、トラブルシューティングのそれぞれにおいて、Javaの新しい便利な機能をあらためて整理します。この機会に、「自分はJavaの力を発揮しきれているか」を確認してみてください。

イラスト　高野 涼香



復習「総称型」「コレクション」「列挙型」……

Java 5/6/7の機能にみるリファクタリングの要点

業務でずっとJavaを使っていると、古いJDK(Java Development Kit)を使い続けていたり、新しく追加された便利な機能を使わずに古いコードのままだったりしませんか? 本章では、Java 5/6/7で追加されたおもな機能を紹介します。日々のコードを見直すきっかけにしてみてください。

●アリエル・ネットワーク㈱　大谷 弘喜(おおたに ひろき)

総称型

総称型(ジェネリック型)が導入されたのはJava 5です。総称型は、Java以外のC++やC#など、ほかの言語でも利用されているしくみです。**汎用的なクラスを特定の型に縛るために利用します。**ちょっと難しいので、具体的なコードを見ながら総称型を理解しましょう。

Java 5以前の環境でListインターフェースを扱う場合、**リスト1**のように利用します。

①でArrayListオブジェクトを作成して、②で"Hello"と"World"の2つの文字列を追加しています。このとき、追加した2つの文字列はObject型として扱われます。③では、②で追加した文字列のうち、0番目の値をgetメソッドで取得しています。getメソッドの戻り値はObject型ですので、文字列変数strに代入するためにはダウンキャストする必要があります。

しかし、②で追加するオブジェクトはObject型ですので、Integer型のオブジェクトなど、どのようなオブジェクトでも追加できてしまいます。②でString型以外の値を追加した場合、③でIntegerオブジェクトをString型にダウンキャストして代入しようとすると、実行時にClassCastExceptionが発生しエラーになります。

このように、Listに追加されるオブジェクトの型と取得する型が実行時にしか判別できないと、バグのあるコードが顧客先に納入される可能性もあります。総称型は、汎用的なList型に入出力できる値を特定の型に縛り、不用意な値の入出力を抑制するためのしくみです。それでは、**リスト1**のコードを総称型を利用して書き換えてみましょう。

リスト2の④では、総称型で型引数にString型を指定してListオブジェクトを作成しています。型引数とは、総称型に注入する型です。つまり、Listオブジェクトが扱える型をStringに限定しています。⑤では作成したlistオブジェクトに"Hello"と"World"の2つの文字列を追加しています。このときに、list.add(0)のようにString型以外を指定すると、コンパイル時にエラーが発生します。⑥で、getメソッドでlistのインデックス番号0の位置の値を取得しています。List<String>が扱う型はString型に限定されるので、ダウンキャストの必要がなく、安全にリストを扱えます。

このように総称型を使うことで、次のようなメリットがあります。

▼リスト1　Java 5以前のListの使用

```
List list = new ArrayList(); ←①
list.add("Hello");  ┐
list.add("World");  ┘ ②

String str = (String) list.get(0); ←③
```

▼リスト2　総称型を利用してListを使用

```
List<String> list = new ArrayList<String>(); ←④
list.add("Hello");  ┐
list.add("World");  ┘ ⑤

String str = list.get(0); ←⑥
```

・ダウンキャストが必要なく、型安全である
・型の指定を間違えるとコンパイル時にエラーが発生する
・コレクションで扱う集合などで、扱っている型を明示できる

総称型によるクラス定義

リスト2では、既存の総称型インターフェースであるListを使いましたが、自分で総称型のクラスやインターフェースを定義する場合は、**リスト3**のように記述します。クラス名のあとに<>を記述し、その中に型変数を指定します。型変数とは、型(クラスやインターフェース)をパラメータ化したもので、型を変数のように扱えます。

型引数は、総称型のクラスやメソッドを利用するときに型を特定するための名前です。一方、型変数は、総称型のクラスやメソッドを定義するときに利用します。名前が似ているので注意してください。慣例的に、型変数にはTやEを使うことが多いです。それぞれTypeやElementの頭文字に由来します。型変数は2つ以上指定できます。その場合は、「,」で区切ります。

2つの値を保持できるPairクラスは、総称型を使うと**リスト4**のように書きます。(1)では、型変数にLとRの2つを指定しています。(2)と(3)はPairクラスが内部に持つデータです。2つの変数の型は、型変数で指定したLとRです。(4)のPairクラスのコンストラクタでは、型変数で指定した型の変数を受け取ります。LとRの具体的な型は、「`new Pair<String, Integer>("foo", 1);`」のように、インスタンス化時に決定されます。このように総称型では型をパラメータ(変数)のように扱えます。

制限付きパラメータ

総称型で型変数を定義するときに、**リスト4**のように総称型を定義すると、型引数としてプリミティブ以外のすべての型を指定することができます。しかし、実際に利用するときはある特定のクラス、またはその派生クラスに限定したい場合があります。利用するクラスを限定することで、そのクラスが持つメソッドを実行できるようになります。インスタンス化で指定したクラス以外の型を指定すると、コンパイル時にエラーが発生します。

リスト5のコードは、Pairクラスの型Lを

▼リスト3　総称型によるクラス定義

```
public class クラス名 <型変数> {
    private 型変数 var;
    public void setVar(型変数 var) {
        this.var = var;
    }
    public 型変数 getVar() {
        return this.var;
    }
}
```

▼リスト4　総称型によるPairクラス

```
public class Pair<L, R> { // (1) 型LとRを利用することを宣言
    private L l; // (2) 型Lの変数宣言
    private R r; // (3) 型Rの変数宣言
    public Pair(L l, R r) { // (4) 型LとRの2つの変数を引数に初期化
        this.l = l;
        this.r = r;
    }
    public L getLeft() { return l; }
    public R getRight() { return r; }
}
```

▼リスト5　extendsで制限を付けたPairクラス

```
public class Pair<L extends Number, R> { // (5) 型LはNumber型の派生クラスに限定
    private L l; // 型Lの変数宣言
    private R r; // 型Rの変数宣言
    public Pair(L l, R r) { // 型LとRの2つの変数を引数に初期化
        this.l = l;
        this.r = r;
    }
    public L getLeft() {
        System.out.println(l.intValue()); // (6) 型Lのintvalueメソッドをコール
        return l;
    }
    pulbic R getRight() { return r; }
}
```

▼リスト6　総称型で定義されたクラスを利用

```
クラス名<型引数> var = new クラス名<型引数>(引数...);
```

▼リスト7　Pairクラスの利用

```
Pair<Integer, String> pair = new Pair<Integer, String>(1, "foo");
```

Numberクラス、またはその派生クラスに限定しています。getLeftメソッドをコールすると、標準出力にNumberクラスのintValueメソッドを呼び出して出力します。

(5)では、Lは制限付きパラメータとして型変数を指定されています。制限付きパラメータは「**型変数 extends 基底クラス**」という構文になります。基底クラスには、クラスだけでなくインターフェースも指定できます。制限付きパラメータの定義では、クラスを定義するときとは違い、インターフェースを使用する場合でもextendsを指定します。(6)では、intValueメソッドを呼び出しています。変数lの基底クラスはNumber型です。そのため、Number型が持つintValueメソッドを呼び出せます。

総称型で定義されたクラスの利用

総称型で定義されたクラスを利用する場合は**リスト6**のように記述します。

先ほどのPairクラスを利用する場合は**リスト7**のようになります。今回はPairクラスで型引数として、<>内に数値型と文字列型を指定しています。コンストラクタでは、数値の1と文字列のfooを指定しています。

さて、総称型を使ったクラスをインスタンス化するときに、引数の宣言と実際のコンストラクタに型引数を二重に指定するのは煩雑です。Java 7からは、この手間を低減するためにダイヤモンドオペレータが導入されました。これによりインスタンス化するときの型引数を「**<>**」のように指定して、実際の型の指定を省略できます。**リスト8**はダイヤモンドオペレータを利用してインスタンス化しています。

▼リスト8　ダイヤモンドオペレータの利用

```
Pair<Integer, String> pair = new Pair<>(1, "foo");
```

▼リスト9　List<String>を保持するPairオブジェクトのインスタンス化

```
Pair<List<String>, List<String>> pair = new Pair<List<String>, List<String>>();
```

▼リスト10　Pairオブジェクトのインスタンス化(ダイヤモンドオペレータ使用)

```
Pair<List<String>, List<String>> pair = new Pair<>();
```

今回は型引数がInteger型とString型だけでしたが、指定する型が総称型の場合、このシンタックスシュガー[注1]は威力を発揮します。たとえば、PairがList<String>を保持するオブジェクトをインスタンス化する場合は、**リスト9**のようなコードになります。<>の中がかなり長くなり、可読性が悪いです。

これをダイヤモンドオペレータを使うと、**リスト10**のようにシンプルになります。Pairオブジェクトの型は、宣言を見ればわかるので、具体的にどのようにインスタンス化されるか、プログラマは意識する必要がありません。

メソッドでの利用

総称型はクラスだけでなく、メソッドに対しても利用できます。型宣言は、次のようにメソッドの戻り値の型の直前で指定します。

```
public <T> T method()
```

それでは、具体的にコードを書いてみましょう。**リスト11**は、配列を引数に渡して、その中央の値を取得するメソッドです。

⑦の<T>で型変数Tを指定しています。それ以外は通常のクラスでの型変数と同じように使用します。

型引数を用いたメソッドの呼び出しは、総称型のクラスを利用するときよりも簡単です。**リスト**

注1) 元からある構文をより簡単に読み書きできるようにするために導入された構文。

12の⑧では、リスト11で定義したgetMiddleメソッドを呼び出しています。総称型のクラスのときとは違って、明示的に型引数を指定する必要はありません。メソッドでの型引数は自動で判断されます。

また、リスト13の⑨のようにメソッドの直前で型指定を行うことで、明示的に型引数を指定することもできます。

総称型ではできないこと、制限的なこと

総称型はとても便利な機能ですが、次のような制約もあります。

・型引数を使ってインスタンス化できない
・型引数を使ってinstanceofでの型チェックができない
・共変でない[注2]

コレクションAPI

Java 5以降、コレクションAPIは拡張されています。第2章で説明するJava 8でもストリーム機能のためにコレクションAPIは拡張されています。ここでは、代表的なコレクションクラスのListとSet、Mapの簡単な使用方法を説明します。

List

複数件の要素を扱う場合、配列を利用するプログラマも多いです。事前に配列のサイズがわかっている場合や、配列の途中に要素を挿入／削除しない場合は、配列での操作で十分です。しかし、サイズが不定の場合や、要素の挿入や削除を行う場合は、配列を利用するとコードが煩雑になります。そのような順序付けされた要素を扱うためのデータ型がjava.util.List型です。

List型はおもに次の機能を提供します。

・インデックスによる要素へのアクセス
・要素の指定位置への追加／削除

java.util.Listはインターフェースです。使用する場合は、java.util.ArrayListやjava.util.LinkedListを使います。実際に要素を扱うアルゴリズムによって使用するList型の具象クラスを選択します。

ArrayListは内部の要素を配列で保持して、**要素の追加時に配列のサイズが不足している場合は、配列のサイズを自動で拡張します。**内部データを配列で保持するので、配列の途中への要素の挿入や削除はコストがかかります。一方で、インデックスを指定した要素のアクセスは高速に行えます。

LinkedListはリンクリストの実装です。ArrayListと違い、**指定位置への要素の追加／削除が高速**です。ArrayListのように配列の拡張によるオーバーヘッドが発生しないため、要素の追加は常に一定速度になります。一方で、インデックス指定での要素へのアクセスはArrayListより遅くなります。

それでは、Listを使って要素の追加、取得、削除を見てみましょう。

リスト14の①ではArrayListをインスタンス化しています。②では、listの末尾に要素を追加しています。③のaddメソッドは、要素の挿入位置を指定して追加しています。この場合はリストの先頭(インデックスが0番目の位置)に、要素を

▼リスト11　中央値を取得するメソッド

```
public class A {
    public static <T> T getMiddle(T[] args) { ←⑦
        return args[args.length/2];
    }
}
```

▼リスト12　総称型を用いたメソッドの呼び出し

```
String[] array = new String[] {"foo", "bar", "bazz"};
System.out.println(A.getMiddle(array)); ←⑧
```

▼リスト13　型指定で総称型を用いたメソッドの呼び出し

```
String[] array = new String[] {"foo", "bar", "bazz"};
System.out.println(A.<String>getMiddle(array)); ←⑨
```

注2）List<Integer>をIntegerの基底クラスであるNumberを使ったList<Number>などにキャストできない。

追加しています。このため、listのデータは、「"bar","foo"」になります。④では指定したインデックスの要素を取得しています。この場合は、0番目の要素を指定しているのでbarが返ります。Listのインデックスは配列と同様に0から開始します。⑤ではインデックスで指定した位置の要素を削除しています。この場合はインデックスが1ですので、リストの末尾のデータを削除しています。結果としてlistのデータは、「"bar"」になります。⑥では、リストの要素数を取得しています。⑤で要素を削除したので、ここでは1になります。

Set

Setは**重複要素のないデータの集合を扱います**。重複要素とは、オブジェクトをequalsメソッドで比較して等価である状態です。List型と違い、通常、Set型では順番は保持されません。ただし、java.util.SortedSetを使うとSetに追加したデータは、ソートされた順番に並び替えられます。Setが保持するデータに順番にアクセスする場合は、List型と違いインデックスによるアクセスはできません。後述するイテレータを使い、集合を順番になめていきます。

▼リスト14　Listの使用例

```
List<String> list = new ArrayList<>() ←①
list.add("foo"); ←②
list.add(0, "bar"); ←③

String str = list.get(0); ←④
System.out.println(str);

list.remove(1); ←⑤
int size = list.size(); ←⑥
System.out.println(size);
```

▼リスト15　Setの使用例

```
Set<String> s1 = new HashSet<>();
Set<String> s2 = new HashSet<>();                    ⑦
s1.addAll(Arrays.asList("a", "b", "c", "b"));
s2.addAll(Arrays.asList("b", "c", "d"));             ⑧

s1.retainAll(s2); ←⑨
System.out.println(s1.contains("a"));
System.out.println(s1.contains("b"));                ⑩
for (Iterator<String> it = s1.iterator(); it.hasNext();) {
    String str = it.next();
    System.out.println(str);
}                                                    ⑪
```

リスト15は2つのSetのオブジェクトを作成して、それぞれに文字列を追加しています。次に2つのSetから共通する文字列だけを選別しています。

リスト15の⑦で文字列型を保持するHashSetをインスタンス化しています。⑧では、⑦で作成したオブジェクトに文字列を追加しています。s1には「"a","b","c","b"」の4つの文字列を追加していますが、Setは重複要素を保持しないため、「"a","b","c"」の3つの要素だけを保持します。

⑨では、2つのSetのオブジェクトから、共通の要素だけを選別してしています。この場合は、「"a","b","c"」と「"b","c","d"」の2つの集合のうち、共通するものは"b"と"c"ですので、s1は"b"と"c"を保持します。

⑩のcontainsメソッドで、指定した要素が集合に存在するかを確認しています。"a"という要素は存在しないのでfalseが返ります。"b"という要素は存在しているので、trueになります。

⑪では、イテレータ(後述)を使用してSetの集合にアクセスしています。

Map

Mapは、**キーに関連付けされた値(データ)の集合を扱う**データ構造です。Mapにデータを格納するときに、値に対応するキーを一緒に登録します。Mapからデータを取得するときは、登録時に使用したキーを用いて値を取得します。

Map型の具象クラスには、java.util.HashMapやjava.util.TreeMap、java.util.LinkedHashMapなどがあります。Setと同様に通常はMapも順序を保持しませんが、LinkedHashMapは順序を持ちます。

リスト16はMapにデータを挿入して、取得しています。⑫では、HashMapをインスタンス化しています。HashMapはハッシュテーブルのアルゴリズムを利用したMapの実装です。ここでは、キーとバリュー、それぞれに文字列を指定しています。⑬ではputメソッドでマップにデータを格納しています。List型やSet型と違い、Map型ではデータの追加はaddメソッ

ドではなく、putメソッドを使用します。putの第1引数がキー、第2引数がキーに関連付けられた値になります。ここでは、キーに英語の曜日を表す文字列、値には日本語の曜日を表す文字列を指定して追加しています。

⑭でgetメソッドでキーに対応付けられた値を取得しています。キー"Sunday"に対応付けられた値は"日曜日"ですので、ここでは「日曜日」が出力されます。次に、"Someday"に対応するキーは見つからないので、ここではnullが返ります。

Java 8のgetOrDefaultメソッド

たとえば、英文の中の各単語の出現頻度を算出する場合、単語をキーにその出現頻度を値として格納するマップをデータ構造として使用することが多いと思います。その場合、**リスト17**[注3]のようにnullチェックを行ってマップに出現頻度を保持します。

⑮でwordに対応する出現頻度を取得します。

▼リスト16　Mapの使用例

```
Map<String, String> map = new HashMap<>(); ←⑫
map.put("Sunday", "日曜日");
map.put("Monday", "月曜日");
map.put("Tuesday", "火曜日");
map.put("Wednesday", "水曜日"); ⑬
map.put("Thursday", "木曜日");
map.put("Friday", "金曜日");
map.put("Saturday", "土曜日");

System.out.println(map.get("Sunday"));
System.out.println(map.get("Someday")); ⑭
```

▼リスト17　単語の出現頻度をカウント

```
public Map<String, Integer> getFrequency(String[] words) {
    Map<String, Integer> map = new HashMap<>();
    for (String word: words) {
        Integer count = map.get(word); ←⑮
        if (count == null) { ←⑯
            count = 0; ←⑰
        }
        map.put(word, ++count); ←⑱
    }
    return map;
}
```

⑯ではmap内にデータがまだ存在しない場合は、countを0で初期化します(⑰)。⑱では、countをインクリメントして、マップに単語をキーに出現頻度を格納しています。

この処理フローは、プログラミングの現場ではよく見かけます。そこで、Java 8ではgetOrDefaultメソッドが追加されました(**リスト18**)。getOrDefaultメソッドは、指定したキー(key)が存在すれば、それに関連付けられた値が取得できます。もし、キーが存在しない場合は、引数(defaultValue)に指定したデフォルト値が使用されます。

それでは、**リスト17**のコードをgetOrDefaultメソッドを用いて書き換えてみましょう(**リスト19**)。

リスト17では4行(⑮、⑯、⑰の処理)に渡っていたコードが、getOrDefaultメソッドを使うことで**リスト19**の⑲のように1行で簡潔に記述できます。また、処理の流れもより明確になります。

イテレータ

イテレーションとは、**コレクションの要素を順番にアクセスするしくみ**です。

配列の場合は、**リスト20**のように配列の長

▼リスト18　getOrDefaultのメソッドシグネチャ

```
public V getOrDefault(Object key, V defaultValue);
```

▼リスト19　単語の出現頻度をカウント(getOrDefault版)

```
public Map<String, Integer> getFrequency(String[] words) {
    Map<String, Integer> map = new HashMap<>();
    for (String word: words) {
        Integer count = map.getOrDefult(word, 0); ←⑲
        map.put(word, ++count);
    }
    return map;
}
```

▼リスト20　配列による順次アクセス

```
String[] array = new String[] {"foo", "bar", "bazz"};
for (int index=0; i<array.length; i++) { ←❶
    String str = array[i]; ←❷
    System.out.println(str);
}
```

注3）リスト17とリスト19では、「拡張forループ」という書き方で繰り返し処理を書いています。拡張forループについては後で説明を行います。ここではfor文の内側の処理に注目してください。

さを取得して(❶)、forループの中でインデックスアクセスをします(❷)。

配列をList型に置き換えても、同じようにインデックスでアクセスできます。**リスト21**では、リストのsizeメソッドで要素数を取得して(❸)ループを回しています。❹でインデックスによるアクセスを行っています。

リストのアルゴリズムの選択によっては、インデックスアクセスはコストがかかるケースがあります。ArrayListは内部のデータを配列で保持しているために、高速にインデックスアクセスできます。一方、LinkedListを使用すると、先頭からノードを順番にたどる必要があるため、末端のノードにアクセスするほど、インデックスでのアクセスが遅くなります。

Javaではjava.util.Iteratorを使うことで、要素の個数を取得せずにデータに順番にアクセスできます。このとき、コレクションのアルゴリズムに応じて最適化されているため、インデックスでのアクセスのように速度が極端に落ちることはありません。つまり、イテレータはコレクションを順番にアクセスするしくみを汎用化し、実装の詳細を隠蔽して最適化されたアクセス方法を提供します。

リスト21をイテレータを使って書きなおしてみましょう(**リスト22**)。

❺のlistのiteratorメソッドを呼び出すことでIteratorのオブジェクトを取得しています。このオブジェクトを通して、今後、リストの各要素にアクセスします。Iteratorオブジェクトのメソッドで、現在のイテレータが指し示している次の要素が存在するか、チェックしています。次の要素がない場合は、ループから抜けます。

❻では、イテレータから次の要素を抜き出して、イテレータが指し示す要素の位置を次に進めます。

リスト15のSetの使用例でもイテレータによる順次アクセスを行いました。List型でもSet型でもIteratorを使うことで、同じように各要素に順番にアクセスできます。

次にMap型の場合はどうでしょうか? Mapでは、キーの集合をkeySetメソッドで取得できます。keySetメソッドはSet型のオブジェクトを返すので、このSet型のオブジェクトのiteratorメソッドを呼び出すことでイテレーションできます。

リスト23の❼のmap.keySet()でキーの集合を取得し、イテレータをiteratorメソッドを呼ぶことで取得しています。❽では、nextメソッドを呼び出して、イテレータからキーを取得しています。❾でそのキーに対応する値をマップから取得しています。

リスト23では、キーの集合を取得してから、ループの中でキーに対応する値を取得していました。マップをイテレーションする場合は、キーと値のペアを取得したいことが多いです。その場合、entrySetメソッドを使うことで、キーと値のペアを一度に取得できます。

リスト24の❿のentrySetメソッドでは、Map.EntryインターフェースのSetの集合を返して、順番にイテレーションします。⓫では、

▼リスト21 Listのインデックスによる順次アクセス

```
List<String> list = Arrays.asList("foo", "bar", "bazz");
for (int index=0; i<list.size(); i++) { ←❸
    String str = list.get(index); ←❹
    System.out.println(str);
}
```

▼リスト22 IteratorによるListの順次アクセス

```
List<String> list = Arrays.asList("foo", "bar", "bazz");
for (Iterator<String> it=list.iterator();
     it.hasNext();) {                          ←❺
       String str = it.next(); ←❻
       System.out.println(str);
}
```

▼リスト23 Mapのキーによる順次アクセス

```
Map<String, String> map = new HashMap<>();
map.put("Sunday", "日曜日");
("Monday"～"Friday"の部分はリスト16と同じ)
map.put("Saturday", "土曜日");

for (Iterator<String> it=map.keySet().iterator();
     it.hasNext(); ) {                         ←❼
       String key = it.next(); ←❽
       String value = map.get(key); ←❾
       System.out.println(value);
}
```

▼リスト24　Mapのエントリによる順次アクセス

```
Map<String, String> map = new HashMap<>();
map.put("Sunday", "日曜日");
（"Monday"～"Friday"の部分はリスト16と同じ）
map.put("Saturday", "土曜日");

for (Iterator<Map.Entry<String, String>>
     it=map.entrySet().iterator(); it.hasNext(); ) {  ←⑩
       Map.Entry<String, String> entry = it.next();  ←⑪
       String key = entry.getKey();  ←⑫
       String value = entry.getValue();  ←⑬
       System.out.println(value);
}
```

▼リスト25　Mapのエントリによる順次アクセス（拡張forループ版）

```
Map<String, String> map = new HashMap<>();
map.put("Sunday", "日曜日");
（"Monday"～"Friday"の部分はリスト16と同じ）
map.put("Saturday", "土曜日");

for (Map.Entry<String, String> entry:
     map.entrySet()) {                    ←⑭
       String key = entry.getKey();
       String value = entry.getValue();
       System.out.println(value);
}
```

▼リスト26　拡張forループの書式

```
for（要素の型 変数 :  Iterableオブジェクト）
```

▼リスト27　配列による順次アクセス（拡張forループ版）

```
String[] array = new String[] {"foo", "bar","bazz"};
for (String str: array) {  ←⑮
    System.out.println(str);
}
```

イテレータからEntryオブジェクトを取得しています。Entryオブジェクトは各要素のキーと値を保持しています。⑫と⑬で、Entryオブジェクトからキーと値を取り出しています。

拡張forループ

さて、前項ではイテレータによる順次アクセスで、データ構造に依存せずに統一的に扱えることがわかりました。しかし、List型やSet型のイテレーションと違って、Map型のentrySetによるイテレーションはとても煩雑に見えます。総称型を使うことで、for文がとても見難くなってしまいました。また、イテレータのnextメソッドの呼び出しも冗長に感じます。

拡張forループを使うと、イテレータによる順次アクセス（インデックスの移動と、次の要素の有無チェック）を隠蔽してくれて、コードがスッキリします。**リスト25**は、**リスト24**のMapのエントリによる順次アクセスを拡張forループにより書き換えたものです。

呪文のようなコードが消えて、すっきりしました。拡張forループは**リスト26**のような書式になります。

java.util.Iterableインターフェースを実装したクラスであれば、拡張forループを使用してループを回せます。**リスト25**では⑭のMap.Entry<String, String>が要素の型、entryが変数名になります。map.entrySet()でSet型のオブジェクトを返していますが、SetはIterableインターフェースを実装しているので、拡張forループで回せます。

拡張forループのほうが、明示的にiteratorを使用するよりコードが簡潔になります。ただ、iteratorを使う場合、removeメソッドを呼ぶことでループの途中で要素の削除が行えます。しかし、拡張forループを使用すると、ループの途中で要素を削除できません。通常は拡張forループを使って、要素の削除を行いたいケースだけ、iteratorを明示的に指定して古いforループを使ってください。

さて、拡張forループは、Iterableオブジェクトだけでなく、配列も同様に回すことができます。**リスト27**は「配列による順次アクセス」を拡張forループで書きなおしたものです。⑮でIterableオブジェクトの代わりに配列を拡張forループで指定しています。インデックスによるアクセスと見比べてみて、拡張forループによるアクセスのほうがコードがスッキリしていると思います。

列挙型

列挙型とは、**同じ型の定数の集合を列挙して1つの型として扱うためのしくみ**です。

基本的な使い方

列挙型が導入されるJava 5以前では、曜日を扱うためのしくみは**リスト28**のように数値型として記述していました。

リスト29は、**リスト28**の定数から曜日を表す文字列を返すメソッドです。①のメソッドの宣言では、引数にint型の値をとり、名前を取得するメソッドであることがわかります。曜日を数値型として扱っていますが、変数の型に意味はありません。利用する側は、変数weekdayに指定できる値をドキュメントからしか確認できません。また、プログラム上の不具合から事前に定義した変数の範囲外の値がこのメソッドに渡される危険もあります。こうした危険は、プログラムの実行時にしか確認できません。

列挙型はSUNDAYからSATURDAYまでの定数の集合を列挙して、Week型として扱えます。

▼リスト28　列挙型以前の定数の定義

```
public class Week {
    final public static int SUNDAY=0;
    final public static int MONDAY=1;
    final public static int TUESDAY=2;
    final public static int WEDNESDAY=3;
    final public static int THURSDAY=4;
    final public static int FRIDAY=5;
    final public static int SATURDAY=6;
}
```

▼リスト29　列挙型以前の定数の利用

```
public String getName(int weekday)  { ←①
    switch (weekday) {
    case SUNDAY:
        return "SUNDAY";
    case MONDAY:
        return "MONDAY";
    case TUESDAY:
        return "TUESDAY";
    case WEDNESDAY:
        return "WEDNESDAY";
    case THURSDAY:
        return "THURSDAY";
    case FRIDAY:
        return "FRIDAY";
    case SATURDAY:
        return "SATURDAY";
    default :
        return "";
    }
}
```

列挙型は**リスト30**のフォーマットで記述します。まず、列挙型はenumで記述します。続いて列挙名を指定します。ブロック内には列挙子を列挙します。列挙子と列挙子は「,」で区切ります。最後の列挙子のあとの「,」は、あってもなくてもかまいません。通常の定数と同様に列挙子も慣例的に大文字で記述します。

リスト28のWeekクラスを列挙型を用いて書くと、**リスト31**のようになります。

列挙型を使って**リスト29**のgetNameメソッドを書き換えてみましょう。**リスト32**の②のようにメソッドの引数がint型から列挙型Weekになりました。このことで、このメソッドの引数は曜日を表す列挙型Weekであることがわかります。また、ドキュメントやメソッドの実装を確認しなくても、列挙型Weekが持っている列挙子がすぐに確認できます。さらに、Weekで定義した値以外を引数に指定できないため、コンパイル時に値の妥当性チェックが行えます。

列挙型は**リスト32**のように値の比較をswitch

▼リスト30　列挙型の記述フォーマット

```
enum 列挙名 {
    列挙子1,
    列挙子2,
      :
}
```

▼リスト31　列挙子によるWeek型の定義

```
enum Week {
    SUNDAY,
    MONDAY,
    TUESDAY,
    WEDNESDAY,
    THURSDAY,
    FRIDAY,
    SATURDAY
}
```

▼リスト32　列挙型でのgetNameメソッド

```
public String getName(Week weekday) { ←②
    switch (weekday) {
    case SUNDAY:
        return "SUNDAY";
    (MONDAY～FRIDAYの定義はリスト29と同じ)
    case SATURDAY:
        return "SATURDAY";
    }
}
```

文でできるほか、「if (weekday == Week.SUNDAY)」のようにif文で真偽比較もできます。

クラスとしてのenum

C言語の列挙型は定数値の定義しか行えません。しかし、Javaの列挙型は、java.lang.Enumクラスを継承したクラスになります。つまり、Javaの列挙型は定数の定義を列挙する機能だけでなく、内部に状態やメソッドなどクラスとしての特性を併せ持ちます。

列挙型には、よく使うものとして**表1**のようなメソッドがあります。

リスト32のgetNameメソッドでは、switch文により分岐していましたが、列挙型のnameメソッドを使えば**リスト33**のように簡単に記述できます。

独自コンストラクタ／メソッドの定義

たとえば、列挙型で定義している値をデータベースに格納したい場合があります。そのときに、列挙子の名前や列挙子の順番ではなく、自分で定義した名前で保存したいとします。列挙子Week.SUNDAYの場合は文字列"SUN"で、Week.MONDAYの場合は文字列"MON"で保存するケースです。この場合、**リスト33**の例のようにgetNameメソッドを別途用意して、switch文で保存する文字列を設定できますが、それ以外に列挙子自体にデータベースに保存する文字列を持たせることもできます。

列挙型はコンストラクタとメソッド、内部フィールドを通常のクラスのように独自に定義できます。コンストラクタは、列挙子を列挙するときに引数と一緒に指定できます。

リスト34は、"SUN"や"MON"のような短縮名を列挙型の内部に持たせています。

まず、④で各列挙子が持つフィールドを定義しています。ここでは短縮名を保持するabbrを定義しています。⑤は列挙子のコンストラクタです。ここでは引数を1つとり、abbrにセットしています。このコンストラクタは③で列挙子を列挙するときに使用され、初期値をセットしています。独自のコンストラクタやメソッド、フィールドを定義する場合、列挙子の列挙はenumのブロックの先頭に書く必要があります。また、最後の列挙子の末尾に「;」を追加する必要があります。

⑥は、短縮名を取得するための、独自のメソッドを追加しています。Week.WEDNESDAY.getAbbr()をコールすれば、引数で初期化した値"WED"が返ります。このように列挙型は通常のクラスと同様に、内部に追加の状態を持たせたり、コンストラクタ、メソッドを定義したりできます。

▼リスト33　getNameメソッドのnameメソッドでの実装

```
public String getName(Week weekday) {
    return weekday.name();
}
```

▼リスト34　列挙型でのコンストラクタとメソッド

```
enum Week {
    SUNDAY("SUN"),
    MONDAY("MON"),
    TUESDAY("TUE"),
    WEDNESDAY("WED"),   ③
    THURSDAY("THU"),
    FRIDAY("FRI"),
    SATURDAY("SAT");
    final private String abbr; ←④
    Week(String abbr) { ←⑤
        this.abbr = abbr;
    }

    public String getAbbr() { ←⑥
        return abbr;
        }
    }
}
```

▼表1　よく使う列挙型メソッド

メソッド	戻り値	説明	例
values	列挙子[]	列挙子を並べた順番に配列として返す	Week.values()
valueOf	列挙子	指定した名前の列挙子を返す	Week.valueOf("SUNDAY")
name	文字列	列挙子の名前を返す	Week.SUNDAY.name()
toString	文字列	nameメソッドと同じ	Week.SUNDAY.toString()
ordinal	int	列挙子の順番を返す	Week.SUNDAY.ordinal()

アノテーション

アノテーションはJava 5から導入された機能です。アノテーションは「注釈」という意味で、**パッケージ／クラス／メソッド／フィールド／変数に対してメタデータ(付加情報)を与えます。**

たとえば、@Overrideというアノテーションがあります。これはインターフェースやクラスからオーバライドしたメソッドに対して、オーバーライドしていることを明示するものです。このアノテーションを付けなくてもプログラム上の動作は変わりませんが、プログラマがそのメソッドがオーバーライドしていることがわかりやすくなります。

これ以外によく使われるケースとして、標準ライブラリではありませんが、JUnitがあります。テストケースの各テストメソッドに@Testのアノテーションを付けることで、そのメソッドがテスト対象のメソッドとして認識されて実行されます。JUnit3までは各テストケースはメソッド名がtestで始まるという規約がありましたが、アノテーションによりテストするメソッドを明示できるようになりました。また、メソッド名の規約と違い、プログラマにとってもテストメソッドとそれ以外のメソッドが区別しやすくなります。

@Overrideは、プログラマの識別のためのものでしたが、JUnitのアノテーションのようにプログラムの実行時に解釈されてプログラムの動作に影響を与えるものもあります。

標準アノテーション型

Javaの標準ライブラリはいくつかのアノテーションを提供しています。ここでは代表的な3つのアノテーションについて説明します。

@Override

@Overrideは、インターフェースやクラスでオーバーライドしたメソッドに、オーバーライドしていることを明示するために付けます(**リスト35**)。通常、EclipseなどのIDEの機能を使ってメソッドをオーバーライドしたり、クラスを作成したりすると、オーバーライドしたメソッドに対して自動でこのアノテーションが追加されます。

また、コンパイル時にそのメソッドが本当にオーバーライドしたメソッドかのチェックが行われます。スペルミスなどでメソッドシグネチャが一致しない場合は、コンパイル時にエラーになるので、実行時に予期しない動作が発生することがなくなります。

@Deprecated

@Deprecatedは、標準ライブラリのメソッドによく付加されています。メソッドやクラスが将来廃止予定で互換性維持のために残っている場合、それらが非推奨であることを明示します。このアノテーションが付いているメソッドを使ったソースコードをコンパイルすると、コンパイル時に非推奨のメソッドを使っている旨の警告が出ます。特別な事情がない限りは、非推奨のメソッドは使わないようにして、警告を取り除いてください。

▼リスト35　@Overrideをメソッドに付与

```
public class MyThread implements Runnable {
    @Override
    public void run() {
        System.out.println("@override");
    }
}
```

▼リスト36　@Deprecatedをメソッドに付与

```
public class A {
    @Deprecated ←①
    public static void deprecatedMethod() {
        System.out.println("deprecated");
    }
}
```

▼リスト37　deprecatedMethodの呼び出し側

```
public class B {
    public static void main(String... argv) {
        A.deprecatedMethod(); ←②
    }
}
```

@Deprecatedはクラスやメソッドに付与できます。**リスト36**はメソッドに付与している例です。①でdeprecatedMethodメソッドにアノテーションを付与しています。

リスト37はdeprecatedMethodメソッドを利用しているコードです。②でAクラスのdeprecatedMethodメソッドを呼び出しています。

呼び出しは問題なく行えますが、javacでコンパイルすると、**図1**のように警告が出力されます。警告どおりにオプションを指定してコンパイルすると、**図2**のように詳細な警告が表示されます。

@SuppressWarnings

総称型を利用していない**リスト38**のA.javaのようなコードは、歴史が長いアプリケーションであればよく見かけます。このコードは、toListメソッドで文字の配列をListに変換しています。

リスト39はtoListメソッドを利用するコードです。toListの戻り値は総称型のList<String>が利用できることがわかっているので、キャストして利用しようとしています。

このコードをコンパイルすると、**図3**のような警告が出力されます。警告文にしたがってオプションを追加してコンパイルすると**図4**のような詳細なメッセージが表示されます。

AクラスのtoListメソッドも同時に総称型に対応させるべきですが、外部ライブラリだったりすると、必ずしも呼び出し先のコードを変更できるわけではありません。その場合、@SuppressWarningsを付けることで警告を抑えることができます。@SuppressWarningsはメソッドに付けると、メソッド内のすべての警告を抑制してしまいます。メソッド内のローカル変数にアノテーションを付けることで、その変数に対してだけ警告を抑制できます。

警告は出力を抑制するのではなく、警告が発生しないようにソースコードを修正することが大事ですので、抑制する場合は必要最小限の範囲に限定してください。

リスト40は@SuppressWarningsで警告を抑

▼図1　コンパイル結果

```
$ javac B.java
注意:B.javaは非推奨のAPIを使用またはオーバーライドしています。
注意:詳細は、-Xlint:deprecationオプションを指定して再コンパイルしてください。
```

▼図2　-Xlint:deprecationオプションを指定して再コンパイル

```
$ javac -Xlint:deprecation  B.java
B.java:3: 警告: [deprecation] Aのdeprecated Method()は非推奨になりました
        A.deprecatedMethod();
          ^
警告1個
```

▼リスト38　総称型を利用していないA.java

```
import java.util.*;

public class A {
    public static List toList(String[] argv) {
        List l = Arrays.asList(argv);
        return l;
    }
}
```

▼リスト39　総称型を使ってA.javaを利用するB.java

```
import java.util.*;

public class B {
    public static void main(String[] argv) {
        List<String> l = (List<String>)toList(argv);
        for (String s: l) {
            System.out.println(s);
        }
    }
}
```

▼図3　コンパイルによる警告

```
$ javac B.java
注意:B.javaの操作は、未チェックまたは安全ではありません。
注意:詳細は、-Xlint:uncheckedオプションを指定して再コンパイルしてください。
```

▼図4　オプション付きでコンパイル

```
$ javac -Xlint:unchecked  B.java
B.java:5: 警告: [unchecked] 無検査キャスト
        List<String> l = (List<String>)A.toList(argv);
                                        ^
  期待値: List<String>
  検出値:    List
警告1個
```

制しています。③では変数l1に対して@SuppressWarningsを指定しています。@SuppressWarningsを引数なしで呼び出すとすべての警告を抑制します。引数を指定した場合、指定した引数のタイプの警告を抑制します。ここではunchecked(無検査キャスト)の警告を抑制しています。④の変数l2には@SuppressWarningsは付けていません。

リスト40をコンパイルすると**図5**のようになります。コンパイルによる警告は、変数l1への無検査キャストへの警告は③のおかげで抑制されています。しかし、④の変数l2に対しては警告が抑制されずにコンパイラが警告を発しています。

▼リスト40　@SuppressWarningsの利用例

```
import java.util.*;

public class B {
    public static void main(String... argv) {
        @SuppressWarnings("unchecked") ←③
        List<String> l1
          = (List<String>)A.toList(argv);
        for (String s: l1) {
            System.out.println(s);
        }

        List<String> l2                    ┐④
          = (List<String>)A.toList(argv); ┘
        for (String s: l2) {
            System.out.println(s);
        }
    }
}
```

▼図5　コンパイル結果

```
$ javac -Xlint:unchecked  B.java
B.java:11: 警告: [unchecked] 無検査キャスト
        List<String> l2 = (List<String>)A.toList(argv);
                                                 ^
  期待値: List<String>
  検出値:    List
警告1個
```

終わりに

駆け足でJava 5からJava 7までに追加された気になる機能を紹介しました。これ以外にも、java.util.concurrentパッケージなど、魅力的な機能も追加されています。concurrentパッケージは並行処理を行ううえで必須の機能になりますので、これを機に触れてみてください。でわでわ。SD

COLUMN Javaのコードを簡潔にできるlombok

lombok(http://projectlombok.org/)というライブラリをご存じでしょうか？　このライブラリは、Javaの冗長な記述を簡潔に記述するためのしくみを提供しています。その機能の多くがアノテーションを利用しています。たとえば、Javaのコードでsetterやgetterを記述することが多いです。IDEを利用すれば、それらを自動生成することができます。しかし、ソースコード上にsetterやgetterのお決まりの記述が並び、可読性がいいとは言えません。lombokでは、アノテーション@Getter/@Setterをフィールドに付加することで、getter/setterをコンパイル時に自動生成します。

リスト41のようなコードは、lombokを使うと**リスト42**のように記述できます。

これだけでgetter/setterがコンパイル時に自動生成できます。また、getter/setterがソースコード内に散在しないので、フィールドが多い場合、どのフィールドがgetterやsetterを持つのかが変数宣言を見るだけでわかり、メンテナンス性も向上します。IDEのサポートも進んでいるので、lombokを使用してもシームレスにコンパイルやデバッグができます。

@Getter/@Setter以外にもコンストラクタの自動生成やイミュータブルなクラスの自動生成など、便利な機能が多く提供されています。筆者はlombokがなければJavaのコードを書けない体になってしまいました。ただし、ときどきlombokは悪い子になってコンパイルできなくなることがあります。その場合は、エラーになった部分だけ以前の書き方に戻してください。

▼リスト41　getter/setterを自前で実装

```
public class A {
    private String a;
    public void setA(String a) {this.a = a; }
    public String getA() {return a; }
}
```

▼リスト42　@Getter/@Setterを適用

```
import lombok.*;
public class A {
    @Getter @Setter private String a;
}
```

Stream APIと組み合わせて活用!

業務アプリケーションにも使えるJava 8のラムダ式

Java 8でもっとも大きな追加点「ラムダ式」について、文法から具体的な実装方法までを説明します。また、ラムダ式を使ってシンプルな記述でコレクションを操作できる「Stream API」も紹介します。これにより、業務処理の内容が明確なコードが書けるようになります。

●アリエル・ネットワーク㈱　池添 明宏(いけぞえ あきひろ)

業務アプリケーションにおけるJava 8

2014年3月にJavaの新しいメジャーバージョンであるJava 8がリリースされました。Java 8では、ラムダ式を始めとする新しい文法、Stream APIや新しいDate and Time APIなど大きな機能が追加されました。これらの機能をうまく活用すれば、開発効率を向上させたり、メンテナンス性の高いソフトウェアを開発したりすることが可能となるでしょう。

とは言うものの、業務アプリケーションでJava 8が使えるのなんてまだまだ先だ、なんて考えている方も多いのではないでしょうか。しかし、JavaのEnd of Life(EOL)ポリシーでは、Javaは後続のメジャーリリースから1年後に公式アップデートが終了します。つまり、Java 7は2015年4月にはサポート期間が終了してしまいます(**表1**)。EOLを迎えたバージョンのJavaは、不具合やセキュリティホールが見つかってもアップデートが行われません。有償でサポートの期間延長もできますが、多くの場合はJava 8への移行を検討せざるを得ないでしょう。

しかし、既存のアプリケーションをJava 8にアップデートする際にはコストがかかります。Javaは後方互換性を重要視しているため、新しい文法やAPIが追加された際にもできるだけ既存のプログラムが動作するように考慮されています。それでも、数多くの仕様変更が行われているため、よほど規模の小さいアプリケーションでもない限り、問題なく動作することはまれでしょう。また、利用しているツールやフレームワーク、ライブラリがJava 8に対応していなければアップデートは困難になります。もちろん、アップデート後にテストを行うコストも必要になります。

これらのアップデートコスト、セキュリティリスク、生産性・メンテナンス性・パフォーマンスの向上、各種ツールやライブラリの対応状況、業務アプリケーションのライフサイクルなど、広く長期的な視点でアップデート計画を立てることが重要です。

本記事では、Java 8を導入することでどのようなメリットがあるのかを知ってもらうために、ラムダ式を中心にJava 8で導入された新機能を紹介します。

▼表1　Javaのサポート期間

バージョン	リリース	サポートの終了
Java 6	2006年12月	2013年2月
Java 7	2011年7月	2015年4月予定
Java 8	2014年3月	2017年3月予定

ラムダ式

まずは、Java 8の目玉機能「ラムダ式」について解説します。ラムダ式とは、**式中に名前のない関数を定義するための記法**のことです。モダンなプログラミング言語の多くは、ラムダ式かそれに準ずるような記法を持っています。Javaへのラムダ式の導入を待ち望んでいた人も多いことでしょう。一方でJava以外のプログラミ

ング言語に馴染みのない方は、特殊な記法にびっくりしてしまうかもしれません。しかし、文法を正しく理解していればそれほど難解な機能ではありません。

「式中に関数を定義する記法」とはどういうことなのでしょうか？　さっそく簡単なラムダ式の例(**リスト1**)を見てみましょう。

リスト1中に2ヵ所ある**x -> x * 2**の部分がラムダ式です。ラムダ式は、引数のリストと関数の処理内容をアロー演算子「**->**」でつないで記述します。このラムダ式では、Integer型のxという引数を受け取り、それを2倍して返すという関数を定義しています。(1)ではtwiceという変数にラムダ式を代入しています。そして(2)ではmapメソッドの引数にラムダ式を渡しています。

これまでJavaにおいて関数を定義するためには、必ずなんらかのクラスに属したメソッドとして定義する必要がありました。しかし、ラムダ式を利用すると、関数を変数に代入できたり、メソッドの引数に関数を渡すことができたりします。

この特徴がどのようなメリットをもたらすのか、以降で説明していきます。

▼リスト1　ラムダ式のサンプル

```
import java.util.function.Function;
import java.util.stream.IntStream;

public class LambdaSample {
  public static void main(String[] args) {
    // (1) twiceという変数にラムダ式を代入
    Function<Integer, Integer> twice = x -> x * 2;

    // (2) mapメソッドの引数にラムダ式を渡す
    IntStream.range(0, 10).map(x -> x * 2);
  }
}
```

▼リスト2　匿名クラスを利用したコールバック

```
Label label1 = new Label();
Button button1 = new Button("push");
button1.setOnAction(new EventHandler<ActionEvent>() {
  @Override
  public void handle(ActionEvent e) {
    label1.setText("Hello, World!");
  }
});
```

▼リスト3　ラムダ式を利用したコールバック

```
Label label1 = new Label();
Button button1 = new Button("push");
button1.setOnAction(e -> label1.setText("Hello, World!"));
```

ラムダ式の使いみち

Javaでプログラミングをする際には、再利用するための単位としてメソッドを定義することが多いでしょう。しかし、ラムダ式の記法で関数を定義するケースでは、再利用するための関数を定義するのではなく、**その場限りの使い捨ての関数として利用する**ケースが多くなります。

その場限りの関数を適用できるシーンは数多くありますが、代表的なケースとしては次のようなものがあります。

- コールバック処理
- 並列処理
- 処理の移譲
- 述語の指定

この中のコールバック処理を例に挙げて説明したいと思います。コールバック処理は、AndroidアプリケーションやJavaFXなどのGUIプログラミングで多用されます。

たとえば、ボタンを押すとラベルの内容を変更する処理を記述したい場合、匿名クラス(無名クラス)を利用すると**リスト2**のように記述できます。

一方でラムダ式を利用すると、**リスト3**のように記述できます。匿名クラスのインスタンス化やメソッドのオーバーライドの記述がなくなり、簡潔になっていることがわかります。

なお、匿名クラスの代わりにラムダ式を利用すると、記法がシンプルになるだけではなく、オブジェクトの生成コストが少ないというメリットもあります。ラムダ式が導入された現在、匿名クラスを利用しなければならないケースはほとんどないと言えるでしょう。

基本文法

ラムダ式の基本的な文法について解説していきます。ラムダ式ではいくつかの省略記法が用意されているので、どの部分が省略されているのか注意して見ていきましょう。

まずは、もっとも冗長なラムダ式の記述方法を紹介します。たとえば、2つの整数xとyを引数に取り、それらの和を返すラムダ式は次のように記述できます。

```
(int x, int y) -> {
  return x + y;
}
```

ラムダ式の処理が1行で完結する場合は、次のように中括弧とreturn文を省略できます。

```
(int x, int y) -> x + y
```

次のように引数の型を省略することもできます。コンパイラがラムダ式の適用個所に応じて引数の型を推論してくれます。

```
(x, y) -> x + y
```

さらに、引数が1つしかない場合は、引数リストを囲む括弧を省略できます。

```
x -> x * x
```

ただし、引数が1つもない場合は、引数リストを囲む括弧を省略することはできません。

```
() -> 123
```

関数型インターフェース

これまでの説明で、ラムダ式を利用すると関数が定義できると説明しました。しかし、JVMではどのクラスにも所属しない関数を定義することはできません。そのためラムダ式も、内部的には「あるインターフェース」を実装したクラスのインスタンスなのです。

この「あるインターフェース」のことを関数型インターフェースと呼びます。それでは、2つの整数値を引数に取り、戻り値として整数値を返すラムダ式を表現するための、関数型インターフェースを用意してみましょう(**リスト4**)。

関数型インターフェースは、このように実装すべきメソッドを1つだけ持ったインターフェースになります。メソッド名は何でもかまいませんが、ラムダ式と同じ引数と戻り値を持っている必要があります。また、@FunctionalInterfaceアノテーションを付与すると、コンパイル時にメソッドが1つだけかどうかをチェックしてくれます。@FunctionalInterfaceを省略することはできますが、ラムダ式に利用する場合はできるだけ付与しておくべきです。

この関数型インターフェースを利用すると、ラムダ式を変数に代入できます。**リスト5**の例ではxとyという2つの整数値を引数に取りその和を返すラムダ式を定義し、funcという変数に代入しています。

なお、変数に代入しただけではラムダ式の中の処理は実行されません。ラムダ式を実行するには、関数型インターフェースに用意されているメソッドを呼び出します。たとえば、**リスト6**のように引数に1と2を渡してapplyメソッドを呼び出すと、ラムダ式が実行されて1と2の和である3がzに代入されます。

さて、**リスト4**の例では関数型インターフェースを自作しましたが、Javaの標準ライブラリには数十種類の関数型インターフェースが用意されています。代表的なものとして**表2**のようなものが挙げられます。このほかにも、引数の数が2つになったBiFunctionやBiConsumer、プリミティブ型が利用できるIntPredicateやDoubleSupplierなどが用意されています。

これらのインターフェースは、おもにjava.util.functionパッケージに属しています。ラムダ式を利用した機能を実装したい場合には、ま

▼リスト4　関数型インターフェースの定義

```
@FunctionalInterface
interface BiIntFunction {
  int apply(int x, int y);
}
```

▼リスト5　ラムダ式の変数への代入

```
BiIntFunction func = (x, y) -> x + y;
```

▼リスト6　ラムダ式の処理の実行

```
int z = func.apply(1, 2);
System.out.println(z);
```

▼表2　標準で用意されている関数型インターフェース

インターフェース名	引数	戻り値	メソッド
Runnable	なし	なし	run
Consumer<T>	T型	なし	accept
Function<T, R>	T型	R型	apply
Predicate<T>	T型	boolean型	test
Supplier<T>	なし	T型	get

▼図1　java.util.List<E>インターフェースに追加されたメソッド

```
・boolean removeIf(Predicate<? super E> filter)
  ラムダ式で指定した条件に一致する要素をリストから削除する
・void replaceAll(UnaryOperator<E> operator)
  リストの全要素を、ラムダ式で指定した処理で変換する
・void sort(Comparator<? super E> c)
  ラムダ式で指定した条件でリストの要素を並び替える
・void forEach(Consumer<? super E> action)
  リストの全要素に対して、ラムダ式で指定した処理を適用する
```

ず標準で用意されている関数型インターフェースが利用できないかどうかを調べ、それが見つからないのであれば、関数型インターフェースを自作すると良いでしょう。

コレクションクラスの新しいメソッド

ここからは、いよいよラムダ式の活用方法を紹介していきます。

本特集の第1章でコレクションクラスについて紹介しました。Java 8では、標準で用意されているコレクションクラスに新しいメソッドがいくつか追加されました。本節では、追加されたメソッドのうち、ラムダ式を活用したものを紹介していきます。

なお、本記事では紹介しませんが、Java 8で導入されたインターフェースのデフォルト実装という機能により、インターフェースにデフォルトの実装を持たせることができるようになりました。このしくみにより、互換性を壊すことなく既存のインターフェースを拡張することができます。そのため、Java 8未満の既存アプリケーションにおいて、Listインターフェースや Mapインターフェースを実装した独自クラスを作っていたとしても、Java 8にアップデートしたときに新しく追加されたメソッドを実装する必要はありません。

▼リスト7　List<E>に追加されたメソッドの利用例

```
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

public class ListSample {
  public static void main(String[] args) {
    List<Integer> list = new ArrayList<>(Arrays.asList(5, 4, 2, 10, 7, 3, 9, 8, 6, 1));
    // (1) 3で割り切れる要素は取り除く
    list.removeIf(x -> x % 3 == 0);
    // (2) すべての要素を2倍する
    list.replaceAll(x -> x * 2);
    // (3) リストの内容を降順で並び替える
    list.sort((v1, v2) -> v2 - v1);
    // (4) リストの内容をすべて表示する
    list.forEach(x -> System.out.println(x));
  }
}
```

List<E>インターフェースに追加されたメソッド

まずはjava.util.List<E>インターフェースに追加されたメソッドを見てみましょう(**図1**)。

これらのメソッドは**リスト7**のように利用できます。(1)では、コレクションの中から3で割り切れる要素をすべて削除しています。(2)では、コレクションの要素を2倍しています。(3)では、ラムダ式で指定した比較関数を基にコレクションの内容を並び替えています。そして(4)では、コレクションの内容をすべて画面上に表示しています。実行結果は、次のようになります。

```
20
16
14
10
8
4
2
```

Map<K, V>インターフェースに追加されたメソッド

次に、java.util.Map<K, V>インターフェースに追加されたメソッドを見てみましょう(**図2**)。

まずはreplaceAllとforEachの使い方を紹介します(**リスト8**)。

replaceAllでは、マップのすべての要素のキーと値がラムダ式の引数として渡ってくるので、置き換えた値を戻り値として返します。(5)の例ではすべての要素の値を1.1倍しています。

▼図2　java.util.Map<K, V>インターフェースに追加されたメソッド

```
・void forEach(BiConsumer<? super K, ? super B> action)
  マップの全要素に対して、ラムダ式で指定した処理を適用する
・void replaceAll(BiFunction<? super K, ? super V, ? extends V> function)
  マップの全要素を、ラムダ式で指定した処理で変換する
・V compute(K key, BiFunction<? super K, ? super V, ? extends V> remappingFunction)
  指定したキーの要素を、ラムダ式で指定した処理で追加／上書き／削除を行う
・V computeIfAbsent(K key, Function<? super K, ? extends V> mappingFunction)
  指定したキーの要素が存在しない場合、ラムダ式で指定した処理で追加を行う
・V computeIfPresent(K key, BiFunction<? super K, ? super V, ? extends V> remappingFunction)
  指定したキーの要素が存在した場合、ラムダ式で指定した処理で上書きを行う
・V merge(K key, V value, BiFunction<? super V, ? super V, ? extends V> remappingFunction)
  指定したキーが存在しない場合は第2引数の値を登録、キーが存在する場合は登録されている値と第2引数の値をラムダ式でマージして登録する
```

続いて(6)のforEachでは、マップのすべての要素のキーと値がラムダ式の引数として渡ってくるので、それを画面上に表示しています。実行結果は次のようになります。

```
りんごは132円です。
みかんは99円です。
ばななは66円です。
```

つぎにcompute、computeIfAbsent、computeIfPresent、mergeを見てみましょう(**リスト9**)。

(7)のcomputeIfAbsentでは、第1引数で指定したキーがマップ内に存在しない場合に、第2引数で指定したラムダ式の戻り値がマップに登録されます。(8)のcomputeIfPresentでは、第1引数で指定したキーがマップ内に存在した場合に、第2引数で指定したラムダ式にキーと値が渡ってくるので、値を変更できます。(9)のcomputeは、第1引数で指定したキーがマップに存在してもしなくても、第2引数で指定したラムダ式の戻り値がマップに登録されます。ただし、キーが存在しない場合にはラムダ式には値がnullとして渡ってきます。(10)のmergeでは、第1引数で指定したキーが存在しない場合には、第2引数で指定した値をマップに登録します。キーが存在した場合は、ラムダ式に第2引数で指定した値と現在登録されている値が渡ってくるので、それらの値を利用して新しい値を返します。実行結果は次のようになります。

```
りんごは140円です。
ぶどうは200円です。
みかんは100円です。
ばななは30円です。
```

ここで紹介したようなListとMapに追加された新しいメソッドの機能は、いずれも既存のfor文やif文を利用して記述できます。しかし、ラムダ式を活用することでより簡潔に記述でき

▼リスト8　Map<K, V>インターフェースに追加されたメソッドの利用例(replaceAllとforEach)

```
Map<String, Integer> fruits = new HashMap<>();
fruits.put("りんご", 120);
fruits.put("みかん", 90);
fruits.put("ばなな", 60);
// (5) 果物の値段をすべて10%アップする
fruits.replaceAll((k, v) -> (int) Math.round(v * 1.1));
// (6) マップの内容をすべて表示
fruits.forEach((k, v) -> System.out.printf(
  "%sは%d円です。¥n", k, v));
```

▼リスト9　Map<K, V>インターフェースに追加されたメソッドの利用例(computeとmergeなど)

```
Map<String, Integer> fruits = new HashMap<>();
fruits.put("りんご", 120);
fruits.put("みかん", 90);
fruits.put("ばなな", 60);
// (7) ぶどうが存在しなければ200円で追加する
fruits.computeIfAbsent("ぶどう", k -> 200);
// (8) ばななが存在したら値段を半額にする
fruits.computeIfPresent("ばなな", (k, v) -> v / 2);
// (9) りんごが存在したら20円値上げ、存在しなければ130円で追加する
fruits.compute("りんご", (k, v) -> {
  if (v != null) {
    return v + 20;
  } else {
    return 130;
  }
});
// (10) みかんが存在しなければ100円で追加する。存在した場合は現在の値と100円の大きいほうの値に設定する
fruits.merge("みかん", 100, (v1, v2) -> Math.max(v1, v2));
fruits.forEach((k, v) -> System.out.printf(
  "%sは%d円です。¥n", k, v));
```

るようになっています。

Stream API

前節でコレクションクラスに追加された新しいメソッドを紹介しました。これらの新しいメソッドを利用することで、ちょっとした処理が簡潔に記述できるようになりました。しかし、実際の業務アプリケーションは単純な処理ばかりではなく、もっと複雑なロジックが必要になるでしょう。

Java 8では、Stream APIというラムダ式を活用したコレクション操作用のAPIが追加されました。業務アプリケーションでは、for文やif文を利用してコレクションの内容を絞り込んだり、加工したり、グルーピングや集計を行ったりすることが多いと思います。Stream APIではfor文やif文を利用せずに、よりシンプルな記述でコレクションの操作を行うことができます。Stream APIはfor文やif文を利用したコードと比較して次のような特徴があります。

- 宣言的に記述できるため、コードが簡潔で可読性が高くなる
- フィルタリングやグルーピングや集計など、標準で用意されている汎用的な機能を利用できる
- Stream APIを拡張することで、さまざまなデータ型に対して汎用的な処理を用意できる

▼リスト10　for文による記述例

```
List<String> names = new ArrayList<>();
for (Employee employee : employees) {
  // 条件による抽出
  if (employee.getDept().equals("開発部")) {
    // データの加工
    String name = employee.getName() + employee.getRole();
    // コンテナへの登録
    names.add(name);
  }
}
```

▼リスト11　Stream APIによる記述例

```
List<String> names = employees.stream()
    .filter(e -> e.getDept().equals("開発部")) // 条件による抽出
    .map(e -> e.getName() + e.getRole()) // データの加工
    .collect(Collectors.toList()); // コンテナへの登録
```

本記事では3つめのStream APIの拡張については紹介しませんが、コードの簡潔さや標準で用意されている機能について順に紹介していきます。

なおJavaにはFileInputStreamやFileOutputStreamなどファイル入出力を扱うStreamクラスも存在しますが、Stream APIとは無関係です。これらの言葉を混同しないように注意しましょう。

for文との比較

まずは、Stream APIを利用するとこれまでのfor文を利用したコードとどのような違いが出るのかを紹介したいと思います。

たとえば、コレクションに格納されているデータを何らかの条件で抽出し、そのデータを加工し別のコレクションとして取得するというような処理は、一般的なコーディングでは多く登場するかと思います。そのような処理をfor文を利用して記述すると**リスト10**のようになります。

同じ処理をStream APIを利用して書き換えてみましょう(**リスト11**)。

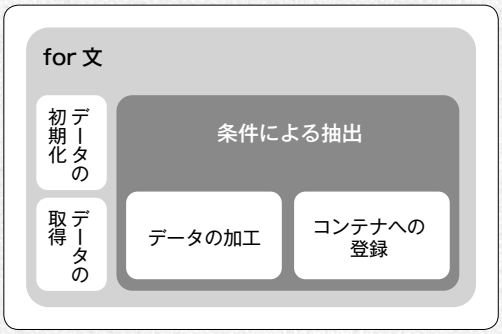
Stream APIの詳細については後述しますが、filterメソッドでは、コレクションの要素をラムダ式で指定した条件で絞り込み、mapメソッドではコレクションの要素をラムダ式で指定した方法で加工しています。そして、最後のcollectメソッドで要素をコンテナに登録しています。

さて、この2つのコードを比較した場合、可読性はどちらが高いでしょうか? もちろん、for文に慣れていてStream APIを知らない人にとってはfor文のほうがわかりやすいでしょう。しかし、filterメソッドは条件抽出であり、mapメソッドは加工処理であるということを知っていれば、何をやりたいのかが明確にわかるコードと言えるのではないでしょうか。

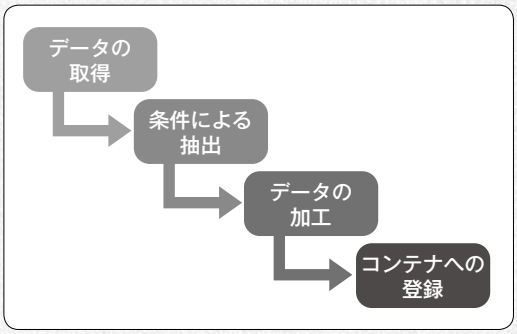
続いて、2つのコードの構造の違いに着目してみましょう。

まずfor文を利用したコードの構造

▼図3　for文を利用した場合のロジックの構造



▼図4　Stream APIの場合のロジックの構造



を図示すると**図3**のようになります。for文のブロックの中に初期化や条件抽出の処理が含まれており、さらに条件抽出のブロック中に加工処理などが含まれていることがわかります。

一方でStream APIによるコードの構造を図示すると**図4**のようになります。条件による抽出、データの加工、コンテナへの登録のブロックが分離されており、データがパイプライン的に流れていく構造になっているのがわかります。

このような構造の違いにどのような意味があるのでしょうか？　たとえば、これらのコードにコレクションのソート処理を追加したい場合や、処理の順番を入れ替えたい場合を考えてみましょう。for文によるソースコードの場合は、入り組んだ構造の中に処理を追加する必要があるでしょう。一方でStream APIであれば、filterやmapのメソッドの並びに新しいメソッドを追加したり順番を入れ替えたりするだけです。

このようにメンテナンス性の高さもStream APIの大きな特徴の1つです。

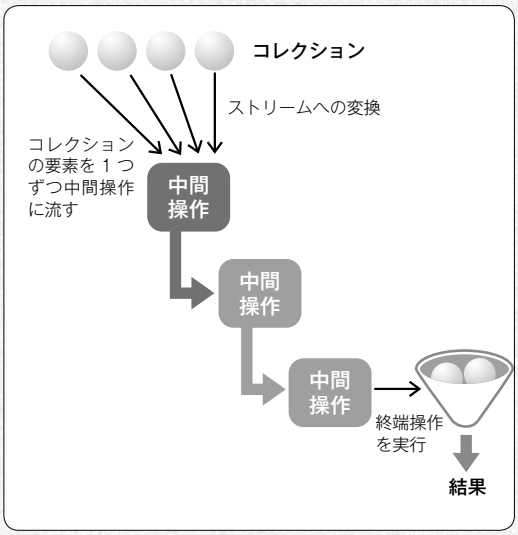
ストリームパイプライン

Stream APIを理解するためには、3種類の操作について知らなければなりません。

・ストリームへの変換
・中間操作
・終端操作

この、ストリームの変換から始まり、中間操作を経て、終端操作で結果を取得する流れをストリームパイプラインと呼びます。ストリームパイプラインを図示すると**図5**のようになります。

▼図5　ストリームパイプライン



まず既存のコレクションデータをストリームに変換し、Stream APIで提供されているさまざまな機能を利用できるようにします。次の中間操作では、コレクションを条件で絞り込んだり、データを加工したり、具体的なコレクション操作処理を行います。なお中間操作は、1つのストリームパイプラインの中で複数回呼び出すことができます。そして、最後の終端操作で、具体的なコレクションや単一の値として結果を取得します。

なお、前節の**リスト7**のremoveIfやreplaceAllなどのメソッドでは、コレクションの要素そのものを変更していましたが、Stream APIではコレクションの要素そのものは変更せず、

変更を加えた新しいコレクションや値を終端操作で返す仕様となっています。巨大なコレクションを扱う場合には、メモリの使用量にも気を配る必要があるでしょう。

それでは、これらの3つの操作をそれぞれ詳しく見ていきましょう。

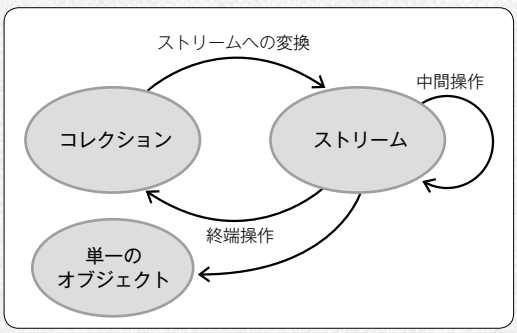
ストリームへの変換

Stream APIによるコレクションの操作を行うためには、まず既存のコレクションをストリームに変換する必要があります。ストリームを表現するためのクラスには、参照型を扱えるStream<T>と、プリミティブ型を扱うためのIntStream、LongStream、DoubleStreamなどがあります。

コレクションとストリームの状態を状態遷移図としてとらえると、**図6**のように図示できます。

コレクションの状態からストリームに変換することによって、条件抽出やデータの加工などの中間操作が行えるようになります。中間操作は何度実行してもストリームの状態から変化しません。そしてストリームの状態から終端操作を行うことによって、コレクションや単一の値を結果として得ることができます。なお、終端操作の結果としてコレクションを得た場合は、再度ストリームに変換することもできます。Stream APIを利用する際には、現在の状態がストリームなのかそうでないのかを常に意識しておくと良いでしょう。また、いったん終端操作を実行したストリームに対して、再度終端操作を実行することはできないので注意しましょう。

▼図6　ストリームの状態



▼リスト12　ストリームの生成(Stream<T>)

```
List<Integer> list = Arrays.asList(5, 4, 2, 10, 7,
                                   3, 9, 8, 6, 1);
// List<T>からストリームに変換
Stream<Integer> stream1 = list.stream();
// 数値の羅列からストリームを生成
Stream<Integer> stream2 = Stream.of(1, 2, 3, 4, 5);
```

▼リスト13　ストリームの生成(IntStream)

```
// 配列からストリームに変換
IntStream stream3 = Arrays.stream(new int[]{1, 2, 3});
// 数値の範囲を指定してストリームを生成
IntStream stream4 = IntStream.range(0, 100);
```

コレクションからストリームに変換する方法はいくつかあるので、順に見ていきましょう。

List.streamメソッドを利用すると、List<T>クラスのインスタンスからストリームに変換できます。また、Stream.ofを利用すると引数で列挙した値を基に、ストリームを生成できます(**リスト12**)。

Arrays.streamを利用すると配列からストリームに変換できます。また、IntStream.rangeを利用すると指定した範囲の数値のストリームを得ることができます(**リスト13**)。

このほかにもファイルから読み込んだデータをストリームに変換するAPIも用意されています。

中間操作

Stream APIでは、コレクションとストリームの2つの状態を行き来すると説明しました。コレクションの状態は、これまでのプログラミングで扱ってきた状態なのでとくに問題はないでしょう。では、ストリームの状態とはどのようなものでしょうか?

たとえば、**リスト14**のような中間操作を実行したストリームのインスンタンスintStreamについて考えてみましょう。この例は、数値のコレクションのうち、2で割り切れるものをfilterメソッドで抽出するという処理になります。

このコードでは終端操作を行っていないため、intStreamを取得した時点ではまだフィルタリング処理は行われていません。すなわち、フィルタリング処理の予約をしたような状態だと言えます。このintStreamに対して終端操作を実行したとき

に初めて、フィルタリング処理が実行されます。Stream APIを使ううえでは、この動きを理解していないと、思わぬ落とし穴にハマってしまうことでしょう。このように中間操作が遅延されるしくみになっているため、for文で記述したコードと比較してもパフォーマンスが悪化しにくくなっています。

中間操作には、たくさんのメソッドが用意されています。代表的な次の4つを紹介します。

・filter
・map
・distinct
・peek

filterメソッドはこれまでに何度か登場しましたが、ラムダ式で指定した条件に基づいてコレクションのデータを絞り込むメソッドです。**リスト15**のコードでは1つめのfilterで値が5以下の値に絞り込み、2つめのfilterで2で割り切れる値に絞り込んでいます。なお、結果を表示するためにforEachという終端操作を利用しています。実行結果は次のようになり、filterの条件で正しく絞りこまれていることがわかります。

```
2
4
```

mapメソッドはコレクションのデータを加工するメソッドです。**リスト16**の例では、数値のコレクションをすべて2乗しています。

実行結果は次のようになります。

```
1
4
9
16
25
```

コレクションの中から重複した値を取り除きたい場合はdistinctメソッドを利用します(**リスト17**)。実行結果は次のようになり、重複した要素が取り除かれていることがわかります。

```
1
2
3
4
6
```

Stream APIは、for文を利用したプログラミングに比べるとデバッグがしにくいと感じることがあります。peekメソッドは、ストリームパイプラインを流れているデータをコンソールに表示するなど、デバッグ用途に利用できます。

たとえばfilterメソッドで正しい結果が得られない場合、**リスト18**のようにfilterの前後にpeekメソッドを差し込むことでフィルタリングが成功しているかどうかを確認できます。

実行結果は次のようになり、filter前には1と3と5が表示されていたものが、filter後には1のみが表示されていることがわかります。

▼リスト14　中間操作

```
List<Integer> list = Arrays.asList(5, 4, 2, 10, 7, 3, 9, 8, 6, 1);
Stream<Integer> intStream = list.stream().filter(x -> x % 2 == 0);
```

▼リスト15　filterの利用例

```
List<Integer> list = Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
list.stream()
    .filter(x -> x < 5)
    .filter(x -> x % 2 == 0)
    .forEach(x -> System.out.println(x));
```

▼リスト16　mapの利用例

```
List<Integer> list = Arrays.asList(1, 2, 3, 4, 5);
list.stream()
    .map(x -> x * x)
    .forEach(x -> System.out.println(x));
```

▼リスト17　distinctの利用例

```
List<Integer> list = Arrays.asList(1, 1, 2, 3, 3, 3, 4, 4, 6);
list.stream()
    .distinct()
    .forEach(x -> System.out.println(x));
```

▼リスト18　peekの利用例

```
List<Integer> list = Arrays.asList(1, 3, 5);
list.stream()
    .peek(x -> System.out.printf("before filter: %d¥n", x))
    .filter(x -> x < 3)
    .peek(x -> System.out.printf("after filter: %d¥n", x))
    .collect(Collectors.toList());
```

```
before filter: 1
after filter: 1
before filter: 3
before filter: 5
```

終端操作

最後に終端操作について解説します。終端操作は、中間操作で予約した処理をすべて実行し、最終的な結果としてコレクションや単一の値を取得するための操作です。

中間操作と同じく終端操作にも数多くのメソッドが用意されていますが、代表的な次のメソッドについて紹介します。

- forEach
- collect
- allMatch、anyMatch

なお以降のサンプルでは、**リスト19**、**リスト20**のような社員情報を表すEmployeeクラスと、そのコレクションを利用します。

最初の終端操作としてforEachを紹介します。

▼リスト19　Employeeクラス

```
public class Employee {
  private String name;
  private String role;
  private String dept;
  private Integer age;
  public Employee(String name, String role,
                  String dept, Integer age) {
    this.name = name;
    this.role = role;
    this.dept = dept;
    this.age = age;
  }
  //getter/setterは省略
}
```

▼リスト20　サンプルコード用の社員データ

```
List<Employee> employees = new ArrayList<>();
employees.add(new Employee(
  "スミス", "部長", "開発部", 54));
employees.add(new Employee(
  "ジョンソン", "平社員", "営業部", 29));
employees.add(new Employee(
  "ウィリアムズ", "課長", "開発部", 41));
employees.add(new Employee(
  "ブラウン", "主任", "開発部", 34));
employees.add(new Employee(
  "ジョーンズ", "部長", "人事部", 52));
employees.add(new Employee(
  "ミラー", "平社員", "マーケティング部", 24));
```

forEachはコレクションの要素1つずつに対して順番にラムダ式で指定した処理を適用します。**リスト21**の例では、役職が部長である社員の名前を順にコンソールに表示しています。

実行結果は次のようになります。

```
スミス
ジョーンズ
```

続いてcollectメソッドについて解説します。collectメソッドは汎用的な終端操作で、引数にCollectorオブジェクトを指定することでさまざまな結果を得ることができます。Collectorオブジェクトの取得方法はいくつかありますが、ここでは標準で用意されているCollectorsクラスから取得できる、代表的な次の4つを紹介します。

- Collectors.toList
- Collectors.joining
- Collectors.groupingBy
- Collectors.summarizing

まず、collectメソッドにCollectors.toListを指定するとストリームをList<T>のインスタンスに変換できます。Listインターフェースのstreamメソッドと対になるしくみだと理解しましょう(**リスト22**)。

Collectors.joiningByを指定すると、引数で指定した区切り文字を利用して連結した文字列を返します(**リスト23**)。

▼リスト21　forEachの利用例

```
employees.stream()
    .filter(e -> e.getRole().equals("部長"))
    .forEach(e -> System.out.println(e.getName()));
```

▼リスト22　collectの利用例(Collectors.toList)

```
// 結果をListとして取得する
List<String> names = employees.stream()
    .map(e -> e.getName())
    .collect(Collectors.toList());
```

▼リスト23　collectの利用例(Collectors.joining)

```
// 結果をカンマで結合する
String joinedName = employees.stream()
    .map(e -> e.getName())
    .collect(Collectors.joining(","));
System.out.println(joinedName);
```

実行結果は次のようになり、社員名がカンマで連結されていることがわかります。

```
スミス,ジョンソン,ウィリアムズ,ブラウン,ジョーンズ,ミラー
```

Collectors.groupingByを指定すると、ラムダ式で指定したキーでグルーピングしたMapのインスタンスを取得できます(**リスト24**)。実行結果は次のようになり、部署ごとにグルーピングされていることがわかります。

```
マーケティング部
 - ミラー
人事部
 - ジョーンズ
開発部
 - スミス
 - ウィリアムズ
 - ブラウン
営業部
 - ジョンソン
```

Collectors.summarizingを指定すると、最小値、最大値、平均値などの集計結果をまとめて取得できます(**リスト25**)。実行結果は次のようになります。

```
社員数: 6
最低年齢: 24
最高年齢: 54
合計年齢: 234
平均年齢: 39.000000
```

最後にanyMatchとallMatchを紹介します(**リスト26**)。anyMatchはコレクションの中に1つでもラムダ式で指定した条件に当てはまるものがあればtrueを返します。一方allMatchはコレクションの中のすべてがラムダ式で指定した条件に当てはまればtrueを返します。

実行するとどちらの条件もtrueとなるため、次のように表示されます。

```
社員の中に営業部の人が少なくとも1人います
社員は全員22歳以上です
```

▼リスト24　collectの利用例(Collectors.groupingBy)

```
// 部署ごとにグルーピングする
Map<String, List<Employee>> deptMap = employees.stream()
    .collect(Collectors.groupingBy(e -> e.getDept()));
deptMap.forEach((dept, es) -> {
  System.out.println(dept);
  es.forEach(e -> System.out.printf(" - %s¥n", e.getName()));
});
```

▼リスト25　collectの利用例(Collectors.summarizing)

```
// 集計結果を取得する
IntSummaryStatistics summary =
    employees.stream().collect(Collectors.summarizingInt(
                                e -> e.getAge()));
System.out.printf("社員数: %d¥n", summary.getCount());
System.out.printf("最低年齢: %d¥n", summary.getMin());
System.out.printf("最高年齢: %d¥n", summary.getMax());
System.out.printf("合計年齢: %d¥n", summary.getSum());
System.out.printf("平均年齢: %f¥n", summary.getAverage());
```

▼リスト26　anyMatch/allMatchの利用例

```
if (employees.stream().anyMatch(e -> e.getDept().equals("営業部"))) {
  System.out.println("社員の中に営業部の人が少なくとも1人います");
}
if (employees.stream().allMatch(e -> e.getAge() > 22)) {
  System.out.println("社員は全員22歳以上です");
}
```

最後に

本章ではJava 8の新機能であるラムダ式と、ラムダ式の活用例として、コレクションクラスの新しいメソッドとStream APIについて解説しました。これらの機能を利用することで、これまでと比べて効率的にプログラミングできることが、少しでも伝わったでしょうか。なお、ここで紹介した内容は基本的なものだけで、ラムダ式の活用方法やStream APIの機能はまだまだたくさんあります。また、Java 8にはOptionalや新しいDate and Time APIなど便利な新機能がたくさんあります。Java 8に興味を持った方はぜひほかの新機能についても調べてみてください。本記事が少しでもJava 8導入の一助となれば幸いです。SD

第3章

第1特集　言語仕様・開発環境・デバッグ機能 あなたはどこまで使いこなせてる？　今ふたたびのJava

自分に合ったIDEを見つけよう

Eclipseだけじゃない! 今どきの統合開発環境

本章では、Javaの統合開発環境として最近注目されつつある「IntelliJ IDEA」、「NetBeans」を紹介します。エディタとしての基本機能、デバッガ、バージョン管理システムとの連携について、Eclipseも合わせた3つの環境の比較を行います。

●日本ユニシス㈱　今井 勝信（いまい まさのぶ）

はじめに

Javaの統合開発環境（以下、IDE）として圧倒的な知名度を誇るEclipseの影に隠れた2つのIDE——IntelliJ IDEA（以下、IDEA）とNetBeansについてスポットを当てます。

中には「EclipseのほかにもJavaのIDEなんてあったの？」と思う方もいると思いますが、実はIDEAやNetBeansもEclipseとほぼ同時期に登場したIDEで、どちらも十分な歴史を持っています（**表1**）。最近になってIDEAやNetBeansも多少目立つようになってきましたが、それは新参だからではなく、長年切磋琢磨してきた結果が芽を出してきたからだと思います。どのIDEにも、そのIDEを印象づける特徴がありますが、同時にそれぞれが培ってきたしがらみも併せ持ちます。

▼表1　Javaの主なIDE

IDE	初回リリース時期	最新版（リリース日）
Eclipse	2001年	4.4（2014年6月）
IDEA	2001年	13.1.4（2014年7月）
NetBeans	1999年	8.0（2014年3月）

IntelliJ IDEA

チェコにあるJetBrains社が開発しているIDEで「いんてりじぇい　あいであ」と読みます[注1、注2]。手の行き届いたリファクタリングやコード補完に定評があり、海外では多くのファンを持ちます。もともとはJava専用のIDEだったのですが、多数の言語をサポートしており、IDEAをベースとしたJava以外の言語用IDEが派生しています。

オープンソースソフトウェア（以下、OSS）として無償公開しているCommunity Edition（以下、CE）とScalaプラグインのおかげで、国内ではScala用IDEとして注目されるようになりました。近年ではGoogleが発表したAndroid StudioがIDEAベースであることで、さらに注目を集めています。とても高機能ですが、その分初心者に優しくありません。また英語版のみで日本語化はされていません。

本章はIDEA CE（v13.1.4）を基準に機能を紹介していきます[注3]。

NetBeans

もともとはチェコの学生たちが開発したIDEでしたが、紆余曲折を経てOracle社を中心としたコミュニティによって提供されています[注4]。EclipseやIDEA CEと同じくOSSで無償公開されています。

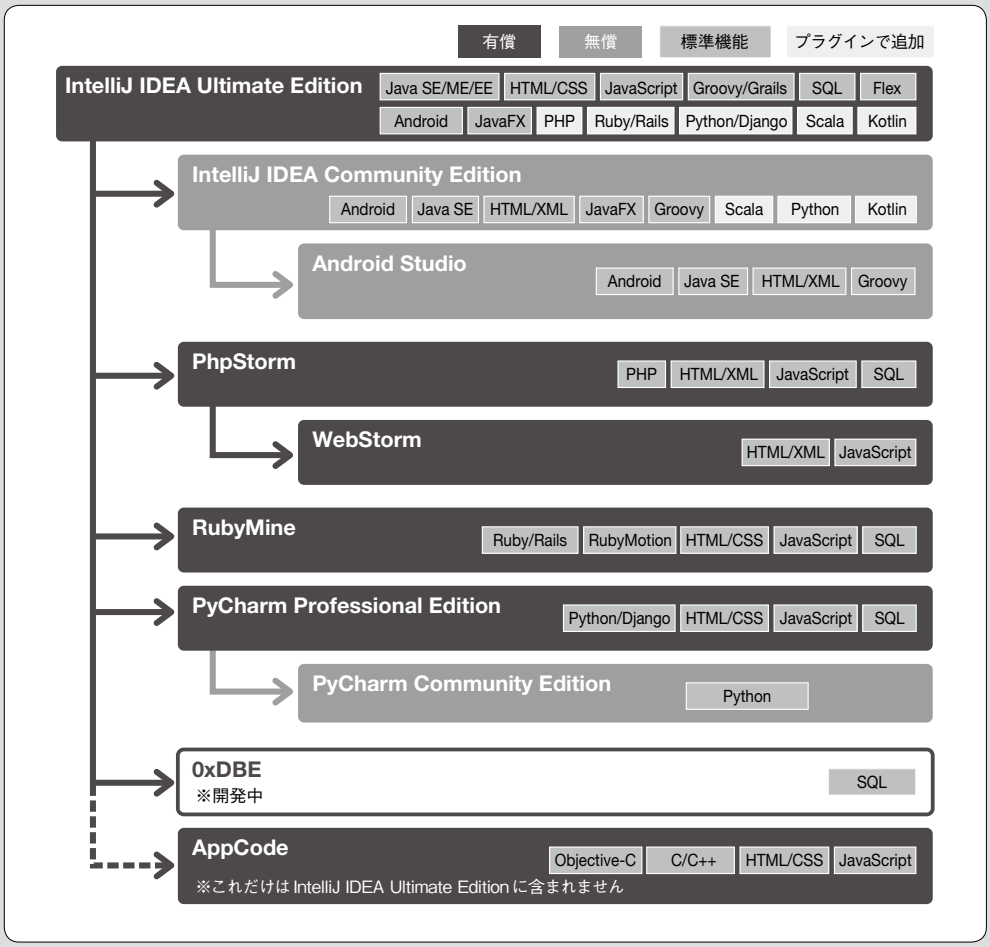
開発にOracle社が関与しているため、Javaの

注1）URL http://www.jetbrains.com/idea
注2）「いであ」と言いたくなる気持ちはわかります。正しい省略表記は「IntelliJ」ではなく「IDEA」のようです。
注3）最近のIDEAは、Darculaというダークテーマが特徴になっていますが、印刷のことを考慮して、デフォルトテーマでスクリーンショットを撮っています。
注4）URL https://netbeans.org/

COLUMN IDEAファミリーの系譜

Ultimate Edition(以下、UE)[注5]を頂点にいくつかの言語専用IDEで構成されています(ほとんどが有償です)。ライセンス体系が若干わかりづらいこともあってか、製品体系を把握するのが難しいです。**図1**にIDEAと、その系譜のIDEの体系をまとめました。基本的な機能はどれも同じなので、ほかのIDEを使っていても本章の内容は通用すると思います。

▼図1 IDEAとその系譜のIDE



COLUMN 「Android Studio最速入門」とのリンク

2013年5月から2014年3月の間、gihyo.jpに表題の連載をさせていただきました[注6]。

タイトルこそ「Android Studio」となっていますが、そのほとんどはIDEAベースのIDE共通の内容になっています[注7]。本章で深掘りできないIDEAのさまざまな機能については、この連載を参照してください。

注5) おサイフに厳しいですが、IDEAの神髄は有償のUEにあります。評価版でも良いので、なるべくIDEA UEに触れてほしいです。
注6) URL http://gihyo.jp/dev/serial/01/android_studio/
注7) 書いていた本人が「Android StudioのフリしたintelliJ IDEA入門」と公言していたくらいですので(笑)

最新技術の追従がもっとも早く、正確です。Javaの正統派IDEといった趣があり、初心者から上級者まで幅広く受け入れられています。はじめから日本語化してあるのも特徴の1つです。

本章では、最新版のNetBeans 8.0を基準に機能を紹介していきます。

エディタとしての基本機能

IDEであっても、主な用途はソースコードの作成になるので、エディタの善し悪しがそのIDEの印象と直結します。コード補完やクイック修正といったIDEならではの機能は、すべてのIDEが備えており、単純な機能比較で優劣は語れません。

いずれのIDEも、10年以上世間に揉まれているだけあって、一通りの機能を備えています。個別の機能、またはいくつかの機能の連携により、それぞれのIDEの書き味が出てきます。この書き味は、とても感覚的で利用者の感性に訴えかけます。それが相性となって表れるため、どのIDEが気に入るかは、その人の感性によってさまざまと言えます[注8]。

操作体系の特徴

示し合わせたわけではないのに、どういうわけかIDEAとNetBeansの操作体系は似通っています(**表2**)。コード補完やクイック修正といった、IDEで最もよく使う編集機能もIDEAとNetBeansは同じショートカットキーが割り当てられています。

ショートカットキーの体系(キーマップ)ですが、**表3**のようにIDEAやNetBeansは、標準でいくつかのキーマップを用意しているので、まずは慣れているキーマップにして使い始めるのも良いでしょう。

個人的な感想になりますが、IDEAのショートカットキーは奇抜な組み合わせが多く、「これを覚えるくらいなら、自己流にカスタマイズしたほうがいいのでは?」と思うほどです(実際、筆者はそうしています)。

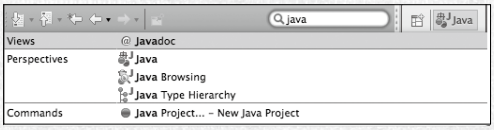
また、最近のIDEは**図2**のようなファイラ兼コマンドラウンチャが提供されています。コマ

▼表2 IDEそれぞれのよく使う機能

IDE	コード補完	クイック修正
Eclipse	Ctrl-Space	Ctrl-1 または Ctrl-2
IDEA	Ctrl-Space	Alt-Enter (またはOption-Enter)
NetBeans	Ctrl-Space	Alt-Enter (またはOption-Enter)

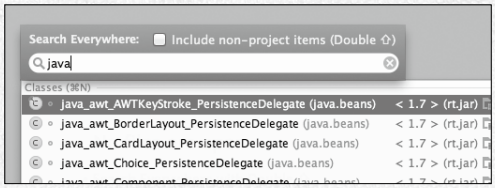
▼図2 IDEごとのコマンドラウンチャの例

Eclipse



NetBeans

IntellliJ IDEA



▼表3 IDEそれぞれのキーマップ

IDE	キーマップ	設定個所
Eclipse	デフォルト、Emacs	ウィンドウ→設定／一般→キー
IDEA	Default、Eclipse、NetBeans、Emacs、Visual Studio、JBuilder	Configure→Preferences／Keymap
NetBeans	NetBeans、Eclipse、Idea、Emacs	ツール→オプション／キーマップ

注8) すでにお気に入りのIDEやテキストエディタがある人ほど、ほかの環境に切り替えるのは苦痛を伴います。

▼表4 コマンドラウンチャのショートカットキー

IDE	「何でも検索」の名称	ショートカットキー
Eclipse	クイック・アクセス	Ctrl-3 (またはCommand-3)
IDEA	Search Everywhere	Shift を2回押す
NetBeans	クイック検索	Ctrl-i (またはCommand-i)

ンド名がうろ覚えでもイイ感じに探してくれるので、まずはこの機能のショートカットキーを覚えるのが良いでしょう(**表4**)。

重箱の隅をつつこう! 5大どうでもいい機能

EclipseユーザがIDEAやNetBeansに乗り換えたときに、「この機能はどこにあるんだ?」と探しては、その結果に不満を言わずにはいられない5大機能です。「たかだかこれくらい……」と思うかもしれませんが、使いこなすほど、こういう差違が気になるものです(まさに神は細部に宿る、ですね)。

保存時アクション

ファイルの保存時にコードフォーマットを実行する、といったアレです。NetBeansはほぼ同等の機能を持っています。残念なことにIDEAにはこの機能がありませんが、SVNやGitなどにコミットする直前なら保存時アクションに近いことができます。

ファイルの保存

Eclipseはよくあるアプリケーション同様、利用者が指示しないとファイルを保存しません。NetBeansもそれに近い振る舞いをしますが、ビルドや実行などのアクションを行うと、その時点で未保存のファイルを強制的に保存します。IDEAは自動保存であるため、そもそも「保存(Save)」機能がありません[注13]。

NetBeansやIDEAの場合は「編集中のファイルを意図的に保存しないでおく」ことができないので、これを習慣にしている人にとっては、とてもストレスフルです。

自動コンパイル

IDEAもNetBeansも、自動コンパイル機能を有しています。いずれも、Eclipseのインクリメンタルビルドを模倣したため、機能が実装されてから、それほど時間が経っていません。そのため、この機能が生粋のEclipseユーザにとって満足がいくものかどうかは評価がわかれると思います。

問題個所の一覧表示

Eclipseの「問題ビュー」のように、コンパイ

COLUMN
vimプラグインについて

「どのIDEにもデフォルトでEmacsのキーマップが用意されているのに、vimはどうしたんだ?」と憤ったvimファンのみなさん、ごきげんよう。はじめに断っておきますが、IDEのEmacsキーマップを「よかった」というEmacsユーザに会ったことはありません。つまりは、そういうことです。

そうは言っても、vimキーマップがない悔しい気持ちもわかります。そんなvimユーザのために、どのIDEにもvimプラグインがあります。

・Eclipse → Vrapper[注9]
・IDEA → IdeaVIM[注10]
・NetBeans → jVi[注11]

モードレスエディタが当たり前なIDEにムリヤリvimモードを実装してしまうvimユーザの執念たるや、すさまじいものがありますね。

ただし、注意事項というか心構えがあります。どのvimプラグインも一定以上の完成度なのですが、vimモドキであることには変わりありません。熱心なvimユーザほど、この些細な挙動の違いが気になるのですが、そこは大らかな心で受け止めてほしいものです。vimプラグインを使うために必要な心のありようは「寛容さ」です[注12]。

注9) URL http://vrapper.sourceforge.net/home/
注10) URL http://plugins.jetbrains.com/plugin/164
注11) URL http://jvi.sourceforge.net/
注12) vimに敵うエディタなぞあり得ません。王者の貫禄で、このvimモドキを受け入れてあげてください。
注13) 「名前を付けて保存(Save as)」もなく、唯一「すべて保存(Save All)」だけがあります。

ルエラーや警告、TODOを常時参照できるかどうかです。NetBeansには、類似の「アクション項目ウィンドウ」がありますが、IDEAには、これに相当するユーザインターフェース(以下、UI)がありません。

見た目が気に入らない

「それ、機能じゃない」と思うかもしれませんが、とても大事なことです。おもしろいことに、Eclipseの見た目が良いという人もいれば、IDEAやNetBeansの、いかにもJavaでございな見た目が良いという人もいます。わかっているのは、この両者は相容れられないということです。

せっかく使うなら自分の美意識に合ったIDEのほうが使っていて気分も良いですが、それを動かすOSや設定したフォントなど、深みにはまるとキリがないので、ある程度の慎みと自制心が必要です。

◆◆◆

Eclipseをほとんど知らず、IDEAばかり使っている筆者には、どれも不満に感じたことはありません。立場が逆になれば、このような不満を漏らすのは容易に想像ができます。ホント、慣れって怖いですね。

一番の違いはテストコードを管理できること

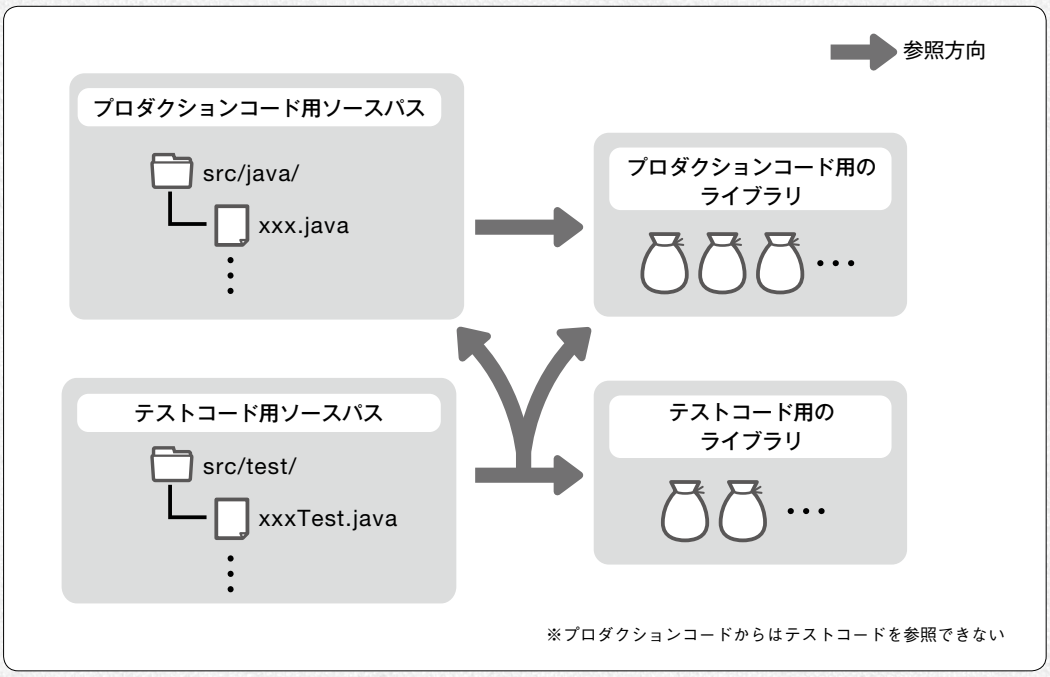
IDEAやNetBeansを使っていると、「何でEclipseにコレがないんだ!」とついつい憤慨してしまうのが、この機能です。ちょっと話は飛躍しますが、ビルドツールMavenの数少ない功績に、次の2つがあります。

- 標準的なディレクトリ構成を決めたこと
- ソースパスやライブラリパスに対するスコープを決めたこと

具体的にいうと、ソースコードはプロダクションコードとテストコードがあり、それぞれが**図3**のような参照関係になるべきだという考え方です。

この発想はとても理にかなっており広く普及しましたが、Eclipseは伝統的にソースパスをプロダクションコードとテストコードに分ける

▼ 図3 プロダクションコードとテストコードを別に管理できる



COLUMN

細か過ぎて伝わらない、テスト実行の違い

テストがらみで1つ小話を。どのIDEもJUnitのテストケースを実行できますが、その実行形式に微妙な違いがあります。それは、JUnitのforkモード[注14]のことで、IDEごとにデフォルトが**表5**のように異なります。この違いを知らないと、環境によってはテストが動作しなくなるのでご注意を。

NetBeansが比較的行儀の良い設定なのですが、こちらはさらに「テストクラスの名前が**`**Test.java`**で終わること」という暗黙のルールがあります[注15]。

▼表5 JUnitのforkモードの実行形式の違い

IDE	JUnitのforkモード	変更の可否
Eclipse	すべてのテストクラスを同じJava Virtual Machine(以下、JVM)プロセスで動かす	固定。変更できない
IDEA	同上	テストクラスごと／テストメソッドごとに変更できる
NetBeans	テストクラスごとにJVMのプロセスが異なる	ビルドスクリプトを直接修正すれば、ほかのforkモードに変更できる

ことができません。Eclipseしか知らない人は、一度で良いのでIDEAやNetBeansの「プロダクションコードとテストコードは別れていて当然」という世界を味わってほしいものです。

デバッガ

テスト駆動開発(TDD)の一派によってはデバッガを使うのは屈辱の極みと感じることもあるようですが、IDEならではの便利機能であるのも事実です。

ソースコードにブレークポイントを設定し／プログラムの実行を止め／変数の値などを確認しながら／ステップ実行していく、といったデバッガの基本機能はどのIDEも備えています。デバッガに求められるUIも、IDEによって極端に代わり映えしないので、どれかを使い慣れていれば、ほかのIDEでも多少戸惑うとしても途方に暮れるほどではないでしょう。

ただ、デバッガの用語や機能のショートカットキーが微妙に異なるため(**表6**)、複数のIDEを使い分けていると地味にイラッとします。

IDEAのデバッガの特徴

実のところ、これといった特徴はありません。プログラム実行は、Eclipseと同じような「実行構成(Run Configuration)」という概念があるため、Eclipseユーザから見ても、そう違和感な

▼表6 IDEそれぞれのデバッガ機能

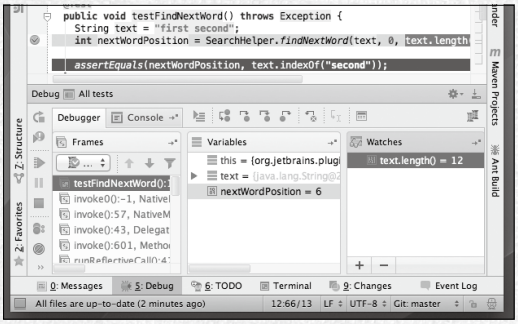
IDE	Eclipse	IDEA	NetBeans
デバッガ機能の場所	「実行」メニュー	「Run」メニュー	「デバッグ」メニュー
デバッガの機能名	再開	Resume Program	続行
	中断	Pause Program	一時停止
	終了	Stop	デバッガ・セッションを終了
	ステップイン	Step Into	ステップ・イン
	選択項目にステップイン	Smart Step Into	次のメソッドにステップ・イン
	ステップ・オーバー	Step Over	ステップ・オーバー
	ステップ・リターン	Step Out	ステップ・アウト
	指定行まで実行	Run to Cursor	カーソルまで実行

注14) テストを実行するJVMプロセスをどう使い回すか指定します。
注15) Antベースの標準的なNetBeansプロジェクトの場合です。

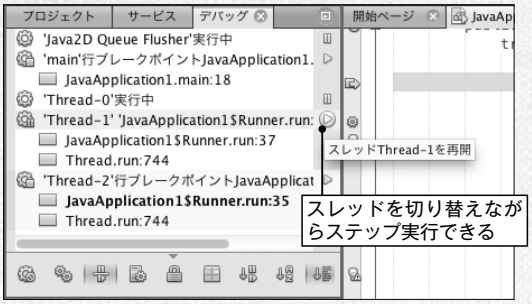
く受け入れられると思います。

違和感を覚えるとすれば、そのUIでしょう。IDEAは、EclipseのビューやNetBeansのウィンドウのような単機能のUIを複数組み合わせるのではなく、デバッグに必要な機能は「Debugツールウィンドウ」(**図4**)に集約されています。EclipseやNetBeansに慣れている人からみると、この「Debugツールウィンドウ」のレイアウトに不自由さを感じるのではないでしょうか。

▼図4 IDEAの「Debugツールウィンドウ」



▼図5 NetBeansのMulti-Threaded Debugger



NetBeansのデバッガの特徴

こちらもデバッガの基本機能に大きな差はありませんが、デバッグする対象に大きな違いがあります。簡単にいうと、テストのデバッグが「テストクラスの個別実行」[注16]でしかできません。NetBeansのプロジェクトはビルドシステムにAntを利用しているため、EclipseやIDEAのように複数の実行構成を自由に持つことができません。その延長線上に、このデメリットがあります。

テストのデバッグがまったくできないということではないので、この制約を甘んじて受け入れるしかありません。その代わりと言ってはなんですが、NetBeansにはMulti-Threaded Debugger(**図5**)という他のIDEにはない機能を持っています。

このMulti-Threaded Debuggerは名前が示すとおり、マルチスレッド処理に向いたデバッグ機能です。具体的には次のことができます。

①特定のスレッドの処理だけを追跡できる
②指定したスレッドの処理をデバッガから解放できる
③ステップ実行中に、ほかのスレッドのブレークポイントを確認できる
④さらに、そのスレッドに切り替えることができる

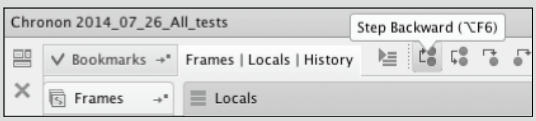
EclipseやIDEAでもブレークポイントの条件設定で、特定のスレッドだけを追跡すること

COLUMN 時間を巻き戻すデバッガ「Chronon Debugger」

有償版のIDEA UEにしか付いていませんが、Chronon Debugger[注17]という未来を感じさせるデバッガがあります。このデバッガは、プログラムの実行状況を記録し、あとから再生できます(**図6**)。

たとえば、テストプログラムをChrononに記録させておくと、テストに失敗した場所までトレースするだけではなく、失敗した場所から処理を巻き戻して何度でもトレースできます。Chrononを使うことで、デバッガでありがちな「ステップ実行が行き過ぎたので、やりなおし」という凡ミスや、ブレークポイントを慎重に設置する気苦労から解放されます。

▼図6 Chrononの巻き戻し(Step Backward)ボタン



注16) 「デバッグ」メニューの「ファイルのテストをデバッグ」のことです。
注17) Chronon自体は単独の商品なので、別途購入すればEclipseからも使えます。URL http://chrononsystems.com/

はできますが、②～④のような手の込んだ操作はできません。

筆者の経験上、Multi-Threaded Debuggerはめったに使う機能ではありませんが、必要になったときはとても頼りになる機能です。

バージョン管理システムとの連携

最後にバージョン管理システム(以下、VCS)との連携について紹介します。VCSも数多くありますが、最近とくに注目されているGitを取り上げます。Gitも含め、IDEが標準でサポートしているVCSを**表7**にまとめました。IDEAだけは別途**git**(または**git.exe**)が必要になります。

ユーザインターフェースの共通性のなさ

今まで紹介してきたエディタやデバッガは、IDEごとに多少の違いはありますが、どれか1つを知っていればほかのIDEでもなんとなく使うことができます。しかし、VCSとの連携については、IDEごとに特徴があり過ぎて、どれか1つを知っていても、別のIDEでは何をしたら良いのかわからず途方に暮れることも珍しくありません。それほどまで、IDEごとにUIが異なります。わずかな手がかりとして、IDEごとのVCSメニューの場所を**表8**に示します。

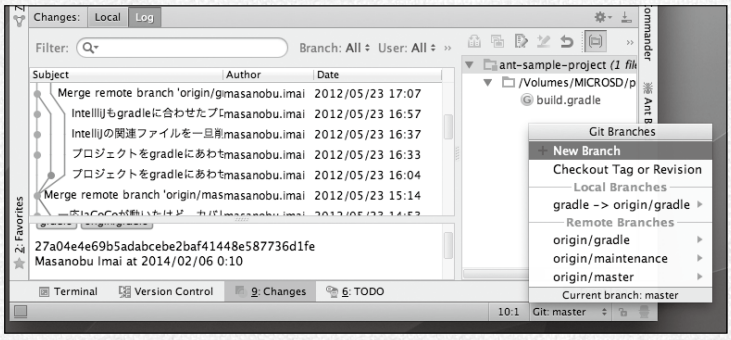
▼表7 IDEが標準でサポートしているVCS

IDE	サポートしているVCS
Eclipse	CVS、SVN[注18]、Git
IDEA	CVS、SVN、Git
NetBeans	SVN、Git、hg(Mercurial)

▼表8 IDEごとのVCSメニュー

IDE	VCS連携メニューの場所
Eclipse	コンテキストメニューの「チーム」に集約
IDEA	メニューバーの「VCS」メニューに集約
NetBeans	メニューバーの「チーム」メニューに集約

▼図7 IDEAの「Changesツールウィンドウ」



COLUMN

VCS専用クライアントが便利

IDEのVCS連携のUIはどれも個性的です。個人的には、3つのIDEのVCS連携を習熟するくらいなら、VCS専用のクライアントもしくはコマンドラインの操作を習得したほうが効果的だと考えています。

もともとIDEのVCS連携はSVN、Gitといった異なるVCSに対して統一されたUIを提供しているので、特定のVCSの機能を十分に発揮するには不向きなところがあります。その分、TortoiseGitやSourceTreeのような特定のVCS(この場合はGit)専用クライアントのほうが使い勝手が良いです。

とくにGitはできることが多彩なので、直接**git**コマンドを叩いたほうが細かい操作に向きます。ミもフタもありませんが、IDEAもNetBeansもIDE上にターミナルを持っている[注19]ので、そこから**git**コマンドを実行したほうが楽という場面が多々あります。

IDEAのVCS連携の特徴

IDEAのVCS連携では「Changesツールウィンドウ」(**図7**)を使いこなせるかどうかがポイントになります。ここで作業コピーの操作、リポジトリのログの確認といった、ほとんどの操作を行えます。

さらに、Git連携中はステータスバーに現在のブランチが表示されます。ここもクリック可能で、ブラン

注18) Pleiadesには、SVNプラグインが同梱されています。
注19) Windows版NetBeansはcygwinが必要です。Eclipseもターミナルプラグインを入れることで実現できます(Windows未対応)。

チの作成や切り替えを行えます。

また、IDEAはGitHubに対して次の操作を行えます。GitHubへの操作はある程度までIDEAだけで完結するので、意外と便利に使えています。

❶GitHubのプロジェクトをローカルに取ってくる(VCS→Checkout from Version Control→GitHub)
❷プロジェクトをGitHubに公開する(VCS→Import into Version Control→Share Project on GitHub)
❸fork元のプロジェクトの変更をforkしたプロジェクトに取り込む(VCS→Git→Rebase my GitHub fork)
❹プルリクエストを作成する(VCS→Git→Create Pull Request)

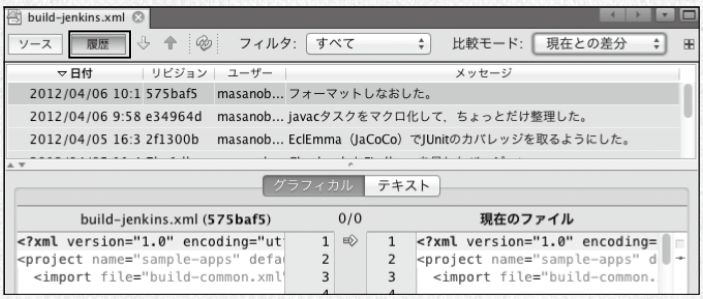
❸は、たまにしか使いませんが、よく**git**コマンドの操作を忘れるのでたいへん重宝しています。それと、ほかのGitHub連携とは毛色が異なりますが、エディタの選択範囲をGistに登録することもできます[注20]。

NetBeansのVCS連携の特徴

NetBeansもIDEAに負けず劣らず特徴的なUIを持っています。意外に見落としがちなのは、エディタのタブ直下にある「履歴」ボタンです(**図8**)。これをクリックすると、エディタがそのファイルの履歴確認画面に切り替わります。

リポジトリ全体、もしくは一部の履歴を確認するのは若干わかりづらく、「プロジェクトウィンドウ」もしくは「Gitリポジトリ・ブラウザ」からコンテキストメニューの「Git→履歴を表示」[注21]を実行します。履歴確認画面が「エディタウィンドウ」上に出てくるのもNetBeansの特徴といえます(EclipseやIDEAはエディタとは別のUIに表示します)。

▼図8 NetBeansのエディタツールバーの「履歴」ボタン



最後に

かなり駆け足でしたが、IDEAとNetBeansについて紹介しました。自分の引き出しを増やすためにも、複数のIDEを嗜(たしな)んでおくことは無駄になりません。ただし、使い込むとなると話は別です。常用しているIDEによほどの不満がない限り、たいていは隣の芝生でしかないです。どのIDEもそれなりに不便で、それなりに便利です。なんだかんだで、馴れているIDEが最も使いやすいIDEなのです。

よく見かける「○○と比べたら、□□の△△は最高」というのは、そのIDEの一側面に過ぎず、必ずそれに見合ったデメリットがあります[注22]。

大事なのは自分の感性に合うかで、他人の評判なんてアテにはなりません(せいぜい参考程度です)。興味のわいたIDEは実際に試用してみて、ご自身の手に馴染むかどうかを感じてみてください。SD

注20) コンテキストメニューから「Create Gist...」を実行します。
注21) 「Gitリポジトリ・ブラウザ」からは、直接「履歴を表示」を実行します。
注22) 筆者もIDEAに対する不満は山のようにあるのですが、それを打ち消すほど気に入っているので、気に留めたことはありません。すべてにおいて万能というIDEは実在しません。

第4章

メモリ不足、無応答、スローダウンに備える
トラブル時に頼りになるJDKの解析ツール

JDK（Java Development Kit）には、アプリケーションのメモリ不足、スローダウンなどの原因を追究するための解析ツールが豊富に存在します。せっかくJavaで開発しているのなら、これらのツールも使いこなし、高品質なアプリケーションを開発したいものです。よくあるトラブル事例ごとに、解析に適したツールを紹介していきます。

●NTTコムウェア㈱　上妻 宜人（あげつま のりと）

はじめに

プログラミング言語の選定の際、エンタープライズシステムを構築／運用していくうえでの重要な観点として、言語の使いやすさやライブラリの豊富さだけではなく、トラブル発生時に迅速に問題を特定して原因の改修が可能かという点があります。

Javaには、開発キットであるJDK（Java Development Kit）にさまざまな解析ツールが同梱されており、トラブル解析に備えた充実した環境が整っています。本章では、トラブル対応に便利なJava起動オプションおよび、JDK付属解析ツールの状況に応じた使い分け方法について紹介します。

トラブルに備える編

いざトラブルに遭遇するとJavaプロセス再起動による復旧を急ぐあまり、「情報がない」「再現方法が不明」などの理由から原因がお蔵入りになってしまうケースが数多くあります。

JDK付属のJava VMである「HotSpot」には、このような状況に備えて、起動オプションの設定によりトラブル発生時の解析に必要な情報を自動的に出力する機能があります。

OutOfMemoryErrorに備える

Javaの代表的なトラブルとして、生成されたオブジェクトが格納されるJavaヒープメモリの枯渇を示す「OutOfMemoryError」があります。次のエラーメッセージを一度は見たことがある方も多いのではないでしょうか。

```
java.lang.OutOfMemoryError: Java heap space
```

OutOfMemoryErrorが発生した場合には、可能な限り早くJavaプロセスを再起動することが必要です。ヒープメモリの枯渇は多くの場合、Javaプロセスを再起動しない限り解消されず、エラー再発の原因となります。また、OutOfMemoryErrorが発生したスレッドは多くの場合、エラーハンドリングされずにそのまま終了するため、アプリケーションが不定な状態となります。

たとえば、Javaシステムでよく使用されるサーブレットコンテナ「Tomcat」において、リクエスト待ち受けスレッドがOutOfMemoryErrorにより終了した場合には、プロセス監視では正常に見えていても、実際には無応答になっているなど不定な状態が発生し、サービスに深刻な影響を与えてしまいます。

GCログの収集と考察

OutOfMemoryErrorを未然に防ぐために、負荷テスト時にGC（ガベージコレクション）ログを収集して考察すると、Javaヒープメモリ枯渇の兆候がないかを確認することができます。商用運用時にも定常的に収集することで、GC長期化などのトラブル原因を切り分ける際のヒ

▼図1 GCログを収集するJava起動オプション

```
-Xloggc:/var/log/java/gc_%p_%t.log
                ❶
-XX:+PrintGCDetails -XX:+PrintGCDateStamps
        ❷                   ❸
-XX:+UseGCLogFileRotation -XX:NumberOfGCLogFiles=5 -XX:GCLogFileSize=10M
        ❹                         ❺                       ❻
```

❶GCログの出力先ファイル名。JDK 8より%pでプロセスID、%tで日時をファイル名に含めることが可能
❷より詳細なGC情報をロギングする ❸GCの発生日時をログに含める
❹GCログファイルのサイズローテーション有効化(JDK 6u34/JDK 7u2から)
❺ローテーション有効時の管理世代数 ❻ローテーション契機のファイルサイズ

▼図2 GCログ出力例

```
2014-07-21T21:31:26.390+0900: ←①
  754.036: [GC (Allocation Failure) ←②
  754.036: [DefNew: 141748K->1962K(157248K), 0.0196019 secs] ←③
                          259053K->119266K(506816K), 0.0197680 secs] ←④
[Times: user=0.02 sys=0.00, real=0.02 secs] ←⑤
```

※上記のログは、実際は1行のログ。今回は、説明のために改行を入れている
①GCが発生した時刻。-XX:+PrintGCDateStampsにより日付形式で出力 ②GCが発生した要因
③New領域のヒープ情報。 GC前サイズ->GC後サイズ(最大New領域サイズ)、GC時間
④ヒープ全体の情報。フォーマットはNew領域のログと同じ
⑤GC時間。userはユーザモード時間、sysはカーネルモード時間、realは実際にかかった時間

ントになります。

GCログは**図1**のようにJava起動オプションに設定を追加すると、**図2**のような形式で出力されます。

GCログは「GCViewer」[注1]などのツールに読み込ませて考察します。よくあるトラブル時のGCViewer表示例を見てみましょう。

図3のGCログでは、途中までは安定しているものの、グラフの右端において急激にJavaヒープメモリ使用量が増大しています。このような場合のよくある原因は、DBやファイルから大量のデータをロードしていることです。少しずつデータを取得するようにアプリケーションを見直す必要があります。

図4のじわじわと上昇するGCログもトラブルの傾向が確認できます。このような場合のよくある原因は、O/Rマッパや HTTP セッションなどのキャッシュが除々に蓄積し、最終的にOutOfMemoryErrorに至ることです。キャッシュサイズの見直しや、セッションタイムアウト期間の短縮化などの対処が必要となります。

OutOfMemoryError発生時ヒープダンプ

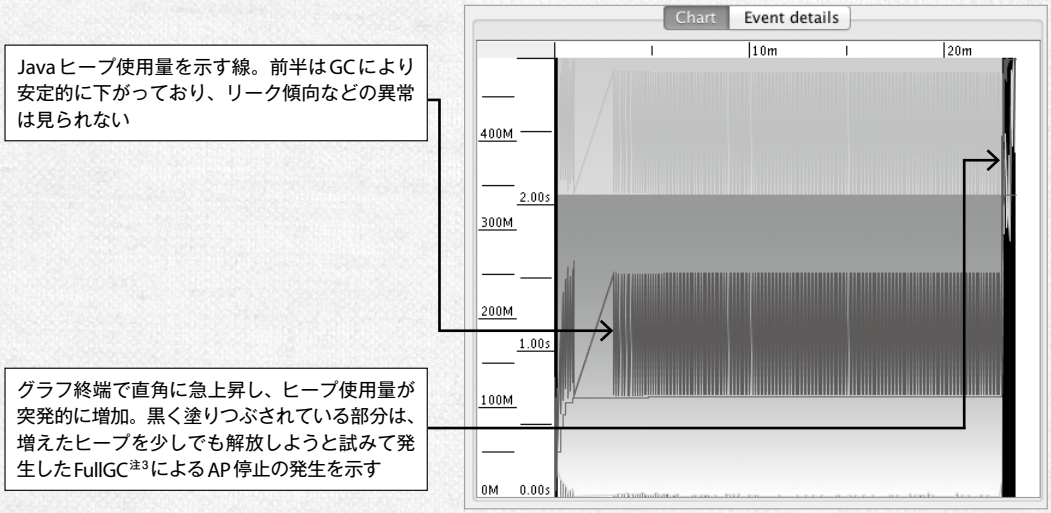
GCログはあくまで問題の傾向をつかむのみで、Javaヒープメモリを大量に消費している犯人を特定することは困難です。原因解析にはJavaヒープメモリの状態をバイナリ形式で一括出力したヒープダンプを収集します。HotSpotにはOutOfMemoryError発生時に自動的にヒープダンプを出力する機能があり、情報収集漏れによる原因のお蔵入りを防ぐことができます。

OutOfMemoryError時自動ヒープダンプを有効化するためには、Java起動オプションに**図5**の設定を追加します。

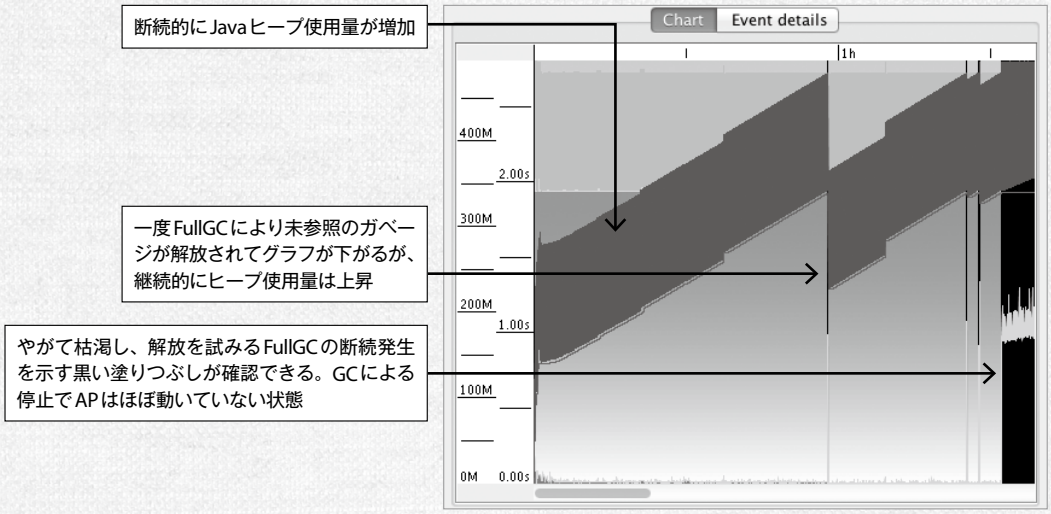
ヒープダンプの解析はJDK付属ツールの「jhat」でも可能ですが、「Eclipse Memory Analyzer」[注2]、通称「MAT」が直感的な操作で使いやすくお勧めです。よくあるトラブルとして、Javaの代表的なO/Rマッパである「Hibernate」を使用して一度

注1) GCViewer URL https://github.com/chewiebug/GCViewer
注2) Eclipse Memory Analyzer URL http://www.eclipse.org/mat/

▼図3　GCViewerの表示例　異常時のGCログ（突出型）



▼図4　GCViewerの表示例　異常時のGCログ（じわじわ型）



▼図5　OutOfMemoryError発生時にヒープダンプを自動出力するJava起動オプション

```
-XX:+HeapDumpOnOutOfMemoryError -XX:HeapDumpPath=/var/log/java
                ❶                          ❷
```

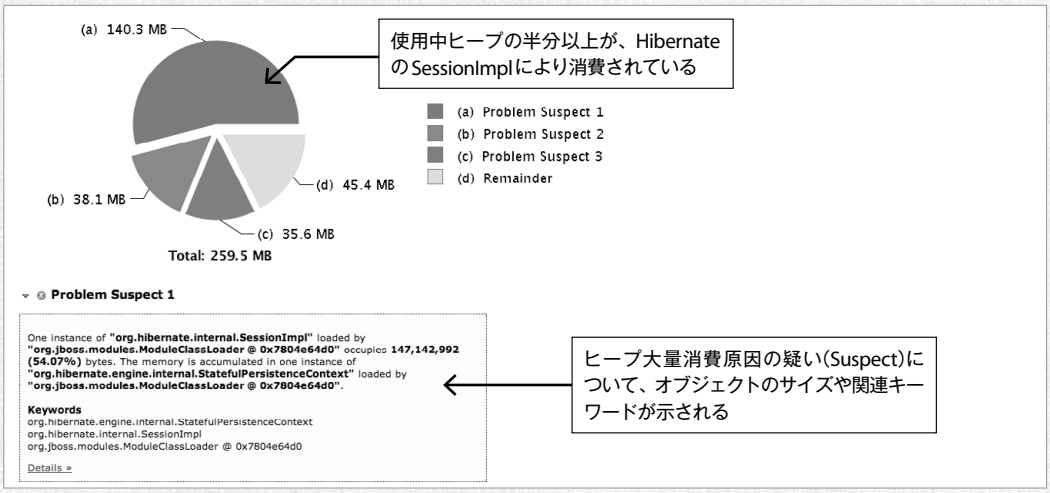
❶OutOfMemoryError発生時にヒープダンプを自動出力する　❷ヒープダンプの出力先パス

に大量のデータをDBからロードした場合のヒープダンプを確認してみましょう。

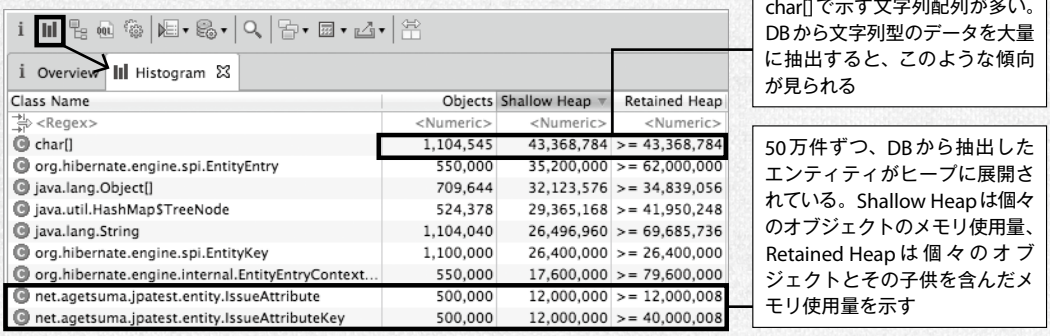
ヒープダンプファイルを開くときのウィザードでLeak Suspectsを有効にすると、**図6**のように、Javaヒープメモリに含まれるオブジェクトの内訳を円グラフで表示し、ヒープ圧迫の原因の候補を示します。この場合、Hibernateのセッション情報を保持するSessionImplクラスのサイズが大きくなっているため、トラブルにDBアクセス処理が関連していることがわかります。

注3）FullGCとは、Javaヒープメモリのうち、Old領域を対象とした比較的AP停止時間の長いGC。マシン環境やヒープサイズにも依存するが、数百ミリ～数秒程度かかる。対照的にNew領域を対象としたScavenge GCは通常数十ミリ程度と短い。

▼図6 Leak Suspects



▼図7 ヒストグラム



Class Name	Objects	Shallow Heap	Retained Heap
<Regex>	<Numeric>	<Numeric>	<Numeric>
char[]	1,104,545	43,368,784	>= 43,368,784
org.hibernate.engine.spi.EntityEntry	550,000	35,200,000	>= 62,000,000
java.lang.Object[]	709,644	32,123,576	>= 34,839,056
java.util.HashMap$TreeNode	524,378	29,365,168	>= 41,950,248
java.lang.String	1,104,040	26,496,960	>= 69,685,736
org.hibernate.engine.spi.EntityKey	1,100,000	26,400,000	>= 26,400,000
org.hibernate.engine.internal.EntityEntryContext...	550,000	17,600,000	>= 79,600,000
net.agetsuma.jpatest.entity.IssueAttribute	500,000	12,000,000	>= 12,000,008
net.agetsuma.jpatest.entity.IssueAttributeKey	500,000	12,000,000	>= 40,000,008

▼図8 GCログ収集とヒープダンプ自動出力を指定したJava起動コマンド

```
java -Xms?g -Xmx?g  ←?にはヒープのサイズを数字で指定する
-Xloggc:/var/log/java/gc_%p_%t.log
-XX:+PrintGCDetails -XX:+PrintGCDateStamps
-XX:+UseGCLogFileRotation -XX:NumberOfGCLogFiles=5 -XX:GCLogFileSize=10M
-XX:+HeapDumpOnOutOfMemoryError -XX:HeapDumpPath=/var/log/java
```

次にクラスごとのサイズをランキング形式で表示するヒストグラムを確認します(**図7**)。ヒストグラムの内容より、char配列による文字データや、DBより抽出したレコードをオブジェクトにマッピングしたエンティティクラスが、ヒープのうちの多くを占めていることがわかります。

このケースでは、クエリのWHERE条件変更や、javax.persistence.Query.setMaxResult(int maxResult)による取得エンティティ数の制限により、データを少しずつ取得するように範囲を限定する対処が考えられます。

トラブルに備える編 まとめ

ここまで紹介してきたJava起動オプションをまとめると**図8**のようになります。Java起動オプションには、ヒープサイズの設定(-Xms/-Xmx)以外にも、これだけの多くの便利な機能が備わっています。ぜひこれらの起動オプションの組み合わせを参考にしてみてください。

トラブル時の初動対応編

開発時に入念なテストが行われていたとしても、いざ商用サービスとしてリリースすると思いがけないトラブルに遭遇してしまうことは、よくある苦い経験です。ここからは、実際にトラブルが発生してしまった際の初動対応として、JDK付属の解析ツールの使い分け方法を紹介します。環境を選ばずに使えるコマンドラインツールを中心に解説します。

アプリケーションの無応答

アプリケーションの応答がない場合は、Javaプロセスを再起動する前に必ずスレッドダンプを収集しましょう。スレッドダンプとは、その瞬間に生存していた各スレッドの動作状態およびスタックトレースをテキスト形式で出力した情報です。

最新のJDK 8ではスレッドダンプの取得方法が複数あります。いずれもスレッドダンプを取得したいJavaプロセスが起動しているマシンにおいて、**表1**のコマンドを実行します。以降のコマンド例の**${JAVA_HOME}**は、JDKのインストールディレクトリを示します。

かねて主流であったシグナル送信によるスレッドダンプ取得は、対象Javaプロセス側のコンソールに出力されてしまうため、サーバアプリで/dev/nullに標準出力を破棄していた場合、スレッドダンプが収集できない問題がありましたが、jstackおよびjcmdではコマンド実行側のコンソールに出力されるため便利です。

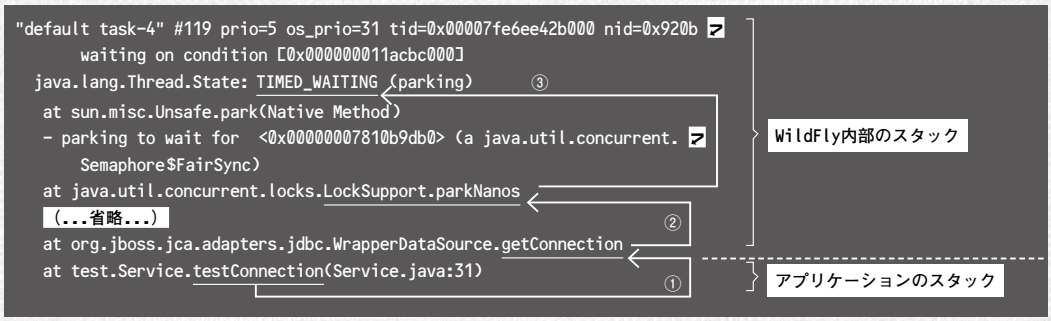
スレッドダンプの取得例

よくある無応答の原因として、DBコネクションプールの枯渇があります。最新仕様のJava EE 7に準拠したアプリケーションサーバ「WildFly 8」でコネクションプール枯渇のトラブルを再現させた場合、**図9**のようなスレッドダンプが出力されます。この場合、アプリケーションのtest.Serviceクラスの31行目でDBコネクションを取得するgetConnectionメソッド

▼表1　スレッドダンプの取得方法

JDKのバージョン	コマンドなど
JDK 5未満(Windows)	CTRL + Break
JDK 5未満(UNIX系OS)	kill -3 <pid>
JDK 5以降	${JAVA_HOME}/bin/jstack <pid>注4
JDK 7u4以降	${JAVA_HOME}/bin/jcmd <pid> Thread.print

▼図9　スレッドダンプの出力例



①アプリケーションのtestConnectionメソッドよりプール済みコネクションを取得。DataSource.getConnection()を実行
②プールが枯渇していたため、時間制限付き解放待ちをLockSupportクラスで行う
③結果的に、現在のスレッド状態はTIMED_WAITING(待ち状態)で停止中

注4）JDK 5のjstackコマンドはSolaris/Linuxのみ対応。WindowsはJDK 6より対応。

を呼び出したところ、処理内部でスレッドがブロックされて、応答がないことを示しています。

コネクション取得時に長時間ブロックされるおもな原因はDBコネクションプールの枯渇です。対処として、最大プール数の拡大や、長時間コネクションを拘束する遅いSQLのチューニングが考えられます。

アプリケーションのスローダウン

スローダウンは再現性がないなど、原因の捕捉がしにくく、解析が難しいトラブルの1つです。よくあるスローダウンの原因は大きく2つに分けられ、GC頻度の増加や長時間化などのGCトラブル、または特定メソッドの処理遅延です。

GCトラブルについては前述のとおり、GCログを収集してJavaヒープメモリの不足によるGC多発が発生していないかを確認します。特定メソッドの処理遅延の場合、無応答時と同様にスレッドダンプの収集は有効です。しかし、遅延しつつも止まることなく動作している状況では、スレッドダンプからはどのメソッドがスローダウンの原因となっているか、わかりにくいことがあります。

このような場合には、テスト環境でスローダウンを再現させてJDK付属のプロファイラ「HPROF」によりCPUプロファイリングを行って原因解析します。

HPROFによるプロファイリング

HPROFを有効化すると、CTRL + Break(Windows)/`kill -3`(UNIX系OS)によるシグナルの受信時、またはJavaプロセスの終了時に、テキスト形式のプロファイル結果を出力します。

HPROFプロファイラによるCPUプロファイリングを有効にするには、**図10**の設定をJava起動オプション追加します。サンプリングによるプロファイル(`cpu=samples`)では、デフォルトで10ミリ秒間隔でスタック情報を収集し、「あるメソッドがスタックに出現した回数÷スタック収集回数」で各メソッドのCPU使用率を算出します。スタックの上位4メソッドが計測対象とされ、ログ上部のTRACEに出力されます。ログの最後には、各メソッドのCPU使用率がランキング形式で表示されます(**図11**)。

HPROFの弱点として、スレッドスタックへの出現回数が多いメソッドをカウントしているシンプルな構造上、リクエスト受け付け待ちのようなアプリケーションサーバで定常的に動いているメソッドが上位にあがってきてしまい、本当のスローダウンの原因が見えにくいことがあります。

図11の例では、rank1と2はリクエスト受け付け待ちの処理で、スローダウンとは関連のないものです。rank3にスローダウンのサンプル

▼図10 HPROFのCPUプロファイリングを有効にするJava起動オプション

```
-agentlib:hprof=cpu=samples,file=/var/log/java/hprof_cpu.txt
```

▼図11 CPUプロファイリングの出力例

```
TRACE 302235:
    test.SlowServlet.slow(SlowServlet.java:47)
    test.SlowServlet.processRequest(SlowServlet.java:27)
    test.SlowServlet.doGet(SlowServlet.java:66)
    javax.servlet.http.HttpServlet.service(HttpServlet.java:687)
 (...省略...)
CPU SAMPLES BEGIN (total = 58116) Sun Aug  3 00:33:39 2014
rank   self  accum   count trace method
   1 85.03% 85.03%   49414 301270 sun.nio.ch.KQueueArrayWrapper.kevent0
   2  6.83% 91.86%    3971 302212 java.net.PlainSocketImpl.socketConnect
   3  2.43% 94.29%    1411 302235 test.SlowServlet.slow
```

クラスを示すtest.SlowServlet.slowメソッドが確認できます。アプリケーションのクラスのうち、一番上位に出てきているものを探すのがプロファイル結果を考察するコツです。

Javaヒープメモリの枯渇

トラブルに備える編でも紹介したJavaヒープメモリ枯渇の解析ですが、-XX:+HeapDumpOnOutOfMemoryErrorの設定を忘れてしまった場合や、OutOfMemoryErrorには至らないものの徐々にヒープが枯渇する状況を解析する場合のために、コマンド実行による情報収集が可能です。

Javaヒープメモリの解析には、先ほども紹介したヒープダンプと、今回紹介するクラスヒストグラムの2つの方法があります。

ヒープダンプの収集

ヒープダンプ収集対象のJavaプロセスが起動しているマシンにおいて、**表2**のコマンドを実行します。JDK 7u4以降ではどちらも有効ですが、jcmdのほうがコマンドが直感的な表記でお勧めです。

注意したい点として、ヒープダンプの出力中はアプリケーションが停止するため、大きなヒープサイズを持つアプリケーションの運用中にヒープダンプを収集すると、サービスに影響を与えてしまいます。停止時間はJavaヒープメモリの使用量とマシンスペックに依存します。参考として、筆者の手元のマシン[注6]において、3GBのヒープダンプを取得した場合、約30秒停止しました。

日中帯でアクセスが多い状況など、ヒープダンプ取得による一時停止が許容できない場合は、比較的動作の軽いクラスヒストグラムによりJavaヒープメモリ内容の情報を収集します。

クラスヒストグラムの収集

クラスヒストグラムは、ヒープ使用量の多いクラス順に情報をテキスト形式で出力します。収集時にアプリケーション停止を伴うFullGCが発生するため、クラスヒストグラムも軽い処理ではないですが、ヒープダンプに比べると短い停止時間で情報収集が可能です。クラスヒストグラムの収集は、対象のJavaプロセスが起動しているマシンにおいて、**表3**のコマンドを実行します。

図12は、先ほどのヒープダンプの例と同様に、Hibernateで大量のデータを一度にDBから

▼表2　ヒープダンプ取得方法

JDKのバージョン	コマンド
JDK 5以降[注5]	`${JAVA_HOME}/bin/jmap -dump:live,format=b,file=<filename> <pid>`
JDK 7u4以降	`${JAVA_HOME}/bin/jcmd <pid> GC.heap_dump <filename>`

▼表3　クラスヒストグラムの取得方法

JDKのバージョン	コマンド
JDK 5以降	`${JAVA_HOME}/bin/jmap -histo:live <pid>`
JDK 7u4以降	`${JAVA_HOME}/bin/jcmd <pid> GC.class_histogram`

▼図12　クラスヒストグラムの取得例

```
 num     #instances         #bytes  class name
----------------------------------------------
   1:        702063       29823264  [C
   2:        330000       21120000  org.hibernate.engine.spi.EntityEntry
   3:        456309       18615880  [Ljava.lang.Object;
```

注5） JDK 5のjmapコマンドはSolaris/Linuxのみ対応。WindowsはJDK 6より対応。
注6） Macbook AIR MC965（Corei5 1.7GHz）

Javaヒープメモリへロードした場合のヒストグラムです。MATでヒープダンプを確認した場合と同じく、[Cが示すchar型の配列が、ヒープのうち一番多くを占めていることが確認できます。

トラブル時の初動対応編 まとめ

ここまで紹介したトラブル発生時の初動対応方法をまとめます。事前に練習して、いざというときにすぐ情報収集できる準備を整えておきましょう。

- 無応答の場合は必ずスレッドダンプ
- スローダウン時はGCログとスレッドダンプの収集。原因がわからない場合は、テスト環境で再現させてHPROFによるプロファイリング
- Javaヒープメモリ枯渇時は自動ヒープダンプ出力の有無を確認。出力されていなかった場合は、まずはクラスヒストグラムを収集。アプリケーションが多少停止しても許容できる場合は手動ヒープダンプ

JDK 7/8から導入された新しい解析ツール

Oracle社はHotRockitプロジェクトとして、買収により手中に納めた2つのJava VM実装「HotSpot」と「JRockit」の統合を段階的に進めています。統合に伴い、JDK 7からJDK 8にかけてJRockitにしかなかった便利な解析ツールが追加されています。

jcmd

jcmdはJRockitに含まれていたjrcmdコマンドの後継として、JDK 7 update4から導入されたコマンドライン解析ツールです。**表4**のように既存のコマンドラインツールは、取得したい情報に応じてコマンド名が異なりましたが、現在はほとんどの情報がjcmdによって取得可能です。jcmdのコマンド一覧はjcmdのヘルプから確認できます。

```
${JAVA_HOME}/bin/jcmd <pid> help
```

Java Mission Control/Flight Recorder

JRockitに含まれていた強力な解析ツール「JRockit Mission Control」の特徴を引き継いだ、「Java Misson Control」および「Java Flight Recorder」がJDK 7 update40より正式にサポートされています。

いずれのツールもJDKに同梱されていますが、評価／検証以外を目的とした商用利用には、Oracle社が提供する有償ライセンス[注7]が必要なため注意が必要です。

Java Mission Controlは、以前よりJDKにバンドルされていたGUI解析ツール「JConsole」や「Java VisualVM」のように現在のJava VMの状態をグラフィカルに表示する機能(**図13**)と、事前にJava Flight Recorderで収集したプロファイルデータをロードして表示する機能を持ちます。

トラブルシューティングにとくに効果的なの

▼表4 JDK既存ツールとjcmdの対応

収集情報	既存のJDKツール	jcmd(JDK 7u4〜)
起動中Javaプロセス ID取得	jps	jcmd
スレッドダンプ	jstack <pid>	jcmd <pid> Thread.print
ヒープダンプ	jmap -dump:live,format=b,file=<filename> <pid>	jcmd <pid> GC.heap_dump <filename>
クラスヒストグラム	jmap -histo:live <pid>	jcmd <pid> GC.class_histogram
Java起動オプションの確認	jinfo -flags <pid>	jcmd <pid> VM.flags

注7) Java SE Advanced、またはJava SE Suiteの契約が必要。

が、後者のフライトレコーダとの組み合わせ機能です。Java Flight Recorderは、商用運用中のマシンでの収集に耐え得る低いオーバーヘッドで、CPU使用率などの基本的なマシンリソース情報をはじめ、Javaヒープ情報、スレッド状態の遷移、実行時間の長いホットメソッド情報など、さまざまな解析情報を収集します。

Java Flight Recorderの使い方

Java Flight Recorderは3つのステップで使用します。

①フライトレコーダの有効化
②フライト記録の開始
③フライト記録のダンプ

①フライトレコーダの有効化

まずフライトレコーダの有効化ですが、Java起動オプションに**図14**の設定を追加することで有効化されます。ただ、これらのオプションを付与するだけでは、フライトレコーダが使用可能となるだけであり、実際のプロファイルデータの収集は開始されません。

②フライト記録の開始

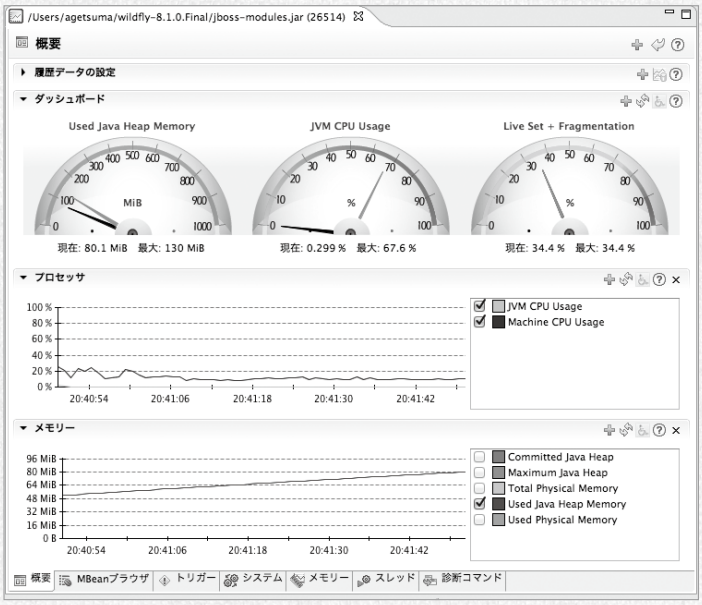
次に、フライトデータの収集を開始します。フライトデータの収集には多くのオプション設定がありますが、ここでは実際によく使用するJava起動オプションに設定する方法と、jcmdコマンドで明示的に収集を開始する2つの方法を紹介します(**図15**)。詳細なオプション設定は、Oracle社が公開しているガイド[注8]を参照してください。

Java起動オプションに設定した場合は、起動時よりプロファイル情報を定常的に収集し、デフォルトで直近15分間のデータをメモリおよびシステムの一時領域(/tmpなど)に保持します。オーバーヘッドも少ないので、将来的なトラブルに備えてあらかじめ情報収集する用途に有効です。

図15の①で示しているのは、最大で直近60分間のフライトデータを保持し続ける場合のJava起動オプションの例です。

一方、オーバーヘッドを極力抑えるために、トラブルの傾向が見え始めてから明示的にデータ収集を開始することも可能です。フライト記録の開始と停止には、**図15**の②のように、対象のJavaプロセ

▼図13　Java Mission Control

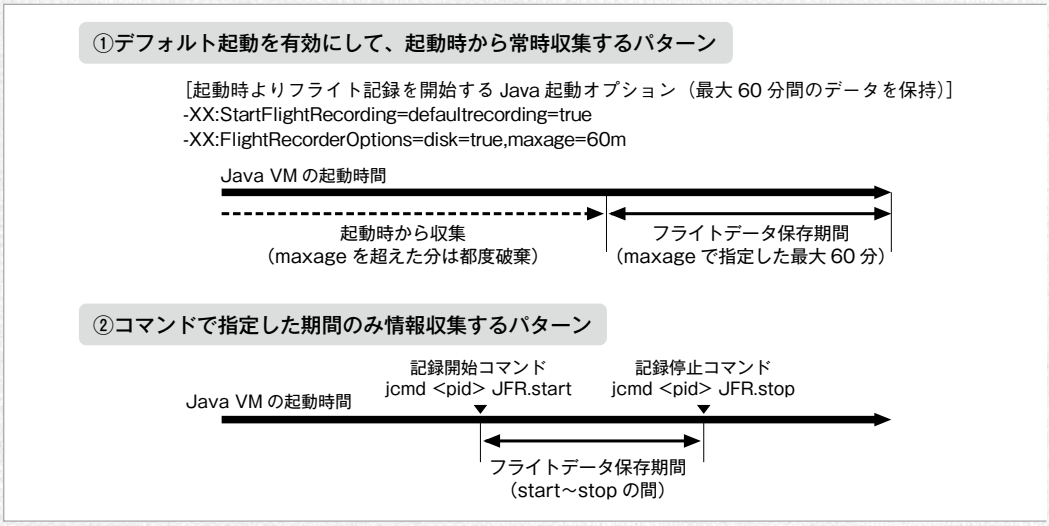


▼図14　Java Flight Recorder有効化のJava起動オプション

```
-XX:+UnlockCommercialFeatures -XX:+FlightRecorder
```

注8）Java Flight Recorder Runtime Guide URL http://docs.oracle.com/javase/7/docs/technotes/guides/jfr/toc.html

▼図15 Java Flight Recorderの収集設定



▼図16 フライト記録の確認とダンプ

```
フライト記録の確認
${JAVA_HOME}/bin/jcmd <pid> JFR.check
11208:
Recording: recording=0 name="HotSpot default" maxage=1h (running)
フライト記録のダンプ
${JAVA_HOME}/bin/jcmd <pid> JFR.dump recording=0 filename=/var/log/java/my.jfr
```

スが起動しているマシンにおいて、次のコマンドを実行します。

```
${JAVA_HOME}/bin/jcmd <pid> JFR.start
${JAVA_HOME}/bin/jcmd <pid> JFR.stop recording
=<停止対象のレコード番号>
```

③フライト記録のダンプ

3つ目のステップがフライト記録のダンプです。Java Mission Controlで読み取り可能なフライト記録ファイル(.jfr形式)を出力します。

ダンプするためには、ダンプ対象のフライトレコード名、またはレコード番号を指定する必要があります。収集済みフライト記録の一覧は**jcmd <pid> JFR.check**により確認できます。**図16**は、**図15**の①のJava起動時よりフライト記録を開始していた場合の実行例です。レコード名は"HotSpot default"、レコード番号は0であることが確認できます。その後、確認したレコード番号を指定してダンプしています。

フライト記録ファイルをサーバから収集し、解析用のローカルマシンのMission Controlに読み込ませることで、前述のGCログやHPROFのプロファイルデータ、ヒープダンプなどの個別の情報を収集しなくても、統合的な情報を持つフライト記録1つで原因解析が可能です。

終わりに

トラブルの現場ではサービス復旧が最優先であるため、障害情報収集に与えられる時間はほとんどありません。しかし、ここまで解説してきたように、Javaには事前のオプション設定によるエラー情報の自動収集や、豊富な解析コマンドにより、障害情報を収集できる環境は十分に整っています。

ぜひ今回紹介したJava起動オプションや解析コマンドを練習して、いざというときに実践できる準備を整えていきましょう。SD

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

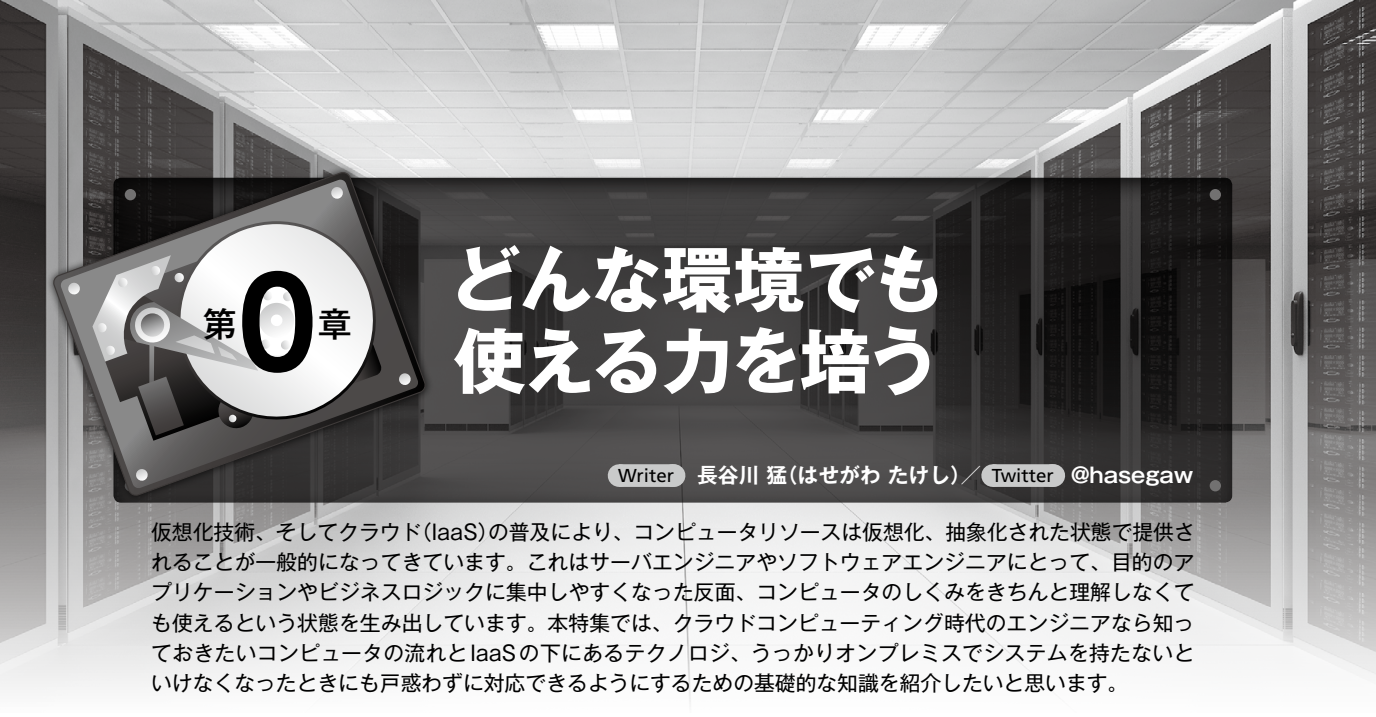
サーバの目利きになる方法

x86サーバハードウェア入門 前編

クラウドが普及し当たり前のものになり、サーバを買う前にインスタンスを買う昨今。ビジネスのスピードが上がり、すぐに対応できるクラウドサービスは魅力的ですが、エンジニアとしては隅々まで自分の理解をめぐらせておきたいものです。ブラックボックスになってしまったシステムはとくにそうです。中身を知りたいのが性(さが)といえましょう。本特集は、クラウドだって物理的なサーバがなければ成り立たないよね！——というスタンスで、x86 サーバの機能を総点検します。ハードウェアとその機能をしっかりと押さえることで「目利き」になりましょう。前編では、サーバの心臓部であるプロセッサとシステムメモリと拡張バスである PCI Express を解説します。次号の後編では、ネットワークとストレージを解説する予定です。システムを隅々まで知りたい——エンジニアの知的好奇心と根源的欲求を満たしましょう！

CONTENTS

Writer 長谷川 猛（はせがわ たけし）



第0章 どんな環境でも使える力を培う

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

仮想化技術、そしてクラウド(IaaS)の普及により、コンピュータリソースは仮想化、抽象化された状態で提供されることが一般的になってきています。これはサーバエンジニアやソフトウェアエンジニアにとって、目的のアプリケーションやビジネスロジックに集中しやすくなった反面、コンピュータのしくみをきちんと理解しなくても使えるという状態を生み出しています。本特集では、クラウドコンピューティング時代のエンジニアなら知っておきたいコンピュータの流れとIaaSの下にあるテクノロジ、うっかりオンプレミスでシステムを持たないといけなくなったときにも戸惑わずに対応できるようにするための基礎的な知識を紹介したいと思います。

コンピュータのしくみを振り返る

筆者が初めて本誌を手にした1990年代後半、LinuxやFreeBSDといえばオープンソースで無償利用できるオペレーティングシステムで、旧スペックのマシンにこれらのOSをインストールするのがトレンドになっていたころでした。当時高校生だった筆者も、FreeBSDマシンでSambaやApache、IMAP、PPPサーバ、そして無線LANのブリッジ機能などを持たせ、いわゆる自宅サーバを運用し始め、その経験は就職後のサーバの設計、構築、運用業務にも大いに役立つ経験となりました。

今ですと、こんなことは自宅にマシンをわざわざ置かなくても、月数百円～数千円で利用できるVPSやクラウド上のサーバで十分です。インターネットやサーバは私たちにとって身近なものとなりましたが、モノを持たずにサーバを利用できるようになった今、物理的にサーバを構築したり、そのための機材選定を行う機会は激減したのではないかと思います。幸いにも、筆者は現在半導体ストレージを扱う立場から、高負荷なワークロードをオンプレミスで捌く状況を見ています。本特集では、筆者がこれまでに見てきた経験を目利きになるための方法としてシェアできればと思います。

計算機から始まったコンピュータ

コンピュータはプロセッサ、システムメモリ、そして入出力(I/O)の3つのしくみの組み合わせでできています。身近かつシンプルなコンピュータの一例として、電卓を考えてみましょう(**図1**)。電卓は、キーボードから入力した数を計算して、計算結果を画面に表示します。ユーザの入力中は、数をメモリ上に記憶しておき、式が入力された時点で計算し、結果を画面に表示しています。この場合、入力デバイスはキーボード、そして出力デバイスとしてはディスプレイが存在します。この基本は、現在みなさんが手許で使われているコンピュータでも、サーバでも通用する、コンピュータの基本的なしくみです。

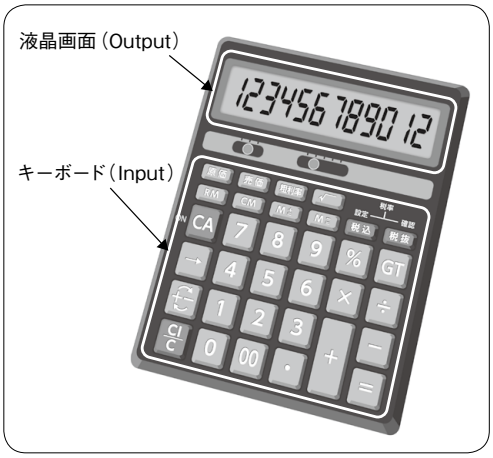
コンピュータの歴史と、今どきの電子式計算機、いわゆる電卓の間には深い関係があります。1971年に日本のビジコン社は、世界初のワンチップLSIによる電子式計算機LE-120Aを発売しました。この製品が生まれる過程で造り出されたLSIこそが、世界初のマイクロプロセッサと言われるIntel 4004でした。今私たちが使っているコンピュータは、まさに電卓の延長線上にあるテクノロジなんですね。

現代のサーバの構成

現代のサーバに話を戻しましょう。サーバといってもいろいろなアーキテクチャがありますが、多くの方が使われているであろう、x86サーバを前提に話を進めていきたいと思います。**写真1**はIBMのRedBooks(情報サイト)で紹介されている、System x3650 M4の内部写真です[注1]。このサーバの場合プロセッサ用スロットが中央部に2ヵ所あり、そのまわりには合計24個のメモリソケットがあることがわかります。また、本体後面にはPCI Express仕様に準拠したスロットがあり、各種デバイスやインターフェースを接続できます。これらはライザーになっており交換が可能になっています。

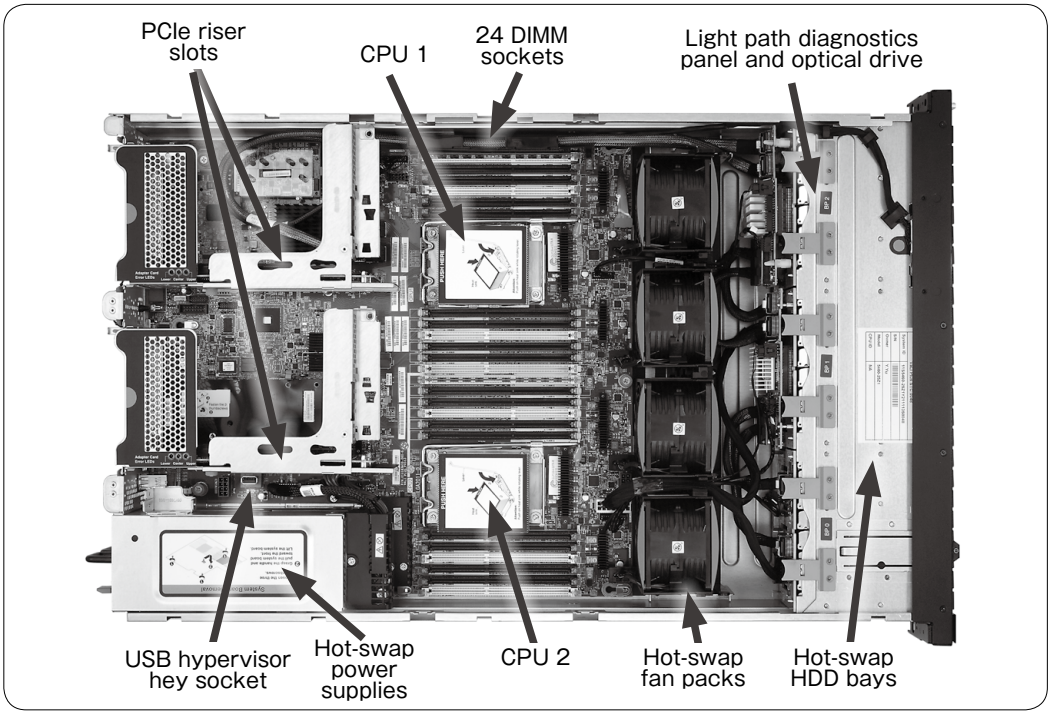
電源ユニットやファン、前面のHDDベイがホットスワップに対応しており、システム停止なしで障害コンポーネントが交換できる点も、サーバ機ならではです。写真からは読み取れない部分ですが、描画能力は控えめながらリモート管理に対応するビデオ機能や、ホストプロセッサで実行されるOSとは独立して動作する管理プロセッサ(BMC：Base Management Controller)などがある点も特徴的です。

▼図1　電卓の構造を考える



注1）http://www.redbooks.ibm.com/abstracts/tips0850.html

▼写真1　IBM System x3650 M4の内部



サーバの構成を検討する

サーバのハードウェアは、メーカーによりますが、各モデルごとにベースユニットと追加(変更)が可能なパーツのリスト、依存関係などの情報が記載された資料が用意されており、PDFのフォーマットで入手できます[注2]。たとえば、2Uの2プロセッサのサーバを選択し、載せるプロセッサを選び、DIMMの容量と個数を決め、必要に応じてネットワークインターフェース、ハードディスクとアレイコントローラを選び……とサーバの構成を決定していくわけです。

メーカーによっては、見積もりを出力したり実際に注文が可能な通販サイト(**図2**)を提供しているベンダ、サーバの構成を検証するためのアプリケーションを提供するベンダもあります。「このパーツはこの組み合わせで大丈夫なのだろうか」と思ったら、想定する構成をこれらに入力してみるのも1つの手です。

サーバ構成にはさまざまな注意ポイントがある

見積もりサイトや構成検証アプリケーションでサーバの構成を試していると、特定のメモリスロットやPCI Expressスロットを利用するためにはプロセッサの増設が必要だったり、ハードディスクの構成によってはコントローラの指定が必要であるなどの制約にも気づかれるかと思います。これらは見積もり段階で、ある程度把握できますが、「1プロセッサの検証機からグレードアップして本番機は2プロセッサ構成にしたものの性能が思ったほど出ない」といった経験をされている方も見かけます。次章以降では、これらの原因や理由、そうなった経緯などについて見ていくことにしましょう。SD

注2) 日本ヒューレットパッカードが公開しているシステム構成図(http://h50146.www5.hp.com/products/servers/proliant/sh_system.html)は、サーバの構成を検討する立場の人なら見たことがあると思いますが、日本国内向けのみで海外にはないのだそうです。日本人で良かったですね!

▼図2 Webサイト上での構成見積もり

第1章 プロセッサの見方

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

コンピュータの心臓部といえばプロセッサです。プロセッサはしばらくの間、おもにクロック周波数の高速化により性能を押し上げてきましたが、ここ10年はマルチコア化によりプロセッサあたりの演算性能を稼ぐ方針に転換しています。x86の歴史を簡単に振り返りながら、これまでの経緯を確認し、イマドキなプロセッサの特性について考えてみましょう。

x86の系譜と押さえるポイント

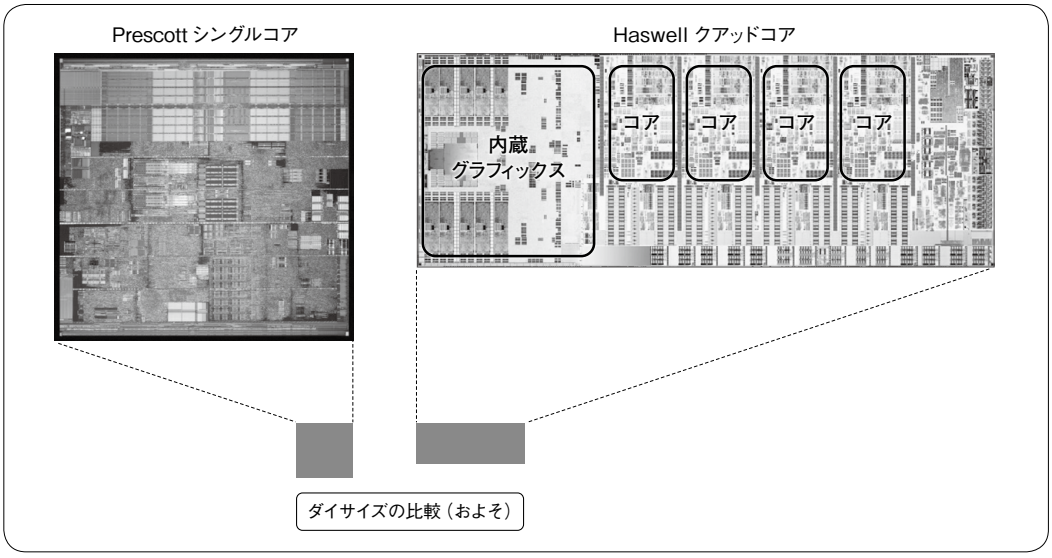
8086の登場以来、x86系プロセッサは半導体の製造プロセスの微細化、クロック周波数の向上、そしてマイクロアーキテクチャのアップデートにより、互換性を維持したまま性能を上げてきました。とくにここ10年ではマルチコア化など大きな変化が生じています。

NetBurstアーキテクチャと発熱量の限界

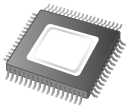
ちょうど今から10年ほど前、NetBurstアーキテクチャと総称されるPentium4（2000年～）では、クロック周波数を引き上げることで性能を稼ぐ戦略のアーキテクチャが採られており、現在のプロセッサとほぼ遜色ない3GHz台後半までたどり着きました。2004年に発表されたPrescott（**図1**）では、トランジスタ数はついに1億2500万に達しました。

しかし、プロセスの微細化が進み、高性能なプロセッサを作ろうとすると、より小さなスペースに多数のトランジスタを押し込むことになり、発熱が集中することになります。発熱を減らすためには、プロセッサの駆動電圧を下げる必要が

▼図1　PrescottとHaswell



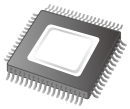
ありますが、Prescottの時代になるとプロセッサの稼働電圧を下げても漏れ電流が減らず消費電力が下がらないという問題が立ちはだかり[注1]、クロック周波数に頼ってプロセッサの性能を向上させることが技術的に難しくなりました。



時代はマルチコアへ

最近では、制御効率がよいトライゲートトランジスタの採用により漏れ電流を削減できるようになったそうですが、ここ10年はクロック周波数は無理のない程度に据え置き、マイクロアーキテクチャの改善によりクロックあたりの処理性能を向上させたり、プロセッサ内に複数のコアを実装して命令実行の並列度を上げたりする戦略で性能向上を果たしてきました。2013年に発表されたHaswellのクアッドコアモデル(**図1**)では、177平方ミリメートルのダイ上に14億ものトランジスタが実装されています[注2]。

現代のx86プロセッサでは、デスクトップ向けのプロセッサであれば、4〜6、サーバ向けでは12ものコアを搭載しています。基本的にはコア単体の処理能力も向上し続けていますが、現在のプロセッサの性能を最大限に活かすためには、プロセッサ内にある複数のコアをワークロードでうまく使いきることがカギとなります。



マルチコアとマルチプロセッサ

プロセッサの性能は、世代(マイクロアーキテクチャの効率)、プロセッサ内のコアの動作速度、そしてその多重度であるコア数によりトータル性能が決まります。

最近はマルチコアのプロセッサが一般的になりました。デスクトップやサーバでは4つ、もしくはそれ以上搭載しており、プログラムを同時に複数実行できます。さらにマルチプロセッサ構成とすると、コンピュータで同時に複数のプロセッサを稼働させることで並列度を増やすことができます(**図2**)。

実際のところ、クロック周波数が高いほうがよいか、コア数が多いかほうがよいかはアプリケーションに依存します。アルゴリズムの都合上、もしくはプログラムの内部的な並列度が低いためにコア間の分散が効かないようなワークロード向

注1) 筆者自身もPrescottの3.4GHzのプロセッサを使用していましたが、消費電力も発熱も非常に多いプロセッサだったという印象を強く憶えています。また、客先で導入されたPentium4 3.6GHz搭載のパソコンについてもファン音や熱に関して評判がよくなかったのを思い出します。

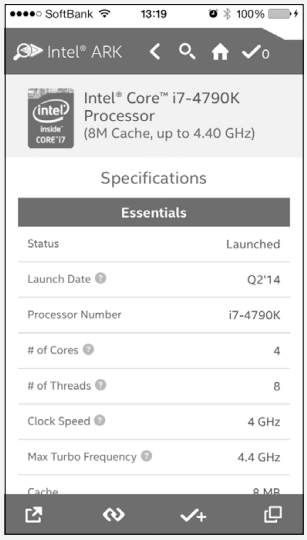
注2) Intel Reveals New Haswell Details at ISSCC 2014(http://www.anandtech.com/show/7744/intel-reveals-new-haswell-details-at-isscc-2014)

Column Intel™ ARKアプリをインストールしよう!

打ち合わせ中に相手がポロっと口にしたプロセッサのスペックを確認したい時、店先でプロセッサの型番から詳細スペックを知りたい時、また通勤中の暇つぶしに82559のデータシート読みたいとき……さまざまなシチュエーションで願いを叶えてくれる便利なアプリがIntel ARKアプリです。

ARKとはIntelが提供する同社の製品データベースであり、Webサイトからアクセス可能なほか、iOSやAndroid向けの「ARKアプリ」が用意されています(**図A**)。ARKアプリは起動時に最新のカタログをダウンロードするほか、ネットワーク接続されていない場合でもキャッシュ済みのデータを閲覧できます。

▼図A Intel™ ARKアプリ



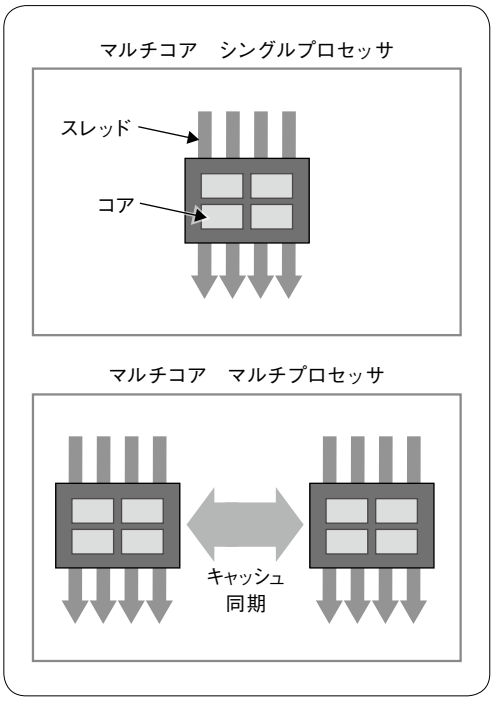
けでは、コアの動作クロックが高いほうが性能が伸びたりします。逆にマルチスレッドを意識して作られていれば、コア数が多いほうが性能が伸びたりもします。

多コアモデルと高クロックモデル、どちらを選ぶ?

クロック周波数が高く、コア数が多いプロセッサを選ぶ、といっても、クロック周波数とコア数の関係は若干ですが反比例の関係にあります。コア数が多いモデルではクロック周波数が若干劣り、クロック周波数が高いモデルではコア数が少ない、といった形のラインナップから、手許のワークロードに向いたプロセッサを選ぶことになります。**図3**は、Ivy Bridge EP(Xeon E5-2600 v2)シリーズにおけるクロック周波数とコア数の関係です。

Web 2.0などの利用用途で、同時に多数のアクセスに対してレスポンスするようなワークロードでは、コア数が多いほうが有利と言われます。

▼図2　マルチコア、マルチプロセッサ



少ないコアを速く動かすよりも、コアが多いモデルのほうが、プロセッサが処理できる量は多い傾向にあるからです。

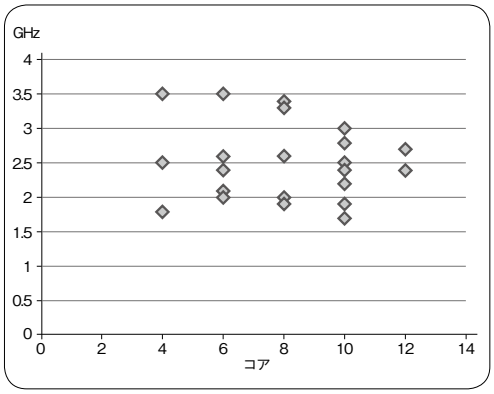
直列処理されるバッチジョブが多い場合など、ワークロードの並列度が確保できず、一部コアに負荷が偏ってしまうことが予期される場合は、コア数を必要以上に増やすのではなく、クロック周波数が高いプロセッサを選んだほうがよいかもしれません。

ミドルウェアもマルチコアを意識したチューニングへ

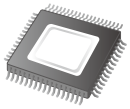
図4、**図5**はPercona社によるMySQL各種バージョンのベンチマーク結果[注3]です。**図4**では、特定のワークロードをさまざまなスレッド数で実行したときの性能を比較したものです。このグラフではMySQL 5.1以前(MyISAM)では8スレッドを超えるとあまり性能がスケールしていないことが読み取れます。対象的に、現在主流のInnoDBストレージエンジンでは32スレッドの条件時に性能がピークとなっており、MySQL 5.1単独のときよりも約1.7倍のスループットが向上したことがわかります。次に**図5**を見てみましょう。こちらは、先のグラフから1スレッド(シングルスレッド)時の性能を切り

注3) 引用元:MySQL Performance Blog(http://www.mysqlperformanceblog.com/2011/10/10/mysql-versions-shootout/)

▼図3　Ivy Bridge EPシリーズにおけるクロック周波数とコア数の関係



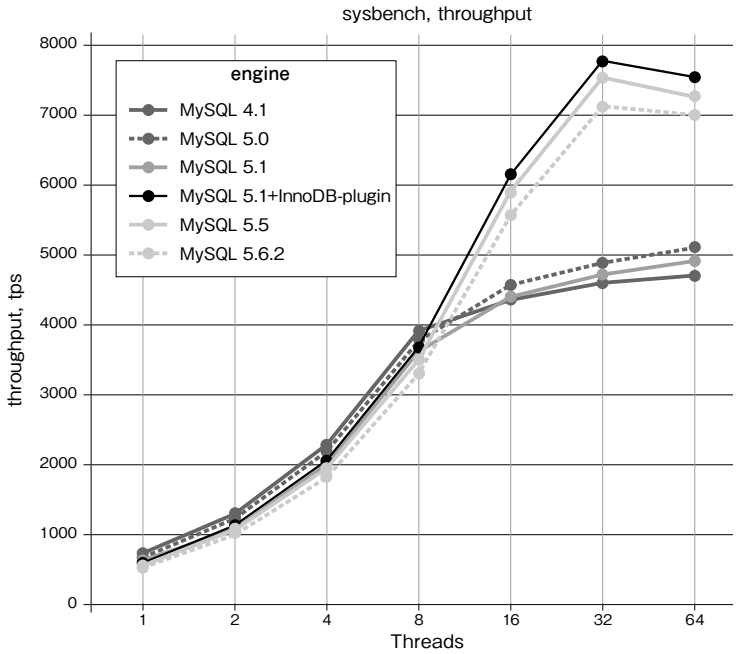
出したものとなっています。興味深いことに、MySQLのバージョンが上がるにつれて、シングルスレッドあたりの性能は下がりつつあります。このベンチマーク結果から、MySQLはバージョンアップとともに、シングルスレッドスループットよりも、現在主流のマルチコア環境で利用した場合の総合スループットを高めるようにチューニングが変化していることが伺えます。



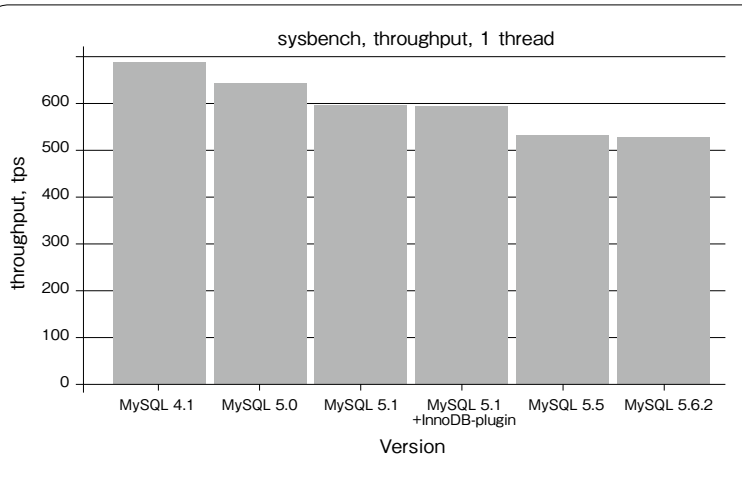
性能以外の要件にも注意

2ソケット以上のサーバモデルでは、サーバ内のPCI Express拡張スロットやDIMMスロットの利用条件として、マルチプロセッサ構成をとる必要があります。理由については次の章以降にて述べますが、搭載するDIMMや拡張スロットの利用予定から、想定するプロセッサ構成で問題がないか、忘れずに確認してください。SD

▼図4　MySQLバージョン間のスループット比較（スレッド合計）



▼図5　MySQLバージョン間のスループット比較（1スレッド）



第2章 システムメモリ

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

システムメモリ(一次記憶)は、コンピュータを構成する要素の中ではプロセッサの次に重要な存在です。実行対象のプログラムや処理対象のデータは、システムメモリ上に置かれている必要があります。本章では、システムメモリの位置づけやDIMMの仕様の読み方、メモリの構成を決める際に知っておきたいポイントを説明します。

メモリの基礎知識

システムメモリ、レジスタとキャッシュ

システムメモリは必要な分のDIMM(Dual Inline Memory Module)をシステムボードに装着して利用します。しかし、DIMMの応答速度はプロセッサからすると1～2桁違い、非常に低速なため、効率を上げるためにプロセッサ内のキャッシュ(Cache)が併用されています。

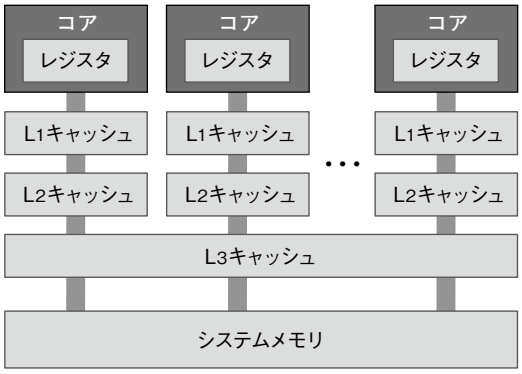
システムメモリのほかにも、プロセッサには、ソフトウェアが演算に利用できる記憶素子であるレジスタ、またシステムメモリとの速度差を隠蔽するためのキャッシュメモリを備えています。

レジスタとは、プロセッサ内に用意されているメモリ部分で、システムメモリの読み書きよりも圧倒的に高速であり、プロセッサの状態や計算途中の値を保持するために利用されます。レジスタの一部は利用用途があらかじめ決まっていますが、現在のIA32(64bit)では16本の汎用レジスタ(General Purpose Register)が備わっており、ソフトウェアが自由に利用できます。

メモリの階層構造

プロセッサの処理スピードからみればDIMMの応答速度は非常に遅いため、このギャップを隠蔽するべく低密度ながら高速にアクセス可能なSRAM(Static RAM)によるL1、L2、L3キャッシュが備わっています(**図1**)。

▼図1　メモリの階層構造と応答速度の例



L1キャッシュ	1～2サイクル前後	32KB/コア
L2キャッシュ	4サイクル前後	256KB/コア
L3キャッシュ	24サイクル前後	10～30MB
システムメモリ	50ns～	1GB～

プロセッサは、システムメモリを直接扱うのではなく、キャッシュを通して扱うことで高速に動作します。しかし、キャッシュに載っていないデータへのアクセスが発生すると、システムメモリからキャッシュにロードする間、プログラムの実行が停止してしまいます。

メモリアロケータやカーネルの開発者[注1]、プロセッサを利用効率を上げるための本格的なチューニング作業を行っている方などでなければ、これらのキャッシュを意識している方はそれほど多くないでしょう。しかし、とある調査によれば、OLTP(Online Tran saction Processing)ワークロードの実行の75%はメモリアクセス中のブロックによるものと示されており、メモリアクセスがプロセッサの命令処理時間に大きな影響を及ぼしていることがわかります[注2]。

UMAとNUMA

x86アーキテクチャでも、2007年に発売されたHarpertown(Xeon 5400シリーズ)までは、対称共有型メモリアクセス(UMA)の構成をとっていました。しかしUMAでは1つのバス上に複数のプロセッサが存在し、メモリにつながるバスを取り合うため、プロセッサ数が増えてくるとうまくスケールしません。

2008年のNehalem(Xeon 5500シリーズ)からはNUMA(Non-Uniform Memory Access:**図2**)の構成に代わり、現在に到ります。NUMAでは、各ノード(プロセッサ)がローカルメモリを持ち、プロセッサ間がインターコネクトで相互接続された状態となっています。また、ノードAがノードBに接続されているメモリにアクセスする必要がある場合には、間のノードを介して目的のメモリにアクセスすることができます。

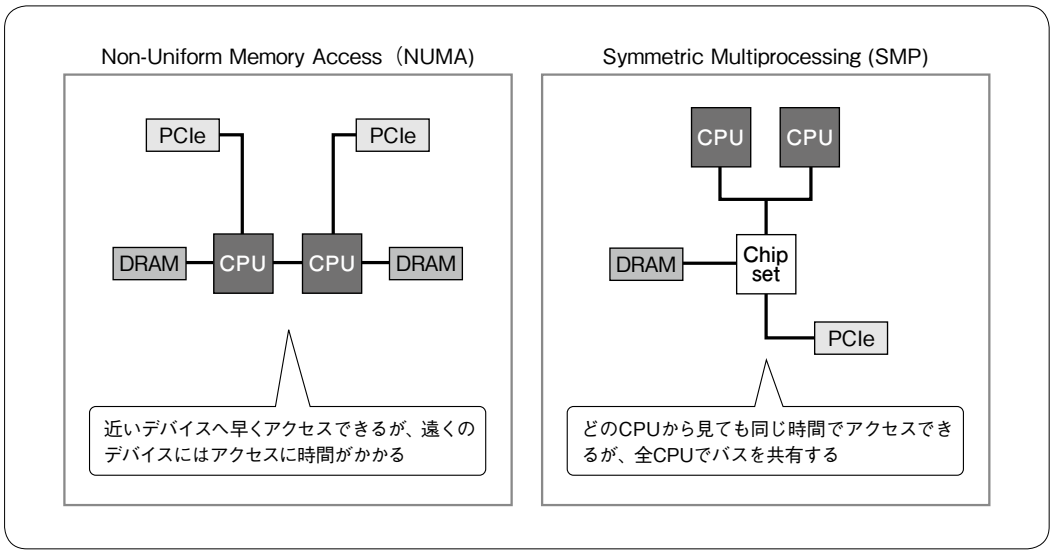
一般的に、プログラムがアクセスするメモリ範囲には偏りが生じますので、NUMAの構成にすることで頻繁に利用されるデータをプロセッサの側に置け、高速に処理できるようになりました。また、プロセッサを追加することで、より多くのメモリチャンネルを搭載できるというメリットもあります。

現在のIntelのプロセッサでは1基あたり12枚のDIMMを接続でき、数百GBのシステムメモリを利用できます。マルチプロセッサ構成ではプロセッサの数だけ多くのメモリを接続でき

注1) 最近、この手の方が本誌の売上に大きな影響を及ぼしているそうで:-)

注2) Memory System Characterization of Commercial Workloads : http://www.hpl.hp.com/techreports/Compaq-DEC/WRL-98-9.pdf

▼図2 SMPとNUMA



ますので、4プロセッサのサーバにもなるとTB級のシステムメモリを搭載できます。

NUMAではプロセッサからアクセスするメモリ領域により処理速度が変わるため、プロセッサやメモリコントローラの変更だけで実現できるデザインではなく、ソフトウェアもNUMAを意識した動作が求められます。たとえば、オペレーティングシステムにはプロセスが使用するデータを特定のノードのメモリに集める、データの近くのノードにプロセスをスケジュールするといった工夫が必要となります。また、ソフトウェア開発者にとっては、メモリレイアウトやスレッドのデザインで大きな速度差が出るという課題も同時に生まれました。

メモリのキャッシュコヒーレンシ

あるコアがシステムメモリの内容を書き換えた際、他のコアにも矛盾なく変更が見えなければいけません。しかし、実際のプロセッサの実装としては、各コアが別々にL1/L2キャッシュを持っているため、システムメモリを書き換える際には各コアが持つキャッシュメモリと一貫性を保たなければなりません。これをキャッシュコヒーレンシと呼びます。

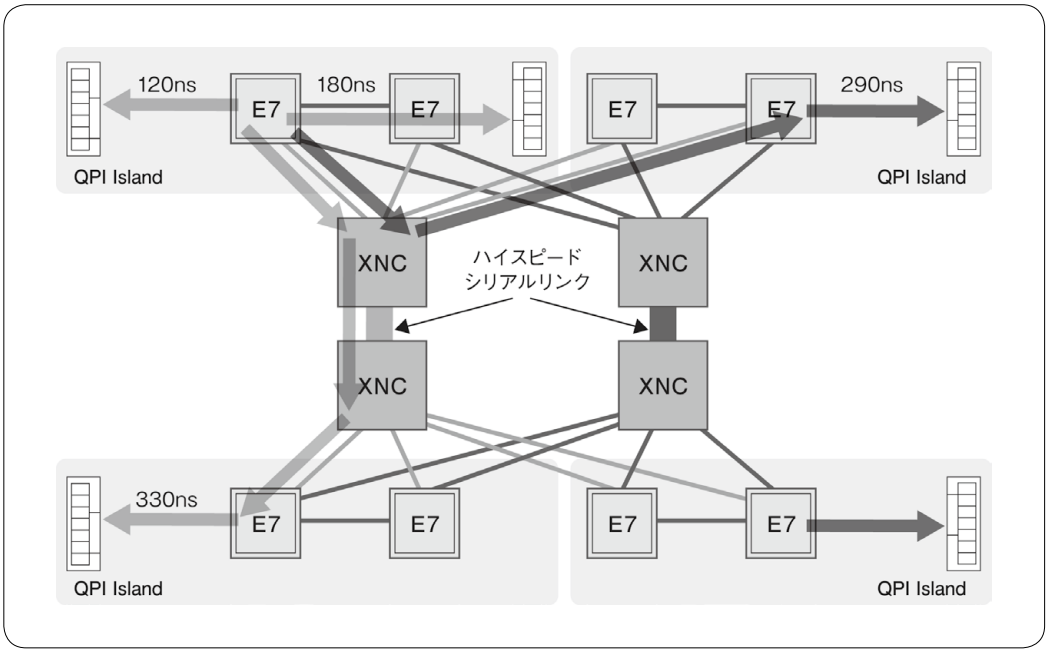
現在のx86プロセッサではL1/L2キャッシュを各コアごとに持ち、L3キャッシュはプロセッサ内のコア間で共有しています。さらに、マルチプロセッサ環境では、より遠くのコアともキャッシュコヒーレンシを保つため、インターコネクトを通じての通信が発生します。

NUMA構成の実例

ここでNUMA構成の具体的なサーバ例を見てみましょう[注3](**図3**)。Hewlett PackardのProLiant DL980 G7は最大8プロセッサまで搭載できます。このシステムでは2プロセッサごとにグループ化されており、さらに4グループが2つのグループにわけられ、疎結合されています。

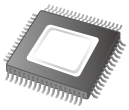
注3) ホワイトペーパー「HP ProLiant DL980 G7でのGaussian09の評価」: http://h50146.www5.hp.com/products/servers/proliant/whitepaper/pdfs/WP_DL980_Gaussian09_0307.pdf

▼図3　HP ProLiant DL980 G7のNUMA構成とメモリレイテンシ



このシステムでは、プロセッサがメモリにアクセスする際、アクセス先のメモリが、そのプロセッサに接続されたローカルメモリであれば120nsでアクセスできますが、インターコネクトを通じてすぐ隣にあるプロセッサのメモリへは180nsの時間がかかります。さらに、隣のグループのメモリへは290nsとなっており、ローカルメモリと比べると2.4倍も応答速度が遅くなり、一番遠いメモリへのアクセスには300ns以上の時間がかかり、ソフトウェアのNUMA対応の重要性がわかります。

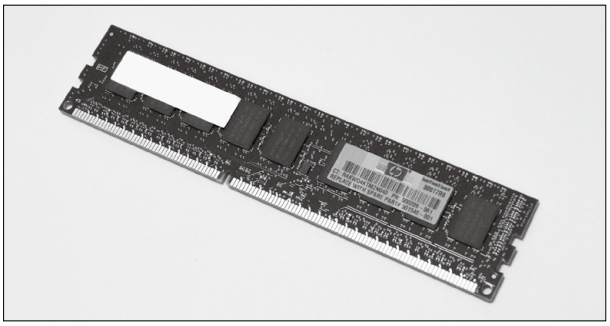
上記からもわかるとおり、マルチプロセッサ構成は多コア化ができますがメモリのアクセス速度の観点ではペナルティにもなり得ます。現在ではx86プロセッサでも最大12コアまであり、ひと昔前と比べると並列度が非常に高くなっていますので、NUMA構成にせずとも要件を満たせる場合は、まずはシングルプロセッサでの構成を検討するとよいかと思われます。



DIMMの規格と選定方法

現在よく使われるのがDDR3 SDRAMと呼ばれるメモリモジュールです(**写真1**)。SDRAMとは(Synchronous Dynamic Random Access Memory)を意味します。SDRAMのあと、メモリモジュールへのクロックの立ち上がり／立ち下がり両方で稼働するよう拡張されたDDR SDRAM(Double Data Rate SDRAM)が登場しました。DDR SDRAMは、SDRAMと同じクロック周波数でも倍速で動作するため、SDRAMの2倍のスループットが得られます。そのあと、DDR2、DDR3、そして最近になり市場に流通しはじめたDDR4まで、世代ごとにスループットは倍増してきました。

▼写真1　DDR3 DIMM



DIMMの仕様の読み方

DIMMの規格はPC3-12800やDDR3-1600などの表記で表され、メモリモジュールのデータ転送レートおよび最大スループットを示しています。PC3-12800と表記されたDIMMの場合、そのメモリモジュールはDDR3のSDRAMであり、理論上の最高転送速度は12800MB/s、つまり12.8GB/sの転送速度であることを示します。PC2-6400であれば、そのDIMMはDDR2 SDRAM、6.4GB/sとなります。DDR3-1600といった表記では、DDR3 SDRAMが1600MHzで駆動していることがわかります。

サーバに装着するDIMMは、サーバに合わせて選ぶ必要があります。DDR、DDR2、DDR3などはすべて同じ物理形状をしていますが、駆動電圧が違うなど、仕様に互換性はないため装着時には注意が必要です。DDR3-1600とDDR3-1333といった、同じ世代のメモリでもクロック違いであれば互換性がありますが、動作クロックはメモリチャンネル上で最も遅いDIMMのクロックに合わせられます。

RegisteredとUnbuffered

DIMMを選定する場合にRegisteredタイプとUnbufferedタイプ、もしくはこれらの記載がないDIMMを見かけることがあります。Registeredタイプは、メモリ内部のアドレス信号とコントロール信号を内部でバッファリングを行い、信号の安定化とタイミングの補正をするメモリです。

またUnbufferedタイプは、上記のバッファリング機能に対応していないメモリを指します。とくに明記がない場合

はUnbufferedのものと考えて、差し支えありません。

ユーザにとってRegisteredとUnbufferedの最大の違いは、システムボード側の仕様によって対応するメモリが違う点です。Registeredメモリを必要とするシステムボードではRegisteredタイプのみ利用でき、逆にRegisteredタイプに対応しないシステムボードではUnbufferedタイプのみが利用できます。DIMMを入手する際には、搭載先のシステムがRegistered対応か、非対応かを確認しなければなりません。

チャンネルとDIMMの枚数

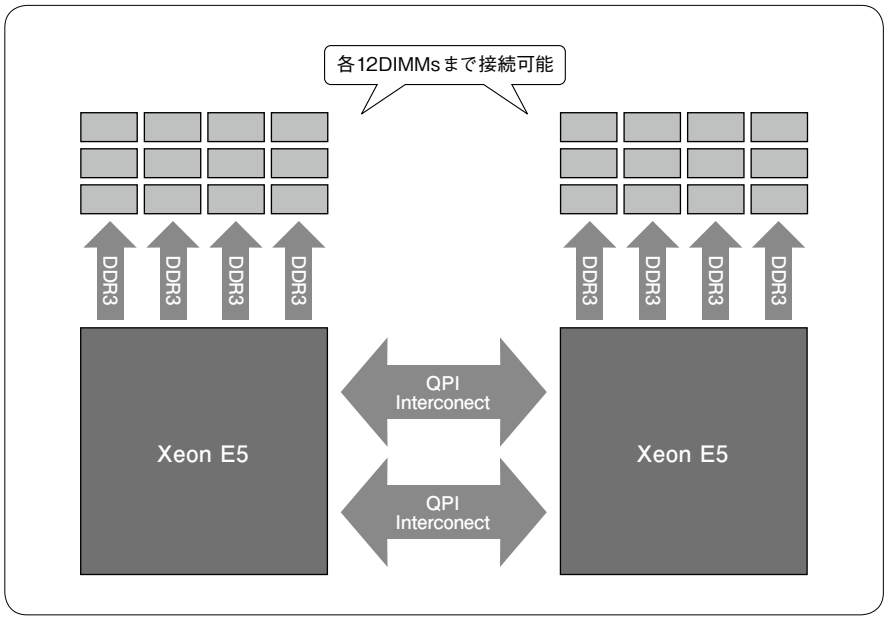
Xeon E5プロセッサの場合、3枚までDIMMを接続できるチャンネルが4本あり、最低1、最大12のDIMMを扱えます(**図4**)。このためプロセッサあたり12個のDIMMスロットがサーバにあることがほとんどです。また、各チャンネルにDIMMを挿しておくと、インターリーブアクセスにより、スループットが向上します。

逆に、同じチャンネル内にDIMMを複数搭載すると、メモリのクロック(アクセス速度)が低下します。たとえばDIMM 1333MHzで動作できるサーバとプロセッサをそろえても、チャンネル内の全3ソケットにDIMMを搭載すると動作クロックが800MHzに落ちてしまう、といったことが起こります[注4]。このため、たとえば4GBのDIMMを12枚挿した場合と、16GBのDIMMを4枚挿した場合では、スループット的には後者のほうが高くなります。

メモリ容量が足りなくなっては本末転倒ですし、あまり神経質になる部分ではありませんが、常にプロセッサとメモリの帯域で勝負する科学技術計算のようなワークロードでは、枚数を最小限に抑えられるか気を付けるとよいでしょう。DIMMが処分セールになっていたり、古いマシンで余っているから、といった安易な理由で小さな容量のDIMMを大量に搭載すると、メモリアクセス性能が低下してしまいます。できれば容量が大きいDIMMを積んで、一部ス

注4) この仕様については各社のサーバの技術資料などから読み取るか、サーバベンダや販売会社などから情報を入手できます。

▼図4　Xeon E5のメモリチャンネル構成



ロットは空けておいたほうが、メモリの動作速度を保ったまま、あとに増設する余地も得られます。

Swap Insanity

現在のオペレーティングシステムは、NUMAの構成を検出してメモリやコアの割り当てを最適化しようとします。プロセスが必要とするメモリができるだけノードをまたがないように割り当て、またプロセスをスケジュールする際にはそのノード上のコアを優先的に割り当てて、リモートノードへのアクセスによるオーバーヘッドを回避します。

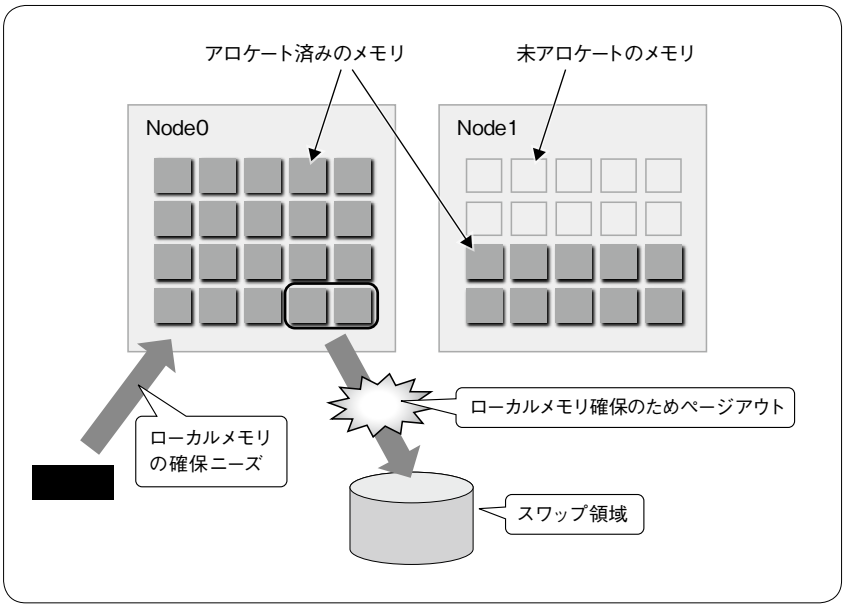
しかしその代償として、ノード間でメモリの消費量が偏る、といった状況も発生します。Linuxの場合、特定ノード上のメモリが足りなくなった状態でローカルメモリを割り当てようとした場合に、そのノードのメモリ上にあるプロセスのイメージをスワップ領域にページアウトし空きメモリを強制的に確保することがあります(**図5**)。

どちらかと言えばOSの実装上の問題ですが、一見メモリが足りているにもかかわらずページアウトが発生し、性能が大幅に劣化することからシステム性能に大きな影響を与えます。この問題はSwap Insanityとして知られています。

この問題には、メモリを大量に使用するプロセスのメモリ割り当てポリシーをインターリーブに設定したり、スワップ対象にならないHugePageとしたり、OSのNUMA対応を無効化して対処されることが多いようです。ほかにもスワップデバイスをなくしたり、vm.swappinessをゼロに設定しページアウトが発生しにくいようにする方法をとることもありますが、できれば避けたい手段です[注5]。SD

注5) vm.swappiness=0に設定したサーバでデータベースに負荷がかかると、メモリ消費が突発的に増えた際にデッドロックし、DBミドルウェアがブロック状態となり、動作を停止するデータベースサーバを見たことがあります。カーネルがスワップしようとしているときは何らかの理由でメモリアロケーションに困っている場合ですので、スワップをさせないという対応も、新たな問題を生む可能性があります。

▼図5 Swap Insanity



第3章 PCI Express

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

PCI Expressは2002年7月にPCI-SIGにより規格化された、汎用的なインターフェースです。現在ではパソコンやサーバだけでなく、スマートフォンなどのさまざまな組み込み機器のプロセッサでもサポートされており、ほとんどのコンピュータで利用されています。本章では、PCI Expressの特徴、およびサーバでの構成時の着目ポイントについて紹介します。

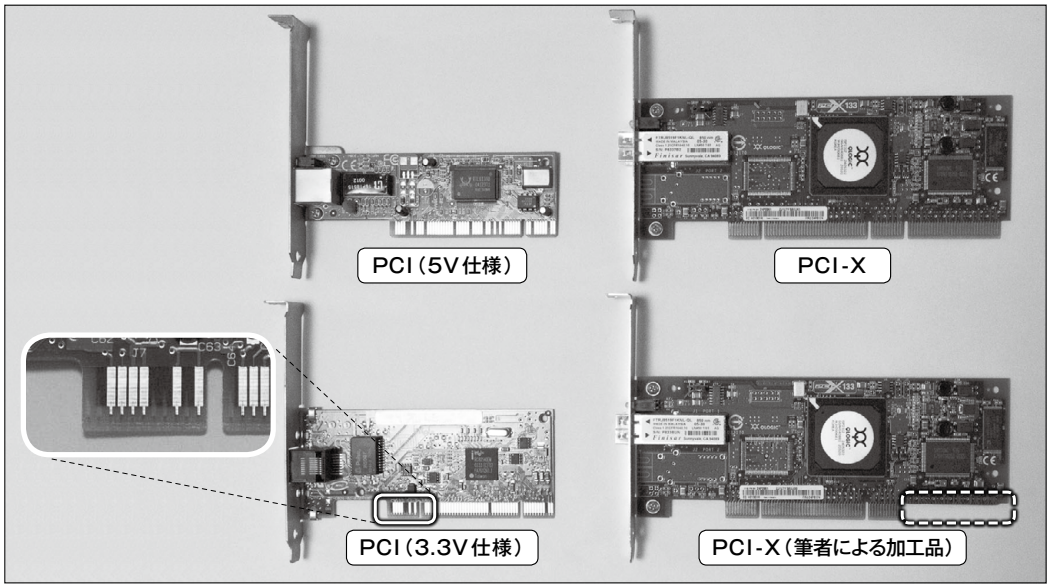
PCI Expressが使われるようになったのはここ10年ぐらいで、それ以前は20年以上、ISAバス（ATバス）やPCIバス、そのほかいくつかのバス規格が使われてきました。まずはISAバスやPCIバスについて振り返り、PCIeの特徴について見ていくことにしましょう。

ISAバス時代

ISAバスは、16bitのパラレル転送によりプロセッサとデバイス群を接続します。ISAバスでは、歴史的理由によりI/Oポートや割り込みチャネルやDMAチャネルの割り当てがマジックナンバーで決まっています。ISAバスの時代は、ハードウェア構成にあわせてOSにパラメータを指定したり、デバイスを追加する場合などにはほかのデバイスとリソース競合が起きないようにやりくりする、などの苦労がありました[注1]。

注1）筆者が1997年に構成したPentium Proベースのマシンでは、当初、CD-ROMドライブで音楽を再生しているとごくまれにシステムがクラッシュする問題がありました。調べてみるとISAのサウンドカードとのリソースの競合が原因でした。

▼写真1　PCI、PCI-Xの拡張カード各種



PCIの登場

1991年に登場したPeripheral Component Interconnect(PCI)は、登場当時の仕様で動作クロック33MHz、133MB/sのパラレル転送により、それまでより高速なI/Oができました。性能面以外でも、PCIデバイスにはPCI configuration Spaceと呼ばれるレジスタがあり、この情報をもとにBIOSやOSがデバイスのリソース割り当てを自動調整できるようになりました。マニュアルでリソースの調整がいらなくなったのはISAからの大きな前進でした。PCIのデバイスは、今でも、ひと昔前のコンピュータなどで見かけるかと思います(**写真1**)。

PCIの高速化と拡張規格

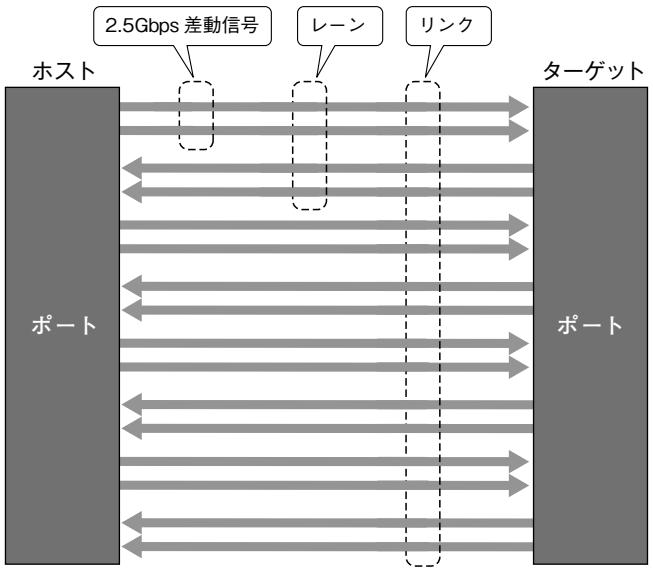
PCIバス時代末期には、66MHz化およびバス幅の拡張により533MB/sでの転送が可能になっていましたが、さらにPCIとは独立して動作クロックを133MHzに引き上げ、バス幅を64bit化したPCI-Xが出てきました。PCI-Xでは1.06GB/sの転送速度があり、サーバでは、RAIDカードやネットワークアダプタなどの負荷がかかりやすいデバイスでPCI-Xを見かけました。PCIとPCI-Xには互換性があったため、PCI-Xの拡張カードは物理的干渉がなければ[注2]PCIバスに接続でき、逆にPCI-XバスにPCIデバイスを接続することもできます。

PCIまたPCI-Xは、動作クロックの引き上げおよびバス幅の拡張により転送速度の向上を図りましたが、のちに登場したPCI Expressにより置き換えられていくこととなります。

PCI Expressのしくみ

PCIeはシリアル転送方式を使用した、PCIの後継インターフェースです。PCIe 1.0 x1では2.5Gb/s、x16では40Gb/sの論理転送速度を持ちます。PCIやPCI-Xではパラレル転送を使用しており、多数の信号線を使ってデータを転送しますが、クロック周波数が上がれば、配線長の差により信号の到着タイミングにズレ(スキュー)が生じ、回路設計がシビアになります。一方PCIeの場合は、TX(送

▼図1　PCIeのレーン構成



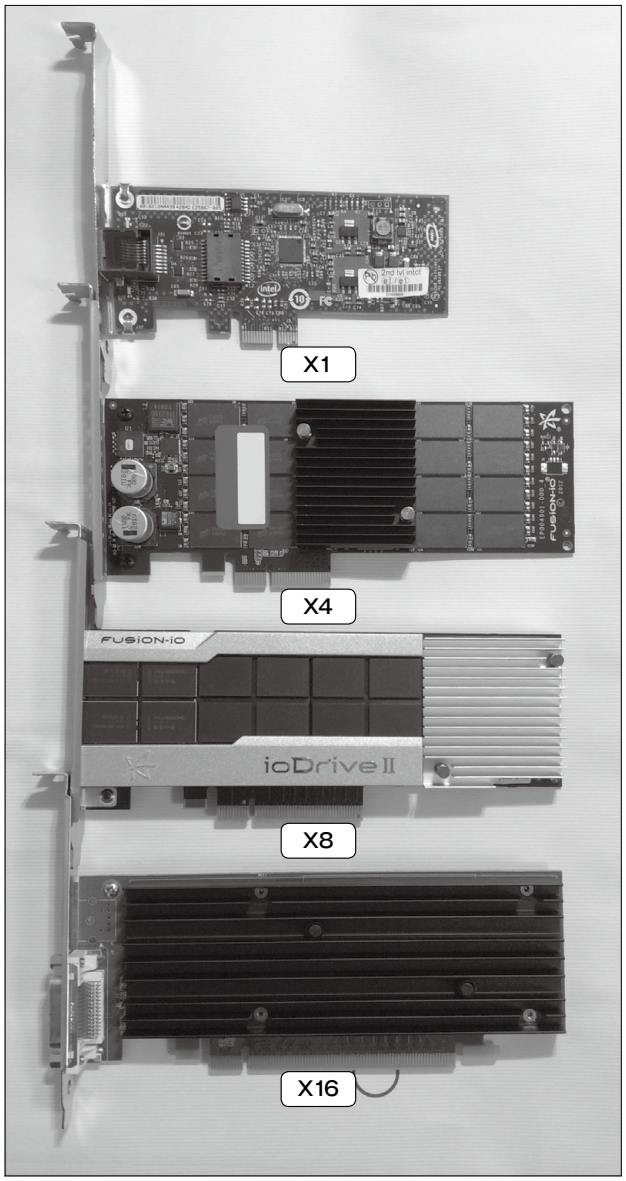
レーン：2.5Gbps差動信号の送受信ペア
リンク：レーンの集まり
ポート：デバイス内にあるリンクを形成するための送受信グループ

注2) ショートなどのリスクがありますのでお勧めできる方法ではありませんが、PCI-XカードをPCIバスに装着する際に物理的に干渉してしまう場合には、PCI-Xで追加された端子部分をカットしてしまうことで認識させることも可能なことがあります。

信)とRX(受信)の差動ペア、合計4本の信号線でホスト(コンピュータ側)とターゲット(デバイス側)を接続し、**レーン**を形成します(**図1**)。PCIeの各レーンは信号線の数が少ないため、転送速度を上げやすいのです。

データを送信する側は、データを複数レーンに分散させて送信し、受信する側は、受信したデータをいったんバッファリングし、組み立て直して元のデータを復元します。このように高速なレーンを複数束ねることで、これまでより広帯域でデバイスを接続できるようになりました。さらにPCIe 2.0、PCIe 3.0ではレーン速度が5.0Gb/s、8.0Gb/sとなり、世代ごとに2倍の転送速度を実現しています[注3]。

▼写真2　4種類のPCIe



プロトコルのネゴシエーション

PCIeにはバージョンやレーン数によりいろいろな組み合わせがありますが、カードの物理形状はすべて統一されており、上位互換性を保っています(**写真2**)。たとえば4レーン仕様のPCIeカードを16レーンのスロットに装着すれば論理4レーンで接続されますし、PCIe 2.0仕様のカードをPCIe 3.0仕様のスロットに装着すれば、PCIe 2.0で接続が確立されます。

逆に、PCIe 3.0仕様のデバイスをPCIe 2.0仕様のスロットに装着することもできますし、16レーン仕様のPCIeカードも、(ホスト側のスロット形状が許せば)1レーン仕様のPCIeスロットに接続することもできます[注4]。当然、デバイス側のレーン数よりホスト側のレーン数が少ない場合や下位規格でリンクした場合にはデバイスの設計どおりの性能がでないかもしれません。

プロセッサ内蔵PCIeコントローラ

現在のプロセッサで特徴的な部分としては、PCI Expressのコントローラをプロセッサ自体が持っている、

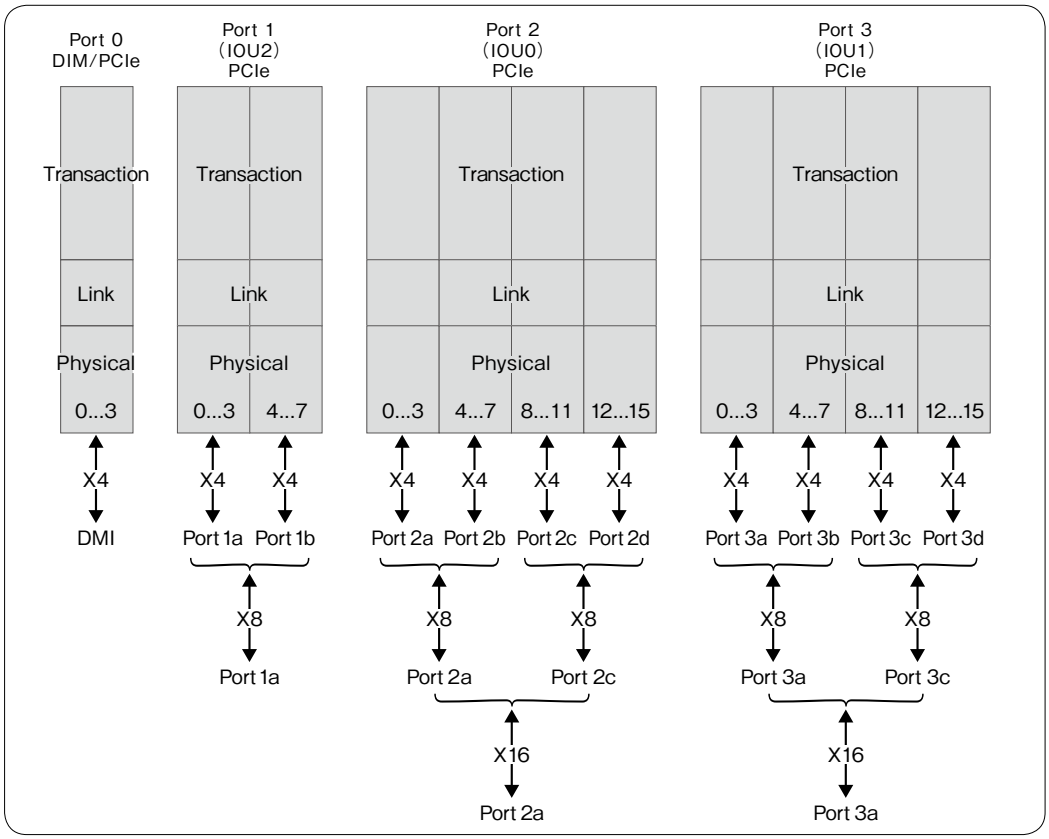
注3) PCIe 1.0/2.0では8b/10bのトランスコーディング、PCIe 3.0では128b/130bのトランスコーディングを使用しており、PCIe 3.0の転送性能は実質的に2倍となっています。

注4) やはりお勧めできる方法ではありませんが、ホスト側のPCIe x1に、ビデオカードの接点部をx16からx1に改造し使用している方もいらっしゃるようです。

ということです。以前のx86では、ノースブリッジ(Northbridge)と呼ばれるチップセット側にPCIeコントローラ機能がありましたが、Sandy Bridge以降のプロセッサでは、メモリコントローラ同様にPCIeコントローラもプロセッサに内蔵されました。

図2はIvy Bridgeプロセッサ内のPCIeの構成です。PCIeデバイスを接続できる数はプロセッサにより違いますが、Xeon E3 v2であれば合計20レーンを決められたパターンで分割し、

▼図2　Xeon E5(Ivy Bridge)のPCIeコントローラのレーン構成



Column バス高度化の功罪

昔はパラレル／シリアルポートを通じてコンピュータと連携するデバイスを簡単に作れましたので、高校時代に学校の体育館の照明設備とコンピュータを接続し、制御するという無茶をやったことがあります。ハードウェアは同級生が担当し、ソフトウェアは筆者が書いて動かしたことがあります。OSの特権保護すらなくハードウェアを触り放題なころでしたのでそれほど難しくはありませんでした。また、ISAバス程度であれば個人でデバイスを設計・実装された方もいたようです。

現在では、ArduinoやRaspberry PiなどI/Oポートが露出しているマイコンボードを使う選択肢もありますが、x86に何かを接続する場合はUSB-Serialアダプタを使ったり、AVRやPICなどのマイクロプロセッサを使ったUSBターゲットとして実装すれば、割と手軽に新しいデバイスを製作、接続できます。PCIeバスに何かを直接ぶら下げるとすると、FPGAやPCIeのIPが必要になるでしょう。

1～3デバイスに接続できます。Xeon E5 v2であれば40レーンあり、より多くのデバイスを接続できます。

最近のx86プロセッサは、システムに1つPCH(Platform Controller Hub)を持ちますが、PCHとの接続バスも実質的にはPCIeです。このため、E5-2600v2/4600v2シリーズでマルチプロセッサ構成を組む場合、第2プロセッサ以降に限りさらに4レーン分のPCIeが利用できます。

拡張スロットまでの道のり

プロセッサから出ている各PCIeインターフェースは、サーバ後部のPCIeスロットに電気的に接続されます。実際にはシステムボード上のRAIDコントローラなどに接続され、拡張スロットが露出していないこともありますし、プロセッサからのPCIeインターフェースとスロットの間にブリッジLSIが挟まれており、インターフェース数やPCIeのバージョン、レーン数などの仕様がプロセッサ側の仕様と一致していない場合もあり、構成時に注意が必要なポイントです[注5]。またマルチプロセッサ対応サーバでは、各スロットに対応するプロセッサが装着されていないと、そのPCIeスロットが動作しないため注意が必要です。

具体的な例としてIBM社のサーバのシステムガイドを見てみましょう(**図3**)。このサーバの場合、スロット1～3は最初のプロセッサに接続されていますが、スロット4～6はふたつ目のプロセッサに接続されているため、利用には2プロセッサ構成とライザーの追加が必須条件となります。ライザーは機器見積りの際にうっかり見落としやすいため、注意しましょう[注6]。

PCI Expressの電力仕様

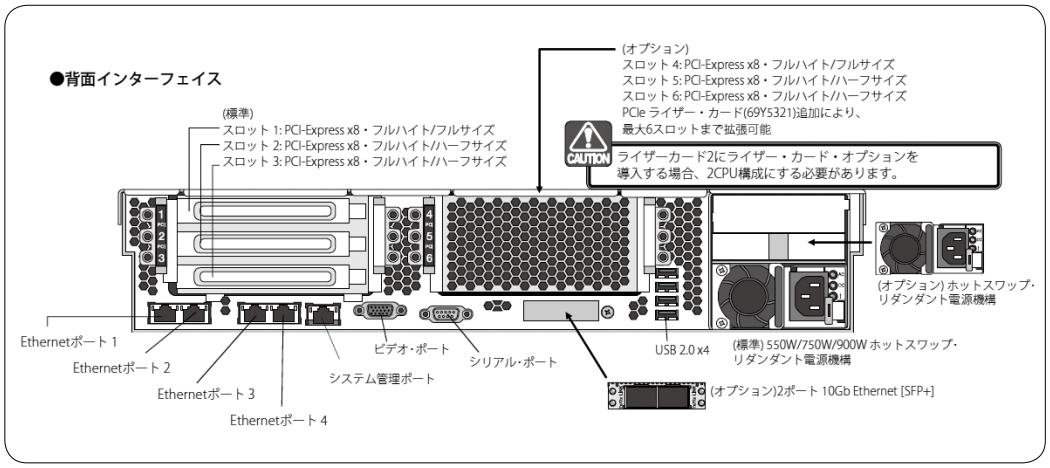
拡張カードは電力を消費しますが、必要な電力は原則としてスロットから供給されます。PCI Expressの技術仕様(**表1**)ではx4以上のスロットは25Wの電力供給が保証されています。経験上、x86サーバのほとんどのスロットがこの仕様に準拠していますが、ごく希(まれ)にこの仕様を満たさないものを見かけたこともあります。

消費電力が大きいGPGPU、大型のPCIe Flash

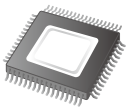
注5) I/O帯域幅が重要となるシステムでは、使用するサーバの構成図を注意深く確認したり、サーバベンダの方にブロック図などを見せていただいたり、想定した構成で期待どおりのスループットが出るか検証するなどの確認が必要になるでしょう。

注6) モデルによってはPCIやPCI-XとPCI Expressを変換するブリッジチップを搭載したライザーも用意されていることがあります。特殊なインターフェースカードを必要とする要件や、ソフトウェアのライセンスがNICのMACアドレスで認証されるシステムで既存のデバイスを継続使用したい場合など、手持ちの資産を再利用したい場合に役立ちます。

▼図3　PCIeスロットついての説明例(IBM社システムガイドより抜粋)



などでは25Wを超える電力を消費する場合があります。この場合は、16レーンフルハイトのスロットから最大75Wの電力供給が仕様で定められているほか、ATX仕様の6ピン／8ピンのプラグから追加電力を供給できる場合がありますが、最終的には基板のデザインに依存します。残念ながら、具体的な電源仕様はスロットの形状をみるだけでは判断がつかない場合がほとんどですが、各スロットが供給可能なワット数が明記されている場合もあります(**写真3**)。PCIeスロットの電源仕様がわからない場合にはサーバベンダなどへ仕様を確認してください。



次回予告

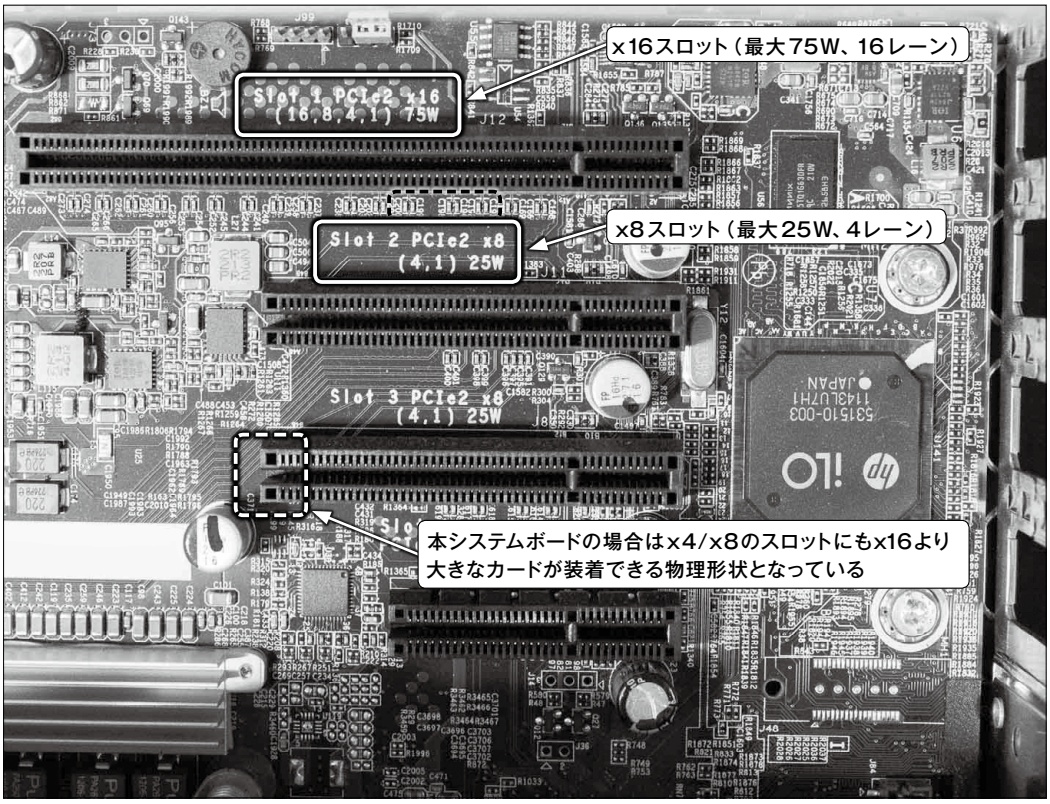
今回はx86サーバの全体的な構成、プロセッサの歴史と特性、DIMM、そしてPCI Expressバスについて説明しました。サーバを構成するにはまだまだ足りないものが多いですね。ディスクなどのストレージやネットワークインターフェースです。次回の後編では、これらのコンポーネントをカバーしていきたいと思います。SD

▼表1　PCIeの電力仕様

	スロットからの最大供給電力		
スロット形状	x1	x4/x8	x16
フルハイト	10W/25W(High Power)	25W	25W/75W(グラフィックカード)
ロープロファイル	10W	25W	25W

※引用元資料 PCI Express Card Electromechanical Specification 1.1

▼写真3　最大消費電力が明記されたPCIeスロット



第10回

イラスト・題字／aico

『鋼鉄都市』（アイザック・アシモフ／早川書房）

SF作家アシモフの代表作といえば、前回紹介した「ファウンデーション」シリーズと、今回紹介する「ロボット」シリーズ。ロボットというと、日本では鉄人28号からマジンガーZやガンダムのような、人が操縦する人型機械まで「ロボット」ですが、「ロボット」という言葉がはじめて登場したカレル・チャペックの「R.U.R」以来、本来のロボットは鉄腕アトムやドラえもんのような自律機械と相場が決まっております。まあ、人型でコクピットがあっても本来のロボットの定義を満たす「翠星のガルガンティア」チェインバーみたいな奴も中にはいますが。

このとおり、ロボットはアシモフの発明ではありませんが、にも関わらずそれがアシモフの代表作になったのは、一連の作品を通してロボットの近代的な定義と意義を与えたからにほかなりません。それが、SFを読んだことがない人でも一度は目にしたであろう、ロボット工学三原則。

- 第一条：ロボットは人間に危害を加えてはならない。また、その危険を看過することによって、人間に危害を及ぼしてはならない。
- 第二条：ロボットは人間にあたえられた命令に服従しなければならない。ただし、あたえられた命令が、第一条に反する場合は、この限りでない。
- 第三条：ロボットは、前掲第一条および第二条に反するおそれのないかぎり、自己をまもらなければならない。

人の役に立ち、人を傷つけず、簡単に壊れない。ある意味理想の道具の定義にもなっているのですが、そんな理想の道具を手に入れたとき、人はどうなるのか？　ロボットという存在を深く考えることは、人という存在を深く考えることと同義であることを示したのがアシモフでした。

その「ロボット」シリーズの最高傑作に挙げる人も少なくない「鋼鉄都市」の世界には、2種類の人類がいます。地球の「鋼鉄都市（cave of steel）」で暮らす80億人の地球人と、超光速航法とロボットを擁し、宇宙を支配するごく一握りの「宇宙人（spacers）」。物語は、スペイサーが地球上で何物かによって殺されたところからはじまります。捜査を命じられたニューヨーク市警の刑事イライジャ・ベイリには、1つの条件が課せられます。それはスペイサーのロボット、R・ダニール・オリヴォーを相棒とすること。やがて捜査線上に1人の人「物」が浮かび上がります。ロボットは三原則に逆らって人を殺すことができるのでしょうか……。

銀河帝国の興亡とはまるで関係なさそうな話に見えるのですが、アシモフはファウンデーションとロボットを後に統合してしまいます。どのように統合した、いや、してしまったのか。次回、アシモフ編最終回をお見逃しなく！

特別企画

オーケストレーションツール Serf・Consul入門 Consul編

前佛 雅人(ぜんぶつ まさひと)
Twitter @zembutsu Web http://pocketstudio.jp/log3/author/admin/

前回はオーケストレーションツールとしてシステム全体に一斉に処理を行う「Serf(サーフ)」の解説を行いました。今回は、サービス単位での検出や監視を通し、オーケストレーションを支援する「Consul(コンサル)」について解説します。

Consul

Consulとは

Consul[注1]は、Serfと同じくHashicorp社によって公開されているツールです。2014年の春に公開されたばかりです。こちらもGitHub[注2]を通して公開されており、IRC[注3]やメーリングリスト[注4]を通して議論が行われています。Consulは、Serfの機能を一部取り込んでいるため、ほぼ同じような操作感で扱うことができます。

ConsulとSerfの違い

なぜSerfではなくConsulが必要になったのでしょうか。それは、ノードを見ているレイヤが異なるからです。Serfは、エージェントを通したノード単位でのメンバ管理を行ってきました。対してConsulは、サービス単位での管理ができるようになります(**表1**)。

Consulはノードの単位でのメンバ管理に加え、Apacheのポートや MySQL データベースといった、サービス単位での管理ができます。あらかじめコマンドで規定しておけば、サービスの監視にも利用できます。さらに、HTTP APIとDNSのインターフェースを通して、監視結果やメンバ情報の取得もできます。ただし、Consulそのものにオーケストレーションツールとして、何かを実行する機能はありません[注5]。

これができると、Serfでは実現できなかったこと、たとえばHTTPの応答に障害を検知した場合、ロードバランサの切り離しや復旧が可能となります。

注1) 公式サイトはhttp://www.consul.io/です。
注2) https://github.com/hashicorp/consul
注3) #consul(freenode.net)
注4) Consul Google Group: https://groups.google.com/group/consul-tool
注5) 原稿執筆時点。ロードマップには、将来的にSerfのeventやqueryの機構を取り入れる予定とあります。

▼表1 SerfとConsulの比較

	Serf	Consul
目的	サービス検出とオーケストレーション	サービス検出と設定
ヘルスチェック	低レベル(ノード死活監視)	サービス単位で高度な調整
キーバリューストア	なし	あり
メンバーシップ	ノード単位	サービス単位
Web API	なし	あり
DNSインターフェース	なし	あり
アーキテクチャ	AP型(一貫性重視、可用性を犠牲)	CP型(可用性より一貫性重視)

※参考:Serf vs. Consul(http://www.serfdom.io/intro/vs-consul.html)

機能

Consulには4つの機能があります。

♪ 1 サービス検出

ノード上で動作するさまざまなアプリケーションを、サービスとして定義できます。サービスの一覧情報はConsulサーバ側で管理します。これらの情報を取得するために、HTTP API、Web UI、DNSの各インターフェースを備えています。具体的には、Consulノード上で定義を行います(**図1**)。任意の対象に対して“api”や“mysql”などの名前を付けることができます。また、サービスはAPIを通して登録することもできます。

♪ 2 障害検知

サービスに対するヘルスチェックを、ノードが定期的に行います。もし、チェックで異常検出など状況の変化があれば、ただちにサーバへ情報が伝えられます。障害の発生状況は、各インターフェースから取得できます。

♪ 3 キーバリューストア(KVS)

Consul内部で使用する領域と、ユーザが任意で利用できるデータ領域があります。HTTP APIを使えば、純粋なKVSとしてConsulを使うこともできます。

♪ 4 複数のネットワークに対応(マルチデータセンタ)

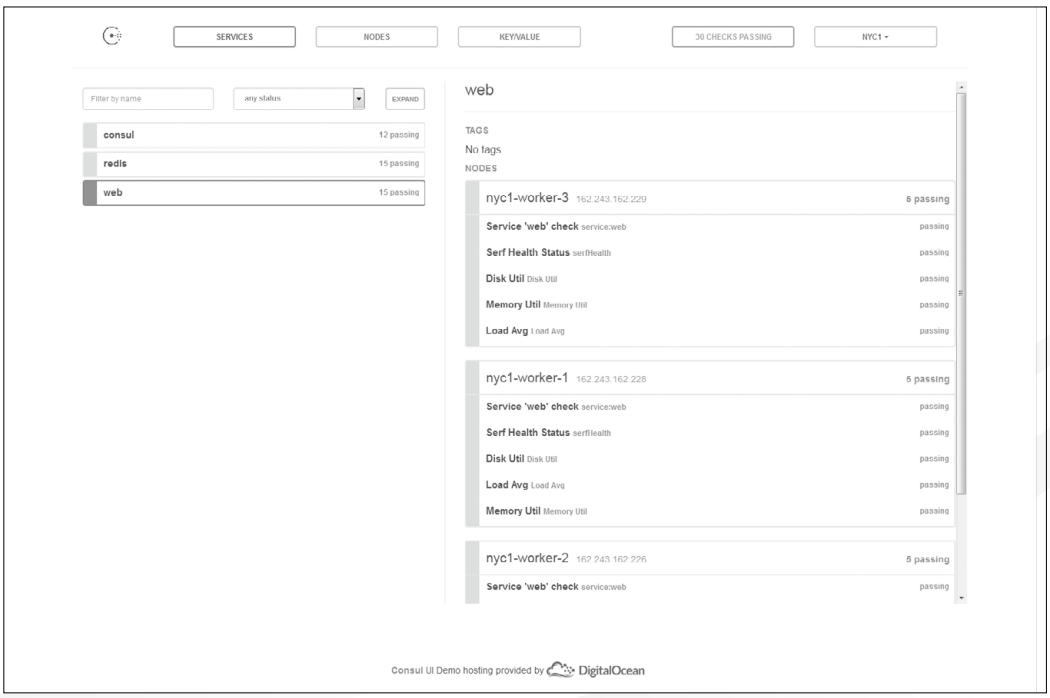
Consulサーバのクラスタは、複数のネットワーク(Consulドキュメントではデータセンタと表現)にまたがって連携できます。特定のサーバに対して問い合わせを行ったとき、対象が自分のネットワーク(Consulのクラスタ)に存在しなければ、外部のクラスタに問い合わせます。

アーキテクチャ

♪ 非中央集権型のクラスタ

Consulのクラスタは、Serf同様に非中央集権

▼図1 Web UI(デモサイト:http://demo.consul.io/ui/)



型のしくみです。“consul”のバイナリ1つで、クライアントとエージェントの役割を持ちます(**図2**)。

♪ Consulサーバ

Consulサーバは Serfにはなかった概念です。Consulサーバには、次の役割があります。

- Consulノードの情報を記録する
- キーバリューストア(KVS)にデータを格納する
- 複数のインターフェースからKVSの情報を返す
- ほかのネットワーク上のConsulサーバと通信する

サーバは1台でも起動できますが、可用性を高めるには3台以上でクラスタを構成する必要があります。また、次のような技術要素でConsulは構成されています。

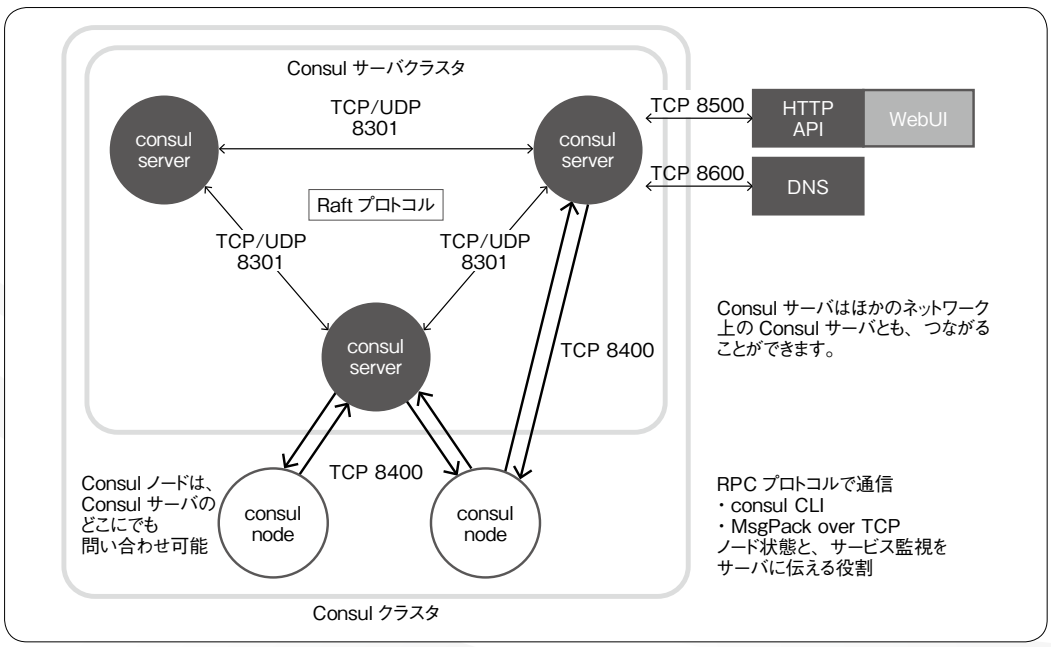
- メッセージング：SWIM、Serf
- リーダー選出：Raft
- セキュリティ：TLS
- データストレージ：UMDP

♪ Consulノード

Consulノードは、Consulサーバの手足のような役割を持ちます。コマンドラインのクライアントとして、サーバに対して命令を出したり、情報を取得できます。また、実際にサービスの検出や、死活状況の確認を行うのはノードです(**図3**)。各々のノードで逐次状況の監視を行い、変化があれば、ただちにサーバ側に伝えます。

一般的なクライアント／サーバ型の監視システムは、サーバ側がノードに対してチェックを行うスタイルが多く見受けられます。一方のConsulは、実際にチェックを行うのはConsulノードです。ノードがcurlやpingなどのコマンドを、あるいはSensuプラグインを実行し、その結果をConsulサーバに伝えます。そのため、ノード単位で細かな監視間隔が設定できるほか、サーバ側での情報把握が迅速に行えます。一方で、個々のノードの監視設定が煩雑であるという課題もあります。

▼図2 Consulのアーキテクチャ



環境構築

動作環境

ConsulはSerf同様に、1つのバイナリファイルで実行ができます。エージェント起動時のオプションで、Consulのサーバになるかノードになるかを選べるほか、コマンドライン用のツールとしても動作します。

♪ Linux版(x86_64)のダウンロードと展開

バイナリはダウンロード用のページ[注6]から取得できます。現在のバージョン0.3では、Linux、Mac OS X、Windowsのバイナリが配付されています。次はwgetを用いた展開例です。

注6) http://www.consul.io/downloads.html

```
$ wget -O 0.3.0_linux_amd64.zip https://dl.bintray.com/mitchellh/consul/0.3.0_linux_amd64.zip
$ unzip ./0.3.0_linux_amd64.zip
# cp ./consul /usr/bin/consul
```

ファイルを設置後は、“consul -v”と入力することで、バージョン情報が確認できます。

Web UIのセットアップ

ブラウザからサービスの状態を見たり、KVSの参照／操作を行う場合は、別途セットアップを行う必要があります。最新のものは、UI用のダウンロードページ[注7]から取得できます。

コードの取得と展開は、**図4**のように行えます。

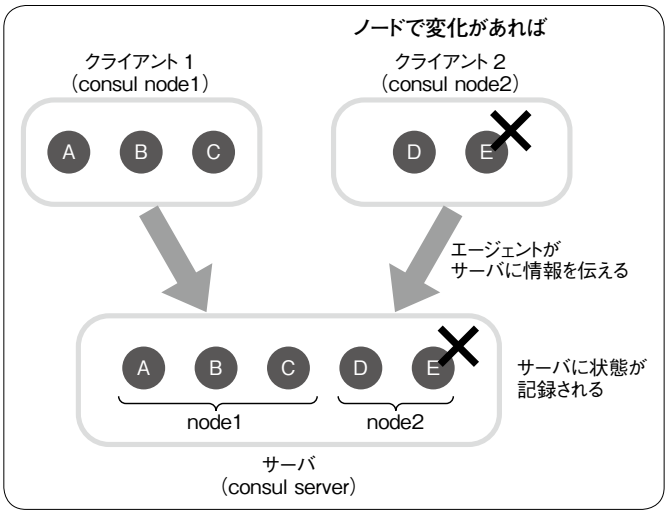
あとは、Consulサーバ起動時に“-ui-dir”オプションを指定します。

```
$ consul agent -server [..略..] -ui-dir=/opt/consul/dist
```

最後に、ブラウザから“http://<ホスト>:8500/ui/”にアクセスすると、WebUIの参照／操作ができます。

注7) http://www.consul.io/downloads_web_ui.html

▼図3 Consulのサービス検出はノードが主体



▼図4 コードの取得と展開

```
# mkdir /opt/consul
$ wget -O 0.3.0_web_ui.zip https://dl.bintray.com/mitchellh/consul/0.3.0_web_ui.zip
$ unzip ./0.3.0_web_ui.zip
# mv ./dist /opt/consul/
```

Consulサーバとクライアントの起動

Consulサーバ

Consulを使うためには、まずConsulサーバを起動させる必要があります(**図5**)。

重要なのは、初回起動するConsulサーバでは“-bootstrap”オプションを付ける必要があるということです。2台目以降のConsulサーバでクラスタを形成する際には、“-server”と“-join”を使い、1台目のサーバを指定します。なお、Consulサーバは1台でも稼働できますが、本来の障害耐性(fault tolerance)を高めるには、3台以上でクラスタを組む必要があります[注8]。

なお、起動すると、次のようなエージェントの状態が画面に表示されます。

- Node name：ノード名(エージェント固有のもの)。“-node”フラグで指定可
- Datacenter：データセンタ名。“-dc”フラグで指定可
- Server：サーバかクライアント、どちらで動作しているか
- Client Addr：HTTP・DNS・RPCインターフェースの情報

注8) 参考：http://www.consul.io/docs/internals/consensus.htmlのページ末尾“Deployment Table”

▼図5 Consilサーバ起動例

```
$ consul agent -server -bootstrap -client=192.168.39.5 -dc=local ¥
    -node=consul1 -data-dir=/tmp/consul  -bind=192.168.39.5
```

▼図6 Consulノード起動例

```
$ consul agent -dc=local -node=consul2 -data-dir=/tmp/consul2 ¥
    -bind=192.168.39.6 -join=192.168.39.5
```

▼図7 サーバやクライアントの一覧と、死活状況を表示

```
# consul members
Node     Address              Status  Type    Build  Protocol
consul3  192.168.39.13:8301  alive   client  0.3.0  2
consul1  192.168.39.11:8301  alive   server  0.3.0  2
```

Consulノード

ノードを起動するオプションは、サーバとほぼ同様です(**図6**)。“-server”オプションがなければ、ノードとしてクラスタに参加します。

クラスタ構成後は“consul members”コマンドを実行することで、サーバやクライアントの一覧と、死活状況を参照できます(**図7**)。

HTTPインターフェース

Consulには複数のインターフェースが実装されています。

HTTP APIを使う

Consulの情報はHTTP経由で取得する方法が、比較的に簡単です。Consulサーバとの通信はJSON形式でデータのやりとりを行います。デフォルトではポート8500が、HTTP用のインターフェースです。処理結果は“jq”コマンド[注9]を併用すると見やすくなります。

図8の例は、ノード情報一覧を表示するものです。

キーバリューストアの読み書き

ConsulのKVSにデータの読み書きを、HTTP経由で行うことができます(**図9**)。

こちらは、キー名称“hello/key”の中に、値“hello, world!”を格納するものです。“true”と表示されれば登録に成功し、“false”であれば登録失敗です。データを取得するときは、“base64”を使ってデコードする必要があります。

注9) http://stedolan.github.io/jq/で配付。CentOSでEPELリポジトリが利用可能であれば、“yum install jq”でセットアップ可能。

さまざまなHTTP API

Consul はRESTful なHTTP APIを持ち、ノードやサービスのチェックに関する設定や、設定変更、削除を行います。現行のAPIのバージョンはv1です。エンドポイントには、次のような種類があります。

・kv ……キーバリューストア
・agent ……エージェント制御
・catalog ……ノードやサービス管理
・health ……ヘルスチェックの管理
・status ……Consul のシステム状態
・session ……セッション管理やロック
・internal ……Consul 内部で使用

♪ キーバリューストア(kv)

"/v1/kv/"に、"GET"、"PUT"、"DELETE"などのメソッドを使い、キーバリューストアとしてユーザが任意に利用できます。ただし、"GET"で取得したデータは、そのままでは読めません。base64でデコードする必要があります。

♪ エージェント(agent)

ローカルのConsulエージェントと連携するために使用します(**表2**)。

▼図8　ノード情報一覧を表示

```
$ curl -s http://192.168.39.5:8500/v1/catalog/nodes  | jq '.'
[
  {
    "Address": "192.168.39.5",
    "Node": "consul1.pocketstudio.net"
  },
  {
    "Address": "192.168.39.6",
    "Node": "consul2.pocketstudio.net"
  }
]
```

▼図9　HTTP経由でのデータの読み書き

```
$ curl -XPUT -d 'hello, world!' http://192.168.39.5:8500/v1/kv/hello/key
true
```

▼図10　キー名称に値を格納

```
$ curl -s http://192.168.39.5:8500/v1/kv/hello/key | jq '.[].Value | .' -r | base64 -d
hello, world!
```

▼表2　エージェント

API	動作
/v1/agent/checks	ローカルエージェントが管理しているcheckを返す
/v1/agent/services	ローカルエージェントが管理しているserviceを返す
/v1/agent/members	ローカルserfエージェントが見えているメンバを返す
/v1/agent/join/←address→	ローカルエージェントがノードにjoinするトリガ
/v1/agent/force-leave/←node→	ノードをforce remove(強制削除)
/v1/agent/check/register	新しいローカルcheckの登録
/v1/agent/check/pass/←checkID→	ローカルテストを通過(passing)したとマーク
/v1/agent/check/warn/←checkID→	ローカルテストの警告(warning)をマーク
/v1/agent/check/fail/←checkID→	ローカルテストの障害(critical)をマーク
/v1/agent/service/register	新しいローカルserviceの登録
/v1/agent/service/deregister/←serviceID→	ローカルserviceの削除

♪ カタログ(catalog)

nodeやserviceなどに対する監視(checks)の登録や削除を行います(**表3**)。

♪ ヘルスチェック(health)

各サービスやノードの状態を確認をします(**表4**)。

♪ クラスタ状態(status)

Consulクラスタに関する情報を返します(**表5**)。

♪ セッション(session)

セッションの作成、破棄、問い合わせに使用します(**表6**)。

DNSインターフェース

使い方

Consulサーバは DNSインターフェースを兼ね備えているため、ホスト名の名前解決を行うことができます(**図11**)。つまり、ホスト名さえ把握できれば、しくみ的にIPアドレスを知る必要がなくなります。なお、逆引きを行うことはできません。

digを使って名前解決する場合は、"-p"でポート番号8600を指定します。FQDNの形式は、<consulホスト名>.node.consulになります。データセンタの指定がある場合は、<ホスト名>.node.<データセンタ名>.consulで名前解決をします。

▼表3 カタログ

API	動作
/v1/catalog/register	新しいnode、service、checkの登録
/v1/catalog/datacenters	既知のデータセンター一覧
/v1/catalog/nodes	指定したデータセンタ上に存在するノード一覧
/v1/catalog/services	指定したデータセンタ上に存在するサービス一覧
/v1/catalog/service/←service→	指定したサービスが存在するノード一覧
/v1/catalog/node/←node→	指定したノード上のサービス一覧

▼表4 ヘルスチェック

API	動作
/v1/health/node/←node→	ノードのヘルス情報を返す
/v1/health/checks/←service→	serviceのヘルス情報に関するcheckを返す
/v1/health/service/←service→	serviceを持つノードのヘルス情報を返す
/v1/health/state/←state→	指定したstate状態にあるcheckを返す

▼表5 クラスタ状態

API	動作
/v1/status/leader	現在のRaftリーダーを返す
/v1/status/peers	現在のRaftピアの一覧を返す

▼表6 セッション

API	動作
/v1/session/create	新しいセッションの作成
/v1/session/destroy/←session→	指定したセッションの破棄
/v1/session/info/←session→	指定したセッションに問い合わせ
/v1/session/node/←node→	特定ノードに紐付くセッション一覧
/v1/session/list:	アクティブなセッションの一覧

図12のようにノード情報を確認できます。

DNSインターフェースの使いどころ

DNSで返ってくるのは「正常」と認識されているノードの情報です。これのしくみを使えば、

・正常なサービスを提供しているマスタ側サーバの指定
・サービスを提供しているホスト一覧の取得
・DNSラウンドロビンによる負荷分散

といった応用ができます。ただし、Consulが提供するのはDNS情報のみです。実際に情報を取得するプログラムや、受け取った結果を処理する「何か」にあたる部分は自分で実装する必要があります。

まとめ

これまで駆け足でSerfとConsulの使い方を追ってきました。Serf・Consulともに開発途上であり、これからもさまざまな機能が拡張される模様です。どちらも、オーケストレーションツールとして、引き続き目が離せません。

ただ気を付けなくてはいけないのは、これらのツールが何かを解決してくれるわけではないことです。純粋に道具であり、どこで使うかは自分で考える必要があるのです。

そして、どちらにも共通しているのは、手軽なのに高機能な点。本来は複雑であるクラスタの同期や処理を、簡単に行える点です。ほかのツールと比べると、導入の敷居が低いのではないでしょうか。もし、身の回りで使えそうなところがあれば、試していただければと思います。

SD

♪ 参考URL

・Serf と Consul の記事まとめ - Qiita

URL http://qiita.com/zembutsu/items/aaffab81f9d5b60d7ecc

・Serf Cheat Sheet 日本語版 - Qiita

URL http://qiita.com/zembutsu/items/1e2cddd0a424ef7a4895

・Consul Cheat Sheet 日本語版 - Qiita

URL http://qiita.com/zembutsu/items/3efb7ebc1d8dba521d3c

▼図11　問い合わせの形式

```
<タグ>.<サービス名>.<ノード名>.node.<データセンタ名>.consul
```

※タグおよびサービス名は省略できます。

▼図12　実行例

```
$ dig @192.168.39.5 -p 8600 consul1.node.consul any

; <<>> DiG 9.8.2rc1-RedHat-9.8.2-0.23.rc1.el6_5.1 <<>> @192.168.39.5 -p 8600 consul1.node.consul any
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 8882
;; flags: qr aa rd; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0
;; WARNING: recursion requested but not available

;; QUESTION SECTION:
;consul1.node.consul.           IN      ANY

;; ANSWER SECTION:
consul1.node.consul.    0       IN      A       192.168.39.5
```

特別企画 IBMがリリースする真打ちクラウド

SoftLayerを使ってみませんか?

ベアメタルサーバクラウド活用入門

第2回　サーバ構築の実際(その1)

前回では、SoftLayerでトライアルキャンペーンを利用して物理サーバを起動するまでを説明しました。今回はSoftLayer上で実際にサーバを構築するうえでのヒントやAPIの簡単な説明、そしてネットワークについて2回に分けて解説します。

Writer 常田 秀明(ときだ ひであき)　日本情報通信㈱　Hideaki_Tokida@NIandC.co.jp　Twitter@tokida

トピックス

ロンドンに続き、8月にトロントデータセンターが開設されました。それに伴いトロントでも$500キャンペーンが開催されています。また、ベアメタルサーバのOS選択で"without OS(OSなし)"が選べるようになりました。独自のOSを導入したいユーザは初期のプロビジョニング時に時間のかかるOSインストール作業時間を待つ必要がなくなりました(筆者も今度利用してみます)。加えて新機能としてAutoScaleの発表がありました。これで負荷に応じてシステムを自動的に拡張できるようになります。

サーバ構築時に利用したい機能

SoftLayerでは、CPUやメモリといったリソースを柔軟に構成できるのも特徴となりますが、それ以外にも構築時に便利に使える機能があるので、これらの説明をします。

クラウドではサーバを新規に作成する作業を「プロビジョニング」と呼んでいますが、この作業を行う際、同時にユーザは任意の処理を実行させることができます。これによってサーバを購入後すぐに目的の機能を利用できます。実際にはプロビジョニング時に、シェルスクリプトや、Chef、Puppetなどのサーバ自動構成ツールを用いることで複雑なシステムを構築できます。

たとえば、最近話題のOpenStackの「ノード」を追加したいと考えた場合、ベアメタルサーバの起動時に「ノード」を構成するためのChefを指定します。これで起動直後から「ノード」として利用可能にできます。簡単なところでは、パッケージ管理のデータベースを最新にしておく(`apt-get update`)など、処理を指定することで作業を簡略化できます。

SoftLayerでは、このプロビジョニング時に指定可能なスクリプトを「Provision Script」と呼びます。またこの機能と合わせて利用すると便利な「User Metadata」というしくみがあります。これはサーバに対して任意の情報を関連づけることができます。「Provision Script」から「User Metadata」の値を利用できますので、スクリプトを汎用的に作り、メタデータで固有の情報を引き渡すような実装が考えられます。

またLinuxサーバをOSに選択した場合、初期状態でSSH接続は「rootユーザ」でパスワード認証となっていますが、SSH公開鍵の登録を自動で実施することも可能です。

それ以外にもネットワーク系の便利ツールがあるので併せて紹介します。

SSH Keysの指定

SoftLayerでは、SSH公開鍵認証を簡単に使うための機能が用意されています。プロビジョ

ニング時または「OS Reload」時に指定することにより rootユーザの公開鍵が指定可能です。この機能を利用すると自動的にrootユーザの**.ssh/authorized_keys**に追記されている状態になります。SSH公開鍵認証を利用するためには、事前に公開鍵を登録しておく必要があります。一度登録すると次回以降も利用できるため便利です。今回はUbuntuを使って解説していきます。手順は次のとおりです。

①利用したい鍵を事前にローカルのPCなどで作成する。**図1**ではMacOS上で実行（Linuxは同じ手順。Windowsの場合にはTeraTermなどを利用）
②管理ポータル上のメニューの[Device]→[Manage]→[SSH Keys]を選択
③[ADD]ボタンをクリックして登録画面を表示
④[Key Contents]にローカルのPCで作成した「test_rsa.pub」の内容を貼り付ける
⑤[Label]に名前を付けておく

その後、プロビジョニング時に「SSH-KEY」を選択するフィールドがあるので利用したい鍵の[Label]を選択します。サーバの起動後は、管理ポータル上からはSSH公開鍵の変更ができないので注意が必要です（通常のLinuxのお作法で登録することはもちろん可能です）。この段階では、パスワード認証も公開鍵認証もどちらも利用可能な状態になっています。

Provision ScriptsとUserMetaDataの使い方

プロビジョニングするときに、シェルスクリプトを実行できます。SoftLayerではこのスクリプトを「Provision Scripts」と呼びます。ここで指定するスクリプトはSoftLayerがHTTPSでアクセス可能な場所に置く必要があります。HTTPからでは実行されずHTTPSのURLだけが実行されるので注意が必要です。HTTPSサーバの準備が難しい場合には、GitHub社の提供するGistなどのサービスを利用するのが便利です。SSH公開鍵認証の設定と同様に事前に登録することも可能ですし、その都度指定することもできます。

ここではサンプルとして**リスト1**のようなスクリプトを用意して実行します。このスクリプトをGistに登録しておくことで起動時に実行

▼図1　ssh-keygenの実行例

```
$ cd .ssh
~/.ssh$ ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/Users/hideaki/.ssh/id_rsa): test_rsa 任意の文字
Enter passphrase (empty for no passphrase): パスフレーズを入力
Enter same passphrase again: 再度パスフレーズを入力
Your identification has been saved in test_rsa.
Your public key has been saved in test_rsa.pub.
The key fingerprint is:
92:c7:61:32:6e:99:15:3c:75:5c:be:ff:0b:62:0a:de hideaki@princess.local
The key's randomart image is:
+--[ RSA 2048]----+
|       ........  |
|        o. ...   |
|      o +.    .  |
|     . X .     . |
|      B S      .  |
|     . o       . |
|       .   o .  .|
|      . o o . . .|
|       . E     .o|
+-----------------+
hideaki@princess:~/.ssh$ ls -l test_rsa*
-rw-------  1 hideaki  staff  1766  8 18 19:19 test_rsa
-rw-r--r--  1 hideaki  staff   404  8 18 19:19 test_rsa.pub
```

▼リスト1 スクリプトサンプル例(https://gist.github.com/tokida/5b58831c0d94ce7b25f2)

```
#=================================================================
# bootstrap4Ubuntu.sh on SoftLayer
#=================================================================
sudo apt-get update
sudo apt-get upgrade

# Install Util
sudo apt-get install git -y
sudo apt-get install curl -y

# Install SoftLayer  Command line Interface
sudo apt-get install python-pip -y
pip install softlayer

# Japanese Locale
sudo apt-get install -y language-pack-ja-base language-pack-ja
update-locale LANG=ja_JP.UTF-8 LANGUAGE="ja_JP:ja"
source /etc/default/locale

# Japan TimeZone
cp -p /usr/share/zoneinfo/Asia/Tokyo /etc/localtime
echo "Asia/Tokyo" > /etc/timezone

# Please make sure to set SSH-KEY

sed -ri 's/#PermitRootLogin yes/PermitRootLogin yes/g' /etc/ssh/sshd_config
sed -ri 's/#PasswordAuthentication yes/PasswordAuthentication no/g' /etc/ssh/sshd_config
sed -ri 's/#PermitEmptyPasswords no/PermitEmptyPasswords yes/g' /etc/ssh/sshd_config

# get User_data
curl -k https://api.service.softlayer.com/rest/v3/SoftLayer_Resource_Metadata/ㄱ
getUserMetadata > ~/userMetadata.txt
```

▼図2 sl metadataコマンドの実行例

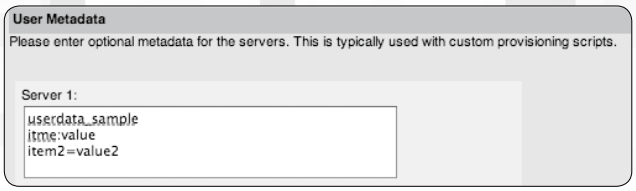
```
  backend_ip       Primary backend ip address
  backend_mac      Backend mac addresses
  datacenter       Datacenter name
  datacenter_id    Datacenter id
  fqdn             Fully qualified domain name
  frontend_mac     Frontend mac addresses
  hostname         Hostname
  id               Id
  ip               Primary ip address
  network          Details about either the public or private network
  provision_state  Provision state
  tags             Tags
  user_data        User-defined data

…… (中略) ……

root@provisionTest01:~# sl metadata network `sl metadata ip`
:..............:.................:
:          Name : Value           :
:..............:.................:
: mac addresses : 06:5a:94:08:1b:93 :
:        router : fcr01.sjc01     :
:         vlans : 848             :
:      vlan ids : 477454          :
:..............:.................:
```

されます。

リスト1を解説します。**apt-get-update**を実施して導入パッケージを最新化しています。次によく使う**git**、**curl**を導入しています。

そのあとSoftLayerのCLI(Command Line Interface)を利用するため、**python-softlayer**を導入しています。次にSSH公開鍵認証を利用しているのでrootのパスワードログイン認証を無効にしておきましょう。最後に、初期に登録できる**UserData**の値を取得

▼図3　User Metadataの指定例



▼図4　プロビジョニング実行例

```
$ curl -k https://api.service.softlayer.com/rest/v3/
SoftLayer_Resource_Metadata/getUserMetadata
"userdata_sample\nitme:value\nitem2=value2\n"
```

で「User Metadata」の値をREST API経由で取得できます。

とくにフォーマットされることなく、テキストデータとして取得できるので、この値を何らかのプログラミングで処理して利用することになります[注2]。

して**~/userMetadata.txt**に保管しています。

こうした一連の作業が可能ですので、いろいろ組み合わせて構成を実施できることがわかります。実行されるスクリプトは**/root/post_install.KRCp**のような形式で保管されているため起動後に確認できます。

SoftLayerの管理コマンドは**リスト1**の**pip**で導入されたことになります。そのほかの導入についてはSoftLayerのGitHub[注1]を参照してください。CLIを利用するには、あらかじめ認証されたユーザのIDとAPIキーが必要ですが、サーバからCLIを利用した場合にはキーの設定をしなくてもいくつかの値を取得できます（**図2**）。コマンドは**sl metadata**です。

ここまででプロビジョニング時にスクリプトを実行して、何らかのソフトウェアを導入することができるようになりました。実際に利用してみるともう少し柔軟に変更したい内容が出てきたりします。その場合はプロビジョニング時に「変数」として値を渡す手段があります。それが「User Metadata」と呼ばれるものです。すこし取り扱いが難しいかもしれないのですが、起動時にさまざまな値を指定できます。ここで記載した値はそのままの文字列でAPIやCLIから取得できます。

サーバを**図3**のように設定して構築した場合、**図4**のようにプロビジョニングされたサーバ上

SoftLayer上の便利なツール

サーバやネットワークに対して便利に使えるツールなどが用意されています。SoftLayerのサービスの状況を確認したい際にも便利な情報となります。

ネットワークステータス確認（Network Status）

サーバやネットワークの状態を視覚的に確認できます。「Local」と「Global」の2種類が用意されており、それぞれモニターできることが違います。いずれも、管理ポータル上の「Network」→「Status」から選択をします。

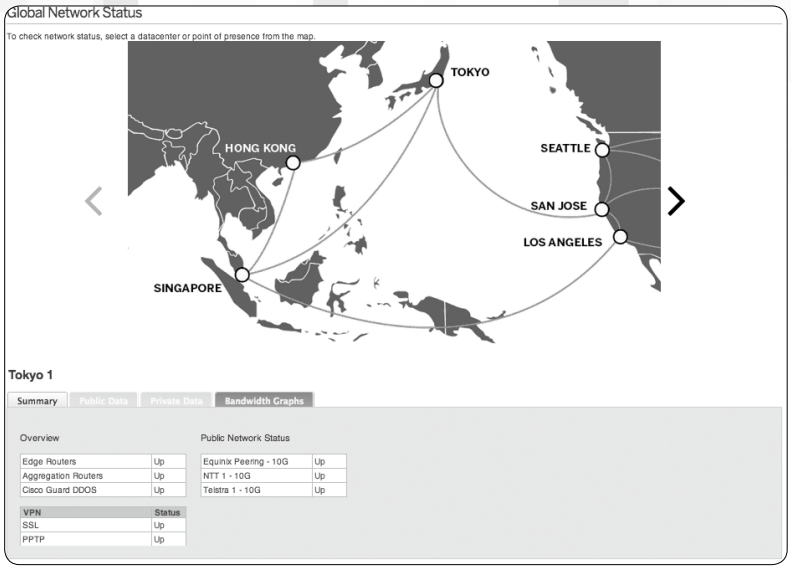
Global Network Status

図5のように、各DataCenterでのネットワークの状況がわかります。Bandwidth Graphsからは接続されているキャリアのbpsが視覚的に確認できます。それぞれのDataCenterのつながりもこれを見るとわかりますね。TokyoにはPoP（Points of Presence：接続ポイント）があり、Equinix、NTT、Telstraが10Gで接続しているのがわかります。また、SSL/PPTPなどのステータスもわかります。SSL-VPN接続などが不安定な場合などは確認してみると問題の切り分けになります。

注1） http://softlayer.github.io/softlayer-python/getting-started/

注2） SoftLayerにはサーバのOSを初期化する「OS Reload」という便利な機能があります。この「OS Reload」の際に「Script」は再設定ができるのですが「Provision User Metadata」は、再指定ができませんので気をつけてください。

▼図5 Global Network Status画面



Local Network Status

「Local」のネットワーク状況は、データセンター内部のネットワークの状況を確認できます。**図6**の例では、ルータ「fcr01a」→スイッチ「fas02a.sr01」→スイッチ「fcs24a.sr01」→サーバ「docker01.niandc.co.jp」となっており、そのときの状況でアイコンが変わります。全体をサッと見たいときにはこちらが便利なのでブックマークに登録しておくと良いかもしれません。

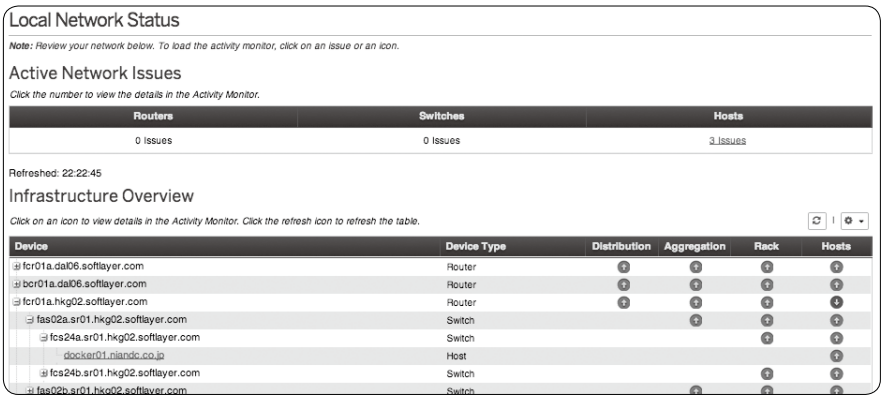
Network Tools

一般的なNetwork系のツールを、管理ポータルから実行できます。ブラウザで確認できるので便利です。管理ポータルのメニューから、[Network Tools]を選択します。SoftLayer上のサーバについてはプルダウン・メニューから選択できます。利用できるコマンドは、Ping、Traceroute、NSLookup、Whois、CheckDNSです(**図7参照**)。

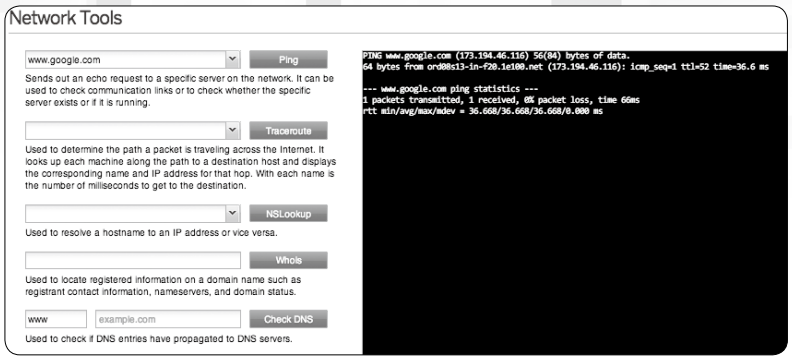
通信利用量の確認

通信量は、Privateは無料ですがPublic側は利用量に対して課金がされます。管理ポータルの「Network」→「Bandwidth」→「Summary」から参照できます。**図8**では各サーバの通信量が確認できます。この図では「Current Billing Cycle」の期間を参照しておりdocker02.niandc.co.jpは640MBほどIn通信をしていることがわかります。

▼図6 Local Network Status画面



▼図7　Network Tools



コマンドラインとAPIの利用

APIの豊富さが特長のSoftLayerですが、実際に利用しようとして筆者はつまづきました。まず、情報があまり見つかりません。今回は参考になればと、APIを使うサンプルを作ってみましたので紹介します。APIは「C#、Perl、PHP、Python、Ruby、VB.net」の言語にライブラリとして実装されています。プロトコルとしては「SOAP（XML-RPC）」そしてリクエストは「REST」により行います。

SoftLayerで提供されているサービスの多くは、APIで制御できます。これによりユーザはプログラミングでインフラストラクチャを操作することが（流行の「Infrastructure as a Code」です！）可能となり、既存の管理ポータルで不足している機能などを作ることもできます。実際のシステムの運用に入った段階で自動化を進めたいときなどは、このAPI利用により、より柔軟な対応ができるようになります。

実際はAPIだけでは学習コストがかかることもあり、導入が厳しい場合にはPython製CLIも用意されています（Windows、LinuxおよびMac OSでも動作）。

CLIについては各サーバで利用することにより便利な機能もあるため学習しておくと損はないかと思います。

ここでは簡単に利用すために用意されているコマンドラインインターフェース（以下CLI）の利用のしかたを紹介したいと思います。次回にAPIのサンプルとしてPHPを利用したサーバの状況を確認する例を紹介します。

CLIの導入方法

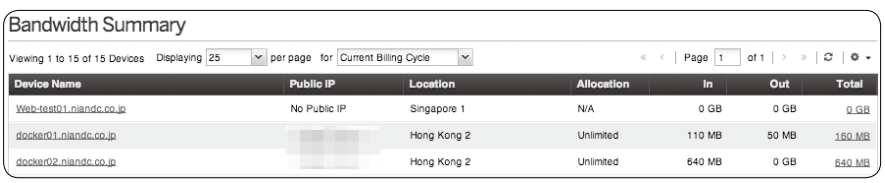
まず事前にPythonを導入します。ここではUbuntu 14.04環境を利用します。Windowsでは先にPython環境を構築した後に導入を行います[注3]。

```
# Install SoftLayer  Command line
Interface
sudo apt-get install python-pip -y
pip install softlayer
```

前節の「Provision Script」で使用していたコマンドです。Ubuntu 14.04以降では**apt-get**

注3）「Getting Started」http://softlayer.github.io/softlayer-python/getting-started/

▼図8　通信利用料確認



install python-softlayerでも導入ができます。CLIを利用するにはいくつかの情報をCLI側に設定する必要があります。

・API Endpoint
・API認証キー

の2つが必要です。

まず、API Endpointとは、API呼び出し先のサーバとなりパブリックネットワーク側、プライベートネットワーク側で用意されています。

API認証キーは、ユーザIDに紐づくAPI認証キーとなります。これは、ポータルのUser Profileで設定ができます。

①管理ポータルへログインする
②メニューの「Account」からプルダウンで表示された「Users」を選択する
③表示されている表のなかで、欲しいユーザ名に対応している「API Key」のセル上の「Generate」をクリックする
④表示が「View」へと変わると生成完了されるので、クリックするとAPIキーが表示

CLIを利用するにあたり、そのつどAPIキーを入力するのはたいへんですので、"**sl config setup**"で設定をします。この設定では、ユーザIDとそれに紐づく、先ほど入手した「API認証キー」を入力します。

図9のような形式で登録ができます。実体はログにも記載されていますが**~/.softlayer**となるため適宜切り替えて使います。**sl**コマンドによりサーバの追加、起動停止、またDNSサービスなどのそのほかのサービスの操作などを行うことができます。プロビジョニングされたサーバの状態を見るのは**sl vs detail <id>**で実施できます。

これらのオプションは多岐にわたり、チケットなどもCLIコマンドでできます。ブラウザを起動してログインして、といった繰り返し作業プロセスを顧みると、チケットをサーバ上で確認できるのでかなり便利です。

◆ ◆ ◆

少し敷居が高そうなベアメタルサーバですが、

注4) SoftLayer以外の環境で利用する場合には、Publicを選択しましょう。SoftLayer環境で利用する際にはセキュリティや通信面からもPrivateを選択することをお勧めします。

▼図9 sl config showコマンドの実行

```
root@provisiontest1:~# sl config show
:.............:.....................................:
:         Name : Value                               :
:.............:.....................................:
:     Username : not set                             :
:      API Key : not set                             :
: Endpoint URL : https://api.softlayer.com/xmlrpc/v3.1 :
:      Timeout : not set                             :
:.............:.....................................:
root@provisiontest1:~# sl config setup
Username []: SLのアカウントIDを入力
API Key or Password []: API Keyを入力
Endpoint (public|private|custom): private 注4
:.............:.....................................................................:
:         Name : Value                                                              :
:.............:.....................................................................:
:     Username : SL29******                                                         :
:      API Key : ab2f9****************************************92a92b6 :
: Endpoint URL : https://api.softlayer.com/xmlrpc/v3.1/                              :
:      Timeout : not set                                                             :
:.............:.....................................................................:
Are you sure you want to write settings to "/root/.softlayer"? [Y/n]: y
Configuration Updated Successfully
```

月課金だけでなく時間課金のモデルもあります。

従来からの「ベアメタルインスタンス」に加え今月から「ベアメタル」でも4種類のモデルが時間課金に対応しました。ぜひパフォーマンスを試してみてください。

まとめ

SoftLayerのサービスは常に更新されているため、筆者も知らない機能や先日までスクリプトなどで頑張っていた機能が、いつの間にか標準で使えるなど日々新しい発見があったりします。

今回はサーバの構築時によく利用する機能の紹介をしました。次号でWebサーバ構築の流れを紹介します。SD

COLUMN　日本SoftLayerユーザー会（JSLUG）の紹介

日本SoftLayerユーザー会は（以降 JSLUG）は、日本語によるSoftLayerに関する情報発信、共有を行い、SoftLayerの普及および人材育成に貢献するために、2014年5月23日に設立されました。2014年8月20日現在、ユーザー会の運営委員は12名、メンバーは374名です。

主な活動を紹介します。

勉強会の開催

東京では2ヵ月に1回、地方都市では3ヵ月～半年に1回の程度で、勉強会を開催しています。5月23日に第1回東京SoftLayer勉強会を開催してから、すでに8回もの勉強会を全国各地で開催しています。勉強会はイベント管理サービス「connpass[注5]」で告知されますので、connpass内のコミュニティのメンバーになれば自動で通知されます（ぜひ登録を！）。

Webでの情報発信

ユーザー会公式サイト[注6]でSoftLayerに関連する情報を発信しています。勉強会の資料や動画などを公開していますので、定期的に確認することをお勧めします。

▼図　日本SoftLayerユーザー会のアイコン

メーリングリストによるQ＆A

メーリングリスト（users@jslug.jp）でボランティアベースのサポートを行っています。SoftLayerのサポートサービスは定評があり評価されていますが、2014年7月現在は英語のみです。メーリングリストへの参加方法は、次の2つの方法があります。

・ユーザー会Webサイト[注7]から登録
・users-join@jslug.jpに空メールを送り、admin-bounces@jslug.jpからのメールに返信

雑誌やメディアへの寄稿

ユーザー会の運営委員は、1ヵ月に1回のペースでどのよう情報を発信するかなどの会合をしています。最近は運営委員が主体となって雑誌やメディアへの寄稿をしています。

ソーシャルメディアでの活動

@softlayerjpは、JSLUGの公式Twitterアカウントです。ぜひFollowしてください。JSLUG運営委員のアカウント[注8]もフォローしてみるといいかもしれません。

注5） http://softlayer.connpass.com
注6） http://jslug.jp
注7） http://jslug.jp/mailman/listinfo/users
注8） @kkitase、@kimotuki、@urasoko、@tokida、@MahoTakara、@zembutsu、@yasudatadahiro、@letrrb、@yuya_lush

実力検証

NICをまとめて高速通信！

後編 リンク・アグリゲーションの実力は？

ネットワークを行き来するデータは増え続け、インフラ担当者には常に高速化が要求されます。より高速な通信機器に交換できればよいですが、立ちはだかるのはコストの壁です。そこで以前からある手法、「チーミング」を再考してみましょう。本稿では前後編に分け、チーミングの手法と特性、そしてどの程度の高速化が見込めるのかを実験によって検証していきます。

後藤 大地（ごとう だいち）　(有)オングス　代表取締役

記事中、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。

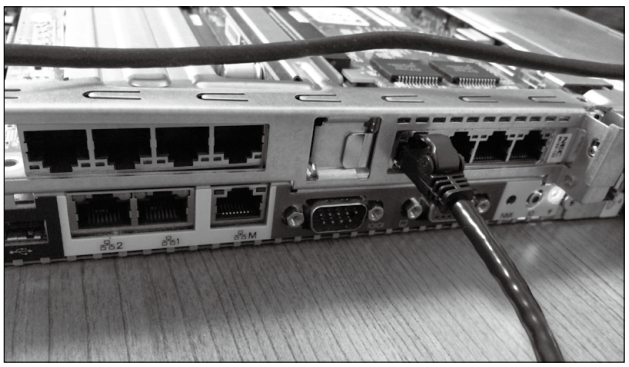
8ポートをまとめて通信実験！

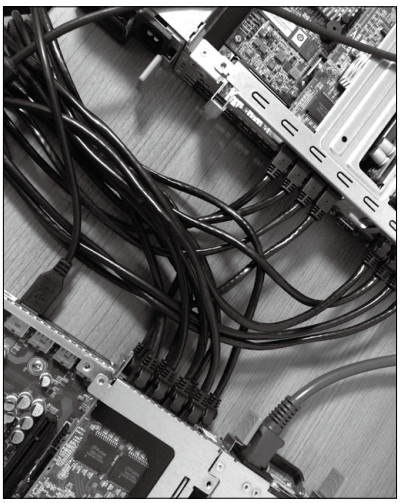
9月号掲載の前編では、複数のNICのポートをまとめあげて1つの論理チャンネルとして利用するチーミング、なかでも高速化を目的としたリンク・アグリゲーションの手法と、適した利用例について解説しました。後編の今回は、リンク・アグリゲーションがどれほど高速化に貢献するのかを実験で確かめてみます。

今回、実験用にNEC Express5800シリーズのラックマウントサーバ2台、NEC Express5800/R120d-1MとNEC Express5800/R120d-1Eを調達しました[注1]。NEC Express5800/R120d-1Mにはベースに2ポートのイーサネットポートがあり、追加で4ポートの拡張ボードを2枚挿して合計10ポートを装備（**写真1**）。NEC Express5800/R120d-1Eはベースに4ポート、追加で4ポートの拡張ボードを1枚挿しての合計8ポートです。この8ポートをカテゴリ6のLANケーブルで直接接続して通信実験を行います（**写真2**）。使用したオペレーティングシステム（以下、OS）はFreeBSD 10.0-RELEASEです（**写真3、表1**）。

注1）http://www.nec.co.jp/products/express/

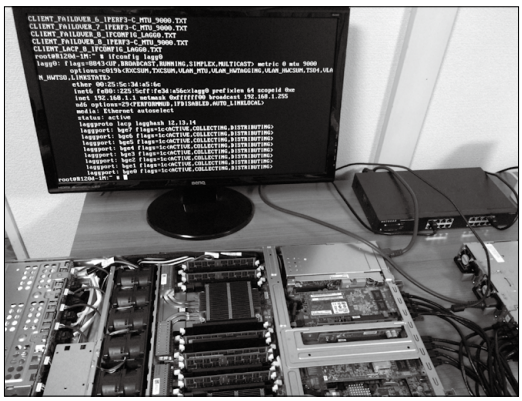
▼写真1　1000BASE-Tのポートが10個搭載されたNEC Express5800/R120d-1M





▶写真2　2台のNEC Express5800ラックマウントサーバを1対1でダイレクトに結線

▼写真3　計測環境



▼表1　テスト環境

ホストA	NEC Express5800/R120d-1M (Ethernet 10ポートのうち8ポート使用)
ホストB	NEC Express5800/R120d-1E (Ethernet 8ポート)
OS	FreeBSD 10.0-RELEASE

リンク・アグリゲーション実験のための設定

ホスト2台を1対1で接続して通信させるために、片方はゲートウェイとして機能する必要があります。そのうえで8ポートをまとめて扱えるようにするために、**リスト1**のような設定を片方のホストの/etc/rc.confファイルに追加します(もう一方のホストには**リスト1**の1行目を削除したものを追加します)。

リンク・アグリゲーションを行うための**lagg**(4)インターフェースを作成する前に、「mtu 9000 up」のように個々のポートのMTU[注2]の値を設定しているところがポイントです。こうしておくことで、作成されたlagg(4)インターフェースにおけるMTU値がここで指定されたものになります。

システムを再起動するなどしてこの設定を反映させます。このあたりの設定は**ifconfig**(8)コマンドで行いますので、再起動しないで個別にifconfig(8)コマンドを実行してもよいでしょう。ifconfig(8)コマンドの使い方などは以降で順次説明します。

再起動せずに手動でゲートウェイとして動作させるには、sysctl値のnet.inet.ip.forwardingを1に設定するといった操作が必要です。興味がある場合には/etc/rc.d/routingを読んでみてください。ここでの説明は割愛します。

▼リスト1　lagg(4)インターフェースを有効にする設定/etc/rc.confのサンプル(IEEE 802.1AX LACPに対応)

```
gateway_enable="YES"
cloned_interfaces="lagg0"
ifconfig_bge0="mtu 9000 up"
ifconfig_bge1="mtu 9000 up"
ifconfig_bge2="mtu 9000 up"
ifconfig_bge3="mtu 9000 up"
ifconfig_bge4="mtu 9000 up"
ifconfig_bge5="mtu 9000 up"
ifconfig_bge6="mtu 9000 up"
ifconfig_bge7="mtu 9000 up"
ifconfig_lagg0="laggproto lacp laggport↵
bge0 laggport bge1 laggport bge2 laggport↵
bge3 laggport bge4 laggport bge5 laggport↵
bge6 laggport bge7 192.168.1.1/24 up"
```

MTU値の違いによる通信速度の違いは？

前項の**リスト1**でMTUの値を9000にしてありますが、その理由を説明しましょう。

LANのように比較的通信が安定しているネットワークでは、MTUの値を引き上げることで通信性能の向上を実現できることが知られています。通常、イーサネットのMTU値には1,500バイトが使われます。現在のネットワーク機器はこの値よりも大きな値が設定できるようになっています。今回は通信速度を調べたいので、MTUの変動による通信速度の変化も調べます。この調査は1本での1対1接続で行います。

MTUの値を変更するには**図1**のようにifconfig(8)コマンドを実行します。2つのホストの双方でこの設定を実施します。

ここで一点、注意が必要です。ポートのMTUの値を変更したとしても、ホストからホストまで、すべての経路(たとえば送信元ホス

注2) Maximum Transmission Unitの略。ネットワークにおいて一度に送ることのできるデータの最大値を示す伝送単位のこと。各通信機器には各々の特性に応じたMTUが設定されている。

トのポート、経路上のスイッチ、経路上のルータ、送信先ホストのポート)が設定したMTUと同じ値に設定されている、またはその値に対応していなければ、指定したMTU値での通信はできません。

対象となる経路が設定したMTU値になっているかは**図2**のように`route`(8)コマンドで確認します。報告されるMTUの値が指定したものになっていれば、設定が有効になっています。

通信速度の計測にはiperf3(benchmarks/iperf3)を使います(**図3**)。片方のホストで「iperf3 -s」、もう片方のホストで「iperf3 -c IPアドレス」のように実行することで、このホスト間の通信速度を計測できます(**図4、5**)。

▼図1　ifconfig(8)でMTUを変更する

```
ifconfig bge0 inet 192.168.1.1 netmask 255.255.255.0 mtu 3000 up
```

▼図2　経路のMTUが本当に変更されたのかroute(8)コマンドで確認

```
% route get 192.168.1.2
   route to: 192.168.1.2
destination: 192.168.1.0
       mask: 255.255.255.0
        fib: 0
  interface: bge0
      flags: <UP,DONE,PINNED>                ↓ ここの値に注目
 recvpipe  sendpipe  ssthresh  rtt,msec    mtu        weight    expire
       0         0         0         0     3000         1         0
%
```

▼図3　通信速度計測用にiperf3をインストール

```
pkg install iperf3
```

▼図4　どちらか一方でiperf3サーバを起動

```
iperf3 -s
```

▼図5　もう一方でiperf3クライアントを起動してサーバとの通信を実施

```
iperf3 -c 192.168.1.2
```

▼図6　iperf3で通信速度を計測したサンプル

```
% iperf3 -c 192.168.1.2
Connecting to host 192.168.1.2, port 5201
[  4] local 192.168.1.1 port 25569 connected to 192.168.1.2 port 5201
[ ID] Interval           Transfer     Bandwidth
[  4]   0.00-1.01   sec  25.6 MBytes   213 Mbits/sec
[  4]   1.01-2.01   sec   110 MBytes   919 Mbits/sec
[  4]   2.01-3.00   sec   112 MBytes   943 Mbits/sec
[  4]   3.00-4.01   sec   113 MBytes   944 Mbits/sec
[  4]   4.01-5.01   sec   112 MBytes   941 Mbits/sec
[  4]   5.01-6.01   sec   113 MBytes   944 Mbits/sec
[  4]   6.01-7.01   sec   112 MBytes   941 Mbits/sec
[  4]   7.01-8.01   sec  90.6 MBytes   760 Mbits/sec
[  4]   8.01-9.34   sec  94.6 MBytes   596 Mbits/sec
[  4]   9.34-10.21  sec  3.75 MBytes  36.3 Mbits/sec
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bandwidth
[  4]   0.00-10.21  sec   886 MBytes   728 Mbits/sec                  sender   ← ここの値です
[  4]   0.00-10.21  sec   886 MBytes   728 Mbits/sec                  receiver ← ここの値です

iperf Done.
%
```

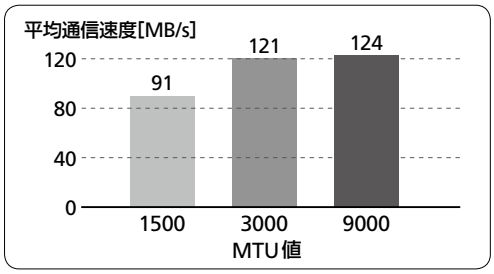
実行すると**図6**のような結果が得られました。コマンドが出力する最後のほうに通信速度の平均値が表示されています。この例ですと送信が728Mbit/秒(91MB/s)、受信が728Mbit/秒(91MB/s)ということになります。

MTUの値を1,500バイト、3,000バイト、9,000バイトの3つの値に設定して通信速度を計測した結果、**図7**のような結果が得られました。MTU値を大きくしたときのほうが通信速度が高速になっていることがわかります。以降の実験では、もっとも通信速度が速くなったMTU値9,000バイトを設定して通信実験を行います。

LACPで通信速度向上！

それでは話を戻して、IEEE 802.1AX LACP(Link Aggregation Control Protocol)を指定したlagg(4)インターフェースを作成し、それぞれ2本から8本まで集約した場合の通信速度の違いを計測します。ここからの計測は、冒頭の**リスト1**で設定した状態で行います。

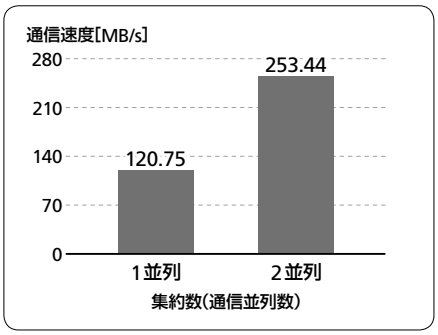
▼図7　MTUの違いによる通信速度の違い



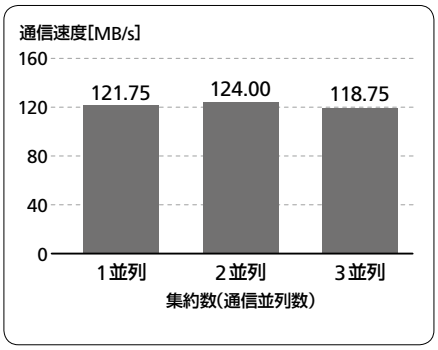
IEEE 802.1AX LACPでは通信の単位はそれぞれのケーブル単位ですので、集約数を増やした場合にはそれぞれ接続の並列数も増やして、全体としての通信速度の向上を計測します。単一の通信があるといった用途ではなく、常に複数のコネクションが発生するような通信における、全体としての通信速度の計測ということになります。

計測結果は**図8～14**のとおりです。集約数が1本から2本に増えた段階では、通信速度がほ

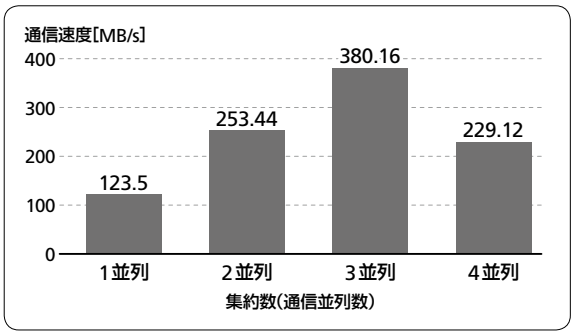
▼図8　2本(1～2並列)



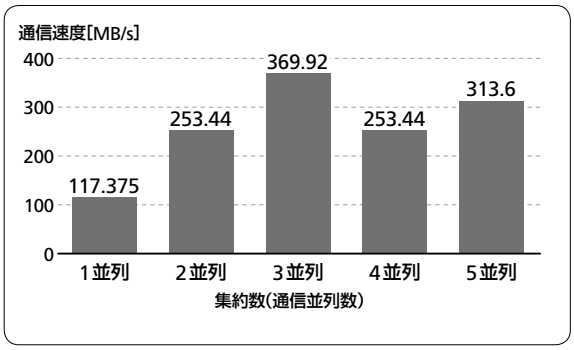
▼図9　3本(1～3並列)



▼図10　4本(1～4並列)

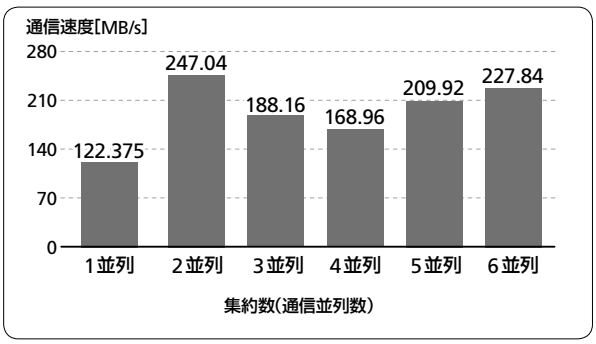


▼図11　5本(1～5並列)

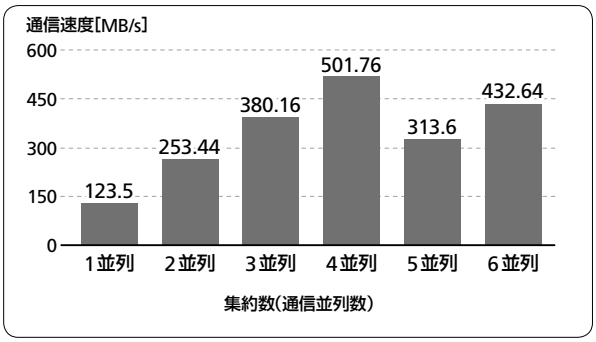


ぼ2倍になっています。しかしこれが3本、4本と集約数を増やした場合、本数によって通信速度の向上の仕方に違いが見られたほか、本数を増やせば増やすほど、実験ごとに出力される値にばらつきが現れるようになりました。通信速度が速いときは理想的な(集約した本数の分だけ)数値が計測されますが、そうでないときは合計で1本分の通信速度しかでない、といった状況が確認されました。本数を増やせばそれだけスケールするといった実装ではないことがわかります。

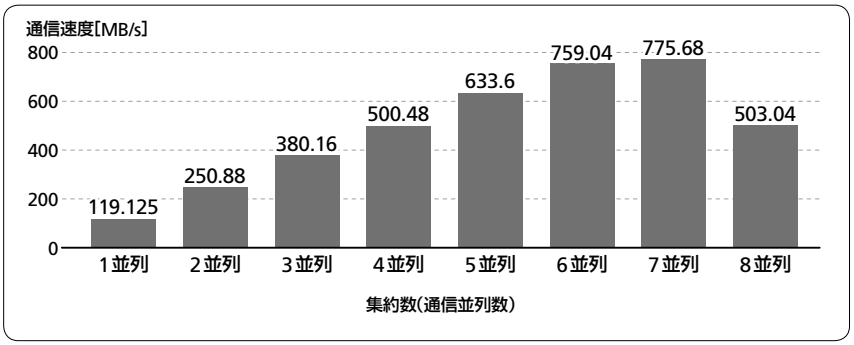
▼図12　6本(1〜6並列)



▼図13　7本(1〜6並列)



▼図14　8本(1〜8並列)



さらに別の実験を実施して、これがどういった理由によるものかを調べていきましょう。

並列数の増加と通信速度の関係

今度は集約数を4本に固定して、この場合で通信の並列数を1から12まで変化させてみます。

結果は**図15**のようになります。3並列までは理想的なスケールを見せますが、それ以上になると計測される通信速度にばらつきがでるようになります。11並列を超えたあたりで3並列よりも高い値が計測されるようになり、12並列ではさらに良い値がでるようになります。

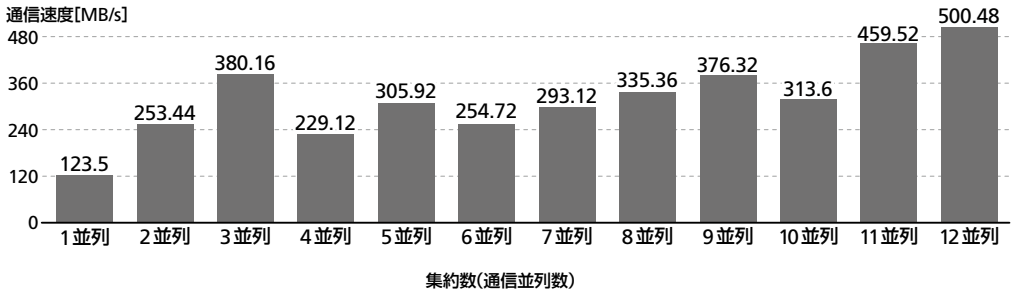
集約数を増やすことがそのまま通信速度の向上につながらないのは、IEEE 802.1AX LACPプロトコルが複数のポートをまとめて単一の論理チャンネルとして扱えるようにするプロトコルであって、どのストリームがどのポートを使うといった部分までは規定していないところに理由があります。

iperf3は指定された並列数までストリームを作成して同時通信を実施します。この並列通信は、次のソフトウェアの影響を受けます。

・カーネルのネットワークスタック
・カーネルのプロセス／スレッドスケジューラ

ネットワークスタックが複数のポート

▼図15　4本（1～12並列）



に対してスケールしない作りになっていれば、そもそも通信速度は上がりません。ネットワークスタックやスケジューラがマルチコアおよび複数のポートに対してスケールする作りになっており、かつ、作成したストリームがきれいにすべてのポートを使うようにあてがわれたときには、理想的な通信速度を見せることになります。たとえば、8つあるうちの4つのポートを使うような状況になった場合、通信速度はどうやっても4本分までしかでません。集約数が増えるほど通信速度が不安定になったというのにはこういった背景があります。

つまり、このあたりの通信性能はカーネル内部の実装にも影響を受けますので、どういった性能を見せるのか、どういった通信の傾向を見せるのかは、実際に使用するハードウェアとソフトウェアの組み合わせで調べるしかないということになります。

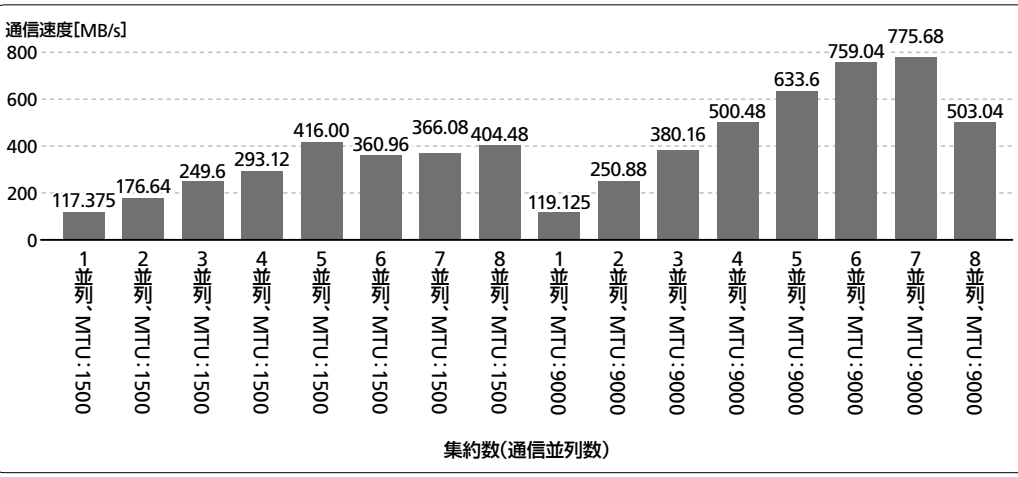
なお、特定の用途にしか使わないことがわかっているのであれば、lagg(4)インターフェースを含めカーネルを書き換え、常にすべてのポートが使われるように変更するといった方法をとることができます。

並列性能とMTUの関係は？

リンク・アグリゲーションを行う前に、1本においてMTUの値を変更させて通信速度の違いを計測しました。ここでは8本を集約して1から8並列までストリームを作成した場合に、MTUの違いでどのように通信速度が変化するかを計測します。比較対象はMTUの値が1,500バイトと9,000バイトの2つです。

結果は**図16**のようになりました。全体的に

▼図16　8本（1～8並列、MTU：1500）および8本（1～8並列、MTU：9000）



MTUの値は1,500バイトよりも9,000バイトを指定したときのほうが高い性能がでていることがわかります。

フェイルオーバーと通信速度

今回は通信速度を引き上げるという目的でlagg(4)インターフェースを使っていますが、通常業務でlagg(4)インターフェースを通信速度の引き上げに使うということはあまりありません。多くの場合はフェイルオーバーの目的で使用するでしょう。

フェイルオーバー目的の場合、lagg(4)インターフェースを作成する段階で「laggproto lacp」ではなく「laggproto failover」のようにプロトコルを指定します。lagg(4)インターフェースはプロトコルとしてfailover、fec、lacp、loadbalance、roundrobin、noneを指定することが可能で、それぞれ異なる動きをみせます。

フェイルオーバープロトコルが指定されたlagg(4)インターフェースをifconfig(8)で確認すると**図17**のように見えます。ここではbge0、bge1、bge2がlagg(4)インターフェースに追加されており、bge0がマスタになっていることがわかります。

ここでbge0につながっているLANケーブルを抜くといった操作を行うと、アクティブなポートがbge0からbge2へ移行することを**図18**の実行で確認できます。1つの回線が不通になっても、あと2つが予備として控えているため、そちらへ切り替えて通信が継続されます。

試しに2本から8本までのそれぞれで、フェイルオーバー目的のlagg(4)インターフェースを作成して通信速度を計測しました。得られた結果は**表2**です。当然ですが、フェイルオーバー目的のlagg(4)インターフェースでは同時に通信

▼表2　lagg(4)フェイルオーバーと通信速度

集約数	通信速度[MB/s]
2	123.5
3	123.5
4	123.5
5	123.5
6	123.5
7	123.5
8	123.5

▼図17　lagg(4)インターフェースでフェイルオーバーを設定している場合

```
% ifconfig lagg0
lagg0: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> metric 0 mtu 9000
        options=c019b<RXCSUM,TXCSUM,VLAN_MTU,VLAN_HWTAGGING,VLAN_HWCSUM,TSO4,VLAN_
HWTSO,LINKSTATE>
        ether 00:25:5c:3d:a5:6c
        inet6 fe80::225:5cff:fe3d:a56c%lagg0 prefixlen 64 scopeid 0xe
        inet 192.168.1.1 netmask 0xffffff00 broadcast 192.168.1.255
        nd6 options=29<PERFORMNUD,IFDISABLED,AUTO_LINKLOCAL>
        media: Ethernet autoselect
        status: active
        laggproto failover lagghash l2,l3,l4
        laggport: bge2 flags=4<>
        laggport: bge1 flags=0<>
        laggport: bge0 flags=1<MASTER,ACTIVE>  ← bge0がマスタで実際にアクティブにもなっている
%
```

▼図18　bge0のケーブルを引っこ抜くとbge2が替わりに使われるようになる

```
% ifconfig lagg0
…略…
        laggport: bge2 flags=4<ACTIVE>   ← bge2がアクティブになっている
        laggport: bge1 flags=0<>
        laggport: bge0 flags=1<MASTER>   ← bge0が通信できない状態
%
```

するのは1本だけですので、通信速度はケーブルの集約数が2本でも8本でもすべて同じです。

ロードバランスとラウンドロビン

前述のとおり、lagg(4)インターフェースではフェイルオーバーとLACP以外のプロトコルも指定できます。試しにラウンドロビン、ロードバランス、LACPで8本集約の1～8並列といったケースで性能の違いを計測してみました。

結果は**図19**です(ラウンドロビンは3集約以上は機能しなかったので、2本集約までの値を掲載してあります)。通信速度の向上という目的では、今回のケースではLACPがもっとも高い性能が期待できそうです。

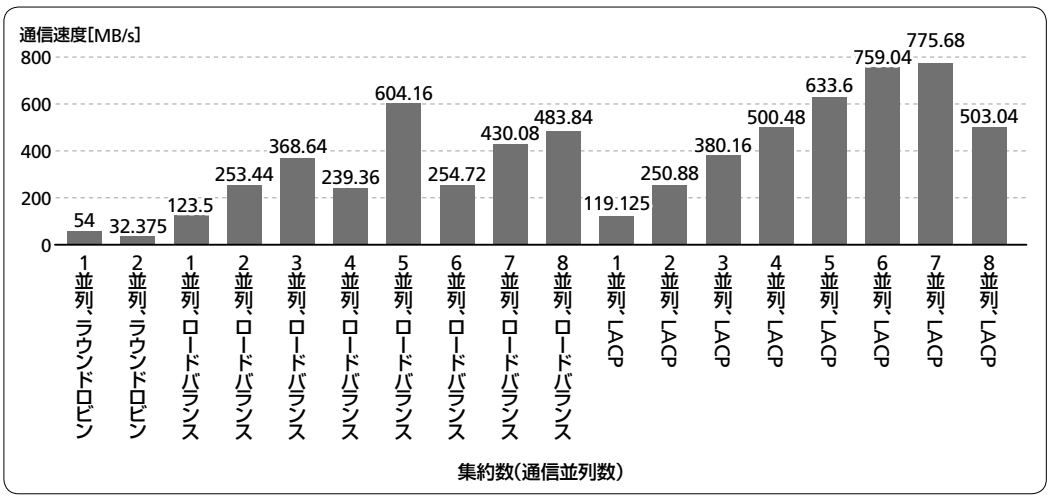
実際に試してわかるエンジニアリング

リンク・アグリゲーションの機能を使えば総合的な通信速度を向上させることはできそうだ、ということは想定できますが、実際にどのような挙動を見せるかは実験してみないとわからないところがあります。複数のポートを使うタイプの通信はさまざまな影響を受けますし、その影響はネットワークスタックのみならずハードウェアやスケジューラの影響も受けますので、ソースコードから動きを想像するだけでは推測できないところがあります。実際に実験して試してみるのが一番ですので、本記事がその参考になれば幸いです。

余談になりますが、複数のケーブルを束ねて通信速度を引き上げる目的では、現在策定が進められ、LinuxやFreeBSDで実装が進められている「Multipath TCP」などの技術を挙げることができます。TCPのエクステンションとして実装されているため、既存の環境との互換性がよいという特徴があります。今回紹介したlagg(4)インターフェースと同じで、この機能を利用するためにはソフトウェア側を書き換える必要がなく、カーネルが対応すれば透過的にソフトウェアはその効果を受けることができるという特徴もあります。

Multipath TCPはすでにAppleやCitrixなどのプロダクトでも実装され使用されてもいます。今後、インターネットにおける汎用的な通信技術として普及するかどうかはまだわかりませんが、LinuxやFreeBSDのネットワークスタックがデフォルトでMultipath TCPの機能を持つようになれば、高速通信やフェイルオーバー目的でこうした機能が使われていく可能性もあるでしょう。SD

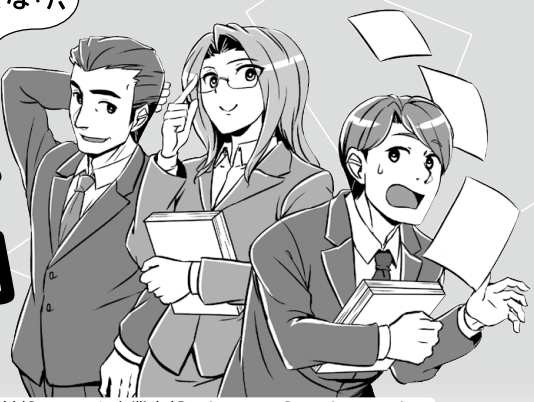
▼図19　8本(1～8並列)をラウンドロビン、ロードバランス、LACPで比較



帰宅が5分早くなり、休出もなくなる!?

目指せ! 定時帰りのエンジニア!

Hinemosで学ぶ ジョブ管理超入門



茶納 佑季（ちゃのう ゆうき） ㈱NTTデータ 基盤システム事業本部 chanouy@nttdata.co.jp

この物語は、新人SE藤井君がとあるシステムの運用の効率化に取り組み、定時帰りのエンジニアを目指すものです。昨今流行っている「運用自動化」をキーワードに、全9回の連載で、シェルスクリプトやcronを用いた基本的なシステムのジョブ管理から、Hinemosという運用管理ツールを使ってのジョブ管理・運用効率化を学ぶことができます。 イラスト（高野 涼香）

第1回 処理を自動化? ジョブ管理ってなんだろう?

新人藤井君の受難

10月某日――藤井君は悩んでいました。研修もそこそこに、日々雑務や目新しいことに追われながら業務をこなしてきました。上司に呼ばれた彼は「システムの運用自動化」の仕事の依頼を受けたのでした。

上司「あー、藤井君、ちょっといいかな? 今、我が社で使っている『勤怠管理システム』の運用を任せたい」

藤井「システムの運用……ですか?」

上司「そうだ。運用だ。前任の運用管理者が急遽退職してしまうことになってね。運用手順書も残っていないし、いろいろ非効率なところもあるようだ。これを機に最近流行りの「運用自動化」を取り入れた見直しも考慮に入れて、効率的な手順化を頑張ってみてくれ。あぁ、あまりコストをかけずに頼むよ」

藤井「はい……『勤怠管理システム』の運用を自動化ですね?……頑張ります!」

システムの運用とは?

藤井「うーん……運用の自動化って言われても何をどうすればいいんだろう? その前にシステムの運用ってどんなことをするんだったかな? 結構やることが多かった覚えがあるけど、確か研修でやったなぁ……」

登場人物紹介

藤井 今年SI企業に入社した新人SE。上司からとある仕事を任されるらしいが……。

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。

次回登場!

システムの運用とは、運用対象のシステムが、利用者にサービスを滞りなく提供できるよう、維持管理することを言います。

そのためには、システム運行に必要な処理の実施、システムに異常が発生した際の検知、ハードウェアやソフトウェアの変更管理や資産管理、セキュリティの管理など、さまざまなことを行う必要があります。また、異常が発生し、システムが正常に利用できなくなった場合（インシデント）の、問い合わせの対応やそのインシデントの管理も含まれます。

このように、一言で運用と言っても、実施対象や範囲は多岐にわたります。

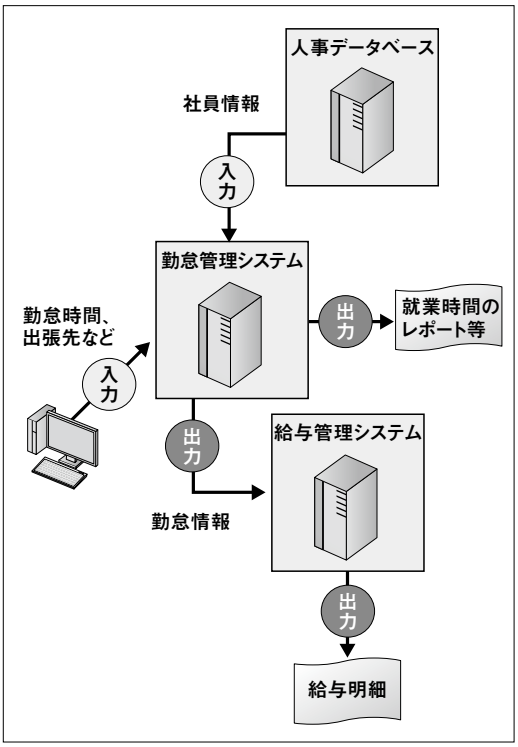
今回の連載では運用自動化に向け、システム運行に必要な処理の自動化にスポットライトを当てて、説明していきます。

対象システムの概要

藤井「やっぱり運用っていろんなことを考えて実施しないとならないな。でも、自動化って言われても何をどうすればいいんだろう？　今だって誰かが手動で常時動かしているようにも見えないしな……そもそもどうやって動いているんだ？　まずは『勤怠管理システム』がどんな運用をしているのかを見てみるか……って運用手順書残ってないんだった……あ、でも前任者の「作業ノート」が残っているぞ。ほとんどメモレベルだけど。どれどれ……」

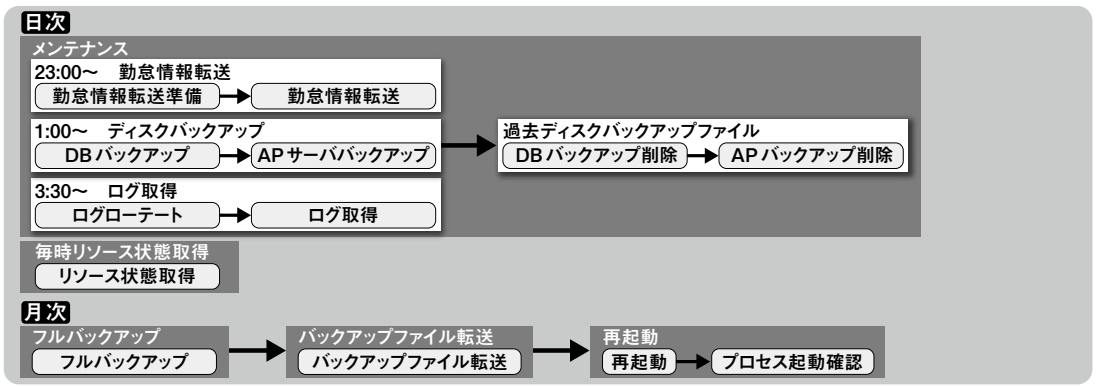
今回、藤井君が運用を任された『勤怠管理システム』は、日々、社員が出社時間や退社時間、出張先などの勤怠に関する情報を入力します。入力された勤怠情報はDB内に蓄積され、その情報をもとに、給与計算や就業時間のレポートなどを出力します（**図1**）。

▼図1　勤怠管理システムの概要



『勤怠管理システム』の業務部分（アプリケーション部分）に加えて、バックアップなどの裏方的な基盤処理はおおまかに**図2**のような流れで動いています。

▼図2　勤怠管理システムの基盤処理の流れ（簡略）



藤井「なんだ、自動的に動いているじゃないか。安心した。でもOSは全部Linuxか……ん？　手書きのコメントだ。何々……？　『cronだけで動かしていてツライ』『人事異動や監査対応といった特別対応があるので注意すること。とくに四半期や年末年始、年度末は要注意』『祝日や特別対応の日は手動で切り替えること』えぇーっ！？　手で動かすの！？ていうか祝日出社なの！？」

運用自動化ってなんだろう

藤井「ただでさえ忙しいのに、祝日出社なんてしたくない。これは『運用自動化』を本気で考える必要があるな。ただ……運用自動化とはいったい？」

さて、藤井君も困っていますが、「運用自動化」とは何なのでしょうか。運用自動化とは、システム維持管理業務を可能な限り自動化することによって、システムの運用を効率化することです。

システムを運用する際、サービスが正常に提供されているか、システムを構成しているハードウェアやソフトウェアが正常に動いているかといった挙動の検知を「監視」、システム運行に必要な処理を「ジョブ」と呼ぶことがあります。今回の連載では、オープン系システム[注1]の運用管理の中で、コンピュータに実行させる処理単位であり、バッチ処理やそれを実現するためのスクリプトをジョブと呼ぶこととします。また、システム内の複数のサーバで連携させてジョブを実行させたり、正常確認や異常対処を行ったりして、ジョブを管理することをジョブ管理と呼ぶこととします。

たとえば、データのバックアップや、バックアップ処理のログ転送などといった処理のひとつひとつがジョブとなり、バックアップやログ転送が意図した時刻に実行されているか管理することがジョブ管理となります。このジョブがそれぞれ連携し合い、正常に動き、綿密に管理されることで、システムが滞ることなく運用されていきます。

運用自動化の重要性

システムの運用を自動化するためには、これらの「監視」や「ジョブ管理」を自動化する必要があります。従来、人が手作業で行っていたことを、システム化を図ることで、コンピュータに代わりに行ってもらい、作業の自動化や効率化といったメリットを享受してきました。

しかし、システム化のなかに人のオペレーションは残りました。一例として、監視では、何らかの異常検知後、人が異常コードを膨大なマニュアルから検索し、運用管理担当者へ電話するオペレーションを行っていたり、ジョブ管理では、深夜のジョブ実行を人の手で行い、ジョブの終了を見届け、エラーが発生していないかチェックするといったオペレーションを行っていたりします。

システム化のなかに人のオペレーションが残る原因としては、システム化したことによる新たな作業の発生であったり、システムの段階的な拡充による、当初は想定されていなかった運用要件の発生などさまざまな要因が考えられます。

このように、システム化のなかの手作業は平然と残ってきました。さらに最近では、仮想化環境やクラウド環境を利用することにより、柔軟な構成変更などができる難しいシステムが早いサイクルで作ることができるようになりました。このような環境に対する運用は、もはや手作業では限界を迎えつつあります。

読者のみなさんの中でも、現在このような問題に直面されている方もいるかもしれませんし、今後、このような問題に直面する方もいるかもしれません。

このような問題が顕在化し波及する前に監視やジョブ管理の、さらなる自動化や効率化を図ることが重要なのです。

注1) オープン系システム……メインフレームなどの汎用機を利用したシステムではなく、それぞれに役割を持つ複数のサーバやネットワーク機器、アプライアンス機器などを組み合わせることにより、1つの業務を達成するシステム。UNIX系サーバやWindows・Linuxが稼働するIAサーバが用いられることが一般的。

運用管理のための ジョブ管理の方法は？

藤井「ふむふむ……運用自動化のためには「監視」や「ジョブ管理」を自動化する必要があるのか！
作業ノートを読んだ感じでは、ジョブ管理は自動にできてるところもあるけど、もっと自動化できるんじゃないかなぁ。ジョブは何をどうやって管理すればいいのか調査してみよう」

ジョブ管理の要素

ジョブを管理するためには、大きく分けて次の要素を管理する必要があります。

- **ジョブ実行のスケジュール管理**
- **ジョブの状態（実行状態、終了状態）の管理**
- **ジョブの実行履歴の管理**

それぞれの要素の例としては、スケジュール管理については、平日（月～金）の3:00にジョブを実行する、ファイルの作成や変更をきっかけとしてジョブを実行する、といった例が挙げられます。ジョブの状態管理については、単体のジョブやジョブのまとまりが問題なく実行され、終了するかを確認することや、仮に異常終了した際には、再実行やリカバリーといったアクションを行うことが求められます。ジョブ実行履歴の管理については、実行したジョブの結果を管理簿やログに記録し、後日確認できるようにすることが挙げられます。

ジョブ管理の方法

藤井「うわぁ……これを手作業でやるのたいへんだな……。何かいいツールとか便利なものってないのかさらに調査だ！」

ジョブ管理を行うためには、大きく分けて3つの方法があります（次ページ）。

▼表1　ジョブ管理方法のメリット・デメリット

	手作業	OS標準機能	ツール
代表例	—	cron（Linux） タスクスケジューラ（Windows）	・商用製品 JP1 Systemwalker WebSAM Senju A-AUTO ・OSS Hinemos JobScheduler
メリット	・任意のタイミングでジョブを実行することができる ・運用管理者が現場で実行するため、不測の事態にすぐ対処できる	・ジョブをスケジューリングすることで、定時実行できる ・ジョブとなるスクリプトを作りこむことで、たいていのジョブを管理することができる	・ジョブをスケジューリングし、業務カレンダに沿った対応など、例外的な内容を含めて自動化できる ・実行結果の確認を視覚的に行うことができる ・ひとつのサーバだけではなく、複数のサーバとの連携や、ジョブとジョブの連携が容易に行える
デメリット	・運用管理者が現場に常駐している必要がある ・スクリプトのログにより実行結果を確認する必要がある ・人員の交代やシステムのリプレイスといった際の引継ぎが煩雑となり大変	・cronの場合、実行結果の確認が困難 ・他システム・他サーバとの連携のためにはスクリプトを作り込む必要がある ・スクリプトを作り込み維持するスキルが必要となる ・祝日など例外的な日時に対応できない ・スクリプトの前後の関係性が視覚的にわからず、全体像を確認することが困難	・商用製品の場合、導入のために製品を購入するため、コストがかかる ・運用管理で使うツールに精通する必要がある ・ツールのバージョンアップに対応する必要がある

・手作業

作成したスクリプトを決まった時間になったら、手で実行する

・OS標準機能

Linuxであれば「cron」、Windowsであれば「タスクスケジューラ」などを使い、定期的にスクリプトを実行する

・ツール

商用製品であれば「JP1」、「Systemwalker」、「WebSAM」、オープンソースソフトウェア（OSS）であれば「Hinemos」などを使い、ジョブ実行だけではなく運用面を考慮しながら自動的に実行する

それぞれのメリット・デメリットをまとめると、前ページの**表1**のようになります。

もちろん、手作業でも運用は可能です。ただし、そこには属人化された手法や、体制（ローテーション）や人件費を考える必要があります。その点、ツールでは特定の人が持っている運用ノウハウを凝縮し、ソフトウェアとして提供しています。ツールを利用することで、人のかかわり度合いを減らし、運用担当者の負担を下げ、効率的な運用を実現できます。

藤井「調べたら少しずつわかってきたぞ……。今は手作業とOS標準機能の合わせ技で運用していることだし、ツールを使いたいな。とりあえず着手できるしOSSかな。まぁ、困ったらマニュアルとか公式ページ見ればいいや。英語は苦手だけど、有償サポートしてくれる日本の会社もあるみたいだし。ん、このHinemosって日本製じゃん！　しかもマニュアル類も日本語だし、公式ページ[注2]には技術情報とかも書いてある！　よしよし、Hinemosを使ってみようかな。ここまでの経過を上司に報告だ！」

今月の時短ポイント

システム化することで、コスト低減や利便性の向上を図ってきました。しかし、その中で人の手による運用は残り続けました。

今回のようにシステム運用を見直し、運用自動化に取り組むことで作業時間短縮（時短）への第一歩を踏む出すことができます。

次回予告

藤井「『勤怠管理システム』の運用自動化についてですが、Hinemosを使って自動化してみようと思います！」

上司「ふむ。よく調べたね。ところで、実際に動いているスクリプトのしくみや、今動かしているcronは理解している？」

藤井「えっ……？」

上司「ま、技術を身に着けるためにも早く手を動かすんだな」

藤井「は、はいー」（ぴゅーん）

スクリプト作成は研修で少しかじった程度の藤井君。スクリプトのしくみはおぼろげにしかわかっていないようです。はたして、業務で使えるスクリプトを理解することはできるのか！？

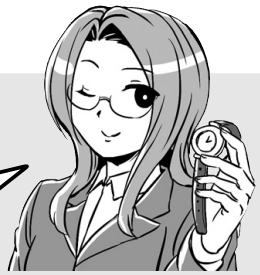
次回『「ジョブ管理」の第一歩 シェルスクリプトを動かそう』SD

To Be Continued...

注2）http://www.hinemos.info/

Column

運用管理ツールとOSS



日本企業（日系ベンダ）の強い運用管理

日本の運用管理業界では、JP1（日立製作所）、Systemwalker（富士通）、WebSAM（日本電気）といったような日本企業の運用管理ツールのシェアが高いという調査結果があります。この要因については諸説ありますが、「日本人のジョブへのこだわりが強い」ということが要因の1つと言えます。日次、月次、年次、稼働日、非稼働日、平日、休日、祝前日……など、いろいろな言葉がありますが、日本人の時間へのこだわりはとても強いものがあります。このこだわりの強さが、ジョブ管理にも表れており、緻密に時間を計算し、電車の運行表のようなジョブを組み上げることが多いのです。これらのようなニーズに合わせて、日本企業の運用管理ツールは進化してきました。つまり日本人に合ったソフトウェアなのです。

OSSとは？

OSS（Open Source Software／オープンソースソフトウェア）はソースコードが公開されたソフトウェアです。通常、ソフトウェアのソースコードは非公開であり、修正や改変のために手を加えることは認められていません。

一方OSSは、ソースコードを確認したうえで、修正や改変を行うことができます。ただし、OSSは著作権を放棄したソフトウェアではありません。著作者がライセンスのあり方を設定していて、そのライセンスに従った使い方のもと扱う必要があります。

いくつかの有名なライセンスを紹介します。

- GNU GPL（General Public License）
- BSD License
- MIT License
- Apache License

たとえば今回、藤井君が選ぼうとしている「Hinemos」はGNU GPL（General Public License）で公開しており、SourceForge.JPのHinemosプロジェクト（http://sourceforge.jp/projects/hinemos/）にて公開されています。

また、OSSはフリーソフトではないため、すべてを無償でダウンロードできるわけではありません。OSSのOSの1つであるRed Hat Enterprise Linuxはサブスクリプション形式を採用しており、サブスクリプション契約を結ぶことでサポートを受けることやerrataを入手できます。

統合運用管理のためのOSS「Hinemos」

OSSは商用製品と比べ、「気軽に導入できる」「導入コストを抑えられる」「ベンダロックインを回避できる」といったメリットから採用されることが多くなってきました。

しかし、OSSは単一の機能に特化していることが多く、機能単位で異なるツールの導入が必要となり、煩雑さが増すといったデメリットもあります。

たとえば、統合運用管理の場合では一般的に「監視機能」「ジョブ機能」と別々のツールの導入が必要となります。**表2**に有名な運用管理OSSを示します。

▼表2　運用管理OSS

ツール名	開発国	監視機能	ジョブ機能
Zabbix	ラトビア	○	×
Nagios	—	○	×
JobScheduler	ドイツ	×	○
Hinemos	日本	○	○

そのなかでもHinemosは監視機能とジョブ機能の両機能を併せ持つ唯一のOSSです。まさにHinemosは統合運用管理を実現するためのOSSなのです。

サポートエンジニアのクラウドワークスタイル

Heroku女子の開発日記

第2回 最初の一植え
Herokuにデプロイしてみましょう

第2回となる今月は、ローカルにあるアプリケーションをHeroku上にデプロイする方法を解説します。すべての基本となる操作・コマンドを紹介しているので、きちんとおさえておきましょう。「サンフランシスコだより」では、サンフランシスコの住宅事情をこっそりお教えします。

Heroku　織田 敬子（おだ けいこ）

デプロイしよう!

第1回は、Heroku toolbeltをインストールしてCLI(Command line interface)を使ってログインするステップまでを行いました。今回は、実際にアプリケーションを作成し、そのアプリケーションをHeroku上にデプロイしてみます。前回を見逃した方は、ぜひSign up[注1]をして、Toolbelt[注2]をインストールしてみてください。

前回でも触れましたが、Herokuではさまざまな言語をサポートしています。本連載ではRuby on Railsを使用したアプリケーションを例として話を進めていきますが、読者の方がすでに開発済みの他言語のアプリケーションをデプロイする場合にも、お使いいただける内容です。

アプリケーションを用意する

Ruby on Railsを使用したアプリケーションということで、今回はHerokuで提供しているサンプルアプリケーションを使っていきます。`git clone`を用いてコピーします。

```
$ git clone https://github.com/heroku/ruby-rails-sample
$ cd rsuby-rails-sample
```

次に、ローカルで動くかどうか確認してみましょう。Herokuに上げる前にローカルで動くことをしっかり確認することは大切です。

```
$ bundle
$ bundle exesc rake bootstrap
$ foreman start
```

アプリケーションが起動したら、ブラウザで http://localhost:5000/ を開き、正常に動いていることを確認しましょう。

Gitでバージョン管理

Herokuではデプロイに Gitを使用します。Gitは、ご存じの方も多いかと思いますが、アプリケーション開発のお供とも言えるバージョン管理システムです。Herokuではこの、開発者には馴染み深いGitを用いてアプリケーションをデプロイすることができます。

Herokuを使ううえでGitの達人である必要はありませんが、これから説明していくような基本的なところはぜひおさえておきたいところです。

git init

このコマンドで、まずローカルのリポジトリを初期化します。`git init`するだけではとくに何も起こりませんが、`git init`を行ったフォルダ直下に.gitという名前のフォルダができ、そこでこれから行われるあらゆる変更を管理していきます。

今回のサンプルアプリケーションは、すでにGitで管理しているリポジトリをcloneしてき

注1）URL https://www.heroku.com
注2）URL https://toolbelt.heroku.com

たので、初期化の必要はありません。

git add

先に述べたとおり、**git init**だけではまだ、空のローカルリポジトリができただけの状態です。**git add**コマンドにより、そのファイルをリポジトリに追加する準備ができます。ここで、次に説明するコミットを行うまでは、実際にリポジトリに追加されていないことに注意してください。**git add <file/directory>**によって任意のファイル・ディレクトリを追加できますし、**git add .**によってフォルダ直下のすべてのファイルを追加できます。

Gitのバージョン管理下に置きたくないファイルは、**.gitignore**ファイルを使用して管理できます。

今回は、すでにファイルがaddされているので**git add**の必要はありません。もしファイルのどこか(app/views/welcome/index.html.erbなど)を変更した場合は、それを**git add**しておきましょう。

git commit

このコマンドにより、変更がローカルリポジトリに「コミット」されます。このコミットされた単位で、Heroku上にもデプロイができます。コミットを行う前には、必ず**git status**を使用して自分がコミットしようとしているファイルの確認をするようにしましょう。**-m**によってコメントを追加することもできます。

今回はコミットするものがありませんが、もし前のステップでファイルに変更を加えていた場合はコミットしてください。

```
$ git commit -m 'my first commit'
```

Heroku上にアプリケーションを作成

Heroku上でアプリケーションを作成するには、**heroku create**コマンドを使用します。このコマンドによって、Heroku上にアプリケーションが作られると同時に、ローカルGitリポジトリに**heroku**という名前のリモートリポジトリが追加されます。ここで、「リポジトリには**ローカルリポジトリ**と**Herokuリポジトリ**がある」ということをしっかり理解してください。ほかにも、もしGithubなどでソースコードを管理している場合は、これに加えてGithubにもリポジトリがあることになります。リモートリポジトリの一覧は**git remote -v**で確認することができます。

heroku createコマンドではアプリケーション名を指定することもできます。指定しなかった場合は、Herokuで適当な名前を付けます。このアプリケーション名は全世界で早い者勝ちで、アプリケーションを作成すると**http://appname.herokuapp.com**というURLが付与されます。あとで**heroku rename**コマンドを利用して名前を変更することもできます。

```
$ heroku create
Creating evening-thicket-3022... done, stack is cedar
http://evening-thicket-3022.herokuapp.com/ | git@heroku.com:evening-thicket-3022.git
Git remote heroku added
```

Herokuにデプロイ

さて、ここまででHerokuへデプロイするローカル側、Heroku側の準備が整いました。ここでようやく、Herokuにアプリケーションをpushできます。コマンドはいたって簡単で、**git push heroku master**です。これでローカルリポジトリで今までコミットされた分のアプリケーションがHerokuへデプロイされます。

ここで、SSHキーの登録がうまく行っていない人はもしかしたら、

```
Permission denied (publickey).
```

といったようなエラーが出るかもしれません。そんな場合は、「Managing Your SSH Keys」[注3]という記事に従い、キーの削除・再追加を行ってみてください。

デプロイがうまく行くと、最後に次のようなメッセージが出るはずです。これが出ない場合は、何かしらのエラーでデプロイが失敗している可能性があるので、デプロイログをよく読んでみてください。

```
$ git push heroku master

Initializing repository, done.
Counting objects: 161, done.
(…中略…)
-----> Compressing... done, 23.1MB
-----> Launching... done, v7
       http://evening-thicket-3022.herokuapp
.com/deployed to Heroku

To git@heroku.com:evening-thicket-3022.git
 * [new branch]      master -> master
```

デプロイが確認できたら、**heroku open**コマンドを使って自分のアプリケーションを開いてみましょう。

git push heroku masterの裏側

ここでは少し**git push heroku master**の裏側で何が行われているのかについても説明したいと思います。

リポジトリがpushされたとき、Herokuではその中身を見てどの言語のアプリケーションなのかを判断します。たとえば、Gemfileが直下にあればRubyのアプリケーションだと判断し、デフォルトのRubyのbuildpackを走らせます。package.jsonがあればNode.jsといった具合です。自分のカスタムのbuildpackをBUILDPACK_URL環境変数を使用して指定することもできます。

Rubyのbuildpackでは、Railsアプリケーションかどうか、バージョンは何かなどを判断し、それぞれに合わせた処理が走ります。Railsアプリケーションなら、bundle installをしたあとにrake assets:precompileを走らせます。どのような処理が走っているのか詳しく知りたい人は、Buildpackがオープンソースになっているのでgithubからコードを見ることができます。それぞれのリポジトリへのリンクはBuildpackのページ[注4]を参照してください。

Buildpackの処理が終わると、その結果と合わせてslugと呼ばれるものが作られ、それがdynoにデプロイされます。

Heroku Button

さてみなさん、なんとかアプリケーションがデプロイできたかと思います。Gitやコマンドラインに馴染みのない人にはちょっとたいへんだったのではないでしょうか。また、いくら手軽にデプロイできるとはいえ「じゃあシンプルなHello Worldアプリケーションを3分で作ってHerokuで公開してみてください」と言われると、よほど手が速い人でないと難しいと思います。筆者も、「rails newして……いや、それは面倒だからsinatraにしようか……」などと迷っている間に3分経ってしまいそうです。

そこでお勧めなのが、最近リリースされた**Heroku Button**です。これを使うと、3分どころか1分でとりあえずHeroku上にアプリケーションが作れます。Githubでforkするくらいの気軽さで、あなたのHerokuアカウント上にアプリケーションを作ることができます。詳しい解説はぜひブログ[注5]をご覧ください。それにしてもなぜサンプルのアプリケーション名が**oshizushi（押し鮨）**なのでしょうか……。

今回はここまでです。ちょっと物足りないな、という方はアプリケーションに変更を加えて

注3）URL https://devcenter.heroku.com/articles/keys
注4）URL https://devcenter.heroku.com/articles/buildpacks
注5）URL https://blog.heroku.com/archives/2014/8/7/heroku-button

pushしてみたり、自分のHeroku Buttonを作ってみたりしましょう。

実は、第1回の記事はサンフランシスコに引っ越す前に書いていたのですが、引っ越しは無事終わり、現在はサンフランシスコ生活を楽しんでいます。サンフランシスコは夏だというのに、夕方には上着が必要なほど涼しいです。サンフランシスコは西海岸カリフォルニアだから、年中晴れていてみんな短パン・ショーパンにTシャツ！といった姿を想像されている方も多いかと思いますが(少なくとも筆者はそうでした)、実は、年中涼しいです。夏でも気温は高くても20℃前半で、夜などは寒いときで10℃前半となります。昼のおひさまが一番出ているときにうっかりTシャツ1枚で出かけてしまうと、夕方には寒さでぶるぶるしてしまうことでしょう。

サンフランシスコの住宅事情

サンフランシスコは現在、世界中で最も家賃の高い地域の1つに数えられます。これは、いわゆる「高給取り」であるエンジニアたちが多く住んでいることも大きな要因の1つです。サンフランシスコに住みたいという人の数が多いので家賃が上がり、上がってしまっている家賃を「払えてしまう」高級取りの人たちが多く移り住んできて……といったようなインフレーションにも思える現象が発生しています。

8月に公開された「The San Francisco Rent Explosion: Part II」[注6]というブログでは、サンフランシスコを細かい地域に分けて、それぞれの地域での家賃の中央値についての分析がなされています。1年ほど前にこのPart Iを日本で見ていたときには「またまたそんな、大げさな」と思っていました。ですが、単身サンフランシスコに乗り込んでこの時期にアパートメントハンティングをした筆者からすると、この分析は「だいたい合ってる」の範囲内なのです。

たとえば、1人暮らしですとStudio(日本のワンルーム／1K)か1BR(日本の1DK／1LDK)が候補になるかと思いますが、ブログによるとこの中央値は、$2,300と$3,120となっています。もちろんこれは単なる中央値ですので、物件や地域によって、これより低いところも高いところもたくさんあります。たとえば、治安が良かったり交通の便が良い地域では、とても広いとは言えない7畳ほどの物件が、平気で月20万円ほどします。ブログの中の一番最初の地図を見ていただくと、「あれ、でも真ん中の方にも安価なところがあるじゃない」と思われるかもしれないですが、この「Tenderloin」と呼ばれる地区は誰もが「あそこはやめておきなさい」と言うほど治安の悪いところなのです。

また、気に入った物件を見つけられたとしても、その後がたいへんです。

サンフランシスコでは、Craigslist[注7]などの情報サイトで物件の情報収集をし、気に入ったものがあればオープンハウスと呼ばれる内覧会に行きます。会場には契約書があり、気に入れば契約書に記入して不動産業者、または大家さんに渡します。

このオープンハウス、常にたくさんの人が内覧に来ており、その人たちの多くがその物件に応募します。不動産業者はそれぞれの申込書類を見て、誰を入居させるか決めるのです。

ちょっとびっくりする内容だったのではないでしょうか。サンフランシスコへの不満がちょっぴりにじみ出る話だったと思いますが、いいところですのでぜひ遊びに来てください！SD

注6) URL http://priceonomics.com/the-san-francisco-rent-explosion-part-ii/
注7) 広告が収集・分類され、掲載されているWebサイト URL http://www.craigslist.org/about/sites

サーバーワークスの瑞雲吉兆仕事術

第3回　コンシューマライゼーションはベルリンの壁崩壊と同じか？

第1回目ではモバイルに、第2回ではクラウドに出会ったエピソードをお話させていただきました。モバイルに出会うことでツールが替わり、そしてクラウドに出会うことでアプリケーションとのつきあい方が変わりました。私たちはこうした経験を通じて、以前本誌にも寄稿した「社内LAN撲滅運動（2013年3月号）」という取り組みに突き進むことになります。

Writer ㈱サーバーワークス　代表取締役　大石 良（おおいし りょう）／ http://blog.serverworks.co.jp/ceo/

ISMS認証取得のきっかけ

2011年頃から急激に大手企業との取引が増加し、「セキュリティ体制はどうなっていますか？」と聞かれることが多くなりました。AWS専業を銘打っていた当社にも「本格的にAWSを使いはじめたい」というリクエストが急増したのですが、当初の筆者たちの予想に反し、大企業から使い始めるということが起きました――クラウドは企業規模に関係なくコンピュータを柔軟に利用できますので、中小企業にこそ大きなメリットがあると思っていた筆者たちは事業規模の小さい会社がおもな顧客になるだろうと予想していました――こうした事情もあり、筆者たちはクラウド時代のセキュリティ、そして社内の制度について考えることになったのです。

ベルリンの壁

2009年くらいから、セキュリティ対策をやり過ぎた、もしくは方向を間違えていたために、そこで働いている方々の生産性が落ちたり、モチベーションを失ったりするという様子を目にする機会が増えてきました。

よく話を聞いてみると、「会社から支給されるPC以外使ってはいけない」「会社に入る前にはチェックがあって、スマホなどは持ち込んではいけない」「PCを持ち出したりしてはいけない」「アクセスできるサイトには制限を課す」などさまざまなルールが定められているようで、筆者はこれを「情報共産主義」などといって冷やかしていました。武装は配給制で、ベルリンの壁があって社内外を自由に行き来できない、というわけです。

もちろん、セキュリティは大切です。そのための適切な措置も当然必要です。ただ、そのために数千人、数万人の社員を抱える企業が、箸の上げ下げを含めて細かく指導され、会社の敷地に入るたびにチェックを受けるさまを見ると、何とも言えないバランスの悪さを感じざるを得ませんでした。

ITというのは道具であり、計算を肩代わりしたり、生産性を高めたり、そういった「得られるもの」があって初めて適切なIT投資といえるはずです。ところが情報共産主義を強いている企業では、「守り」一辺倒で「IT投資によって得られるもの」への注意が欠けているケースが多く見られたのです。

社内LAN撲滅運動

筆者たちは、こうした生産性を封じる「負のセキュリティ対策」を行うのではなく、もっと前向きな、売上の上がるセキュリティ対策を講

COLUMN　「金盾」と「ベルリンの壁」の違い

金盾という言葉をご存じでしょうか？　グレート・ファイヤーウォールとも呼ばれている中国の検閲システムで、中国国内から「天安門事件」などのキーワードで検索ができないようにしているしくみとして知られています。私は本稿中でベルリンの壁のことを「社内と社外を隔てる壁」の比喩で用いましたが、金盾にはベルリンの壁とはちょっと異なる意味があるようなのです。

金盾は、世界で流行っているコミュニケーションツール（facebook、twitter、LINE etc.）をことごとくブロックしており、中国国内では一部地域を除いて使えない状況です。ところが、どうもこれが「関税」として機能しているようなのです。

日本には、Googleやfacebook、twitterなどの米国で流行しているサービスがすぐに入ってくるため、なかなか「これからGoogleやfacebookを倒すぞ！」という威勢の良いベンチャーは生まれにくい環境にあります。ところが中国ではこれらが金盾でブロックされているので、BaiduやWeibo、WeChatのように、米国ですでに成功しているモデルを中国国内で再現しやすい環境にあるのです。そして国内で十分に世界で戦える素地を整えてから、世界に挑戦することができる。つまり「金盾が、国外コミュニケーションツールを中国国内に輸入させないための関税」としても機能しているということなのです。

私はこのやり方は間違っていると思いますし、自分が住む国にはこんなものを絶対に導入してほしくないと思うものの、このようなやり方でベンチャーを育てているという国があることも知っておいて損はないと思います。

じることにしました。キャッチフレーズは「社内LAN撲滅運動」です。

社内LANにサーバを残しておくと、物理的な対策で相当な手間がかかってしまいます。何より、社内に置いてあることで「LANだから安全」という誤った考えを持ってしまうことを懸念したのです。いつの時代でも、一番深刻な情報漏洩は社内から起こるものです。

そして、ひとつひとつシステムの棚卸しをしていった結果、「社内LANに残したほうが（クラウド化するよりも）セキュリティレベルが上がるものは何一つない」と判断するに至りました。サーバとしては2台だけ技術的な理由で社内に残すことにしましたが、後はすべてクラウド化し、ISMS認証を取得しました[注1]。

少し上がることは事実ですので、慎重な計画のもと、次のステップで実施する予定です。

BYOD（Bring Your Own Device）

そしてもう1つ、社内LAN撲滅運動とあわせて筆者たちが力を入れて取り組んだことがあります。それがBYODです。単にセキュリティ対策を実現する、という観点では会社支給の携帯、スマホを社員に配布したほうが都合は良いのですが、筆者たちのテーマである「売上が向上するセキュリティ対策」という点で見た場合、どうしても社員のモチベーションは下げたくありません。iPhoneが使いたいという社員にAndroidを渡してもガッカリされるだけなのはわかりきった話で、やはり自分で使う端末は自分で選んだものを使ってもらいたいものです。料理人がお店から包丁を支給されることはないのと同じで、プロのエンジニアであれば自分の道具は自分で選び、自分でチューニングするほうがよいと考えたのです。

もちろん、BYODになれば社員の持ち物に

注1） 詳しくは本誌2013年3月号の「社内LAN撲滅運動」か筆者のブログ（http://blog.serverworks.co.jp/ceo/?p=328）を参照ください。

COLUMN 社内に残った2台のサーバ

筆者たちが、いまだクラウドに移行していない2台のサーバのうち、1台はIP電話用のSIPサーバです。もう1台はActiveDirectoryのドメインコントローラです。SIPサーバはクラウド化を進めていた2011年当初に「安心して任せられそうだ」というクラウドPBXのサービスがなかったためにやむなく残しておいたのですが、今は複数のサービスが大手通信事業者から提供されているので、そのサービスを使うことでSIPサーバも廃棄する予定です。

一方ドメインコントローラは少し配慮が必要です。AWSでは起動時にF8キーを押すことができないため、そのままAWSに移行してしまうとDSRM(ディレクトリサービス復元モード)が利用できないのです。もちろん、適切なバックアップなど運用でカバーすればドメインコントローラをAWSに移行可能ですが、難易度が少し上がることは事実ですので、慎重な計画のもと、次のステップで実施する予定です。

なお、BYOD化を進めるにあたり、グループポリシーの配布はMDM(Model Device Management)側に任せてしまったため、ActiveDirectoryはID管理機構としてのみ利用しています。「それなのにActiveDirectoryが必要なのか」という議論もあったのですが、純粋にIDを管理するDBとしてActiveDirectoryは非常に優れていることに加え、クラウド上で提供されるSSO(Single Sign-On)のサービスもほとんどActiveDirectory対応がされていることから、クラウド時代でもID基盤としてActiveDirectoryは有益だと考えています。

はなりますが、紛失時に会社の情報などが漏れることのないよう、MDMのクライアントを必ずインストールしてもらうことにしました。

BYOD加速の工夫

しかし、これだけでは少し足りないものがありました。このルールのままBYODをすると「会社は得をするが、社員は何も得をしない」という状況になってしまったのです。会社としては、端末を購入するお金もかからないし、セキュリティ対策も最小限で済むという2つの大きな金銭的メリットがあるのに対して、社員からすれば「本来会社から支給されてもいいはずの端末代金を、結果的に個人で負担する」ことになってしまうわけです。

こうした金銭的負担の不均衡を解消するため、PCは月3,000円、スマホは月2,000円、モバイルルータは月1,500円など、BYODする端末の種類に合わせて補助金を出すことにしました。結果として、この取り組みはうまくいき、BYODによるモチベーションの向上、セキュリティレベルの担保といった当初の目的は達成できているのではないかと思います[注2]。

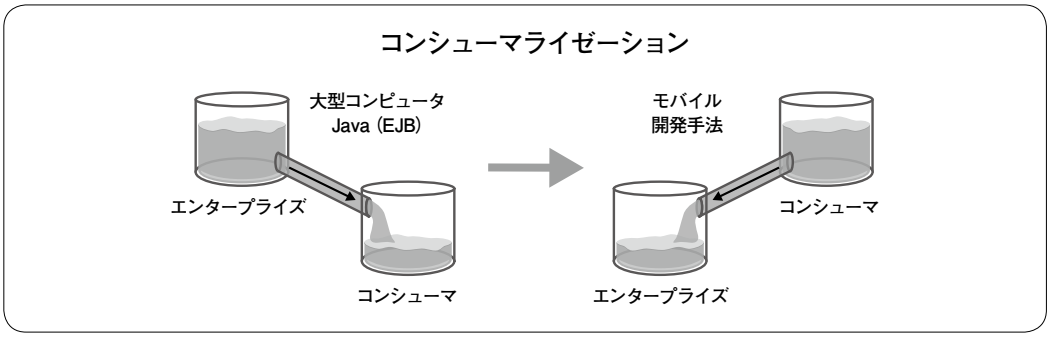
コンシューマライゼーション

ここまでの流れからわかるように、これまでエンタープライズの世界では相手にされなかったが、一般消費者は好んで使いそうなもの、たとえばiPhone、Gmail、MacBook Airといった「消費者向けサービス、製品を使いたい」という要求を率先して取り込んできました。こうした動きのことを「コンシューマライゼーション」と呼びますが、「新しい技術の流れる方向」が変わってきていることは、みなさんも感じることが多いと思います。

1990年頃のコンピュータ業界では「ダウンサイジング」という、大型で強力なコンピュータ

注2) まだ母数が少ないためデータに裏付けされたわけではありませんが、BYODによって端末の紛失事故も確率的に減るのではないかと期待しています。誰だって、会社支給の端末より自分の端末のほうを大切にしますよね?

▼図1　コンシューマライゼーションの流れの変化



が一般企業・消費者でも利用できるように「サイズを小さくする」ことによってコンピュータの普及が図られました。これは「エンタープライズ用途が最先端で、時間差でコンシューマ側に技術が降りていた」ことを意味します。2000年頃のJavaも同じ状況です。2000年頃の先端エンジニアといえば難解なEJB（Enterprise JavaBeans）をゴリゴリ書く人で、エンタープライズ用途でJavaを適用する事例が最高にクールだったわけです。

ところが今はこの流れが逆になっています。Ruby、Python、Scala、Goなど「いかにもエンタープライズ用途で使われなさそうなコンシューマプロダクト向け言語」を使いこなす人がイケてるエンジニアですよね？　開発手法でもアジャイルやCI（継続的統合）などの考え方は、おもにコンシューマ向けサービスで十分に使われた後、エンタープライズの世界にもゆっくり降り始めるという順番で、昔とは明らかに逆になっています（**図1**）。

おそらく今エンタープライズ開発の現場にいらっしゃる方が持っている「技術が伸びないかもしれない」という危機感は、コンシューマ向け製品・サービスを作っている人たちがどんどん技術を伸長しているのに対して、エンタープライズの領域にその取り組みが流れてくる動きが遅々として進まないことに原因があるのではないかと筆者は考えています。

次回

次稿では、私たちが「情報民主化」を進めた結果、社内でどういうことが起きたのか。また、コンシューマライゼーションの時代に企業向けSIなどをやっているエンジニアは自分のキャリアをどう考えればいいのか、筆者の考えを伝えたいと思います。SD

㈱サーバーワークス
代表取締役
大石 良（おおいし りょう）

・昭和48年7月20日新潟市生まれ
・コンピュータとの出会いは10歳の頃
・当時はPC-8001にベーマガのプログラムを入力する日々
・コンピュータの購入は11歳／SHARP X1
・中2のときに初めてプログラムが書籍に掲載
・高校入学記念にX68000を購入
・大学生のときにパソコン通信開始。本格的にシェアウェアを販売
・総合商社でインターネットサービスプロバイダ事業に携わる
・2000年にECのASPを立ち上げるべく起業

思考をカタチにするエディタの使い方

るびきち流 Emacs超入門

Writer | るびきち twitter@rubikitch | http://rubikitch.com/

第6回 検索、置換でピンポイント編集!

長いコード・文章を書いていて間違いに気づいたとき、始めから1つずつ修正するのは骨が折れますよね。そんなときは、「occur」、「grep」での全検索、「query-replace」での文字列置換がとても便利です。それらの発展として、複数のファイルの中身を変更できる「wgrep」も紹介します。

検索と置換

ども、ドメイン取得以降さらにEmacs愛が加熱したるびきちです。前回は日常的にEmacsを使ううえで便利な機能を紹介しました。カーソル移動と入力支援はテキストエディタの両輪となる機能です。しっかりと押えておきましょう。

今回取り上げるのは、検索と置換です。前回でもisearch、正規表現isearchを取り上げましたが、それらはたった1ヵ所が対象です。今回は、一度に複数個所を編集する方法です。これも知っておくと楽しくなってきます。

この前、興味本意でメモ帳を触ってみましたが、編集機能があまりに貧弱過ぎて発狂しそうでした(笑)。

置換

順番が逆になってしまいますが、まずは置換についてお話していきます。というのは、今回は検索といっても全検索であり、検索結果を編集することで実際のファイルに反映させるのが目的だからです。検索結果を編集する多くの場合は置換を使うので、最初に置換を知っておくべきなのです。

置換は多くの場面で使われます。プログラミングにおけるリファクタリングはもちろんのこと、文書作成においても用語を統一したり、データを見やすいレイアウトに整えたりなど、あらゆる場面で活躍します。もし、置換を知らなかった場合は1ヵ所1ヵ所isearchで移動して、元の文字列を消して新しい文字列に置き換えるという単純作業を何度も繰り返すハメになってしまいます。数ヵ所ならともかく、5ヵ所以上あったらうんざりですよね。

置換は前方検索、すなわちカーソル位置よりも後で行われます。よって、置換に入る前準備として置換対象の文字列よりも前にカーソルを持って行く必要があります。とくにバッファ全体を置換対象にするときは、M-<でバッファ先頭に移動しておきます。

単純な文字列置換はM-%を使います。たとえば、aをAに置き換えるときは、M-% a RET A RETと操作します。実行すると、「a」が見つかった場所にカーソルが移動し、置換するかどうか聞いてきます。

```
Query replacing a with A: (? for help)
```

このようにミニバッファに出てくるので、まだ使ったことのない方は「?」を押してみましょう。*Help*に操作方法が出てきますね。とくに「y(カーソル位置を置換する)」、「n(カーソル位置を置換しない)」、「!(すべて置換する)」、「q

(置換をやめる)」を覚えていれば困りません。

これは置換するかどうか毎回尋ねてくるので「query-replace」というコマンド名です。すべて置換することがわかっている場合は尋ねないほうがうれしいですよね。M-%で置換前後の文字列を入力してから「!」を押せば一気に置換してくれます。尋ねないバージョンM-x replace-stringも存在しますが筆者は使っていません。M-%は尋ねてほしいときとほしくないときの両方をまかなえるからです。

置換には正規表現も使えます。M-%の正規表現版C-M-%(query-replace-regexp)があるのですが、慣れるまで難しいです。C-M-%というイカれたキーバインドもそれを暗示しています。正規表現を知っていても確実にマッチさせるのは難しいものがあります。実は筆者も複雑な正規表現置換は使いきれていません。ですが、御安心ください。初級者にでもそれと同等のことができる方法があるのです。

バッファ内を全検索する

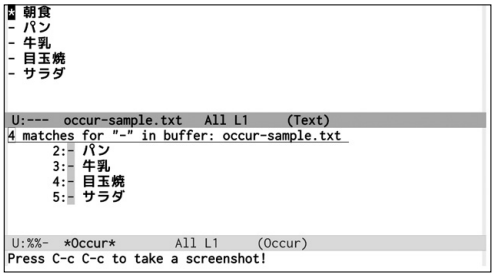
isearchはカーソル位置から見て次の位置を検索しますが、バッファ全体を検索したいときがありますよね。端的に言えばバッファ内grepです。特定のキーワードが含まれる行をリストアップするときにものすごく役立ちます。プログラミングでいえば関数定義のリストがほしいときがありますよね。

この場合に手軽に使える標準コマンドがM-s o(occur)です。実行すると、表示する行の正規表現を訊いてきます。たとえば、**図1**の上の画面に表示されているテキストにおいてM-s oのあとに「-」を押すと、「-」が含まれている行が下に表示されます。

occurの実行後、マッチした行にジャンプできます。そのための方法は2つあります。

①*Occur*バッファから、該当行にジャンプする

▼図1 M-s oの実行例



②元のバッファにいながら、マッチした行にジャンプする

わかりやすいのは①の方法です。ただoccurを実行したあと、カレントウィンドウは元のバッファのままで、*Occur*バッファにはありません。そのため*Occur*のウィンドウを選択する必要があります。**図1**の*Occur*ウィンドウの4行目(目玉焼)にカーソルを移動して RET を押すと、occur-sample.txtの該当行に移動します。

②の方法はマッチした行に順番に移動する方法です。言ってみれば、isearchにマッチ行一覧が付属したようなものです。M-g M-n(next-error)で次のマッチ行、M-g M-p(previous-error)で前のマッチ行にカーソルを移動します。この2つのコマンドはoccur以外にも後述するM-x grepやコンパイラのエラー行に進むなどの用途があります。

2つの方法は場合によって使い分けてください。特定の行にジャンプしたいのならば、①の方法が良いです。そのときは*Occur*バッファでisearchをするなどして絞り込むことになるでしょう。対して、すべてのマッチ行を見たいのならばウィンドウ選択なしで使える②の方法が良いです。

①の方法がわかりやすくて大好きという人がいると思います。かつての筆者もそうでした。それならば解決策はいくつかあります。

❶ウィンドウ選択を楽にする
❷ace-jump-modeで見えている*Occur*の行にジャンプする

❸マウスで*Occur*の行をクリックする
❹元からoccurのウィンドウを選択させる

ウィンドウ選択をしやすくする

❶のウィンドウ選択を楽にするというのは、C-x oを別のキーに割り当て直すことです。Emacsで複数のウィンドウを使っていると、本当に頻繁にC-x oを使います。それならば押しやすいキーに割り当て直すべきです。筆者は大昔からC-tに割り当てています。このたった1行の設定で、ウィンドウ選択が本当にやりやすくなり、ウィンドウ選択をするほかのコマンドがいらなくなるくらいです。

```
(global-set-key (kbd "C-t") 'other-window)
```

元のC-tはカーソル直前の2文字を入れ替える「transpose-chars」です。このコマンドが好きという人もいますが、筆者は使っていません。タイプミスでよくあるのが2つのキーの入れ違いです。たとえばlsをslとタイプしてしまうとかです。それに気づいたら即座にC-tを押せば直ります。元からなじめなかったのもありますが、何よりローマ字での日本語入力で使えないのが痛いです。

「ください」はローマ字入力だとkudasaiですが、iとaを入れ違いにしてkudasiaになると「くだしあ」になってしまいます。そこでC-tを押しても「くだあし」になってしまいます。C-tを押すのがクセになっている人にとって、日本語入力でそれが使えないのはものすごい苦痛です。ローマ字入力に対応したtranspose-charsが待ち望まれます。それまではC-tはいらない子というのが僕の結論です。それならば頻繁に使うコマンドにC-tを譲ってあげるべきだと考えています。

マウスの代用として ace-jump-modeが大活躍

❷のace-jump-modeは前回で紹介しましたが、近距離のカーソル移動手段として超強力な方法です。isearchは画面外にも検索範囲が伸びますが、ace-jump-modeは画面内移動に特化しています。それもカレントウィンドウに限らず、ほかのウィンドウにも3ストローク以内で移動できるのです。

❸も画面内に特化した移動方法で、*Occur*ウィンドウ内をワンクリックで該当行に移動できます。occurの局面においてマウスも強力な方法ですが、筆者の場合、ace-jump-modeを使ってからEmacsでマウスに手を伸ばすことがほとんどなくなりました。

図1ではスペースの関係上横幅を取っていないので上下にウィンドウが分割されていますが、最近主流となっているワイド画面でoccurを実行すれば、左右に分割されます。左右分割されれば40〜60行も画面に表示されます。よって、*Occur*が画面内に収まるのであれば、ace-jump-modeで*Occur*のカーソルを移動してRETを押せば、元のバッファの該当行にジャンプできます。

最初から*Occur*を選択させる

❹のように、最初から*Occur*を選択する方法もあります。このアプローチが好きならば、M-x occur-and-selectを定義して使ってください。ついでにoccurに割り当てられているキーも置き換えておきます(**リスト1**)。

それをさらに推し進めた外部Emacs Lispに「color-moccur」というのがあります。occurを超強化したもので、かつての筆者も使っていました。でもcolor-moccurの機能は別の便利な方法で実現できてしまう今ではもう使っていません。お好みで。

▼リスト1　M-x occur-and-selectの定義

```
(defun occur-and-select (regexp &optional nlines)
  (interactive (occur-read-primary-args))
  (occur regexp nlines)
  (select-window (get-buffer-window "*Occur*"))
  (forward-line 1))
(global-set-key (kbd "M-s o") 'occur-and-select)
```

occurの結果を編集する

occurが提供する機能はこれだけではありません。なんと、*Occur*バッファを編集して元のバッファに反映させられるのです！　この機能を実現している外部Emacs Lispプログラムが昔からありますが、今や標準機能であるのです。

これを使えば正規表現置換に躊躇する人でも、単純な文字列置換でそれと同等の処理が行えます。正規表現置換は本記事に書ききれないほどの機能がありますが、高度な正規表現置換が使われることはめったにありません。なぜならば、直観的でわかりやすいoccur編集で間に合うからです。

今、**図1**の状態にあるとします。つまり、occur-sample.txtでM-x occur -を実行した直後です。「-」を「**」に置き換えてみましょう。次の手順で操作します。

①*Occur*バッファを選択する(C-x o)
②eを押してoccur-edit-modeに入る
③M-% - RET ** RETで置換する

すると、置換しただけで元のバッファが変更されます(**図2**)。M-x occur-and-selectを使えば①は省略できます。

カスタマイズはほどほどに

なお、いきなりoccur-edit-modeに入るコマンドM-x occur-and-editを定義することはできますが、筆者はやり過ぎだと思います。なぜなら、割に合わないからです。M-x occurの目的は普通にバッファ内grepとして使うことが多く、occur-edit-modeを使う頻度は多くありません。それにたった1ストロークでoccur-edit-modeに入れます。たかが1ストロークを節約するために頻度の低いコマンドを定義するのは割に合わないのです。M-x occur-and-editで起動するとなると、M-s o → C-x o → eでoccur-and-editを起動する場合よりもかえってストロークが多くなってしまいます。

▼図2　occur編集

```
* 朝食
** パン
** 牛乳
** 目玉焼
** サラダ


U:**-  occur-sample.txt   All L5     (Text)
4 matches for "-" in buffer: occur-sample.txt
       2:** パン
       3:** 牛乳
       4:** 目玉焼
       5:**サラダ


U:**-  *Occur*        All L5     (Occur-Edit)
```

このように、カスタマイズはやり過ぎないで一定のところで止めることが大事です。この線引きについては経験がものを言います。そして常に「このカスタマイズをすることによるメリットとコストは何なのか」と自問してください。カスタマイズをし過ぎると管理コストが発生します。なにより使用頻度の低いコマンドはそのうち忘れてしまいます。

筆者もかつて猛烈にカスタマイズしまくった時期がありました。世界有数レベルでカスタマイズに没頭していました。間違いなく日本一Emacsをいじくり回していました。今はかなり落ち着いていますが、その時期があったからこそ、今こうしてあなたにレッスンをお伝えできるのです。

grepの結果にジャンプする

ここまでは、単一のバッファに対して検索・編集を行う内容でした。M-x grepはEmacsの中でgrepプログラムを動かし、検索結果にジャンプするものです。これにより、複数のファイルやディレクトリにまたがる検索もできるようになります。しかもelispよりもはるかに高速に。

M-x multi-occurは複数のバッファに対してoccurしますが、あらかじめ検索対象のバッファを指定する必要があります。仮にすべてのバッファを検索対象にしたら、遅過ぎて日が暮れます。なぜなら、Emacs Lispでgrepの真似事をしても、しょせんは子供の遊びレベルだからで

す。Emacs Lispはユーザインターフェースを記述するのは得意ですが、大量のデータを扱うのが大の苦手です。おまけにマルチスレッドやマルチコアに対応していないので、現在の高性能なコンピュータの性能を活かせません。

対してgrepプログラムは検索のプロです。とくにGNU grepは爆速で、数GB程度のデータなら数秒あれば結果を出力してくれます。フルEmacs Lispで検索するより何千何万倍も速いです。適材適所、餅は餅屋です。

そこで、検索はgrepプログラムに任せて、表示および検索結果へのジャンプはEmacs Lispで書くという役割分担をすることにしました。それならば速度と利便性を両立できます。おまけにEmacs Lisp部分の行数も削減できます。M-x grep以外にもこの方式を採っているEmacsコマンドはたくさんあります。

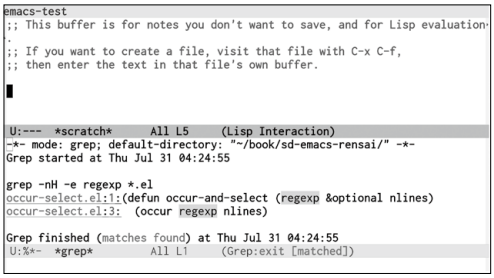
M-x grepを実行するとミニバッファに「Run grep (like this): 」というプロンプトと「grep -nH -e 」などの初期入力が出てきます。grepのオプションは環境によって異なるのですが、この調整はM-x grep側がやってくれます。

大事なのは「-n」、「-H」オプションでそれぞれ検索結果の行番号、ファイル名を出力することです。検索結果にジャンプするためにはこれらの情報が必要です。「-H」オプションが使えない場合は、検索ファイルの指定のあとに/dev/nullなどのヌルデバイスを付加し、強制的にファイル名を出力させるようにします。

あとはいつもどおりgrepプログラムを実行するコマンドラインを入力するだけです。つまり正規表現とファイル名を入力します。もちろんほかのオプションを入力してもかまいません。

M-x grepを実行したら、別ウィンドウの*grep*バッファに検索結果が出てきます(**図3**)。grepを実行するには時間がかかることがあるので、grepプログラム実行中でもEmacsの操作ができます。grepプログラムが出力するたびに*grep*バッファが更新されます。実行中であっても、現時点での検索結果にアクセスできます。

▼図3　grepの実行例



M-x grepの検索結果にアクセスする方法は、M-x occurとまったく同じです。*grep*バッファで RET を押すか、M-g M-n／M-g M-pを実行するかです。

M-x grepには、もうひとつ大事な特徴があります。それはほかのプログラムも実行できるということです。たとえば、grepの代わりにgzipされたファイルも検索するzgrepを実行できます。オプションを設定する必要がありますが、ソースコード検索に特化した高速grepのack/ag/ptを実行させることもできます。grepそのものではなくてソースコード検索ツールmilkode(gmilk)をも実行できてしまいます。grep -nH形式、すなわち「ファイル名:行番号:」を出力してくれるプログラムであればなんでも良いわけです。M-x grepのように外部プログラム丸投げ方式のEmacsコマンドは高速性だけでなく柔軟性をも獲得したのです。

grepの結果を編集してファイルに反映させる奥義

M-x occurではoccur-edit-modeで検索結果を編集できますが、M-x grepでも同じようなことができないでしょうか？　それを実現するパッケージがwgrepです。wgrepとはWritable Grepのことで、*grep*を編集することで元のファイルにも反映させていくものです。

wgrepはMELPAというパッケージ登録所に登録されているので、パッケージの設定(**リスト2**)さえしてしまえばEmacsの中でインストールできます。

▼リスト2　パッケージを使うための初期設定

```
(package-initialize)
(add-to-list 'package-archives '("marmalade" . "http://marmalade-repo.org/packages/"))
(add-to-list 'package-archives '("melpa" . "http://melpa.milkbox.net/packages/") t)
```

```
M-x package-refresh-contents
M-x package-install wgrep
```

そして、wgrepの設定もしておきます。

```
(require 'wgrep)
(setq wgrep-change-readonly-file t)
(setq wgrep-enable-key "e")
```

*grep*バッファでeを押すことで編集可能になります。eを選んだのは、occur-edit-modeとそろえるためです。似たコンセプトのコマンドのキー割り当てをそろえておくことで、ストレスなく使うことができます。

図3の状態でwgrepを使うには、*grep*バッファにウィンドウを切り替え、eを押します。すると、

```
Press C-x C-s when finished or C-c C-k to
abort changes.
```

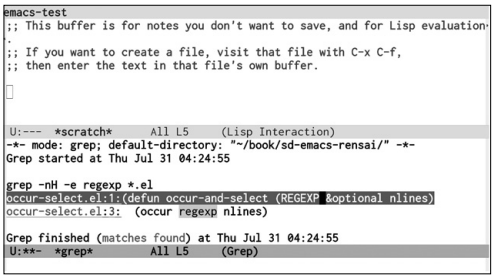
とメッセージが出てきます。つまり、*grep*バッファの変更をファイルに反映させたければC-x C-sを、取り止めたければC-c C-kを押せということです。

図4は最初のregexpをREGEXPに置き換えたところです。ここでC-x C-sを押すと実際のファイルに反映されますが、その時点ではファイルは保存されていません。反映された部分は**図5**の1行目のように色が付きます。occur-select.elが修正済み状態になっている(モードラインの左に「**」と表示されている)ことに注見してください。

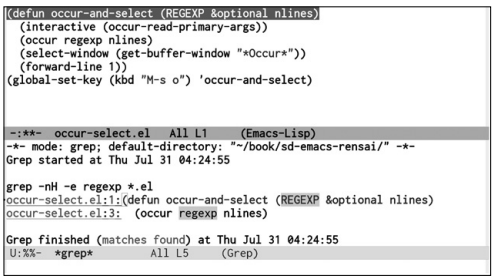
wgrepで複数のファイルを変更し、すべてのファイルを保存するにはC-x sのあとに!を押してください。

このようにwgrepは大きな編集をこなせる超強力なコマンドです。言うまでもなくM-x grepの強みは生きており、任意のgrep -nH形式のプログラムの出力結果を編集してファイルに反映できます。wgrepはさほど使用頻度が高いわけではありませんが、飛び道具として覚えておいてください。

▼図4　wgrep実行例~検索結果を編集



▼図5　wgrep実行例~ファイルに反映



終わりに

今回は置換と全検索を取り上げました。ここまでの段階で、かなり複雑な編集ができるようになったはずです。とくに検索結果を編集する機能には驚かれたと思います。

筆者のサイトrubikitch.comではEmacsの情報発信基地を目指すべく、定番情報や最新情報を日々更新しています。さらにステップアップしたい方はメルマガ登録[注1]お願いします。Happy Emacsing！SD

注1) URL http://www.mag2.com/m/0001373

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7回　AWS APIでのデジタル署名の全体像を明らかにする①

Writer　波田野 裕一（はたの ひろかず）／運用設計ラボ　operation@office.operation-lab.co.jp

トピック

今夏もAWSは新サービスや新機能のリリースラッシュでした。本トピックではこれらのうち、AWSのDNSマネージドサービスであるRoute53の新機能として追加された「ドメイン名の登録・管理」機能について簡単に取り上げます。

この新機能はドメインの新規取得、登録情報、外部レジストラから、もしくは外部レジストラへの移管を管理できる機能で、取得したドメインについてはそのままRoute53上で運用できることも特徴となっています[注1]。

たとえば、Route53から新規ドメインを取得すると、Hosted Zoneの作成、Whoisデータベースへの Name Serverの登録など、従来であれば初期に手動で必要だった作業が自動的に行われるため、ドメインを取得してしばらくすると、何もしなくてもインターネット上でそのドメインのNSレコードを引くことができる、という点は驚きでした。

AWS CLI 1.3.25以降であれば、route53 domainsコマンドが利用できるようになっています。たとえば利用可能なドメインを探すときには、次のコマンドを1発叩けば即座に結果がわかるので、CLIの便利さを伝えるサンプルとしても使えるのではないでしょうか。

・コマンド実行例：

```
$ aws --region us-east-1 route53domains ↵
check-domain-availability --domain-name ↵
example.com
```

・結果例（取得できる例）：

```
{
  "Availability": "AVAILABLE"
}
```

・結果例（取得できない例）：

```
{
  "Availability": "UNAVAILABLE"
}
```

この新機能追加に伴い、route53domains.us-east-1.amazonaws.comが新たなエンドポイントとして追加されています。AWS CLIでも、リージョンとしてus-east-1を指定する必要があるので注意してください。

もう1つ、AWS CLIのトピックとして比較的大きなリリースが8月上旬にありました。第6回で紹介したemrコマンド（Elastic Map Reduce）がPreview版から正式版に移行[注2]しました。あわせてAWS CLIのバージョンも1.4.0とメジャーバージョンアップしています。

これら新機能にご興味がある方は早速AWS CLIをアップデートしてみましょう。

注1）URL http://aws.typepad.com/aws_japan/2014/08/route-53-domain-reg-geo-route-price-drop.html

注2）URL http://aws.amazon.com/releasenotes/8270985098793055

・AWS CLIの更新コマンド実行例：

```
$ sudo pip install -U awscli
```

個人的には、あとはCloudFrontが正式版になってくれれば、と心待ちにしているところです。

今回の流れ

今回から、数回にわたってAWS APIを直接操作するために必要なデジタル署名について解説していきます。

2014年8月現在、以下の3つのデジタル署名作成方法が提供されています。

・Signature Version 2
・Signature Version 3
・Signature Version 4

今回は署名バージョンの概要と、3つの署名方法の中で最もシンプルなSignature Version 3について実例も含めて解説します。

署名のバージョン

2014年8月現在の各AWSサービスにおける推奨署名バージョンは**表1**のようになっています。

ほとんどのAWSサービスはSignature Version 4(以下「v4」)に対応しています。このv4は3つの署名方法の中で一番複雑な手順になっています。

逆に、v4に対応していないのは、Amazon Route53(以下「Route53」)、Amazon Elastic Cloud Computing(以下「EC2」)およびEC2と同じAPIを利用しているAmazon Virtual Private Cloud(以下「VPC」)、東京リージョンでは提供されていないAWS Import/Exportのみとなります。このうちRoute53はSignature Version 3(以下「v3」)のみ対応で、残るEC2(VPC)、

▼表1　AWSのサービスと推奨署名バージョン

	ネットワーキング	推奨署名
インフラストラクチャーレイヤ	Amazon Route 53	v3
	Amazon Route 53(Domains)	v4
	Amazon Virtual Private Cloud(Amazon VPC)	v2
	Elastic Load Balancing	v4
	コンピューティング	
	Amazon Elastic Compute Cloud (Amazon EC2)	v2
	Auto Scaling	v4
	ストレージ	
	Amazon Simple Storage Service(Amazon S3)	v4
	Amazon Glacier	v4
	AWS Import/Export	v2
	AWS Storage Gateway	v4
	データベース	
	Amazon DynamoDB	v4
	Amazon Relational Database Service(Amazon RDS)	v4
	Amazon ElastiCache	v4
	Amazon Redshift	v4
アプリケーションサービスレイヤ	**メッセージ**	
	Amazon Simple Notifiction Service(Amazon SNS)	v4
	Amazon Simple Queue Service(Amazon SQS)	v4
	メール配信	
	Amazon Simple Email Service(Amazon SES)	v4
	ワークフロー	
	Amazon Simple Workflow(Amazon SWF)	v4
	メディア変換	
	Amazon Elastic Transcoder	v4
	コンテンツ配信	
	Amazon CloudFront	v4
	分散処理	
	Amazon Elastic MapReduce	v4
	データ連携	
	AWS Data Pipeline	v4
	検索	
	Amazon CloudSearch	v4
	ストリーミング	
	Amazon AppStream	v4
	分析	
	Amazon Kinesis	v4
デプロイ・アドミニストレーションレイヤ	**モニタリング**	
	Amazon CloudWatch	v4
	アイデンティティ & アクセス	
	AWS Identity and Access Management(IAM)	v4
	AWS Security Token Service(AWS STS)	v4
	デプロイ & マネジメント	
	AWS Elastic Beanstalk	v4
	AWS CloudFormation	v4
	AWS OpsWorks	v4
	AWS CloudTrail	v4
	AWS Support	v4

AWS Import/Export は Signature Version 2（以下「v2」）だけに対応、となっています。

事前準備

AWS APIに直接アクセスするには、下記の環境が必要です。これら環境の詳細、およびAWS APIにリクエストを投入して'404 Bad Request'が返ってきた場合のための検証環境については、連載第2回「AWS APIの利用方法と環境の構築」をご参照ください。

AWS認証情報

シェルスクリプトから利用する認証情報として、**リスト1**のファイルがあることを前提にしています。ご利用のIAMユーザの認証情報にあわせて作成してください。

コマンド

今回は次のコマンドを利用します。環境によっては標準でインストールされていない可能性がありますので、事前に確認ください。

▼リスト1　~/.aws/default.rc

```
aws_access_key_id=AKIXXXXXXXXX0EXAMPLE
aws_secret_access_key='xXxxxXXxxXXXX/ﾌ
X0XXXXX/xXxXxxXXEXAMPLEKEY'
```

・opensslコマンド
・base64コマンド

Signature Version 3

v3は、3つの署名方法のうち最もシンプルな署名手順となっています。とくにAPIへのリクエストデータ本体と署名に必要なデータ（v3では日付情報）が完全に分離されている点が他の2つの署名方法との大きな違いとなっており、手動で実行したときにエラーになりにくいという特徴があります。

v3の署名付きリクエストデータの作成手順は**図1**のとおりです。

手順①リクエストデータの作成

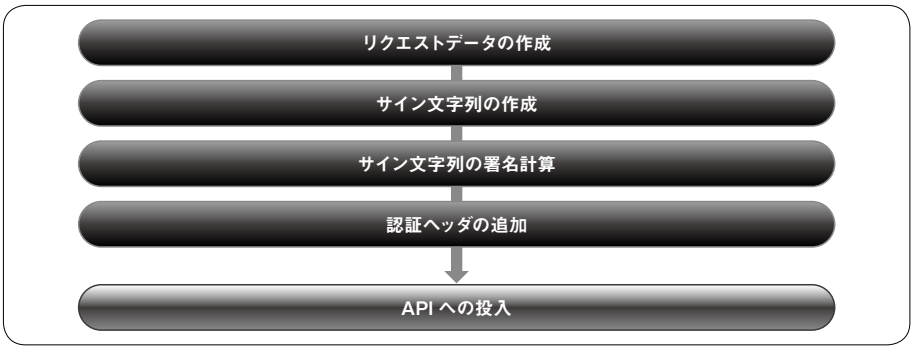
まず次のようなリクエストデータを作成します。

・リクエストデータ：

```
GET /2013-04-01/hostedzone HTTP/1.1
Host: route53.amazonaws.com

?maxitems=5
```

これは、Route53上に作成されているHosted Zoneの一覧を取得するリクエストデータです。

▼図1　リクエストデータの作成手順



最初の行の、'2013-04-01' はRoute53のAPIバージョンになります。最新のAPIバージョンについては、各サービスのAPIリファレンスページのトップ[注3]に記載があります。

2行目のHost行には、AWS APIのEndPointを記載します。Route53にはEndPointが1つしかないので、'route53.amazonaws.com' を記載しています。今回のサンプルはGETメソッドなのでヘッダー部分はこれですべてとなります(POSTメソッドの場合は、これに加えてContent-Lengthヘッダが必要になります)。ボディ部分について、今回の例では最大表示件数の指定のみ記述しています。このリクエストデータは、手順④の認証ヘッダの追加のときに必要になります。

手順②サイン文字列の作成

次に、サイン文字列(String to Sign)を作成します。サイン文字列は署名の基となる文字列群で、v3では日付情報(GMT)のみ利用します。

・入力コマンド：

```
$ LC_ALL=en
$ TZ="GMT";
$ DATETIME=`date "+%a, %d %b %Y %H:%M:
%S %Z"`
$ echo ${DATETIME}
```

・実行結果(例)：

```
Mon, 18 Aug 2014 03:49:25 GMT
```

この日付文字列がv3におけるサイン文字列すなわち「署名に必要なデータ」となります。デジタル署名においては、最後の行末に不要な改行などが含まれていると、署名計算が正しくできないため、次のように変数の最後にある改行を削除して一時ファイルに保存するとよいでしょう。

・OSXの/bin/shの場合(例)：

```
$ FILE_BEFORE_SIGN="${HOME}/tmp/v3.tmp"
$ echo "${DATETIME}\c" > ${FILE_BEFORE_
SIGN}
```

・bash、Linuxのsh(bash)、FreeBSDのsh(ash)の場合(例)：

```
$ FILE_BEFORE_SIGN="${HOME}/tmp/v3.tmp"
$ echo -n "${DATETIME}" > ${FILE_BEFORE_
SIGN}
```

最後の改行が削除されているかどうかは、echoコマンドでファイルを表示すれば確認できます。ファイル内容と次のコマンドプロンプトが改行されずに表示されていればOKです。

・コマンド(プロンプト'sh-4.1$'が表示されている例)：

```
sh-4.1$ cat ~/tmp/v3.tmp
```

・改行が削除されている場合(成功例)：

```
Mon, 18 Aug 2014 03:49:25 GMTsh-4.1$
```

・改行が削除されていない場合(失敗例)：

```
Mon, 18 Aug 2014 03:49:25 GMT
sh-4.1$
```

手順③サイン文字列の署名計算

サイン文字列の作成が完了したら、シークレットアクセスキーを利用して署名計算をします。ここではopensslコマンドを利用し、HMAC-SHA256で署名計算をします。opensslコマンドの出力そのままではバイナリデータとなってしまうので、Base64エンコードもあわせて行います。

・入力コマンド：

```
$ SIGNATURE=`openssl dgst -binary -hmac
${aws_secret_access_key} -sha256 ${FILE_
BEFORE_SIGN} | base64`
$ echo "signature: ${SIGNATURE}"
```

注3) APIへのリクエストの詳細はRoute53 APIリファレンスを参照ください(http://docs.aws.amazon.com/Route53/latest/APIReference/Welcome.html)。

・実行結果(例)：

```
XXp5XlXXXXX9xXXiXqXorXrv8XXfumXwZXXXXbgXkXw=
```

出力結果の文字列がこのリクエストにおける署名(Signature)となります。

手順④認証ヘッダの追加

署名の作成が完了したら、手順①で作成したリクエストデータに、手順②③で作成した情報をヘッダとして追加します。まず手順②で作成した日付情報をDateヘッダとして追加します(3行目)。

・Dateヘッダ情報作成(例)：

```
$ STR_DATE="Date: ${DATETIME}" && echo ${STR_DATE}
```

・Dateヘッダ情報(出力例)：

```
Date: Mon, 18 Aug 2014 03:49:25 GMT
```

次に手順③で作成した署名文字列、AWSアカウントのアクセスキーIDをX-Amzn-Authorizationヘッダとして追加します(4行目)。

・X-Amzn-Authorizationヘッダ情報作成(例)：

```
$ STR_X="X-Amzn-Authorization: AWS3-HTTPS AWSAccessKeyId=${aws_access_key_id},Algorithm=${SIGNATURE_METHOD},Signature=${SIGNATURE}" && echo ${STR_X}
```

・X-Amzn-Authorizationヘッダ情報(出力例)：

```
X-Amzn-Authorization: AWS3-HTTPS AWSAccessKeyId=AKIXXXXXXXXX0EXAMPLE,Algorithm=HmacSHA256,Signature=XXp5XlXXXXX9xXXiXqXorXrv8XXfumXwZXXXXbgXkXw=
```

上記の2つのヘッダを追加すると、リクエストデータは下記のような内容になります。

・リクエストデータ(サンプル)：

```
GET /2013-04-01/hostedzone HTTP/1.1
Host: route53.amazonaws.com
Date: Mon, 18 Aug 2014 03:49:25 GMT
X-Amzn-Authorization: AWS3-HTTPS AWSAccessKeyId=AKIXXXXXXXXX0EXAMPLE,Algorithm=HmacSHA256,Signature=XXp5XlXXXXX9xXXiXqXorXrv8XXfumXwZXXXXbgXkXw=

?maxitems=5
```

手順⑤APIへの投入

AWS APIへのリクエスト投入には、opensslコマンドをHTTPS(SSL/TLS)クライアントとして利用します。

・入力コマンド：

```
$ openssl s_client -connect route53.amazonaws.com:443
```

コマンドを実行すると、次のように入力待ちになります。

```
……(中略)……
Key-Arg   : None
PSK identity: None
PSK identity hint: None
SRP username: None
Start Time: 1408332809
Timeout   : 300 (sec)
Verify return code: 20 (unable to get local issuer certificate)
```

ここに、リクエストデータを貼り付けると、すぐにAPIからレスポンスがあります(**図2**、**図3**)。

図2、**図3**ともに、1行目が'HTTP/1.1 200 OK'となっていれば、AWS APIへのリクエストは正常に処理されています。

図3の例ではRoute53上に、'example.jp'というHosted Zoneが1つだけあり、そのHosted Zone IDは'ZONXXXXXXXXXX'で、リソースレコードセットが3つある、ということが表示されています[注4]。

注4) URL http://docs.aws.amazon.com/Route53/latest/APIReference/requests-rest-authentication.html

▼図2　Hosted Zoneが存在しない例

```
HTTP/1.1 200 OK
x-amzn-RequestId: XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX
Content-Type: text/xml
Content-Length: 197
Date: Mon, 18 Aug 2014 03:49:25 GMT

<?xml version="1.0"?>
<ListHostedZonesResponse xmlns="https://route53.amazonaws.com/doc/2013-04-01/">
<HostedZones/><IsTruncated>false</IsTruncated><MaxItems>100</MaxItems>
</ListHostedZonesResponse>
```

▼図3　Hosted Zoneが存在する例

```
HTTP/1.1 200 OK
x-amzn-RequestId: XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX
Content-Type: text/xml
Content-Length: 495
Date: Mon, 18 Aug 2014 03:49:25 GMT

<?xml version="1.0"?>
<ListHostedZonesResponse xmlns="https://route53.amazonaws.com/doc/2013-04-
01/"><HostedZones><HostedZone><Id>/hostedzone/ZONXXXXXXXXX</Id><Name>example.jp.</Name>
<CallerReference>RISWorkflow-XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX</CallerReference><Config>
<Comment>HostedZone created by Route53 Registrar</Comment></Config><ResourceRecordSetCount>
3</ResourceRecordSetCount></HostedZone></HostedZones><IsTruncated>false</IsTruncated>
<MaxItems>100</MaxItems></ListHostedZonesResponse>
```

JAWS-UGへのお誘い

さて現在、日本におけるAWS利用者の増加とともに、AWSに関連した勉強会やコミュニティ活動が活発に行われています。

なかでも、AWSの利用者が有志で運営しているJAWS-UG[注5](Japan AWS User Group：「ジョーズ・ユージー」と発音されることが多いようです)は、2014年8月現在で48支部を展開しており、全国各地で活発にコミュニティ活動を行っています。

仙台、東京、大阪をはじめとした大都市では活動の規模も大きく、数百人規模のイベントを開催し、レベルの高い内容で参加者の満足度も高いようです。

2014年7月には、AWS CLIをターゲットとした専門支部として、「JAWS-UG CLI専門支部」が設立されました。活動範囲は東京に限られていますが、2014年8月末現在で勉強会が3回開催されています(筆者も主催者の1人です)。

ネット上の記事やドキュメントももちろん有用ですが、AWSを日々使っている人同士ならではの情報交換ができるのがリアルの場でのコミュニティ活動の醍醐味です。興味が少しでもあれば、まずは身近なJAWS-UG支部に参加してみてはどうでしょうか。

注5）URL http://jaws-ug.jp/

次回は

次回は、今回解説したSignature Version 3の署名付きリクエストデータを作成するシェルスクリプトの紹介とSignature Version 2の概要について解説する予定です。SD

ハイパーバイザの作り方

ちゃんと理解する仮想化技術

第23回

bhyveにおける仮想シリアルポートの実装（その3）

Writer 浅田 拓也（あさだ たくや） Twitter @syuu1228

はじめに

今回の記事では、引き続きbhyveにおける仮想シリアルポートの実装について解説します。

UARTエミュレーションのコード

UARTレジスタの読み込みについてソースコードの解説を行います（**リスト1**）。

▼リスト1 uart_read

```
uint8_t
uart_read(struct uart_softc *sc, int offset)
{
  uint8_t iir, intr_reason, reg;

  pthread_mutex_lock(&sc->mtx);

  /*
   * Take care of the special case DLAB accesses first
   */
  if ((sc->lcr & LCR_DLAB) != 0) {
    if (offset == REG_DLL) {
      reg = sc->dll;
      goto done;
    }

    if (offset == REG_DLH) {
      reg = sc->dlh;
      goto done;
    }
  }

  switch (offset) {
  case REG_DATA:
    reg = fifo_getchar(&sc->rxfifo);
    break;
  case REG_IER:
    reg = sc->ier;
    break;
  case REG_IIR:
    iir = (sc->fcr & FCR_ENABLE) ? IIR_FIFO_MASK : 0;

    intr_reason = uart_intr_reason(sc);

    /*
```

DLLレジスタ、DLMレジスタへのアクセスを検出するためにLCRレジスタのDLABビットをチェックしている。uart_softc構造体のdllメンバ、dlmメンバの値を読み出し値として返す

DATAレジスタの読み出し値としてfifo_getcharから1文字返す

IERレジスタの読み出し値としてsc->ierの値を返す

IIRレジスタの読み出し値として現在の割り込み要因のステートをuart_intr_reasonで構築して返す

次ページへ

（リスト1のつづき）

```
     * Deal with side effects of reading the IIR register
     */
    if (intr_reason == IIR_TXRDY)
      sc->thre_int_pending = false;

    iir |= intr_reason;

    reg = iir;
    break;
  case REG_LCR:
    reg = sc->lcr;
    break;
  case REG_MCR:
    reg = sc->mcr;
    break;
  case REG_LSR:
    /* Transmitter is always ready for more data */
    sc->lsr |= LSR_TEMT | LSR_THRE;

    /* Check for new receive data */
    if (fifo_numchars(&sc->rxfifo) > 0)
      sc->lsr |= LSR_RXRDY;
    else
      sc->lsr &= ~LSR_RXRDY;

    reg = sc->lsr;

    /* The LSR_OE bit is cleared on LSR read */
    sc->lsr &= ~LSR_OE;
    break;
  case REG_MSR:
    /*
     * MSR delta bits are cleared on read
     */
    reg = sc->msr;
    sc->msr &= ~MSR_DELTA_MASK;
    break;
  case REG_SCR:
    reg = sc->scr;
    break;
  default:
    reg = 0xFF;
    break;
  }

done:
  uart_toggle_intr(sc);
  pthread_mutex_unlock(&sc->mtx);

  return (reg);
}
```

LCRレジスタの読み出し値としてsc->lcrの値を返す

MCRレジスタの読み出し値としてsc->mcrの値を返す

fifo_numcharsを呼び出し、読み出し可能なデータがあればsc->lsrにデータ着信ビットを立てる。また、常にsc->lsrへ送信可能ビットを立てる。LSRレジスタの読み出し値としてsc->lsrの値を返す

MSRレジスタの読み出し値としてsc->msrの値を返す

SCRレジスタの読み出し値としてsc->scrの値を返す

更新されたUARTコントローラのステートを元に、割り込みをアサートまたはアサート解除する

ttyデバイスのポーリング

UARTへの書き込みの場合、DATAレジスタへの書き込みをそのままttyへ出力しています。

一方、UARTからの読み込みはもう少し複雑で、ttyへの書き込みをイベントポーリングで監視して、書き込まれた文字列をRX FIFOにバッファし、DATAレジスタへの読み込み時にはFIFOから1文字取り出しています（**リスト2**）。

リスト3にttyreadのコードを、**リスト4**にfifo_putcharのコードを示します。

▼リスト2　uart_drain

```
static void
uart_drain(int fd, enum ev_type ev, void *arg)
{
  struct uart_softc *sc;
  int ch;

  sc = arg;

  assert(fd == sc->tty.fd);
  assert(ev == EVF_READ);

  /*
   * This routine is called in the context of the mevent thread
   * to take out the softc lock to protect against concurrent
   * access from a vCPU i/o exit
   */
  pthread_mutex_lock(&sc->mtx);

  if ((sc->mcr & MCR_LOOPBACK) != 0) {
    (void) ttyread(&sc->tty);
  } else {
    while (fifo_available(&sc->rxfifo) &&
           ((ch = ttyread(&sc->tty)) != -1)) {
      fifo_putchar(&sc->rxfifo, ch);
    }
    uart_toggle_intr(sc);
  }

  pthread_mutex_unlock(&sc->mtx);
}
```

fifo_availableを呼び出しFIFOに空きがあるかチェック

FIFOに空きがあればttyreadでttyデバイスから1文字読み出す

fifo_putcharで1文字FIFOにバッファ

RX割り込みをアサートする

▼リスト3　ttyread

```
static int
ttyread(struct ttyfd *tf)
{
  char rb;

  if (tty_char_available(tf)) {
    read(tf->fd, &rb, 1);
    return (rb & 0xff);
  } else {
    return (-1);
  }
}

static bool
tty_char_available(struct ttyfd *tf)
{
  fd_set rfds;
  struct timeval tv;

  FD_ZERO(&rfds);
  FD_SET(tf->fd, &rfds);
  tv.tv_sec = 0;
  tv.tv_usec = 0;
  if (select(tf->fd + 1, &rfds, NULL, NULL, &tv) > 0 ) {
    return (true);
  } else {
    return (false);
  }
}
```

tty_char_availableでttyに読み出し可能な文字があるかチェック

ttyより1文字読み出し

selectで文字が届くまでブロック

▼リスト4　fifo_putchar

```
static int
fifo_putchar(struct fifo *fifo, uint8_t ch)
{

  if (fifo->num < fifo->size) {
    fifo->buf[fifo->windex] = ch;
    fifo->windex = (fifo->windex + 1) % fifo->size;
    fifo->num++;
    return (0);
  } else
    return (-1);
}
```

RX FIFOバッファに1文字書き込み

FIFOのインデックス値更新

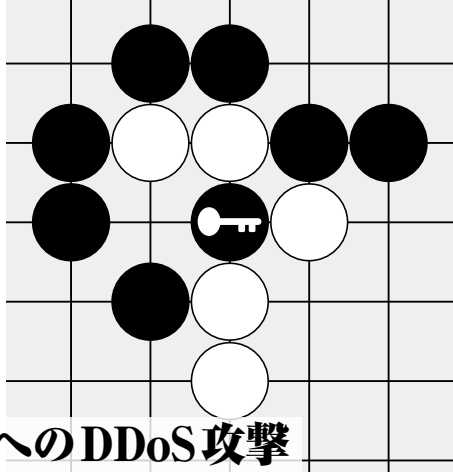
使用済みバイト数を更新

まとめ

今回まで3回に渡ってbhyveの仮想シリアルポートを解説しました。次回は、BIOS・UEFIブートについて解説します。SD

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

【第十四回】現実の脅威となったDNSサーバへのDDoS攻撃

本連載第10回「根深くはびこるDDoS攻撃の脅威」（本誌2014年4月号）の中で取り上げたDNSオープンリゾルバを悪用したDDoS攻撃が、日本国内でも現実の脅威となってしまいました。今回はDNSオープンリゾルバを使った新しいタイプのDDoS攻撃について取り上げます。

DNSサーバを狙ったDDoS攻撃

今年に入って、日本国内のISP（Internet Service Provider）のDNSサーバに対してDDoS攻撃が発生し、ISPユーザがインターネット接続に支障をきたす事例がいくつか発生しています[注1]。今はISPレベルですが[注2]、潜在的にはもっと大きな脅威と言ってかまわない問題です。今後、とくに注意深く経過を見ていく必要があるでしょう。

DDoS攻撃とは何か

すでに本連載でも過去に取り上げましたが[注3]、おさらいとしてDDoSとは何かをもう一度確認しましょう。

DoS（Denial of Service）攻撃とは、攻撃対象のシステムが提供するサービスを不能にさせる攻撃です。今回の話題の範囲では、とくに攻撃対象に対してシステムの容量を越えるデータを与える、あるいは要求することによってシステムを麻痺させるものを指しています。

この条件では、攻撃側は相手のシステムを麻痺させるのに十分な能力を持っている必要があります。そこで、十分な攻撃資源を手に入れるために攻撃元を複数に分散させるのが、DDoS（Distributed DoS）攻撃です。

たとえば、攻撃元が一般家庭のネットワーク回線で、一方、攻撃先はデータセンターのような圧倒的なキャパシティを確保している場合、攻撃をしかけたとしても、効果的な攻撃となるのは難しいでしょう。しかしながら、そのような環境で、かつ少数の拠点からの攻撃であれば十分に耐えられるものであっても、攻撃元が分散し数が増えていけば、いつかは力関係は逆転し、耐えられなくなります。

DNSのしくみ

DNSに対する攻撃について説明する前に、まず、DNS（Domain Name Server）の役目を説明しましょう。DNSは簡単に言うと、ドメイン名とIPアドレスとを関連付ける一種の分散データベースです。

注1）・INTERNET Watch「ケイ・オプティコム、DDoS攻撃によるDNSサーバー障害が復旧したと発表」
http://internet.watch.impress.co.jp/docs/news/20140707_656788.html
・MY J:COM「DNSサーバの障害発生について」
http://information.myjcom.jp/outage/99.html
・YOMIURI ONLINE「初心者ハッカーも可能、ルーター攻撃多発」
http://www.yomiuri.co.jp/it/20140801-OYT1T50196.html
注2）それでも単独ISPで200万世帯を上回る規模で影響が出ています。
注3）DoS/DDoS攻撃に関するより詳しい情報は、本連載第10回「根深くはびこるDDoS攻撃の脅威」（本誌2014年4月号）を参照ください。

DNSにドメイン名やホスト名を問い合わせると、それに対応したIPアドレスを教えてくれます。

DNSの機能は、インターネット(前身であるARPANETも含む)が生まれた当初にはありませんでした。DNSのコンセプトを実装したBIND(Berkeley Internet Name Domain)の誕生は1983年です。DNSのRFCドキュメントであるRFC1034の発行が1987年です。このようにDNSのしくみは1980年代中期に作られたものです。

ちなみに初期のころは、スタンフォード大学の関連組織であるStanford Research Instituteが運用していたサーバ上にHOST.TXTというファイルがあって、そこにアドレスとホスト名の対応が書かれており、そのファイルを各サイトの管理者がftpでダウンロードするといった形で運用していました。

DNSはクライアントからの問い合わせに対し、まず自分が知っていれば応答を返します。自分が知らなければ、上位のDNSに問い合わせます。最終的には、権威ネームサーバ(Authoritative Name Server)と呼ばれるDNSサーバに問い合わせます。

世間では、「Authoritative Name Server」の訳語を「権威ネームサーバ」としていますが、これは「信頼できる(authoritative)ネームサーバ」と理解してください。なお、JPNICはAuthoritative Name Serverを「権威DNSサーバ」と呼んでいるので、それに従い、本稿でも権威DNSサーバと呼びます。

では、ホスト名をDNSに問い合わせてDNSの情報を確認するdigコマンドを使って、nic.ad.jpを確認してみます(**図1**)。

筆者はUbuntuを使っています。**図1**のdigも、その環境で実行した結果です。また、筆者のUbuntu上では、lwresdというDNSのキャッシュサーバを動かして、DNS応答の効率をあげています。digは、まずローカルのlwresdにnic.ad.jpを問い合わせます。もしローカルのキャッシュに情報がなければ、上位のDNSに問い合わせなどをします。それでもなければ、最終的に権威DNSサーバに問い合わせます。

nic.ad.jpは2つの権威DNSサーバを用意して多重化しています。最初に問い合わせをしたDNSサーバが何らかの理由で反応を示さなくても、(タイムアウトして)もう片方のDNSサーバに問い合わせる運用です。

教科書的な説明としては、インターネット上のDNSは、1つのDNSサーバで管理するのではなく、このような分散化したデータベースとして振る舞う特徴を持っています。どこかのDNSサーバが故障しても、影響は極めて局所的な問題にとどまるか、あるいはバックアップのDNSサーバがあるため影響は軽微になります。

韓国の1.25大乱

しかし、現実には運用に依存する部分が非常に大きいと言えます。たとえば、複数のDNSサーバを用意していても、同一のネットワークセグメントに用意してあれば、そのネットワークセグメントへの接続性が確保できなくなった瞬間にDNSが使えなくなります。そのDNSに頼っているユーザは、実質的にインターネットへのアクセスができない状態になります。

◆**図1　digコマンドでDNSの情報を取得する**

```
% dig nic.ad.jp

; <<>> DiG 9.8.1-P1 <<>> nic.ad.jp
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 53418
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 2, ADDITIONAL: 0

;; QUESTION SECTION:
;nic.ad.jp. IN A

;; ANSWER SECTION:
nic.ad.jp. 3600 IN A 192.41.192.129

;; AUTHORITY SECTION:
nic.ad.jp. 86400 IN NS ns3.nic.ad.jp.
nic.ad.jp. 86400 IN NS ns5.nic.ad.jp.
  (... 省略 ...)
```

2003年1月25日に韓国で発生した大規模インターネット障害が、このケースにあたります[注4]。これはSQL Slammerワームが大規模に発生し、そのために韓国全体のインターネットが麻痺したと理解されています。もちろんSQL Slammerワームが発生しなければこの障害は発生しませんでしたが、本質的な問題は韓国のDNSの構成と運用が脆弱であったという部分です。

この背景を理解するには、韓国の特殊事情を理解する必要があります。当時の韓国では、DNSサーバはISPが一括して管理するというのが一般的でした。DNSは分散データベースであり、複数用意することで性能の確保やリスク分散を行えるという利点があります。しかし一方で、DNSサーバの数を絞れば、設備コストや運用コストを下げることができます。

なお、誤解がないように付け加えますが、筆者は、これはトレードオフの関係にあるとは思いません。なぜならば、ネットワークは動かなければ、そもそもの意味をなさないからです。また、インターネットがまだ本格的に普及する前の実験的なネットワークの段階を経験していれば、「ネットワークというものは、そんなに安定して動くものではない」ということを、嫌というほど知っているはずです。それゆえに、そのころを知っている者であれば、インターネットのシステムでは、多くの部分が多重化を前提としているのが当たり前だと思っているはずです[注5]。

一方で、インターネットも安定してしまい、障害が起こることは極めてまれな状況になってくると、そのような多重化を必要とは考えず、むしろ重複投資だと考えるようになるのも、そう不思議な論理ではありません。韓国が極めて速いペースでインターネットの整備が進み、日本に比べていち早くインターネット大国と呼ばれるようになったのはご存じのとおりですが、いい意味でも悪い意味でも、極めて効率の良い設備投資をしたからです。

韓国のISPは少数の大手が寡占しているのが特徴です。国内ユーザの50%はKT(Korea Telecom)1社で抱えているほどです。韓国内の多くの企業や組織は、独自のDNSサーバを運用するようなことはせず、ISPのDNSサーバに依存するというのが一般的でした。

そして、ISPのDNSサーバはネットワーク的に多重化されておらず、SQL Slammerの影響が現れるようなネットワークセグメントに設置されていました。SQL Slammerによりそのネットワークセグメントでサチュレーション(飽和)が発生してしまい、ユーザがDNSサーバにアクセスできなくなった瞬間、韓国全土のインターネットが麻痺した状態になったのです(コラム「多重性/多様性を持たないDNS運用が招いた結果とは」参照)。

DNSオープンリゾルバ

リゾルバ(resolver)とは、ホスト名からIPアドレスを引いたり、IPアドレスからホスト名を引いたりして名前解決をする機能、あるいは、その機能を提供する機材を指します。オープンリゾルバとは、誰でも問い合わせできるリゾルバのことです。具体的には、どのクライアントからの問い合わせにも答えるDNSサーバ(含むDNSキャッシュサーバ)などです。

古き良き時代には、オープンリゾルバの存在に関して、とくに気にはしていなかったと記憶しています。DNSを引くコストのひとつひとつは小さいので、「外部から使いたいクライアント(利用者)がいたら使ってもかまわない」くらいに考えていたと思います。ですから古い機材でDNSキャッシュ機能を提供しているものはデフォルトでオープンリゾルバだったものが多数ありました。

現実にも、DNSオープンリゾルバ問題が発生するまで権威DNSサーバとキャッシュDNSサーバを区別せず1つの外向けDNSサーバで運用しておくと

注4) 『「セキュリティワーキンググループ」中間報告書(案)』2.4.3 韓国の取り組み
http://www.soumu.go.jp/main_sosiki/joho_tsusin/policyreports/chousa/soft_kondan/pdf/030715_2_2c.pdf
注5) そうであってほしいという願望も含めて。

いうのが当たり前のように行われていました[注6]。

現在、GoogleがIPアドレス8.8.8.8、および8.8.4.4でパブリックアクセスのDNSサーバを提供する時代ですので、（GoogleのDNSサーバに比べ）個々の小さなDNSキャッシュが外部のユーザにサービスを提供する利点を見つけることは難しいでしょう。少なくとも筆者は思い浮かびません。

これまでのDNSオープンリゾルバ問題

これまでのDNSオープンリゾルバ問題とは、オープンリゾルバとなっているDNSサーバに対してIPアドレスを偽装した問い合わせパケットを送り、その応答トラフィックを偽装したIPアドレスのもとに送りつけるというものでした。この問題は

注6）@IT「JANOG 31.5 Interim Meetingレポート　オープンリゾルバ問題、立ちふさがるはデフォルト設定？」
http://www.atmarkit.co.jp/ait/articles/1305/09/news013.html

◉多重性／多様性を持たないDNS運用が招いた結果とは

韓国でのSQL Slammerの大規模感染が原因で、韓国全土に及んだインターネットの混乱ですが、韓国では発生月日を取って「1.25大乱」と呼ばれます。

その後にまとめられた韓国官民合同調査団報告では、「国内にルートDNSがないために国内のDNSが過負荷になった」と分析しています。しかし、これは理解が誤っています。正しくは、「本来のインターネットが持つ多重性を無視したDNS運用が招いた結果である」と理解すべきです。

本来のDNSサーバ運用のようにDNSサーバが分散している環境であれば、運悪くいくつかのDNSサーバがダウンし、局所的な問題が発生したとしても、全体でみれば生き残ったDNSサーバが存在しています。パフォーマンスは低くなるとはいえ、国家レベルでインターネットがダウンしてしまうような最悪の状況は回避できていたでしょう。

筆者にとって印象深いのは、かつて韓国のネットワーク関係者と議論したときの記憶です。筆者は韓国の技術者から「日本では何台のDNSサーバが運用されているのか」という質問を受けました。

日本の場合、ある程度大きな規模の企業や大学では、独自にDNSサーバを用意して自サイトのユーザに提供しているのが一般的です。筆者は、「数千から数万の範囲だろうが、日本全体でどれだけのDNSサーバが運用されているかはわからない」と答えました。韓国の技術者は、訝（いぶか）しげな顔をして「DNSサーバの数を尋ねているのだ」と再度質問するのですが、やはり同じように答えるしかなく、結局、話が通じませんでした。

後々に、韓国から詳細な報告書が出てきてはじめて、なぜ話が通じなかったのかが理解できました。当時の韓国内の運用では、DNSサーバはISP単位で設置するものであり、そんな何千台も国内に存在するようなものではなかったのです。1.25大乱は当時の韓国国内の脆弱なネットワーク運用という特殊な事情が招いた結果です。ほかの国でもSQL Slammerが発生したのに、国全体が麻痺するような大規模な問題に発展しなかったのはこのためです。

日本のインターネットの歴史は、国内にISPなどが存在しないときから始まっています。当時はデータを通す専用線だけが存在していて、その上のインターネットで必要なサービス類は、個々のユーザやあるいは研究グループ組織が手弁当で運用するといった形で発展してきました。初期のインターネット（と、今日呼ばれるもの）は、機材やソフトウェアも不安定で、そのためにいろいろな工夫や運用能力を磨いていかざるを得ませんでした。当時を思い出しても、ネットワークの調子が悪く、1日電子メールが届かなくても、「まあよくあること」くらいの感覚でした。

そんな感じで、日本のインターネットはゼロからの手作りであるわけです。その流れがあるため、これまで日本では、インターネットが本来持つ多重化や分散のコンセプトが浸透しており、障害が発生しても局所化して閉じる傾向がありました。

しかし、これは古き良き時代のインターネットの名残であって、今後の耐障害性を保証するものではありません。日本の環境においても、日々発生する新しい問題を解決しながら、より安全でより強固なインターネットのインフラを構築する努力を怠らないようにしないといけないのは、言うまでもありません。

2006年当時から現在まで継続して問題になっています[注7]。

再帰的な問い合わせを行った場合は、応答トラフィック量は問い合わせトラフィック量の40～90倍[注8]の量に増幅されるというものです。

これは現状でも十分に脅威であり、また実際にDDoS攻撃に使われ続けています。2006年から7年もの月日がたった2013年4月にも、JPCERT/CCは「DNSの再帰的な問い合わせを使ったDDoS攻撃に関する注意喚起」を出しています[注9]。

DNSオープンリゾルバによる新しい脅威

最近、オープンリゾルバによる新しい脅威が出てきました。しくみの全般を指してオープンリゾルバ・フォワーダーという言い方がされますが、これから説明する攻撃には、まだ特定の名称はついていないようです。筆者は「ランダムホスト名攻撃」と呼んでいますが、あくまでも仮名称として扱ってください。

この攻撃はDNSオープンリゾルバを使うDDoS攻撃の一種ですが、権威DNSサーバに対して集中して効果的に攻撃するのが特徴です。

ステップごとに説明しましょう。ドメインexample.comに属するホストにアクセスさせないことを目的とした攻撃シナリオを例にします。また、既存のDNSオープンリゾルバを使うDDoS攻撃と同じ攻撃プラットフォームが存在していることを前提とします。

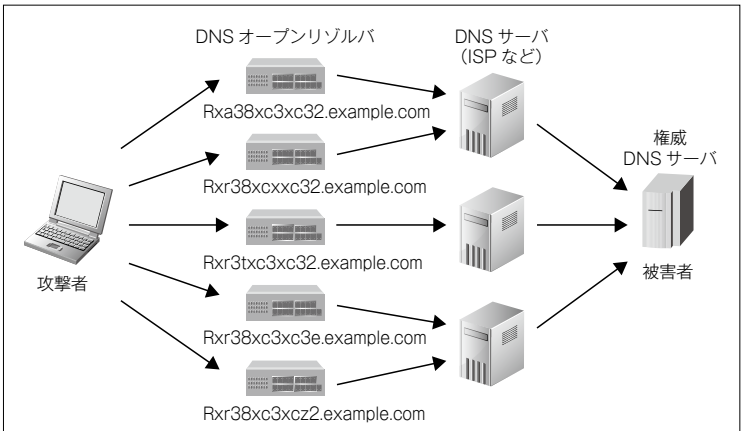
攻撃のシナリオ

- **Step1：DNSオープンリゾルバに対して、example.comドメインに存在しないホスト名を問い合わせる（図2）**
- **Step2：DNSオープンリゾルバは知らないホスト名なので、（ISPのDNSを経由する形で）example.comの権威DNSサーバに問い合わせを行う**
- **Step3：example.comの権威DNSサーバは存在しないホスト名であるとDNSオープンリゾルバに応答する**

詳しく見ていきましょう。たとえば、xedheu12e43x.example.comといった具合に、問い合わせるホスト名はでたらめにします。ホスト名のパターンを英数字12文字の組み合わせで作ると、78京パターンを越えるので事実上無制限です。どんどんホスト名を変化させてDNSオープンリゾルバにホスト名の問い合わせを送ります。

DNSオープンリゾルバは、具体的にはDNSキャッシュサーバ相当の機能を持った家庭／SOHO向けブ

◆図2　DNSオープンリゾルバにホスト名を大量に問い合わせる



この問い合わせの流れは、流れだけみると正当な問い合わせと区別がつかない

注7）・警察庁「DNSの再帰的な問い合わせを悪用したDDoS攻撃手法の検証について」
http://www.cyberpolice.go.jp/detect/pdf/20060711_DNS-DDoS.pdf
・JPCERT/CC「DNSの再帰的な問合せを使ったDDoS攻撃に関する注意喚起」
http://www.jpcert.or.jp/at/2006/at060004.txt
・㈱日本レジストリサービス「DNSの再帰的な問合せを使ったDDoS攻撃の対策について」
http://jprs.jp/tech/notice/2006-03-29-dns-cache-server.html

注8）40倍というのは、注7の警察庁調べ。90倍というのは、注6のJANOG 31.5 Interim Meetingレポートでの発言。

注9）・JPCERT/CC「DNSの再帰的な問い合わせを使ったDDoS攻撃に関する注意喚起」
https://www.jpcert.or.jp/at/2013/at130022.html

ロードバンドルータが中心です。攻撃側は、ホスト名が重複しないようにするので、当然ながらキャッシュ中にはそのホスト名DNS情報は存在せず、権威DNSサーバに対して問い合わせが発生します。

権威DNSサーバは、存在しないホスト名の問い合わせに対して応答を返します。多数のDNSオープンリゾルバに対して大量の問い合わせを行うことによって、権威DNSサーバは応答するために大量の計算資源が必要となり、DDoS攻撃を受けた状態になります。また、中継で使われたISPのDNSに負荷がかかりダウンするようなことが発生すると、本来は攻撃対象ではなかったISPと、そのユーザにも被害が及びます。

このDDoS攻撃の最大の特徴は、どの攻撃の段階でもDNSへの問い合わせに使うパケットのIPアドレスを偽造する必要がないことです。つまり、すべては正常な動作ですので、アウトバンド方向のIPパケットの送信元アドレスをチェックし、偽造したIPアドレスをフィルタリングするBCP38のような対策は現状では役に立ちません。

攻撃対象となっている権威DNSサーバ側からみると、ISPの正式なDNSサーバから大量に問い合わせが来ているように見えます。もし、このトラフィックを遮断すると、正式なホスト名の問い合わせが来ても応答しない、つまり、自らのサイトがクライアントから見えなくなる、ということになります。当然、この対応は取れません。

さらに、この攻撃をマルウェア感染型DDoS攻撃と組み合わせて、マルウェアに感染した大量のPCから攻撃を開始されると、簡単には手に負えないことになるでしょう。しかも、ピンポイントにDNSサーバに攻撃できるので、ネットワークの帯域を飽和させる攻撃よりもはるかに少ない攻撃資源で効果的に影響を与えられるはずです。

警察庁は2014年7月23日づけで、この攻撃が国内で発生していることを警告しています[注10]。また、DNSキャッシュサーバの機能をオープンリゾルバにしたまま出荷している大量のブロードバンドルータによって発生している可能性も示唆しています。

現状での対策

今のところ、被害者側が積極的に取れる防御方法は、権威DNSサーバの処理能力を増やすことぐらいです。1台あたりの権威DNSサーバの能力を上げたうえで、攻撃に耐えられるまでDNSサーバを多重化する。つまり、提供するDNSサーバの数を確保することです。

自分でDNSサーバ群を構築し運用するノウハウがなければ、たとえばAkamai社のFast DNSのサービスなどを使うという選択肢があるでしょう。しかし、これまでプライマリDNSとセカンダリDNSの2台で運用する程度で十分だったものが、大規模なDDoS攻撃に耐え得るほど多重化するようになれば、設備コスト、運用コストも膨らむという問題が発生します。その対策費はバカにならないはずです。

◆ ◆ ◆

筆者が強く感じているのは、2006年から問題視されていたDNSオープンリゾルバが、現在もまだ日本国内に大量に残っているという部分です。デフォルトでDNSオープンリゾルバが設定されて出荷された古いブロードバンドルータも寿命がつき、年々少なくなっていく傾向にはあるのですが、それでもいまだに本質的な脅威として懸念するレベルでネットワーク上に散らばっています。

国内のネットワーク上に残るたくさんのDNSオープンリゾルバをなくしていくことが、遠回りのようで近道ではないかと考えています。

今後、日本のインターネットのセキュリティに関係する組織や関係者、運用している組織や関係者を巻き込んで、より実効性の高い対策を考えいく必要があります。SD

注10）・ITPro「DNSサーバーを狙ったDDoS攻撃、オープンリゾルバーを踏み台に」
http://itpro.nikkeibp.co.jp/atcl/news/14/072500214/
・@police「日本国内のオープン・リゾルバを踏み台としたDDoS攻撃発生に起因すると考えられるパケットの増加について」
www.npa.go.jp/cyberpolice/detect/pdf/20140723.pdf

第51回 Dropbox連携アプリを作るには

モバイルデバイス初のオープンソースプラットフォームとして、エンジニアから高い関心を集めるGoogle Android。いち早くそのノウハウを蓄積したAndroidエンジニアたちが展開するテクニックや情報を参考にして、大きく開かれたAndroidの世界へふみだそう！

重村 浩二　SHIGEMURA Koji
日本Androidの会 中国支部長
Mail k-shigemura@android-group.jp

Androidでもクラウドを

Androidが日本の市場に登場して、5年が経過しようとしています。市民権を得たAndroidは世界中のユーザに利用されています。国内で言うと2年縛りの契約から、そろそろ2〜3回目の機種変にぶつかっているユーザが出てきているころです。

ユーザにとって面倒なのは、これまで使っていた端末の情報(データ)を移行する行為です。必要なアプリをインストールしたタイミングで、前の機種で使っていた情報が移ったほうが嬉しい場面があるでしょう。また、現在使っている端末内の情報を、PCなど別の端末から見たいといった要望もよくあるものです。

Androidの機能単体ではなかなか難しい上記の問題も、クラウドにつなぐことで解消することが可能です。昨今、さまざまなクラウドサービスが出てきています。今回はその中でも、とくに人気のあるDropboxへのアクセス手法を身につけましょう。

Dropboxが提供するSDK

Dropboxでは、Dropbox Developersという開発者向けのサイトを通じて、Dropboxにアクセスするための SDKを提供しています。

- Dropbox Developers
  https://www.dropbox.com/developers

提供されている機能には、簡易アクセスを行うDrop-insや、同期を行うSync APIなどが提供されています。Dropbox SDKを利用するには、App key(アプリを認証するのに必要な鍵)の取得を事前に実施する必要があるので、まずはそこから解説しましょう。

DropboxのApp keyを取得しよう

DropboxのApp keyは、App Consoleのサイトを通じて取得します。

- App Console
  https://www.dropbox.com/developers/apps

App Consoleのサイトの右上にある「Create app」から新しいアプリケーションの登録を行うことができます。アプリの登録時には、「Drop-insアプリ」なのか、「Dropbox APIアプリ」なのかを聞かれます[注1]。この後解説を行うサンプルアプリ「DropboxPreview」では、Drop-insのChooserという機能を実装しているので、「Drop-insアプリ」で登録します。もう1つのサンプル「TextSaver」はSync APIを使うので、

注1) 執筆時点(2014年8月10日)では、Drop-ins用のAndroid向けのSDKは「Chooser」しか提供されていません。「Saver」については"Coming soon"と記載されていることから、今後提供を予定しているのでしょう。

「Dropbox APIアプリ」での登録が必要です。このように作成するアプリに応じて使い分けます。登録を行うと取得できるApp key、App secret（ChooserではApp keyのみ）が、Dropboxにアクセスするために必要となります。

DropboxPreviewアプリの開発

それでは、Dropbox上に保存したファイルを取得するというシンプルな機能を提供する、Drop-insのChooserから見ていきましょう。

Dropbox上のファイル操作を行うサンプルアプリとして、「DropboxPreview」を用意しました。このアプリは画面中央のボタンをクリックすると現れるファイル選択画面から、選択したDropbox上のファイルをプレビューで表示する機能を有しています。

開発前の事前準備

Chooserを利用するには、次のサイトからSDKをダウンロードする必要があります。

- Using the Chooser
  https://www.dropbox.com/developers/dropins/chooser/android

zipファイルの中にはDropboxChooserSDKのライブラリプロジェクトとサンプルアプリが、Eclipseのプロジェクトとして用意されています（Android Studioで利用する場合も、プロジェクトをインポートすれば使えます）。DropboxChooserSDKをEclipseにインポートし、新規に作ったプロジェクトのプロパティから［Android］－［Library］に、DropboxChooserSDKを追加することでアプリの開発を行うことが可能です。

ファイル選択画面の呼び出しと結果の取得

Dropbox上のファイル選択には、**リスト1**にあるように、DbxChooserというクラスのインスタンスを生成し、launchメソッドでファイル選択画面を呼び出す必要があります。

ファイル選択画面の呼び出しには、DbxChooserのインスタンス生成時に引数として、App Consoleで取得したApp keyを指定する必要があります。App keyは**リスト2**にあるように、クラス内のプライベートな定数として定義すると使いやすいでしょう。launchメソッドの第2引数で指定しているリクエストコードは、ファイル選択画面の結果を受け取るためのリクエストコードとなります。

▼リスト1　ファイル選択画面の呼び出し

```
public void onButtonClick(View v) {
    new DbxChooser(APP_KEY)
        // プレビュー表示するリクエストタイプの設定
        .forResultType(ResultType.PREVIEW_LINK)
        // Chooserの呼び出し
        .launch(this, DBX_REQUEST_CODE);
}
```

Column

アプリ開発時は実機の用意を

Dropboxと連携したアプリを作る際の注意点として、アプリの実装はエミュレータではできない点に注意しましょう。DropboxのAndroidアプリがインストールされている端末である必要があるためです（インストールされていない場合は、Google Playストアからインストールするように促されます）。通常のやり方ではエミュレータにDropboxをインストールできないので、実機を用意して開発するようにしましょう。

▼リスト2　App keyの定義とファイル選択画面呼び出し時のリクエストコード定義

```
// APIキーの設定                                    ↓ App Consoleで取得したApp keyを指定してください
private static final String APP_KEY = "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX";
// Dropboxのファイル選択画面呼び出しを行うときのリクエストコード
private static final int DBX_REQUEST_CODE = 100;
```

▼リスト3　ファイル選択画面の処理結果

```
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    // リクエストコードのチェック
    if (requestCode == DBX_REQUEST_CODE) {
        // resultCodeがRESULT_OKの(ファイルが選択された)場合に処理実施
        if (resultCode == Activity.RESULT_OK) {
            // プレビューのURIを取得
            Uri uri = new DbxChooser.Result(data).getLink();

            // 指定のURIを呼び出し
            Intent startIntent = new Intent(Intent.ACTION_VIEW, uri);
            startActivity(startIntent);
        }
    } else {
        super.onActivityResult(requestCode, resultCode, data);
    }
}
```

Dropbox上のファイルをプレビュー表示したい場合には、forResultTypeメソッドにPREVIEW_LINKを指定します。PREVIEW_LINKの指定により、Chooserからはプレビュー表示のためのURIが返されます。これ以外にも、DIRECT_LINK(ファイルに直接アクセスするURI)とFILE_CONTENT(ローカル上でアクセスするURI)が用意されています。

ファイル選択画面の処理結果は、onActivityResultメソッドに返却されます(**リスト3**)。

ファイル選択画面のリクエストに対する結果を処理するために、**リスト2**で定義した定数でチェックし、resultCodeでファイルが選択された(RESULT_OK)かどうかのチェックを事前に実施しています。複数のActivityを呼び出し、結果を受け取るアプリに改修していくこともあるでしょうから、リクエストコードは必ずチェックするようにしておきましょう。

プレビューのURIを取得するには、DbxChooser#ResultメソッドにonActivityResultで渡されてきたIntent(第3引数：data)を渡し、getLinkメソッドを呼び出すことで実装できます。サンプルでは、取得したURIを元にstartActivityメソッドを呼び出して、プレビュー表示を行っています。

Chooser利用時の注意点

簡単に利用できるChooserですが、Android Support Libraryのバージョン違いに注意しましょう。DropboxChooserSDKと新しいプロジェクトのAndroid Support Library間でバージョン違い(差分)となると、コンフリクトが発生してしまいます。この現象が発生した場合には、DropboxChooserSDKのlibsディレクトリに、新しいプロジェクトからandroid-support-v4.jarを上書きすることで対処してください。

TextSaverアプリの開発

Chooserを用いてシンプルなDropboxからのファイル取得方法を学んだ次は、Dropbox上にアプリのデータを保存する方法を学びましょう。サンプルとして、「TextSaver」というアプリを用意しました。「保存！」ボタンをクリックすることで、テキスト入力欄に入力したテキストをDropbox上に保存するアプリとなります。

開発前の事前準備

TextSaverの開発でも、Chooserと同様に開発前の事前準備が必要です。Sync APIを利用するために、次のサイトからSDKをダウンロードしましょう。

▼リスト4　パーミッションの追加

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
```

▼リスト5　Dropbox認証用アクティビティの定義

```
<activity android:name="com.dropbox.sync.android.DbxAuthActivity" />
<activity
    android:name="com.dropbox.client2.android.AuthActivity"
    android:launchMode="singleTask" >
    <intent-filter>                    ↓ App Consoleで取得したApp keyを指定
        <data android:scheme="db-XXXXXXXXXXXXXXXXXXXXXXXXXXX" />
        <action android:name="android.intent.action.VIEW" />
        <category android:name="android.intent.category.BROWSABLE" />
        <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
</activity>
<service
    android:name="com.dropbox.sync.android.DbxSyncService"
    android:enabled="true"
    android:exported="false"
    android:label="Dropbox Sync" />
```

▼リスト6　Dropboxへの接続処理

```
// Dropboxに接続
mAccountMgr = DbxAccountManager.getInstance(getApplicationContext(),
                                            APP_KEY, APP_SECRET);
// アカウントにアプリが接続しているかチェック
if (!mAccountMgr.hasLinkedAccount()) {
    // アプリの接続開始
    mAccountMgr.startLink((Activity)this, REQUEST_DROPBOX);
}
```

- Sync API SDKs
  https://www.dropbox.com/developers/sync/sdks/android

zipファイルの中にはライブラリとドキュメントなどが格納されています。ChooserのSDKはライブラリプロジェクトとして提供されていますが、Sync APIを利用する場合はSDKに含まれるlibsディレクトリの中身を、新規作成したプロジェクトのlibsディレクトリ内にコピーする必要があります。

AndroidManifestの追記

コピーが完了したら、続いてAndroidManifest.xmlへの追記を行いましょう。Dropbox APIでは「INTERNET」「ACCESS_NETWORK_STATE」のパーミッション利用を許可する必要があります(**リスト4**)。

あわせて、Dropboxへの認証を得るためのアクティビティを呼び出す定義を<application>タグ内に追加する必要があります(**リスト5**)。このとき、AuthActivityのIntentフィルタに対して設定するandroid:schemeには、"db-" + "*App Consoleで取得したApp key*"を指定するようにしてください。

Dropboxへの接続

AndroidManifest.xmlへの定義が終わったら、いよいよDropboxに接続しましょう(誌面の都合上、画面の実装などについては割愛します)。Dropboxへの接続は、onCreateメソッド内に**リスト6**にあるような実装を追加します。

DbxAccountManager#getInstanceメソッドでインスタンスを取得します。getInstanceメソッ

▼リスト7　ファイルの書き込み処理

```
public void onSaveEvent(View v) {
    DbxPath filePath = new DbxPath(DbxPath.ROOT, DROPBOX_FILENAME);

    try {
        // Dropboxの同期したファイルシステムを生成
        DbxFileSystem dbxFs = DbxFileSystem.forAccount(mAccountMgr.getLinkedAccount());

        // DROPBOX_FILENAMEのファイル存在チェック
        DbxFile dbxFile = null;
        EditText editSaveText = (EditText) findViewById(R.id.editSaveText);
        if (!dbxFs.exists(filePath)) {
            // 存在しない場合、ファイルを生成
            dbxFile = dbxFs.create(filePath);
        } else {
            // 存在した場合、ファイルを開く
            dbxFile = dbxFs.open(filePath);
        }
        try {
            // テキストの保存
            dbxFile.writeString(editSaveText.getText().toString());
            Toast.makeText(this, "テキストを保存しました", Toast.LENGTH_SHORT).show();
        } finally {
            dbxFile.close();
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

ドの第2、第3引数に渡す引数が、App consoleで取得したApp keyとApp secretとなります。その後、hasLinkedAccountメソッドですでにDropboxに接続しているかどうかをチェックし、接続されていなければDbxAccountManager#startLinkメソッドで接続を行います。

接続の結果はChooserの流れと同様に、onActivityResultメソッドに返ってきますので、そちらで結果を見て、必要な処理を行ってください(サンプルではrequestCodeをチェックし、Dropbox接続の成否をトーストで表示しています)。

Dropbox上のファイル書き込み処理

Dropboxに接続できたので、ファイルの保存を見ていきましょう。Dropboxへのファイル保存は**リスト7**にあるように、DbxFileSystemのインスタンスをDbxFileSystem#forAccountメソッドで取得することから始まります。

取得できたら、Dropbox上のファイルへのパスを示すDbxPathのインスタンスを元にファイルが存在するかしないかをexistsメソッドでチェックし、存在しなければcreateメソッドでファイル生成を、存在する場合はopenメソッドでファイルを開く処理を行います。

ファイルへの書き込み処理はwriteStringメソッドを使い、書き込み終了後はfinally句の中でcloseメソッドを呼び出して書き込み処理を終了しています。書き込んだデータは、"ROOT:/アプリ/プロジェクト名/ファイル名"に保存されます。

Dropbox上のファイル読み込み処理

Dropbox上のファイル読み込みも、ほとんど書き込み処理と同じ流れになります(**リスト8**)。writeStringメソッドの代わりに、readStringメソッドでテキストを取得します。

読み込み時の処理で、DbxFileSystemのインスタンス取得直後の処理に注目してください。awaitFirstSyncメソッドは、アプリとDropboxが初回接続時の同期処理を待つためのメソッド

▼リスト8　ファイルの読み込み処理

```
public void readSaveText() {
    DbxPath filePath = new DbxPath(DbxPath.ROOT, DROPBOX_FILENAME);

    try {
        // Dropboxの同期したファイルシステムを生成
        DbxFileSystem dbxFs = DbxFileSystem.forAccount(mAccountMgr.getLinkedAccount());

        // 最初の同期を待つ
        dbxFs.awaitFirstSync();

        // DROPBOX_FILENAMEのファイル存在チェック
        DbxFile dbxFile = null;
        EditText editSaveText = (EditText) findViewById(R.id.editSaveText);
        if (dbxFs.exists(filePath)) {
            // ファイルが存在した場合、テキストを読み込む
            dbxFile = dbxFs.open(filePath);
            try {
                editSaveText.setText(dbxFile.readString());
            } finally {
                dbxFile.close();
            }
        }
    } catch (DbxException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

です。どのようなケースで必要になるかというと、アプリを入れてDropbox上にファイルが生成された後、アプリの削除→再インストールするというようなケースで、アプリとDropboxの初回接続直後にこのメソッドを呼び出していないと、すでに存在するファイルからのテキスト読み込みが画面に反映されないといった動きになってしまいます。ぜひ用意したサンプル[注2]で、awaitFirstSyncメソッドの呼び出しをコメントアウトし、動きを確認してみてください。

注2）今回用意したサンプルプロジェクトは、本誌Webサイトのサポートページで公開中です。http://gihyo.jp/magazine/SD/archive/2014/201410/support

まとめ

早足でしたが、Dropboxからファイルを取得する方法と、Dropboxにアプリ上のデータを保存し、読み込む方法を見てきました。今回は取り上げませんでしたが、Dropbox APIではSync API以外にも、Datastore APIなどが提供されています。

アプリの情報をクラウドに出すことはセキュリティを考えると一長一短の面はあると思いますが、より大勢のユーザがより良い利便性を享受できるようにするためにも、選択肢の1つとして活用を検討してみてください。SD

重村 浩二（しげむら こうじ）　日本Androidの会 運営委員 中国支部長

日本Androidの会にて運営委員、中国支部長として毎月山口県・広島県を中心に自作アプリの発表やハッカソン、ハンズオンなどを中心とした勉強会を開催。主な著書に『Android SDKポケットリファレンス』（技術評論社刊）など。
URL http://buildbox.net/ Twitter @shige0501

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6回 Red Hat Satellite 6で多数のサーバを一元管理

Red Hat Satelliteは十数台以上から数万台規模のシステムを一元管理するためのソリューションです。ソフトウェアリポジトリだけではなくさまざまな機能を提供します。今回はSatelliteの基本を紹介します。

レッドハット(株)グローバルサービス本部プラットフォームソリューション統括部
ソリューションアーキテクト部長　藤田 稜 (ふじた りょう)

Satelliteの歴史

Red Hat Satellite 6のβ版が2014年7月1日にリリースされました。GA[注1]まではまだ少し時間がかかりますが、今回はこのβ版をベースに技術的な側面からSatelliteを紹介します。なお、旧来Red Hat Network Satellite Serverが製品呼称でしたが、現在はRed Hat Satelliteに名称が変更されているため、以降、Satelliteと記述します。

Red Hatは2000年にRed Hat Network(以降、RHN)の提供を開始しました。Red Hat Linux 6(以降、RHL。RHELではありません!)には企業向けのサポートを提供するRHL 6.2Eというバージョンがあり、このころにはインターネットを経由したパッケージの配布やバージョン管理システムの必要性が強く認識され始めていました。一方でインターネット接続回線の帯域はまだまだ貧弱[注2]だったため、翌2001年にはRPMパッケージをキャッシュして配布するためのRHN Proxy Serverが提供されました。さらに一部顧客からの強い要望[注3]により、RHNをスタンドアローンとして実現するSatelliteが翌2002年に提供開始されました。ここに至り現在のSatelliteの原型がほぼ整ったと言えます。

2004年に提供開始されたSatellite 3.2では、プロビジョニングモジュールと呼ばれる機能が追加されました。ソフトウェアリポジトリだけでなく、個々のシステムにRHELをインストールし設定を配布できるようになりました。

2007年の2月に提供が開始されたSatellite 4.2は管理対象システムとしてRHEL 5をサポートし、以降、2010年10月リリースのSatellite 5.4がRHEL 6を、2013年10月リリースのSatellite 5.6がRHEL 7をサポートしています。

企業向けのサポートバージョンであるRHELと同時期に提供が開始され歴史が長いSatelliteですが、日本国内ではバージョン4.xになるまで一般に販売していなかったため、あまり知名度が高いとは言えません。しかし、その出自からもわかるように、米国を中心に多数のRHL/RHELサーバを管理する顧客のシステムでは非常に採用率が高く、管理対象のサーバが数万台を超える事例や、ニューヨーク証券取引所の

注1) General Availability、一般向けに提供されるバージョン。正式版とも言う。
注2) 国内では2001年頃からADSL(Asymmetric Digital Subscriber Line:非対称デジタル加入者線)が普及し始め、2001年は「ブロードバンド元年」と呼ばれる。当初1.5Mbps程度の帯域だったが、それまでのINS64/128の64Kbps/128Kbpsと比較するとかなり高速だった。
注3) 要するに「RHNをうちのデータセンタに置きたいから、持ってきて」だった(らしい)。

ようなミッションクリティカルなシステムでも採用されています。

完全に書き直されたSatellite 6

歴史が長い一方で、Satelliteも当初の設計では現在のコンピューティングに対応しきれない、あるいは実装を整理する必要が出てきたのも事実です。2008年の6月にSpacewalkというプロジェクトでオープンソース化[注4]され、バックエンドのデータベースをエンベディッド用のOracle DBからPostgreSQLにスイッチしたり、SELinuxに対応したりといった努力は続けられてきましたが、仮想化やクラウドコンピューティング全盛の時代にはやはり取り残されている感が否めませんでした。

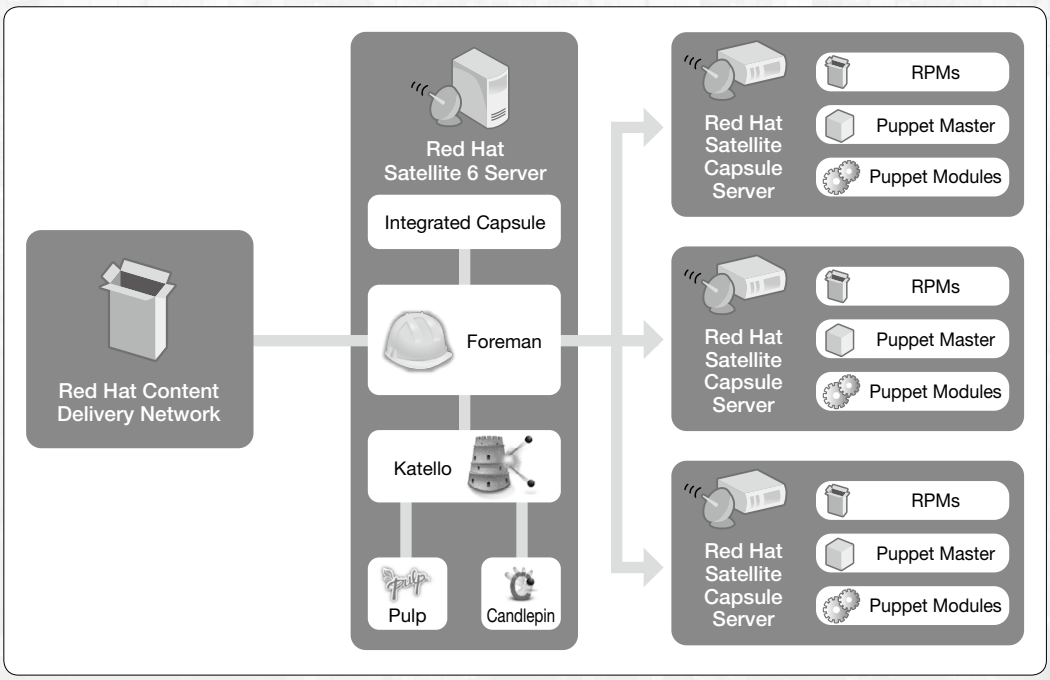
そこでSatellite 6では全面的に再設計・再実装が行われ、最新の技術トレンドを採用しつつ、古いバージョンのRHELも管理対象とすることが可能な製品として生まれ変わりました。

Satellite 6の実装

従来のSatelliteはクローズドソースであったため、内包する個別の機能にモジュールという名称が付いていたものの、Satellite全体が1つのプロジェクトとしてオープンソース化されました。Satellite 6は当初からオープンソースのプロジェクトを組み合わせることで実装されており、大まかには**図1**のような構成になっています。

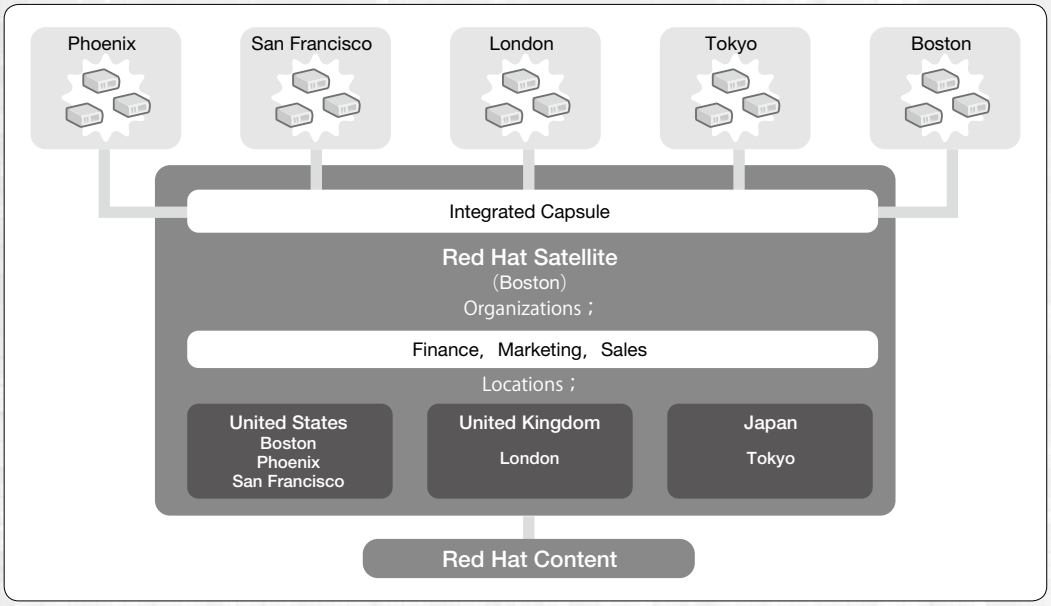
Satellite 6の核となっているのはForeman[注5]です。Foremanは管理対象が物理・仮想マシンのいずれかを問わず、プロビジョニング、設定、モニタリングを可能とします。ForemanはPuppet[注6]やChef[注7]を利用できますが、

▼図1 Satellite 6の構成要素



注4）http://spacewalk.redhat.com/
注5）http://theforeman.org/
注6）http://puppetlabs.com/
注7）http://www.getchef.com/chef/

▼図2　Satelliteの基本的な配置



Satellite 6ではPuppetを利用しています。また、ForemanはRed Hatが提供するOpenStackのディストリビューションであるRed Hat Enterprise Linux OpenStack Platformのインストーラとしても利用されています。

Satellite 6のもう1つの核はKatello[注8]です。KatelloはRed Hatのサブスクリプションとリポジトリの管理機能を提供するアプリケーションで、RHNから配布されるコンテンツ[注9]をRed Hat CDN[注10]から取得します。Katelloはユーザが定義するさまざまな基準でシステムをグルーピングし、アプリケーションのライフサイクル管理を容易に行える機能も提供します。

Pulp[注11]とCandlepin[注12]はKatelloの機能を実装しているいわば「本体」で、Pulpがリポジトリとコンテンツの管理を、Candlepinがサブスクリプションの管理を行います。

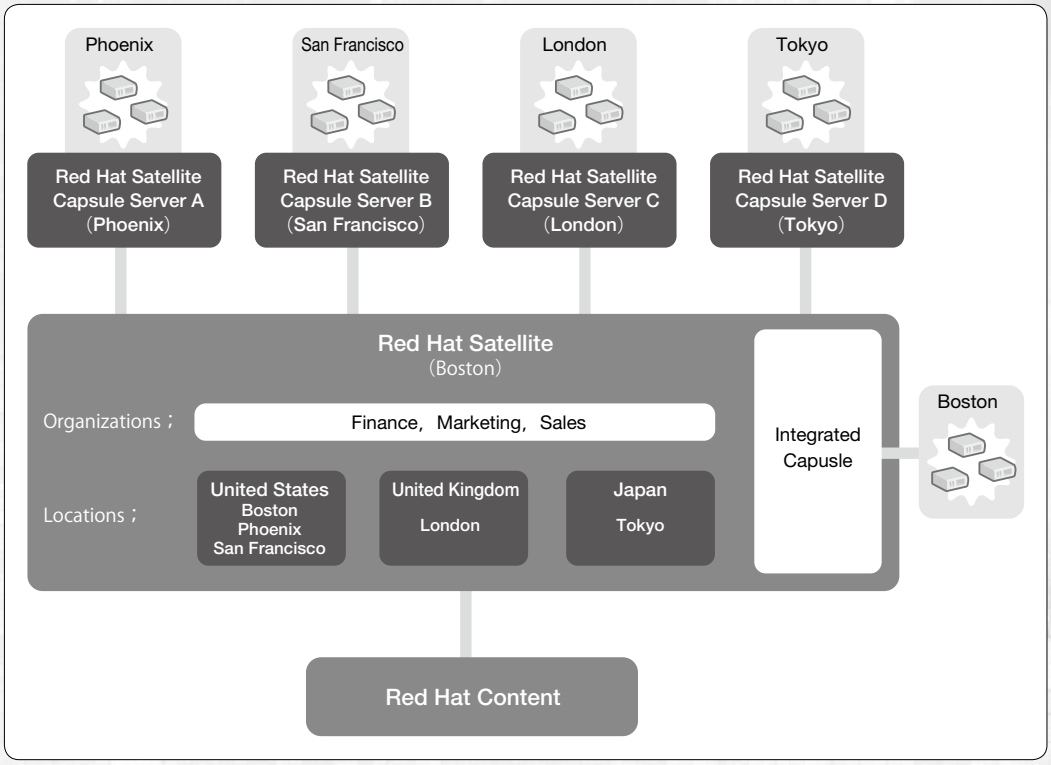
Satelliteを構成するもう1つの要素はRed Hat Satellite Capsule Serverです。Satellite本体にも統合されており、Satelliteのリポジトリ、DNS、DHCP及びPuppet Masterの設定のプロキシとして動作します。Capsuleは旧来のRHN Proxyの機能を拡張したものとも言え、地理的に離れた拠点に設置することで、スケーラビリティやコンテンツ配布のパフォーマンスを向上できます。

SatelliteとCapsuleサーバの配置

SatelliteとCapsuleサーバの配置例を2つほど示します。まず、**図2**は最も基本となるSatellite 1台での構成を表したものです。

注8） http://www.katello.org/
注9） RPMパッケージや、エラータを含むメタ情報など。
注10） Content Delivery Network。Red Hat CDNはAkamaiを利用しているため、利用しているインターネットプロバイダにAkamaiのミラーサーバが設定されていれば非常に高速にコンテンツを入手できる。
注11） http://www.pulpproject.org/
注12） http://www.candlepinproject.org/

▼図3　Satellite 1台＋Capsuleサーバ4台による配置



各拠点が高速なネットワークで接続されており、管理対象のシステムの台数が少ない場合、Satelliteが1台でも十分な性能を発揮できます。また地理的な属性によるグルーピングと、企業内の組織によるグルーピングを設定することで、さまざまな基準でシステムを管理できます。

次に示す**図3**ではSatellite 1台に加え、4拠点に1台ずつ計4台のCapsuleサーバを配置し、スケーラビリティとコンテンツ配信の速度の向上を図る配置になっています。

図2の構成例とグルーピングの方法は変わりませんが、各拠点にCapsuleサーバが存在するため、コンテンツの配信速度はLANの速度になります。パッケージの更新に加え、頻繁に仮想マシン上にRHELをインストールする場合、配信するデータ容量が大きくなるため、この構成が推奨されます。

Satellite 6のインストール

Satellite 6のβ版はパブリックベータではないため、入手するにはSatelliteのサブスクリプション契約が必要です。もし試用する場合には評価版[注13]を**図4**のリンクから申請してください。既

▼図4　評価版の申請用のリンク（"Request Access"）

注13）https://access.redhat.com/products/red-hat-satellite/beta

存のあるいは評価用に新規に作成したRed Hatアカウントに Satelliteのサブスクリプションがエンタイトルされ、インストーラのDVD ISOイメージを入手する、あるいはsubscription-managerで有効化することでSatelliteのインストールが可能になります。

執筆時点における Satellite 6のベータ版はバージョン6.0.3であり、RHEL 6.5(for x86_64)をベースとしています。Satellite 6をインストールするには、まずRHEL 6.5を「最低限[注14]」のパッケージグループでインストールします。

インストール終了後、Satellite 6のインストールの準備をします。最初にターミナルを起動してsubscription-managerコマンドにregisterオプションを付与して実行します。

```
# subscription-manager register
ユーザ名: redhat_account
パスワード:
システムは ID で登録されています: df71e1fb-c17c-4c➥
77-9b22-xxxxxxxxxxxx
```

次に、Satelliteが含まれるプール[注15]を見つけるため、subscription-managerコマンドにlistオプションを付与して実行します。

```
# subscription-manager list --available --all
+-------------------------------------------+
    利用可能なサブスクリプション
+-------------------------------------------+
サブスクリプション名:     Red Hat Employee ➥
Subscription
提供:                     Red Hat Enterprise Linux➥
Server HTB
<< snip >>
                          Red Hat Satellite 6 Beta➥
<< snip >>
プール ID:                8a85f98442cef1c70142fc22x➥
xxxxxxx
```

Red Hat Satellite 6 Betaを含むプールIDがわかったので、このプールIDを用いてシステムにサブスクリプションを割り当てます。もし同じプールIDに“Red Hat Software Collections”が含まれていなければ、同様の手順でプールIDを検索し、サブスクリプションを割り当てます。

```
# subscription-manager attach --pool=8a85f98442➥
cef1c70142fc22xxxxxxxx
サブスクリプションが正しく割り当てられました:Red Hat➥
Satellite Employee Subscription
```

リポジトリをすべて無効に設定したあと、Satellite 6で必要となるリポジトリだけを有効にします。

```
# subscription-manager repos --disable "*"
# subscription-manager repos --enable rhel-6-➥
server-rpms --enable rhel-server-rhscl-6-rpms➥
--enable rhel-server-6-satellite-6-beta-rpms
```

これでSatellite 6をインストールするための準備が整いました。yumコマンドでKatelloをインストール[注16]しましょう。

```
# yum -y install katello
```

Katelloのインストール終了後、katello-installerを実行します(**図5**)。

これでSatellite 6のインストールは完了です。

執筆時点ではkatello-installerはファイアウォールの設定はしないため、iptablesコマンドで設定されているルールを確認したあと、適切な位置にルールを追加します(**図6**)。

ではFirefoxでアクセスしてみましょう。インストール完了後に表示されたURLにアクセスすると**図7**の画面が表示され、ユーザ名：admin、パスワード：changemeでログインでき

注14) 日本語のインストーラでは「最低限」、英語では“Minimal”。この指定をすると@coreと@server-policyというパッケージグループだけがインストールされる。“@”(アットマーク)はRPMパッケージを複数まとめたグループの表記の方法で、Linuxの自動インストールのしくみであるkickstartや、インストール後に/root/anaconda-ks.cfgというファイルの“%packages”セクションの指定でも用いられる。

注15) subscription-managerによる証明書ベースの管理下では、複数のRPMパッケージから構成される「チャネル」、複数のチャネルから構成される「プール」というようにヒエラルキーが構成され、複数のプールが「サブスクリプション」にひもづけられる。

注16) 相当数のRPMパッケージをダウンロードするため、インストールに長時間かかることがあります。

▼図5　katello-installerの実行

```
# katello-installer
Installing             Done                                               [100%] [.......................
.........]
  Success!
  * Katello is running at https://sat.rio.st
      Default credentials are 'admin:changeme'
  * Capsule is running at https://sat.rio.st:9090
  * To install additional capsule on separate machine continue by running:"

      capsule-certs-generate --capsule-fqdn "$CAPSULE" --certs-tar "~/$CAPSULE-certs.tar"

  The full log is at /var/log/katello-installer/katello-installer.log
```

▼図6　iptablesで確認後、ルールを追加

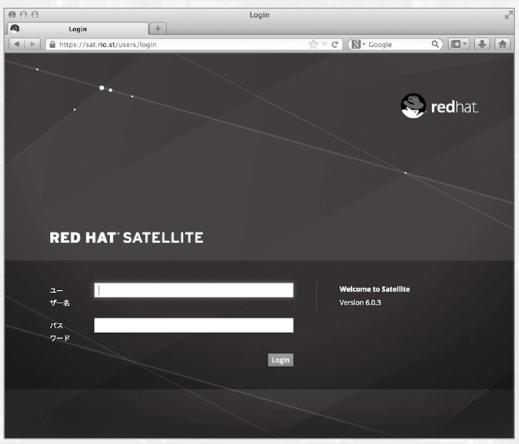
```
# iptables -L INPUT
target     prot opt source      destination
ACCEPT     all  --  anywhere    anywhere    state RELATED,ESTABLISHED
ACCEPT     icmp --  anywhere    anywhere
ACCEPT     all  --  anywhere    anywhere
ACCEPT     tcp  --  anywhere    anywhere    state NEW tcp dpt:ssh
REJECT     all  --  anywhere    anywhere    reject-with icmp-host-prohibited

# iptables -I INPUT 5 -p tcp -m state --state NEW -m tcp --dport 443 -j ACCEPT
# iptables -I INPUT 6 -p tcp -m state --state NEW -m tcp --dport 9090 -j ACCEPT
# iptables -L INPUT
Chain INPUT (policy ACCEPT)
target     prot opt source      destination
ACCEPT     all  --  anywhere    anywhere    state RELATED,ESTABLISHED
ACCEPT     icmp --  anywhere    anywhere
ACCEPT     all  --  anywhere    anywhere
ACCEPT     tcp  --  anywhere    anywhere    state NEW tcp dpt:ssh
ACCEPT     tcp  --  anywhere    anywhere    state NEW tcp dpt:https
ACCEPT     tcp  --  anywhere    anywhere    state NEW tcp dpt:websm
REJECT     all  --  anywhere    anywhere    reject-with icmp-host-prohibited

# service iptables save
```

るはずです。

次回は

今回はSatelliteのインストールまでを紹介しましたが、まだコンテンツもシステムも登録されていないため、このままではとくに何もできません。次回はSatelliteへのコンテンツの導入とシステムの登録について紹介します。SD

▼図7　Satelliteのログイン画面



Be familiar with FreeBSD.

チャーリー・ルートからの手紙

第12回 ◇ コンテナ型仮想化 jail(8) ～リソースを隔離して使うしくみ～

◉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。

コンテナ型仮想化 Jail

FreeBSDにはコンテナ型（あるいはコンパートメント型、セパレート型）といったように呼ばれることがある仮想化機能が用意されています。オペレーティングシステム（以下、OS）が提供しているさまざまなリソースを「区切る」ことで、あたかも複数のOSが動作しているかのように「見せかける」機能です。

最近はDockerと呼ばれる管理ソフトウェアに人気がありますが、Dockerが利用することになる根底の技術がJailです。Linuxでのcgroupsに相当します。こうした技術はホスティングサーバとして使われることが多かったFreeBSDにおいては早い時期に登場しました。KVMやXenといったタイプの仮想化技術と比較して、動作がきわめて軽量で高速という特徴があります。

Jailの使いどころ

Jailが使われるのはおもに2つのシーンです。1つはアプリケーションをホストのOSとは隔離した空間で動作させることでセキュリティを確保したい場合、もう1つはユーザに対してroot権限を与える必要がある場合です。

UNIX系のOSには基本的に“root”と“それ以外のユーザ”という区別しかありません。root以外のユーザにrootが持っている機能を使わせるようにするのはしくみ上困難なところがあります。Jailはこうした問題に対する1つの答えとなるもので、空間を区切ることでその閉じ込められた空間の中であれば限定的なrootの機能を使っても問題がないようにしています。

Jailの考え方のベースとなったchroot(8)

Jailを使ってみる前に、Jailのモデルとなったような機能である**chroot**(8)を見てみましょう。**図1**のように、ちょっとしたコマンドとそれを実行するのに必要になるライブラリ、デバイスファイルをマ

▼**図1 ミニマムなシェル環境を用意**

```
% pwd
/Users/daichi/j
% tree $(pwd)
/Users/daichi/j
├── bin
│   ├── ps
│   ├── sh
│   └── tree
├── dev
│   └── null
├── lib
│   ├── libc.so.7
│   ├── libedit.so.7
│   ├── libkvm.so.6
│   ├── libm.so.5
│   └── libncurses.so.8
└── libexec
    └── ld-elf.so.1

4 directories, 10 files
%
```

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。

ウントした領域を用意したとします。

詳しい説明は省きますが、/dev/nullだけが存在するようなデバイスファイルを作成するには**図2**のようにコマンドを実行します。この作業はrootユーザで実施してください。

環境が用意できたら、**図3**のようにchroot(8)コマンドをrootユーザで実行します。プロンプトが同じなのでわかりにくいですが、4行目のプロンプトはすでにchroot(8)環境下に入っています。

この状態で**pwd**(1)コマンドを実行してカレントディレクトリのパスを表示させると「/」と返ってきます(**図4**)。もともとのパスは「/Users/daichi/j」でしたが、chroot(8)したことで「/Users/daichi/j」が「/」に置き換わりました。

この機能はもともとは、FTPサーバを立ち上げたときにログインしたユーザが見える領域を、ユーザのホームディレクトリ以下に限定したいという目的で開発されました。当時はセキュリティについては考慮していませんし、実装には結構抜け穴もありました。

chroot(8)の機能はファイルシステムという空間を「隔離」または「分離」する機能といえます。この機能をファイルシステムのみならず、さまざまなリソースや空間についても適用すればOS全体を「仮想化」したように振る舞わせることができる。Jailはこうした発想に基づいて開発されています。

試しに、chroot(8)環境下において**ps**(1)コマンドを実行してみましょう。**図5**のように、ホストで実行するのと同じ結果が見えます。chroot(8)が隔離しているのはファイルシステム空間だけで、それ以外はホストと共通です。

▼図2　null(4)しかないデバイスファイルの作り方

```
# mount -t devfs devfs /Users/daichi/j/dev
# devfs -m /Users/daichi/j/dev rule apply hide
# devfs -m /Users/daichi/j/dev rule apply path null unhide
```

▼図3　chroot(8)で/Users/daichi/jをルートディレクトリへ変更

```
# chroot /Users/daichi/j /bin/sh
Cannot read termcap database;
using dumb terminal settings.
#
```

▼図4　用意したミニマム環境にしかアクセスできなくなっている

```
# pwd
/
# tree $(pwd)
/
|-- bin
|   |-- sh
|   `-- tree
|-- lib
|   |-- libc.so.7
|   |-- libedit.so.7
|   `-- libncurses.so.8
`-- libexec
    `-- ld-elf.so.1

3 directories, 6 files
#
```

▼図5　ps(1)コマンドでシステム全体のプロセスが見える

```
# ps
  PID TT  STAT    TIME COMMAND
 1041 v0  Is+  0:00.00 /usr/libexec/getty Pc ttyv0
 1042 v1  Is+  0:00.00 /usr/libexec/getty Pc ttyv1
 1043 v2  Is+  0:00.00 /usr/libexec/getty Pc ttyv2
 1044 v3  Is+  0:00.00 /usr/libexec/getty Pc ttyv3
 1045 v4  Is+  0:00.00 /usr/libexec/getty Pc ttyv4
 1046 v5  Is+  0:00.00 /usr/libexec/getty Pc ttyv5
 1047 v6  Is+  0:00.00 /usr/libexec/getty Pc ttyv6
 1048 v7  Is+  0:00.00 /usr/libexec/getty Pc ttyv7
 1581  1  S    0:00.01 su -l -l
 1582  1  S    0:00.00 -su (sh)
 1583  1  S    0:00.00 /bin/sh
 1584  1  R+   0:00.00 ps
#
```

使ってみようJail

Jailを使ってみましょう。**jail**(8)コマンドがJailを作成したり削除したりするための制御コマンドです。作成されたJail環境は**jls**(8)コマンドで確認できます。Jail環境が1つもなければ**図6**のような表示が返ってきます。

先ほど作成した/Users/daichi/jを指定してJail環境を構築します。**図7**のコマンドで「/Users/daichi/j」を「/」としたJail環境を作成し、その環境下の/bin/shを実行する、という意味になります。環境下での/bin/shということは、ホストから見れば「/Users/daichi/j/bin/sh」が実行されるということになります。

プロンプトはすでに作成したJail環境下に入っています。pwd(1)を実行すれば自分のいる場所が「/」になったことがわかります(**図8**)。

chroot(8)との大きな違いは、この環境は**プロセス空間も隔離されている**ということです。**図9**のようにps(1)コマンドを実行すると、Jailの中で動作しているプロセスのみが表示され、ホスト側のプロセスを見ることはできなくなります(**図5**と比べてみてください)。

ホスト側のターミナルからjls(8)コマンドを実行すると、**図10**のようにJail環境が1つ作成されたことを確認できます。JailはJIDと呼ばれる番号でアクセスできるほか、ホスト名を指定すればその名前でアクセスすることもできます。

Jail環境ではroot権限で動作す

▼**図6　Jail環境がないときのjls(8)コマンドの出力結果**

```
# jls
   JID  IP Address      Hostname                      Path
#
```

▼**図7　Jail環境の作成とその環境内部でのシェルの実行**

```
# jail -c path=/Users/daichi/j command=/bin/sh
Cannot read termcap database;
using dumb terminal settings.
#
```

▼**図8　指定したディレクトリがルートディレクトリに置き換わる**

```
# pwd
/
# tree $(pwd)
/
|-- bin
|   |-- ps
|   |-- sh
|   `-- tree
|-- dev
|   `-- null
|-- lib
|   |-- libc.so.7
|   |-- libedit.so.7
|   |-- libkvm.so.6
|   |-- libm.so.5
|   `-- libncurses.so.8
`-- libexec
    `-- ld-elf.so.1

4 directories, 10 files
#
```

p.s. こんなところに注意しよう

FreeBSDカーネルはさまざまな機能をJailに対応させていますが、いくつかの機能は対応していないので注意が必要です。

まず、ファイルシステムクォータはJailに対応していません。クォータはユーザIDとグループIDをチェックして動作するだけの機能なので、それぞれのJail環境で同じユーザIDを使っていると、ファイルシステム全体でクォータがかかります。そのため、あるJail環境で容量を使い切ると、ほかのJail環境でもそれ以上ストレージを利用できなくなります。

またファイルシステムの特性上、Jail環境下にあるディレクトリをホスト側がmv(1)してJail環境外のディレクトリに移動させると、Jail環境の制限をはずれてアクセスできるようになってしまいます。Jail環境下にあるディレクトリはmv(1)しないほうがよいといえます。

るといっても、ホストと比べるとその機能は限定されています。**sysctl**(8)の値もホストとJail環境とでは異なります。たとえば、security.jail.jailedという変数は自分がJail環境の中にいるのか外にいるのか判断するために使われるので、ホストとJailとで異なる値が返ってきます(**図11**、**12**)。

Jail環境下ではホストに影響を与えるような機能も動作しません。Jail環境で**図13**のように**shutdown**(8)コマンドを実行しても、メッセージは表示されますが何も起こりません。

ユーザは1つのホストにいくつもJail環境を作成できますし、Jail環境の中でさらにJail環境を作成することもできます。Jail環境下ではファイルシステムのマウントやローソケットへのアクセスなどもデフォルトで禁止されていますが、これらはjail(8)コマンドの引数で利用を許可することもできますので、必要に応じてさまざまな環境を構築できます。

次回はJailにFreeBSDを構築してみましょう。SD

▼図9　Jail環境ではホスト側のプロセスを見ることはできない

```
# ps auxww
USER  PID %CPU %MEM   VSZ  RSS TT  STAT STARTED    TIME COMMAND
0    2065  0.0  0.0 16988 2280  1  SJ    9:18AM 0:00.02 /bin/sh
0    2072  0.0  0.0 16588 1808  1  R+J   9:19AM 0:00.00 ps auxww
#
```

▼図10　ホストから作成したJail環境を確認

```
% jls
   JID  IP Address      Hostname                      Path
     1  -                                            /Users/daichi/j
%
```

▼図11　ホストでのsecurity.jail.jailedは0

```
% sysctl security.jail.jailed
security.jail.jailed: 0
%
```

▼図12　Jailでのsecurity.jail.jailedは1

```
# sysctl security.jail.jailed
security.jail.jailed: 1
#
```

▼図13　Jail環境ではshutdown(8)コマンドのようにホストに影響を与えるコマンドは実行できない

```
# shutdown -p now
Shutdown NOW!
shutdown: [pid 92026]
#
System shutdown time has arrived

#
```

p.s. ちょっとした開発の歴史

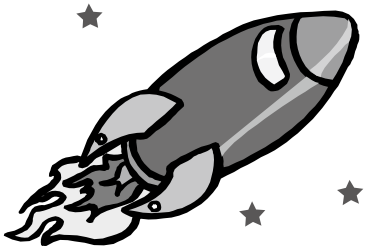
jail(8)コマンドが導入されたのはFreeBSD 4.0が最初です。Jailの機能はリリースごとに拡張を続け、さまざまなリソースを分離できるようになりました。FreeBSD 8.0ではJailの中にJailを作成するといったことが可能になり、FreeBSD 9.1以降はシステムの設定ファイルとしてJailの設定が記述できるようになりました。

JailはもともとFreeBSDデベロッパであるPoul-Henning Kamp氏がR&D Associates向けに開発した機能です。その後、さまざまなデベロッパによって機能の拡張が続き、FreeBSDを特徴付ける機能の1つになりました。

19

Debian Developer　やまねひでき　henrich@debian.org

RHELとの比較第2弾「yum/apt徹底比較」



Debian Hot Topics

Debian8 “Jessie” の開発進捗

Jessieのカーネルバージョン

Debian8 “Jessie” のリリースに向けて、徐々に体制が固まってきています。Linuxカーネルパッケージのメンテナ Ben Hutchingsさんによって、Jessieのカーネルは3.16をベースにすることが発表されました。

時期的に一番良いバージョンであることと、3.16のサポートはLTSではないものの、Ubuntuでも採用するという見通しから、その後のメンテナンスも容易であろうことが述べられています。3.16で不足する機能やモジュールについては、相談すれば追加なども可能であること[注1]も付記されていますので、気になるところがある方は3.16を使ってみて早めに相談するのが良いでしょう。

デフォルトデスクトップの行方

さて、Debianの「デフォルト」デスクトップは、「1枚目のCD/DVDに収める」というサイズ上の問題から、現在は、gitリポジトリ上ではXfceが選ばれるようになっています。これをそろそろ再考しよう、ということで「Reverting to GNOME for jessie's default desktop」(GNOMEのデフォルトデスクトップ環境への復帰)というスレッドが盛り上がっています。

おもな議論は、「GNOMEのサイズの大きさは、開発途上国のユーザがダウンロードする際につらい」、「なぜCD/DVDのサイズにこだわるの？今はUSBメモリがあるじゃないか」、「Xfceは機能不足」、「GNOME3のインターフェースはあまり初心者向きではない」あたりのトピックでしょうか。

デフォルトデスクトップ環境はインストーラで別のデスクトップ環境に選択を変更すれば良いだけですので、個人的には「そこまで大きな影響があるものでもないのでは……」という気がしますが、多くの人にとっては関心を寄せることなのでしょう。この号が発売されるころには結論が出ているとは思いますが、debian-develメーリングリストの行方を見守ってみたいと思います。

arm64およびppc64elアーキテクチャの追加

Debianの公式ミラーに新規にarm64およびppc64elアーキテクチャが追加されることがアナウンスされました[注2]。

64bit ARMのarm64アーキテクチャはわかるとして、ppc64elというのは聞き慣れない人も多いでしょう。IBMが開発したCPUに「Power」シリーズというものがあり、その中の1つがppcアーキテクチャ(＝PowerPC)です。その最新のCPU「Power8」がDebian/Ubuntuでいうppc64el

注1）当然、ケースバイケースですのでその点はご留意を。

注2）URL https://lists.debian.org/debian-mirrors-announce/2014/08/msg00000.html

アーキテクチャとなります。

このPower8を採用したIBMのPower Systemsを、Ubuntuを開発するCanonical社が、Red Hat社とSUSEに続いて、サポートする形になりました[注3]。大量のVMを動かす基盤としてPower Systemsを使い、その上でCanonical社のJujuなどのオーケストレーションソフトウェアを使えば、大量のサーバデプロイメントを簡易に実施できる——そのようなことを売りにするものと推測されます。

またIBMは「OpenPOWER Foundation」というコミュニティを2013年に設立し、多くの企業と連携して開発を進めているようで、GoogleでもPower8を採用したマザーボードを開発したことが表明されています[注4]。サーバ分野でのポテンシャルを持ったアーキテクチャと言えるでしょう。

yum/apt徹底比較

前回、簡単にyum/aptの違いについて触れましたが、より詳細に比較をしましょう(**表1**)。

yumがaptより優れている点は?

どうしても筆者やその知り合いにはapt派が多くなってしまうのですが、「yumがaptより

注3) ですので、ppc64elサポート関連のバグレポートはCanonical関係者が投稿しているものがほとんどです。

注4) URL http://goo.gl/4yB6ph 参照。ただ、大量に採用しているのかどうかまでは触れられていません。

▼表1　yum/aptのおもな操作とオプション

おもな操作	yum(RHEL)	apt(Debian)
リポジトリの追加と削除	`yum-config-manager --add-repo =<repository>`	/etc/apt/souces.list (または/etc/apt/souces.list.d以下)を直接編集(後述)
パッケージデータベースの更新	なし (強制更新であれば`yum makecache`)	`apt-get update`
パッケージ自体の更新	`yum update` (`yum upgrade`も似た動作)	`apt-get upgrade`
パッケージ情報の参照	`yum info <package>`	`apt-cache show <package>`
パッケージのインストール	`yum install <package>`	`apt-get install <package>`
特定バージョンのインストール	`yum install <package>-<version>`	`apt-get install <package>=<version>` (例:`apt-get install sl=3.03-17`)
特定バージョンへのアップグレード	`yum update-to <package>-<version>` (例:`yum update-to 3.03-15.fc20`)	
特定バージョンに固定	なし	`apt-mark hold <package> <version>`
パッケージ群のインストール	`yum groupinstall <group>` (`yum groups install <group>`) ※1	`tasksel install <task>`
ファイルの検索	`yum whatprovides <file>` ※2	`apt-file search <file>` (apt-fileパッケージが必要)
語句を元に関連パッケージの検索	`yum search <keyword>`	`apt-cache search <keyword>`
パッケージ一覧の表示	`yum list`	`apt list`
パッケージの削除	`yum remove <package>` (あるいは`yum erase <package>`)	`apt-get remove <package>`
パッケージファイルの取得	`yumdownloader <package>`	`apt-get download <package>`
パッケージのソースの取得	`yumdownloader --source <package>`	`apt-get source <package>`

※1　RHEL6までは「groupinstall」、RHEL7からは「groups install」です。
※2　apt-fileと比べると、「yum whatprovides」ではファイルについてパスを含めて指定しなければならない点がちょっと面倒な印象です。

優れている点は何か？」と聞いたところ、「ローカルのrpmパッケージファイルを、yumでそのままインストールできる」[注5]という点が挙げられました。

確かにaptにはそのような機能はないので、「# dpkg -i foobar_1.0-1.all_deb」などとdpkgコマンドを併用しなければなりません。パッケージといえばaptしか知らない人にはちょっと不親切ですね。

筆者がyumを調べていて感心したのが「トランザクション機能」です。**図1**のようにすることで、インストール／アンインストールを取り消すことができます[注6]。

Debianの場合、パッケージマネージャーの動作記録としてdpkgのログが/var/log/dpkg.logに、そしてaptのログが/var/log/apt/history.logなどに残ってはいますが、残念ながらこれらを元にしてのundoやrollbackはできません。

yumで付加情報を元に細やかなアップデート

また、yumの場合には、パッケージの更新についてそれぞれ「バグ修正(Bugfix)」「セキュリティ修正(Security)」「機能拡張(Enhancement)」のいずれかの情報が付加されています。加えて「どのBugzilla IDを修正するものか」「どのSecurity Advisory／どのCVEに対応したセキュリティ修正なのか」「重要度がどの程度なのか」などの

注5) 「$ sudo yum install http://example.com/hoge.rpm」というように、リモートにあるRPMパッケージもインストール可能だそう。テストのときには便利ですね。

注6) このトランザクション機能は便利……ですが、パッケージのアップデートについては、ダウングレードに失敗して役に立たないこともしばしばあります。
rawhide(Debianでいうところのunstable)で大量にアップデートをしたが、それをキャンセルしたい……という用途には不向きです。

▼図1　yumのトランザクション機能

```
$ yum history rollback <transaction id>
$ yum history undo <transaction id>
```

▼図2　yumのパッケージ更新ごとに付けられている付加情報(区分)

```
$ yum updateinfo
読み込んだプラグイン:langpacks, refresh-packagekit, upgrade-helper, verify
Updates Information Summary: available
    7 Security notice(s)
    30 Bugfix notice(s)
    8 Enhancement notice(s)
updateinfo summary done

$ yum updateinfo list
読み込んだプラグイン:langpacks, refresh-packagekit, upgrade-helper, verify
FEDORA-2014-8033 bugfix      NetworkManager-1:0.9.9.0-41.git20131003.fc20.x86_64
FEDORA-2014-8033 bugfix      NetworkManager-glib-1:0.9.9.0-41.git20131003.fc20.x86_64
FEDORA-2014-7193 enhancement audit-2.3.7-1.fc20.x86_64
FEDORA-2014-7193 enhancement audit-libs-2.3.7-1.fc20.x86_64
FEDORA-2014-7193 enhancement audit-libs-python-2.3.7-1.fc20.x86_64
FEDORA-2014-7992 security    file-5.19-1.fc20.x86_64
FEDORA-2014-7992 security    file-libs-5.19-1.fc20.x86_64
(...省略...)
updateinfo list done
```

各区分についてそれぞれのアップデート数が表示される

↑どの区分かが表示される

▼図3　yumにおける細かな条件指定によるパッケージ更新

```
$ sudo yum --security update                    ←セキュリティアップデート関連だけ更新
$ sudo yum --security update-minimal kernel*    ←カーネル周りのセキュリティアップデートのみ摘要
$ sudo yum update --cve CVE-2014-0195           ←特定のCVEに対応するパッケージのみ更新※
$ sudo yum update --advisory RHSA-2014:0679     ←特定のSecurity Advisoryに対応するパッケージのみ更新
```

※　RHELでは動いたけど、Fedora20だと動かなかったり……このあたり深追いはできていません。

追加情報もあります(**図2**)。

これらを利用して、セキュリティ修正だけ／一定の重要度以上の修正だけ／特定のバグに関する修正だけを適用できるのが好ましい、という方もいらっしゃるでしょう。オプションを見ると、セキュリティ修正フラグが立っているもの／セキュリティ修正の重大さ／BugzillaのID／CVE／アドバイザリ名などでの指定が可能となっているようです(**図3**)。

残念ながらaptでは、Debianパッケージにここまで細かな付加情報がないので、このようなハンドリングは難しいですね(逆に言えばメタ情報として追加するようにすれば、対応できるということでもあります)。

aptも頑張れ

ここまで読んでくると、筆者がまるでyum派のように見えるかもしれませんが、使い慣れているのはもちろんaptのほうです。私的な視点ではaptが頑張ってほしいところは、

- リポジトリを手軽に有効／無効にできるようにする
- 差分パッケージの取扱いができるようにして、パッケージのダウンロード時間を減らす[注7]
- プラグイン機構で機能を追加できるようにする[注8]
- トランザクションをサポートして、undo/rollback機能を追加する

あたりでしょうか。逆にaptのほうが優れているな、と感じる点は次のとおりです。

- インストール時などの高速な動作(重要！)
- オプションがシンプル
- 「Recommends(推奨)」パッケージの導入が可能(無効にすることもできる)
- 上記に加え「Suggests(提案)」パッケージの情報がある
- (メタ)データが破損して修復作業が必要になることが、yumと比較して少ない

apt/yum比較総評

筆者の感想ですが、yumは後発だけあって多彩なオプションがあり、使いこなしを考えると今回取り上げたように良いところも多いです。

ですが、パッケージインストールの際に、細かくオプションを指定する場面もそう多くはありません。日常的な利用ではアップデートやインストールだけを把握していれば良く、そこまでの違いは感じられないかもしれません。SD

注7) yumだとdelta RPMの取扱いをPrestoプラグインで実現していますが、ダウンロード後に差分からパッケージを生成する処理が発生するので、ディスクが遅いとそれはそれで時間がかかります。この点はPrestoプラグインでも「高速な回線で低速なマシンを使うときは無効にしたほうが速い」と触れられています。

注8) 「`yum search yum-plugin`」とすればさまざまなプラグインが見つかります。

COLUMN 「apt」コマンド

Debian/Ubuntuユーザにはお馴染みのapt-getコマンドですが、aptバージョン1.0のリリースとともに、シンプルなaptコマンドが追加されました。

機能はapt-getの一部を抜粋したものですが、「進捗ステータスが表示される」「出力がカラフルになっている」などユーザが馴染みやすくなっています。

COLUMN Debianにもyumをインストールできます

実はDebianにもyumパッケージが存在していて「`apt-get install yum`」でインストールが可能です……が、インストールできるだけです。初期状態では全然設定が整っていませんし(何もリポジトリが設定されていない)、そもそもaptのパッケージデータベースと互換性もなく、連携もしていません。

何も考えずにyumでパッケージを入れたら、aptで入れてある既存のファイルを上書きしてしまってシステムを壊すだけのオチになります。

ちなみにcreaterepoパッケージもありますので、Debianでrpmファイルを元にRHEL/CentOS用リポジトリを作成して提供することが可能です。

第25回

オープンソースソフトウェアを開発するということ

菅原 健
SUGAWARA Ken
レッドハット（株）グローバルサービス本部
プロフェッショナルサービス部
プラットフォームシニアコンサルタント

はじめに

2度目の登場となりましたプラットフォームコンサルタント菅原です。今回もよろしくおつき合いください。今回は「オープンソースソフトウェアを開発するということ」というテーマを設定してみました。本連載の第1回から藤田も書いておりますように、Red Hatは「オープンソースカンパニー」を標榜しております。それこそ水や空気のように日々オープンソースどっぷりの生活を送っているわけですが、なぜオープンソースなのか、ということを少し掘り下げて考えていきたいと思います。

なぜオープンソースなのか

フリーソフトウェア・オープンソースソフトウェア（以下Free/Libre and Open Source Softwareを略したFLOSSと記します）黎明期には、著作権を意味する「copyright」をもじった「copyleft」という概念や、ソフトウェアを無償で公開することなどの理由でFLOSS推進者達が「ソフトウェア共産主義者」などと呼ばれたことがあったりしたことなどから、FLOSS推進はイデオロギーの一種、IT業界でよく言う「宗教」の1つのように見られることもよくあるように思います。しかし、Red Hatを始めとして、FLOSSをビジネスとして商業的に成功を収めている会社がいくつもあることを見ても、この見方は（完全な誤りではなかったとしても）かなり偏った、穿った見方のように思われます。あるいは、FLOSSが成功することにより既得権益が侵される？——と恐れた勢力による意図的なレッテル貼りだったのかもしれません。

少なくとも、筆者がFLOSSを生業（なりわい）として選んでいるのは宗教的信念やイデオロギーではなく現実的な利点によるものであり、社会の趨勢（すうせい）として今後FLOSSが——プロプライエタリソフトウェアを完全に駆逐することはないかもしれませんが——ソフトウェア開発モデルの主流となっていくだろうと予測しているからです。筆者の予測の正誤はともかくとして、私企業がビジネスを継続して発展させていくためには利潤を無視できないという大前提に立ったうえで、多くの企業がFLOSSを自社システムに採用し開発にも参画、さらには自社製品をFLOSSとして公開までしているわけですから、FLOSSが企業の利潤に対しプラスの効果を持っているのは間違いないところだと感じています。

では、現実的な利点としてどのようなものがあるか、代表的なものをいくつか示したいと思います。

FLOSSの利点

ユーザのIT統制強化に有益

ソースコードなど内部原理が未公開であるプロプライエタリソフトウェアにおいては、製品の利用上何か不都合があったときのサポートは必然的にベンダ任せとせざるを得ません。FLOSSでは当然ながらソースコードが公開されていますから、Red Hatのようなベンダに問

い合わせを行うのと平行して自己調査ができます。仮に自らソースコードを調査してバグの該当個所を特定できないまでも、似た現象に対する修正がアップストリームに提案あるいはコミットされていないかメーリングリストなどを検索するなど、自らできる調査の幅がプロプライエタリ製品よりも幅広くなっています。

このように、公開されている情報を有効に活用してユーザ自ら調査を行うことにより、何から何までベンダ任せでおんぶにだっこのベンダ依存体質を改め、ひいてはIT統制の強化に利用しているユーザ企業がすでに数年前から現れてきています。

開発と利用の継続性が高い

プロプライエタリ製品は、開発元がビジネス上の理由で開発をやめてしまったり倒産してしまったりすると、そのユーザは多くの場合利用の継続は困難となります。まあ、今あるバイナリをそのまま使い続ければいいのでは、という考え方もありますが、ハードウェアが新しくなるとOSもバージョンアップする必要があり、その影響でアプリケーションが動かなくなったり、延命には限界があるのが普通です。

ところが、FLOSSの場合は開発元が開発をやめても最悪倒産して存在しなくなったとしても、FLOSSライセンスの下で公開されたソースコードは残りますので、それを利用して元の開発元とは違う別の誰かが開発を継続していくことができます。その気になれば、ユーザ企業が開発を引き継ぎ新たなOS環境への対応などを継続していくことだってできます。

また、これはある起業家のブログを読んで知ったことです。その方の場合は前に興した企業が倒産してしまったのですが、そこで開発していたソフトウェアをFLOSSライセンスで公開していたため、そのあとの開発を自ら継続できた、というケースもあるそうです。プロプライエタリとしていたならば、会社が倒産した時点でソフトウェアは資産として管財人の管理下に入り、自ら開発した製品であったとしても勝手に開発を継続することはできなかったでしょう。

組織を超えた開発規模

FLOSSとして成功しているソフトウェアはほとんどの場合、単一の組織の枠を超えた大きな開発者コミュニティによって支えられています。たとえばLinux kernelですが、The Linux Foundationが公開している最新の情報[注1]によれば、組織としてのRed Hatの貢献は10.2%に過ぎません。非常に大雑把な言い方ですが、現在のLinux kernelの開発速度をRed Hat単体で維持しようとすると、今の10倍の数の開発者を雇い入れる必要があることになります。逆に言えば、単一の企業にとってはプロプライエタリな開発モデルと同じコストでより大規模かつ速い開発ペースを実現できるのがオープンソース化の魅力だということになります。

プロプライエタリソフトウェアの場合は基本的に1社だけで開発を進めるわけですから、製品規模や開発ペースを上げようとすれば必然的に開発者数も増えて開発コストが増大し、それが製品価格に跳ね返ってくるわけですね。

FLOSSのリスク

逆にオープンソース開発モデルには特有のリスクもあります。リスクも正しく理解しないとリスクと利益を比較して採否を決定できません。こちらも代表的なものを示します。

成果を独占できない

当然ながらソースコードを公開するわけですから誰も成果物(=ソフトウェア)を独占的に利用することはできません。トレードシークレットにかかわるアルゴリズムをFLOSSとして実

注1) "Linux Kernel Development - How Fast It is Going, Who is Doing It, What They are Doing, and Who is Sponsoring It", The Linux Foundation, September 2013, Jonathan Corbet, Greg Kroah-Hartman, and Amanda McPherson

装することは自殺行為に等しい場合もあり得ますからFLOSS開発へ参画する場合は相応の検討が必要です。

開発ロードマップ

とくに活発な開発者コミュニティが存在する場合に、多くの開発者からさまざまな貢献がありますので、開発の道筋を1社が決めることは困難となります。これは、1社だけの思惑で将来のロードマップが描きにくくなることを示します。この特性はユーザからみれば利点となり得ますが、開発者としてはプロジェクトが制御不能になり得るリスク要因として検討しておく必要があります。

安易なオープンソース化に対する疑問

と、ここまでFLOSSの特徴を見てきたわけですが、自社開発したソフトウェアのFLOSS化もしくは初めからFLOSSとして自社開発を決定する際に重要なのは、リスクベネフィット分析の結果、潜在リスクよりもFLOSS化の利益のほうが大きいとの評価が出せることです。ところが、世の中には上記のような利点をまったく活用できていないか、活用していたとしても非常に薄弱な「FLOSS」が見られます。とくに3つめの利点には活発な開発者コミュニティの醸成が不可欠ですので必ずしも達成できるものではないのですが、もはや初めから開発加速効果の達成をあきらめているとしか思えない「FLOSS」があります。

それはどんなものかというと、名前だけオープンなFLOSS、つまり、既存のFLOSSライセンスもしくはその派生形ライセンスのもとでソースコードが公開されている「だけ」のFLOSSです(上で「FLOSS」とカギ括弧つきにしているのは名前だけのFLOSSだからという意味を込めたものです)。とくにソースコードがtarボールなどでダウンロード可能になっているだけで、GitHubで公開されているわけでもパッチの送り先も示されていないような類のものを見ると、この開発者はいったいどういう目的でこのソフトウェアをFLOSSにしたのだろうか、と不思議になります。まさにFLOSS開発モデルの利益とリスクを比較検討することなく、「初めにFLOSS化ありき」で公開されているのではないか、と勘ぐりたくなるものがあるのです。

個人の趣味の産物ならまだしも、企業が投資して開発するFLOSSについては、その企業自身のビジネスにプラスになるという評価のうえで開発が行われ、活発な開発者コミュニティが醸成されて誰もが開発のスケールメリットを享受できるようなプロジェクトが増えてくることを切に願うものです。

レッドハットの自転車クラスタ

さて、話変わってゆる〜い話題です。

実は当社には自転車乗りがけっこういます。SNSなどで見てもITクラスタと自転車クラスタがかなり被るのはよく知られていますので意外でもなんでもないのですが、レースなどのイベントにチームとして出場したりはしていませんので、外から見てあまり目立っていないと思います。何せチームの母体となる日本の社員数という意味では、ほかの大手IT企業様とは比較にならない少なさですし、残念ながら会社公認の部活動を行うには至っていないという点も大きいかと思います。

しかし、US本社のほうでプロモーションとしてRed Hatロゴ入りジャージを作った実績があって一定数以上の発注をすれば日本からでも買えることがわかり、ついに去年の秋にRed Hatジャージを共同購入しました!——というわけで、去年からレース会場にもRed Hatジャージが、細々と少人数ではありますが出没しております。もしどこかでRed Hatジャージを見かけたらきっとそれは恵比寿の誰かですので、気軽に声をかけてくださいね。SD

BOOK no.1 Java8ではじめる「ラムダ式」

清水 美樹 【著】
A5判、192ページ／価格＝2,300円＋税／発行＝工学社
ISBN＝978-4-7775-1841-8

今年3月にリリースされた「Java 8」の「ラムダ式」を学ぶ本。ラムダ式の基礎から、データのソートを行うComparatorインターフェースや、Swing・JavaFXのイベント実装にラムダ式を活用する例といった、実践的な内容を扱っている。Javaの従来の記法と、ラムダ式で書かれたコードを比較しながら解説しており、また、なぜラムダ式がJavaに導入されたのかの考察もされている。ラムダ式がどんな個所に適用できるのか、適用するとどのようなコードになるのか、勘所をおさえながら学べるだろう。Java 8から追加された、コレクションを扱う「Stream API」についても、ラムダ式を引数に使って効率的にデータを処理する方法が紹介されている。

BOOK no.2 エンジニアのための フィードバック制御入門

Philipp K. Janert 【著】／野原 勉 【監訳】／星 義克、米元 謙介 【訳】
A5判、344ページ／価格＝3,200円＋税／発行＝オライリー・ジャパン
ISBN＝978-4-87311-684-6

フィードバック制御とは、「システムの実際の挙動とその望ましい挙動を常に比較しつつ動作させる」ための方法である。産業界ではよく使われるが、ソフトウェアエンジニアには疎遠なこの理論について、原理から実装、ソフトウェアシステムへの活用、応用的な理論までを扱った本である。キャッシュの最適サイズ、サーバ数の調整、メモリ消費の制御のようなソフトウェア技術に関する課題に対して、フィードバック制御を応用するケーススタディを、Pythonによるシミュレーションコードを載せながら解説している。内容はかなり専門的で、微積分や偏差など、数学的知識を前提に解説しているので、事前の勉強は必要となるだろう。

BOOK no.3 Vim scriptテクニックバイブル
Vim 使いの魔法の杖

Vim scriptサポーターズ 【著】
A5判、320ページ／価格＝2,580円＋税／発行＝技術評論社
ISBN＝978-4-7741-6634-6

Vimに組み込まれたスクリプト言語「Vim script」について、基本、使用例から、実践的なテクニック、プラグインの作成・公開の手順までを解説している。「Hello World」の表示から説明されており、初心者にも優しい。第5章では「保存時に自動的に行末の空白スペースを削除するプラグインの作成」をケーススタディとして取り上げており、ほかの章の基本事項をおさえたうえで、自分だけのプラグイン作成の参考にしてほしい。第6章では、addやdeleteをはじめ、よく使われる組込み関数が網羅的に紹介されており、辞書的に使うこともできる。自分のVim環境をカスタマイズする目線で読むと良いだろう。なお本書は、Vim7.4に対応している。

BOOK no.4 Hinemos 統合管理［実践］入門

倉田 晃次、澤井 健、幸坂 大輔 【著】
B5変形判、520ページ／価格＝3,700円＋税／発行＝技術評論社
ISBN＝978-4-7741-6984-2

「ひねもす」は「終日」という意味で、HinemosはNTTデータが提供しているオープンソースソフトウェアである。近年のITシステムはオープン化、仮想化、クラウド化など複雑化してきている。Hinemosは、複数のコンピュータ群を単一のコンピュータのようなイメージで、一元的に運用管理できるのが特徴だ。本書では、その中でも重要な「監視」「性能管理」「ジョブ管理」などの項目ついて解説している。機能や使い方については公式ドキュメントもあるが、本書は、Hinemosにおける設計の考え方、設定方法について、より具体的に想定される事例に基づいて記載している。また、Hinemos技術者認定プログラムのテキストとしても利用できる。

mini-builddでパッケージのビルド&配布環境を構築する

Ubuntu Japanese Team
あわしろいくや　ikuya@fruitsbasket.info

今回は、ローカルにパッケージのビルド環境と配付環境を整え、実際にパッケージをインストールするところまでを紹介します。

パッケージの配布環境

UbuntuにはPPAという優れたしくみがあります。PPAはPersonal Package Archiveの略で、その名のとおり個人向けのパッケージリポジトリです。一定の条件を満たせば誰でも任意のパッケージをアップロードできます[注1]。とはいえ、PPAにアップロードするということは公開するということであり、中には公開したくないパッケージもあるでしょう。また、十分な速度が出るインターネット環境がない場合はPPAへのアップロードもダウンロードも待ち時間が長くなります。それに、PPAにアップロードできるのはソースパッケージやバイナリパッケージも含めて2GBまでですが、それよりもっとたくさんのパッケージをアップロードしたいということもあるでしょう。

そういった場合には、今回紹介するmini-builddを使用すると便利です。Webアプリなので設定方法は難しくないのですが、設定項目が多くて戸惑うことはあるかもしれません。ひとつひとつ確実に行いましょう。

注1）もちろんライセンスなどの制限はありますので、なんでもということではありませんが。

インストール

今回は仮想環境にUbuntu 14.04 LTS Serverをインストールしました。ホスト名は"UbuntuTrusty Buildd"です。mini-builddのパッケージはUbuntu 14.04にもありますが、古いので新しいものを使用します。とはいえ、なぜかPPAのビルドに失敗してしまうので、ちょっと長いですが14.10のパッケージをダウンロードします（**図1**）。

ログイン

まずはWebブラウザを起動します。avahi-daemonが動作している場合、mini-builddが動作しているUbuntuのホスト名に.localをつけると簡単にアクセ

図1　mini-builddのインストール

```
$ wget http://jp.archive.ubuntu.com/ubuntu/pool/universe/m/mini-buildd/mini-buildd-common_1.0.3_all.deb http://jp.archive.ubuntu.com/ubuntu/pool/universe/m/mini-buildd/mini-buildd_1.0.3_all.deb http://jp.archive.ubuntu.com/ubuntu/pool/universe/m/mini-buildd/python-mini-buildd_1.0.3_all.deb
$ sudo apt-get install mini-buildd
$ sudo dpkg -i  mini-buildd-common_1.0.3_all.deb  mini-buildd_1.0.3_all.deb python-mini-buildd_1.0.3_all.deb
```

※もしダウンロードできない時は、1.0.3を1.0.4や1.0.5などに変えてみてください。
※インストールの際にadminのパスワードを入力する必要があります。
※最後にhavegedパッケージをインストールしてください。

スできます。たとえば、

```
http://ubuntutrustybuildd.local:8066/
```

のような感じです[注2]。

右上に“logged off”というリンクがありますが、これをクリックするとログインのユーザ名とパスワードを入力できます。ユーザ名は“admin”で、パスワードは先ほど決定したものです。赤字で“User admin”となっていれば、ログインができていることになります。

いよいよ設定に移ります。まずは左上の“Configure”をクリックしてください。“Site administration”に移動しました(**図2**)。“Mini_Buildd”、“Auth”、“Registration”という大項目がありますが、今回は“Mini_Buildd”しか設定しません。“Mini_Buildd”も5つの項目にわかれていますが、上4つしか設定しません。では、細かくみていきましょう。

Daemon

“Daemon”には“Daemon”という項目しかありません。“Change”をクリックしてページを変遷し、“Daemon”の下にある、ホスト名から始まるリンクをクリックします。“Change daemon”というページに遷移するので、“Archive idエンティティ”の“Identity”と“Hostname”と“Email address”を変更します。“Identity”はこのままでもかまいませんが、短いほうが楽です。今回は“trusty”とします。“Hostname”は、前述のとおりavahi-daemonをインストールしているのでホスト名に.localを付けたものにします。すなわち今回は“UbuntuTrustyBuildd.local”です。“Email address”はご自分のメールアドレスにしてください。あとの変更は任意です。一番下までスクロールし、“save”をクリックしてください。

項目の左端にチェックを入れるところがありますので、ここにチェックを入れて“Prepare”をクリックしてください。するとステータスが黄色になりますので、もう一度チェックを入れて“Check”をクリックしてください。最後にもう一度チェックを入れて“Activate”をクリックすると作業が完了します。この操作は今後よく出てきます。あとは“Home → Mini_buildd → Daemon”の“Mini_buildd”をクリックして戻ってください。

Sources

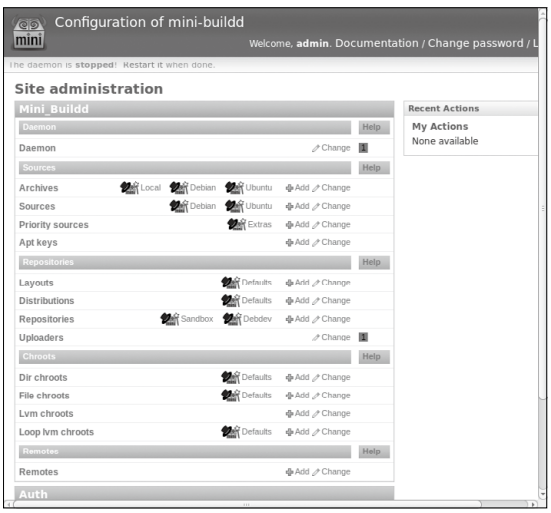
“Sources”は“Archives”と“Sources”を設定します。“Archives”はまず“local”をクリックしてください。次に“Changes”をクリックし、もしあれば“/”と“cdrom”を削除します。それぞれクリックすると左下に“Delete”が表示されるのでこれをクリックし、“Yes, I'm sure”をクリックします。

“Source”は“Ubuntu”をクリックします。次に“Change”をクリックして画面遷移し、“Ubuntu 'trusty' from 'None'”の行にチェックを入れ、“Prepare”→チェック→“Check”→チェック→“Active”でアクティブにします。これで完了なので前の画面に戻ってください。

Repositories

“Repositories”は4つ全部の設定が必要です。まず、“Layouts”の“Defaults”をクリックします。次に“Distributions”の“Defaults”をクリックします。さ

図2　設定前



注2） avahiでは、ホスト名に大文字小文字の区別はありません。

らに“Repositories”の“Sandbox”をクリックします。最後に、“Changes”をクリックして画面遷移し、“Ubuntu 'trusty' from 'None'”の行にチェックを入れ、“Prepare”→チェック→“Check”→チェック→“Active”でアクティブにします。これで完了なので前の画面に戻ってください。

“Uploaders”の“Change”をクリックし、さらに“'admin' may upload to '' with key ': '”をクリックしてください。ここでソースパッケージのアップロードに使用するGPGキーを指定します。GPGキーの作成方法は省略します。“Prepare”→チェック→“Check”→チェック→“Active”でアクティブにします。これで完了なので最初の画面に戻ってください。

Chroots

“Chroots”で設定するのは“Dir Chroots”だけですが、ちょっとトリッキーなことを行いますので注意してください。まずは“Dir chroots”の“Defaults”をクリックします。次に“Changes”をクリックします。画面を遷移すると2行表示されているはずなので、どちらにもチェックを入れて“Prepare”をクリックします。Chrootsの設定はパッケージのダウンロードを行うため、少々時間がかかります。完了したら、ここでいったんWebブラウザから離れて端末に移動してください。

この2つのファイル（**図3**）の中身を、それぞれ**図4**のように変更してください。

2つのファイルとも変更が終わったら、Webブラウザに戻り、2行ともにチェックを入れて“Check”をクリックしてください。これも終了するまで時間がかかります。最後にまた2行ともチェックを入れて、“Active”をクリックしてください。これで設定は完了です（**図5**）。

図5　設定完了後

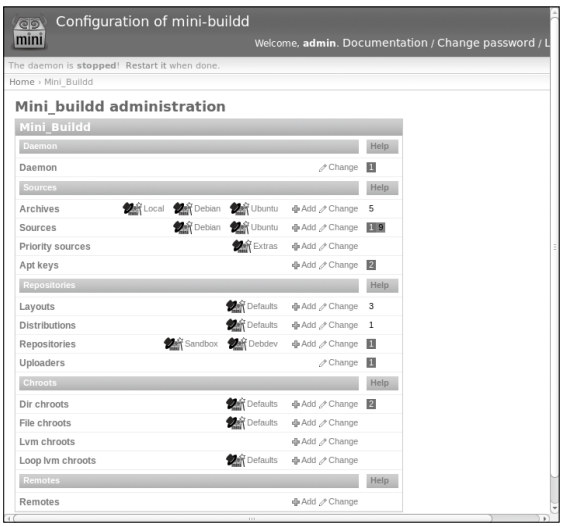


図4　変更内容

```
deb http://jp.archive.ubuntu.com/ubuntu trusty main
```

↓

```
deb http://jp.archive.ubuntu.com/ubuntu trusty main universe
```

キーリングを作成する

左上のmini-builddアイコンをクリックすると、トップページに戻ります。戻ると、そのアイコンのすぐ右下に“Start”があるので、これをクリックしてデーモンを起動してください。続けて“Keyring packages”をクリックします。何度か再読み込みしていると、パッケージのビルドが終わったことがわかります。これで準備が整いました。

パッケージのアップロード

パッケージのアップロードはLAN内からであればどこから行ってもいいのですが、今回はmini-builddが動いている環境をそのまま使用します。アップロードするパッケージは何でもいいのですが、今回はSylpheedにします。次のコマンドを入力して、環境を整えてください。

図3　編集するファイル

```
/var/lib/mini-buildd/var/chroots/trusty/amd64/source/etc/apt/sources.list
/var/lib/mini-buildd/var/chroots/trusty/i386/source/etc/apt/sources.list
```

```
$ sudo apt-get install ubuntu-dev-tools
$ pull-lp-source sylpheed
$ cd sylpheed-(バージョン)
$ dch -i
```

Changelogの編集画面になりますが、**図6**のようにします。

まず1行目、今回はSylpheedのバージョンが“3.4.1-1ubuntu2”でした。その後ろに“~test1404+1”を追加してください。その次は“trusty-test-unstable”です。これらは、作成されたkeyringの“Distribution”と“Version”を見ると、そうしなければいけないことがわかります。3行目の変更点は任意にしてください。5行目の名前とメールアドレスは、署名するGPGのものと一致している必要があります。

Changelogの保存が完了したら、次のコマンドを実行してください。

```
$ cd ../
$ dpkg-buildpackage -S -sa
```

これでGPGのパスフレーズを聞かれるので、入力します。

ソースパッケージの作成は完了したので、次にアップロードに必要な設定を行います。次のコマンドを実行してください。

```
$ mini-buildd-tool (mini-buildd サーバーの
ホスト名):8066 getdputconf >>~/.dput.cf
```

もし“mini-buildd-tool”が見つからない場合は、“python-mini-buildd”パッケージをインストールしてください。 mini-buildd サーバのホスト名は、今回の例では“ubuntutrustybuildd.local”です。

では、ソースパッケージをアップロードします。次のコマンドを実行してください。

```
$ dput mini-buildd-trusty sylpheed_3.4.1-
1ubuntu2~test1404+1_source.changes
```

“dput”はコマンドですが、引数の“mini-buildd-trusty”は、“mini-buildd-(最初に入力したIdentity)”というルールがあります。よくわからない場合は~/.dput.cfの1行目にありますのでそれを参考にしてください。

パッケージのインストールは別のUbuntu 14.04で行います。aptラインはトップページの“Overview”→“(Show)”をクリック→“trusty-test-unstable”の行をクリックすると表示されます。今回の例では**図7**のとおりです。次に、**図8**のコマンドを実行するとパッケージがインストールされます。SD

図6　Changelogの編集例

```
sylpheed (3.4.1-1ubuntu2~test1404+1) trusty-test-unstable; urgency=medium

  * Upload to mini-buildd.

 -- Awashiro Ikuya <ikuya@fruitsbasket.info>  Sat, 16 Aug 2014 16:11:18 +0900
```

図7　今回のAPTライン

```
$ deb http://UbuntuTrustyBuildd.local:8066/repositories/test/ trusty-test-unstable main multiverse
restricted universe
```

図8　パッケージのインストール

```
$ sudo bash -c 'echo "deb http://UbuntuTrustyBuildd.local:8066/repositories/test/ trusty-test-unstable
main multiverse restricted universe" >/etc/apt/sources.list.d/tmp.list'
$ sudo apt-get update
$ sudo apt-get --allow-unauthenticated install trusty-archive-keyring
$ sudo rm -rf /etc/apt/sources.list.d/tmp.list
$ cd /etc/apt/sources.list.d/
$ sudo ln -s /usr/share/mini-buildd/sources.list.d/trusty_trusty_test_unstable.list .
$ x-mac-japanese apt-get update
$ sudo apt-get install -t trusty-test-unstable sylpheed
```

第31回

Linux 3.15の機能
FUSEとサスペンドからの復帰の高速化

Text：青田 直大　AOTA Naohiro

8月3日にLinux 3.16がリリースされ、そのバグを修正したLinux 3.16.1も8月14日にリリースされています。さらに、Linux 3.17-rc1もすでにリリースされていて、Linux 3.17に入る機能がすでにでそろってきています。

今月は残りのLinux 3.15の機能として、おもにFUSEの更新を紹介します。

FUSEとは

まずは、新しくwritebackサポートが行われたFUSEについて見ていきます。FUSEとは“Filesystem in Userspace”の略称で、その名のとおりファイルシステムをユーザランドで実装できるようにするフレームワークです。一般にLinuxのファイルシステムはカーネルの中で実装されていますが、カーネルの中でファイルシステムを実装するのにはいくつかの制限がついてしまいます。たとえば、カーネルの開発言語であるC言語を使わなければいけないことや、既存のユーザランドで動作するライブラリを使用できないことが挙げられます。この問題を解決するのがFUSEです。FUSEを使うことでファイルシステムをユーザランドで実装できるようになります。ユーザランド上で実装を書くことができるので、さまざまな言語で自分の好きなファイルシステムを実現できますし、万が一バグが起きてもカーネルが止まってしまうことはなくなるので比較的安全な開発が可能になります。Wikipediaにリストされている例[注1]の中では、ssh接続先のファイルシステムをNFSライクに、自分の手元にあるようにmountできるsshfsや、Windowsのファイルシステムであるntfsをmountするntfs-3Gなどが有名でしょうか。変わったものではGmail上にメールとしてデータを保存するGmailFSや、Nagiosで監視しているマシンの監視項目とその値をファイルシステムとして読むことができるNagiosFSといったものまで実装されています。

FUSEのしくみ

FUSEはカーネル側の実装であるfuse.koとユーザランドのプログラムとが/dev/fuseを通じてデータをやりとりすることで実現されています。ユーザがFUSEのファイルシステムがmountされているディレクトリ下に操作を行うとそのことがユーザランドのプログラムに通知されるので、プログラムはそのリクエストを処理して、その結果をカーネルに返します。そして、

注1）http://en.Wikipedia.org/Wiki/Filesystem_in_Userspace#Example_uses

カーネルから元のユーザへとその結果が転送されていきます。

たとえばfusedirにFUSEをmountしているとしましょう。“ls fusedir/foo”というコマンドを実行すると、lsがシステムコールstatを発行します。カーネルがシステムコールに対応するfuse.ko内の関数を呼び出し、FUSEはstatであれば“LOOKUP”というメッセージのIDと、その引数であるファイル名の“foo”をプログラムに通知します。FUSEのプログラムはこれに対して、ファイル“foo”のinode番号やパーミッション、リンク数といったデータを返します。fuse.koはそのデータを適宜変換して、“ls”に“fusedir/foo”のstat情報が返されます。

通常FUSEのプログラムはlibfuseを用いて実装します(**図1**)。このlibfuseが/dev/fuseとの通信を実現するので、プログラムはstatやreadやwriteといったそれぞれのファイルシステムの処理を行う関数を登録し、カーネルからのメッセージを待つイベントループを実行する関数を呼び出すだけで、通信内容を知らずにプログラムを実装できます。

しかし、今回は/dev/fuseを使ったプログラムとカーネルとのやりとりを詳しく見るために、あえてlibfuseを使わずに簡単なファイルシステムを実装してみましょう(**リスト1**)。今回の実装ではmountしたディレクトリの直下のファイルを“ls -l”で参照し、ファイルを読み込むと“test data”が3行出力され、書き込みができるだけのファイルシステムを作ってみます。

FUSE：初期化

FUSEのプログラムが実行されるとまずはmountを行わなければいけません。nmountオプションには/dev/fuseを開いているファイルデスクリプタ番号を指定する必要があるので、/dev/fuseを開いて、そのほかのmountオプションをつけてmount()を呼び出します(**リスト1❶**)。mountのソース部分はとくに解釈されません。また、typeの部分を“fuse”としていますが、“fuse.sshfs”のように“.”のあとに、より詳しいtypeを指定することもできます。

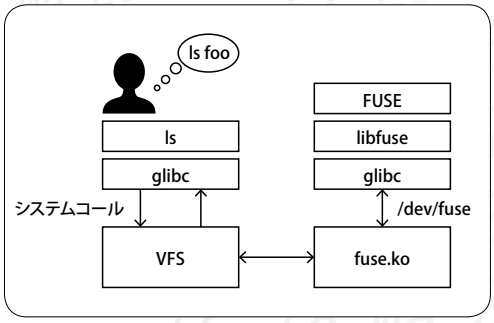
mountのあとは、まずINITメッセージを受信し、その返信から行います。カーネルからのメッセージは/dev/fuseを開き、mountオプションに指定したファイルデスクリプタから読み込むことで受信できます(**リスト1❷**)。読み込まれたデータには、まず先頭にすべての種類のメッセージに共通で先頭にstruct fuse_in_headerのデータが入っています。ここには次のデータが入っています。

- メッセージ長
- オペコード
- メッセージ番号
- ノードID
- ユーザID・グループID
- プロセスID

オペコードはFUSE_LOOKUPなどファイルシステムの操作の種類を示し、ノードIDはinode番号と同じようなファイル識別のための番号で、ファイルシステムのディレクトリの場合、1となっています。ユーザID・グループID・プロセスIDがファイルシステムの操作を行ったユーザ・グループ・プロセスを示すために使われます。

INITメッセージの場合には、このヘッダのあとにカーネル側のFUSE実装のバージョン番号や、有効な機能などを示すフラグなどのデータが入ったstruct fuse_init_inのデータが続いて

▼図1　libfuseを用いたFUSEの動作



▼リスト1　fuse.c FUSEを使った簡単なファイルシステム実装

```
#include <stdio.h>
#include <errno.h>
#include <fcntl.h>
#include <linux/fuse.h>
#include <sys/mount.h>
#include <string.h>
#include <unistd.h>

#define SIZE 132*1024
#define DATA "test data\ntest data\ntest data\n"
#define min(a, b) (((a)<(b))?a:b)

void reply(int fd, int unique, int error, void* data, int len)
{
  char buf[SIZE];
  struct fuse_out_header out;
  out.unique = unique;
  out.error = error;
  out.len = sizeof(out) + len;

  memcpy(buf, &out, sizeof(out));
  if (len != 0)
    memcpy(buf + sizeof(out), data, len);
  write(fd, buf, out.len);
}

int main()
{
  // open /dev/fuse
  int fd = open("/dev/fuse", O_RDWR);

  // ❶ mount
  char tmp[128];
  snprintf(tmp, sizeof(tmp),  "fd=%i,rootmode=40000,user_id=0,group_id=0", fd);
  mount("", "/home/naota/fusemount", "fuse", MS_NOSUID | MS_NODEV, tmp);

  struct fuse_in_header *in;
  char buf[SIZE];
  size_t n;

  // ❷ init
  n = read(fd, &buf, SIZE);
  in = (struct fuse_in_header*)&buf;
  printf("unique: %llu, opcode: %i, nodeid: %lu, insize: %zu, pid: %u\n",
         (unsigned long long) in->unique, in->opcode,
         (unsigned long) in->nodeid, n, in->pid);
  struct fuse_init_in *init_arg = (struct fuse_init_in*)((void*)&buf + sizeof(struct fuse_in_
header));
  printf("INIT: %u.%u\n", init_arg->major, init_arg->minor);

  // ❸ init reply
  struct fuse_init_out outarg;
  memset(&outarg, 0, sizeof(outarg));
  outarg.major = FUSE_KERNEL_VERSION;
  outarg.minor = FUSE_KERNEL_MINOR_VERSION;
  outarg.max_write = outarg.max_readahead = 0x20000;
  //outarg.flags = FUSE_WRITEBACK_CACHE;
  //outarg.flags = FUSE_BIG_WRITES;
  reply(fd, in->unique, 0, &outarg, sizeof(outarg));

  while(1) {
    n = read(fd, &buf, SIZE);
    in = (struct fuse_in_header*)&buf;
    printf("unique: %llu, opcode: %i, nodeid: %lu, insize: %zu, pid: %u\n",
           (unsigned long long) in->unique, in->opcode,
           (unsigned long) in->nodeid, n, in->pid);
    switch(in->opcode) {
    case FUSE_LOOKUP:
      {
```

```
        char *name = buf + sizeof(struct fuse_in_header);
        printf("lookup: %s\n", name);
        struct fuse_entry_out e;
        memset(&e, 0, sizeof(e));
        e.nodeid = e.attr.ino = 2;
        e.attr_valid = e.entry_valid = 1;
        e.attr.mode = S_IFREG | 0644;
        e.attr.nlink = 1;
        e.attr.size = strlen(DATA);
        reply(fd, in->unique, 0, &e, sizeof(e));
      }
      break;
    case FUSE_OPEN:
      {
        struct fuse_open_out o;
        memset(&o, 0, sizeof(o));
        reply(fd, in->unique, 0, &o, sizeof(o));
      }
      break;
    case FUSE_READ:
      {
        struct fuse_read_in *rin = (struct fuse_read_in*)((void*)buf + sizeof(struct fuse_in_➥
header));
        if(rin->offset < strlen(DATA)) {
          char *d = strdup(DATA);
          reply(fd, in->unique, 0, d + rin->offset, min(strlen(DATA)-rin->offset, rin->size));
        } else {
          reply(fd, in->unique, 0, NULL, 0);
        }
      }
      break;
    case FUSE_GETATTR:
    case FUSE_SETATTR:
      {
        struct fuse_attr_out a;
        memset(&a, 0, sizeof(a));
        a.attr_valid = 1;
        a.attr.mode = S_IFREG | 0644;
        a.attr.nlink = 1;
        a.attr.size = strlen(DATA);
        reply(fd, in->unique, 0, &a, sizeof(a));
      }
      break;
    case FUSE_WRITE:
      {
        struct fuse_write_in *w = (struct fuse_write_in*)((void*)buf + sizeof(struct fuse_in_➥
header));
        struct fuse_write_out wout;
        int offset = sizeof(struct fuse_in_header)+sizeof(struct fuse_write_in);
        buf[offset+w->size] = '\0'; buf[offset+10] = '\0';
        printf("write(%d): %s\n", w->size, buf+offset);
        memset(&wout, 0, sizeof(wout));
        wout.size = w->size;
        usleep(1000000UL * w->size / 131072);
        reply(fd, in->unique, 0, &wout, sizeof(wout));
      }
      break;
    case FUSE_RELEASE:
      reply(fd, in->unique, 0, NULL, 0);
      break;
    default:
      // その他の操作には実装されていないというエラーを返す
      reply(fd, in->unique, -ENOSYS, NULL, 0);
    }
  }

  close(fd);
  return 0;
}
```

います。今回のコードではとりあえずバージョン番号を表示しているだけでとくに使いません。INITメッセージを受け取ると、受け取ったことをfuse.koへと返信する必要があります。返信用に今回はreply関数を作っています。返信は受信のときと同様に、/dev/fuseを開いたファイルデスクリプタへの書き込みで行います。書き込まれるデータにも、読み込み時と同様にヘッダとしてstruct fuse_out_headerを必ず付けます。ここにはメッセージ長とエラー番号(成功の場合は0)とメッセージ番号を指定します。ヘッダのあとにメッセージの種類に対応したデータを付けます。INITの場合、struct fuse_init_outのデータを入力します。ここでフラグの設定をすることでプログラムが対応している拡張が有効となります。今回はバージョン情報と書き込みサイズとreadaheadのサイズだけを指定しておきます(**リスト1❸**)。

FUSE:各操作への対応

初期化が終われば、あとはループを行いfuse.koから/dev/fuseを通して読みとったリクエストに対応していくコードが続きます。ここではLOOKUP、OPEN、GETATTR、SETATTR、READ、WRITE、RELEASEだけに対応するコードを書き、ほかの操作については-ENOSYSをエラー番号に指定した返信を行い、実装されていないことをファイルシステム操作を行った側に通知しています。

各操作について見ていきましょう。LOOKUPは前述したようにファイルが存在するかどうかを調べる操作です。引数としてファイル名そのものがヘッダのあとに入っています。指定されたファイルのパーミッションなどの情報をstruct fuse_entry_outに設定して返信します。今回はどんな名前であろうと、inode番号が2の通常ファイルであるという現実的にはおかしな返答を行います。OPENはフラグとファイルハンドルの情報が入ったstruct fuse_release_inのデータを送ってきますが、今回はそれらを完全に無視してゼロ埋めstruct fuse_open_outを返しています。

READではstruct fuse_read_inにファイルハンドルや読み取り位置とサイズなどの情報が入ってきます。指定されたオフセットとサイズに合わせて、ファイルデータをヘッダのあとに付けて返信しています。GETATTRとSETATTRの場合はLOOKUPと同様にパーミッションなどの情報をstruct fuse_attr_outに設定して返信します。ノードIDが1の場合はディレクトリとして、それ以外の場合にはLOOKUP同様に通常ファイルとしての返信を行っています。本来はSETATTRの場合には、struct fuse_setattr_inに設定すべき属性情報が入ってくるのですが今回は完全に無視してしまいます。

WRITEではstruct fuse_write_inに書き込み位置やサイズなどのデータが入り、さらにそのあとに書き込みデータが入ってきます。書き込みを行い、struct fuse_write_outに書き込んだサイズを設定して返答します。今回の実装では、書き込まれたデータとサイズとを表示し、それ以外は何もせずにあたかもすべて書き込まれたかのような返答を行います。さらに、今回はusleep()を入れて書き込みが呼び出されるたびに128KBあたり1秒かかるような設定にしています。RELEASEはclose()に対応する操作です。単純に成功の返信を行います。

FUSE:実行例

このプログラムをコンパイルしroot権限で実行することでプログラム中に記載されているディレクトリにこのプログラムによるファイルシステムがmountされます。この後**図2**のようにlsやcatといったコマンドを実行してみると、プログラムから**図3**のような出力が得られます。プログラムで設定したパーミッションやファイルサイズといった情報が出ていること、ファイルの中身が読み取れることがわかります。さらに

ディレクトリを"ls"すると、実装していないオペコード=27(READDIR)のメッセージが来ていることがわかります。ここで-ENOSYSを返しているので、"ls"も"Function not implemented"というエラーを出力しています。

FUSE：writebackサポート

さて、ここで適当なサイズのファイルを作り、cpコマンドを使ってファイルをこのファイルシステム下にコピーすると4KBごとにWRITEが呼び出されます。これまでFUSEの書き込みはwritethroughに実現されていました。すなわちWRITEのたびに、これが完了しFUSEのプログラムが返信するまでwrite()を行った側は待たされてしまいます。つまり、今回WRITEのたびにスリープしているので、たとえば1,280KBのデータをコピーするのに10秒間cpが終了しないことになります。今回はてきとうなFUSE実装ですが、現実的にもネットワークを使ったFUSE実装の場合にも同様の問題が起きることがあります。

この問題に対処する1つ目の方法はより大きなバッファを使うことです。INIT時にFUSE_BIG_WRITESをフラグに指定すると、4KB以上最大128KBのバッファでWRITEが行われるようになります。これによってWRITEが呼び出される回数は減少します。もう1つの方法がLinux 3.15で実装されたwritebackキャッシュを使う方法です。INIT時にFUSE_WRITE

▼図2　FUSE上での操作例

```
$ sudo ls -l fusemount/{foo,bar}
-rw-r--r-- 1 root root 30 Jan  1  1970 fusemount/bar
-rw-r--r-- 1 root root 30 Jan  1  1970 fusemount/foo
$ sudo cat fusemount/foo
test data
test data
test data
$ sudo ls -l fusemount
ls: cannot open directory fusemount: Function not implemented
```

▼図3　プログラムの出力

```
$ gcc fuse.c; sudo ./a.out; sudo umount fusemount
(INITの部分)
unique: 1, opcode: 26, nodeid: 0, insize: 56, pid: 0
INIT: 7.23
(ls -l fusemount/fooに対応)
unique: 2, opcode: 1, nodeid: 1, insize: 44, pid: 3730
lookup: foo
unique: 3, opcode: 22, nodeid: 2, insize: 72, pid: 3730
(ls -l fusemount/barに対応)
unique: 4, opcode: 1, nodeid: 1, insize: 44, pid: 3730
lookup: bar
(cat fusemount/fooに対応)
unique: 5, opcode: 1, nodeid: 1, insize: 44, pid: 3774
lookup: foo
unique: 6, opcode: 14, nodeid: 2, insize: 48, pid: 3774
unique: 7, opcode: 15, nodeid: 2, insize: 80, pid: 3774
unique: 8, opcode: 3, nodeid: 2, insize: 56, pid: 3774
unique: 9, opcode: 25, nodeid: 2, insize: 64, pid: 3774
unique: 10, opcode: 18, nodeid: 2, insize: 64, pid: 0
(ls -l fusemountに対応)
unique: 11, opcode: 3, nodeid: 1, insize: 56, pid: 3986
unique: 12, opcode: 27, nodeid: 1, insize: 48, pid: 3986
unique: 13, opcode: 27, nodeid: 1, insize: 48, pid: 3986
```

BACK_CACHEをフラグに設定することで、デフォルトのwritethroughのキャッシュポリシーからwritebackのキャッシュポリシーへ切り替わります。これでwriteを行っている側が待たされることはなくなり、速くなることが期待されます。

では、writebackとwritethroughとで1,280KBのファイルをコピーしてみましょう(**図4**)。writethroughの場合は10回の呼び出しが行われ毎回1秒待機するのでどのcpも10秒かかっています。一方で、writebackの場合にはwriteが待たされるかどうかはシステムのメモリ管理ルーチンが判断することになり、ばらつきが大きくなっています。また、FUSEプログラムのwriteでの出力を見ていると、writethroughでは書き込みサイズが毎回128KBであったのに対して、より小さいサイズで呼び出されていることもあることがわかります。

今回の実験ではいまひとつの結果になっていますが、FUSEでの書き込みパフォーマンス向上が期待されます。sshfsなどにも実装されるといいですね。

サスペンドからの復帰の高速化

Linux 3.15での最後の機能紹介はサスペンドからの復帰の高速化です。これまでサスペンドからの復帰時には、ディスクに復帰コマンドを送り、ディスクが復帰して返答してくるまで待ち続けていました。待っている間はカーネルは何も作業をしていないので、この時間は無駄になっています。そこでLinux 3.15ではコマンドを送ってから即時にほかの作業を行い、復帰後の作業は返答が来てから行われるように変更されました。この変更によってサスペンドからの復帰が速くなっています。01.orgにパッチ前、パッチ後の復帰を比較するグラフがあるのでぜひそれも見てみてください[注2]。

まとめ

今回はLinux 3.15の最後の機能紹介として、FUSEとサスペンドからの復帰の高速化についての紹介を行いました。次回はLinux 3.16とLinux 3.17について扱います。SD

注2) https://01.org/suspendresume/blogs/tebrandt/2013/hard-disk-resume-optimization-simpler-approach

▼図4　writebackの実験

```
(writethroughの場合)
$ echo 3 |sudo tee /proc/sys/vm/drop_caches; for x in `seq 1 5`;do time sudo cp testdata2 ↵
fusemount2/foo;done
3
sudo cp testdata2 fusemount2/foo  0.00s user 0.01s system 0% cpu 10.044 total
sudo cp testdata2 fusemount2/foo  0.00s user 0.01s system 0% cpu 10.032 total
sudo cp testdata2 fusemount2/foo  0.00s user 0.00s system 0% cpu 10.062 total
sudo cp testdata2 fusemount2/foo  0.00s user 0.12s system 1% cpu 10.311 total
sudo cp testdata2 fusemount2/foo  0.00s user 0.01s system 0% cpu 10.227 total

(writebackの場合)
$ echo 3 |sudo tee /proc/sys/vm/drop_caches; for x in `seq 1 5`;do time sudo cp testdata2 ↵
fusemount2/foo;done
3
sudo cp testdata2 fusemount2/foo  0.00s user 0.01s system 0% cpu 9.704 total
sudo cp testdata2 fusemount2/foo  0.00s user 0.01s system 0% cpu 14.071 total
sudo cp testdata2 fusemount2/foo  0.00s user 0.01s system 0% cpu 9.013 total
sudo cp testdata2 fusemount2/foo  0.00s user 0.01s system 0% cpu 11.003 total
sudo cp testdata2 fusemount2/foo  0.00s user 0.01s system 0% cpu 9.062 total
```

バックナンバーのお知らせ　BACKNUMBER

バックナンバー好評発売中！常備取り扱い店もしくはWebより購入いただけます

本誌バックナンバー（紙版）はお近くの書店に注文されるか、下記のバックナンバー常備取り扱い店にてお求めいただけます。また、「Fujisan.co.jp」（http://www.fujisan.co.jp/sd）や、e-hon（http://www.e-hon.ne.jp）にて、Webから注文し、ご自宅やお近くの書店へお届けするサービスもございます。

2014年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［Serf編］
特別定価（本体1,300円+税）

2014年8月号

第1特集
システムログからWebやDB、ビッグデータの基礎まで
ログを読む技術

第2特集
forkを通して考える・試す・コードを読む
Linuxカーネルのしくみを探る

一般記事
・OpenSSLの脆弱性"Heartbleed"の教訓（後編）
・使ってみよう! tcpdump
定価（本体1,220円+税）

2014年7月号

第1特集
［多機能］［高速処理］［高負荷対策］
そろそろNginx移行を考えているあなたへ

第2特集
知っているようで知らない
DHCPサーバの教科書

一般記事
・OpenSSLの脆弱性"Heartbleed"の教訓（前編）
・Webアプリのパフォーマンス改善（最終回）ほか
定価（本体1,220円+税）

2014年6月号

第1特集
設定ファイルの読み方・書き方でわかる
Linuxのしくみ

第2特集
Windows XPからの乗り換えにいかが?
Ubuntu 14.04 "Trusty Tahr"過酷な環境でも信頼できるLTSバージョン

一般記事
・Google Glassアプリ開発事情
・OpenTSDB（前編）ほか
定価（本体1,220円+税）

2014年5月号

第1特集
ネットワーク・ビギナー向け基礎講座
「ポート」と「ソケット」がわかればTCP/IPネットワークがわかる!

第2特集
UNIX必須コマンドトレーニング
rmコマンドからcadaverまで基本を押さえる

一般記事
・Rettyのサービス拡大を支えた「たたき上げ」DevOps
・Webアプリのパフォーマンス改善 ほか
定価（本体1,220円+税）

2014年4月号

第1特集
<Java/JavaScript/PHP>言語別で考える
なぜMVCモデルは誤解されるのか?

第2特集
ネットワークエンジニア養成
ロードバランサの教科書

一般記事
・さらに踏み込む、Mac OS Xと仮想デスクトップ #2
・SIMのしくみ
定価（本体1,219円+税）

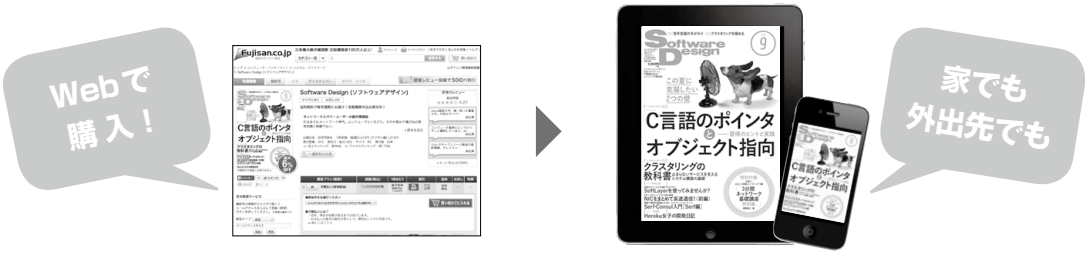
Software Design　バックナンバー常備取り扱い店							
北海道	札幌市中央区	MARUZEN & ジュンク堂書店　札幌店	011-223-1911	神奈川県	川崎市高津区	文教堂書店　溝の口本店	044-812-0063
	札幌市中央区	紀伊國屋書店　札幌本店	011-231-2131	静岡県	静岡市葵区	戸田書店　静岡本店	054-205-6111
東京都	豊島区	ジュンク堂書店　池袋本店	03-5956-6111	愛知県	名古屋市中区	三洋堂書店　上前津店	052-251-8334
	新宿区	紀伊國屋書店　新宿本店	03-3354-0131	大阪府	大阪市北区	ジュンク堂書店　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。



October 2014

NO.36

Monthly News from jus

Japan UNIX Society

日本UNIXユーザ会　http://www.jus.or.jp/
波田野 裕一　HATANO Hirokazu　tcsh@tcsh.csh.sh
法林 浩之　HOURIN Hiroyuki　hourin@suplex.gr.jp

インターネットの今後について考えた２連戦

今回は、7月に名古屋と福岡で行った研究会の報告をお送りします。

名古屋大会

■IPv6時代のWebアプリケーション＆プラットフォーム2014
【講師】波田野 裕一（日本UNIXユーザ会）
【日時】2014年7月4日（金）16:25～17:10
【会場】名古屋国際センター4F展示室

名古屋大会は筆者（波田野）が講師となり、Webアプリケーション（以下、Webアプリ）のIPv6対応の基本的なポイント、クラウドやVPSなどでのIPv6利用について解説しました。参加者は42名でした。

はじめに、IPv6を取り巻く環境について説明をしました。2011年前半に、アジア太平洋地域のIPv4アドレスの在庫が枯渇してから3年が経過し、通信事業者やISP各社のIPv6対応サービスが拡大してきています。一方で今年の6月に、北アメリカのARINと南アメリカのLACNICが相次いでIPv4アドレスの事実上の枯渇宣言をしており、IPv4アドレスは事実上枯渇したと言って良い状況です。

このようなIPv6環境の普及およびIPv4アドレスの枯渇により、「IPv4のみ」の運用から「IPv4とIPv6の共存期」に移行する時期にきています。Webアプリについては、上記の状況をふまえて「IPv6を意識しなければいけない個所」「IPv4との違い」の2点を意識していく必要があります。

続いて、IPv6に対応したWebアプリ開発に必要な基礎的事項として、4つのポイントを解説しました。

最初のポイントは、アプリケーション開発におけるIPv6対応です。IPv4とIPv6の共存期間は当面続くと予想されており、その間はIPv4とIPv6の双方の環境で動作するシステムが求められます。単一のソースコードで両方の通信環境に対応しなければなりません。また、IPv4かIPv6かを意識させないために、FQDN（Fully Qualified Domain Name）でサーバを識別し、IPアドレスでホストやユーザを識別しないことが重要となってきます。

2つ目のポイントは、プログラミング言語と実行環境のIPv6対応です。とくに名前解決機構（正引き／逆引き）がIPv6アドレスを適切に扱えること、IPv6で通信できることが重要になります。このあたりはプロダクトによってサポート状況に差異があるのが実情です。開発するアプリケーションの提供機能を考慮して、利用するプログラミング言語やプロダクトごとに個別に判断する必要があるでしょう。

3つ目のポイントは、通信処理のIPv6対応です。IPv4/IPv6両方で通信するということは、そのいずれかで通信できない状況を想定する必要があります。また、クライアントとサーバがともに複数のIPアドレスを持つことも意味します。それぞれどのアドレスが使用されるかはWebアプリ側では予測できないため、特定のアドレスを想定したシステムを構成するべきではありません。

4つ目のポイントは、IPv6アドレスをデータとして扱う場合に、表記のゆらぎを防止するために、完全表記もしくはRFC5952（推奨表記）を利用するということです。これらが問題になるケースとして、デー

タベースへの格納、ログ出力、Webフォームへの IP アドレス入力、IPv6アドレスの検索や整列の場合などが挙げられます。

最後に、IPv6対応プラットフォームの国内での状況について解説しました。2013年半ばからVPS（Virtual Private Server）でIPv6ネイティブの接続を提供するサービスが増えてきており、個人でもIPv6対応サーバを構築することが容易になってきていることをお伝えしました。

参加者42名のうち、日常的にIPv6を利用している方は1名だけで、まだまだこれからの技術だと認識している方が多かったようです。一方で熱心に聴講している方が多く、IPv6時代が近いことを感じていただけたかな、というのが講演後の印象でした。

福岡大会

■APRICOT-APANにみる世界のインターネット事情
【講師】谷崎 文義（NTT西日本）、法林 浩之（日本UNIXユーザ会）
【日時】2014年7月12日（土）15:00～15:45
【会場】福岡工業大学 B棟 B45教室

福岡大会はFuture Syncというイベントの中で開催しました。内容は、来年早春に福岡で開催されるAPRICOT-APANの紹介と最近のインターネット事情の解説です。講師はAPRICOT-APANの会場ネットワーク担当の谷崎さんと筆者（法林）の2名でした。

講演は、APRICOT-APANの開催概要から説明しました。APRICOTもAPANもアジア太平洋地域のインターネット技術を扱う国際会議です。APRICOTがおもに管理／運用分野の集まりであるのに対して、APANは学術研究分野の集まりです。両イベントともそれぞれ年1～2回の会合を持っていますが、同時に開催されるのは4年ぶりで、さらに日本での同時開催は初となります。

次に、世界のインターネットがどのように管理されているかのしくみを説明し、続いてAPRICOT-APAN 2015のスケジュールを紹介しました。2月24日（火）～28日（土）はAPRICOT Workshopsで、ハンズオン形式で学習するセッションが行われます。翌週の3月2日（月）～5日（木）がAPRICOT 2015とAPAN 39で、これがいわば本編です。基調講演、カンファレンス、チュートリアルなど50以上のセッションが行われ、両イベント合わせて1000人程度が参加します。最終日の3月6日（金）はAPNICが抱える課題を話し合うAPNIC Member Meetingで、これのみ参加費が無料です。

APRICOT Workshopsで行うハンズオンのテーマは、BGP、MPLS、ネットワーク管理などが多いです。一方、カンファレンスではセキュリティやIPv6への移行などがよく出る話題ですが、最近はそれらに加えてインターネットガバナンスに関連したセッションが多く実施されています。

インターネットのルール策定に関する議論では、参加資格を問わないオープン性、参加者の立案によるボトムアップ、意思決定の過程が公開される透明性が尊重されてきました。しかし、これは電話を中心とした電気通信の世界とは正反対の考え方であり、インターネットにも厳格な管理や秩序を求める人たちが出てきています。このような状況が、インターネットの意思決定のしくみを議論するガバナンス関連の話題がさかんになっている理由であり、今後のインターネットの方向性を考えるうえで避けては通れない重要な話題であることをとくに強調しました。

最後にAPRICOT-APANの参加方法と、国内で行われる関連イベントとしてInternet Week、JPNICオープンポリシーミーティング、JANOGなどを紹介して講演を締めくくりました。

今回、Future Syncに参加したことで、オープンソースカンファレンスとは異なる客層にも話を聴いてもらえたのはとても有意義でした。とくにインターネットガバナンスの話題は、ネットワーク管理／運用分野ではよく知られていますが、ほかの分野の人はおそらく初めて聞く話だったと予想されます。今後もこのような機会を見つけて啓蒙活動を行っていきたいと思います。SD

Hack For Japan
エンジニアだからこそできる復興への一歩

● Hack For Japan スタッフ
及川 卓也　Takuya Oikawa
Twitter @takoratta
鎌田 篤慎　KAMATA Shigenori
Twitter @4niruddha

第34回 エフサミ2014レポート

“東日本大震災に対し、自分たちの開発スキルを役立てたい”というエンジニアの声をもとに発足された「Hack For Japan」。今回は福島県で行われたITイベント「エフサミ」の報告です。

福島のITコミュニティが一丸となって開催

今回は本連載第14回（2013年2月号）でも紹介させていただいた「ITスキルアップコミュニティ エフスタ!!注1」が、2014年7月12日から13日にかけて開催した福島県内最大のITイベント「エフサミ注2」のレポートをお送りします。エフスタ!!は原発問題を抱える福島県は郡山から、ITによって福島県を盛り上げ、県外の人達にも楽しんでもらいつつ福島の今を知ってもらうための活動を精力的に続けています。

福島県は明治9年に3つの県が統合されてできた大きな県で、浜通り、中通り、会津といった3つの縦断された地域で成り立っています。そうした背景もあり、3つの地域に存在するITコミュニティも震災以前はそれほど連携はありませんでした。それが震災を契機に徐々につながりはじめ、連載第31回（2014年7月号）で紹介した震災後3年を振り返ったイベントのとおり、会津、中通り、浜通りの開発者達がつながり、協力し合い、今回の「エフサミ」の開催となりました。

イベントの様子

エフサミは土曜日と日曜日の2日間に分けて、いくつかのトラック構成でさまざまなセッションが催されました（**写真1**）。

Hack For Japanスタッフでもある及川卓也による基調講演でスタートした初日は、5つのトラック＋子供向けのワークショップトラックで構成されました。

「Fukushima」トラックはこれからのITを知ってもらうことをテーマとし、後述の「Fab蔵（くら）注3」の紹介や、Kinect2に代表されるモーションセンサーの活用、福島県でのセンサー活用の紹介がメインとなるセッション構成でした。

「Koriyama」はITの面白さを知り、強い興味を持ってもらうためのトラックで、セキュリティの裏側を詳しく解説するものから、Webや感情のデザイン、ゲームを遊ぶ側から作る側へといった中身の濃い話が中心でした。

「Aizu」トラックは福島県の地元企業や学生、コミュニティメンバーによる発表が中心で、福島からIT活動を発信することのおもしろさ、やりがいを伝えてくれました。

「Iwaki」トラックの目玉である自由参加なライトニングトークでは、エフスタメンバーのほかにも精力的に活動している熱い人達の話が聞け、「Kura」トラックではAWSやAzureといったプラットフォームのエヴァンジェリストが実際に手取り足取り教えてくれるハンズオンスタイルで、来場者が最新のクラウド環境での開発を学べました。

2日目はYahoo!やGoogle、Microsoftといった企業による講演や「福島のITで日本を元気に！」をテーマ

注1　http://www.efsta.com/
注2　http://summit.efsta.com/2014/
注3　http://www.fabkura.org/

◆写真1　会場の様子

にしたアイデアソン、また、GoogleやMicrosoftによるハンズオンのワークショップが中心となりました。

両日共に開催された「Mirai」トラックでは、未来の希望である子供達にITの面白さを知ってもらうために、デジタル絵本のワークショップや描いた絵をゲームに登場させたり、モーションセンサーを使った遊びを行うセッションがあり、子供達にも大好評でした（**写真2**）。そうした盛況な様子が話題となり、テレビ局からの取材も入りました。子供達の様子や各セッションの様子などとあわせ、代表の大久保仁さんへのインタビューなどが収録され、イベント開催中に放送されました。

◆写真2　Miraiトラックで遊ぶ子供達

また、今回のイベントでは会場でコーヒーが振る舞われたり（**写真3**）、似顔絵コーナーを設ける（**写真4**）など、幅広い層の参加者が楽しめる空間作りがされていました。ほかにも登壇者の気さくな一面を知ってもらおうという企画のラジオ番組風対談も実施され、オンラインで生放送されるなど、勉強会を中心としたITコミュニティのイベントの中でも、これほどまで細部にわたり企画され、盛りだくさんで楽しめるイベントはなかなかないと思います。

◆写真3　屋上のカフェの様子

◆写真4　似顔絵コーナーの様子

セッションの紹介

行われたセッションの主なものを紹介します。

基調講演「ITから福島、東北、日本の未来を考える」

基調講演ではHack For Japanスタッフの及川がインターネットの誕生から今までを振り返りました。TCP/IPのデザインの根幹にあるSlow Startを紹介し、私たち一般の考え方にも“Think Big, Start Small, Scale Fast”という考えを適用しようと呼びかけました。

インターネットの世界で働くことについて

2日目の大手インターネット企業が講演する「Fukushima」トラックでは、Hack For Japanスタッフでもあり、普段はYahoo! JAPANに勤務している鎌田篤慎から、インターネットの世界で働くことについての話を中心に、Yahoo! JAPANの紹介もしました。

今回のエフサミではITを学ぶ学生の参加者が多かったことから、学生達にIT業界で働くことに興味を持ってもらうように心がけ、鎌田自身のインターネット企業に転職するまでの経緯から、未知の領域に挑むインターネット企業の仕事内容をさまざまな角度から比較をしてみました。そして世界のトレンドを紹介し、未来を担う学生にIT業界で働くことの意味と、さらなる可能性を秘めた仕事である

ことを繰り返し伝えました。

福島や復興に関係するもの

エフサミは、福島のコミュニティによって福島で開催されたイベントということもあり、福島や復興に関連するセッションがいくつか行われました。

●Fab蔵のすべて

Hack For JapanスタッフでもあるGClueの佐々木陽さんは現在、会津若松市に「Fab蔵」というInternet of Things（IoT）の時代に向けたモノづくりを支援する施設を立ち上げています。このセッションでは、IoTの現状と今後の説明やFab蔵の紹介が行われました。

会津若松ということもあり、Fab蔵は使われなくなった本物の蔵を再利用しており、紹介された写真で見るだけですが、非常に雰囲気があります。このようなユニークな施設にレーザーカッターや3Dプリンタなど必要な機器が常設されており、すぐにアイデアを試してみることができます。

Fab蔵では定期的なワークショップを行い、コミュニティとしての知識の底上げを図るとともに、ハッカソンや長めのハッカソン（Longハッカソンと名づけています）、ハードウェアを中心としたイベントであるガジェットソンなどを企画しています。市外からの参加も歓迎のようですので、興味のある人はぜひとも参加を検討してみてください。

●福島のITで日本を元気に！ in 浜通り

Aizuトラックでは、浜通り、中通り、会津の3地域から発表がありました。

まず、浜通りからは、いわき情報技術研究会[注4]理事長の高山文雄さんと南相馬ITコンソーシアム監事である南相馬市議の但野謙介さんからお話がありました。ご存じのとおり、原発事故の影響もあり、厳しい状況にある浜通りです。たとえば、いわき明星大学では科学技術学部の募集を停止し、今後は2学部体制（薬学部と教養学部）となるなど、事態は深刻です。

いわき情報技術研究会は震災発生直後の2011年5月に発足し、月1回の研究会を開催しています。また、南相馬ITコンソーシアムも地域の産業復興を担う人材の育成を目指して発足しました。実績として、南相馬発のアプリをすでにいくつもリリースしています。

いずれも復旧から復興に至る中で、人口減少など地方の課題に取り組む手段としてのITに活路を見出しています。

●福島のITで日本を元気に！ in 中通り

Aizuトラックの2つ目はエフスタ!!SENDAIの八巻雄哉さんの司会で、㈱エフコムの斉藤広二さん、林容崇さん、㈱福島情報処理センターの大和田洋介さん、㈱会津ラボの稲澤麻弓さん、㈱ネクストの藤原裕也氏さん、Studioisaacの清水俊之介さんという地方と首都圏のエンジニアがそれぞれの立場でのエンジニアライフを語る、ちょっと変わったセッションでした。

地方と都市部という違いだけでなく、企業とフリーランスなどの異なる立場による考えの違いもあり、会場含めて有意義な議論がされました。多くの学生も参加していましたが、何をやりたいのかビジョンを明確にすべきという登壇者からのメッセージは彼らにも響いたのではないでしょうか。

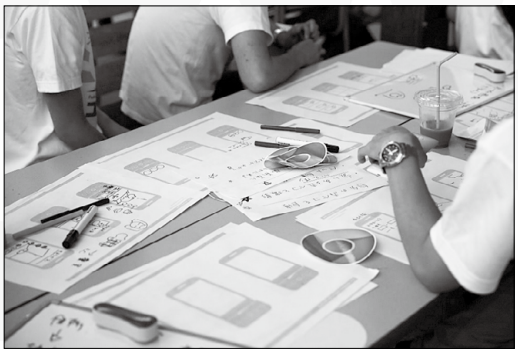
●福島のITで日本を元気に！ in 会津

CODE for AIZU[注5]は会津地域のIT企業・団体・行政の有志や学生などが中心となり、地域の抱えるさまざまな課題を解決する方法を考え、アプリケーションやWebサービスとして開発・提供する草の根的なコミュニティです。このセッションでは、CODE for AIZUで活躍する3名が登壇しました。

藤井靖史さんからは自らの持つスキルを使って地域に貢献できるCODE for AIZUについての説明がありました。活動内容として、市役所やベンチャー企業との連携、Code for Japan[注6]や他のCode for X

注4　https://sites.google.com/site/iwakiit/home
注5　http://aizu.io/
注6　http://code4japan.org/

◆写真5　青空ソンの様子



との地域連携などの紹介がされました。

前田諭志さんからはオープンデータによる地域エコシステム構築が紹介されました。オープンデータとして提供されるデータはData for Citizen[注7]で整備されているそうです。

徳納弘和さんからは実際にオープンデータを使ったデモがを紹介されました。開発の際に、課題発見、要求把握、オープンデータとしてのデータの存在を検討することが重要と話されていました。

●ITで地方を元気に！ コミュニティリーダートーク！

「ITで地方を元気に！」は東北と北海道のコミュニティリーダーによるトークセッションです。登壇したのは、北海道を中心に活動するLOCAL[注8]の八巻正行さん、仙台を中心に活動する東北デベロッパーズコミュニティ（TDC）[注9]の小泉勝志郎さん、この連載でも何度か取り上げたことのあるイトナブ石巻[注10]の古山隆幸さん、そしてエフスタ代表の大久保仁さんとエフスタ東京の代表である影山哲也さんです。ファシリテータであるエフスタの浅井渉さんからの質問に登壇者が答える形でセッションは進みました。笑いを交えて、各コミュニティリーダーの思いが伝わる大変熱いセッションでした。

●アイデアソン

2日目には、会場外でアイデアソンも開催されました。最初は小雨がぱらつくあいにくの空模様だったのですが、午後からは雨も小降りになり、屋外ということもあって、和気あいあいとした雰囲気で開催されました。

◆写真6　エフサミ終了後の集合写真

参加者は4～6人のチームに分かれ、スマートフォンアプリケーションを考えました。前半でアイデアを議論し、後半はペーパープロトタイプを行います（**写真5**）。最初は知らない人同士であっても、議論をしているうちに立派なチームメートになっていく。そんなことを体験できるアイデアソンでした。次回はこのアイデアソンの名前である「青空ソン」のとおり、青空の下で行ってみたいと、そんなことを思わせる一日となりました。

まとめ

以上、エフスタが行ったエフサミ2014の2日間の模様をお伝えしました。Hack For Japanスタッフがコンタクトをした際には、まだ小規模で、あくまでも中通りの勉強会コミュニティであったエフスタが、福島の3地域のみならず東京をはじめとする関東のエンジニアや東北地方のエンジニアを巻き込むまでの規模になったのも、エフスタスタッフをはじめとする関係者の努力の賜物でしょう（**写真6**）。

とくに、今回のエフサミは学生が多く参加していたり、子供が楽しめる仕掛けがされているなど、首都圏のイベントでも参考にすべき点が多くあったように思います。Hack For Japanとして、これからも継続して応援をしていきたいと考えています。

エフスタスタッフの方々、お疲れ様でした。SD

注7　http://www.data4citizen.jp/
注8　http://www.local.or.jp/
注9　http://tohoku-dev.jp/
注10　http://itnav.jp/

新連載！ もう表計算ソフトに頼らない

迷えるマネージャのための プロジェクト管理ツール再入門

第1回 Excelでのプロジェクト管理からの脱却

Software Design編集部

プロジェクトの工程管理や課題管理において、古来(?)から使われ続けているのがExcelです。ただExcelでのプロジェクト管理にはさまざまな課題があり、管理効率という意味ではけっしてベストな選択肢ではありません。この連載では、こうした課題を解決できる「プロジェクト管理ツール」について解説していきます。第1回は、アトラシアンが開発・提供し、リックソフトが販売・サポートするJIRA(図1)を紹介します。

どのファイルが最新? Excelによるプロジェクト管理の問題点

ソフトウェア開発などのプロジェクトをスムーズに進めるためには、作業内容やスケジュール、解決すべき課題などを適切に管理することが重要になります。この目的を達成するために、従来使われてきたのがMicrosoftの表計算ソフトであるExcelです。たとえば作業内容とそのスケジュールをまとめる工程管理では、左側の列に作業内容を記載し、その右側に横棒で作業の進捗状況を表すガントチャートを作成するケースが多いでしょう。対応すべき課題は、横軸に「項番」「課題の内容」「優先度」「対応状況」といった項目を並べ、縦軸に個々の課題を並べる表を作成して管理するのが一般的です。

Excelは極めて柔軟性の高い表計算ソフトであり、上記のようなガントチャートや課題管理表の作成にも十分に対応できます。しかし、そもそもExcelはプロジェクト管理に特化したツールではなく、管理作業を効率化するという観点ではけっして使い勝手がよいとは言えません。

たとえば複数のメンバがいるプロジェクトにおいて、Excelで作成したガントチャートのファイルをサーバ上で共有し、それぞれのメンバがそのファイルを更新することで進捗状況を管理するとしましょう。Excelでは、1人がファイルを開いていると、ほかの人は内容を更新できず、ファイルを開いている人が作業を終えるまで待たなければなりません。とくにチーム規模が拡大すると、いつまでたっても更新できないといった状況が発生してしまいます。

このような事態を避けるため、プロジェクト内のチームごとにファイルを分け、適当なタイミングでマージするという方法が採られることもあります。ただ、この方法はそもそもめんどうなうえ、操作を誤ると一部の内容が先祖返りするといったこ

▼図1 アトラシアンが開発したプロジェクト管理ツールである「JIRA」。Webアプリケーションとして動作し、各ユーザはWebブラウザを使ってアクセスする

とも起こりえます。

同じ内容のファイルが多数存在し、どれが最新か判断できないということもよくあります。たとえば、ファイルサーバ上に「課題管理表.xlsx」「課題管理表-140901.xlsx」「課題管理表-最新.xlsx」「課題管理表-コピー.xlsx」といったファイルが並んでいるような状況です。これでは、どのファイルが最新で、新たな課題をどれに登録すべきか頭を悩ませることになるでしょう。

マネージャの負担を大幅に軽減するプロジェクト管理ツール

このような課題を解決するために、多くのプロジェクトで利用され始めているのが「プロジェクト管理ツール」です。複数のユーザが同時に利用できるサーバアプリケーションで、さまざまな情報を一元的に管理することにより、Excelベースでのプロジェクト管理の問題を解決し、さらに多くのメリットをプロジェクトメンバにもたらします。

それでは、プロジェクト管理ツールを利用することで、実際にどのようなメリットがあるのでしょうか。提供されている機能はプロダクトによって異なりますが、多くのプロジェクト管理ツールで共通しているのがタスクや課題を「チケット」と呼ばれる単位で管理するしくみです。

このチケットの使い方を具体的に見ていきましょう。作業すべきタスク、あるいは解決すべき課題が発生したら、まずチケットを作成し、タスクや課題の内容と優先度、担当者、開始日や期日などを設定します。さらに各々のチケットにはステータスがあり、作業開始時には「作業中」、終了すれば「作業完了」などとステータスを変更します。このチケットの一覧を見れば、現状で発生しているタスクや課題とその担当者、現在の状況などを把握できるというわけです。

さらに多くのプロジェクト管理ツールでは、タスクや課題の管理だけでなく、チケットごとのコメント欄やファイル共有スペースなど、メンバ間の情報共有ツールとして使うための機能も提供しています。

メンバ間のコミュニケーションにプロジェクト管理ツールを利用するメリットとして、コメント欄に投稿した内容、あるいはアップロードしたファイルをプロジェクトメンバ全員で共有できることが挙げられるでしょう。多くのプロジェクトでは、このようなコミュニケーションのためにメーリングリストが使われていますが、プロジェクト単位のメーリングリストでは細かなタスクや課題を議論するには不向きで、個人対個人でのやりとりになりがちです。そうすると情報が分散することになり、たとえば新たに参加したメンバにこれまでの経緯を説明するという場面で、過去のメールを洗い出して資料を作るなどといった苦労をすることになりかねません。その点、プロジェクト管理ツール上でコミュニケーションを行っていれば、誰でも議論の内容を把握でき、あとから参加したメンバもこれまでの経緯を追うことができます。

ステータスを見ることでプロジェクトメンバの作業状況をチェックできることもメリットです。もちろん一般的な工程管理表でも各メンバの作業内容を捕捉できますが、プロジェクトが複雑になったり、メンバが増えたりすると更新頻度が週1回程度になり、現状を正しく把握することが困難になります。プロジェクト管理ツールを使えば、各メンバがチケットのステータスを適切に更新することで、今何をやっているのか、これまでどの程度のチケットを処理しているのか、たまっているチケットはどれくらいあるのかなどをいつでも確認できるというわけです。

アトラシアンのJIRAでプロジェクト管理ツールを体験

プロジェクト管理ツールは現在数多く出回っており、アトラシアンの「JIRA」やオープンソースの「Redmine」、ヌーラボがサービスとして提供している「Backlog」などがあります。中でも、オープンソースのApacheプロジェクトなどで活用されていることからとくに注目されている

▼図2　実際にJIRAでプロジェクトを作成しているところ。複数のプロジェクトタイプがあり、目的に応じて選択できる

▼図3　プロジェクト配下にバージョンを作成した。作成したタスクや課題は、これらのバージョンに紐付けて管理できる

のがJIRAです。実際、シスコシステムズやsalesforce.com、Adobe、LinkedInなどさまざまな企業で活用されているほか、日本においてもANAシステムズやインターネットイニシアティブ(IIJ)などがJIRAを採用しています。以降では、このJIRAを利用して、プロジェクト管理ツールの使い勝手をチェックしていきましょう。

JIRAでプロジェクト管理を行うには、まず「プロジェクト」を作成します。タスクや課題は、このプロジェクトにチケットを紐付けて管理するという形です。JIRAではいくつかのプロジェクトタイプ(テンプレート)があり、「簡単な課題トラッキング」「プロジェクト管理」「アジャイル かんばん」「アジャイル スクラム」などがあらかじめ用意されています(**図2**)。

このプロジェクトの配下には、さらに「コンポーネント」と「バージョン」という課題を分類するための項目があります(**図3**)。たとえばコンポーネントであれば「サーバ」「データベース」「ユーザインターフェース」など、バージョンは「1.0」「2.0」といった形で設定します。これで課題を作成する際、どのコンポーネントの課題なのか、どのバージョンで対応するのかを設定できるというわけです。

ここまで設定したら、実際にタスクや課題を登録してみましょう。画面上部にある「作成」をクリックすると、課題の作成画面に遷移します(**図4**)。ここでプロジェクトと課題タイプを選んで進むと、タスクや課題の詳細を作成する画面が現れます。ここでタスクの内容や優先度、期限、担当者などを指定します。

この画面には、先ほど設定したコンポーネントやバージョンを指定する項目もあります。バージョンは「影響バージョン」と「修正バージョン」の2つに別れており、たとえばバグ修正をタスクとして登録する場合であれば、そのバグが影響するバージョンを「影響バージョン」に、そのバグを修正するバージョンを「修正バージョン」として登録すればよいでしょう。

登録した課題のページを開くと、設定した内容を確認できます(**図5**)。上部にある「処理開始」ボタンを押すとステータスが「進行中」に切り替わり、「完了」をクリックすればステータスも「完了」になります。「コメント」ボタンもあり、そのプロジェクトのメンバがその課題に対するメッセージを投稿することも可能です。

ダッシュボードや高度な検索など多彩な機能を提供

JIRAの大きな特長として、こうして登録したタスクや課題をさまざまな観点から管理できることが挙げられます。JIRAへのログイン直後に表示される「ダッシュボード」には「ガジェット」を追加でき、課題をさまざまな形で表示できます。標準で用意されているガジェットには、作成済みの課題と解決済みの課題の数をグラフ表示する「作成済み vs 解決済みグラフ」や、課題を解決するのにかかった時間をグラフ化する

「解決時間」、担当者として自分が割り当てられている課題をリスト表示する「自分の担当課題」などがあり、これらのガジェットを組み合わせてオリジナルのダッシュボードが作れます。

プロジェクト単位でのサマリーを表示したり、レポートを作成したりするための機能も用意されています。レポートの種類にはバージョンやユーザごとの作業負荷を表示する「バージョン作業負荷レポート」や「ユーザ作業負荷レポート」、指定した項目におけるプロジェクト全体の課題の割合を円グラフで表示する「円グラフレポート」などがあり、プロジェクトマネージャはさまざまな角度からプロジェクトの状態を把握できるようになっています(**図6**)。

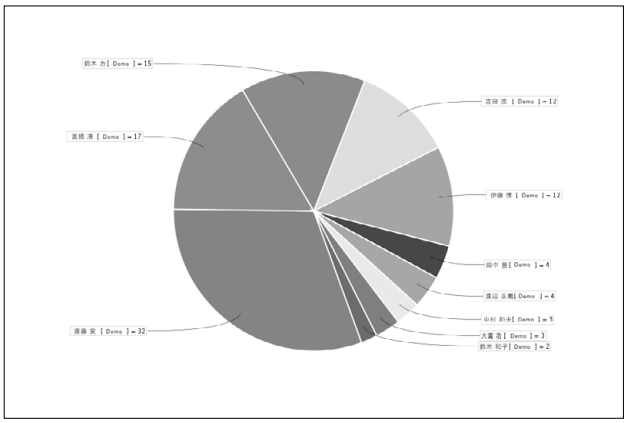
高度な検索機能やフィルタ機能が提供されていることも、JIRAの大きな特長です。JIRAには「JQL(JIRA Query Language)」と呼ばれる専用のクエリ言語があり、さまざまな条件を設定して課題を検索できるほか、その検索内容をフィルタとして使うこともできます。また、作成したフィルタをダッシュボードのガジェットから利用できるのも便利でしょう。ほかにもJIRAには便利な機能が多数盛り込まれており、プロジェクト管理におけるさまざまな課題を解決できます。Excelによるプロジェクト管理に限界を感じているのであれば、まずはJIRAを試してみましょう。なおリックソフトでは、JIRAのデモ環境や体験版を提供しています。ぜひアクセスしてみてください。SD

JIRA体験版：
https://www.ricksoft.jp/product/atlassian/jira

▼図4　タスク登録画面では、要約や優先度、期限、影響バージョン、担当者などを登録することが可能

▼図5　実際に登録した課題を表示したところ。ステータスを切り替えることで、このタスクに対する作業状況を明示できる。「コメント」ボタンでメッセージも残せる

▼図6　登録されているタスクや課題の数を、担当者別やバージョン別、解決状況別などの形で集計し、グラフ化できる



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。

ひみつのLinux通信

作)くつなりょうすけ
@ryosuke927

第10回 コマンド古今東西

食欲の秋を迎え撃つために、毎晩のラーメン二郎を欠かさないくつな先生に、愛のムチを！（ツイートとかメールとか）

to be Continued

夜間作業・ランチ後などの睡魔とアツいバトルのお供に、Linuxに用意されている数えきれないほどのコマンドを使った知的な遊びをするをのはいかがでしょうか。ここでご提案する遊びの1つが「片手入力コマンド古今東西」でございます。コマンドを入力しまくりの読者諸氏はスラスラと出てくることでしょう。「あれ？　こういうコマンドを打ったことがあるような」とmanで調べてみたら元カノと入力したコマンドで甘く切ない思い出もあわせてよみがえ……なワケないな。別の遊びで「コマンドしりとり」もあります。よーし、「a」から始めるよー。at tac cut tail lpr rev vmstat top ps strace ed df free……続きはみなさんでやってください。

Report

LL Diver、開催

8月23日、お台場の日本科学未来館（東京江東区）で2014年のLightweight Language（以下、LL）イベント「LL Diver（昼の部）」が開催された。LLイベントは2003年に始まり、今年で12回目。毎年イベントの名称が変わり、今年は「12年の間に『どうやって仕事で使うか』から『現場投入されているLLをどうよく扱うか』まで状況が大きく変わったLLを深掘りしたい」という思いから、LL Diverという名称になった。主催は、日本UNIXユーザ会を中心とした団体「LLイベント実行委員会」。法林浩之氏がメインの司会を務めた。会場では、

▲IT系専門書の物販、見本誌展示

LLに関する講演･セッションが開催されたほか、オライリー・ジャパンやユニバーサル・シェル・プログラミング研究所などが物販や見本誌展示を行った。

エディタ対決（仮）

Emacs、Vim、Atom、Sublime Textの4つのエディタについて、それぞれの愛好者4人（吉田昌平氏、kaoriya氏、mizchi氏、平出弥彦氏）が壇上に登り、パネルディスカッションを行った。

ディスカッション開始前、会場の観客に対して、仕事で使用しているエディタについてのアンケートが出され、「Vim→Emacs→Atom→Sublime Text」の順に人気だという結果が出た。壇上のスピーカーによると、昔は、「Emacsは若い人に、Vimは若くない人に人気がある」印象だったが、今はまったく逆だという。Emacsか、Vimか、どちらのエディタが優れているかという話題では、カスタマブルなEmacs、そのまま使えるVimという構図で、意見が交わされた。

新興のAtom、Sublime Textについては、日本語の扱いにまだまだ難があるが、Atomの、手軽にライブラリをインストールできる「apm」、Sublime Textの、すばやくファイルを探せる「Goto Anything」などが紹介されるなど、伝統的な2つのエディタに負けない魅力が感じられた。

▲エディタ対決（仮）

「これからのエディタ」という話題に話が移ると、統合開発環境の普及により、コードを書くのにエディタを使う機会は減るのではないか、というやや悲観的な意見も出たが、それに対して、開発言語ごとのプラグイン（Javaのjava.vimやGo言語のgocodeなど）も充実してきており、コーディングする環境としてはまだまだ価値がある、といった意見も出された。

bashでCMS作った上田だけどなんか質問ある？

USP友の会会長の上田隆一氏は、司会役の法林浩之氏とともに、シェルスクリプトについてのトークショーを行った。

上田氏はシェルスクリプトで開発を行うようになった経緯について、「できるだけ速く作り上げたい、を追求していった結果、シェルにたどり着いた」と語った。また、シェルで開発を続けるモチベーションについて質問された際は、「シェルを普及させたいという気持ちでやっている」と答えた。7月に発売された、上田氏が執筆した『フルスクラッチから1日でCMSを作る　シェルスクリプト高速開発手法入門』（KADOKAWA）の紹介では、「話をもらった時点ではどのような人が買うのだろうと不安だったが、予想外に多く売れて驚いている」と語った。

▲USP友の会会長　上田隆一氏

▲日本UNIXユーザ会幹事　法林浩之氏

毎年恒例「夜の部」

昼の部終了後、会場は東京カルチャーカルチャーに移り、ソフトウェア開発者が目撃したひどい設計のプログラムを紹介し合う「この設計がひどい2014」、使っているプログラミング言語のダメなところを語り合う「帰ってきただめ自慢」の2つのイベントが行われ、お酒を交えながらの歯に衣着せぬトークが繰り広げられた。

▲夜の部「この設計がひどい2014」

CONTACT　LL Diver公式ページ
URL http://ll.jus.or.jp/2014/

Report

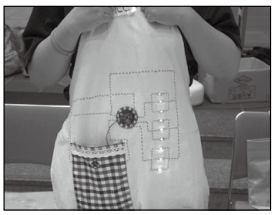
オープンソースカンファレンス2014 Kansai@Kyoto、開催

8月1日および2日、京都リサーチパークにおいて、「オープンソースカンファレンス2014 Kansai@Kyoto」（以下OSC京都）が開催された。京都でのオープンソースカンファレンス開催は今年で8回目。オープンソースにかかわるさまざまな企業・団体が、ブースの出展とセミナーを行った。

LilyPad研究会@ノートルダム

OSC京都の運営にも携わった京都ノートルダム女子大学の学生数名を中心とした「LilyPad研究会@ノートルダム」のブースでは、マイコンLilyPadと、日本語プログラミング環境PENを使った「エプロンコンピュータ」が展示された。このエプロンコンピュータには温度センサーを使った相性診断の機能が実装されており、エプロンのポケット越しに2人の人間がセンサーを触ると温度が感知され、相性を診断できる。

開発に参加した女子大生はいずれも文科系の学部所属で、プログラミングは初心者。エプロンコンピュータの開発は2014年6月ごろから始まり、大学の試験期間と重なったこともあってなかなか開発に時間がとれなかったそうだが、楽しんでプログラミングの勉強と開発ができたとのことだ。



▲エプロンコンピュータ

Ubuntu Japanese Team

イベント2日目、Ubuntu Japanese Teamの長南浩氏がセミナーを行った。最初に、10月23日にリリースされる「Ubuntu 14.10（Utopic Unicorn）」を紹介した。Utopic Unicornは9ヵ月間のサポート期間ということで、LTSよりも期間が短いことに注意する必要がある。

現行の「Ubuntu 14.04（Trusty Tahr）」については、Japanese Remixの日本語環境が強化されたこと、アプリケーションのメニューをグローバルメニューに表示できるようになったことが紹介された。また、利用にあたってのTipsとして、非公式のリポジトリ「パーソナル・パッケージ・アーカイブ（PPA）」を紹介し、これによって、エディタ「Atom」のインストールや、「Ambiance & Radiance Colors Version」でデスクトップのカスタマイズが可能になると語った。

▲Ubuntu Japanese Team 長南浩氏

日本アイ・ビー・エム

IaaS（Infrastructure as a Service）製品である「SoftLayer」について、日本アイ・ビー・エム㈱の2名のスピーカーがセミナーを行った。

北瀬公彦氏はSoftLayerについて、①グローバルな高速ネットワーク、②物理サーバと仮想サーバの柔軟なコントロール、③豊富なAPIという点で、ほかのクラウド製品と比べて優位性があると語った。とくに①については、㈱データホテルの事例を取り上げて説明した。同社は日本と韓国にデータセンターを持ち、そのほかのアジア・北米・欧州でのサービスにはSoftLayerを利用している。また、今年2014年には、日本にもSoftLayerのデータセンターが開設される予定だ。

また、高良真穂氏はSoftLayerが、Parallels、VMware、Dockerといった複数の仮想化システム、Evault、Chef、Vyattaといったさまざまなサービスと連携できることを説明した。最後には、マイコンRaspberry PiにSoftLayerのAPIをインストールし、カメラの映像をオブジェクトストレージへ自動的に保存する自作の監視カメラを紹介し、会場を驚かせた。

▲日本アイ・ビー・エム 北瀬公彦氏

▲日本アイ・ビー・エム 高良真穂氏

イベントフィナーレ

2日目最後のセミナープログラムとして、計9組によるライトニングトークが行われた。

2014年4月に鹿児島で発足したLinuxやPC UNIX、OSS関連の勉強会「鹿児島らぐ」や、同じく今年4月から活動を始めた、コンピュータの原理に関する京都での勉強会「MyCom Cafe Kyoto」など、小規模ながらも魅力的な団体の発表が目を引いた。

閉会式では、OSC京都実行委員長である京都ノートルダム女子大学の吉田智子氏が閉会式に集まった大勢の出展者・来場者に向けて、感謝の意と閉会の挨拶を述べた。

▲京都ノートルダム女子大学 吉田智子氏

CONTACT オープンソースカンファレンス2014 Kansai@Kyoto
URL http://www.ospn.jp/osc2014-kyoto/

Software

パラレルス、「Parallels Desktop 10 for Mac」を発表

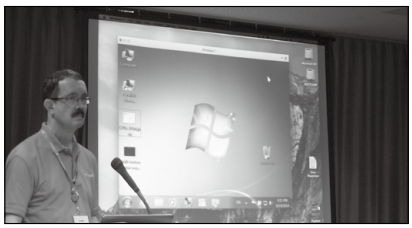
パラレルス㈱は、Mac上でWindowsを実行できる仮想化ソフトウェア「Parallels Desktop 10 for Mac」を8月26日に発売した（既存ユーザ向けのアップグレード版は8月20日から）。

「Parallels Desktop for Mac」シリーズはMac用の仮想化ソフトウェア。Mac上で、Windowsをはじめ、OS X、各種Linux OSなどの仮想マシン（以下、VM）を同時に起動できる。最新版ではおもに、次のように性能が向上した。

- 次期のOS XであるYosemiteを含むAppleの最新OSへの対応
- DockのVMアイコンにファイルをドラッグ&ドロップすることでWindowsアプリケーションを簡単に開けるなど、操作性が向上
- CPUは16スレッド、RAMは64GBまでサポート
- 仮想マシンがMacのディスク領域を必要な容量だけ使用
- VMのメモリ使用率の低減やバッテリー持続時間の延長など、全体的なパフォーマンスの向上

8月20日に行われた当製品の発表会では、同社のクロスプラットフォームソリューションズ担当プロダクトマネージャのカート・シュマッカー氏が製品のデモを行った。氏は、持参した2年前発売のMacBook AirにYosemiteをインストールし、そのうえでWindows 7、8、OS X LionのVMを同時に稼働させ、ることで製品のパフォーマンスをアピールした。

価格は、8,500円（税込み）、アップグレード版は5,300円（税込み）となっている。



▲パラレルス㈱　カート・シュマッカー氏

CONTACT　パラレルス㈱
URL http://www.parallels.com

Service

リンク、エーティーワークス、Trimコマンドに対応したSSD RAIDマシンの提供を開始

㈱リンクと㈱エーティーワークスが共同で展開しているホスティングサービス「at+link」は、9月10日から、専用サーバサービスのオリジナルマシンATシリーズにおいて、SSDをRAID構成で搭載した「AT02-G2（SSDタイプ）」の提供を開始した。

AT02-G2（SSDタイプ）は、障害耐性・可用性が要求されるデータベースサーバや、高い信頼性が必要なアプリケーションサーバなどの用途に適している。また、RAID構成でありながらTrimコマンド（OSからSSDに対し、あらかじめ削除を行うデータ領域を通知するコマンド）に対応しており、SSDの残り容量が少ない状態で使用した場合でも、高速な読み書きを実現できる。

▲AT02-G2（SSDタイプ）

記憶媒体としてFlashメモリを採用したSSDは、ランダムアクセス時の読み書き性能がHDDなどの磁気ディスクと比べて高速な点を特長としており、at+linkでは2009年6月からSSDを搭載したサーバを提供している。「SSDを障害耐性・可用性の高い構成で利用したい」というユーザ企業からの要望に応えて、今回、SSDをRAID構成としたモデルを提供するに至ったとのこと。

▼AT02-G2（SSDタイプ）詳細

CPU	Xeon E3-1230 v3 (1.8GHz)
ストレージ	標準：SSD 120GB×2 (RAID 1) 最大：SSD 240GB×2 (RAID 1)
メモリ	標準：4GB 最大：16GB
OS	CentOS 6（x64）
初期費用	168,000 円
月間利用料	19,000 円～ （2 台目以降は 14,000 円～）

CONTACT　㈱リンク
URL http://www.link.co.jp/
㈱エーティーワークス
URL http://www.atworks.co.jp/

Service

日立システムズ、トレンドマイクロ、「Trend Micro Deep Security」の監視サービスを提供開始

8月5日、トレンドマイクロ㈱の総合サーバセキュリティ対策製品「Trend Micro Deep Security」をリモートで運用、監視するサービスを、㈱日立システムズが提供開始することが発表された。

ユーザのITシステム上にあるセキュリティデバイス（ファイアウォール、IPSなど）をSecurity Operation Center（以下、SOC）からリモートで運用、監視する日立システムズのサービス「SHIELDセキュリティデバイス監視サービス」の対象に、同セキュリティ対策製品を追加する形での提供となる。

サーバ上のOSやアプリケーションの脆弱性を突く攻撃パケットを検知・防御する「Trend Micro Deep Security」を日立システムズのSOCからリモートで運用、監視することで、分散されたITシステムにおいても、セキュリティ対策として均一的なポリシーの適用が可能となり、分散したITシステムでも、ホスト型で強固なセキュリティを保つことができる。

CONTACT
㈱日立システムズ
URL http://www.hitachi-systems.com/
トレンドマイクロ㈱
URL http://www.trendmicro.co.jp/

Topic

LPI-Japan、「HTML5 プロフェッショナル認定試験レベル2」を配信開始

特定非営利活動法人エルピーアイジャパン（以下、LPI-Japan）は、「HTML5プロフェッショナル認定試験レベル2」を9月24日に配信開始する。

試験は、HTML5、CSS3、JavaScriptなど、最新のマークアップ言語に関する技術力と知識を、LPI-Japanが公平かつ厳正に、中立的な立場で認定する認定資格である。2014年1月から配信している「レベル1」は、HTML5の技術やメリットを活かした、静的コンテンツを構築する実力を認定するための試験。出題範囲はHTML5の要素やCSSを中心に設計されている。今回配信開始される「レベル2」は、HTML5を活用した動的なWebアプリケーションを設計・作成する実力を認定することを目的としている。おもに、JavaScript言語や、HTML5で利用できる主要なJavaScript APIなどの知識とスキルが問われる内容となっている（詳しい試験範囲はhttp://html5exam.jp/outline/objectives_lv2.html を参照）。

受験料は15,000円、試験方式はコンピュータベーストテスト（団体受験用にペーパーテストも実施される）。

CONTACT
特定非営利活動法人 エルピーアイジャパン
URL http://www.lpi.or.jp/

Software

アックス、わさらぼの「Synthesijer」をサポート

㈱アックスは、FPGA（Field-Programmable Gate Array：製造後でも回路の書き換えができる集積回路）開発マーケットに向けて、わさらぼ合同会社が開発した抽象度の高い高位合成処理系「Synthesijer」をサポートすることを発表した。

Synthesijerは、プログラミング言語Javaで書かれたプログラムをFPGAの論理回路に変換できるツール。ソフトウェアとして書かれたアルゴリズムから、FPGA上の専用ハードウェアを合成できる。Javaを用いることで、VHDL（回路設計用のハードウェア記述言語）では記述が難しいとされる、変数が多用されたアルゴリズムを容易に記述できる。

アックスは、自社のハードウェア・コンサルティング、ソフトウェア技術、オープンソースサポート体制により、「Synthesijer」をサポートするとのこと。わさらぼは、Synthesijerを通じて「ソフトウェアとハードウェアの垣根を越えて目的に合致したシステムを手軽に開発できるようにすること」を大きな目的としている。

CONTACT
㈱アックス
URL http://www.axe-inc.co.jp/
わさらぼ合同会社
URL http://wasa-labo.com/

Report

Bluetooth Special Interest Group、Bluetooth最新動向～Bluetooth Smartの普及

Bluetooth Special Interest Group（以下、BSIG）は8月22日、品川の東京コンファレンスセンターでBluetoothの最新動向に関する発表を行った。発表会では、全世界におけるBluetoothのブランド戦略を担当するBSIGのエリット・クローター氏を始め、BSIGのメンバ企業のスピーカーが登壇した。

BSIGのクローター氏は、Bluetooth Low Energy（以下、BLE）に対応した機器「Bluetooth Smart」の普及について、ウェアラブル端末に適した低消費電力がカギになったと語った。とくに、家庭内の家電製品をネットワークにつなぎ、自動制御を行う「スマートホーム」の分野でのBluetooth Smartの成長率は、2013年から2014年までで232%増と見込んでいる。


▲BSIG　Errett Kroeter氏

Nordic Semiconductor ASAの山崎光男氏

同社カントリー・マネージャの山崎氏は、システム・オン・チップデバイス「nRF51 IC」を紹介した。このデバイスでは、Bluetooth Smartや、2.4GHzの独自プロトコル、ANTプロトコルを同時に動作させることができる。また、アプリケーションのコードは、プロトコル・スタックから独立した形でコンパイル／リンクされるため、アプリケーションの修正がスタックに影響を与えない。それにより、安全で迅速な開発ができるとのこと。

シーエスアール㈱の篠崎泰宏氏

同社開発の独自技術「CSRmesh」は、Bluetooth機器を相互接続し、1台のBluetoothハブから、そのネットワーク上のすべての端末を操作できるというもの。FAEディレクターの篠崎氏は、CSRmeshを活用して建物の中の照明機器を1つのスマートフォンで操作する内容のデモビデオを見せた。

CONTACT
Bluetooth Special Interest Group
URL https://www.bluetooth.org
Nordic Semiconductor ASA
URL http://www.nordicsemi.com/
シーエスアール㈱
URL http://www.csr.com/

Software

CommVault Systems Japan、データ管理プラットフォーム「Simpana」新ソリューションセットを発表

8月26日、CommVault System社は、統合データ管理プラットフォーム「Simpana」について、クラウドデータ自動管理機能を備えた新しいソリューションセットをリリースすることを発表した。

Simpanaは、バックアップ・アーカイブ・レプリケーション・ストレージリソース管理・全文検索などの機能を、単一プラットフォームで行えるソフトウェア。

今回リリースされるソリューションセットはAmazon Web Services、Microsoft Azureを新たにサポートし、堅牢なクラウドレポーティングや、セルフサービスプロビジョニング、リカバリ、ソフトウェアスナップショット、仮想マシンのリソース管理といった機能が利用できる。これら新しいSimpanaソリューションセットは、日本では2014年10月1日から利用できる。ソリューションセットのラインナップは次のとおり。

- Simpana for VM Backup, Recovery and Cloud Management
  対応するすべてのプラットフォームに対して、単一のインターフェースで、自身のVMプロビジョニング、管理、バックアップ、リカバリ、リタイヤメント、アーカイブ操作を行える

- Simpana for IntelliSnap Recovery
  アプリケーション整合のあるハードウェアスナップショット、レプリケーション、リカバリを統合／合理化できる

- Simpana for Endpoint Data Protection
  効率的にデバイスをバックアップし、セキュアなアクセスやセルフサービス検索機能を提供することにより、モバイルワークフォースの保護と有効活用を実現できる

- Simpana for Email Archive
  Microsoft Office 365、Outlook、Exchange、IBM Lotus Noteをサポートし、オンプレミス／クラウド環境で、デプロイできる

CONTACT
CommVault Systems Japan㈱
URL http://www.commvault.jp/

Hardware

メカトラックス、
Raspberry Piと接続できる3G通信モジュール「3GPI」を発売

8月14日、メカトラックス㈱は、小型コンピュータボード「Raspberry Pi」と簡単に接続できる携帯電話無線網（3G）通信モジュール「3GPI」を発売した。

本製品は、Raspberry Pi（typeB、B+推奨）に接続でき、電源を供給する機能も備えている3G通信モジュール搭載基板。低価格のデータ通信専用SIMなどでの利用を想定している。同梱されるSDカード内に、アクセスポイント設定や通信不調時の自動リセット機能などが、Linux環境（raspbian）とともに構築されており、3G通信モジュールの使用に慣れていない人でも、HTTPプロトコルなどを用いて簡単にM2M（Machine-to-Machine）機器などの開発ができる。

先行リリースの限定20台はすでに完売。今後の発売時期は未定だが、価格は3万円程度を想定しているとのこと。


▲3GPI

CONTACT メカトラックス㈱
URL http://www.mechatrax.com/

Report

日経BP社、
ITpro EXPO Special in Summer 2014「2020年へのITパラダイムシフト～さらばWindows Server 2003～」開催

㈱日経BP社は、Microsoft Windows Server 2003（以降、WS2003）のサポート終了が2015年7月15日（日本時間）に迫る中、「2020年へのITパラダイムシフト～さらばWindows Server 2003～」と題したセミナーイベントを開催した（東京：8月26日／大阪：8月28日）。

特別協賛の日本マイクロソフトとインテル、そして移行をサポートするベンダー各社による本セミナーは、WS2003が稼働している企業に対してサポートが終了することによって発生するリスクを周知することと、移行に際しての選択肢を提供することが主な目的。

マイクロソフトの調査によれば、2014年6月末時点で国内で稼働しているWS2003の台数は約30万台。移行はオンプレミスだけではなくホスティングやクラウドを視野に入れ、「適材適所で組み合わせて全体を最適化する」というのがほぼ統一の見解として語られていた。早急に（1）既存環境の棚卸し（2）移行先の選択（3）予算とスケジュールの確認を進めることを呼びかけた。

詳細な情報は、Windows Server 2003移行ポータルサイト（**http://aka.ms/ws03mig**）を参照。

CONTACT 主催　ITpro
URL http://itpro.nikkeibp.co.jp/

Event

2014年10月3日、
「Design Solution Forum 2014」開催

2014年10月3日、新横浜国際ホテル（神奈川県）において、設計者主体のセミナーイベント「Design Solution Forum 2014」が開催される。

本イベントは、「エンジニア同士のさまざまな情報共有を目指し、個人のスキルアップやキャリアアップ、業務におけるエンジニアリングの改善・効率化、新たなビジネスの創出・育成につながる、『人』と『技術情報』の交流の場を創出すること」が目的。「Design」、「Verification」、「Software」、「FPGA」といったキーワードを基に、設計現場のエンジニアがプログラムを企画し、デザイン事例と技術トレンドの紹介を中心にセッションを行う。イベントの最初に行われる基調講演では、「IoT、次世代移動通信5Gの動向とエレクトロニクスの展望」と題して、㈱NTTドコモの二方敏之氏が登壇する予定。

入場は無料だが、Web（**http://www.dsforum.jp/registration.html**）での登録が必要となる。

CONTACT Design Solution Forum 2014実行委員会
URL http://www.dsforum.jp/
一般社団法人日本エレクトロニクスショー協会
URL http://www.jesa.or.jp/

Letters from Readers

言語系イベントが熱い!!

「LL Diver」「YAPC」「PyCon」「RubyKaigi」など、8～9月にかけて言語系のイベントが続きました。取材でイベントへ行くと、いろいろな国の人たち、若い人からそうでない人まで、プログラミング技術について真剣に議論している様子がみられます。仕事の延長としてだけではなく、生活の一部として取り組んでいるからこそ、真剣になれるのだろう、と思いました。

2014年8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 ログを読む技術

システムログ、アクセスログ、クエリログ、エラーログなど、システムを日々運用していると、ログが蓄積されていきますが、放っておいては宝の持ち腐れです。本章ではログを分析・利用することで、管理・運用の効率を上げる方法を解説しました。

この少ないページ数でここまで幅広くカバーできているのは驚き。ログの由来であったり、anacronの情報であったり、知らないtipsがたくさんあった。

東京都／藤田さん

前回アンケートで「ログまわりについて書いてほしい」と書いたら、特集されたのでうれしかったです。とても助かりました。

長野県／金井さん

オープン系のエンジニアではないが、アプリケーション側からの解析に役立つと感じた。

東京都／Shimizusさん

第5章は、ログをCLIで分析するときに参考になる例が多くて楽しんで読めました。また、FluentdとMongoDBの連携も試してみたくなりました。このように、通常の業務で使っているノウハウがあると参考になるため、ぜひまた特集してほしいです。

千葉県／今井さん

自作PCでWheezy（Debian 7）をインストールして使っているのですが、起動時にエラーがゾロゾロと出ます。ATA、すなわちドライブのエラーのようなのですが、一瞬で流れていくので、よくわかりません。起動時のログを読めるようになると、解決策が見つかるかもしれませんが、まだそこまで到達していません。この特集を参考にして、何とか解決したいです。

愛知県／kmさん

Webalizerやtailコマンドなど、実際の現場で使われているログの読み方は貴重な情報でしたね。エラーが出ていないからといって読んでいなかったログをいざ見てみると、改善できる部分を発見できるかもしれません。

第2特集 Linuxカーネルのしくみを探る

Linuxシステムを構築・運用するときに、OSの核となる「カーネル」の中身について知っておけばいろいろと役立つことがあるでしょう。カーネルの機能のうち、「プロセス管理」に注目して、そのしくみを解説しました。

裏側のしくみを知る機会が得られてとてもよかった。自分で調べるにも時間が限られていたりするので。

東京都／大塚さん

Linuxカーネルの中身についてプロセスやスレッドがどのように働いているのか順序だてて書かれていてよかった。Linuxだけでなくほかの OSにおけるカーネルでも同じように動いているのだろうか。

北海道／村橋さん

入門編としてよかった。もっと周辺の知識もちりばめてもよかった。

大阪府／ドモチェフスキーさん

普段使っているシステムの内部がどうなっているか、何が起きているかを知ることは大切です。時間に余裕があるときなど、カーネルのソースコードを一から眺めてみるのもいいかもしれませんね。

一般記事　OpenSSLの脆弱性“Heartbleed”の教訓（後編）

C言語で書かれたOpenSSLのソースコードを見ながら、Heartbeat Buffer Overreadの脆弱性がどのように実装され

たかを説明しました。

報道されてから、ずっと気になっていました。

大阪府／牧さん

メモリに読み込まれてバイナリで解析されてしまう弱点は困りますね。

千葉県／Tayuさん

心電図と記事がマッチしていた。

福島県／ライヘンさん

単純な実装ミスが世界規模の大問題になった「Heartbleed」。世の中ではRubyやPerlなどのスクリプト言語が流行っていますが、これを機にC言語でメモリの扱い方について勉強を始めてみるのはいかがでしょうか。

一般記事 使ってみよう! tcpdump

手軽に使えるコマンド「tcpdump」を使って、ネットワークを流れるデータを出力し、分析していきました。

その系統のソフトウェアはWireSharkしかまともに使ったことがないので、あらためて使ってみようと思いました。

神奈川県／miffさん

パケットレベルでの説明だったので非常に参考になった。

大阪府／てんぷるさん

パケットのデータを、各プロトコルのフォーマットに照らし合わせて見ることで、ネットワークの流れを具体的にイメージできたのではないでしょうか。

フリートーク

自宅などでサーバを運用していると、自分がLinuxやサーバについて何でも知っているかのような錯覚を起こす。しかし、こういった専門書を読むことで自分を現実に引き戻すことができます。あらためて、自分がOSの数%しか利用していないことに気づかされ、無知を恥じながら学習する毎日です。

福島県／SonneTagebuchさん

弊誌ではさまざまな経歴を持った著者の方に記事をお願いしています。OSひとつをとっても、仮想化やカーネルの中身など、さまざまな観点から書かれた記事を載せていますので、幅広い知識が得られると思います。

エンジニアの能率を高める一品

仕事の能率を高める道具は、ソフトウェアやスマートフォンのようなデジタルなものだけではありません。
このコーナーではエンジニアの能率アップ心をくすぐるアナログな文具、雑貨、小物を紹介していきます。

Tablet Strap 360 by HELO Strap

4,500円（税別）　スペックコンピュータ／http://spec-computer.co.jp/

写真1▶ストラップを付けたままで、縦横の向きを変えられる

iPad Airなどの9.7インチサイズのタブレット端末は、画面が大きくて見やすいのですが、片手で扱うには少々重たくて持ちにくいです。長時間、画面の縁をつかんで持っていると指が疲れてきます。

そんな場合には、タブレット端末用のハンドストラップ「Tablet Strap 360 by HELO Strap」を使うと、端末を手のひらの上に置くような状態で持てるので指が痛くなりません。ストラップでしっかり固定されているため、落とす心配も少なくなります。たとえば、電車の車内では、軽く人とぶつかったり、急ブレーキで体勢が崩れたりしますが、そんなシーンでも容易には落とさないので安心です。

ストラップを手にはめたまま、端末を360度回転させることもできます（**写真1**）。素手で端末を持っている場合、縦から横に持ちかえるときは、両手で持ちかえなければいけませんが、本ストラップを付けていると、指を添えて押すだけで縦横の向きを変えられるので、外出時に限らず室内でタブレット端末を使うときでも、本製品の有用性を実感できます。本製品は9.7インチサイズ以上であれば、iPad以外のタブレット端末でも使用できます。ただ、いくら便利でも「歩きスマホ」ならぬ「歩きタブレット」はやめましょうね。

祝 8月号のプレゼント当選者は、次の皆さまです

①Microsoft All-in-One Media Keyboard ..... 東京都　室井武雄様
②OTG USBメモリ Mobile X20 ..... 大阪府　宮嵜一臣様

★その他のプレゼントの当選者の発表は、発送をもって代えさせていただきます。ご了承ください。

次号予告

Software Design
November 2014

2014年11月号
定価（本体1,220円+税）
176ページ

10月18日発売

［第1特集］新技術動向に追いつけ!

LAMP環境再点検
昔のままの環境で大丈夫?

［第2特集］x86サーバの性能を見抜く

サーバの目利きになる方法［後編］
ネットワークとストレージを極める

※特集・記事内容は、予告なく変更される場合があります。あらかじめご容赦ください。

お詫びと訂正

以下の記事に誤りがございました。読者のみなさま、および関係者の方々にご迷惑をおかけしたことをお詫び申し上げます。

■2014年9月号
- ●目次②「SoftLayerを使ってみませんか?」
  ［誤］常田 秀樹　［正］常田 秀明
- ●連載「ITエンジニア必須の最新用語解説」（第69回） P. ED-1
  2014年8月号に掲載した第68回の記事を再掲載してしまいました。今号に第69回の記事を掲載しております。
  また9月号のサポートWebサイト（http://gihyo.jp/magazine/SD/archive/2014/201409/support）から記事PDFもダウンロード可能です。
- ●連載「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」（第5回） P.122［リスト1］
  正しいコードは、「3行目にある(package-initialize)を1行目に移動させたもの」となります。
  詳しくは9月号のサポートWebサイトでご確認ください。

休載のお知らせ

「温故知新　ITむかしばなし」（第38回）は、著者急病のためお休みさせていただきます。

SD Staff Room

●今年の夏は80年代ロックを聴くことが多かった。Duran DuranからスピンアウトのThe Power Stationのデジタルリマスターを買ったりした。当時はテープだったので音が悪かったのがよくわかった。今聴くと、音の繊細さや当時のこだわりがわかってテクノロジーに感謝した。（本）

●2000年に作成した仕事用のExcel VBAのプログラムを改良した。14年前に、自分はこんな（無駄のたくさんある）プログラムを書いていたのだなぁと実感。ついでにバグも発見。最近はRubyでスクリプトを書く案件もあり、頭の体操にいいなと思いつつ、久々に夢の中でデバッグを実施した。（幕）

●夏休みは家族で千葉のマザー牧場に行ってきました。園内にコテージタイプの宿泊施設があって、ちょっとした別荘気分。台所もあってみんなで夕食を作ったり、畳の部屋でゴロゴロしたりと、かなりくつろげました。外では牛の乳搾りをしたり、子ヒツジを愛でたり、広い原っぱでゴロゴロしたり……（キ）

●Java特集では、コードばかりの誌面に少しかわいい要素を加えたいと思い、イラストレータさんに女の子の絵を描いてもらいました。そこにデザイナーさんのセンスが加わり、想像以上に「萌えー」な誌面になりました。出来は大いに満足ですが、「萌え」は容易に制御できるものではないな、と感じました。（よし）

●大阪で7月から8月にかけて開催された「レゴブロックで作った世界遺産展」に行って来ました。ピサの斜塔や自由の女神像、金閣寺などがすべてレゴで作られ会場中に展示されており、その再現度たるや、本当にお見事でした。子供時代はレゴに熱中していたこともあり、レゴ熱が再燃しそうです!（な）

●にわか家庭菜園は難しかった。生き残ったキュウリの苗は結局変な虫がついて枯れ、オクラは元気だけれども実ができる気配はなし。豆苗と分葱は収穫できたけれども、一回限り……。水をあげてれば大丈夫だろうと簡単に考えていたけど、きちんと勉強しないと収穫まではたどり着かないみたいです。（ま）

ご案内
編集部へのニュースリリース、各種案内などがございましたら、下記宛までお送りくださいますようお願いいたします。

Software Design 編集部
ニュース担当係

［FAX］
03-3513-6173
※FAX番号は変更される可能性もありますので、ご確認のうえご利用ください。

［E-mail］
sd@gihyo.co.jp

Software Design
2014年10月号

発行日
2014年10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。