special feature 1 苦手克服の手がかり　special feature 2 クラスタリングを極める

Software Design

2014 9

2014年9月18日発行
毎月1回18日発行
通巻353号
（発刊287号）
1985年7月1日
第三種郵便物認可
ISSN 0916-6297
定価 本体 1,300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

この夏に克服したい2つの壁

special feature 01

C言語のポインタ と ——習得のヒントと実践 オブジェクト指向

special feature 02

クラスタリングの教科書 止まらないサービスを支えるシステム構築の基礎

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?

8ポートのチーミングで実力検証
NICをまとめて高速通信!（前編）

開発や運用を最適化するオーケストレーションツール
Serf・Consul入門［Serf編］

新連載

Heroku女子の開発日記

特別付録
お試し!
まるごと収録［ルーティング］編
3分間ネットワーク基礎講座 特別編
網野衛二 著

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

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

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

第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/

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 Sep. 2014

［第1特集］

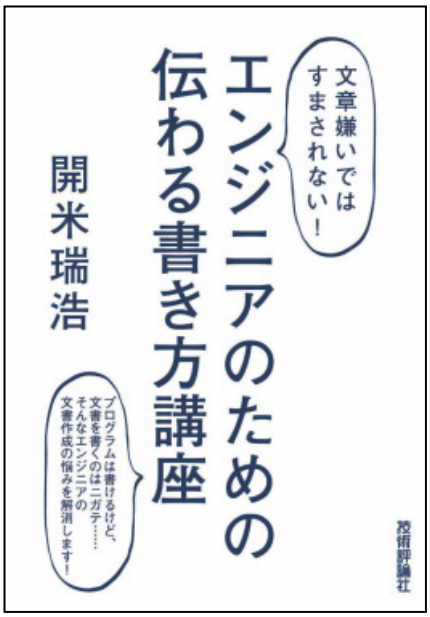
この夏に
克服したい
2つの壁

「C言語のポインタ」と「オブジェクト指向」

―― 習得のヒントと実践

［特別付録］

BOOKS INFOMATION



ISBN978-4-7741-6576-9
A5判／200ページ
定価（本体1980円+税）

＜文章嫌いではすまされない！＞

エンジニアのための伝わる書き方講座

●開米瑞浩　著

文書作成でお悩みですか？　ビジネスで使う文書を作るのに、文筆家が書くような「うまい文章」の書き方を修得する必要はありません。エンジニアに必要なのは、難しくなりがちな内容を相手に理解してもらうための「わかりやすい文書」の書き方です。
文書作成能力を向上させるための企業研修を請け負ってきた著者が、さまざまな悩みの相談を受けてきた経験から得られた「わかりやすく書くためのポイント」が、本書にはロジカルにまとめられています。自分の悩みに近いものから実践し、毎日数分の練習を続けることで、コミュニケーション能力の高い、頼れるエンジニアになりましょう！

ISBN978-4-7741-6613-1
B5変形判／176ページ
定価（本体2180円+税）

伝わるデザインの基本

よい資料を作るためのレイアウトのルール

●高橋佑磨、片山なつ　著

プレゼン用スライド・広報資料・企画書から、チラシ・ポスターなどまで、さまざまな資料を自分で気軽に作れるようになりました。しかし、なかなか魅力的なデザインにならず苦労することが多いようです。その原因は、デザインの基本ルールを知らないことにあります。
本書では、フォントの選び方から文字の配置、図表やグラフ、資料全体のレイアウトや配色まで、押さえておきたい基本ルールを豊富な事例とともに解説します。基本ルールをマスターすれば、WordやPowerPointであっても読みやすく伝わりやすいそしてかっこいい資料が作れます！

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

Contents

Software Design Sep. 2014

2

［広告索引］

実践的
テクニック開発の
決定版

テクニックバイブルシリーズ

最新刊！

Vim script テクニックバイブル

ISBN978-4-7741-6634-6

Vim scriptサポーターズ　著／A5判／320ページ　定価（本体2580円+税）

Vimスクリプトの基礎からプラグインの作成・公開方法まで。
自分だけのVim探しにでかけよう。

Vim テクニックバイブル

ISBN978-4-7741-4795-6

Vimサポーターズ　著／A5判／384ページ　定価（本体2980円+税）

unite.vimで進化する新しいVimの常識を教えます。
作業効率をカイゼンする150の技。

Emacs テクニックバイブル

ISBN978-4-7741-4327-9

るびきち　著／A5判／384ページ　定価（本体2780円+税）

anything.elで新しいEmacsの世界を教えます。
作業効率をカイゼンする200の技。

Emacs Lisp テクニックバイブル

ISBN978-4-7741-4897-7

るびきち　著／A5判／336ページ　定価（本体2980円+税）

逆引きリファレンスにも対応。
この本であなたをEmacs Lisp使いに進化させます！

Visual Basic テクニックバイブル

ISBN978-4-7741-5768-9

高橋広樹　著／A5判／384ページ　定価（本体2680円+税）

コントロールの使いこなしがアプリ開発の近道。
VBプログラミングの達人が贈る、実践的テクニックの決定版！

JavaScript テクニックバイブル

ISBN978-4-7741-5243-1

JSサポーターズ　著／A5判／432ページ　定価（本体2980円+税）

開発をする際の効率を改善するための手法やライブラリを重点的に解説。
現場のJSerが贈る実践的テクニックの決定版！

HTML 5 テクニックバイブル

ISBN978-4-7741-6193-8

HTML5サポーターズ　著／A5判／368ページ　定価（本体2780円+税）

ユーザビリティからパフォーマンスまで最新のHTML5関連の技術を解説。
Yahoo! Japanの技術者が贈る実践的テクニックの集大成。

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

Contents

Software Design Sep. 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	GK Hart/Vikki Hart /gettyimages
Illustration イラスト	フクモトミホ
Page Design 本文デザイン	岩井 栄子、 ごぼうデザイン事務所、 近藤 しのぶ、 SeaGrape、 安達 恵美子 [トップスタジオデザイン室] 轟木 亜紀子、阿保 裕美、佐藤 みどり [BUCH+] 伊勢 歩、横山 慎昌、 森井 一三、 Re:D、 [マップス] 石田 昌治

この個所は、雑誌発行時には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。

紙面版
A4判・16頁
オールカラー

電脳会議

DENNOUKAIGI

一切無料

新規購読会員受付中!

『電脳会議』は情報の宝庫、世の中の動きに遅れるな!

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

今が旬の情報満載!

新規送付のお申し込みは…

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

電脳会議事務局　検索

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

一切無料!

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

毎号、厳選ブックガイドも付いてくる!!

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

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

OSとネットワーク、
IT環境を支えるエンジニアの総合誌

Software Design

毎月18日発売

PDF電子版
Gihyo Digital
Publishingにて
販売開始

年間定期購読と電子版販売のご案内

1年購読(12回)

14,880円(税込み、送料無料)1冊あたり1,240円(6%割引)

PDF電子版の購入については

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

をご覧ください。
PDF電子版の年間定期購読も受け付けております。

・インターネットから最新号、バックナンバーも1冊からお求めいただけます!
・紙版のほかにデジタル版もご購入いただけます!
デジタル版はPCのほかにiPad/iPhoneにも対応し、購入するとどちらでも追加料金を支払うことなく読むことができます。
※ご利用に際しては、／~＼Fujisan.co.jp(http://www.fujisan.co.jp/)に記載の利用規約に準じます。

Fujisan.co.jpからのお申し込み方法

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

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

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 189

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

» Google I/O 2014で出会ったデジタルガジェットたち

開発者の祭典 Google I/O 2014

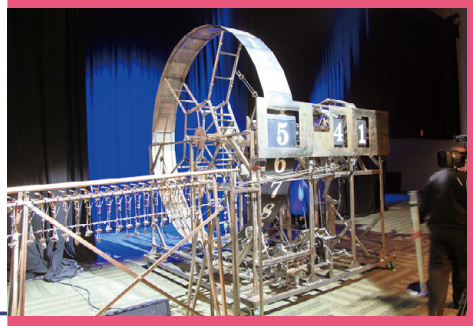
2014年6月25日と26日の2日間、「Google I/O 2014」が開催されました。Google I/OはGoogleの各種技術に関して、多くの発表とセッション、展示が行われる開発者向けのカンファレンスです。今年も世界各国85ヵ国から会場の定員いっぱいの約6,000人を集め、盛大なイベントとなりました。今年はとくにシリコンバレー近辺の開発者はもとより、インド、アフリカ圏、アジア圏と多様な国々のさまざまな言語が飛びかう会場でした。

一部のものを除き、ほとんどすべてのセッション内容はYouTubeで公開され、さらにI/O Bytesと呼ばれる各テクノロジを要約した5分前後のダイジェスト動画も数多く公開されています。今年の傾向は次のとおりです。

- スマートフォン、タブレット、パソコン、ブラウザにとどまらず、マルチプラットフォーム戦略がより強固に。テレビや車、ヘルスケア分野、家にも手を広げる
- ライバルを牽制。今までGoogleはライバルを気にしない姿勢が強かったが、最近はFacebook、Apple、Amazonに対抗意識をあらわにしてきた印象が強い
- 技術的な要素は大切にしつつも、デザインやユーザ体験をより重視する傾向に
- コンテキストアウェアネス:利用者の文脈を理解し、サービス間のシームレスな連携を考える
- Androidブランドを前面に押し出してきた。Phone & Tablet/Wear/TV/Auto/PlayとすべてAndroidブランドを中心に展開
- クラウドをとても重要視。クラウド開

« 会場となった米国サンフランシスコMoscone CENTER WEST

» 会場入り口に設置された色の変わる巨大なGoogle I/Oロゴ



« キーノート会場に設置された巨大な機械式カウントダウン時計

» 基調講演会場に映し出されたスローガン “Keep building.”

※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。

発環境や安価に利用できるビッグデータ解析基盤なども（ただしクラウド関連は、2014年3月にクラウド専門のイベントが開催されており、重要な事柄は発表済み）

テレビとの融合を進めるAndroid TV、車載でもAndroidを推進するAndroid Auto、腕時計型ウェアラブル端末Google Wearや、新しいデザイン思想Material Designの発表など盛りだくさんでしたが、それとは対照的にGoogleが推進するSNSであるGoogle+は、ほとんど話題にのぼりませんでした。また、Gmail関連APIの充実や、YouTubeではフルハイビジョン映像の48fps/60fps対応に関する発表など、Google I/Oの各種発表の影に隠れてしまった重要な発表もいくつかありました。

Google I/Oキーノート（基調講演）会場に掲げられていたスローガン「Keep building.」の文言が、開発者たちに向かっていろいろなものを作り続けてほしいという想いを込めているとともに、Google自身も、今もいろいろなものを作り続けており、これからも作り続けるのだという意思表示として強く感じられた2日間でした。

巨大な会場と興味深いセッションの数々

今年のGoogle I/Oのセッションは、各テクノロジごとに分類されたものではなく、目的や環境ごとにまとまった話の内容に変化してきていました。その理由は、ある1つの技術だけで何かサービスが作れるわけではなく、何かユーザに価値のあるサービスを提供しようと思ったら、さまざまな技術の組み合せで作り上げられるからだという話でした。

Google I/Oセッションビデオ一覧

https://www.google.com/events/io/io14videos

動画は英語字幕を一緒に見ると理解しやすいと思います。また現在は機械翻訳で日本語字幕を見ることもでき、今後日本語翻訳字幕も予定されているそうです。I/O Bytesと呼ばれる5分前後のダイジェスト映像も含めると、全部で180本もの動画が公開されています。

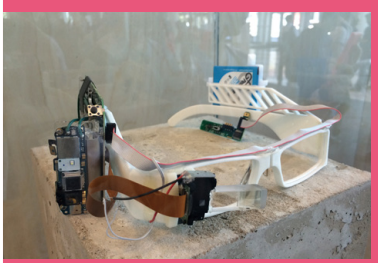
今年の一番人気は、段ボールの組み立てキットとして配布された、Google CardBoardでした（https://developers.google.com/cardboard/）。CardBoardとは英語で「段ボール」を示す言葉です。安価で高性能なVRメガネOculusがFacebookに約2,000億円で買収されたことを相当意識しているらしく、コストは20ドルだけれども結構な品質で立体視YouTube動画やGoogle Earthを立体で見られます。そこそこのクオリティで没入感のある体験が段ボールとスマートフォンでできており、スマートフォンの地磁気センサーを騙（だま）してスイッチにするための磁石や、NFCタグも装備されています。

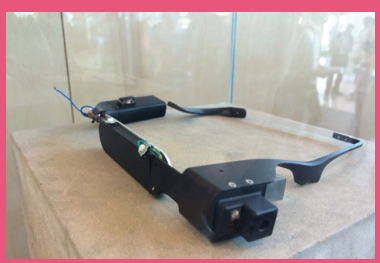
CardBoardの設計図やレーザーカッター用の図面は無料で公開されています。工作の得意な人であれば、薄手の段ボールと、Amazonや百円均一で入手できるレンズや磁石などを利用して同等品がすぐに作れるそうです。また、インターネットで同等品を販売し始めたところもあります。

DoDoCase CARDBOARD VR TOOLKIT

19.95ドル（NFCタグ搭載版は24.95ドル）

http://www.dodocase.com/products/google-cardboard-vr-goggle-toolkit

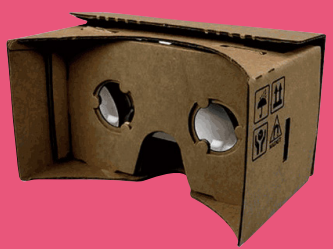




会場ロビーに展示されたGoogle Glassの試作機

現在のGoogle Glass。度付きレンズ用のフレームも会場で販売されていた

展示会場風景。ジャンルごとのミニセッション会場、コードラボと呼ばれるプログラミング講座なども



Google CardBoard：段ボールで作られた安価なVR（仮想現実）用メガネ。スマートフォンを入れて利用します。Android用デモアプリ、デモアプリのソースコード、専用アプリ開発用のSDK、ブラウザ用立体表示プラグインなどが提供されています。
https://developers.google.com/cardboard/

Googleが推し進める、文脈理解（コンテキストアウェアネス）

今回のGoogle I/O 2014、とくに新しいAndroid環境や、車載環境Android Autoなどで盛んに取り上げられていたのは、「コンテキストアウェアネス（Context Awareness：文脈理解）」という言葉でした。現在は、さまざまな検索キーワードを検索エンジンに入力して、検索結果から人が適切なものを見いだしています。将来的には、検索せずとも適切な情報を提示することを目指しており、Google Nowの提示する各種の情報がその先駆けとなっています。

今後は、たとえばスマートフォンに地図を表示した状態で、車に乗ってカーナビにスマートフォンを接続すると、スマートフォンで表示していた地図を使ってナビを開始。スマートフォンをスリープ状態にしてカーナビに接続したときは単に充電するだけ。あるいは、親が車で子供を保育園に迎えに行くときは大人向けの音楽が流れるが、保育園に到着し、車のドアが開閉して乗員が増えたことがわかると、子供用の音楽に切り替わるなど、その場の状況や履歴をうまく活用した振る舞いが自然にできるよう、いろいろと考慮し始めているそうです。

コンテキストアウェアネスを支える技術としては、ある領域に侵入したら通知するジオフェンシングやセンサー群による移動状態（徒歩、自転車、車、電車）を自動で見分ける手法がすでに広く使われています。

Googleの技術は生活のインフラとしてなくてはならないものになりつつあります。Googleもさまざまなライバルたちとしのぎを削りながら、そしてGoogle内部の技術者だけではなく、世界中の開発者がGoogleの技術を活用して、多彩なアイデアを実現していく状況が当分続きそうです。SD

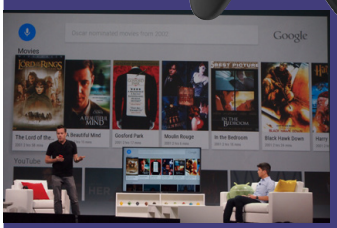
GADGET 1

Android TV

http://www.android.com/tv/

ゲームコントローラでスムーズに操作できるセットトップボックス

Google TV以降、3度目の挑戦となるAndroid TVが発表されました。テレビのHDMI端子に接続し、ゲームコントローラまたはスマートフォンのリモコンアプリで操作する、小型のセットトップボックスです。Google Playで購入した映画やドラマ、YouTubeの動画などを一括検索して楽しむことができます。Android TV専用のゲームアプリを楽しんだり、さらにChromecastの機能も包括しています。



GADGET 2

Android Wear

http://www.android.com/wear/

広がりが期待される、腕時計型ウェアラブル端末

腕時計型のAndroid搭載ウェアラブル端末として、LG、Samsung、Motorolaから発売されることが公にされ、LGとSamsungの四角い画面のタイプは、日本からも2万円強でオンライン購入できるようになりました。Motorolaの丸いタイプは夏の終わり頃、倍くらいの価格で発売予定とのこと。Androidスマートフォンと連携させて使うのが前提で、スマートフォンに届く通知情報を確認したり、Wear専用アプリを動作させたりできます。サービスとアプリによって、さまざまな用途への展開が期待されます。

GADGET 3

Project ARA

http://www.projectara.com/

モジュール組み立て式スマートフォン

プロジェクトARA（アラ）として進行中の、機能モジュールを組み合せて必要な機能／スペックのスマートフォンを組み上げることができる機器です。仕様が公開されており、試作が進んでいます。会場では現在試作中のプロトタイプ上でAndroid OSが起動した映像が公開され、拍手喝采を浴びていました（起動直後にフリーズしてしまいましたが……）。利用シーンに応じて性能の違うカメラモジュールを付け替えたり、バッテリーを付け替えたりできれば、よりフレキシブルにスマートフォンが活用できそうです。

GADGET 4

Project TANGO

https://www.google.com/atap/projecttango/

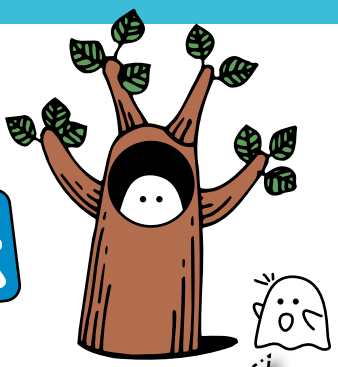
3D距離センサーつきスマートフォン&タブレット

Project TANGOはスマートフォンに人間と同じような空間認識力をつけさせようというプロジェクトで、Microsoft Kinectのような距離（深度）センサー、モーショントラッキングカメラ、高解像度カメラを搭載するスマートフォンやタブレット端末をリリースする予定です。店内に並ぶ商品棚の位置を把握してAR（拡張現実）情報を提示したり、室内のストリートビューのような映像を生成したり、さまざまな挑戦の最中だそう。

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

16

結城浩の再発見の発想法



Scalability

Scalability——スケーラビリティ

スケーラビリティとは

スケーラビリティ(scalability)とは、**システムが処理すべき情報の規模が大きくなったときに対処できる能力**のことです。スケーラビリティはスケール(規模)＋アビリティ(能力)と分解でき、「規模に対処する能力」という意味になります。

典型的な例としてWebサービスのスケーラビリティを考えてみましょう。Webサービスをリリースした直後、ユーザがまだ少ないうちは軽快に動くけれど、ユーザが多くなってくると反応が鈍くなるのはよくあるトラブルです。

このようなWebサービスは、規模(ユーザ数)が大きくなると対処できなくなったので、「スケーラビリティが低い」と言えます。逆に、ユーザが多くなってもずっと同じ反応速度を保っていられるWebサービスは、「スケーラビリティが高い」と言えるでしょう。

Webサービスにとってスケーラビリティは重要です。ユーザの数が多くなったということは、Webサービスに人気が出たことを意味します。そこできちんと対処できなかったら、せっかくのチャンスをみすみす逃してしまうことになりますね。

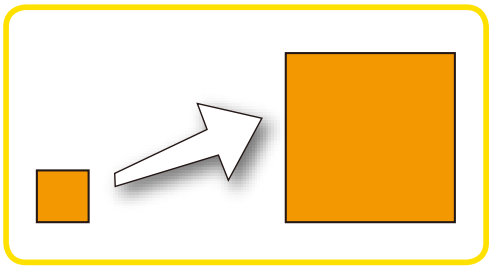
スケールアップとスケールアウト

スケーラビリティを高める方法として、大きく「スケールアップ」と「スケールアウト」という2つの対処方法があります。

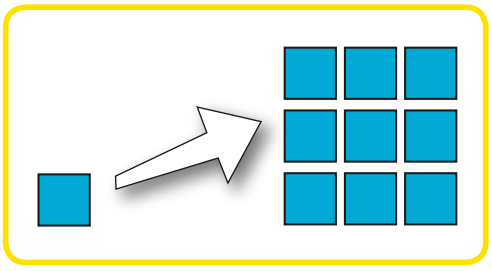
スケールアップは、性能を上げて規模の増加に対処する方法です(**図1**)。Webサービスの例で言えば、性能の高いマシンを購入したり、ネットの帯域を広げたりして対処するのはスケールアップの一種です。これは自然な考えですが、性能を上げるとコストもかかるので注意が必要です。ユーザが少ないうちから高い性能のシステムを構築してしまうとコストの無駄になるでしょう。

スケールアウトは、性能を上げるのではなく、

▼図1　スケールアップ



▼図2　スケールアウト



システムを多数組み合わせることで規模の増加に対処する方法です(**図2**)。Webサービスの例で言えば、クラウド上の仮想サーバを使って実装し、ユーザ数が多くなって負荷が増大したら、インスタンスを増加させて対処するのはスケールアウトの一種です。この方法なら、ユーザが少ないうちはインスタンス数を少なく抑え、ユーザが多くなったらインスタンス数を多くすることで対処できるので、コストの無駄も少なくなるでしょう。

とくに、最近のWebサービスへのアクセスは、TwitterなどのSNSの影響で急激に増大することがあるので、急激な負荷増大に動的に対処するのは大切なことです。

問題の性質とスケーラビリティ

スケーラビリティが高いシステムを構築するためには、対処する問題を詳しく調べる必要があります。スケールアウトしてスケーラビリティを上げるためには、大きな問題を、独立した小さな問題群に分割する必要があります。

たとえば、Webサービスの例で考えてみましょう。ユーザがいくら増えても、Webサービスへのアクセスはユーザごとに独立ですから、複数のインスタンスに振り分けることが容易です。

しかしながら、もしもWebサービスの背後で動作しているデータベースのスケーラビリティが低いと、そこにアクセスが集中してしまった場合、結局スケーラビリティは低くなってしまいます。これはこの連載でも以前お話しした「ボトルネック」がスケーラビリティを低くしてしまう例です。

組織とスケーラビリティ

技術の世界だけではなく、日常生活でもスケーラビリティは重要な概念です。

たとえば、会社などの組織は、個人で対処できない規模の問題を解決するために存在します。個人が能力を高めて問題に対処するのはスケールアップによる問題解決で、組織の中にいる複数の人間が問題に対処するのはスケールアウトによる問題解決と言えるでしょう。

事業が成功して大量の仕事がやってきたとき、うまく「組織が回る」かは組織のスケーラビリティが試されていると言えます。

大きな問題を、独立性が高い小さな問題に分割できるなら、いわゆる「人海戦術」が使えます。とにかく人をたくさん投入すれば早く解決できる場合ですね。しかし、いつも人海戦術に頼るわけにはいきません。

組織には、大きな問題を独立性の高い小さな問題に分割する人と、分割した問題を解く人の両方が必要です。そして経営者は、Webサービスの設計のように、組織のスケーラビリティが高くなるよう設計する必要があります。

個人とスケーラビリティ

私たち個人も、自分のスケーラビリティを考える意味があります。仕事であれ、プライベートであれ、私たちは多くの問題に直面します。これまでうまく対処できていたのに、問題の規模が大きくなると急に対処できなくなるのはよくあることです。

その場合、自分の能力をアップして何とかしようと考えるのはスケールアップの発想です。それは有効ですが、コストや時間がかかりますし、限界もあるでしょう。

ほかの人に助けを頼んだり、専門家に任せたり、あるいは同じ問題を抱えている人が集まって集団として対処することもあるでしょう。これは、スケールアウトの発想です。

手に負えないトラブルが生じたとき、スケールアップだけではなく、スケールアウトで対処してもいいと気づくのは大切です。

あなたの周りを見回して、処理すべき規模がときに大きくなる問題を探してみましょう。その問題はスケールアップとスケールアウトのどちらで対処すべきでしょうか。そのときにコストや時間はどうなるでしょうか。ぜひ考えてみてください。SD

enchant ～創造力を刺激する魔法～

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

第17回（最終回） 新たなる挑戦

enchantMOON S-II

新たに設計しなおされたMOONPhase 2.9.0は、まるで同じハードとは思えないくらい高速に動きました。

実は僕は開発現場から離れている間も、動作が遅いという欠点をソフト以外の観点からなんとかできないか検討を続けていました。CPU世代を一世代新しくCortex-A9にしたり、それをデュアル化、クワッド化して動作を検証しました。しかしいくらCPU世代を上げても、クロック周波数を上げても、バージョン2.8までのMOONPhaseは高速に動作しませんでした。それどころかチューニングしていないぶん、却って遅くなるケースさえありました。

つまりこれはハードウェアの問題に見えがちですが、実際にはソフトウェア設計の問題だったのです。だから次世代機を開発する前に、完全な形でソフトウェアを再構築する必要があると僕は強く思いました。

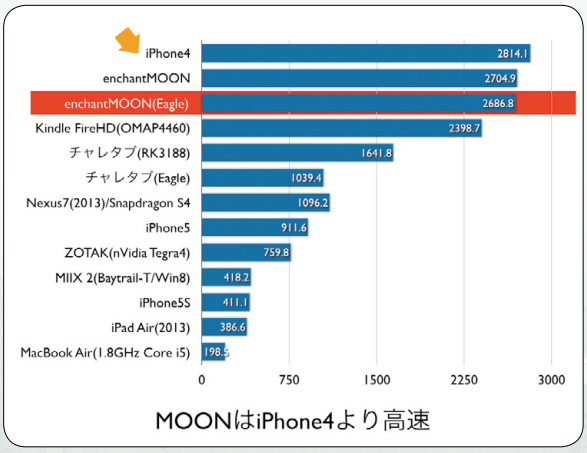
現場との激しい軋轢があるなか、僕はできるだけ現場のエンジニアが納得し、自分たち自身が「このままのやり方では根本的にダメなのだ」ということに気付いてほしいと願っていました。しかし結局のところ、僕が思うようには事態は推移しませんでした。当の開発者たち自身が、「このハードじゃあこんなもの」「これが限界」と思い込み、根本的な最適化を諦めていたからです。「ソースコードをすべて捨て、ゼロから再構築しよう」

数ヵ月ぶりに開発者たちと顔を合わせた僕は、まずベンチマーク結果を見せました。

「いいか？　みんなenchantMOONは遅いという。ハードがダメなのだと。だが本当にそうか？

ベンチマークをとってみた。それがこれだ。実際にはiPhone 4より速く、iPhone 5の3倍遅いだけに過ぎない。では聞くが、iPhone 4はそんなに遅かったか？」

皆、「遅い」という言葉に囚われ、それが「どのくらい遅いのか」まで考えたことがなかった



ベンチマーク結果（SunSpider 1.0.2 JavaScript Benchmarkを使用）

のです。そしてそれがハードウェアの限界なのか、ソフトウェアの限界なのか、切り分けをきちんとしてきませんでした。なにしろドキュメントもろくにない中国製のハードウェア上で、OSをきちんと動かす、ただそれだけで想像を絶するほど大変だったからです。

「なにが遅いのか？　アーキテクチャだ。もっと言えば、ベクトルの読み込みと書き込みだ」

enchantMOONでは、すべてのベクトルデータをJavaScriptから扱いやすいようにJSONデータとして保存しています。しかしJSONデータとは、要はテキストファイルです。テキストファイルを読み込むにはパーサーを通さなければならず、これでは遅いのは道理です。しかも、ページごとにすべてのベクトルデータを保存しています。この構造を持っている限り、実用的な速度でenchantMOONを動かすことはできないと考えました。実際に計測してみると、ページ切り替え時に待たされる時間の大半が、JSONの解釈に費やされていました。

「このJSONを毎回すべて読み込む方式をやめ、JSONを適宜追記していく方式に変更する。認識が必要になったときに、改めて該当する部分のJSONのみを読み込み、ベクトルデータとして再解釈する。データの基本構造をピクセルバッファとし、表示にはピクセルを使う。つまり、データの本質的な内容はJSONで保存するが、処理するときにはピクセルを表示する。それがもっとも高速なはずだ」

僕が提案したのはハイブリッドのアプローチです。

「これなら高速化できるはずだ」

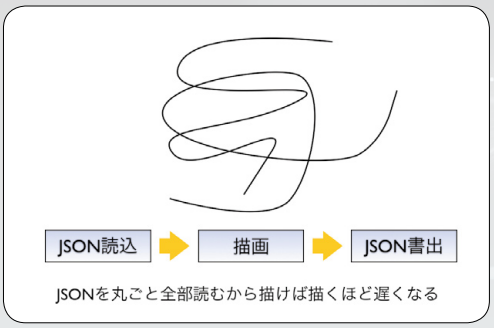
「しかしそれだとソースコードをすべて捨てて、イチから作りなおしになります」

「そうだろうな」

「そんなことをしてもいいんですか？」

「するしかない。それがMOONを蘇らせる唯一の方法なら」

こうして前代未聞の発売後半年経過してからのOSソースコード全破棄を言い渡しました。



遅い原因

メーカーから転職してきた開発者たちにしてみれば、「理屈上それが正しいとわかっていても、メーカーでは決して許されない判断」なのだそうです。しかし僕にとって重要なのは、すでに出荷した製品の性能が劇的に向上するならユーザに評価してもらうことであり、それを短期的な利益やしがらみを優先してやらないという選択肢は考えられませんでした。

その結果、完成したenchantMOON S-II（MOONPhase 2.9.0）はまるで別物、新機種と言っても信じられてしまうほどに高速化しました。たとえばかなりの書き込みデータがあった場合、旧バージョンでは1ページめくるのに40〜50秒程度かかっていた処理も、0.5秒で処理できるようになりました。単純計算で100倍です。過去、どのような歴史を紐解いても、発売から1年以内に、ソフトウェアアップデートだけで100倍高速化したハードウェア製品は存在しないと思います。

もちろん、それだけ最初の製品の完成度が物足りないものだったと後から言うことは簡単ですが、それを実際に成し遂げた開発者たちには、僕は感謝の気持ちしかありません。

そしてGeminiを使ったインビテーションサイトを用意したS-IIの発表会では、MOONPhaseを再構築するに至った思考プロセスを丁寧に説明しました。無料のバージョンアップでなぜ記者会見を開いたのか。単なるバージョンアップではなく、かつてない規模で行われた本当の意味でのアップグレードである、というこ

とを直に伝えたかったのです。

つかの間の休息、そして

しかし現場はそれからの2週間、まさしく戦場と化しました。大量のデバッグに追われていたからです。

そしてついに4月10日、enchantMOON S-IIが正式にリリースされます。予定していた正午のリリースの直前に大きなバグが1つ見つかり、それを修正した後、少し遅れてのリリースとなりました。

「よし、みんなよく頑張った。今から打ち上げに行こう。もちろんオレのおごりだ」

疲れきっていた開発者たちは、ぞろぞろと淡路町の中華料理店に集まりました。食べ放題の中華料理。北京ダックもあります。

「酒もいくらでも飲んでいいぞ。今日までよく頑張った」

彼らは互いを讃え合い、つかの間の休息を満喫しました。そう。これはほんのつかの間の休息に過ぎませんでした。

その日の夕方、僕はふたたび開発者たちを会議室に集めました。

「さて、S-IIは本当に素晴らしい仕事だった。よく頑張ってくれた。君たちは素晴らしい」

僕は開発者たちを一人一人の顔を確認しながら、言いました。彼らは確かに疲れきっていました。しかし彼らの瞳の奥には、まだ光が残っていました。

発表会の様子

「これで終わりってわけじゃないんですよね？」

サターンVと呼ばれる、enchantMOONの基礎となるOS開発を担当した濱津誠が薄笑いを浮かべて言いました。

「なにか新しい修羅場が始まるって聞いてますが……」

大手電機メーカーから、enchantMOONをやりたくて飛び出して来た日高正博も苦笑いします。

「ふむ」

ドワンゴ時代から14年以上、常にもっとも重要なプロジェクトの開発者として僕を支えて来てくれた布留川英一は、僕と目が合うと、いつものようにポーカーフェイスで頷き、その隣でenchant.jsとMOONBlockを開発した高橋諒も口を真一文字に結んで頷きます。インターン時代にEagleVMを開発した凄腕のケヴィン・クラッツァーは、正式にビザを取得し、ドイツから日本に移住してきていました。彼は不思議そうにこの状況を眺めています。これから何が始まるというのだ、という顔です。

僕は薄ら笑いを浮かべました。密かに進めてきた企画が、いまこそ実現するその寸前なのです。enchantMOON S-IIは、その壮大な計画のほんの端緒に過ぎないのです。そして次に待ち受けるのはまさしく真の修羅場です。人類の誰もやろうとしたことがないような、まったく前例のないものを創りだそうというのです。そしてなによりそれは、僕自身がもっとも欲している新しい「道具」でした。

「新しい修羅場……でしょうね、また」

新卒から7年、僕の右腕として常に働く増田哲朗も、真顔とも笑顔ともつかない顔で頷きます。その様子を、MOONPhaseのコンセプトを考え、今はマネージャーとして開発現場の統括を行う辻秀美と、企画担当で新たに参加した渋江さやかは楽しそうに眺めていました。

「そのとおりだ。明日から我が社の歴史上、もっとも重要なプロジェクトに合流してもらう」

やっぱりか、という空気が流れます。

「それはこれまで世界の誰も見たことがないソフトウェアだ。想像したことすらない。だが誰しもに欲しいと思わせる、本物のソフトウェアだ。それを見せよう」

そこで僕は別働隊に密かに作らせていたビデオを見せました。それは僕が密かに温めていた企画を、樋口監督に紹介していただいたとあるアニメ制作会社に依頼して映像化したものです。

新しい企画のコンセプトをまず映像にして見せてしまう、画面から、動きから、使い方から、すべてです。誰もが製品の最終形を最初にイメージできるように、最初に徹底的に動きを作ってしまう。今回はそういうやり方で行こうと僕は決めていました。だからS-IIの開発を監督するその一方で、密かにまったく新しい企画を練り上げて作っていたのです。その場にいる数人を除けば、誰もその存在すら知りません。誰も見たことがない、まったく新しい画面です。

「ははははは……」

濱津は笑い出しました。ほかのメンバーもなにか薄ら笑いを浮かべました。

「これ、凄いですね。いったい誰が作るんでしょう？」

僕は一同を見回し、それから言いました。

「僕たちだ」

僕にはわかっています。enchantMOON S-IIを創りだした僕たちなら必ず素晴らしいソフトウェアを作り上げることができると。そう、彼らではなく、僕たちなら、できるのです。

月へ

僕たちの挑戦はまだ道半ばにあります。これからも数々の困難が待ち受けているでしょう。

コンピュータを実用的な文房具にまで落とし込む——なぜならそこはかつて多くの人々が挑み、そして散って行った本当の未踏領域だからです。そのために、ハードウェア、OS、プログラミング言語まですべてを自分たちで作る。そんな馬鹿げた挑戦を、社員数たったの100人、売上高わずか14億円の小さな会社が行うのは、やはり無謀でしょうか。

しかし一方で、enchantMOONを始めとするプロジェクトは僕たちの夢であり、希望そのものなのです。enchantMOONは、多くのユーザの皆様と、クライアント企業の皆様と、enchantMOONにかかわる社員たちと、それを支えるすべての社員、アルバイトたち。そうしたものによって支えられている、ひとつの大きな夢です。

もし万が一、道半ばで資金が尽きたらどうしよう、会社がなくなってしまったらどうしよう。

僕は株式会社ユビキタスエンターテインメントの社長として、そして清水亮という一人の技術者として、ひとつだけ皆さんにお約束します。それは僕たちは決してこの夢を諦めたりしないということです。なにより僕自身がこの夢に魅了（enchant）されてしまっているからです。

そしてその先にどんな未来が待ち受けているのか、それは神のみぞ知ります。もしかしたら僕たちは奈落の底に突き進んでいるのかもしれません。先のない技術、生涯ものにできない夢を追い求めているのかもしれません。しかし、僕たちは前進を続けます。その先にどれだけ困難があろうとも、僕たちはかならずそれを乗り越えてみせます。

そしてすべての人々がコンピュータを意のままに操り、すべての人々がプログラミングをビデオの録画予約程度にまで簡単に行えるようにする。すべての人々がコンピュータを通して自分の意志や考えを表明し、自分の能力をコンピュータによって無限に拡張していける。そんな世界を実現するため、たゆまぬ努力を続けて行きます。

最後に、この連載を企画してくださり、ページを与えてくださったSoftware Design編集部様に感謝致します。

そして読者の皆様とまたどこかでお会いできる日を楽しみにしつつ、筆を置きたいと思います。SD

軽酔対談

かまぷの部屋

第2献　ゲスト：永淵恭子さん

㈲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

永淵恭子さん

株式会社サーバーワークス　クラウド女子会コアメンバー。美人CDP認定。Redshift Girls。福岡県出身。営業を希望して早3年。自分のあとに新卒女子が入社しないことが目下の悩み。「技術をもっと楽しく！営業ももっと楽しくわかりやすく！そしてAWS女子を増やしたい！」を最近のモットーに日々奮闘中。
https://www.facebook.com/gyori.n

Mac使いで、ショートカットキーに慣れているそう。

こんにちは。USP研究所のかまぷです。今月もほろ酔い気分でゲストをおもてなししながら、お話を聞いてみます。

（鎌田）――ぎょりさん（永淵さんのニックネーム）とは女性向けのイベント（TechGIRL）で初めてお会いしました。元気のいいLightningTalk（LT）を見て、今すぐ友達になりたいって思いました。サーバーワークスで営業職をされているとのことなのですが、この業界、女性の営業職って少ないですよね？

（永淵）そうでしょうか？　あまり気にしたことがありません（笑）。自分が会社の売上に貢献しているほうなので、むしろ女性のほうが向いていると思っています。私は新卒で営業を希望し、それがかなって営業配属になりました。その理由は、営業のキーである数字や数学が好きなのと、いろんな人に会えると思ったからです。とくに、IT業界の営業ですと、お客さんの業界はさまざまです。実際に会って自分の知り得ない新しい情報を聞く、逆に自分からもキャッチアップできた仕事の情報は常に提供する、ということに喜びを感じています。

――入社数年で会社の売上に貢献しているってすごいですね。コツはなんでしょうか？

楽しんでやることですかね！　今期の成績がよかったので、これを維持しなきゃというプレッシャーもあります。営業は理系が向いていると思いませんか。自分はもともと数字を追うのが好きなので、グラフを作ったり、必要な数字をまとめたりするのはまったく苦ではないんです。営業は努力した分、利益として数字で返ってくるのがおもしろいです。AとB、2つの仕事があったら、どちらが長期的に売上へ貢献するか？などなど、常に頭を働かせています。

――とっても元気なのですっかり体育会系かと思っていました！　営業職なのにAWSの資格をお持ちだと聞きましたが。

SA（Solution Architect - Associate）を持っています。確かに営業職で持っている人は少ないと思います。私はJAWS-UGなどの勉強会でエンジニアの方と接することが多いので、資格を勉強することは自然でした。Twitterなどネットで情報を仕入れるのも好きです。それから表舞台に立つのも好きですよ。LTや講演など機会があったら手を挙げるようにしています。先日はRedshift Girlsというユニットを作って巫女さんの衣装を着て皆に楽しんでもらえる工夫をしました。本気で楽しくプレゼンするにはコスプレですよ！（笑）

――とても楽しそうですね。ちなみにJAWS-UGはAWSユーザの日本のローカルコミュニティですが、何人ぐらいいらっしゃるんでしょうか？

具体的な数は把握していないのですが、Facebookで先日600「いいね」をカウントしました。東京支部から沖縄支部まで地域ごとにグループがあるので、勉強会などが日本全国各地で行われています。一緒に勉強する仲間を増やしたいと思っているので、読者のみなさんにもぜひ勉強会に参加してもらいたいです。このコミュニティから産まれたものでCDPというものがあります。




——CDPとは何の略ですか？

AWSのサービスはおよそ40ぐらいあります。CDP(クラウドデザインパターン)はそれらのサービスの設計や機能を使いこなすためのノウハウを整理したものです。すでに多くで活用されていて、書籍も出ています。ちなみに美人CDP認定をいただきました(照)。私が個人でも使っているもので、「Amazon Simple Storage Service(S3)」というストレージサービスがあります。これは個人ユーザに向いているかもしれません。あまり知られていませんが、DropboxはS3をラッピングしているものなのですよ。中身はAWSということです。

——認定受けているだけあって、詳しいですね(笑)。話は変わりますが、営業職だと出張はあるのでしょうか？

営業は都内が多いですね。毎日歩き回っています。平均訪問件数は1日3件ぐらいです。クラウド系のイベントが地方で行われるときに、出張することがほとんどですね。2、3ヵ月に1回ぐらいなので、そんなに多くないと思います。先ほども触れましたが、出張先でRedshift Girlsとしてプレゼンしてきました。

——Redshiftはあまり聞き慣れない単語ですが、どういう意味ですか？

単語そのものの意味は「赤方偏移」ですよね。宇宙が広がっていくイメージでしょうか。AWSが提供するData Ware House専用のサービスです。初期投資なしで始められるので、大絶賛PUSH中です。Redshift Girlsを始めたきっかけは、とある人がJAWS-UGの勉強会に「Red

Bull Girlを呼びたいな〜」という声を上げ、これ対して、弊社の上司が冗談で「"Redshift Girls" なら、なんとか呼べるよ！(うちの女子社員だけど)」と、即席アイデアを提案したという経緯なのですが、その当時はRedshiftもわからずに必死になって、一から勉強したことは今となってはいい思い出です。勉強は営業に限らずどの職種でも必要です。でも、息抜きも大事ですよね(笑)。

——そうそう、お酒好きそうですよね。お好きな食べ物はなんでしょうか？

ビール、日本酒、芋焼酎……。コロッケやチャーハン、塩昆布キャベツなどなど……居酒屋で出てくる料理が好きですね。今日のチキンも、ビールに合いますね！　美味しい♡

——家でお料理されたりするんですか？

たまにするくらいです。おつまみを作ったりとか(笑)。食べるのは生まれつき速いので、男性と一緒に食事しても同じか先に食べ終わります。でも猫舌なのでラーメンなどになると、とたんに遅くなります(笑)。

——最後に、クラウド女子と営業女子を増やすためにメッセージいただけないでしょうか。

女性は営業に向いていると思うんです。とくに、人一倍早く情報仕入れたり、とにかく良さそうなものを試したり、女性はマルチタスクな思考が得意だと思うのです。それは鍛えようと思って身に付くスキルではないので。また、話がポンポン飛ぶような話し方をしますよね。このことは良く捉えるとアイデアが湧き出ているということだと思うのです。仕事でもロールモデルが大事だと思っているので、自分がロールモデルになろうかと思ってます(笑)。

最後に一言言わせてください。JAWS-UGではクラウド女子勉強会をやっているので、ぜひ、周囲の女性に教えてあげてください。勉強の機会や仲間を増やす機会を共有してほしいです。初心者も大歓迎です！

——素晴らしい！　私もいろいろ頑張りたいです。今回は、楽しいお話ありがとうございました！ SD




はんだづけカフェなう

続・BLEで遊んでみよう

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

はじめに

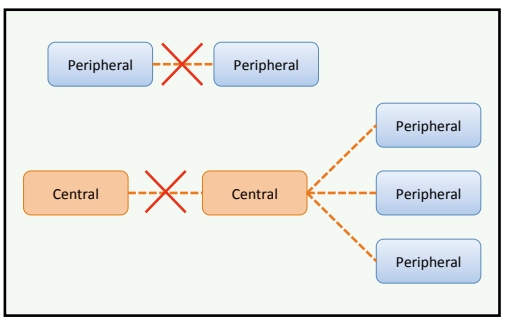
前回に引き続き、今回もBluetooth Low Energy（BLE）の世界を紹介します。今回は、BLEの技術的な詳細にもう少し踏み込んでみたいと思います。Androidについては筆者が動向をよく把握していないため、iOSを中心に話を進めます。

セントラルとペリフェラル

BLEには、大まかにいって、セントラルとペリフェラルという役割が存在します。セントラルとペリフェラルは、ちょうど親機と子機のような関係になっています（図1）。子機に相当するペリフェラルは、アドバタイズと呼ばれる同報送信（ブロードキャスト）を行い、ペリフェラルの名前や提供できる機能などを広告（アドバタイズ）します。一方、セントラルは、アドバタイズの検出を行い、接続したいペリフェラルに対して接続要求を行います。

セントラルどうしや、ペリフェラルどうしは接続できません。セントラルは、複数のペリフェラルと通信できます。iPhoneはセントラルにも、ペリフェラルにもなることができます。一方、先月紹介した、mbed HRM1017は、現在のところ、ペリフェラルにしかなることができません。この役割や、アドバタイズ、接続を司るのが、Generic Access Profile（GAP）と呼ばれるプロファイルです。

▼図1　セントラルとペリフェラル



サービスとキャラクタリスティック

先ほど「提供できる機能」と書きましたが、この機能を表すのがサービスです。Bluetooth Smartデバイスには複数の機能、たとえばBLE体温計であれば、体温計と、体温計の電池残量、といった具合に複数の機能を搭載することができます。サービスは、この「体温計」や「電池残量」といった個々の機能を表します。このサービスの分け方を理解するには、少し慣れが必要かもしれません。しかし、Bluetooth SIGが、いくつかのサービスを公開しています[注1]。これらを眺めると、なんとなくBLEのお作法が見えてきます。このページでは、すでにHealth Thermometer（体温計）や、Battery Service（電池残量）といった具合に、一般的に使われそうなサービスが標準として公開されています。こういった定義されているもののほかに、開発者が独自にサービスを定義することもできます。

このページで、Health Thermometer（体温計）というサービスを見ると、Service Characteristicsという項があります。ここを参照する

注1）https://developer.bluetooth.org/gatt/services/Pages/ServicesHome.aspx

と、Temperature Measurementなど、いくつかの項が並んでいます。これがキャラクタリスティックです。つまり、体温計サービスの下には、温度というキャラクタリスティックなどが定義されています。サービスは複数のキャラクタリスティックを持つことができます。サービスやキャラクタリスティックは、GATT（Generic Attribute Profile）と呼ばれるプロファイルで規定されています（**図2**）。

セントラルは、ペリフェラルのキャラクタリスティックを読んだり書いたりして通信を行います。たとえば、iPhoneと接続したBLE体温計であれば、セントラルであるiPhoneが、BLE体温計の体温が収まっているキャラクタリスティックから値を読んで、iPhoneの画面に表示するといったことを行います。

先ほどのHealth Thermometerのサービスが記述されているページを見ると、「Assigned Number」として、0x1809という値が記されています。この値は、サービスやキャラクタリスティックを識別するためのもので、UUID（Universally Unique Identifier）と呼ばれます。UUIDは、本来128bitの値ですが、このHealth Thermometerのようによく使われるものには、16bit長の0x1809がAssigned Numberとして割り当てられています。このAssigned Numberは、“00001809-0000-1000-8000-00805F9B34FB”という128bitあるUUIDの一部を切り出したものです。先ほど開発者が独自にサービスを定義できると記しましたが、このような独自のものは、128bit長のUUIDを使わなければなりません。

参考になる書籍

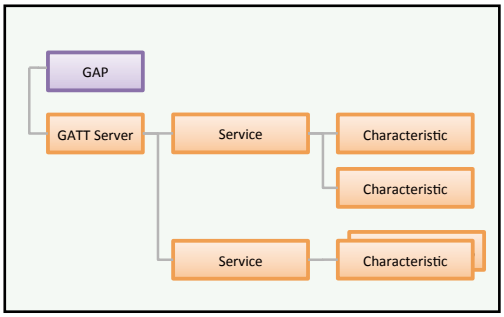
ここまで、とてもざっくりとしたBLEのしくみの説明を行いましたが、BLEのもっと詳しいところを知りたいという読者のために書籍を紹介しておきたいと思います。和書では良い本を知らないのですが、オライリーの『Getting Started with Bluetooth Low Energy（**写真1**）』[注2]と、Prentice Hallの『Bluetooth Low Energy』[注3]の2冊です。まず読む1冊を選ぶのであれば、筆者はGetting Started……のほうをお勧めします[注4]。

iOSでの開発

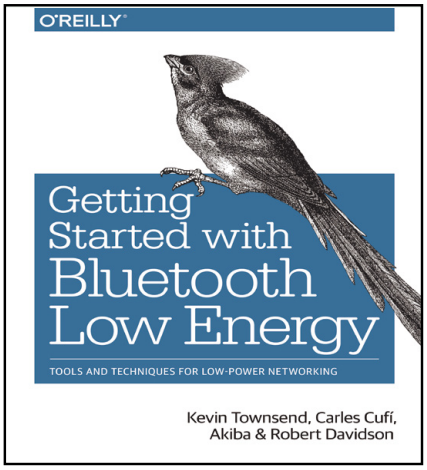
iOSでBLEの通信を行うときには、Core Bluetoothフレームワークを利用します。このCore Bluetoothフレームワークは、BLEの低レイヤの部分を抽象化してくれますので、BLEの深い知識を必要とせず開発を行うことができるようになっています。とはいえ、iOSアプリケーション開発には、それなりのハードルがあ

注2） http://shop.oreilly.com/product/0636920033011.do
注3） http://www.amazon.co.jp/dp/B009XDA1G8/
注4） 筆者はこちらのほうが眠くなりませんでした（笑）。

▼図2　GAPとGATT



▼写真1　『Getting Started with Bluetooth Low Energy』



ります。こういったハードルを避けて、手軽に遊ぶソリューションの1つに「konashi（**写真2**）」があります。

konashi[注5]は、gihyo.jpの記事[注6]でも紹介されているように、Objective-Cでアプリケーションの開発を行うことができます。それ以外にも、JavaScriptを使って開発を行うことができます。jsdo.itというWebサービスに保存したJavaScriptを「konashi.js」というiOSアプリケーションで実行できます。konashiのWebサイトで販売されている10,260円のボードを買ってくるだけで、手軽にJavaScriptでBLEを使ったスマートフォンとマイコンの連携を始めることができます。

前回、少しだけ紹介したtechBASICも、Objective-Cを使わずにiOS側のアプリケーションを書く手段です。konashi.jsと同じように、iOSのアプリケーションの上でBASICを実行できます。techBASICは、エディタ機能もあり、iOS上だけで開発を行うこともできます。が、やはりスマートフォンの上でコーディングするのは辛いので、パソコンで書いたものをコピー&ペーストするのが良いでしょう。techBASICについては、BLEのサンプルコードが付属していますので、それを参照するのも手ですが、オライリーの『Building iPhone and iPad Electronic Projects（**写真3**）』[注7]という本が出ています。techBASICや、この本は、TI（テキサスインスツルメンツ）のSensorTagというBLE評価ボードを対象に書かれています。先述のとおり、BLEはUUIDでサービスやキャラクタリスティックを識別しています。サンプルコードや同書の内容を参考に少し試行錯誤すれば使い方を覚えることができます。techBASICでのBLE関連情報は、micono氏のROBOMICというブログ[注8]にさまざまな情報が記されています。micono氏は、前回紹介したmbed HRM1017にtechBASICから接続するサンプルプログラムをさっそく作ってくださっています[注9]。このプログラムをmbed HRM1017とtechBASICそれぞれに書き込んで実行をすると、動作することが確認できました（**写真4**）。

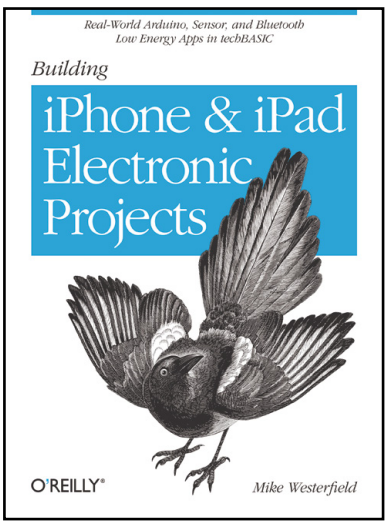
iOSでの開発ではありませんが、micono氏はRCBControllerというiOSアプリ（**写真5**）も作っています[注10]。このコントローラのサービス

注5） http://konashi.ux-xu.com
注6） http://gihyo.jp/dev/serial/01/futuredevice/0001

▼写真2　konashi

注7） http://shop.oreilly.com/product/0636920029281.do
注8） http://micono.cocolog-nifty.com/blog/techbasic/
注9） http://mbed.org/users/micono/code/BLE_ADT7410_TMP102_Sample/
注10） http://rcbcontroller.micutil.com/

▼写真3　Building iPhone and iPad Electronic Projects



とキャラクタリスティックのUUIDと書き込むデータが公開されているので、ペリフェラル側のコードを書くと、手軽にiPhoneやiPadでコントロールできるガジェットを作ることができます。jksoft氏がBLE_RCBControllerという、RCBControllerの操作をmbed HRM1017で取得するサンプルをmbed.orgで公開しています[注11]。これらを組み合わせると、iPhoneで操作できるBLEラジコンを自作する、といったことが手軽に行えます。

jksoft氏は、RCBControllerでコントロールできる、壁にマグネットで貼り付くラジコン、うぉーるぼっとの新型をプロトタイピングをしています[注12]（写真6）。

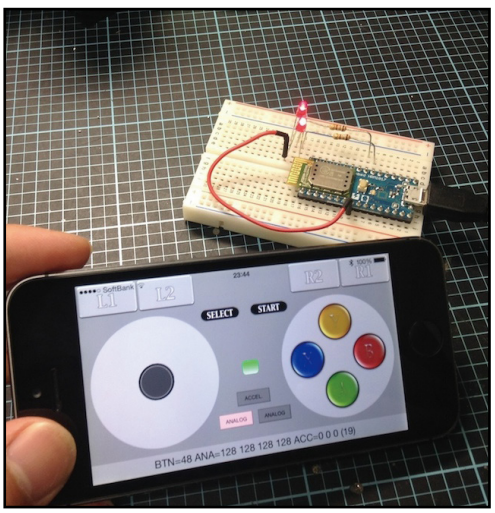
まとめ

紹介した書籍が洋書ばかりでしたので、BLEのハードルを高く感じてしまった方が多いかもしれません。しかし、今回の前半で説明した、GAPとGATT、UUIDの知識を基に、サンプルコードを読んで、いろいろ試してみるとBLEの概要は掴むことができるはずです。スキルは実際に手を動かしてみないと身につかないものだと筆者は信じています。ぜひ手を動かして、BLE入門を果たしていただければと思います。

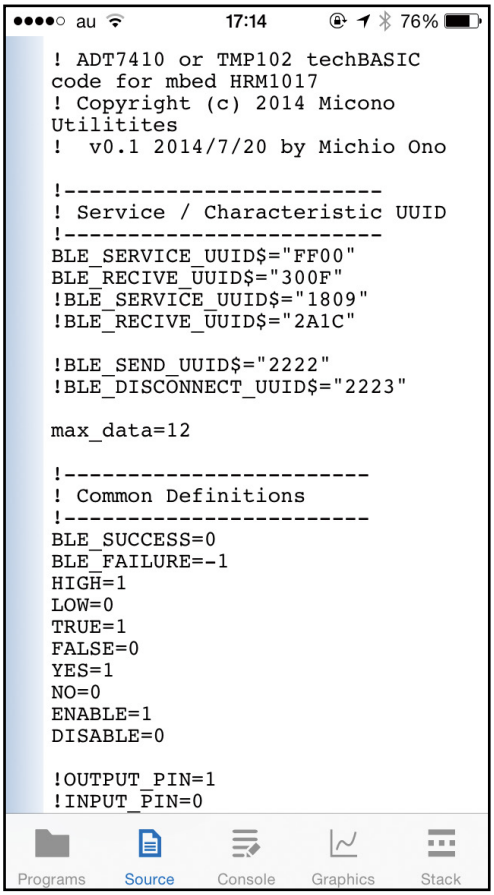
SD

注11）http://mbed.org/users/jksoft/code/BLE_RCBController/
注12）http://jksoft.cocolog-nifty.com/blog/

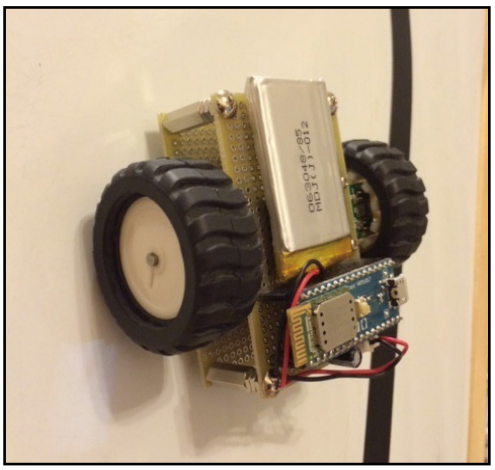
▼写真4　techBASIC



▼写真5　RCBController



▼写真6　新型うぉーるぼっと



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9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 LED キーボード DN-11255

2名

キーの内側からLEDが発光するメカニカルタッチのcherryMX青軸キーボードです。キー配列は87キー英語配列、インターフェースはUSBです。LEDは Fn + ↑ で明るくなり、Fn + ↓ で暗くなり消灯します。ラインナップは、赤いキートップにレッドLEDのモデルと、青いキートップにグリーンLEDのモデルの2種類です。各カラー1名様ずつプレゼントします。

提供元 上海問屋　**URL** http://www.donya.jp/

02 エルゴスティック マウス 400-MA059

1名

人間工学に基づいた、腱鞘炎になりにくいワイヤレスのエルゴスティックマウスです。横から軽く手を添えるように握り、右に傾けると右クリック、左に傾けると左クリックが入力されます。サンワダイレクト（**http://direct.sanwa.co.jp/**）のWeb限定商品です。

提供元 サンワサプライ　**URL** http://www.sanwa.co.jp/

03 パスワード マネージャ

2名

※上の画像は、サンプルです。

パスワード管理のソフトウェア（Windows/Mac/Android/iOS対応）です。製品版と同じ機能、期間で使用できる非売品のカードをプレゼントします。サイトから同製品をダウンロード後、アクティベーションキーを入力して、ご利用ください。

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

04 LinuxCon Japan 2014 T シャツ

1名

LinuxCon Japan 2014の参加者に配られたTシャツ。フロントには「ワタシハ リナックス チョットデキル」、バックには、LINUXCONのロゴが入っています。サイズはMサイズになります。

提供元 The Linux Foundation　**URL** http://www.linuxfoundation.jp/

05 シェルスクリプト 高速開発手法入門

2名

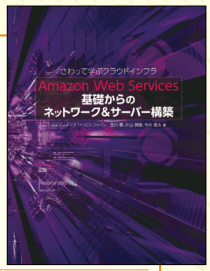
上田 隆一、後藤 大地 著／USP研究所 監修／
B5変形判、280ページ／
ISBN = 978-4-04-866068-60

UNIXのシェルスクリプトを用いて、実用的なアプリケーションを短時間で開発する手法を解説しています。シェルスクリプトでWebサイトを作るという内容を全面的に扱った1冊です。

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

06 Amazon Web Services 基礎からのネットワーク&サーバー構築

2名



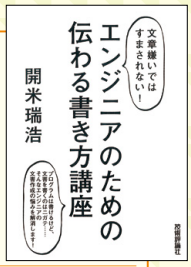
玉川 憲、片山 暁雄、今井 雄太 著／
B5変形判、216ページ／
ISBN = 978-4-8222-6296-9

「Amazon Web Services を実機代わりにしてネットワークを学び直す」をコンセプトにまとめた1冊です。低コストなクラウドでインフラ技術を学びましょう。

提供元 日経BP　**URL** http://www.nikkeibp.co.jp/

07 エンジニアのための伝わる書き方講座

2名



開米 瑞浩 著／
A5判、200ページ／
ISBN = 978-4-7741-6576-97

難しくなりがちな内容を相手に理解してもらうための「わかりやすい文書」の書き方を指導する本です。ITエンジニアに馴染みのある豊富な例文から、自分の仕事に近いものを見つけて学べます。

提供元 技術評論社　**URL** http://gihyo.jp/

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」
―― 習得のヒントと実践

多くのプログラマ、エンジニアにとって避けて通れない２つのテーマ「ポインタとオブジェクト指向」。組込系開発だけでなく、最近流行の Make: 系ガジェットを自在に操りたいときに C 言語が見直されています。さらにゲーム開発では C/C++ が重要な役割にあることは皆さん承知のことでしょう。

オブジェクト指向についても、Java での開発だけでなく、Python など各種スクリプト言語でもまさに必須な考え方になっています。しかしながらいざ執筆依頼をし制作を進めてみると、筆者の皆さんの間でオブジェクト指向プログラミングの習得について意見が分かれるという展開となりました。オブジェクト指向をもとに開発を進めることは、もしかしたら再検討すべき分岐点に来ているのかもしれません。本特集で皆さんも検討してみるのはいかがでしょうか。

CONTENTS

第1部 C言語ポインタの克服編

その1 ポインタの理解と活用

Writer 近藤 正裕(こんどう まさひろ) Twitter @kondoumh

はじめに

ポインタは、Cをシステムプログラミング[注1]言語たらしめている機能であり、メモリなどの計算機資源を効率よく利用するタイトなコードが書けます。反面、癖のある記法が初心者を混乱させ、システムを不安定にするようなバグを生むリスクもあります。

ポインタのメモリイメージ

プログラミングの初学者は、まず変数の使い方から覚えます。変数を宣言し、数量や文字などの「値」を代入し、プリント命令(Cだとprintf関数)に渡して表示したり、別の変数の値と比較したり。値を保持するための器(うつわ)として使いこなせるようになるまでの段階があります。次に配列やリストなどの使い方を覚えれば、いろいろとプログラムが書けるようになります。初めて学ぶプログラミング言語がCだと、値の器としての変数とは別に、ポインタの操作も理解しなくてはいけないため混乱が生じやすいのでしょう。ポインタの初期化をおさらいしてみましょう。

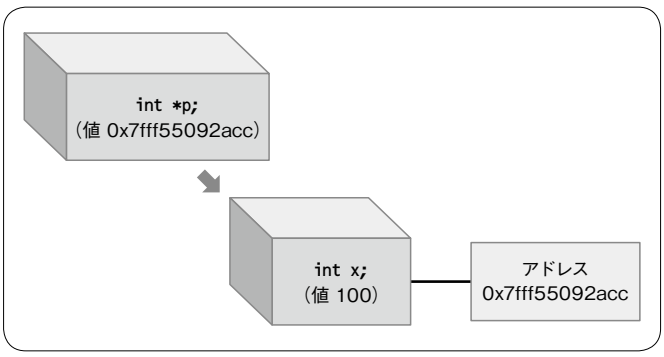
注1) オペレーティングシステムなど、アプリケーションよりハードウェアに近い部分のプログラミング。

▼リスト1 ポインタの初期化

```
int x;   /* int 型の変数 */
int *p;  /* int 型の変数へのポインタ */
x = 100; /* x に整数値100を代入 */
p = &x;  /* p に x のアドレスを代入 */
```

リスト1では、**int**型の変数**x**と**int**型変数へのポインタ**p**を宣言し、**x**に100という整数値を、**p**に**x**のアドレスを代入しています。アドレス演算子「**&**」によって、計算機のメモリに確保された変数xの物理的な位置(アドレス)を取得できます。**p**は**int**型変数へのポインタですので**x**のアドレスを代入できます。**リスト1**実行時にコンピュータ内のメモリは**図1**のような状態になります。

▼図1 リスト1の実行時のメモリイメージ



このようにポインタは「特定の型の変数のアドレス」を扱う特殊な型なのです。コードとメモリイメージの対応を理解することはさほど難しくないと思います。

さて、「アドレスだって値なんだからアドレス型が1個あればいいのでは?」と考える人もいるかもしれません。実際、アドレスも処理系によってbyte数は異なりま

すが数値ですのでintやlongに格納することはできます。しかし、ポインタは単にアドレスを格納するだけでなく、間接演算子「*」によって自身が指している変数の値を取得したり、構造体へのポインタの場合にはアロー演算子によってメンバにアクセスしたりする機能があります。

・間接演算子の利用

```
printf("*p : %d\n", *p); /* *p : 100 */
```

これらの機能を1つのデータ型で実現することは難しく、仮にできたとしても利用側のプログラムも複雑にしてしまうでしょう。それであらゆるデータ型に「*」を付けることでポインタ型が利用できるようにしているのです[注2]。

配列とポインタ

配列の処理を行うとき、ポインタでアクセスするか、添字アクセスにするか迷うことが多いですよね。配列とポインタの関係もメモリイメージで確認しておきましょう。

リスト3のように配列**a**を宣言すると、**int**型の要素を持つ配列が、連続したメモリ領域に確保されます(**図3**)。配列の添字演算子を使って個々の要素にアクセスできますが、ポインタに加算した値に間接演算子を適用してもアクセスできます。ポインタ**p**が配列の要素を指している場合、「**p + n**」は**p**から数えてn番目の要素を指すポインタになるからです。

配列名に添字演算子**[]**がついていない場合、その配列の先頭要素へのポインタとなります。したがって**リスト3**でポインタ**p**に**a**を代入すると、**p**は配列**a**の先頭要素を指すことになり、**p**に間接演算子を適用しても、添字演算子を適用しても配列**a**の要素にアクセスできます。**p**は配列ではありませんが、配列のように扱えるため一種の糖衣構文(シンタックスシュガー)と考えることができます。ただし、ループ処理内の添字アクセスでは、添字の計算が毎回行われるため、ポインタのイン

注2) あらゆるポインタ型を代入できる「**void ***型」がありますが、使用するときは実際の型へのキャストが必要です。

COLUMN

ポインタの宣言と間接演算子が紛らわしい

筆者が新人でCのコードを書いていたころ、先輩にC++を使っている人がいました。先輩は、ポインタを宣言するときに「**int *p;**」ではなく、「**int* p;**」のように書いていました。C++で参照を定義するときには、「**int &r**」ではなく、「**int& r**」と書く慣習があります[注3]。どっちで書いてもコンパイラは怒りませんが、「**int* p;**」と書くと「**int**型変数へのポインタ***p**」ではなく、「**int***型の変数**p**」のようにポインタ型として捉えやすくならないでしょうか。

筆者はこの書き方を見たときにポインタが腹落ちした気がします。宣言時の「**int *p;**」が間接演算と似ているため、わかりにくさを助長していると思います[注4]。ですので、宣言時はいつも「**int***」のように書きたいところですが、Cでは宣言時の「*」を変数側に付ける慣習があります。ANSI-Cでは、関数内で使用する変数の宣言を関数の先頭で行わなければならず[注5]、同じ型の変数をまとめて1行で定義するプログラマが多くいました。

図2をご覧ください。①のように宣言すると「**int***型」になるのは**pa**だけで、**pb**は「**int**型」になってしまいます。ポインタ型変数を複数定義するには、②のようにそれぞれの変数に*を付けなくてはなりません。ということで「**int ***型」として宣言をとらえるというのが無難でしょう。

注3) C++では参照の宣言がCのアドレス演算子と同じ「&」を使っているため、ここでも誤解が生じやすくなっていますね。

注4) C++でも間接演算子はCとの互換のため、「*」を使用します。間接演算はdereferenceの訳ですが、「参照外し」と呼ぶ人もいました。

注5) C99以降は変数宣言の位置の制限はなくなり、必要になった時点で宣言できるようになっています。

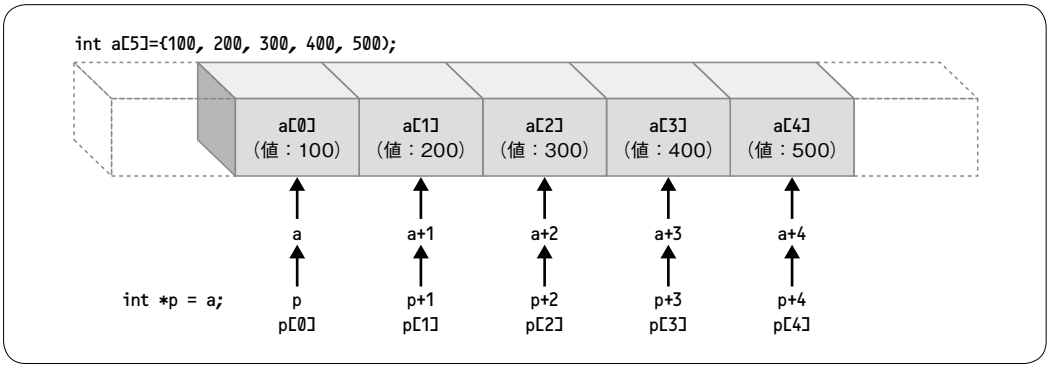
▼図2 複数のポインタ変数宣言

```
①int* pa, pb;  /* int 型へのポインタ pa と int 型変数 pb */
②int *pc, *pd; /* int 型へのポインタ pc とint 型へのポインタ pb */
```

▼リスト3　配列とポインタ

```
int a[5] = {100, 200, 300, 400, 500};
printf("%d\n", a[2]);    /* 300 */
printf("%d\n", *(a+2));  /* 300 */
int *p = a;              /* p は 配列 a の先頭要素を差す */
printf("%d\n", *(p+2));  /* 300 */
printf("%d\n", p[2]);    /* 300 */
```

▼図3　配列の要素と添字演算子、ポインタの関係



クリメントに比べて、実行効率がやや悪くなることには注意が必要です[注6]。

関数ポインタ

関数へのポインタも定義できます。関数もロードされたプログラムの特定のアドレスから取得できるのです。Cの標準ライブラリには、クイックソートアルゴリズムを実装したqsort関数が提供されています。この関数のプロトタイプはリスト4のように定義されています。ポインタbaseを先頭とする要素数num・要素サイズsizeの配列を、関数ポインタcompareで指定する比較関数を使って昇順にソートします。比較関数自体は、「void *型」のポインタを引数に取り、int型の戻り値を返すという規約を守ればいいのです。値の比較をするためには、「void *」を目的のポインタ型にキャストする必要があります。比較関数を自分で定義すれば、どんな型の配列でもソートできます。

▼リスト4　qsortの定義

```
void qsort(void *base, size_t num, size_t size,
      int (*compare)(const void*, const void*))
```

C++ のSTLやJavaのApache Commons/CollectionUtilsなどを使ったことがある方もいるのではないでしょうか。Cでも関数ポインタを使えば、これらのライブラリのように、アルゴリズムとデータ構造を分離した、汎用的で拡張性の高いプログラムを書くことができます[注7]。

おわりに

本稿では、ポインタのメモリイメージ、混乱を生じやすい演算子、関数ポインタの利用などを簡単に説明しました。C言語とポインタを扱うシーンは減っていますが、C++やObjective-CなどC直系の言語を使いこなすうえでも理解しておくと良いでしょう。SD

注6）よほどマイナーな環境でない限りコンパイラにより実行コードが最適化されるため、多くの場合考慮不要です。

注7）キャストするため、型安全なプログラムにはなりませんが。

第1部 C言語ポインタの克服編

その2 メモリとポインタの関係

Writer 岩尾 はるか(いわお はるか) レッドハット㈱ Twitter @Yuryu

ポインタとは

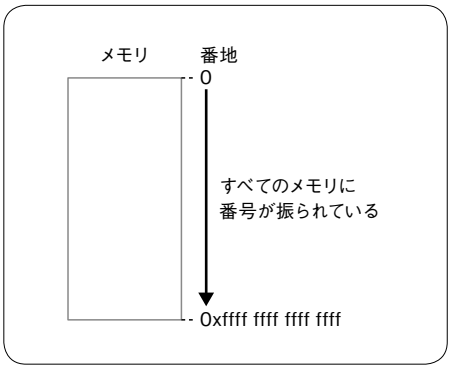
C言語の「ポインタ」は、ほかの言語と大きく違う特徴を持ちます。このパートでは、ポインタの基本的な構文は理解している人を対象に、なぜポインタが生み出されたか、ほかの言語が持つ「参照型」と何が異なるかについて述べます。

ポインタの2つの使い方

コンピュータのメモリは、そのすべてに「番地(アドレス)」が振られています。64bit CPUであれば0から18,446,744,073,709,551,615(16進数で0xffff ffff ffff ffffと書きます)と、すべてのメモリ領域にほかの領域と重複しない番地が付いています。それらを直接読み書きするための道具がポインタです(**図1**)。

実際のプログラムと、メモリ領域のイメージを対応付けながら説明します(**図2**)。

▼図1 メモリと番地

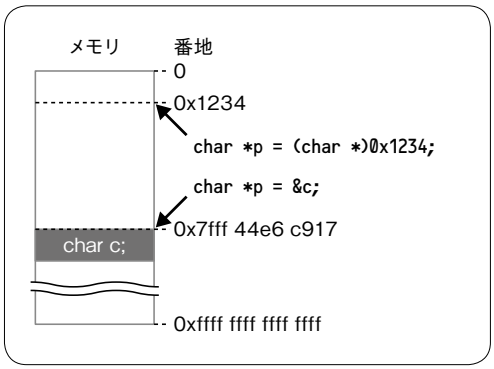


まず**リスト1**に、ポインタ「p」をアドレス0x1234にセットし、そこからデータを読み取るプログラムを示します。実行結果は多くの場合、次のようになります。

```
p = 0x1234
Segmentation fault (core dumped)
```

「Segmentation fault」は、プログラムが本来アクセスできない領域にアクセスしようとして、OSから「待った」がかかり、エラーになったものです。つまり、アドレス0x1234は、指定はすることはできるが、読み取ることができない

▼図2 ポインタとメモリ番地の関係



▼リスト1 直接アドレスを指定してデータを読み出す

```
1:#include <stdio.h>
2:
3:int main()
4:{     ↓ char型へのポインタpを宣言し、アドレス0x1234を代入
5:    char *p = (char *)0x1234;
6:     ↓ ポインタpのアドレスを表示
7:    printf("p = %p\n", p);
8:    printf("*p = %c\n", *p);
9:}     ↑ ポインタpのアドレス0x1234が指す先に存在する、char
        型の値を表示
```

領域ということになります。

通常のポインタの使い方というのは、**リスト2**のようなものになります。実行結果は、実行するたびに変わりますが、おおむね次のようになります。

```
c = A, p = 0x7fff44e6c917, *p = A
```

先ほどの**リスト1**と異なり、「Segmentation fault」が発生せず、変数**c**の中身である**'A'**が表示されます。また、**p**に入っているアドレスが非常に大きな値になっていることがわかります。

&演算子は、通常のCの変数から、0x7fff44e6c917といったアドレスを得る演算子になります。直接0x1234のように代入することもできますが、当てずっぽうで代入しても目的の変数にアクセスできないので、既存の変数のアドレスを求める演算子が用意されています。

C言語以外の多くの言語では、0x1234といった任意のアドレスにアクセスする機能はありません。通常役に立たないからです。ところが、限られた状況下ではこれが役に立つ場面が出てきます。

直接アドレスを指定する場面

なぜC言語では、一見役に立たない「任意のアドレス」にアクセスする機能を付けたのでしょうか。それにはC言語の歴史を振り返ると理解しやすいです。

C言語は、米ベル研究所によって、1969年ごろから開発が行われました。目的は当時アセンブリ言語で書かれていた「Unix」を、別のコンピュータに移植するためでした。アセンブリ言語はコンピュータごとに文法が異なり、互換性がありません。そのため、実行するコンピュータごとにすべて書き直しが必要でした。これを避けるため、コンピュータに依存しない言語としてC言語が開発されました。

「Unix」というOSを開発するためにC言語が設計されたため、当然アセンブリ言語でできることは、ほぼすべてできるように設計されました。その中の1つが「任意のアドレスにアクセスする機能」です。この機能の使い道のひとつに「Memory Mapped I/O（メモリマップドI/O）」があります（**図3**）。

コンピュータのメモリには、同じように番地が振られていても、データを保存するためのメモリと、周辺機器にアクセスするためのメモリの2種類に分かれています。周辺機器にアクセスするためのメモリ領域は、通常のRAMではない特別な回路に接続されていて、そこに読み書きすると周辺機器と通信することができます。このしくみが「Memory mapped I/O」です。

「Memory mapped I/O」の代表的な例がVRAM（ビデオRAM、またはビデオメモリ）です。VRAMは通常、メインメモリとは独立した領域に設けられ、CPUと専用の回路で接続されています。つまり、物理的にはメインメモリとVRAMはまったく別物です。ところがCPUからは、まるで通常のメモリと同じようにアクセスすることが可能になっています。

▼リスト2　通常のC言語のポインタの使われ方

```
1:#include <stdio.h>
2:
3:int main()
4:{    ↓ char型の変数 c を宣言し、文字'A'を代入
5:    char c = 'A';
6:    char *p = &c;
7:    ↑ char型へのポインタ p を宣言し、c のアドレスを代入
8:    printf("c = %c, p = %p, *p = %c\n", c, p, *p);
9:}   ↑ char型の変数c、ポインタpに格納されているアドレス、
        ポインタpのアドレスが指す先に存在するchar型の値を表示
```

▼リスト3　VRAMに直接文字を出力するコード

```
 char型へのポインタvramを宣言し、メモリアドレス0xb8000番地を指すように指定します。PC/AT互換機のVRAMは0xb8000番地と、仕様で決
↓まっています
char *vram = (char *)0xb8000;
*vram = 'A';    ← 0xb8000番地が指す先（VRAM）へ 'A' の文字を書き込みます
*(vram + 1) = 0x07;
↑ 0xb8000 + 1番地、つまり0xb8001番地へ文字の属性を書き込みます。0x07は、黒画面に灰色の文字を表示という指定です
```

PC/AT互換機で、VRAMに直接アクセスをして文字を出力するコードを**リスト3**に、動作の概要を**図4**に示します。

このコードは、通常のLinuxやWindowsといったOSが実行されている状態では動作しません。OSそのものが内部で実行するコードです。個別のプログラムが好き勝手にVRAMを書き換え始めると、ほかのプログラムの出力結果を操作できることになり、深刻なセキュリティ問題となります。そのため、OSが個別のプログラムからはVRAMを書き換えられないように保護しています。

C言語は、OSそのものを開発するために設計されたため、このように特別なメモリ領域に直接アクセスするためのしくみとして「ポインタ」が用意されました。この使い方はC言語特有のもので、ほかの言語にはない特徴になっています。

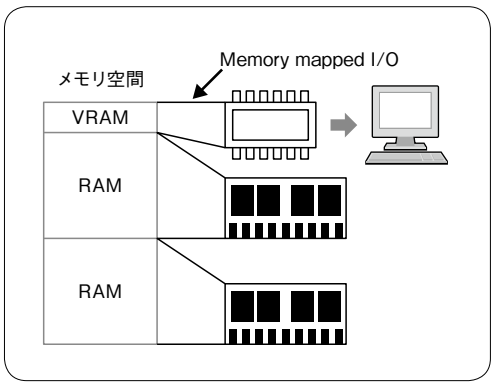
ほかの変数を参照する場面

LinuxやWindowsといったOSのもとで実行される通常のプログラムは、ハードウェアに直接アクセスできず、「Memory mapped I/O」を利用することはありません。すでにプログラム上で使用している、ほかの変数にアクセスする用途で、ポインタが使われます。

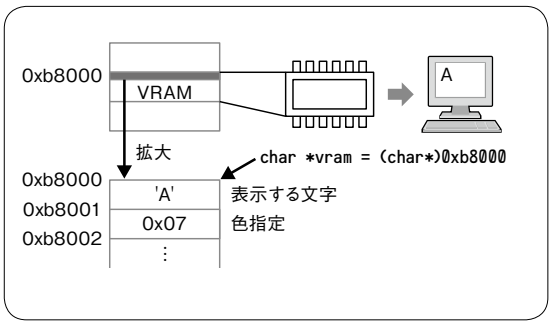
VRAMの例で見たように、「Memory mapped I/O」では、コンピュータの仕様でそれぞれのアドレスが決定されています。ところが、通常のプログラムでは実行するたびにアドレスが変わります。そこで、アドレスを動的に求めるための演算子として「&」が用意されています。

あらためて**リスト2**のプログラムを振り返ってみましょう。6行目で変数cのアドレスを「&c」として取得し、ポインタ変数「p」に保存しています。7行目で表示したときに、アドレスが「0x7fff44e6c917」と表示されています。この値は毎回変化します。ところが、「&」演算子を使うことによって、正しくメモリ上の領域が取得でき、目的のメモリ領域にアクセスできます。

▼図3　メモリ空間とMemory mapped I/O



▼図4　VRAMとポインタ



こちらの使い方は、多くの言語でも「参照型」として同じ考え方を採用しています。

まとめ

C言語のポインタには、アドレスを直接数値で指定してメモリ上の特定領域にアクセスする使い方と、ほかの変数のメモリ領域を「&」演算子によって求め、アクセスする使い方の2種類があることを述べました。

直接数値を指定する使い方はC言語独特のもので、ほかの言語にはない特徴の1つになっています。これは、C言語が「Unix」というOSを開発するために設計されたことに由来します。

ポインタの使い方としては少し特殊な例を紹介しましたが、理解の一助となれば幸いです。

SD

第1部 C言語ポインタの克服編

その3 アドレスに見るポインタの動作

Writer 小山 哲志（こやま てつじ） 合同会社ほげ技研 Twitter @koyhoge

メモリアドレス

ここではポインタを理解するために、コンピュータの内部動作に少しだけ突っ込んで解説していきます。現在のコンピュータは、CPUがメモリから命令やデータを順番に読み込んで実行することで動作しています。CPUが扱うメモリには0から始まる「番地」が付けられていて、その番地を指定することで値を読み書きしています。その番地のことをメモリアドレス、または単に「アドレス」と呼びます。アドレスはCPUが扱えるメモリ空間での、先頭からのbyte数で表されています。つまり1つのアドレスに保持できる値の大きさは1byteです。

C言語で変数を定義すると、その変数に対応したメモリに値が保持されます。たとえば、

```
char c = 3;
```

というプログラムは**図1**のように、特定のアドレスにその変数の値が書き込まれます（なおアドレスの表記には通常16進数が用いられますが、今回はわかりやすさを優先して10進数表記で行うことにします）。

char型の変数を保持するには1byteの領域で済むので、アドレスとその変数は1対1に対応します。では変数を保持するのに2byteが必要な**short**型の場合はどうでしょう（**図2**）。

▼図1 特定アドレスに値が書き込まれる


```
short s = 4;
```

short型の変数**s**を保持するためには、アドレス2つ分の領域を必要とします。この場合104番地から2byte分ということですね。

同様に保持に4byteが必要な**long**型、**float**型や、保持に8byteが必要な**double**型はその分多くのメモリが必要となり、アドレスもたくさん消費するということです。ここで注意しないといけないのは、たとえアドレスがわかったとしてもそれが指している変数の型がわからないと、どこまで読み進めれば良いのか判断できないという点です。

ポインタ＝メモリアドレス？

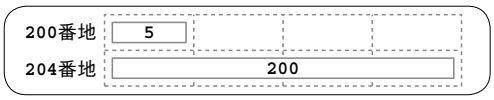
次にポインタを用いると、どのように変数が保持されるのか見てみましょう（**図3**）。

```
char c = 5;
char* cp = &c;
```

char型の変数**c**の場合は、最初の例の場合と同様です。問題なのは**char**のポインタ型の変数**cp**の場合です。C言語の場合は変数名の前に「**&**」を付けるとその変数のアドレスを意味

▼図2 short型の例

▼図3 ポインタを使った例



しますので、cpには変数cが保持されているメモリのアドレスが入ることになります。ポインタ型はこのようにアドレスを保持するための専用の型なのです。

ではポインタ型の場合に保持する領域は何byte必要なのでしょうか？ これはCPUやOSのモードによって異なります。読者のみなさんは32bitや64bitという単語を聞いたことがたぶんあるでしょう。これは現在主流になっているコンピュータの動作環境で、32bitの場合は4byte、64bitの場合は8byteがポインタ型の保持に使われます。つまり**図3**はポインタの保持に4byte使用していますので32bit環境での例ということになります。

ポインタの操作

ポインタ型の変数は、単にアドレスを保持している変数ということではありません。ポインタを操作する場合には、「どの型に対するポインタか」ということが重要になってきます。

ポインタに対する数値の加算・減算

ポインタ型は、メモリのアドレスを保持する専用の型だとすでに書きました。ポインタ型の変数に数値を足したり引いたりすることは、通常の変数とは違う効果をもたらします。次の例を見てください。

```
short s = 6;
short* sp = &s;
sp += 1;
```

変数spはshort型のポインタで、変数sが保持している領域のアドレス、この場合は300が入っています(**図4**)。

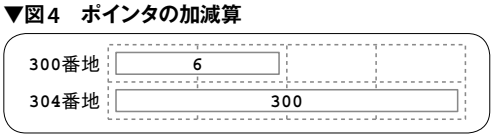
```
sp += 1;
```

ここでspに1を足すと、spの値は301になるように見えますが、実際にはそうなりません。spはshort型のポインタですので、1を加えることは、メモリをshort型の連なりとして考えたときの次のアドレスを指すことになります。つまりshortの保持に必要なbyte数である2を加えた「302」がspの新しい値になります。

これはCの処理系(コンパイラ)が、その変数がどの型へのポインタであるか判断して、自動的にそのような処理を行ってくれるのです。

このようにポインタ型は、ある型のデータが格納された連なりとしてメモリを考えた場合に、指定された場所にすばやくアクセスするのに有効なしくみなのです。

▼図4 ポインタの加減算



配列とポインタ

C言語では、特定の型の変数をまとめて確保する方法に「配列」を用いることができます。配列を使ってもメモリ内に領域が確保されるのは同様です。ただ配列を宣言した際の要素数分だけ、連続して領域が確保されるところが異なります。

```
long la[3];
long* lp = la;
```

この例ではlongの3要素分の変数laを宣言していますので、32bitである環境としてlongの大きさ4byteのメモリが3つ分、連続した領域に確保されます(**図5**)。この変数laのアドレスを、long型のポインタ変数lpに代入していますので、lpの値はこの場合は400になります。

ポインタ型の変数同士で引き算をすることもできます。次の例ではlong型のポインタ変数lp1、lp2を宣言して、lp1には配列laの0番目の要素のアドレス(つまり400)を、lp2には配列laの2番目の要素のアドレス(つまり408)を、それぞれ代入しています。

▼図5 配列のアドレスをポインタに代入

```
long* lp1 = &la[0];
long* lp2 = &la[2];
int distance = lp2 - lp1;
```

lp2からlp1を引くと、アドレスを数値として見た場合の差8には「ならずに」、long型の要素数の差、つまり2になります。

代入されていないポインタ

ポインタ型の変数は、値の代入をせずに宣言だけすることもできます。もちろん使用する前にはきちんと代入をしないといけませんが、とりあえず宣言だけしておくことは可能なのです。

```
long* lp;
```

この例ではlong型のポインタ変数lpを宣言だけしています。このときにメモリ上に確保されるのは、アドレスを入れるための4byte(32bit環境の場合)の領域だけです(**図6**)。代入が行われていないので、その値は「不定」です。おそらく以前にその領域を使ったデータが、そのままゴミとして残っているでしょう。

ポインタのポインタ

C言語の初心者が頭を悩ますものに「ポインタのポインタ」があります。ポインタというよくわからないものが、さらにダブルでやってくるわけですからもうお手上げ、といったところでしょう。

でもここまで読み進んで来た読者の方々は、「ポインタとは本質的にはアドレスのことである」とわかっているので、もう怖がることはないでしょう。

次の例を見てみます。

```
long l = 4126;
long* lp = &l;
long** lpp = &lp;
```

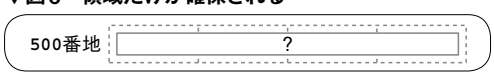
long型の変数lに値が代入されていて、long型のポインタ変数lpには、変数lのアドレスが代入されています。そこからさらに、long型のポインタのポインタ変数lppに、変数lpのアドレスが代入されています。この場合メモリ上は**図7**のようになっていて、lpの領域にはlのアドレスである600が、lppの領域にはlpのアドレスである604が書き込まれています(この例ではそれぞれの領域が連続していますが、必ずしもそうなるとは限りません)。

ポインタのポインタ変数lppから、lの値である4126を取得するには、

```
long new_l = **lpp;
```

のように「*」を用いた参照解決(デリファレンス)を2回行えば良いです。

▼図6　領域だけが確保される



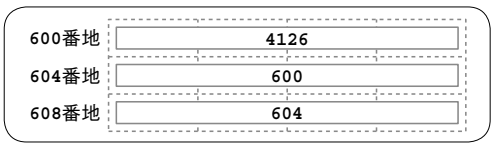
なぜポインタを使うのか

これまで解説してきたように、ポインタはメモリアドレスを与えられた型に合わせて的確に指し示すためのしくみです。ではなぜこのようなしくみが必要なのでしょうか？

その答えは、C言語がハードウェアを直接扱うことができる「低水準言語」だからです。この場合の低水準とは、劣っているという意味ではなく、よりハードウェアに近いという意味です。コンピュータが持つさまざまなデバイス、入出力を司る各種コントローラなどを制御するには、最終的には物理的なメモリを直接扱わなくてはなりません。この分野ではC言語(とC++言語)は圧倒的なシェアを持っており、C言語がまさにそのために生き残っている理由でもあるのです。

つまりポインタを理解し使いこなすことは、ほかの言語では実現不可能な「ハードウェアを直接いじる」というC言語の強みをマスタすることでもあるのです。SD

▼図7　ポインタのポインタ



第1部 C言語ポインタの克服編

その4 ポインタはどんなときに役立つのか

Writer 前橋 和弥(まえばし かずや)

はじめに

Cのポインタについて、初心者がよく持つ疑問として、「ポインタがなんの役に立つのかがわからない」というものがあるようです。

しかし、これはどうも妙な話に思えます。日常的にCのプログラムを書いている人達がポインタを使っている以上、Cのプログラミングにおいて、ポインタは必要です。にもかかわらず初心者が「なんの役に立つのかわからない」という疑問を持ってしまうのだとすれば、それは、多くの教科書において、ポインタを使わなくて良い例でポインタを説明しているからではないかと思います。

本稿では、「ポインタがなんの役に立つのか」という疑問に対し、もっと実践的な面から説明していきます。

ポインタはなんの役に立つのか?

関数から複数の値を返す

関数から何らかの値を返すときには、通常、戻り値を使います。しかし、Cでは、戻り値では1つの値しか返すことができませんので、複数の値を返す場合には引数でポインタを渡します。

たとえば、「ユーザがマウスでクリックした座標を取得する関数」を作るとすれば、その関数からはX座標とY座標を返す必要があります。具体的には次のような関数になるでしょう。

```
void get_clicked_point(int *x, int *y)
{
    *x = {ユーザがクリックしたX座標};
    *y = {ユーザがクリックしたY座標};
}
```

※実際に「クリックした座標」を取得する方法はOSなどにより異なるでしょうから、その部分は「{ユーザがクリックしたX座標}」のような書き方でごまかしています。

この関数を呼び出す側では、次のように書くことになります。

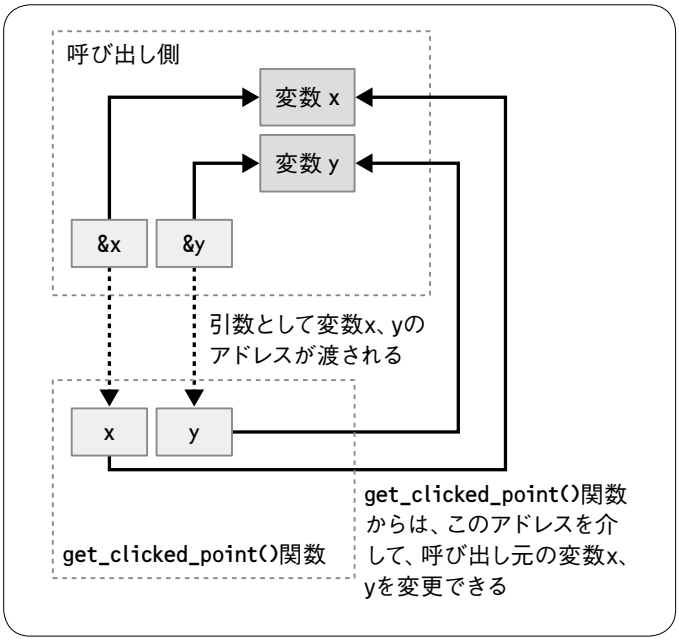
```
int x;
int y;
get_clicked_point(&x, &y);
```

呼び出し側のx, yという変数へのポインタ(アドレス)を**get_clicked_point()**関数に渡し、そのアドレスに、**get_clicked_point()**関数側で値を設定してやるわけです(**図1**)。

関数から複数の値を返すというのは、実際のプログラムでもよく行われます。とくに例外処理機構のないCでは、「処理が正常に終了したかどうかを戻り値で返し、その関数で算出した結果などは、引数として渡されたアドレスに返す」という方法は定番ともいえます。このような使い方においては、「ポインタがなんの役に立つのかわからない」ということはないのではないでしょうか。だって、ポインタを使わないと書けないんですから[注1]!

注1) 構造体を戻り値で返すような方法もありますが、用途によっては構造体を定義すること自体がめんどくさいでしょう。

▼図1　関数から複数の値を返す



にしてポインタに1を加算したとき、pは1byte分進むのではなく、その指す型のサイズ分進むということです。この例では、pはintへのポインタですから、p++するとpは4byte進みます(intが4byteの処理系の場合)。

――さて、こう書けばポインタを使って配列arrayの内容を表示できるのは良いとして、多くの人は、「なぜこう書かなければいけないのか?」という疑問を持つことでしょう。ポインタが「なんの役に立つのかわからない」という声は、こういう例でポインタを使うところから出てくるように思います。

配列操作にポインタを使う

Cの入門書などでよく見かけるのが、「配列操作にポインタを使う」という使い方です。

intの配列array(要素数は10)に格納された値を順に表示する際、次のように配列の添字アクセスの形で書くのではなく、

```
int i;
for (i = 0; i < 10; i++) {
    printf("%d\n", array[i]);
}
```

次のようにポインタ演算を使う、というものです。

```
int *p;
for (p = array; p != &array[10]; p++) {
    printf("%d\n", *p);
}
```

この例では、intへのポインタ型である変数pを最初に配列の先頭に向け、その後p++によりポインタを1つずつ進めています。最後に、pが配列の最後の要素の次の要素を指したらループを抜けます。ここで重要なのは、p++のよう

配列とポインタの関係

配列を使うとき、あなたはすでにポインタを使っている

先ほどの例では、p++のようにして、変数pの値を変更しました。しかし、pの値を変更しなくても、*(p + i)のように書けば、pからiだけ進んだ場所にアクセスできます。

そして、*(p + i)という書き方は、単純にp[i]に置き換えることができます。というより、配列アクセス時に使う[]という演算子(添字演算子といいます)は、もともとそういう意味しかありません。最初の例でarray[i]と書いているときも、コンパイラはこれを*(array + i)と解釈します。もしあなたが「ポインタ演算はなんだか難しいから、いつも添字でアクセスしよう」と決めているのだとしても、配列をアクセスするとき、あなたはすでにポインタ演算を使っているのです。

ポインタはどんなときに役立つのか その4

ポインタ演算はやめてしまおう

配列を使ったループの中で`array[i]`を何度も参照するのであれば、`*(array + i)`という加算を何度も行うより、ポインタ変数`p`を導入して、加算を一度の`p++`にまとめたほうが、性能がよくなる――Cが開発された当初の、大昔のCコンパイラであれば、そういうこともあったのでしょう。実際、「ポインタを使ったほうが一般に高速」と書いてある本もあります。しかし、今どきのコンパイラであれば、この程度の最適化は自動で行うので、性能に差が出るようなことはまずありません。

現状では、ポインタ演算を駆使したプログラムを書く必要性はなく、素直に添字でアクセスすれば良いと思います。

関数に配列を渡す

Cでは、関数に配列を渡すとき、以下の3種類の書き方があります。

```
void func(int *array) ←❶

void func(int array[]) ←❷

void func(int array[10]) ←❸
```

この3つは、すべて同じ意味です。❷のように書いても、実際に`func()`に渡されるのはポインタですし、❸では配列の要素数として10と書いているように見えますが、この10はコンパイラは単に無視します。

そして、上記のどの方法で書いたとしても、渡された配列は`array[i]`のようにしてアクセスできます。

なお、上記❷のような空の`[]`がポインタの宣言を意味するのは、唯一上記のケースだけです。混乱しやすいのは次のような例です。

```
char *str = "abcdefg";
char str[] = "abcdefg";
```

この例において、前者はポインタ変数`str`の初期化であり、後者は文字型配列`str`の初期化であり、それぞれ意味が違います。

malloc()で確保した領域を使用する

Cでは、配列の要素数は通常固定ですが、プログラムによっては、実行するまで必要な要素数が決まらない場合もあります。そのような場合は、`malloc()`を使ってメモリを動的に確保します。

たとえば`int`型n個分のメモリが必要な場合は、次のように書きます。

```
int *p;

p  = malloc(sizeof(int) * n);
```

これにより、`int`のサイズ×n byteのメモリが確保され、ポインタ`p`がその先頭を指します。

こうして確保したメモリを利用する際は、`p[i]`のように書いて配列のようにアクセスできます。前述のとおり、`p[i]`というアクセスは`*(p + i)`と同じ意味ですし、`p`は`int`へのポインタですので、`i`加算すれば`sizeof(int)`×`i`だけ進むからです。

おわりに

Cにおいてポインタは難関と言われますが、実のところ`*p++`のようなポインタ演算を使うのをやめてしまえば、問題の半分は解決します。

残りの半分は「宣言の構文が不可解」ということですが、それはまたの機会としましょう。SD

第1部 C言語ポインタの克服編

その5 ナンカ分カラナイケドで生きていけるポインタ入門

Writer 村上 福之(むらかみ ふくゆき) Writer @fukuyuki

考えるな! 感じろ!

「初心者読者さんが、思わず膝を打って「わかった!」と思うような記事を2ページばかり書いていただけないでしょうか。ポインタがわかった瞬間を紹介、こうするとポインタがわかりやすくなるよ、といったアドバイス、仕事をしながらわかった事例など」とメールを編集からいただきました。わかりました。書きましょう。

ポインタはフォースみたいなものです。
考えるな! 感じろ!

それ以上書くこともないですし、それがわからなければ、あなたにはフォースとともになかった人なので、プログラマなんてゴミみたいな仕事はやめて、荷物をまとめてさっさとタトゥーインに帰ってください。

偉そうなことを言ってますが、そんな僕も、最初からポインタがわかったわけでもありません。アホなんで、そんなのが一発でわかるわけがありません。ナンカ分カラナイケド、&か*付ければいいんだろ!?

ポインタ(pointer)とは、あるオブジェクトがなんらかの論理的位置情報でアクセスできるとき、それを参照するものである。有名な例としてはC/C++でのメモリアドレスを表すポインタが挙げられる(wikipediaより)

ポインタの宣言は*を名前の前に付けます。「int *n;」でnを指すint型ポインタ、「char *str;」でstrを指すchar型ポインタ、「double *dp;」でdpを指すdouble型ポインタです。アドレス演算子(&)はアドレスを得ます。間接演算子(*)はポインタの指す値を得ます……と説明されますが、20歳くらいの当時の僕には意味がわかりませんでした。

ぼくは「とりあえず動けばいいプログラマ」だったことが長いので「わかんないけど&か*を付けて、コンパイルが通って正常に動作したらいいんだろ」ということをずっとやってたら、そのうち覚えました。ポインタ入門でよくあるこういう値の交換ソースコードです(**リスト1**、**図1**、**リスト2**、**図2**)。

これを読むと、アドレスとかなんとかはよく

▼リスト1 aとbを交換しようとするがうまくいかない例

```
#include <stdio.h>

void swap( int a , int b ){
    int tmp;
    tmp=a;
    a=b;
    b=tmp;
}

int main(){
    int a;
    int b;
    a=10;
    b=20;
    printf("a=%d b=%d\n" , a , b );
    swap( a , b );
    printf("a=%d b=%d\n" , a , b );
    return 0;
}
```

ナンカ分カラナイケドで生きていけるポインタ入門 その5

わからないけど、整数値などで中身を書き換えたいときは、呼び出し側で&を付けて、呼ばれるほうは*でもつけておけばいいんだろ？　という理解をしました。

最初のうちはこれで、なんとかゴマカシて動かして、*か&を付けて動いたほうで、printfデバッグを地道に繰り返していたら、なんとかできました。ひどいもんです。それでも仕事はできてしまうものなのです。

ナンカ分カラナイケド文字列は**付ければいいんでしょ？

いわゆるポインタのポインタという意味を理解するのもめんどくさいので、最初は「文字列だったらなんでも**付けていた」ように思います(リスト3)。

▼図1　リスト1の実行結果

```
a=10 b=20
a=10 b=20
```

▼リスト2　アドレス渡しをしてaとbを交換

```
#include <stdio.h>

void swap( int *a , int *b ){
    int tmp;
    tmp=*a;
    *a=*b;
    *b=tmp;
}

int main(){
    int a;
    int b;
    a=10;
    b=20;
    printf("a=%d b=%d\n" , a , b );
    swap( &a , &b );
    printf("a=%d b=%d\n" , a , b );
    return 0;

}
```

▼図2　リスト2の実行結果

```
a=10 b=20
a=20 b=10
```

たとえば、リスト3のように、文字列の中身を変えるコードを書いていて、ナンカ分カラナイケド、使う前にmallocで使うメモリのサイズを指定しないといかんらしい。sizeofって、解説を読んだけど、わからないけど、書いていたし、ナンカ分カラナイケド、最後にfreeで解放しないといけないらしい。ナンカ分カラナイケド、mallocで指定した値より長い文字列を入れるとクラッシュするなど、体で覚えていたように思います。

もちろんよくないですけど、システム的にはそれで動いてしまうので、間違ったやり方でしたが、それで覚えていたように思います。ただ、このやり方は間違えると、明らかにクラッシュするので、まったくもってよくありませんが、小手先の技術を使えば、適当に切り抜けていたように思います。

そんなわけで、ポインタの意味を理解せずに、パターンで覚えて、適当に使いまわしていたように思います。「文字列だから」「整数だから」「Windowsのハンドルだから」などなど、意味もわからずコードを書いていたように思います。

その後、就職して、ユルいGUIを使ったヘナチョコWindowsのアプリくらいだとポインタの意味なんて理解しなくても作れました。ポ

▼リスト3　**付きプログラム

```
#include <stdio.h>

void change_string( char** s  ){
    *s="bye";
}

int main(){

    char *str=(char*)malloc( sizeof
( char )*5 );

    strcpy( str , "hello" );
    printf("before:str=%s\n" , str );
    change_string( &str );
    printf("after:str=%s\n" , str );

    free( str );

    return 0;
}
```

インタの意味をよくわかってない僕が書いたコードが記録されたCD-ROMが工場で数十万枚量産され、世界中の家電量販店で売られていたのが、ぼくの90年代です。時効だから言いますけど、リコーさんやキャノンさんのプリンタについていたバンドルアプリとかは、ポインタなんか知らずに書いてました。

ナンカ分カラナイケドだけが人生です

こんなことを言うと怒られるかもしれませんが、外国語でも、プログラミング言語でも、最初は模範解答みたいなものを大量に見て、写して、体で覚えて使えばいいと思います。たとえば、僕が海外で働きながら英語を覚えたときも、「ナンカ分カラナイケド、みんなメールの文末にRegardsって付いているから、付けときゃいいんでしょ？」みたいな感じで仕事もできてしまいます。

だいたい、学校でも数学の問題をまじめに考えている奴は成績が悪くて、赤本の模範解答を写しまくって体で覚えている奴のほうが偏差値がよかったりします。分数の割り算をするのに、どうして分子と分母をひっくり返すのかを理由をいちいち考えるよりも、ナンカ分カラナイケド、そんなもんなんだと思っている奴のほうが成績がいいのです。

嫁さんがなぜか意味もなく怒っていていても、ナンカ分カラナイケドそんなもんなんだと思うし、機嫌を取るために、東京レストランの最初のページに載ってたレストランにとりあえず連れて行くと、美味しいのか美味しくないのかわからないけど、機嫌がよくなったりしますが、ナンカ分カラナイケドそんなもんなんだと思うわけです。

子供が生まれて、幼いころは「お父さんのお嫁さんになるー」と言っていた娘も、中学の中頃から、いつの間にかまったく口をきいてくれなくなったりしますし、ナンカ分カラナイケドそんなもんなんだと思うわけです。

プログラミング言語も、外国語も、社会のしくみも、男女の関係も、家族の関係も、人生そのものも、最初はナンカ分カラナイケドそんなもんなんだと思うしかないことが多いです。ナンカ分カラナイケド前に進めることができる人こそが、人生を1つ前に進めることができるのです。ポインタでも人生でも、本当の意味を考えてないといけないときは、自動的にやってきます。

なんかわからないけどで、すまなくなってきたとき

そんなことをやっているうちに、残念ながら、ポインタの本当の意味を考えないといけないことがやってきました。プリンタドライバ開発やコンパイラ開発などのお仕事をしないといけないときにポインタとは何なのか理解しないと開発ができない時期になってしまいました。

ほかの執筆陣がかなりガチなことを書いているらしいので、はしょりますが、ようするにポインタは単純にメモリの場所を示しており、デバッガなどでポインタの指し示すアドレスのメモリの内容をダンプせざるを得ないような仕事をもらったら、ポインタなんて、勝手に覚えます。

画像処理のコードを書いたり、複雑なデータ処理をしたり、プロトコルスタックを組むような仕事をもらうと、イヤでも覚えます。

逆に、GUIをちょこちょこいじって、「ボタンガー！」「リストボックスガー！」「テキストボックスガー！」とかのコードしか組まないヘナチョコさんは、ポインタの意味なんか理解しなくてもコピペで書けます。

そんなわけで、ポインタを理解できないというのは、たぶん、ポインタを理解しないといけない時期にきてないだけだと思います。最近は、あまりメモリを直接意識する開発がどんどん減ってきたので、一生ポインタの意味を理解しなくてもいいプログラマも増えてくると思います。

SD

第1部 C言語ポインタの克服編

その6 ポインタの魅力と危険性

Writer 田中 邦裕(たなか くにひろ) さくらインターネット㈱ Writer @kunihirotanaka

はじめに

私はC言語より前にアセンブラを使っていたのですが、初めてC言語に触れたときの印象は「アセンブラより楽な低級言語」というものでした。実際にC言語のコードをコンパイルしてできたバイナリを見てみると、C言語の記述と対になってアセンブラのコードが生成されているのがわかり、非常に勉強になったことを思い出します。

その中でもポインタというのは、CPUのメモリ空間を直に操作することができ、アセンブラを理解していると非常に使いやすいなという印象を持ちました。

ポインタのメリットは、メモリ空間を自分の頭の中で想像しながら何でもできるということであり、デメリットは何でもできてしまうためにバグを生みやすいということといえます。ここでは、カンマの入った文字列を、カンマで区切って画面に表示するコードを見ながら、メリットとデメリットを見てみることにします。

ポインタを使わないPerlのコード

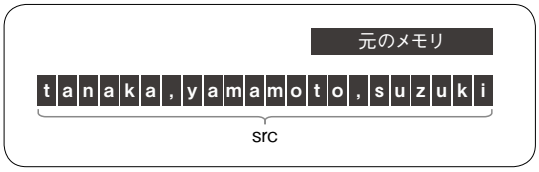
まず、Perlで文字列をカンマで区切って表示するコードを見てみます(**リスト1**)。**$src**に初期の文字列(**図1**)を代入し、**split**関数を使ってカンマで区切り、**result**という配列に代入するというものです。

このコードの場合、分割されたそれぞれの文字列は新たなメモリ空間へと代入されます。つまり、新たなメモリ空間を消費し、元の文字列の入っているメモリ空間からコピーをしていることになります(**図2**)。この例では小さな文字列ですのでオーバーヘッドは少ないのですが、

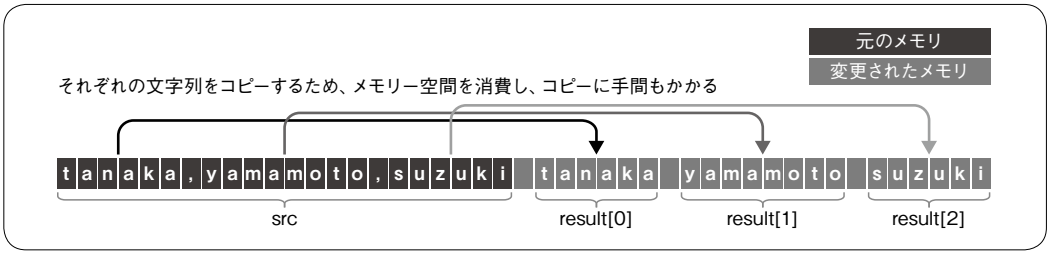
▼リスト1 Perlのサンプルコード

```
my $src = "tanaka,yamamoto,suzuki";
my @result = split(/,/, $src);
print $result[0] . "\n";
```

▼図1 元の文字列



▼図2 ポインタを使わない場合



これが数百MBクラスの大きなデータだとすると、消費メモリ、コピー時間ともに無視できないでしょう。

ポインタを使ったC言語のコード

これをC言語のポインタを駆使して書くとどのようになるでしょうか？　**リスト2**がC言語のコードです。

まず、6行目で**string**という配列に初期の文字列を代入し、7行目で**string**へのポインタを**src**というポインタに代入します(**図3**)。併せて8行目で結果のポインタを格納するための**result**という配列を定義しておきます。そのあと、10行目で**result**の最初に文字列の先頭を示すポインタを代入します。これで準備は完了です。

ここからは、11行目から17行目のループで、文字列の最後を示す**'\0'**に到達するまで、カンマを**'\0'**に置き換えていきます。ちなみに、**src**というポインタに「*****」を付与すると、そのアドレスにある文字にアクセスできます。つまり**while**が始まった直後は「***src**」とすることで、**t**という文字にアクセスできます。その後、「**src++**」とすると、ポインタの位置が1つ後の文字に移り、**while**の2回目のループでは「***src**」とすることで、**a**という2番目の文字にアクセスできます。

これを繰り返すと、文字列の最後を示す**'\0'**に到達し、**while**を抜けます。**while**の中では12行目でカンマを探し、見つけたときは13行目で**'\0'**を代入して文字列を分割し、14行目で**'\0'**の1つあとの文字を示すポインタを**result**に代入します。

このコードの場合には、消費するメモリ量は増えておらず、かつコピーも発生していません。カンマのあった2ヵ所に**'\0'**を代入しただけであり、非常に高速に処理を行うことができます。

ポインタを使うデメリット

ただ、デメリットもあります。このコードを実行したあとは、**src**の位置が文字列の最後になっており、元の文字列が含まれる**string**の中身も変更されています。**string**を表示しようとしても**tanaka**という文字列しか返ってきません。それは、**tanaka**のあとのカンマが**'\0'**に置き換えられてしまったからです。

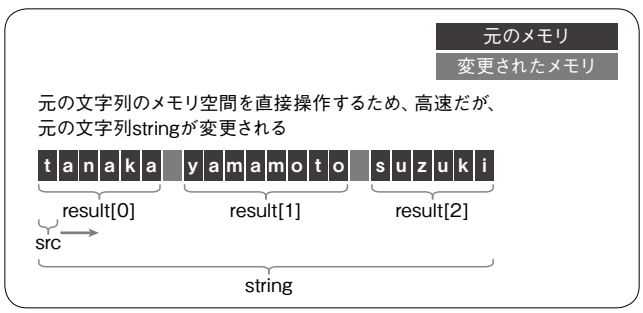
これが意図したものであれば良いのですが、意図せずにポインタの指し示す位置を変更してしまうと、データの破壊を引き起こすことにつながります。

また、ポインタを使うとコードの可読性が低

▼リスト2　C言語でのサンプルコード

```
01: #include<stdio.h>
02:
03: int main()
04: {
05:     int i=0;
06:     char string[] = "tanaka,yamamoto,suzuki";
07:     char *src = string;
08:     char *result[3];
09:
10:     result[i++] = src;
11:     while(*src != '\0'){
12:         if( *src == ',' ){
13:             *src = '\0';
14:             result[i++] = src+1;
15:         }
16:         src++;
17:     }
18:     printf("%s\n", result[1]);
19:     return 0;
20: }
```

▼図3　ポインタを使う場合



くなるというのも注意点です。今回はsrcというポインタを操作しましたが、プログラムの実行途上においてはsrcの指し示す文字の位置が時々刻々と変化します。そのため、自分がどのデータを操作しているのかがわかりにくいという問題が発生しがちです。

なお、このコードには致命的なバグが含まれています。それはresultという配列の長さをチェックしていないということです。14行目において、「result[i++]」という形で配列にアクセスしていますが、8行目で配列を定義したときには3個しか確保していません。

今回の例では、カンマが2つでしたので、文字列は3つに分割されるわけですが、もしカンマが3つ以上あったとすると、resultで確保した配列のメモリ空間をオーバーフローすることになります。

Segmentation Faultになる場合とならない場合

ポインタや配列へのアクセスにおいては、言語側でオーバーフローしているかどうかのチェックをしてくれません。つまり、「result[3]」というアクセスも言語的には通用してしまいます。ただ、OS的にはメモリ空間を確保していないため、「Segmentation Fault」になってしまいます(図4)。そのため、14行目の代入の前に、「if(i <= 3)」などといった、オーバーフローしないようなチェックが必要です。

ちなみに、メモリ空間が別の変数によって確保されている場合には、「Segmentation Fault」にならない場合も考えられます。このときは、意図しないデータへのアクセスが成立してしまうなどの問題が発生する可能性があります。

たとえば、リスト3のようなコードを書いて、図5のようにresultという配列で2個のポインタと、passwordというポインタを格納するための変数を定義したとします。

このときに、result[2]とアクセスすると、passwordへのポインタを取得できることになります。

ここまでわかりやすい間違いは少ないかもしれませんが、もしこれがWebサービスなどで、CGIの引数で配列にアクセスできるとしたら、外部からパスワードを抜き取られてしまうことになります。

おわりに

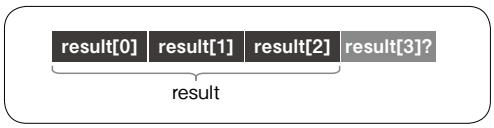
今回、実際のコードを見ながらポインタのメリットとデメリットを見てきました。最近ではコンピュータも高速化し、高級言語で記述する機会も増えてきたことから、さほどポインタについて意識しなくても良いようになりました。

しかし、本当に高速なコードを書くのであれば、C言語とポインタを駆使することは大きなメリットであるといえます。そのうえで、ポインタは何でもできる分、致命的なバグを生みやすいということを知ってもらえればと思います。

メリット／デメリットを知って、良いポインタライフを送られることを期待しております。

SD

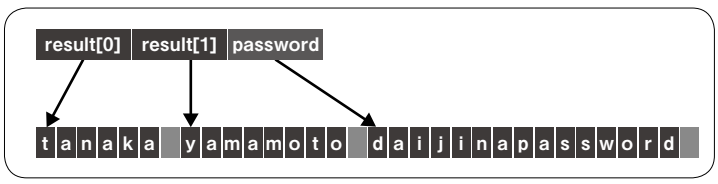
▼図4　Segmentation Faultになる例



▼リスト3　ポインタ定義

```
char *result[2];
char *password;
```

▼図5　パスワードの漏えい



第2部 オブジェクト指向の克服編

その1 Javaでオブジェクト指向を知るための3つの基礎練習

Writer 増田 亨（ますだ とおる） ギルドワークス㈱／Twitter @masuda220

Javaとオブジェクト指向

Javaは、オブジェクト指向の考え方を重視して開発されました。しかしJavaはオブジェクト指向「らしくない」書き方もできてしまいます。残念なことにJava入門書はオブジェクト指向「らしくない」内容に多くのページを割いています。

Java言語を習得するためには「らしくない」部分の学習も必要です。しかしJavaを使うからにはオブジェクト指向「らしい」書き方を早くから体験すべきです。

この記事はJavaの初心者を対象に、Javaでオブジェクト指向らしさに触れるための練習方法を3つ紹介します。

・BigDecimalを使ってみる
・「コレクション」を使ってみる
・列挙型を使ってみる

こういう「オブジェクト指向」らしい内容を早くから体験し、なじんでおくことこそ、Javaでオブジェクト指向らしいプログラミングを習得する効果的で実践的な学習方法です。

BigDecimalを使ってみる

オブジェクト指向らしさの第1段は「BigDecimalクラス」です。

BigDecimalクラスを使って、数値計算をしてみましょう。この練習で、次の2つのオブジェクト指向の基本の「感覚」が体験できます。

・オブジェクトを作る感覚
・複数のオブジェクトを組み合わせる感覚

リスト1のサンプルコードを見てみましょう。これの結果は、小数点以下2桁までに丸めた、333.33になります。単純な割り算（1000 ÷ 3）なのに、なぜ、こんな面倒くさい書き方をするのか、最初はピンとこないかもしれません。

しかし、BigDecimalクラスとMathContextクラスで記述した**リスト1**の書き方には「オブジェクト指向らしさ」の基本がぎゅっと凝縮されています。

オブジェクトを作る

リスト1の①、②、③で3つのオブジェクトを作っています。

①と②は、どちらもBigDecimalクラスのコ

▼リスト1　割り算（1000÷3）の例

```
BigDecimal a = new BigDecimal("1000");                          ←①
BigDecimal b = new BigDecimal("3");                             ←②
MathContext context = new MathContext(5, RoundingMode.HALF_EVEN);  ←③
BigDecimal result = a.divide(b,context);                        ←④
System.out.println(result);
```

Javaでオブジェクト指向を知るための3つの基礎練習 その1

ンストラクタを呼び出しています。引数にはそれぞれ“1000”と“3”という異なる値を渡して、別のオブジェクトを作っています。

「同じクラス」のコンストラクタに「別の値」を渡して、値の「異なるオブジェクト」を作る。これがオブジェクト指向らしいプログラミングの基本になります。

③は桁数や丸め方法を指定するために、MathContextクラスのインスタンスであるcontextオブジェクトを作成しています。

オブジェクトを組み合わせる

全部で3つのオブジェクトを作成しました。この3つのオブジェクトを組み合わせて、④で`a.divide(b,context)`の割り算を実行しています。

この「複数のオブジェクトを組み合わせて1つの仕事をする」ことが、もう1つのオブジェクト指向のたいせつな感覚です。

このサンプルは最初は「面倒なだけ」に見えるかもしれません。しかし入門書にありがちな、`int/double`と`if`文を使う書き方を覚えるだけでは、いつまでたっても「オブジェクト指向らしさ」を習得できません。

BigDecimalクラスとMathContextクラスを使ってオブジェクト指向らしい書き方を練習してみましょう。

「コレクション」を使ってみる

オブジェクト指向らしさの第2弾は「コレクション」です。

Javaには集合を扱うやり方にはString[]などの「配列」があります。しかし、オブジェクト指向「らしい」のはList<String>などのコレクション型です。

入門書では配列だけ説明し、コレクションの説明が無いこともあります。配列を使ったサンプルコードを学習するときには、コレクション型を使った書き方にもぜひチャレンジしてください。Javaでオブジェクト指向らしいプログラミングスタイルを練習するには配列よりも、java.utilパッケージに含まれるコレクション型を使った練習を徹底的にやるほうが効果的です（**リスト2**）。

このコードの断片（スニペット）を参考に、実際のプログラムを書いて動かしてみてください。

ポイントは「型」と「クラス」の違いを理解することです。

リスト2の①、②、③で、変数`a,b,c`はどれも、`Set`「型」で宣言しています。

3つのオブジェクトはそれぞれ別の「クラス」のコンストラクタを使って作っています。

ここが、初心者が間違って覚えやすいところです。

▼リスト2　コレクションのサンプル

```
String[] data = {"one","two","three","one"};
List<String> list = Arrays.asList(data);

Set<String> a = new HashSet<String>(list);          ←①
Set<String> b = new LinkedHashSet<String>(list);    ←②
Set<String> c = new TreeSet<String>(list);          ←③

System.out.println(a);
System.out.println(b);
System.out.println(c);
```

× クラス名 a = new クラス名(引数)
○ 型名 a = new クラス名(引数)

右側の「クラス名」と、左側の「型名」は同じとは限りません。

クラスは「型」の1つですから、クラス名を型名に使うことはできます。

しかし、この例では「型」はSet型です。new演算子で呼び出している「クラス」とは異なる型を宣言しています。

HashSetクラス／LinkedHashSetクラス／TreeSetクラスのオブジェクトを、どれもSet「型」として同じ「型」として扱っています。

この「異なるクラスのオブジェクト」を「同じ型」として扱うことがオブジェクト指向らしいプログラミングの、たいへん重要なポイントです。

オブジェクトは「部品」です。変数の「型」はその部品をはめ込む「差し込み口」の形状です。同じ形状の差し込み口に、異なるクラスから作ったオブジェクトを差し込むことが可能です。さまざまなタイプの部品(オブジェクト)の組み合わせ方を柔軟に変更できるわけです。

この部品の組み立ての柔軟さこそ、プログラムの変更を容易にするオブジェクト指向の狙いでありメリットなのです。

列挙型(enum宣言)

Javaでオブジェクト指向らしさの第3段が列挙型(enum)です。

列挙型を単なる定数宣言として使うだけでは、オブジェクト指向らしさは習得できません。

リスト3のように、コンストラクタを使って、定数ごとに異なる値を設定し、**label()**のように、定数ごとに異なる値を返すメソッドを用意することが、オブジェクト指向らしい列挙型の使い方です。

リスト3の10行あまりのenumには、オブジェクト指向らしい考え方が凝縮されています。if文/switch文を使わないオブジェクト指向らしい書き方の簡単で強力な手段がJavaの列挙型です。

この例のように、コンストラクタとメソッドを持った列挙型をいろいろ試してみてください。

オブジェクト指向らしさを早くから体験する

ここで紹介した、BigDecimalクラス／コレクション／列挙型は、「Java入門レベル」には登場しない内容です。

しかし、こういうオブジェクト指向らしい仕組みに早くから触れることが、オブジェクト指向を習得するために必要であり、また効果的なのです。

3つの練習課題は、初心者にはどれもハードルが高いと思います。

そのハードルにチャレンジするためには「先輩からうまく教わる」とか「参考になる情報をうまく見つける」という「学び方」の工夫が必要です。

Javaプログラマとして成長するためには、この「学び方」の工夫と練習こそいちばんたいせつで効果的なことかもしれません。SD

▼リスト3　enumのサンプル

```
enum Guest
{
  adult("大人"),
  child("子供");

  private String label;

  private Guest(String label)
  {
     this.label = label;
  }

  public String label()
  {
      return label;
  }
}
```

第2部 オブジェクト指向の克服編

その2 急がず・慌てず自然なペースでオブジェクト指向を学ぼう!

Writer 山本 裕介(やまもと ゆうすけ) ㈱サムライズム／Twitter @yusuke

オブジェクト指向学習に近道なし

「これで私は英語がペラペラになりました!」といった記事や広告はよく見かけますが、「私もこれを見習ったらペラペラになりました!」という話はあまり聞きません。人それぞれ勉強方法には向き不向きがあります。オブジェクト指向も英会話と同じく誰もが同じ方法で身につくものではありません。

筆者は小学校から大学までBASICをはじめとする非オブジェクト指向言語に10年近く慣れ親しんだあとにオブジェクト志向にとりかかりました。今どきBASICからプログラミングを始める方はなかなかいないと思いますが、ここでは1つの例として筆者がオブジェクト指向を覚えた過程や、習得のコツを書きたいと思います。

オブジェクト指向はなぜ難しいか

オブジェクト指向の学習が難しい理由の1つとして、オブジェクト指向という言葉が表す領域が広いことが挙げられます。。オブジェクト指向はモデリング手法やデータベースのアーキテクチャまであらゆる方面に応用可能な魔法の技術のように、そしてときにはごちゃごちゃに説明されます。「オブジェクト――つまり『物』――をプログラムで表現できるため、現実世界をプログラムでそのままモデル化・表現できる」といった説明を見たことがあるのではないでしょうか? あれがウソだとは言いませんが、あまり真に受けないほうがよいです。広く使われているオブジェクト指向技術は、あくまでプログラムの拡張性、柔軟性、保守性を上げるための表記方法に過ぎず、オブジェクト指向モデリングやオブジェクト指向データベースといった技術が使われる場面は非常に限られます。

さっぱり理解できなかった大学時代

小学生のころからプログラミングに親しんできた私がはじめてオブジェクト指向に出会ったのは大学3年生になります。研究室で「オブジェクト指向に親しもう」といった主旨で院生が課題を出してくださいました。その課題の冒頭はJavaコードで確かこのように書かれていました。

```
Person person = new Person();
```

さすがにJavaを10年以上やってきている今はすんなりと理解できますが、当時はさっぱり意味がわかりませんでした。「Person型の変数personを定義し、Personクラスのインスタンスを代入する」「Personというクラスのひな形を元に実体化する、上位クラスとしてHuman型やAnimal型を定義することもできる」といった説明を受けた気がします。「クラスは鯛焼きの型で、生地と餡(あん)を垂らして焼いてできた物がインスタンスにあたる」という説明もよく聞きますね。はい、意味がわかりません!

筆者は結局この「パーソン パーソン イコール ニュー パーソン」の意味をよく理解しない

まま大学を卒業しました(よく理解しないながらも見よう見まねでコードは書けたのですが)。

プログラムの中の世界をAnimalだとか鯛焼きの型だといった具体的な物でたとえるのは無理がある気がします。変数や配列を「箱のようなものだ」とするたとえが理解できずつまずく人も多いですね。筆者と同じくたとえ話ですんなり理解できない方はあまり気にせずひたすらコードを読み、書き、実行して感覚的に身に付けていくのがお勧めです。

デザインパターンとの出会い

オブジェクト指向の本質を理解しないまま社会人になった筆者を待ち受けていたのがデザインパターンの研修です。研修ではJavadocで書かれたインターフェース仕様が提示され、仕様に基づいた実装を行うというものでした。実装が完了したら講師にコードレビューをしてもらう……のではなく、指定されたスクリプトを実行するだけでした。

画面には...E..F..といった形で結果が表示され、EもFも表示されなければその課題は合格、そこではじめて講師のコードレビューと解説が行われました。つまり受け入れテストがJUnitのようなもの(JUnit.orgが誕生したのがちょうど同時期の2000年なので恐らくJUnitそのものは使っていませんでした)で書かれており、どういった項目をテストしているかは隠されているというなかなか面白い実習形式でした。そして提示されたインターフェースはデザインパターンを適用する必要のある仕様になっており、自然とデザインパターンを学べる……はずでした。怠慢な筆者はデザインパターンの本質を理解せず、とにかくゴリ押しでテストが通るように実装した覚えがあります。

ゲーム開発で身につけたオブジェクト指向

研究室で与えられた課題はイマイチ理解できず、会社のデザインパターンの研修も「とにかくこなした」不出来な、またはうまいこと世渡りした筆者ですが、期せずしてオブジェクト指向は独学で習得できてしまいました。きっかけは各言語で作ってきた習作です。どの言語を始めるときも、その言語の特性を理解するために同じテーマのコードを書いてきました。テーマは2つあって1つはコンピュータ対戦型のオセロ、もう1つはチャットプログラムです。BASICやZ80のアセンブリ言語で書いたオセロはまったくをもってオブジェクト指向ではありませんでしたが、Javaで書いた際は「オセロの盤面をオブジェクトとして表現するには?」「プレーヤをオブジェクトとして表現するには?」「プレイのターンの概念はオブジェクトにするべき?」などと考えながら書きました。そしてチャットプログラムではサーバソケットという抽象的な存在ですら具象化(インスタンス化)して始めて利用できるようになる、といった概念を学んだ覚えがあります。

オブジェクト指向モデリングは使わない

オセロのモデリングではオブジェクト指向モデリングは非常に難しいことを実感しました。今言わせてみればオセロのモデルをどのようにJavaクラスに落とし込むのが良いかなどと頭を捻(ひね)る必要はありません。現実世界はいかようにでもモデル化ができ、唯一の正解というものはありません。以前であれば「最初にした設計は絶対」であり、一度動いたシステム、ある程度デバッグの進んだプログラムの設計に変更を加えるのは一種のタブーでした。しかし今はモデルにいささか不都合があったとしてもJUnitのようなユニットテストツール、Jenkinsを始めとする継続的インテグレーションツール、Gitを始めとするバージョン管理ツール、仮想化技術、そして進化したIDEの助けを借りて以前と比べると一段と安全に、すばやくリファクタリング(挙動を変更せずにプログラムの記

述、設計を改善すること)が行えます。ですので、優れたモデルを時間をかけて生み出すよりも、手っ取り早く動き、かつ十分な保守性を備えるバランスの取れたコードを生み出すほうがよほど価値があります。

そして実際の業務でシステム化対象をモデル化して分析、などということはめったにありません。オブジェクト指向はあくまでプログラムの便利な記載方法であり、プログラムをシンプルに、そして拡張性と保守性を高めるための手段と割り切ったほうが良いでしょう。

GUIプログラミングで覚えるデザインパターン

デザインパターンは非常に重要です。しかし、そもそも習得の難しいオブジェクト指向が前提となっているデザインパターンは当然ながらさらに難しいです。そして全部で23あるデザインパターンのうち、普段目にするのはほんの数パターンです。デザインパターンを勉強したいと思ったら全パターンを頭に叩き込むのではなく、最初はどんなものがあるかさらっと眺める程度で良いでしょう。筆者の場合は研修でよく理解しないながらも一度叩き込まれたので、そのあと実践の場で「ここはデザインパターンが適用できるかも!」「適用できた!」「考えてはいなかったけれどもよくよく見てみればデザインパターンが適用されていた」といったことが多くありました。オブジェクト指向の理解が進むにつれてデザインパターンも理解できるようになってきます。デザインパターンがしっかりと身につくまでは1年に1回くらいの周期でどんなパターンがあるか反復的に復習するほうが効率はよさそうです。またGUIプログラミングにはデザインパターンが欠かせません。今はWebシステムが流行ですが、デザインパターンを習得するためにGUIプログラミングをやってみるのはたいへん有効です。筆者はチャットプログラムをデスクトップアプリケーションとして開発していましたので、デザインパターンの多くをAWTやSwing(JavaのGUIプログラミング用API)から学びました。自然とAbstract Factory、Composite、Observer、Strategyといった重要なパターンが身につくことでしょう。

オブジェクト指向、デザインパターンは極力使わない!

最後になりますが、これからオブジェクト指向の理解が進んで行くであろう皆さんに肝に銘じてほしいのが「オブジェクト指向やデザインパターンを使わないようにする」ことです。とかく凝った継承関係を持たせたり、デザインパターンを適用したりすれば自己満足はできますが、多くの場合はむやみやたらとクラス数が増えてむしろ冗長になってしまうだけです。「ここの部分を将来的に拡張できるように」などと設計しても多くの場合その部分を拡張する必要は出てこないものです。これはYAGNI(You Ain't Gonna Need It)——どうせあなたはそれを必要としない——などと呼ばれるeXtreme Programmingやアジャイル界隈では有名な原則です。つまり、拡張できるように事前に設計しておく、よりも「拡張する必要が出てきた際にコードをDRY(Don't Repeat Yourself)にまとめる」ために初めてオブジェクト指向やデザインパターンを適用するのが良いでしょう。いっそのことオブジェクト指向やデザインパターンは現代的なプログラミングをするうえでの教養と割り切ってもいいかもしれません。イマドキのフレームワークやライブラリ、APIを利用していれば、もはやオブジェクト指向を理解せずともプログラミングはできてしまいます。しかし、オブジェクト指向がわかっていればフレームワークやライブラリの設計の理解は進み、トラブルシューティングもより自信を持って行えることでしょう。果ては自分がフレームワークを設計する立場になった際、オブジェクト指向とデザインパターン、そしてプログラミングの実践的な経験ほど強力なものはありません。SD

第2部 オブジェクト指向の克服編

その3

社会慣習としてのオブジェクト指向プログラミング

Writer 柏野 雄太(かしの ゆうた) kashino@bakfoo.com／Twitter @yutakashino

オブジェクト指向プログラミングは慣習である

始めから暴論を投げるようで恐縮ですが、オブジェクト指向プログラミング(Object Oriented Programming、以下OOP)とは、「世界をオブジェクト同士のメッセージのやり取りで捉える」というようなポエティックな建前的説明をするより、世界中のプログラマ間で合意された一種の社会的な慣習に過ぎないと説明するほうがより実相に近いと筆者は考えています。社会慣習ですから、人のコードを使ったり、人にコードを使ってもらうにはこれを守る必要があります。そして、多くの慣習がそうであるように、最初は右も左もわからなくてウロウロしていても、その集団にいるうちにいつの間にかその慣習に染まっているというのがOOPにも言えます。

ただし、ここで「OOP」と漠然に述べたときに厄介なのは、プログラム言語によって意味するものが著しく異なることです。たとえば、C++では、クリエイタのビャーネ・ストロヴストルップ(ストラウストラップ)は「型を新たに定義できるデータ抽象化機能をもち、継承などの手法で型を階層化できるものがOOPだ[注1]」としています。

一方で最初に「オブジェクト指向」という言葉を生み出し、Smalltalkのクリエータの一人として有名なアラン・ケイは「(OOPとは)生物の細胞のように、独立したオブジェクトがメッセージをやりとりすることが何より大事だ。……Simula 67のような継承の方法は嫌いで、もっと良い方法が見つかるまで継承は(自分の言語の実装から)取り除くことにした[注2]」と、多くのOOPで必須であるとされる継承を、OOPには本質的ではないとします。

混乱してきました。「継承」というオブジェクト指向における最重要の概念を巡っても、1つの言語では大事であるといい、他の言語では本質的でないという。いったい何が正しいのでしょう。

大きなコードを複数人で書く作法

OOPの根底にある問題意識は、実は1960年代に起こった「ソフトウェア危機[注3]」に端を発しています。1960年代になると大型計算機の発展により、大人数による大規模なソースコードが量産されはじめます。プログラミングは現実の問題を計算機で解決するための手段です。そのために、プログラミングは現実の問題をコンピュータがわかる形で表現し写し取ります。これをモデル化といいます。そして大きく複雑な問題に対しては、モデル化されたプログラムも大きく複雑なものになります。大きく複雑になったプログラムは、ソフトウェアの開発継続性、開発コスト、そしてメンテナンスコストが問題になります。

その結果、大きなソースコードベースでも壊れない、可読性が高く検証可能なプログラミン

注1) Bjarne Stroustrup "What is 'Object-Oriented Programming'?" (http://www.stroustrup.com/whatis.pdf)

注2) Dr.Alan Kay on the Meaning of "Object-Oriented Programming" (http://userpage.fu-berlin.de/~ram/pub/pub_jf47ht81Ht/doc_kay_oop_en)

注3) http://ja.wikipedia.org/wiki/ソフトウェア危機

社会慣習としてのオブジェクト指向プログラミング その3

▼リスト1　exapmle.py（サンプル）

```
# exapmle.py
# coding:utf-8

class machine(object):                          ←①
  def __init__(self, message):                  ←②
    self.message = message                      ←③

  def play(self):                               ←④
    print('recording message: {0}'.format(self.message))

hellomachine = machine('こんにちは')             ←⑤
goodbymachine = machine('さようなら')

hellomachine.play()                             ←⑥
goodbymachine.play()
```

グの手法が方法論として必要になってきたのです。その方法論を追求する過程で、ダイクストラの構造化プログラミング[注4]などのソースコードを整理する手法が提唱されました。その流れにあるのがOOPです。ですから、複数人が大きなプログラムを壊れることなく書き続けるためのお作法を、あらかじめプログラム言語に組み込んでおこうというのがOOP言語で、OOPが広くプログラマの社会慣習として通用しているのです。

つまり、OOPの目的とは、（言語が異なれば機能や実装などに違いはありますが）大きなプログラムを複数人で共有しながら書き上げるためのお作法である、ということです。

Pythonを例にとって

大きく複雑なソースコードでも壊れることなく書き続けるにはどうすればいいのでしょうか？

それは現実の問題をソースコードに写し取ったときに意味的に文節化できる単位で1つにまとめることです。これを**モジュール化**または**隠蔽化（カプセル化）**といいます。プログラミングとは端的に言うと、解決するべき問題を**データ**と**関数**で表現することですから、**モジュール化**も**データ**と**関数**で行われることになります。このモジュール化の単位をOOPではしばしばクラスと呼びます。クラスにおいてデータである**クラス属性**と関数である**クラスメソッド**を持つのはその理由です。

さらには、冗長さを無くして見通しを良くするには、まとめたクラスなどのモジュールを賢く再利用する必要もあります。そのときに使う慣習的な手法が**継承（インヘリタンス）**や**ミックスイン**などです。紙幅の関係上、ここでは取り上げませんが、実はOOPを現実の問題に適用すると、これらの再利用手法はいろいろな問題を生みます。そしてその問題を解決するための長年に渡る複雑な議論があるのです。

さて、Pythonにおいて**クラス**を作るには**class**文を利用します。つまりPythonにおいてOOPを行うには**class**に習熟する必要があります。Pythonにおける非常に簡単で典型的な**class**の利用は**リスト1**のようになります。

リスト1を簡単に解説しますと、①class文によりクラスの記述を始めます[注5]。②__init__メソッドはコンストラクタメソッドとなっていて、クラスからオブジェクト（インスタンス）を作るときに最初に実行されます。Pythonにおいてselfは自分自身のクラスオブジェクトを指し示すための特別な意味を持ちます。③self.messageがこのクラスのクラス属性です。④このクラスにはplayというクラスメソッドを1つだけ持たせます。⑤クラスからオブジェクト（インスタンス）をつくります。machineクラスを原型として、異なる内部データを持ったオブジェクトを2つ作っています。⑥それぞれのオブジェクトのメソッドを実行するには、オブジェクト名とメソッド名を.（ドット）で挟みます。

リスト1を実行すると、それぞれのオブジェクトが内部に持っているデータ・関数が利用され、次のような結果になります。

```
$ python machine.py
recording message: こんにちは
recording message: さようなら
```

注4） http://ja.wikipedia.org/wiki/構造化プログラミング

注5） classではクラス名を書くとともに、継承するべき親クラスをカッコで指定します。ここではobjectクラスを親クラスとして指定しています。実はPythonではすべてのユーザ定義クラスはobjectというPythonのもっとも基本となるクラスが継承することになっているので、クラス文にobjectを継承するように明示するのです（https://docs.python.org/release/2.2.3/whatsnew/sect-rellinks.html）。

オブジェクト指向習得の良書紹介

今まで述べてきましたように、OOPは、ソースコードを流通させるための、プログラマの間で広く行き渡った社会慣習ですから、これをマスターしないと人のコードが読めませんし、人に使ってもらえるコードを書くために必要な作法の大きな部分が欠落するという事態に陥ります。それでは、そのような慣習はどうやってマスターすればいいのでしょうか。それには端的に言うと、良い教材でトレーニングをするしかありません。

ここではPythonにおいてオブジェクト指向プログラミングを学ぶために有効ないくつかの定評のある教材リソースを以下に紹介します。

・『MIT Open Cource Ware "Introduction to Computer Science and Programming"[注6]』

まずは「計算機プログラムの構造と解釈」という名著を生み出したとMITの入門講義の後継です。いまはSchemeでなくPythonで講義を行っています。オブジェクト指向プログラミングは"14: Introduction to Object-oriented Programming"　34:00くらいから始まります。

・『Learn Python Hardway[注7]』

次は有名なハッカーのゼド・ショウの明快かつ本質を突いたPython入門コースです。このコースを手を動かしながら写経のように打ち込み理解すれば、Pythonのイロハが身体に身につきます。オブジェクト指向プログラミングは“Exercise 40: Modules, Classes, and Objects”[注8]から始まります。

・Pythonの標準ライブラリのソースコードを読む[注9]

そして最後に、Pythonの標準ライブラリのソースコードを読むことはとても勉強になります。ある意味これがPythonにおける最後のオブジェクト指向プログラミングのお手本といってもいいです。jsonライブラリでも良いですし、loggingライブラリでも良いです。

ただし……

ただし、これまでいろいろと記したうえであえて言いたいことがあります。実はPythonは、ソースコードとディレクトリ自体が`__init__.py`という名前のファイルがあればひとまとまりのモジュールとパッケージを作ることができる仕様[注10]ですので、Pythonモジュールとパッケージを使えば可読性があり再利用できる、十分に大きなプログラムを作ることができます。従って、Pythonにおけるオブジェクト指向プログラミングは、Javaのように必須ではありませんし、教条的な利用をすることもないというのが筆者の意見です。

結論

この稿のポイントは次になります。

・OOPは広くプログラマで共有された社会慣習なので、好き嫌いにかかわらず、プログラマとして覚えておいたほうが良い
・OOPの定義は言語や言語設計者によりいろいろと異なり混乱を生みがちだが、本質は大きなプログラムを複数人で壊さず書くようにすることを目的としたものである
・OOPを取得するにはトレーニングしかないが、そのためのPythonにおける効果的なトレーニングリソースを紹介した

以上の雑文が読者の皆様のオブジェクト指向プログラミングの理解に少しでもお役に立てば幸いです。SD

注6）http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-00-introduction-to-computer-science-and-programming-fall-2008/video-lectures/lecture-14/
注7）http://learnpythonthehardway.org/book/
注8）http://learnpythonthehardway.org/book/ex40.html
注9）https://github.com/python/cpython/tree/master/Lib
注10）Python2.7 module（ https://docs.python.org/2.7/tutorial/modules.html, Python3.4 module: https://docs.python.org/3.4/tutorial/modules.html）

第2部 オブジェクト指向の克服編

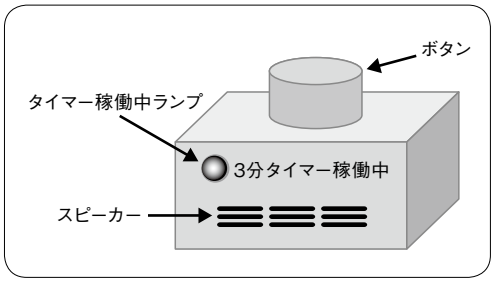
その4 組込エンジニアのためのオブジェクト指向

Writer 星野 香保子(ほしの かほこ) ㈲テクノランド

組込システムには状態遷移表!

状態遷移表をご存じですか？ いきなり唐突な質問ですが、組込システムを設計するときには状態遷移表がよく使われています。簡単な例を挙げます。たとえば**図1**のような「3分タイマー装置」を作るとします。ボタンを押すとタイマーが開始し、3分経過すると「ピピッ」と音が鳴る、という機能を持つ簡素な装置です。タイマー稼働中はランプが点灯し、3分経過するとランプは消灯します。

▼図1 3分タイマー装置



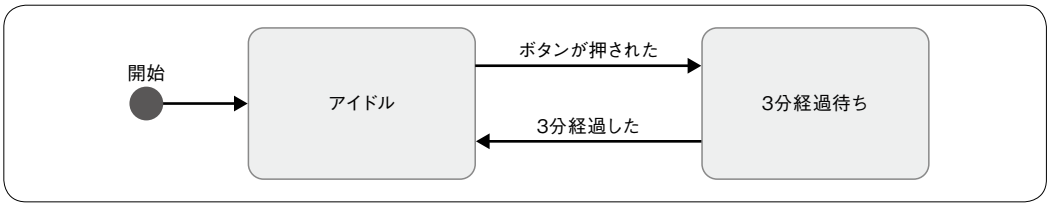
イベントと状態とは

組込システムは、動作中にさまざまなイベントを受け付け、イベントに対応する処理を行いながら動き続ける、という特徴があります。この3分タイマー装置では、「ボタンが押された」、「3分経過した」というのがイベントに該当します。各イベントが発生したときの処理と状態の変化を表にしたものが状態遷移表(**表1**)です。状態遷移図(ステートマシン図：**図2**)も設計時には有用で、状態が変化する流れを図で表現できます。今回作成する3分タイマー装置では、最初の状態は「アイドル」です。ボタンを押すと「3分経過待ち」状態に遷移し、3分経過すると「ア

▼表1 状態遷移表

イベント＼状態	アイドル	3分経過待ち
ボタンが押された	ランプを点灯する 「3分経過待ち」へ	(無視)
3分経過した	(不可)	ランプを消灯してブザーを鳴らす 「アイドル」へ

▼図2 状態遷移図(ステートマシン図)



イドル」に遷移します。

オブジェクト指向でない場合

さて、これらの状態遷移表と状態遷移図を基にしてプログラムを作成する場合、一番簡単なかたちとしては**リスト1**のようになるでしょうか。現在の状態を保持するフラグ変数を用意し、イベント発生時にはフラグ変数を見て、そのときの状態に応じた処理を行います。この方法はとくに問題があるわけではありませんが、イベントの種類や状態の種類が増えた場合、switch文の分岐が増えていき複雑なコードになりそうです。

オブジェクト指向を取り入れてみる

そこで、装置の「状態」に着目し、発想を転換してオブジェクト指向的な考え方を取り入れるように変更してみます。**図3**のようなクラスを作成してみました。Deviceクラスが3分タイマー装置を表しています。そしてここがポイントなのですが、状態ごとにクラスを作成し、その状態のときに発生したイベントに対応する処理を実装します。

ポイントは状態ごとのオブジェクト

プログラム例を**リスト2**から**リスト6**に示します。コンソール上で模擬的に実行した結果が**図4**です。ここでは、アイドル状態(IdleState)と3分経過待ち状態(WaitState)のクラスを作っていて、つまり、状態そのものがオブジェクトになります。現在の状態はDeviceクラスが保持していて、イベントが発生したときは、現在の状態オブジェクトを使って対応する処理を呼び出します。ボタンを押したときは、「アイドル状態」から「3分経過待ち状態」に遷移しますが、**リスト2**の①で遷移先の状態への入れ替えを行っています。Deviceクラスでは、「現在どの状態なのか」を意識しなくとも、イベント発生後には次の状態に遷移するしくみになっています。もし状態の種類が増えた場合は、状態クラスを追加して対応できるので、拡張性が高く見通しの良いプログラムを開発できます。

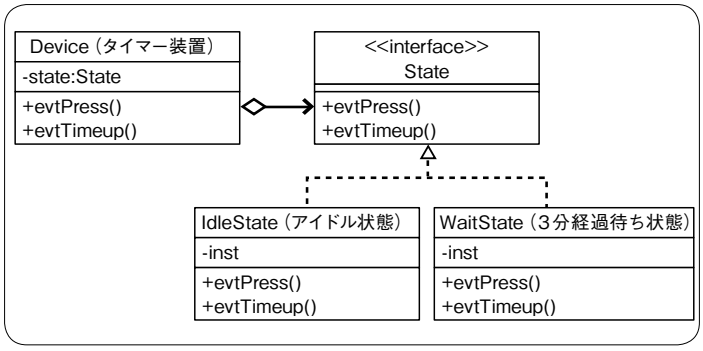
▼リスト1　オブジェクト指向でないコード

```
event = GetEvent(); // 発生したイベントを取得
switch (現在の状態) {
  case アイドル状態:
    switch (event) {
      case ボタンが押された:
      処理;
      次の状態へ遷移;
      break;
    case 3分経過した:
      break;
  }
  case 3分経過待ち状態:
    switch (event) {
      case ボタンが押された:
      break;
      case 3分経過した:
        処理;
      次の状態へ遷移;
      break;
  }
}
```

▼図4　実行結果

```
ボタン押下：ランプを点灯します。
3分経過　：ランプを消灯してブザーを鳴らします。
```

▼図3　クラス図



▼リスト2　Device.java

```
public class Device {
  private State state = null;
  public Device() {
    state = IdleState.getInstance();
  }
  public void evtPress() {  // ボタンが押された時の処理
    state = state.evtPress();                    ←①
  }
  public void evtTimeup() { // 3分経過したときの処理
    state = state.evtTimeup();
  }
}
```

▼リスト3　State.java

```
public interface State {
  public abstract State evtPress();
  public abstract State evtTimeup();
}
```

組込でも使える デザインパターン

デザインパターンとは、再利用可能な設計パターンを集めて分類したものです。オブジェクト指向プログラミングのデザインパターンで広く使われているものとしてGoF(Gang of Four)デザインパターンがあります。今回作成した「状態そのものをオブジェクトにする」というプログラムは、GoFデザインパターンのStateパターンを取り入れたものです。

GoFデザインパターンと言えば、大規模なアプリケーションに適するものと思っている人もいるかもしれませんが、組込システムにも応用できるパターンも存在します。今回のプログラム例ではJava言語を使用していますが、C++言語やC言語などの他の言語でもオブジェクト指向的な考え方を設計に取り入れることは十分に可能です。

システムのすべてをいきなりオブジェクト指向で開発するのは難しいという場合でも、まずは部分的に利用可能なパターンがあるかどうか、デザインパターンの本を眺めてみるのも良いでしょう。組込システム開発において再利用性や作業効率をアップさせるために、オブジェクト指向を取り入れることを検討してみてください。SD

▼リスト4　IdleState.java

```
public class IdleState implements State { // アイドル状態
  private static IdleState inst = new IdleState();
  private IdleState() {}
  public static State getInstance() {
    return inst;
  }
  public State evtPress() {  // ボタンが押された時の処理
      System.out.println("ボタン押下：ランプを点灯します。");
    return WaitState.getInstance();
  }
  public State evtTimeup() { // 3分経過したときの処理
    return this;
 }
}
```

▼リスト5　WaitState.java

```
public class WaitState implements State { // 3分経過待ち状態
  private static WaitState inst = new WaitState();
  private WaitState() {}
  public static State getInstance() {
      return inst;
  }
  public State evtPress() { // ボタンが押された時の処理
      return this;
  }
  public State evtTimeup() {// 3分経過したときの処理
      System.out.println("3分経過　:ランプを消灯してブザーを鳴らします。");
      return IdleState.getInstance();
  }
}
```

▼リスト6　Main.java

```
public class Main {
  public static void main(String[] args) {
    Device dev = new Device(); // 3分タイマー装置

    dev.evtPress(); // ボタンを押したつもり

    // 3分経過待ち…

    dev.evtTimeup();// 3分経過したつもり
  }
}
```

第2部 オブジェクト指向の克服編

その5 Android開発でオブジェクト指向プログラミングするとは

Writer 江川 崇(えがわ たかし) Smartium㈱／Twitter @t_egg

プログラミング言語と向き合う

まず筆者が最も重視していることを最初に1つ述べます。それは、「オブジェクト指向でプログラミングする」ことは、「オブジェクト指向プログラミング言語でプログラムを書く」ことではないという点です。世の中のプログラミング言語は、手続き型言語、オブジェクト指向言語、関数型言語といった観点で分類されることがありますが、多くのプログラミング言語は完全に分類できるわけではなく、複数のパラダイムを併せ持っています。たとえば、Javaという言語はクラスベースのオブジェクト指向言語と呼ばれていますが、手続きや関数を書くこともできますし、書き方を工夫すればクロージャーのような使い方もできます。筆者の好きなプログラミング言語の1つであるErlangは関数型言語の1つに位置付けられていますが、(やろうと思えば)プロセス単位で状態を持つこともできます。プログラミング言語に対して世間から貼られている一般的なレッテルにあまり振り回されることなく、ぜひさまざまなプログラミング言語に触れて自分なりの解釈を培(つちか)ってください。

状態による副作用と自律性

筆者は、もともとC言語で制御系のプログラムを書いていました。当時筆者が関与したプログラムは、入出力が厳密に決まっており、いかに正確かつ高速に処理を実現するかという点にすべての価値がありました。Cは言語仕様がシンプルで覚えやすい言語です。Cでは「関数」で振る舞いを定義します。関数とは、あるデータを渡すと処理を実行して結果を返す命令のことです。関数は状態を持ちません。そのため、同じ入力に対する出力は何度実行しても常に同じです。

しかしながら一般的にはグローバルな領域に変数を定義して利用する方法が採用されることがあります。この方法は副作用を生み、関数間の依存性が高くなるため筆者は好きではありませんし、世間でもあまり推奨されませんが、必要悪として広く使われているのが実状です。このことは、手続きをプログラムとして表現するうえで状態をセットにして考える方が理解しやすい局面があることを示しているのではないでしょうか。

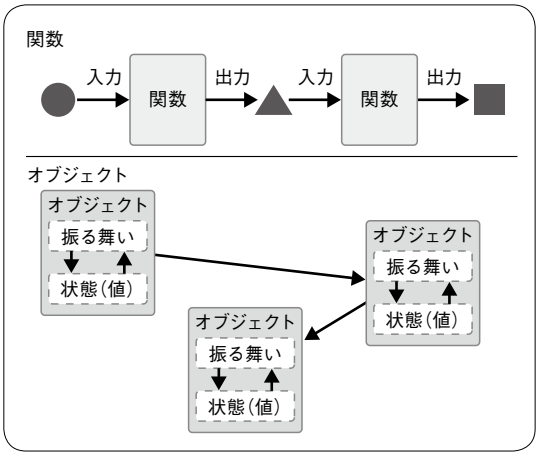
Cとは異なるパラダイムから生まれたものに、クロージャーやオブジェクトといった考え方があります。どちらも振る舞い(関数)と値やデータ構造とを任意の単位でまとめることができるものです。お互いの要素の自律性が高まりますが、外部や内部の情報(環境や状態)に処理が依存するため、副作用が生じる懸念点があります(**図1**)。

オブジェクトの利点

すでに述べたとおりオブジェクトは副作用を含んでいますが、ある程度自律性のあるかたまりの単位で物事を分類・分解できるため、複雑なものを抽象的にとらえることに向いています。

Android開発でオブジェクト指向プログラミングするとは その5

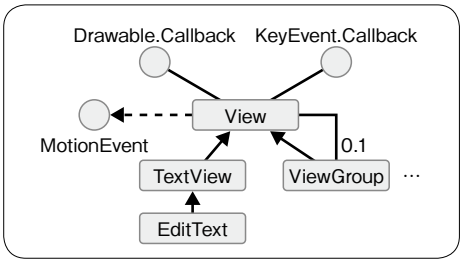
▼図1　関数とオブジェクトのイメージの違い



言い換えれば、自分なりの解釈で説明・表現するための道具として適しています。日常生活での一般的な考え方と似たアプローチや語彙で説明できると言ってもよいでしょう。

あなたがAさんに対してBさんのことを説明する状況を想像してください。おそらくたいていの人は、Aさんが知っていそうなことや、AさんとBさんとの間にある何らかの共通点をきっかけにして説明しようとするはずです。AさんとBさんとの間に共通の知人のCさんがいれば、Cさんのことを話題にするでしょうし、趣味や学校、職業などで共通点があればそのことを話題にするかもしれません。Aさんが記憶している印象的な出来事にBさんが関与していたら格好の話のネタになるでしょう。Bさんについての事実を単純に列挙していくよりも、その方がAさんの興味を惹きつけやすく説明しやすいからです。

▼図2　UI部品の構造（一部のみ）



オブジェクトの考え方は、物事に対する分類や分解の考え方です。オブジェクトはあくまでも思考や表現の方法であり、プログラミング言語の選択に対して大きな制約を与えるものではありません。オブジェクトを使うと、自分の解釈・分類を表現しやすくなり、結果として他者と概念を共有しやすくなります。

Android SDKでの例

ここでは、Android SDK（Software Development Kit）のJavaソースコードから画面まわりの基本的な部品をいくつか抜粋します。Javaではまず型を定義して分類するプログラミングスタイルが一般的ですので、型を中心に見ていきます。

Androidでは、画面を表現するUI部品は、すべてViewです。そして、文字を表示するViewはTextView、文字の入力を受け付けるViewはEditText、複数のViewを束ねてレイアウトするViewGroupといったように、部品の用途に応じて細かなオブジェクトを定義しています。

COLUMN

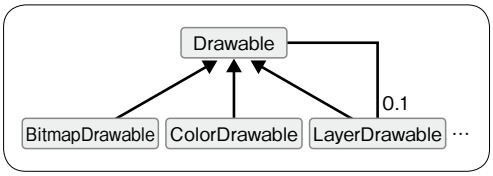
オブジェクトとレビュー

筆者の業務では日常的に分析や設計の成果物やソースコードをレビューする機会があります。筆者のレビューの主な観点は次の2点です。

・分類や整理の観点を自分が理解できるか
・成果物内で一貫しているか

決して、自分と同じ考え方かどうか（筆者が書くものと同じかどうか）という観点では見ません。このレビューの考え方とオブジェクトの考え方は似ています。一貫性のある適切な粒度のオブジェクトに分割されたプログラムは、その人の思考が浮き彫りになります。ある程度の期間レビューをしていると、限定されたメンバーの中では成果物の断片を見ただけで誰が書いたものかわかるようになりますので、利き酒ならぬ利きソースコードといったちょっとした遊びもできます。

▼図3　Drawableの構造（一部のみ）



画像や色などの実際に画面に描画されるものは、Drawableです。Drawable（描画できるもの）という命名は非常に理解しやすく秀逸ではないでしょうか。Drawableも、ビットマップ画像を描画するBitmapDrawableや、複数のDrawableを重ねて描画できるLayerDrawableのように用途に応じてさらに分類されています。

自分ならどう書く?

なぜこのように実装しているのか、自分ならどう書くかを意識しながら、世間で幅広く使われているソースコードを読んでみると非常にいい勉強になります。筆者は、Android SDKのソースコードを初めて読んだ際に、重要な要素の分類に継承という言語機能を多用していることに違和感を感じました。とくにJavaの場合、継承はひとつしか取れませんので非常に強い関係となることから大きなリスクがあるためです。

Android 4.4.2のソースコードをベースにしたViewまわりの行数（コメント含む）を示します。

・View……2万行
・TextView……820行
・EditText……130行

筆者はViewやその周辺のソースコードを確認する中で次のような意志表示を読み取りました。

・UI部品として最低限必要と思われる処理をView内に閉じておきたい
・UI部品としての用途に絞り、実際に使われる各部品での記述量をなるべく減らしたい
・そしてそのためには継承を使うことも辞さない

この設計の是非はここでは述べません。当然異なる実装方法も考えられますが、少なくともこの設計の意図は読み手に伝わってきます。

まとめ

最後に、ぜひ世の中で広く使われているものや、世間からよいと評価されているものをただ使うだけでなく、それらのソースコードをたくさん読んで考え方に触れてほしいと思います。そのことが自分ならどう書くかを考えるきっかけとなり、自分の生きた知識として身につきます。本記事が皆さんのこれからの開発者人生を考える一助となれば幸いです。SD

COLUMN
オブジェクトとモバイル

モバイルデバイス上で動作するプログラムはリソースが限られるため、高速に動作させるためにはなるべくオブジェクトを作らないほうがよいという話を耳にすることがあります。また、Androidの実行ファイル（.dexファイル）には、メソッドを6万4千メソッド以下にしなければならない「64K問題」などがあります。通常はそこまでの数に達することはないものの、GoogleのJavaライブラリであるGuava[注1]は1万2千メソッド程度、Googleのサービスと連携するライブラリであるGoogle Play Servicesは2万4千メソッド程度と、数が多いものがあります。そのようなライブラリと組み合わるために、メソッド数を減らす目的でオブジェクトを分割しない開発スタイルを採用することもあります。

もちろんPCなどと比較するとモバイルデバイスでは依然として制約を意識したほうがよいことは事実ですが、現在のデバイスは性能が格段に向上しています。64K問題もdexを分割するなどの回避方法があります。これらの制約を理由にプログラミングのスタイルを極端に曲げざるを得ない局面は日々少なくなっています。

注1）https://code.google.com/p/guava-libraries/

第2部 オブジェクト指向の克服編

その6 オブジェクト指向はまぼろしか?

Writer きしだ なおき http://d.hatena.ne.jp/nowokay/

オブジェクト指向なんかないよ!

オブジェクト指向の解説ということで定番の説明をまとめようと思ったのですが、オブジェクト指向の説明で挙げられているようなことを、筆者がプログラムを書くときには、考えてないことに筆者は気づきました。

そこで「オブジェクト指向を学ぶには」という記事ではありますが、「これをやればいいプログラムになるというようなオブジェクト指向なんかないよ」ということを書いていこうと思います。

オブジェクト指向の歴史

まず、簡単にオブジェクト指向の歴史をまとめておきましょう。

オブジェクト指向の要素をもった最初の言語は1967年のSimulaだと言われています。そして70年代Smalltalkが現れ、80年代すでに主流となっていたC言語にオブジェクト指向機能を追加したC++が出てから、一気にオブジェクト指向の考えが広まります。

90年代にはさまざまなオブジェクト指向方法論が乱立しました。スリーアミーゴズと呼ばれたブーチ、ヤコブソン、ランボーは記法や用語の異なる方法論を統一しようと試みます。

また90年代にはWindowsが使われるようになりCUIが主体だったソフトウェアがGUIへと移行し始めます。オブジェクト指向はGUIコンポーネント構築と非常に相性がよく、これもオブジェクト指向が必要とされた要因になりました。デザインパターンなど実装に対する理解も進みます。その中で、1995年、オブジェクト指向を設計の中心とするJavaがリリースされます。そして勢いは加熱し、オブジェクト指向はマーケティング用語となり、多くのソフトウェア製品がオブジェクト指向を謳(うた)います。

一方でブーチらの統一手法作成は難航し、まずは記法だけを定義したUMLがリリースされます。しかしながら90年代後半には、「よいプロセスがよいプロダクトを作る」として、分析・設計方法論から開発プロセスへとソフトウェア工学の主流が移りました。ブーチらの統一手法も、分析・設計方法論ではなく、統一プロセスとして発表されました。つまり、オブジェクト指向として統一された手法が現れることはなかったということです。

2000年代に入ると、Webアプリケーションが広まり始めます。勢いにのっていたJavaもオブジェクト指向をベースとしてフレームワークを出しましたが、無駄に手順を増やしていただけという結果に終わりました。

2005年ごろには、名前による処理関連付けを主体とするRuby on Railsが流行り始め、オブジェクト指向主体で設計されていたJavaのフレームワークが否定され、関数型の機能を持った言語が注目され始めます。2014年、今年3月には、Javaにも関数を式として記述する表記が導入され、主要な言語のほぼすべてが関数型の機能を持つようになりました。オブジェクトだけをコンポーネントとする言語がなくなり、これによってオブジェクト指向の時代が完全に終わったと言うことができるかもしれません。

オブジェクト指向とは何であったか

ランボーは、オブジェクト指向を「データ構造と振る舞いが一体となったオブジェクトの集まりとしてソフトウェアを組織化すること」としました。これは当時一般的な定義でした。

またオブジェクトの特性として分類(カプセル化)・継承・多相性(多態)を挙げ「同じデータ構造と同じ振る舞いを持つオブジェクトを1つのクラスにグループ化する」「階層関係に基づいてクラス間で属性と操作を共有する」「同じ操作であっても異なるクラスに対しては異なる振る舞いをする」としています。

そして、分析段階で見いだされたオブジェクトがそのまま設計、実装へと引き継がれることから開発手法の本命とされました。

オブジェクト指向はいかに使われなくなったか

手法

2000年代になってアジャイルプロセスが広まりだすと、分析段階で実装に反映されるオブジェクトを決めてしまうことが馴染まなくなりました。そこで、分析段階ではユースケース駆動、責務駆動、ドメイン駆動のように、より抽象度が高く実装に依存しない単位を対象にする手法が使われるようになります。逆に実装段階では、テスト駆動開発のように、実装のみを対象にした手法が注目されるようになりました。

業務システム

オブジェクト指向が注目されなくなった背景には、オブジェクト指向は業務システム開発にはあまり向かなかったということもあります。業務システムではデータベースを利用するためデータと処理が分離します。開発は、データの関連を設計し、ワークフローを整理しながら画面を抽出するといった流れになります。構造としては、プレゼンテーション層、ロジック層、データベース層のようなレイヤ構造のほうが向いています。

Webアプリケーション

Webの発展とともに、HTTPとHTMLをベースとしたWebアプリケーションが広まっていきます。サーバ側は、HTTPレスポンスをうけとり、それに従ったHTMLを返すという独立したハンドラの集合として実装されます。

リクエストからレスポンスまでの1往復の処理にオブジェクト構造は必要なく、構造として関数的であることから、関数型の機能をもったプログラミング言語が注目されていきました。

崩れるJavaのオブジェクト指向

オブジェクト指向の熱狂の中で発表されたJavaは、オブジェクト指向言語の代表として扱われていました。しかし、バージョンが進むに従って、オブジェクト指向的ではない機能を取り入れています。

アノテーション

Java2SE 5.0では大幅に文法が変更されました。そのときに導入された構文のひとつがアノテーションです。アノテーションは、クラスやメソッド、フィールドなどに補足情報をつけることができる構文です。

アノテーションの処理では、裏側で動的にクラスを生成したり、リフレクション処理を使うなど、言語モデルとは無関係な処理が行われています。ここで、オブジェクト指向とは関係ないモデルがJavaに持ち込まれるようになったわけです。

この時期の記述性の向上をEoDと言っていました。この中で、コードをPOJO、つまり継承を強制されないクラスで記述できるということがメリットとしてあげられていました。このときクラスは単に、記述をまとめ共通の設定を適用する単位となります。EoDとはフレームワーク構築でのオブジェクト指向モデルからの決別

でもありました。

ラムダ式

Java SE 8で、ラムダ式が導入されました。ラムダ式は基本的に匿名クラスの省略形式です。これを記述の点から見ると、関数を値として扱えるようになったように見えます。

ラムダ式はもともと関数型言語でよく使われていた構文をJavaに持ち込んだものです。関数型的な手法を使う場合には、オブジェクト指向的ではないプログラム設計が必要になります。

オブジェクト指向をどのように勉強するか

このような中で、それでもオブジェクト指向を勉強するというときに、なにを注意すればいいかまとめておきます。

オブジェクト指向はよいプログラムの指標ではない

まず知っておくべきなのは、オブジェクト指向はいいプログラムの指標ではないということです。オブジェクト指向になっているからよいプログラムだとか、オブジェクト指向になっていないからダメだとか、そういうプログラムのよさの指標ではありません。

オブジェクト指向の適用範囲を把握する

オブジェクト指向というのは、かなり適用範囲の狭い技術です。もしうまくオブジェクト指向ができないというときは、自分が作っているものがオブジェクト指向に向いていない可能性を考えてみるほうがいいと思います。

言語仕様を把握する

クラス、継承、オーバーライドといった、言語機能を1つずつ把握して全体のイメージを構築するよりは、オブジェクト指向というイメージがあると楽になるとは思います。しかし実用的にはオブジェクトをモジュールではなく型として、継承をモジュール拡張ではなく型の分類として、型の扱いを主体に考えることも重要です。

差分プログラミングの道具だと考える

ひとまとまりの処理のうち共通部分をメソッドとして抜き出す共通化はよく行われます。

差分プログラミングの場合は、逆に違う部分を抜き出します。異なる部分をメソッドとして抜き出しておいて、継承したクラスでそれぞれの処理としてオーバーライドします。このように継承を差分プログラミングの道具と考えます。

ただし、異なる部分を関数として渡すという関数型的な手法も考慮する必要があります。

OMTを勉強する

やはり粋なクラス設計を一度やってみたいというのは自然なことです。その場合は、ランボーの提唱していたOMT(Object Modeling Technique)を勉強するのがいいように思います。OMTは、オブジェクトの静的モデル、動的モデル、機能モデルを構築していくという方法論です。現在ではランボーによる解説書『オブジェクト指向方法論OMT』は絶版ですが、『憂鬱なプログラマによるオブジェクト指向開発講座[注1]』がOMTの流れをおおまかに解説しています。古い本であり記述に指摘すべき点もありますが。しかしながら、原典が絶版になり、新しい解説本も出ていないというのが結局のところ現実的な世間の評価ということでしょう。

脱オブジェクト指向

ここまでに見たように、開発手法、ツール、プログラミング言語などソフトウェアに関わるすべてがオブジェクト指向を乗り越え次へと進んでいます。局所的なオブジェクト技術は必要ですが、開発全体に適用するようなオブジェクト指向は、幻となりました。

夢の技術としてのオブジェクト指向なんかないよ、ということを踏まえながら勉強していきましょう。SD

注1) 『憂鬱なプログラマによるオブジェクト指向開発講座』Tucker(著)、翔泳社、1998年

第2部 オブジェクト指向の克服編

その7

Smalltalkこそオブジェクト指向の克服の手がかり

Writer 語り手：トム・エンゲルバーグ注1　聞き手：長谷川 裕一

インタビューウィズ エンゲルバーグ

――某月某日、湯之上温泉注2を取材していた私は、トム・エンゲルバーグ氏(下の書影の著作がある)がクラッシュパッド注3を担いで岩場に向かう途中に接触することができ、わずかな時間ながらも「新人エンジニアがオブジェクト指向をいかにして克服すべきか？」のインタビューを敢行できた。本稿はそのときの様子をでき得る限り忠実に再現したものである。

『間違いだらけのソフトウェア・アーキテクチャ』
Tom Engelberg(著)、長谷川裕一、土岐孝平(訳)
技術評論社、2010年

注1) 日本ツウのネイティブアメリカン(アパッチ族)。クライミングを趣味とする自称アーキテクト。オブジェクト指向やアジャイルについても詳しく、これまでにも雑誌などを通して、IT業界にさまざまな提言を行っている。

注2) 会津にある、サスペンスドラマの舞台(殺人は「塔のへつり」と呼ばれる景勝地であることが多い)に使われたこともある渋い温泉地。

注3) 比較的小さな岩場で、岩登り(ボルダリング)をするときに使う、落下しても大きな怪我をしないために敷くマット。なお、原理主義的なクライマーは原理原則を重んじ、クラッシュパッドを敷かない。IT業界にも、こうしたオブジェクト指向原理主義者が存在し、その多くはこれから学ぶSmalltalkを愛するSmalltakerだといわれている。

新人にJavaとかC#とかどうよ？

長谷川：以降――)日本の多くのSI企業では、新人研修で学ぶプログラミング言語としてJavaやC#を学んでいます。しかし、そこで多くの新人がオブジェクト指向というキーワードで詰まってしまうらしいのですが、それを打開する良いアイデアはありませんか？

トム・エンゲルバーグ：以降トム)うん、なんだって？　新人研修？　あぁ、あの会社に入って安心しきっている若造が予習も復習もしてこない、ヘナチョコな学校の授業みたいなやつだな。うん、そうだな……オブジェクト指向だとか言う前に、奴らには研修会場にゴミを散らかさないとか、とくに昼休み後のゴミ箱が酷いことになってるからな、帰りに集団で居酒屋に行って、酔って必要以上に大声で騒がないように教えた方が良いぞ、うん、そうだ。

――えー、もちろんそのとおりですが、それは置いておいて、新人エンジニアのオブジェクト指向の克服法についてお伺いしたいのですが。

トム：うん、そうか、そうだなぁ、まず、新人のエンジニアがオブジェクト指向を学ぶには、JavaやC#は中途半端な言語だと私は思うね。そう、きちんとオブジェクト指向を学ぶのであればSmalltalkを学ばせるのが良いだろうな。

――つまり、新人エンジニアにJavaやC#を学ばせるのは間違いだと言うことですか？

トム：いやいや、JavaやC#を学ぶのが間違いだとは言っていない。今後の仕事に必要なら、

Smalltalkこそ オブジェクト指向の 克服の手がかり その7

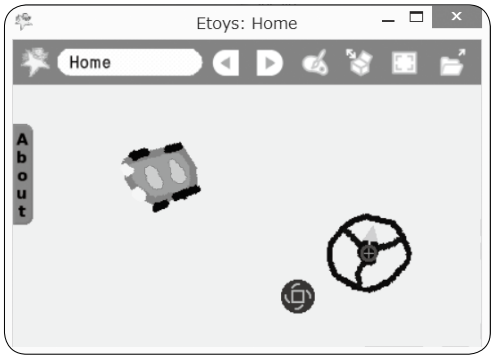
JavaやC#で、RDBMSからデータを読み込んで、それをHTMLにしてブラウザに表示する、そうしたWebアプリケーションを作るとか、今までどおりやれば良いと私は思うね、まぁ、それだけで良いとも決して思わないがね。ただ、オブジェクト指向を理解するっていうのが目的であれば、それにあった、プログラミング言語や課題を新人研修担当者は考えるべきだろうな[注4]。そこがあっていないのに、「オブジェクト指向がわからないのですか？」と新人エンジニアに尋ねるほうが酷というモノだろ。

青年よSmalltalkを学ぶのじゃ

――なるほど、オブジェクト指向がよくわかるような教育体系になっていないのに、わかるほうが無理だと。しかし、なぜオブジェクト指向を学ぶのにSmall「T」alkなのですか？

トム：ナニ！　Small「T」alkだと！　Tを大文字で言ったな。一体全体、Smalltalkのtを大文字にするとは穏やかじゃないな。まぁ、君が本気でSmallTalk[注5]について話を聞きたいというのなら別だがね。うん、まぁそれは良いとして、Smalltalkを勧めるのは私の趣味だな。それに、Smalltalkを学べばオブジェクト指向以外の知見も得ることができると思うのだよ。そう、たとえば、最近流行のアジャイル[注6]だ。アジャイルを最初に世に問うたのはケント・ベック[注7]やマーチン・ファウラー[注8]だが、彼らはいずれも元はSmalltalker[注9]だ。つまり、アジャイルをより理解したいのであれば、彼らの思考を育てたSmalltalkというものを理解したほうが良い。今では開発の定番とも言えるリファクタリングやJUnitなんていうものも同様だ。こうした話は「最近の若い奴は、C言語を知らないからポインタとかアドレスの概念がわかんないんだよ」ってことと私からすれば同じだよ。そうそう、クラウドの中核技術である仮想マシンの概念もSmalltalkでは学べるかな。実行環境しか提供しないJavaのヘナチョコ仮想マシンと違って、Smalltalkの仮想マシンは、開発環境も提供するちゃんとした仮想マシンだからね（そう言いながら、彼はクラッシュパッドの中からチョーク[注10]まみれのノートPCを開くと、次のような画面（**図1**）を見せてくれた）。

▼図1　スクイークEtoys



トム：これはスクイークEtoys[注11]というもので、Squeak[注12]というSmalltalk環境の上に作られている小学生でも使える教育用のオープンソースソフトウェア（フリーソフト）だよ。このクルマとハンドルは私が描いたんだが、こうしてハンドルでクルマを動かすことができる（実際に歪（いびつ）なハンドルをエンゲルバーグ氏がマウスを使って動かすと、クルマがハンドルをきった方向に曲がる）。まぁ、これは子供用のお遊びみたいなものだが、まずは、この辺から始めて、オブジェクトとは何か、メッセージや属性とは何かを自分なりに考えてみたらどうかな？……それに、スクイークEtoysを使えば、さっき言った仮想マシンなんかの話も実感できるだろう。

注4） JavaやC#で何年も仕事をしている先輩エンジニアが、実はオブジェクト指向を全然理解してないってことが、言語と課題があっていないことの証明である。

注5） Smalltalkのジョーク版。http://smalltalk.smalltalk-users.jp/

注6） アジャイルについては本誌（2012年7月号）の第2特集「もしも、新卒女子SEが『アジャイル』をマスターしたら」を参照。

注7） 陽気な外人。

注8） 陰気な外人。

注9） Smalltalkを愛する頭のイカレタ人達。

注10） クライマーが手につける白い粉の精神安定剤。滑り止めとも称される。

注11） http://etoys.jp/squeak/squeak.html

注12） http://squeakland.jp/

――JavaやC#の新人教育とは趣がずいぶんと違いますね。

トム：そうだね、新人研修が課題とするデータとその加工ばかりをしているエンタープライズな世界とはまったく違うね。まぁ、いったんそんなことは忘れたほうがいい。RDBMSとかSpringフレームワーク[注13]とかもな。Smalltalkを学ぶのであれば、そうした世界と決別して、オブジェクトとシミュレーションの世界にどっぷりと浸ることが重要だと思うよ。

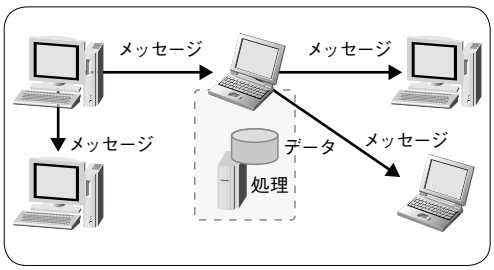
指向を増幅させる道具＝コンピュータ

トム：まぁ、そもそも、Smalltalkは1960年代という現在のインターネットや携帯端末の元ネタになるような技術が現れた時に、アラン・ケイという人物が、誰にでも使え、かつ、使う人の思考を増幅させるためのコンピュータとはどういうものであるべきか、また、今後予見される大掛かりなシステムの複雑さを軽減するために、小さな部品同士を組み合わせて大きな仕事をさせるためにはどうすればよいかを考えた末に、産み出したという側面を持っているんだ。

まず、最初の側面はスクイークEtoysで見たとおりだ。思考の増幅には、現実では実現することが困難なシミュレーションによる試行錯誤が欠かせないだろ、つまりモノ（オブジェクト）を描いて実際にシミュレーションができるということだね。

次の側面だが、大掛かりなシステムは1台のコンピュータではどうにもならん、かといってでたらめに複数のコンピュータをつなげてもいかん。そうすると、ある1台のコンピュータに役割をもたせて、夫々（それぞれ）に役割に合致したデータとそれを使って計算する処理をもたせるはずだ。そして、そうしたコンピュータ同士が通信をしながら何かの仕事をやり遂げることになるわけだ。それこそ、オブジェクト指向で最初に学ぶ、

▼図2　つながるオブジェクト



オブジェクトはデータ（属性）と処理（メソッド）をもっていて、互いに通信（メッセージ）をやり取りしながら動作するということなのだよ（**図2**）。

もっと身近なところで簡単に言えば、TVやDVDレコーダとスピーカをつなげるようなものだと考えてもいいな。近い将来、家電同士がコンピュータを内蔵して、相互につながるなんていう話も同じ類の話だ。もちろん、この話はさっき見せたスクイークEtoysのクルマとハンドルとか、そう、生物の細胞なんかにも言えることだね。

さぁて、本当はもう少し、Smalltalkには、Javaでいうところの＋とか－とかの演算子や、ifとかwhileの制御構文がないとか、文法はA4用紙1枚で納まるとか、いろいろと話をしたいところだが、私はこのあと帰国前にどうしても登らなければいけないボルダーの課題があるからな、あとは『自由自在Squeakプログラミング[注14]』を読んで各自勉強するように伝えてくれるかな（と岩があるとおぼしき河原の方へ歩き出した）。

――エンゲルバーグさん、すいません、最後に！最後に一言だけ、読者である新人エンジニアの方に！

トム：オブジェクト指向の頂きは1つではなく、その頂きに到達する道も1つではないぞ！　そして、人生万事、道化芝居さ！（センバー・ファルシシムス！）。SD

注13） Javaの定番フレームワーク。プロジェクトでの利用事例は多いが、プロジェクト関係者でSpringフレームワークがどのように利用されているかを正確に知っている者は少ない。

注14） 梅沢真史さんの好意により、SRCから出版された『自由自在Squeakプログラミング』のPDF版とサンプルコードがダウンロードできます（http://swikis.ddo.jp/squeak）。ぜひ、お読みください。

第2特集

クラスタリングの教科書

止まらないサービスを支えるシステム構築の基礎

クラスタ技術には大きく2つの側面があります。1つは分散処理による高速化、もう1つは冗長構成による高可用性への適用です。本特集は「高可用性」にフォーカスします。

いまやクラウドサービスによって、ハードウェアの故障を気にすることなくサーバを運用できるようになりました。しかし、使っているインフラがどういうしくみで動いているのかを知ることは重要です。利用するデータセンターやクラウドサービスの高可用性をしっかりと量ることができるからです。もちろんシステム構築に携わるならば必須の知識です。

本特集では、以前からあるクラスタリングの基礎はもちろん、クラウドサービスを使ったシステム構築の際にも役立つ高可用性クラスタリングのテクニックなどを解説します。

第1章 避けられないシステム故障への備え

クラスタシステムのしくみ

本章では「高可用性」に焦点を当てたクラスタリングについて、代表的な構成例をあげながらそのしくみと有用性、欠点などを解説します。また、障害発生時のノード間の連携動作を知り、クラスタシステムの設計方針について概要をつかんでおきましょう。

●㈱IDCフロンティア　田村 晋(たむら しん)

止まらないサービスを目指して

24時間365日のオンデマンドサービスが当たり前となった今、それを支えるインフラに対してはダウンタイムを最小化することがこれまで以上に求められています。ダウンタイムを少なくする、つまり壊れにくい／復旧しやすいシステムを作ることを「可用性を高める」と言います。

インフラはハードウェアとソフトウェアの2要素によって構成されています。これらはいわば車の両輪のようなものです。ハードウェアは、ソフトウェアを止めずに実行し続けること、ソフトウェアは止まらずに実行し続けることが求められています。しっかりしたソフトウェアを作っていても、それが実行されるハードウェアの品質が悪かったり、性能が不足していると、期待したパフォーマンスは得られません。

インフラの可用性を高めるためには、それを可能にするソフトウェアとハードウェア両面の実装が不可欠です。そのための技術的アプローチの1つとして、本特集の主題であるクラスタリングがあります。1章ではクラスタリング、とくに高可用性(High Availability、略してHA)のためのしくみや使われ方ついて解説していきます。

▼表1　ハードウェアに関する主な障害

単純故障	ディスクやメモリ、CPUなどの故障
ファームウェアのバグ	UEFI、BIOSの誤作動など
環境要因	温度上昇、電圧降下、停電など

▼表2　ソフトウェアに関する主な障害

リソース枯渇	メモリ領域やクロックの不足、設定の不備など
アプリケーションのバグ	メモリリーク、デッドロック、予期しない動作など
OSのバグ	208.5日問題、497日問題など
デバイスドライバのバグ	デバイスのハングアップなど

ダウンタイムにつながる障害

計画的なメンテナンス以外でのダウンタイムは、予期せぬインフラの障害によって引き起こされます。**表1、2**に主な障害を挙げます。日々の監視や構成管理、情報収集などで事前に対処できる要素もある一方、ハードウェアの単純故障は防ぐことが難しく、発生時には物理的な交換作業が必要です。

防ぎきれない障害に対しては、事前にそれを想定したうえで発生時の対処法を決めておく必要があります。さらに、それを自動化することができれば、ダウンタイムを最小に抑えることができます。それを可能にするのがクラスタシステムです。

クラスタシステムの構成パターン

クラスタシステムはいくつかの構成パターンがあり、目的に応じて選択することができます。共通しているのは、複数のコンピュータを組み合わせていること、組み合わせたコンピュータ同士でデータ共有していることです。この章の説明では組み合わせの構成を理解しやすくするため、演算処理を実行するシステム部分を「ノー

ド」とし、データを共有する部分を「ストレージ」として分けて説明していきます。

いくつかの構成パターンについて、よく知られている製品やソフトウェアもあわせてご紹介します。興味があれば、実際に触ってみて理解を深めていただければと思います。

ノードの構成パターン

本章ではノードの構成パターンとして、比較的よく使われているフェイルオーバークラスタとロードバランシングクラスタの2種類について説明します。

可用性を高めるフェイルオーバークラスタ

フェイルオーバークラスタは稼動系と待機系の2つのシステムを用意しておき、稼動系で障害が発生した際に待機系に切り替えて稼働を継続させる構成です。通常時に使われるシステムは1系統だけのため、構成がシンプルになります。ただし、同じシステムを2つ用意する必要があるので、ハードウェアやソフトウェアライセンスなどの初期導入コストは倍かかることになります。

フェイルオーバークラスタの待機系の使われ方には、ホットスタンバイとコールドスタンバイの2種類があります。

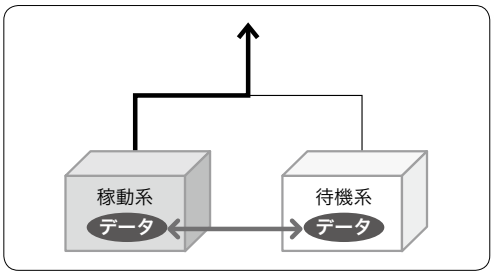
ホットスタンバイは**図1**のように、稼動系と待機系でセッション情報等の内部状態を同期しておき、障害発生時には待機系が稼動系の処理をそのまま引き継ぐ動作をします(**図2**)。こうすることで、障害発生時のダウンタイムを最小化することができるのです。一般に、このときの引き継ぎの動作のことをフェイルオーバー(Failover)と呼びます。

一方、コールドスタンバイは**図3**のように状態の同期は行わず、単純に切り替えるのみです。障害が発生した稼動系を停止し、待機系を稼動系として起動させて使用します。

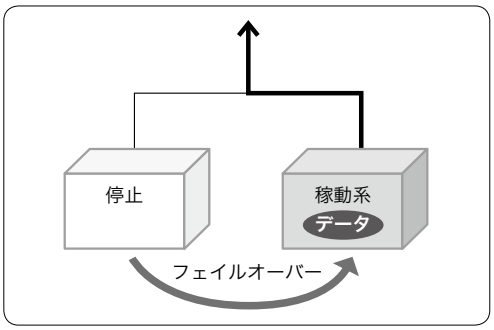
切り替えに伴うデータやセッションのロストが許容されなかったり、よりダウンタイムを短くする必要がある場合には、ホットスタンバイ方式をとります。ただし、データの同期と、フェイルオーバーのためのしくみをソフトウェア側でも作り込む必要があります。コールドスタンバイ方式の場合は、単純に切り替えて起動させるだけのため、ソフトウェアの作り込みができない場合に有効です。

フェイルオーバークラスタを構成するパッケージに、keepalived[注1]やHeartbeat[注2]があります。

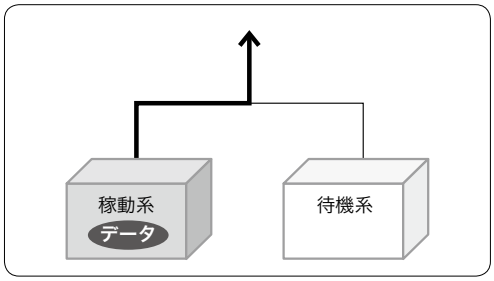
▼図1　ホットスタンバイのフェイルオーバークラスタ(通常時)



▼図2　ホットスタンバイのフェイルオーバークラスタ(障害発生時)



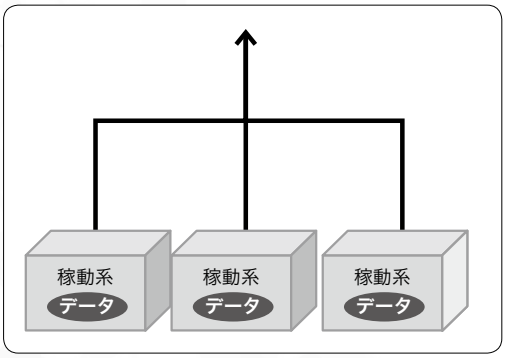
▼図3　コールドスタンバイのフェイルオーバークラスタ(通常時)



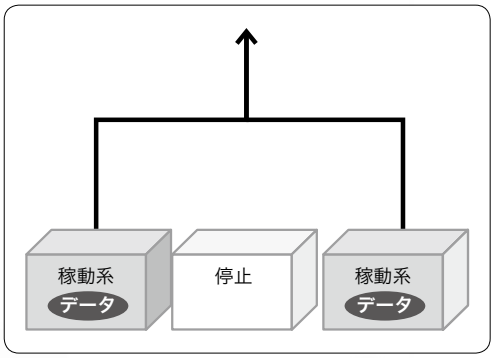
注1) http://www.keepalived.org/

注2) http://linux-ha.org/wiki/Heartbeat

▼図4　ロードバランシングクラスタ（通常時）



▼図5　ロードバランシングクラスタ（障害発生時）



スループットを高めるロードバランシングクラスタ

ロードバランシングクラスタは**図4**のように、システムを複数用意しておき、すべてを稼動系として動作させる構成パターンです。予備となる待機系は用意せずに、障害が発生したシステムは稼動系から外してクラスタシステムとしての動作を継続させます（**図5**）。

複数のシステムを並列で使用するため、フェイルオーバークラスタより効率的にリソースを使用することができます。複数のシステムで処理を分担させることで、同時に作動するシステムが単一となる構成よりも、クラスタ全体でのスループットを高めることができます。

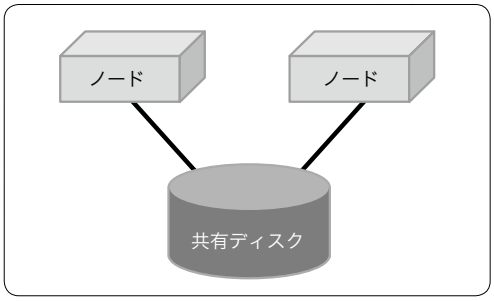
近年の大規模Webサイトのインフラはこの構成が主流です。フェイルオーバークラスタと比べて構成は複雑になりますが、リソースの需要に応じて柔軟にシステムをスケールさせることができるのが最大のメリットです。

ロードバランシングクラスタを構成するパッケージとしてはLVS（Linux Virtual Server）[注3]やHAProxy[注4]が有名です。

ストレージの構成パターン

クラスタシステムとして動作するためには、すべてのノードが同じデータにアクセスできるようにする必要があります。ストレージは、クラスタシステムの中でデータを保存し、共有する役割を担います。

▼図6　共有ディスク方式



データを共有する方法はいくつかあります。大きく分けると、1ヵ所に書き込む方法と、複数個所に分散して書き込む方法に大別されます。

共有ディスク方式

1台のストレージをクラスタ内の複数のノードが参照する構成です（**図6**）。シンプルな構成なので運用しやすい一方、ストレージが単一障害点となってしまうため、注意が必要です。

この構成をとる場合は、ハードウェアレベルでの冗長設計がされている専用のストレージ装置を使います。このストレージ装置は、構成している部品それぞれが冗長化されており、どこか1ヵ所が故障しても停止することなく動作するよう設計されています。

普通のPCサーバにソフトウェアパッケージをインストールして構築する場合は、NFSやtgtd（iSCSI）、Samba（CIFS）を使って構成することができます。

注3）http://www.linuxvirtualserver.org/

注4）http://www.haproxy.org/

ミラーリング方式

共有ディスクは使わずに、ノード同士でデータをリアルタイム同期させて共有する構成です(**図7**)。同じデータを複数個所に書き込む共有方法にあたります。あるノードで障害が発生しても、別のノードに同じデータが保存されているため、データへのアクセスが失われることはありません。

複数個所にデータを書き込むため、1ヵ所に書き込むよりもオーバーヘッドが大きくなります。ただし、データの読み込みはどこか1ヵ所を参照するだけでよいので、読み込み負荷は分散させることができます。

ストレージのミラーリングは、DRBD(Distributed Replicated Block Device)[注5]を使って構成することができます。

レプリケーション方式

ストレージ装置同士やノード同士で、一定間隔でデータの同期をとる構成です(**図8**)。リアルタイムではなく、一定間隔での差分コピーをとるようにすることで、書き込みのオーバーヘッドを抑えることができます。

レプリケーション構成は、データの書き込みを受け付けるマスタと、データの同期先であるレプリカの区別があります。レプリカは、マスタ側のコピーとして整合性を保つために読み込み専用として振る舞います。

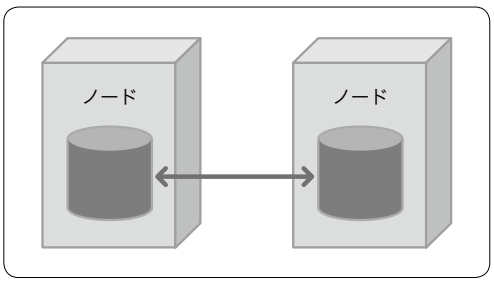
マスタ側で障害が発生した場合には、レプリカをマスタとして切り替えることで、データの読み書きができる状態に復旧させます。ただし、リアルタイム同期ではないため、レプリカ側に反映される前のマスタの書き込みデータは失われることになります。

ハードウェア制御のレプリケーションでは、書き込み途中のアプリケーションのデータ整合性を完全には維持することができないため、使い方によってはバックアップとならない場合があることに注意しなければなりません。

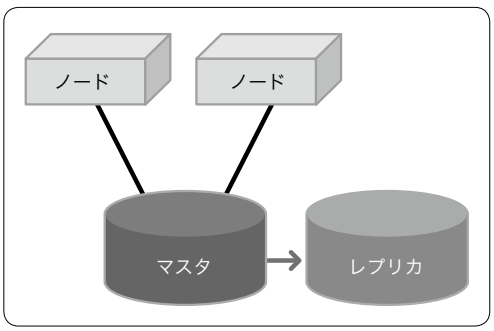
分散ストレージ方式

複数のストレージ装置やノードにデータを分散して保存する構成です(**図9**)。定められた単位でデータを分割して同時に複数個所に書き込みます。単純なミラーリングよりもスループットを高めることができるうえ、規模をスケールさせることが可能になります。

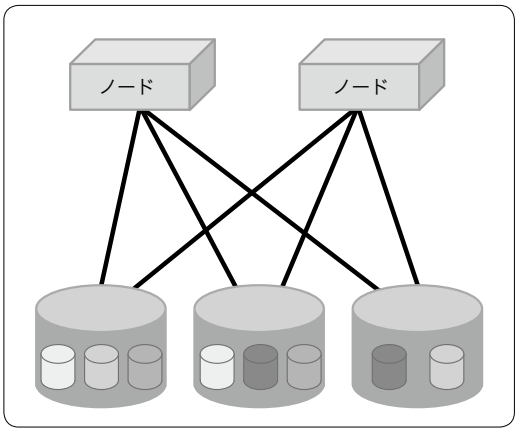
▼図7　ミラーリング方式



▼図8　レプリケーション方式



▼図9　分散ストレージ方式



注5）http://www.drbd.org/ja/home/what-is-drbd/

データは一定のブロックサイズやオブジェクト単位に分割して書き込まれます。分割することにより、複数のノードに分散して高速に書き込むことができる一方、そのうちの一欠片（ひとかけら）でも破損するとデータは読み出すことができなくなります。

このため、分散ストレージは、ハードウェア故障などで欠片の1つが破損しても、ほかから同じデータを読み出せるように、2ないし3ヵ所程度に同じデータを複製しておき、データの消失を防いでいます。欠片の1つが失われた場合には、残りの欠片を再度コピーすることで、冗長を回復させることができます。

分散ストレージは近年注目度があがってきています。構成するためのソフトウェアとしてはRiak[注6]やCeph[注7]が有名です。

クラスタシステムの成り立ち

しくみをざっと理解したところで、クラスタシステムがなぜ必要とされるようになったかを歴史的経緯から見てみましょう。

現在では一般的となったクラスタシステムですが、その起源はかなり古く、メインフレームが主流だった時代までさかのぼります。

コンピュータが民間企業に導入され始めた1980年代、メインフレームと呼ばれる大型のホストコンピュータを中心としたアーキテクチャがとられていました。

当時のコンピュータは大型で、接続に使う端末の性能も限られていたため、高性能なコンピュータを1台導入し、そこに各ユーザが専用の端末で接続して利用する形態をとっていました。

1台のコンピュータが中核となるメインフレームは、停止してしまうとすべての処理ができなくなってしまいます。この弱点を克服するため、電源や記憶装置、CPUの二重化によりシステム単体としての信頼性が高められ、さらに共有ディスクを用いた冗長化のしくみが導入されました。これがフェイルオーバークラスタの始まりとなりました。

90年代に入ると、小型化された安価なサーバが出現しました。ホストコンピュータの代わりに、サーバが処理の中核を担い、クライアントと呼ばれるコンピュータでそれにアクセスするアーキテクチャに変わりました。これがクライアントサーバ型と呼ばれるモデルで、現代のインフラはこの形態が一般的になっています。

サーバはメインフレームのような大型のコンピュータに比べ、個々の性能と信頼性は劣ります。それを克服するために、複数のサーバを連携させてシステム全体で性能と信頼性を高めるロードバランシングクラスタが生み出されました。

スケールアップからスケールアウトへ

システムの性能が不足したとき、性能向上のためには2つのアプローチがあります。そのひとつがスケールアップと呼ばれるもので、これはサーバのメモリやCPUを増設して1台あたりの性能を高めるアプローチです。クラスタ化されていないシステムはスケールアップすることで性能を増強します。

もうひとつのアプローチとして、同性能のサーバを増設していき、システム全体でみた性能を高めるスケールアウトがあります。とくに、サーバを複数台組み合わせるロードバランシングクラスタの場合、スケールアウトのほうが有効です。サーバ1台あたりに搭載できるCPUやメモリ容量には上限があるためです。

近年登場したクラウドコンピューティングによって、サーバやクラスタシステムの在り方も変わってきました。従来は1台1台のサーバが物理的なコンピュータであったため、スペックアップや増設に手間がかかっていましたが、クラウドコンピューティングの一種であるIaaS（Infrastructure as a service）がこれらの物理的な作業を不要にし、簡単にサーバを増設することを可能にしました。

注6）http://basho.co.jp/riak/
注7）http://ceph.com/

IaaSは仮想化技術を基盤として実現されているサービスです。1台の物理的なサーバにハイパーバイザをインストールし、そこに仮想マシンを作成してサーバとして提供します。ソフトウェア的にサーバを作成、構築できるので、物理的な作業の手間を省き、迅速な展開ができるようになりました。

IaaSによって、負荷に応じてオンデマンドでノードを自動増設し、システム全体の処理性能を高めるような、ロードバランシングクラスタのスケールアウトがより簡単にできるようになったのです。

クラスタシステムの動作

冒頭でクラスタシステムの構成パターンをいくつかご紹介しました。ここではクラスタシステムとしての動作をより理解していただくために、ノード同士の連携のしくみについて説明します。

フェイルオーバークラスタ

マスタ／バックアップ

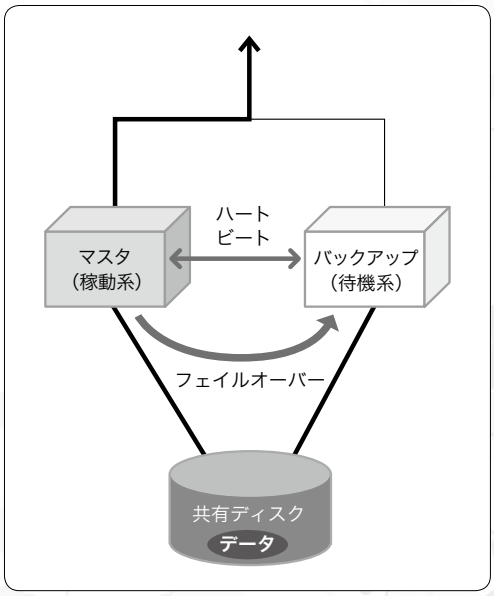
フェイルオーバークラスタ(**図10**)は、稼動系として動作するマスタと、待機系となるバックアップの2つのノードで構成されます。マスタが処理を実行している間、バックアップはマスタを監視して待機します。障害によりマスタが動作を停止した場合、バックアップはそれを検知して自身がマスタに昇格して処理を実行するようになります。

ハートビート

各ノードは、ハートビート(Heartbeat)と呼ばれる信号を発信することによって自身の生存を知らせます。ハートビートは一定間隔で発信される生存を示す信号なので、鼓動を意味するHeartBeatと呼ばれています。

これが途切れると、クラスタシステムはそのノードが機能を停止したとみなし、クラスタのメンバから外す動作をします。

▼図10　フェイルオーバークラスタ



フェイルオーバー

ノードで障害が発生した際には、クラスタとして正常な動作を継続させるためにフェイルオーバーが行われます。フェイルオーバーが発動すると、障害が発生しているノードは処理を停止し、データや機能、役割を別のノードが引き継いで処理を再開します。

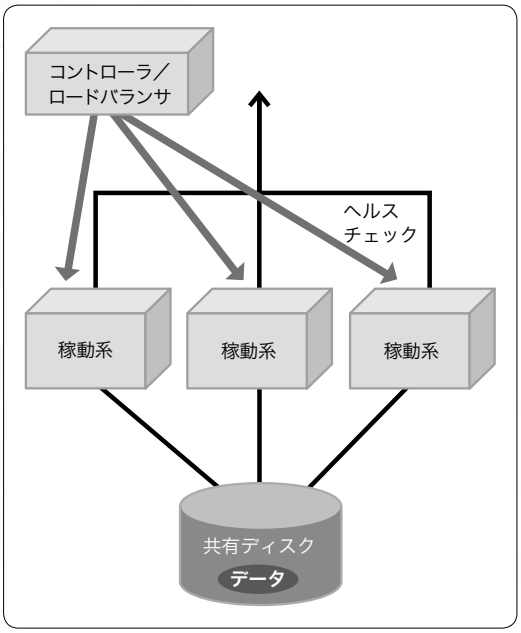
フェイルオーバークラスタでは、バックアップノードはマスタのハートビートを監視し、それが途切れたタイミングで障害が発生したものとみなしてフェイルオーバーのアクションを開始します。

スプリットブレイン(予期しない動作)

何らかの理由でハートビートの通信だけが途切れ、マスタが処理を続けた場合には、スプリットブレインという現象が起こります。

ハートビートが途切れると、バックアップノードは、マスタに昇格して処理を引き継ごうとします。このとき、もともとのマスタがまだ処理を継続していると、マスタが2つに増えてしま

▼図11　ロードバランシングクラスタ



い、データの不整合やリソースの奪い合いが生じます。結果としてクラスタとしての動作を継続することができなくなります。

ハートビートだけが途切れる現象は、ネットワークの障害やプログラムの動作不具合、負荷による一時的な応答遅延などにより起こります。あまり珍しいことではありません。このため、ハートビートは複数の伝達手段を組み合わせて実装するほうが望ましいといえます。

ロードバランシングクラスタ

コントローラ／ロードバランサ

ロードバランシングクラスタ(**図11**)は、演算処理を担当するノードと、クラスタ全体を管理するコントローラで構成されます。ノードはすべてが稼動系として処理を実行します。コントローラは各ノードにデータを渡し、演算処理の指示をする監督者の役割です。また、各ノードの状態監視も行います。

ノードに異常が発生した状態ではそのノードの演算結果は信頼できるものではなくなります。故障によって完全に停止した場合、データへのアクセスが失われる可能性があります。コントローラはこうしたノードを検知し、クラスタのメンバから外すことで、クラスタシステムとしての正常性を維持します。

ヘルスチェック

先述したハートビートでは、各ノードの生存を確認できますが内部の動作まで含めて正常に機能しているかまではわかりません。ノードとして生きていても、実際には演算処理を実行できない内部状態になっていた場合、クラスタシステムは正しい演算結果を出すことができなくなります。

ヘルスチェックと呼ばれるしくみは、ノードに対して決められた操作や入力を定期的に行い、出力結果を評価して、正常に動作しているかをチェックします。こうすることで単純な生存確認ではなく、機能の正常性を確認します。

リバランス

ロードバランシングクラスタでは、参加するすべてのノードが処理を実行します。障害によってあるノードが停止したときには、その分の処理は別のノードに再度分散されます。

クラスタシステムの設計

CAP定理

クラスタシステムの構成を考えるうえでよく出てくる用語に「CAP定理」と呼ばれるものがあります。CAPとは次の3つの事項の頭文字に由来します。

- 一貫性(Consistency)……すべてのノードが同じデータを参照できること
- 可用性(Availability)……一部のノードが障害で停止しても機能すること
- 分断耐性(Parition tolerance)……ノード同士連携できなくなっても機能すること

CAP定理は、これら3つの性質を同時にクラスタに実装することができないことを示しています。

たとえば、ミラーリングは一貫性と可用性を実現できますが、ノード間のデータ連携ができないと機能しないため、分断耐性はありません。単一の装置で構成される共有ディスク方式の場合は、分断されないという意味で分断耐性がありますが、1ノード構成になるため可用性が満たせません。

例を挙げればきりがありませんが、どのような技術で実装されたクラスタであっても、面白いほどこの定理が当てはまります。これからクラスタシステムの開発や運用にかかわっていく方は、そのシステムがどの性質を満たしているか考えてみると、理解を深めることができるものと思います。

単一障害点

クラスタシステムは、複数のノードで構成するものであるため、一見、単一障害点はないように見受けられますが、システムを構成する要素にはさまざまな単一障害点が潜んでいます。主な個所と対処方法をいくつか示します。

- 電源系統：物理的に2系統以上に分ける必要があります。電源ユニットを冗長搭載できない場合は、ノードごとに接続する電源系統を分けるなどして、ブレーカトリップなどの電源障害で全ノードが止まらないようにします
- ネットワーク：2経路以上の物理的な冗長化が必要です。レイヤ2では、輻輳（ふくそう）による通信断に備えてブロードキャストドメインを意識する必要があります
- コントローラ／ロードバランサ：クラスタシステムを制御する重要な役割であるため、コントローラも冗長化が必要です
- 共有ストレージ：すべてのノードがデータを保存しているため、停止した場合の影響が最も大きい個所です。ミラーリングによる冗長化や、レプリケーションによるデータ保護が必要です

▼表3　データの保存のされ方と所在

永続的な領域	ハードディスク／SSD、ノードから独立したストレージ
揮発性の領域	メインメモリ、キャッシュメモリ、ネットワーク

データ保護が最重要

フェイルオーバーやバランシングを前提とするクラスタシステムの場合、データの所在に気を付けて全体を設計する必要があります。

システム上に存在するデータは、永続的なものと揮発性のものに大別できます。これらは、そのデータが存在する場所によって決まります（**表3**）。永続的な領域は通電が停止されても保持される記憶領域です。一方、揮発性の領域は、通電が停止されるとデータを保持することができず消失させてしまいます。フェイルオーバーが発生した際にデータが失われる領域です。

クラスタシステムはフェイルオーバーによってシステムとしての動作を継続しますが、その際に失われるデータがあることを意識しなければなりません。失われては困るデータは、永続的な領域にきちんと保存する実装にする必要があります。

クラスタシステムに限らず、あらゆる情報システムが死守すべきはデータです。システムが止まってしまうような事態に見舞われても、データさえ残っていれば復旧させることが可能ですが、データが失われてしまうと復旧の可能性はゼロになります。

今日のビジネスの根幹を支えているのは情報システムのインフラです。システムの停止やデータロストは、企業の業務を止めてしまったり、最悪の場合存続不可能にしてしまいます。クラスタリングをうまく活用し、強固なシステムを作ることができれば、そのビジネスの成功はより確実なものとなるでしょう。SD

第2章 クラウドはいかにして守られているか

データセンターにおけるクラスタリングの実際

本章ではデータセンターで行われている多面的なクラスタリングの例から、クラウドの背後で行われている高可用性の実現がどのようなものであるかが垣間見えます。また、クラウドサービスを使う場合にユーザ側で行えるクラスタリング手法も紹介します。

●さくらインターネット㈱　大久保 修一（おおくぼ しゅういち）

高可用性を実現するクラスタリング

改めて、クラスタリングを行う目的は何でしょうか？

電気で動いている機器はいつか必ず故障します。それは明日かもしれませんし、5年後かもしれません。いつ壊れるかわかりませんが、そのままではサービス停止が発生します。障害からシステムを守り、サービスを継続するためにクラスタリングを行います。最終的には顧客満足度を高めたり、収益を最大化したり、機会損失を回避したりといったビジネスの成功を支えるためでもありますが、直接の目的の1つは落ちないシステムを構築すること（＝高可用性を実現すること）と言えます[注1]。

本章では、データセンターをケーススタディとして、高可用性を実現するために、実際どのような対策を取っているかを説明します。

高可用性実現の前提となるファシリティ

データセンターにおいて、高可用性を実現するうえで最も重要なポイントはファシリティです。データセンターは、さまざまなシステムを稼働させるための物理的な場所、環境を提供しているという点から、ファシリティは第一に考えなければなりません。具体的には次のような検討が必要です。

- 地震や津波などの災害発生の可能性が低い立地条件、建造物の堅牢性
- 周辺道路などのインフラが整備されており、保守や障害などの緊急の際に現地に駆けつけやすいこと
- 安定した電源供給を受けられること（複数変電所から受電しているなど）
- 商用電源がストップしてもバックアップできる無停電電源装置（UPS）、発電装置を備えていること
- 長期間商用電源がストップした際にも発電装置の燃料供給を継続して受けられる体制が整っていること
- 空調設備が十分なキャパシティを備えていること
- 安定したネットワーク接続が実現されていること（光ファイバが複数経路で引き込まれているなど）

ほか、挙げればきりがありませんが、構築するサーバやネットワークシステムをいくらクラスタリングし、高可用性を実現したとしても、そもそもの電源供給がストップしてしまったり、データセンターが罹災（りさい）して使えなくなったりしては元も子もありません。

インターネットなどの外部ネットワークへの接続性もミッションクリティカルなファシリティの一部です。システムが稼働していてもネットワークが切れてしまってはサービスが停止して

注1）実際には、どんな障害にも耐えられるシステムは存在しません。高可用設計はあらかじめ想定された救済すべき障害について対策を行うものだからです。そういう意味では「落ちにくいシステム」が適切な表現かもしれません。

しまいます。

物理的な光ファイバの引き込みルートやバックボーン機器などを2重化しているかどうか、障害発生時に24時間常に対応可能な体制を整えているかどうかも高可用性を実現するうえで非常に重要なポイントです。

データセンターで動く仮想化基盤のクラスタリング

ここからは本題に入り、仮想化基盤の最たる例であるIaaS(Infrastructure as a Service)クラウドのシステムを取り上げ、高可用性を実現するための構成や対策など、実例を交えて紹介します。筆者が勤めるさくらインターネット㈱では、「さくらのクラウド」というIaaSを提供しています。システム構成は**図1**のようになっており、大きく4つのパートに分けることができます。

① ホストサーバ
② ネットワーク
③ ストレージ、ストレージネットワーク(SAN)
④ クラウドコントローラ、認証基盤、課金システム

なお④については、一般的な業務アプリケーションの構成に近いため本稿では割愛します。仮想化基盤独特である①～③についてこれから詳細を見て行きましょう。

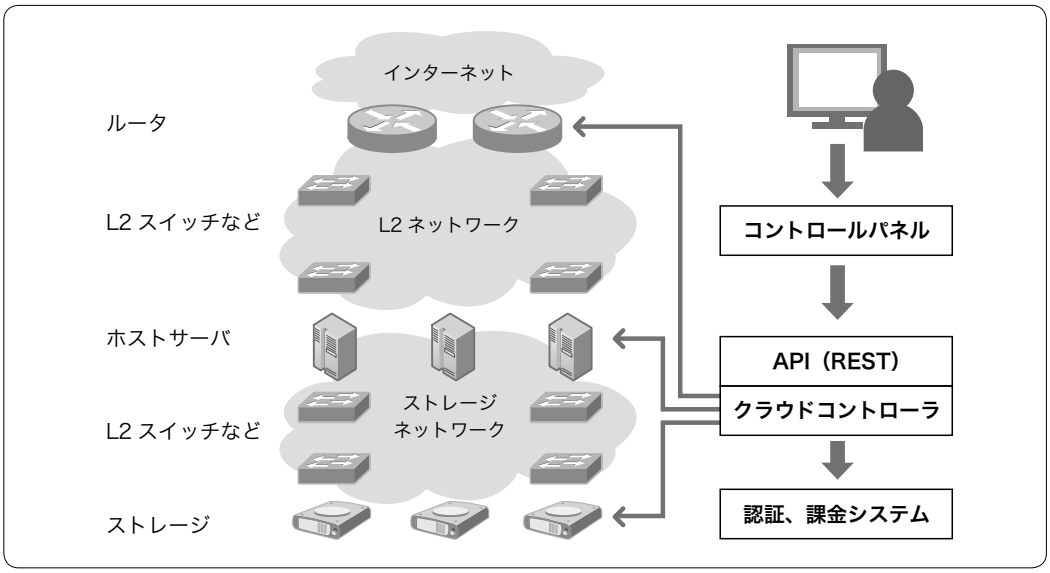
① ホストサーバ

ホストサーバはクラウドのコアになる部分で、ユーザの仮想サーバを収容するハイパーバイザです。多数の仮想サーバを起動するため、ハードウェアスペック(CPUのコア数やメモリ)は非常に高いものです。たとえばさくらのクラウドで導入しているものは、1台あたり12コア(24スレッド)、192GBのメモリを搭載しています。

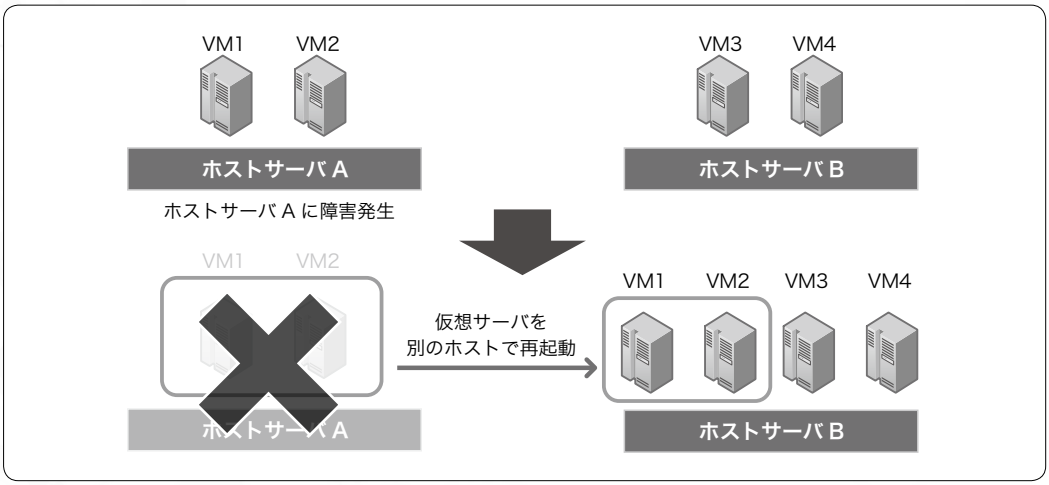
ホストサーバは、電源やNIC、OS起動用ディスクの冗長化を行うことで可能な限り単体での可用性を高める対策を行っています。一方、CPUやメモリ、マザーボードといった交換不可能なパーツが故障することもあります。そのような兆候が発見された際には、仮想サーバの動作を継続したまま別のホストに移行する「ライブマイグレーション」という技術を用いて、ハードウェアメンテナンスを行います。

しかし、突然故障して停止したり、ハイパーバイザがクラッシュしたり、ライブマイグレーションができない状況もあります。その場合、残念ながらホスト上で動作していた仮想サーバ

▼図1　さくらのクラウドのシステム構成



▼図2 ホストサーバ障害時のフェイルオーバー



はいったんダウンしてしまいます。その後、図2のように正常稼働している別のホストで仮想サーバを再起動させ、復旧します。仮想サーバを動作させるためのデータはストレージ側に格納されており、どのホストでも起動できるようになっているのです。

このように、ホストサーバ単体ではカバーできない故障などのケースをシステム全体でカバーし、高可用性を実現しています。このような機構は、クラウドサービスでは一般的に「HA機能」あるいは「自動フェイルオーバー機能」と称されています。仮想サーバがまったくダウンしないわけではないので注意してください。

なお、ホストが突然ダウンしても仮想サーバの動作を継続できるFT(Fault Tolerance)と呼ばれるしくみもあります。これは、2台のホスト上でプライマリ、セカンダリの仮想サーバが動作し、CPUやメモリの状態をリアルタイムにレプリケーションします。プライマリがダウンした場合はセカンダリがプライマリに昇格し、途中のCPU、メモリの状態から動作を継続します。ただし、FTを実現するには構成の制限やコスト上の課題があり、パブリックIaaSクラウドで実装されているケースは少ないようです。

② ネットワーク

クラウドのネットワークの役割は、仮想サーバをインターネットやVPNといった外部のネットワークへ接続し、また仮想サーバ間の通信を実現することです。ネットワークが停止すると仮想サーバが動作していても実質サービスできない状態になります。当然ながら安定した動作、高可用であること、さらにメンテナンスの際にも停止が発生しないことが要求されます。

さくらのクラウドのネットワーク構成を図3に示します。おもに3つのパーツに分かれています。

(a) ルータ……インターネットなどクラウド外のネットワークと接続
(b) Top of Rackスイッチ……ホストサーバやストレージの回線を収容するL2スイッチ
(c) コアスイッチ……ルータやTop of RackスイッチをアグリゲーションするL2スイッチ

(a)ルータのクラスタリング

ルータの高可用性実現にはVRRPが一般的に使用されます[注2]。VRRPは、Virtual Router Redundancy Protocolの略で、複数のルータ間で共通の仮想IPアドレスを保持します。外か

注2) さくらのクラウドでは、VRRPに似たHSRP(Hot Standby Routing Protocol)を使っていますが、これだとどこのメーカーのルータかわかってしまいますね :-)

ら見ると、それらが仮想的な1台のルータとして扱えるわけです。

通常時、VRRPを設定したルータ内ではマスタがその仮想IPアドレスを保持しています。マスタが故障すると、自動的に他のバックアップルータが仮想IPアドレスを引き継ぎます。他のノードからは仮想IPアドレスを参照している限り、ルータの切り替わりを気にする必要はありません。

ところで、VRRPはその名のとおり、もともとルータの冗長化を目的として策定されたプロトコルなのですが、サーバやネットワークアプライアンスの冗長化にも広く活用されています。VRRPの動作や設定は非常に簡単で、VRRPさえ知っておけばクラスタリングが必要なシチュエーションの多くをカバーできます。この章の後半で、keepalivedを用いたLinuxのVRRP設定について解説していますので、ぜひ試してみてください。

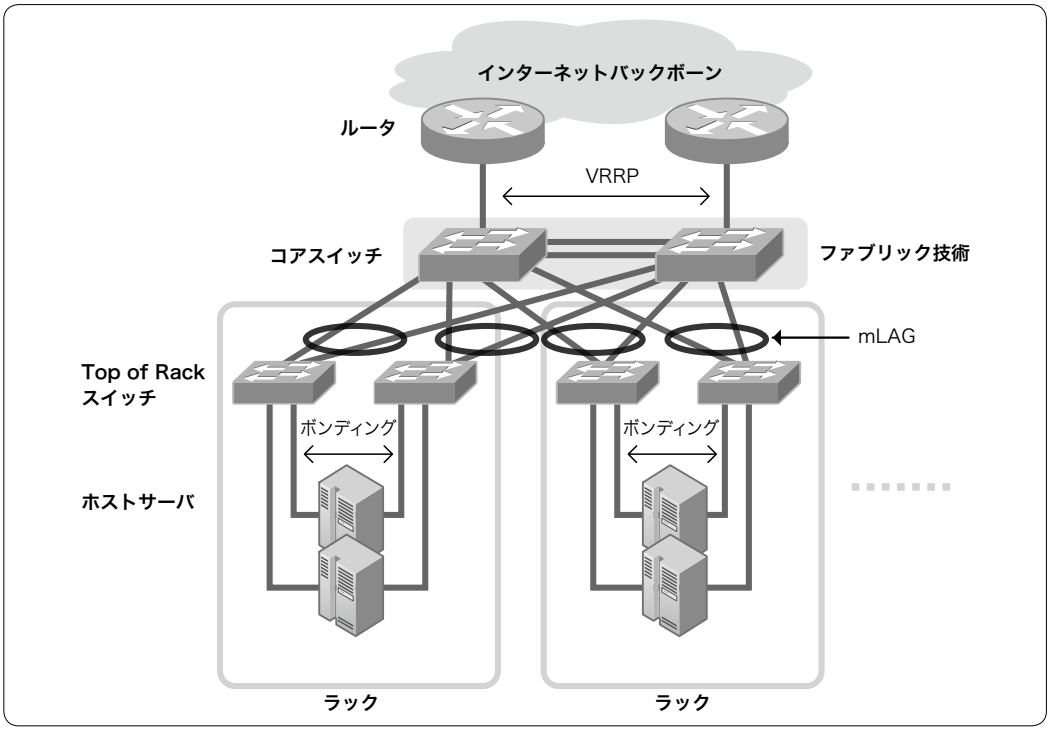
なお、VRRPには仮想IPアドレスを切り替える以上の機能はありません。そのためステートレスなノードのクラスタリングに向いています。ステートフルなノード(サーバやデータベースなど、データが載っているノード)のクラスタリングにはそれらを同期するしくみが別途必要ですので、ご留意ください。

(b)Top of Rackスイッチのクラスタリング

Top of Rackスイッチ(以下、ToR)の高可用性実現には、ボンディング(Bonding)と呼ばれる技術が用いられることが多いです。コアスイッチに接続したToRを1ラックあたり2台ずつ設置し、ホストサーバやストレージは両方のスイッチに接続しておきます。通常、プライマリ側リンクのToRにトラフィックを流します。プライマリ側ToRが故障した際には、サーバやストレージ側でリンクダウンを検出し、バックアップ側リンクのToRにトラフィックを切り替えます。

このように、BondingはToR側ではなく接続

▼図3 さくらのクラウドのネットワーク構成



ノード側で切り替えを行うため、ToR自体には特別なクラスタリングのしくみは必要ありません。なお、Bondingドライバによっては、単純なリンクダウンをトリガーに切り替えるだけでなく、上位ルータにARPパケットを送信し、そのレスポンスの有無で切り替えを行うような、論理故障にも対応できるモードもあります。

(c)コアスイッチのクラスタリング

最近、イーサネットファブリックと呼ばれる技術が広く使用されるようになりました。クラウドのネットワークはコアスイッチ、ToRを含め広域のL2ネットワークとなります。ファブリックが登場する前までは、STP(Spanning Tree Protocol)と呼ばれるしくみが使用されてきました。しかし、STPではそのしくみ上、大きなL2ネットワークを冗長化し、安定稼働させるのは難しかったのです。ファブリック技術の登場以来、そのようなネットワークを比較的簡単に構築、運用することができるようになりました。

ちなみに、STPは古くからの歴史あるプロトコルで、筆者も以前に設計、運用したことがあります。トラブルが発生しやすいしくみで、何度も泣かされた思い出深いプロトコルでもあります(笑)。

STPでは、L2ネットワークでは本来作ってはならないループをあえて構成し、スイッチに設定したコストやプライオリティに基づいて、一部のリンクを論理的にダウン(Block状態)させます。論理的にループにならないようなトポロジを構成しているのです。

リンクやノードに障害が発生した際には、論理トポロジを再度決定し、Block状態のポートをアップ(Forward状態)させることでトラフィックの迂回を行います。その際、TC(Topology Change)というメッセージをネットワーク全体に送信し、MACアドレステーブルをフラッシュ(破棄)します。L2スイッチはMACアドレスを学習しながら転送先ポートを決めているため、ネットワーク構成(トポロジー:Topology)が変化したときにはMACアドレスを再学習しなければならないのです。その際、しばらく通信が不安定な状態が発生します。昨今の通信要件のシビアなアプリケーションには不向きといえるでしょう。

一方、最近導入が進んでいるファブリック技術では、2台以上のL2スイッチを仮想的に1台としてクラスタリングすることができます。そのメリットとしては、異なるスイッチシャーシをまたいでLAG(Link Aggregation)を組めたり、任意のトポロジで構成できたりと、ループフリーである点が挙げられます。STPのようにBlockポートが存在しないため、宛先に近いほうのリンクを選択したり、トラフィック負荷のロードバランスを行うこともできます。

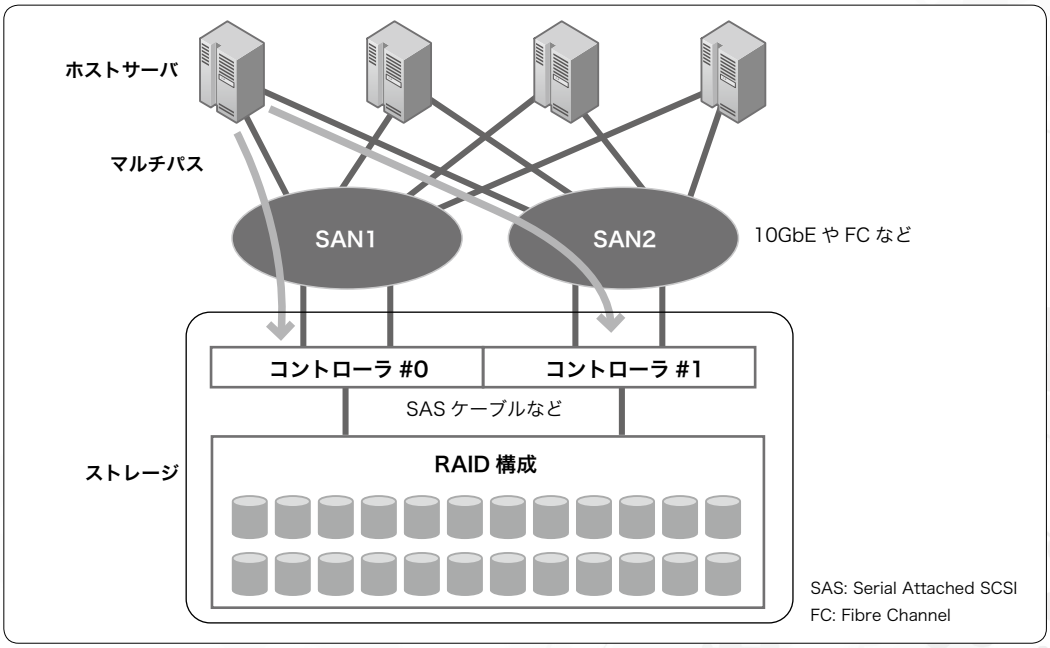
なお、ファブリック技術はメーカーによって実装が異なり、マルチベンダでの構成が事実上できないという欠点もあります。マルチベンダの場合、シャーシまたぎのLAGを用いて相互接続することが多いのですが、メーカーによってmLAG、vLAG、VLTなどの名称/実装が存在し、相性問題もありますので、デプロイには十分な動作確認が必要です。

③ストレージ

クラウドでは、一言にストレージといってもさまざまなものがあります。仮想サーバから内蔵のHDDのように見えるブロックストレージ(たとえばLinuxでは/dev/sdaのようにブロックデバイスとして見える)、HTTPを用いてREST APIにてデータの入出力を行うオブジェクトストレージ、UNIX系OSやWindowsからファイルシステムとしてマウント可能なネットワークファイルシステムなどがあります。ここでは一例として、どのクラウドでも基本的な機能として実装されているブロックストレージのクラスタリングについて説明します。

ホストサーバとストレージ機器間はいわゆるSAN(Storage Area Network)と呼ばれる技術を用いて接続され、ホストサーバは仮想サーバ

▼図4　SANの構成



に必要とするボリュームにアタッチします。**図4**に構成図を示します。

SANを含めたストレージシステムは非常に安定性、高可用性が求められる部分です。レスポンスの遅延は即座に仮想サーバの動作に影響します。ストレージが停止するとクラウドが停止すると言っても過言ではありません。そのような理由から、ストレージ機器は一度運用を始めるとメンテナンスなどでも停止することができません。

ストレージ機器は、電源、コントローラ、ハードディスクやSSDといったすべてのパーツが冗長化されており、一部のパーツが故障してもI/O処理を継続できるようになっています。ホストサーバとストレージ間は2系統のSANで接続され、一般的にFC(Fibre Channel)やiSCSIといったストレージプロトコルを用いて接続されています。コントローラやSANの故障に対しては、マルチパスというしくみを用いて切り替えを行います。

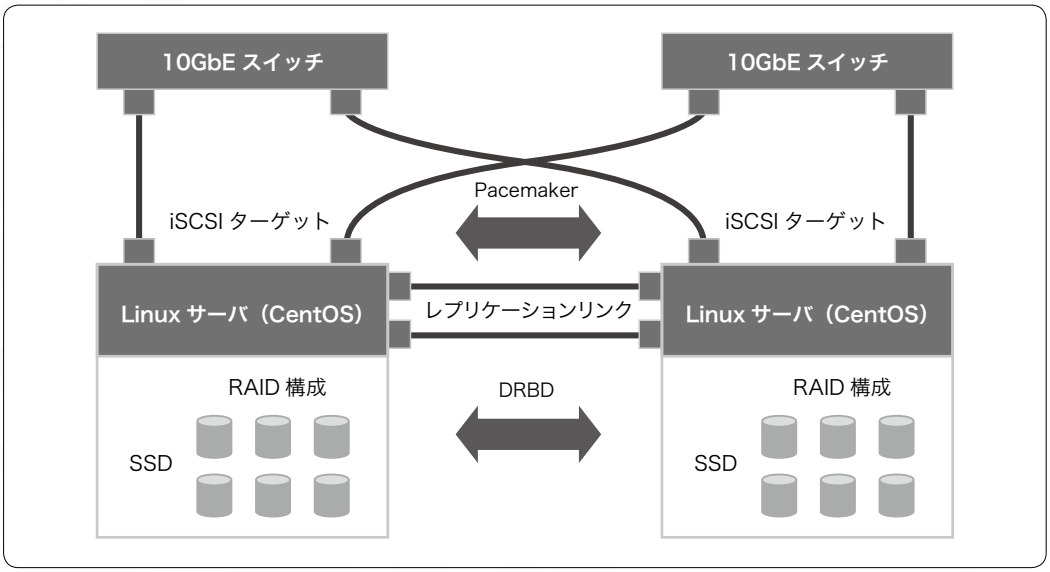
マルチパスは、あらかじめ2系統のSAN経由で仮想ボリュームに対してセッションを張っておき、通常時はプライマリ側コントローラに対してI/O要求を行います。SANやコントローラ故障などでI/Oエラーが返ってきた場合は、バックアップ側コントローラに切り替え、I/O処理を継続します。これにより、SAN自体の故障やストレージコントローラの障害からサービスを保護します。

メーカー製ストレージは上記のようなしくみが一般的に用いられていますが、さくらのクラウドではLinuxにオープンソースを組み合わせた**図5**のようなSSDストレージも運用しています。使用しているソフトウェアは次のとおりです。

・OS：CentOS
・HA：Pacemaker＋DRBD
・ボリューム制御：LVM
・iSCSI Target：scsi-target-utils

SSDをRAID化した同じ構成の2筐体を1セットとし、双方のRAIDプールをDRBDを用いてネットワーク経由で同期レプリケーションしています。DRBDとは、LINBIT社が開発、

▼図5　DRBDを活用したLinuxベースのストレージ



提供しているソフトウェアでオープンソースとして無償で公開されています(一部の機能は有償版となっています)[注3]。

HAの制御にはPacemakerを用いています。マスタ、バックアップ間の生死確認、サービス用の仮想IPアドレスの制御、各種デーモンの起動停止といったコントロールを行っています。

◆ ◆ ◆

以上、IaaSクラウドをケーススタディとした、データセンターで動く仮想化基盤のクラスタリングの実例を紹介しました。クラウドサービスを利用することで、データセンターファシリティから、サーバ、ネットワーク、ストレージまで高可用性を実現したインフラを手軽に利用できることがおわかりいただけたかと思います。

クラウドを利用するうえでのクラスタリングの考え方

ここからはクラウド利用者の視点で、クラウド上に構築するクラスタリングシステムについて、どのような考え方や手法で構成するべきか説明していきます。

前節で説明したように、ネットワークやストレージは無停止の設計が行われていますので、クラウドユーザ側での冗長化は不要です。たとえば、仮想サーバに複数の仮想NICをアタッチし、ネットワークの経路冗長をとる必要はありません。あるいは、複数の仮想ディスクをアタッチして仮想サーバ側でRAID1を構成したりする必要もありません。

一方、ホストサーバの故障時には、別のホストサーバで再起動されるまで、仮想サーバの稼働が一時停止する可能性があります。もし、そのような停止をも許容できないシステムを構築する場合は、クラウド利用者側にて高可用性実現のためのしくみを構築する必要があります[注4]。

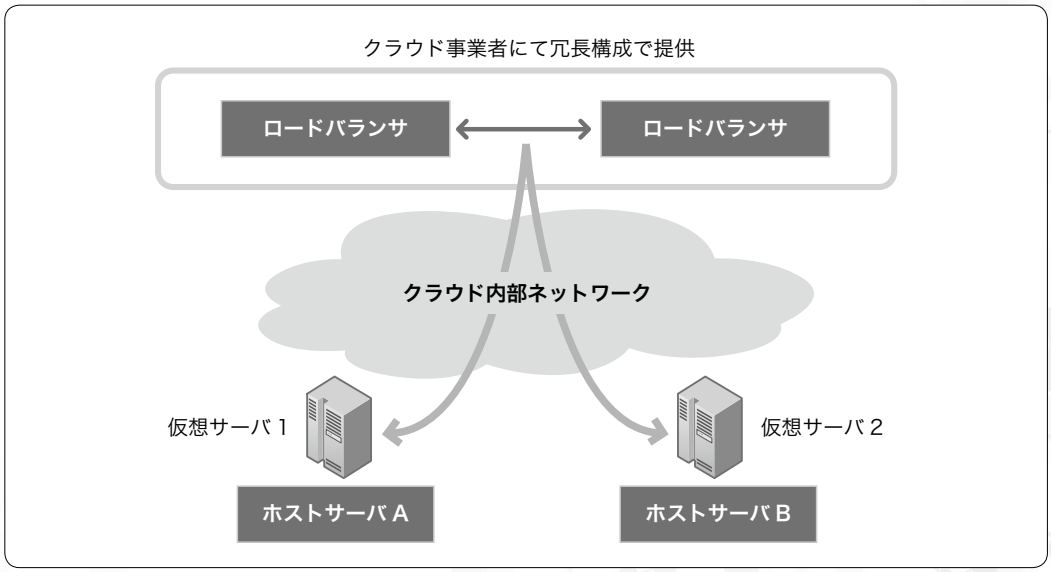
クラウド上に構築した仮想サーバを冗長化する手っ取り早い方法は、クラウドにて提供されているロードバランサを用いる方法です(**図6**)。

通常、ロードバランサは負荷分散目的で使用されますが、高可用性を実現するためのパーツとしても利用可能です。クラウド事業者側でロー

注3) 詳細はDRBDのWebページを参照してください。
http://www.drbd.org/ja/

注4) HA機能が実装されているかどうか、またその場合の再起動時間の目安については、クラウドベンダによって異なります。利用しているクラウド基盤の仕様をあらかじめ確認しておきましょう。

▼ 図6　ロードバランサを用いた仮想サーバの冗長化



ドバランサ自体も冗長化されています[注5]。

なお、通常のロードバランサ(ローカルタイプのもの)では、同一のIPアドレスが引き継げる範囲でしか使用できません。異なるグローバルIPアドレス間でフェイルオーバーする必要のある場合(ゾーンやリージョン、あるいはクラウドをまたぐような環境)では、GSLB(Global Server Load Balancing)を使用してください[注6]。

仮想サーバ冗長化の注意点

仮想サーバの冗長化を行う場合に重要な注意点があります。通常、仮想サーバを収容するホストサーバは、クラウドコントローラのアルゴリズムによって任意に選択されます。冗長化しているつもりの2台の仮想サーバが、たまたま同一のホストサーバに収容されてしまうと同時にダウンする可能性があります。

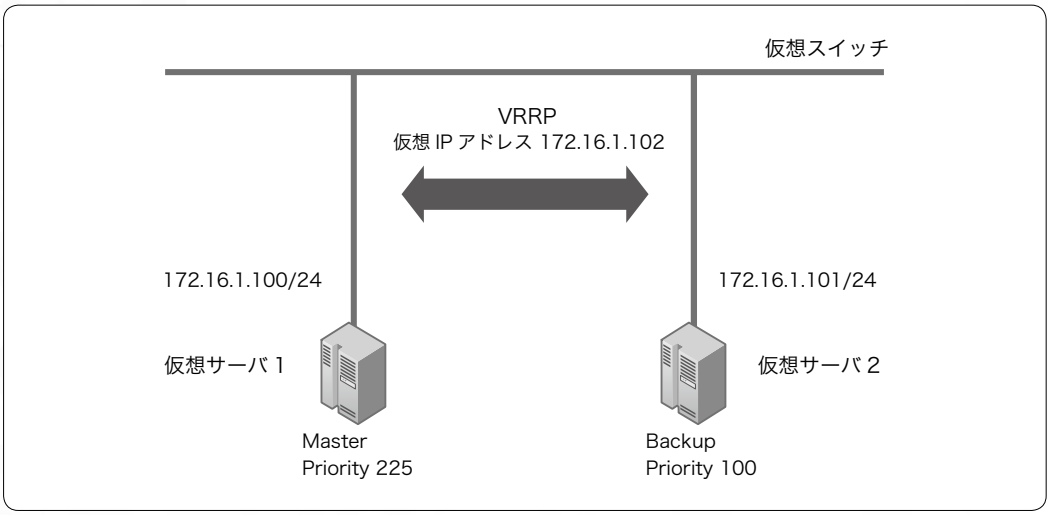
このような事態を回避するため、**図6**のように2台の仮想サーバが異なるホストサーバに収容されるように設定を行ってください。たとえばさくらのクラウドでは、@groupタグを設定することで制御可能です(**図7**)。

▼ 図7　さくらのクラウドにおける@groupタグの設定例

@groupタグは、@group=aから@group=dまでの4種類が存在し、たとえば、@group=aと@group=bのように異なるタグを付与した仮想サーバは必ず別のホストサーバにて起動されます。なお、タグの効果はサーバ起動時に反映されますのでご注意ください[注7]。

注5) さくらのクラウドでは、ロードバランサの冗長化あり／なしが選択できるようになっています。あらかじめ利用しているクラウドの仕様を確認しておきましょう。
注6) 有名なところではAmazon Route 53などがあります。
注7) 詳細は以下URLを参照してください。
http://cloud-news.sakura.ad.jp/special-tags/

▼図8　VRRPを用いた仮想サーバの冗長化



VRRPを用いた仮想サーバのクラスタリング

アプリケーションによってはクラウドサービス側で提供されているロードバランサが使用できなかったり、自前でロードバランサを構築したいといったシチュエーションもあります。そのような場合、クラウド利用者にて冗長化設定を行う必要があります。ここではLinux(CentOS 6.5)を例にVRRPを用いてサーバを冗長化する方法を紹介します。ここで説明する環境は**図8**のようなものです。

なお、クラウドの機能として次の条件を満たす必要がありますので、事前にご確認ください[注8]。

・2台の仮想サーバを同一のセグメント(VLAN)に接続できること
・VRRPのマルチキャストが2台の仮想サーバ間で疎通できること
・IPエイリアスを使用できること

まずは2台の仮想サーバを用意し、keepalivedをインストールします。

▼図9　keepalived.confファイルの内容

```
vrrp_instance V1 {
    state BACKUP ← (a)
    interface eth0 ← (b)
    virtual_router_id 2 ← (c)
    priority 255 ← (d) バックアップ側は100に
    advert_int 5 ← (e)
    virtual_ipaddress {
        172.16.1.102 ←(f)
    }
    notify_master /path/to_master.sh ← (g)必要に応じて
    notify_backup /path/to_backup.sh ← (g)必要に応じて
}
```

```
# yum install keepalived
```

続いて、keepalivedの設定を行います。/etc/keepalived/keepalived.confファイルを**図9**の要領で作成してください。設定内容は次のとおりです(a～gの記号は**図9**に対応)。

(a) state

どちらの仮想サーバも"BACKUP"を指定します。MASTERというキーワードもあるのですが、ここではpriorityに応じてマスタを選出する動作を行うため、初期状態(keepalivedの起動直後)ではバックアップ状態になるように

注8) さくらのクラウドでは、「スイッチ」および「ルータ+スイッチ」にてVRRPを利用可能です。共有セグメントでは使用できません。

▼図10　仮想IPアドレスの確認

```
# ip addr show dev eth0
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500
    link/ether 9c:a3:ba:21:cb:57 brd ff:ff:ff:ff:ff:ff
    inet 172.16.1.100/24 brd 172.16.1.255 scope global eth0
    inet 172.16.1.102/32 scope global eth0 ← 仮想IP
```

しています。

(b) interface

VRRPを動作させるインターフェース名を指定します。

(c) virtual_router_id

VRRPのグループを指定するID(VRID)で、1から255の範囲の数値を指定します。同じサブネットで異なるVRRPグループを動作させる場合、この値がユニークになるように設定します。

(d) priority

priorityの最も高いノードが自動的にマスタとして選出され、仮想IPアドレスを保有します。ここでは、マスタを255、バックアップを100としています。もちろんほかの値でも構いません。

(e) advert_int

死活監視に用いられるVRRP Helloの送出間隔(秒)です。障害発生時には、ここで指定した秒数の3倍程度の時間で切り替わります。小さな値にするほど短時間で切り替わるようになりますが、負荷がかかり、瞬間的に処理が遅延した際に不必要なフェイルオーバーが発生して不安定になる場合もあります。運用しながら値を調整するようにしましょう。

(f) virtual_ipaddress

マスタノードが保有する仮想IPアドレスです。複数必要な場合は、並べて記載します。

(g) notify_master, notify_backup

それぞれマスタ、バックアップに遷移した際に実行するスクリプトを指定します。先に説明したとおり、VRRPは単純に仮想IPアドレスを付け替えるだけの機能しかありません。状態が遷移した際にデーモンを起動・停止したり、何らかの動作を行いたい場合はこのパラメータで制御します。

設定が完了したら、keepalivedを起動します。

```
# service keepalived start
```

しばらくすると、VRRPにてマスタノードが選出され、仮想IPアドレスが付与されることが確認できます(**図10**)。

マスタ側をダウンさせ、仮想IPアドレスがバックアップ側に移るかどうか確認してみてください。

まとめ

本稿では、データセンターのファシリティからクラウドインフラまで、どのようにクラスタリングや高可用性を実現しているのか、また、クラウド上に構築するシステムのクラスタリングの考え方と簡単な設定例を紹介しました。いかがでしたでしょうか？

高可用なシステムを構築するには、機器やオペレーションも含めてコストがかかります。運用するサービスレベルや収支のバランスなどを鑑みて、適切な高可用設計を行いましょう。SD

第3章 MySQLをベースに利点と注意点を整理

データベースのクラスタ構成とミラーリング方式

データベース(DB)には、アプリケーションから常時必要とされ、かつ失われてはいけないデータが保管されているため、可用性が重要となります。ディスク冗長化のほかに、サーバ自身の障害も考慮した高可用性クラスタリング構成が必要とされます。本章ではMySQLを中心に、DBのクラスタリング技術と構築運用時の考慮点を解説します。

●日本オラクル(株)　梶山 隆輔(かじやま りゅうすけ)

データベースのクラスタ構成に共通の考慮点

データベースのクラスタ構成を考える際には、どんな構成の場合にも、考慮すべき点が4つ挙げられます。具体的には、障害検知、フェイルオーバー、スプリットブレイン、ビジネスからの観点の4つです。以下にその詳細を述べます。

障害検知

クラスタリング構成内のデータベースに障害が発生したことを検知するために、複数のレイヤでの監視が行われることがあります。最も一般的なのは、ネットワークレイヤにて相互にハートビートパケットと呼ばれる監視パケットの通信を行うこと、およびサーバにインストールされたエージェントプログラムによるOS上のプロセス監視です。多くのクラスタリング構成ではTCP/IPによる通信経路となる複数のネットワークパスを設定することが推奨されています。環境によっては、TCP/IP経由での監視に加えて、サーバ間をシリアルケーブル経由で監視することもあります。

このハートビートパケットのチューニングには注意が必要です。ハートビートパケットを送信して応答がなく、タイムアウトになったケースが複数回発生すると、障害として検知されることが一般的です。タイムアウト値を小さくし過ぎると、ネットワーク遅延が大きい環境やシリアルケーブル経由の場合は、単に応答が遅いだけでも障害として見なされてしまいます。また、タイムアウト後のリトライ回数が小さい場合には、一時的な応答の遅延でも障害として検知されて、フェイルオーバーが発生してしまう可能性が高まります。

逆に、タイムアウト値を大きくし過ぎた場合やリトライ回数を多くした場合は、障害が検知されるまでの時間が長くなり、アプリケーションやビジネスへの影響が大きくなってしまいます。

データベースプログラムによっては、サーバのプロセスが自動的に再起動されるように構成されていることもあります。たとえば、MySQLの起動スクリプト「mysqld_safe」では、MySQLサーバプロセスの停止時に、自動的に再起動するしくみになっています。クラスタリング構成にて、プロセスの障害をきっかけとしてフェイルオーバーする処理と競合してしまう可能性があるので、mysqld_safeを使わずにMySQLサーバプログラムを直接起動するように設定する必要があります。

また、データベースサーバのプロセスの稼働状況に加えて、クライアントプログラムから接続してSQL文を発行し、データベースとして正しく稼働しているかの確認も求められます。これは、OSが正常に稼働していてハートビートパケットへの応答ができており、かつデータベースサーバのプロセスは起動しているものの、データベースサーバ内で何らかの問題が起きており正しく利用できない状況が考えられるためです。

Oracle DatabaseやMySQLで用意されてい

るDUAL表に対してSELECT文を投げることもありますが、DUAL表では、実際のアプリケーションデータが格納されている領域や、スキーマの状況がわかりません。アプリケーションデータが格納されている領域や、スキーマに置かれたダミーのテーブルを使って確認することもあります。

フェイルオーバー

サーバに障害が発生した場合、該当のサーバをクラスタリング構成から切り離して、別のサーバで処理を継続させます。フェイルオーバーの際には、アプリケーションからデータベースへの通信経路や、データベースサーバからデータを物理的に格納しているデバイス(ストレージ)へのアクセスの切り替えを行います。

アプリケーションからの通信経路は、ロードバランサなどのネットワーク機器で切り替えるケース、仮想IPアドレスを障害が発生したサーバから別のサーバに付け替えるケース、アプリケーションサーバやドライバの接続フェイルオーバー機能を使うケースなどが考えられます。

MySQLのJDBCドライバ「Connector/J」やPHP用ドライバmysqlndのプラグイン「mysqlnd_ms」などは、MySQLサーバ障害発生時に、接続を別のMySQLサーバに切り替えるオプションを持っています。

障害の検知からフェイルオーバーの完了までにかかる時間は、方式や構成によって大きく変わります。全ノードが稼働状態にあるアクティブ／アクティブ型では、障害が起きたサーバを切り離すだけで、切り替えが必要なコンポーネントは少ないためフェイルオーバーにかかる時間は短くなります。平常時は待機しているだけのサーバに切り替えるアクティブ／スタンバイ型では、最悪の場合は、

・ファイルシステムパーティションのマウント
・fsckなどによるファイルシステムの検査
・データを一致させる処理
・データベースプログラムの起動
・データベースの自動リカバリ
・アプリケーションからの接続受け入れ開始
・仮想IPアドレスの切り替え
・アプリケーションの再起動

などが続き、数分から数十分単位での時間がかかってしまうことがあります。

スプリットブレイン

クラスタリング構成で注意すべき課題にスプリットブレイン状態が挙げられます。ネットワークに部分的な障害が起き、サーバ間で相互に通信ができなくなったものの、アプリケーションからは接続できてしまう状態です。それぞれのサーバはほかのサーバに障害が起きたと判断するため、引き続き処理を継続しようとするサーバ同士や、フェイルオーバーで処理を引き継ごうとするサーバとの間で、リソース利用の競合やデータの不整合が起こる可能性があります。

スプリットブレインによる不整合を防ぐためには、どのサーバが処理を継続し、どのサーバを停止すべきかの判定をする必要があります。判定方法には、次のような複数の方式があります。

・通信可能なサーバの数の多数決(quorum vote)
・アクセス可能な範囲に特定のサーバや共有リソースを保有できているかで決定
・外部のサーバが調停役(arbitrator)として判断

この判定の際に、誤って全サーバを停止してしまうことや、停止と再起動が繰り返されてしまわないように設定に注意が必要です。

ビジネスからの観点

高可用性を高めることは、一般的に運用の複雑性やコストの上昇につながります。そのため、ビジネスの観点から可用性の要件を検討することが必須です。データベースやシステムインフラを直接担当していない側からは、「常に動い

ていて当たり前」と、とらえられることが多々ありますが、システムのメンテナンスや障害を想定した停止許容時間(**表1**)を認識することが重要です。

4つのデータベースクラスタリング構成

クラスタリング構成のデータベースにおけるデータ管理は、それぞれのサーバに格納する「データミラー型」と、共有ストレージに格納する「ディスク共有型」に大別されます。また、すべてのサーバがアプリケーションから利用可能かどうかで分類することができます。それぞれの特徴を整理したものが**表2**、図で表したものが**図1**になります。

データミラー型では、データをコピーするタイミングや転送される内容によって応答性能が

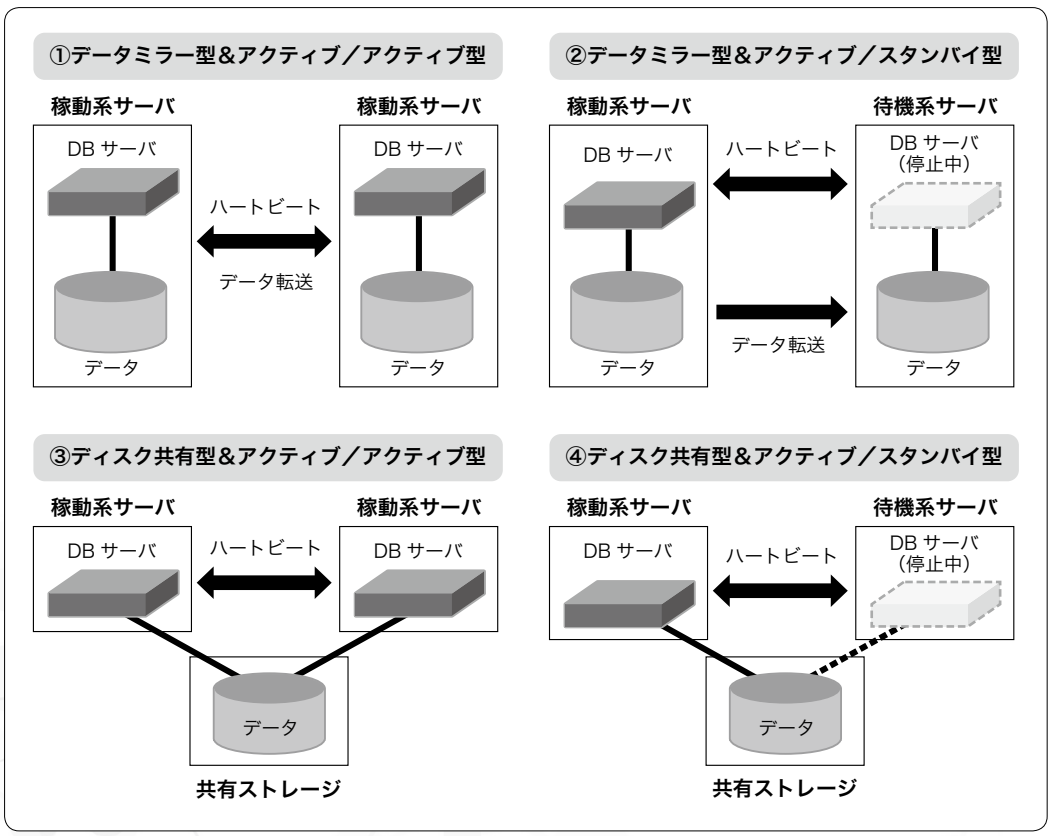
▼表1 年間停止可能時間

可用性	99%	99.9%	99.99%	99.999%
停止可能時間	3.65日	8時間45分	52分30秒	5分15秒

▼表2 4つのデータベースクラスタリング構成(比較)

	①データミラー型&アクティブ/アクティブ型	②データミラー型&アクティブ/スタンバイ型	③ディスク共有型&アクティブ/アクティブ型	④ディスク共有型&アクティブ/スタンバイ型
レスポンス	△〜○	△〜○	○	○
性能拡張性	◎	×	○	×
フェイルオーバー時間	○	△	○	△
構成例	MySQL Cluster	DRBD、SIOS LifeKeeperほか	Oracle Real Application Clusters	SIOS LifeKeeper、NEC CLUSTERPROほか多数

▼図1 4つのデータベースクラスタリング構成(システム構成図)



異なります。詳細は後述します。

① データミラー型&アクティブ／アクティブ型

「MySQL Cluster」に代表されるのが、共有ディスクを使わず、かつすべてのノードがアクセス可能な構成です。データベースへの同時アクセス数やデータ量に応じてサーバを追加して、柔軟に拡張できることが最大のメリットです。また、サーバ台数が2台に限定されますが、双方向にデータレプリケーションを行う構成も類型と言えます。

たとえば、MySQLのレプリケーションでは、双方向のレプリケーション構成とすることが可能です。ただし、トランザクションはそれぞれのノードで管理され、非同期でデータが相互に転送されるため、データの変更順が保証されずデータの矛盾が起きてしまう可能性があります。MySQLのレプリケーションでは、この矛盾の検知や解決はできません。ID番号別にデータ変更するサーバを固定するなど、「同一のレコードは、同時に別のサーバで変更しない」ように、アプリケーション側での作り込みが必要です。

②データミラー型&アクティブ／スタンバイ型

「DRBD（Distributed Replicated Block Device）」[注1]や「SIOS LifeKeeper」などのクラスタリングソフトウェアで実現可能な構成です。このようなデバイスレベルでの実装の場合、スタンバイサーバでは、データがコピーされるパーティションをアクティブサーバからアンマウントしておく必要があります。フェイルオーバー時にはファイルシステムのマウントが行われることにより、時間を要する場合があるので注意が必要です。

また、DRBD単体ではハートビートパケットによる障害検知や障害時のフェイルオーバーが行えないため、ノードの死活監視を行う「Corosync」や、クラスタリソースの切り替えなどを行う「Pacemaker」と組み合わせて利用されます。

一方で、データベースレベルでのデータ複製を行うMySQLのレプリケーションでは、コピー先でもMySQLサーバが起動しており、コピーされたデータの参照が可能となっています。そのため、単に高可用性としての利用だけではなく、複数のサーバによって並列で参照処理に対応する必要があるWebシステムのバックエンドとして広く利用されています。また、常にMySQLサーバが起動しているため、フェイルオーバーが高速になるという利点があります。

③ディスク共有型&アクティブ／アクティブ型

「Oracle Real Application Clusters」に代表される構成で、データを共有ストレージに格納し、すべてのデータベースサーバがアプリケーションから利用可能になっています。1ヵ所にデータが集約されているため、テーブルを広くスキャンするような処理や多くのテーブルを結合する処理などは高速です。

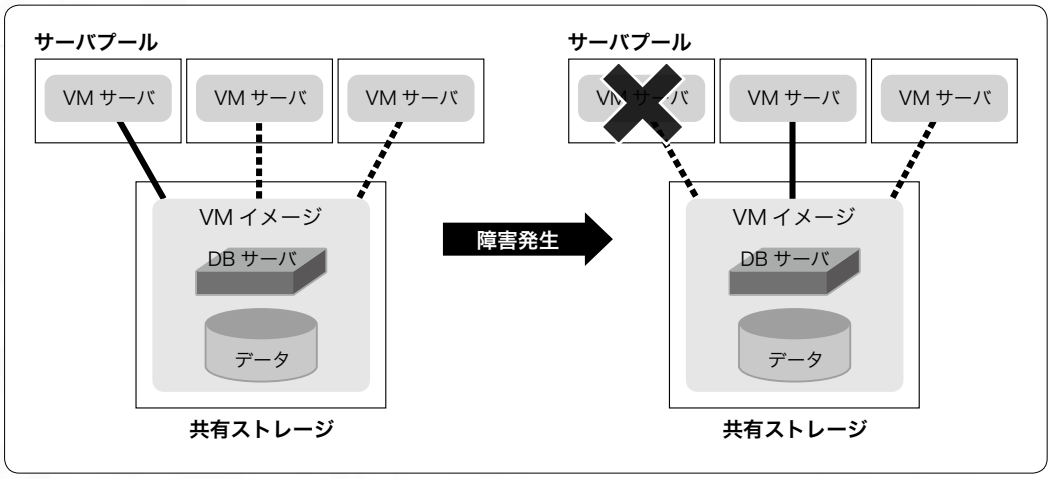
一方で、注意すべき点もあります。複数のアプリケーションが同時に同じデータや同じデータブロック内のレコードを変更する処理の場合、データの一貫性を保つためにデータベースサーバのキャッシュの最新化処理が行われます。この最新化処理により、性能低下が起きないように並列実行を避ける工夫が必要です。また、共有ディスクが単一障害点にならないよう、ストレージデバイス、ファイバチャネル、およびスイッチの多重化を行うため、コストが増大する傾向にあります。

④ディスク共有型&アクティブ／スタンバイ型

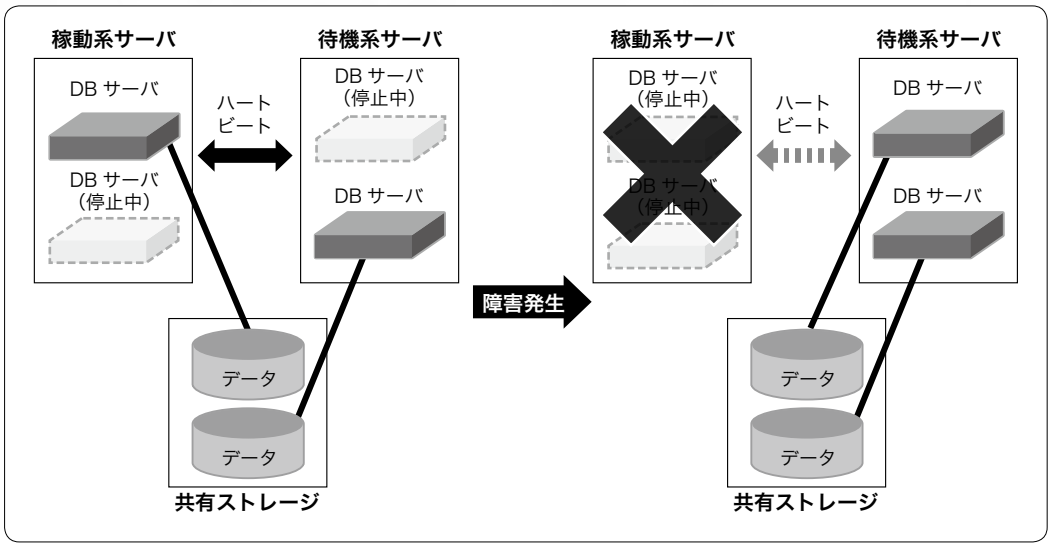
多くのクラスタリングソフトウェアで利用可能であり、小規模なシステムから大規模なシステムまで幅広く利用されている実績のある構成です。類型としては、仮想マシンイメージを共

注1） http://www.drbd.jp/

▼図2　VMイメージを共有ディスクに置いたアクティブ／スタンバイ型



▼図3　アクティブ／スタンバイ型を2組用意した擬似的なアクティブ／アクティブ型



有ディスクに置いて運用する方法もあります(**図2**)。しかし、これはフェイルオーバー時には、仮想マシン上のOSの起動からとなるため、フェイルオーバーに時間がかかる可能性があります。

アクティブ／スタンバイ型の構成を2組用意して、平常時はすべてのサーバが稼働しているように運用することも可能です(**図3**)。共有ディスク上に別のデータディレクトリを用意して、それぞれのサーバで別のデータディレクトリを利用するデータベースプロセス(インスタンス)を起動します。データファイルを共有しないため、アクティブ／アクティブ型ではありません。

片方のサーバに障害が発生すると、もう片方のサーバで2つのデータベースプロセス(インスタンス)が起動され、CPUやメモリなどの利用が増加するため、実際に利用されるケースがかなり限定的になります。

データミラーリング方式の詳細

ここではデータミラー型のクラスタリング構成におけるデータミラーリング方式を、

MySQLサーバに関する構成を例に解説していきます。データミラー型の場合、データをコピーするタイミングや、サーバ間でコピーされる内容はさまざまです。

トランザクションがコミットされた際に、データをコピーするタイミングは大きく3つのパターンに分かれます。

・非同期：データの転送とコピー先でのデータへの反映を待たずに、メインのサーバからアプリケーションに応答を返す
・準同期：データの転送は待つが、コピー内容がデータに反映されるのを待たずに、メインのサーバからアプリケーションに応答を返す。MySQLのレプリケーションの場合、トランザクションログのコピーは待つが、データファイルへの反映は待たない
・同期：データの転送とコピー先でのデータへの反映を待ってから、メインのサーバからアプリケーションに応答を返す

性能の観点では、非同期型がデータの変更処理に対する応答時間が最も短く、同期型が最も長くなっています。一方で、データの耐障害性の観点では、確実にデータがコピーされる同期型のほうが優れています。非同期型の場合、アプリケーションに応答したもののデータの転送が完了していないタイミングで、メインのサーバが停止してしまうと、「コミットが成功したのにデータを失ってしまう」という事象が発生し得ます。

準同期型は、これらの方式のそれぞれのデメリットを克服できます。ただし、コミットの応答がアプリケーションに返ってきた時点では、コピー先のデータがまだ反映されていない可能性があります。MySQLのレプリケーションでは、1台のサーバには準同期型としてデータをコピーし、ほかのサーバには非同期型として同時に配信することもできます。

サーバ間のデータを一致させる方法を整理すると、**表3**のとおりになります。

トランザクションログのSQL文を転送する論理方式

MySQLのレプリケーションのデフォルトの挙動は、マスタのバイナリログに記録されたSQL文をスレーブ[注2]のリレーログに転送する「文ベース（Statement Based Format）」となっています。スレーブではリレーログに書き込まれたSQL文を実行してサーバ間のデータを一致させます。バイナリ化されたSQL文を転送するため、1つのUPDATE文で100万行更新する場合でも、サーバ間で渡されるのはUPDATE文のみとなります。転送されるSQL文は、バイナリログファイルに対してmysqlbinlogコマンドを実行することで確認できます。

この形式で注意すべきは、SQL文で利用す

▼表3　サーバ間のデータミラーリング方式

	論理方式	レコード方式	物理方式	二重永続化方式
構成例	MySQLの文ベースレプリケーション	MySQLの行ベースレプリケーション	DRBD、SIOS LifeKeeperのディスクミラー	MySQL Cluster
転送されるデータ	トランザクション（バイナリログの内容）	行イメージ（バイナリログの内容）	データブロック	トランザクション
タイミング	非同期または準同期	非同期または準同期	同期	同期
コピー先での参照更新	参照のみ可能 ※	参照のみ可能 ※	参照更新不可	参照更新可能
データ不整合	関数により不整合の可能性あり	不整合なし	不整合なし	不整合なし

※MySQLのレプリケーションのスレーブ側では、データ更新はできますが、データ矛盾の検知や解決はされないため注意が必要です

注2）データのミラーリングにおいて、コピー元となるデータベースやディスクを「マスタ」と呼び、マスタからコピーされるほうを「スレーブ」と呼びます。

る関数によってはデータの不整合が起こり得る点です。たとえば、サーバ固有のIDであるUUIDを取得するUUID()関数や、システム日付を取得するSYSDATE()関数などは、マスタとスレーブで実行されるタイミングによって返る値が異なるため、これらの関数を利用して値を変更する場合などに問題となります。問題となり得る関数やSQL文のリストは下記のURLを参考にしてください。

参考URL：http://dev.mysql.com/doc/refman/5.6/en/replication-rbr-safe-unsafe.html#idm47782054744368

これらの関数などを利用する場合には、次項の「行ベース(Row Based Format)」、または両方の方式のうち最適なものを自動的に選択する「MIXED」に設定します。

データベースの行イメージを転送するレコード方式

バイナリログの形式を「行ベース」に設定すると、バイナリログには変更後の行イメージが記録されます。なお、MySQL 5.6から導入されたサーバオプションbinlog_rows_query_log_eventsを有効にすると該当するトランザクションでのSQL文もバイナリログに記録されます。mysqlbinlogコマンドに-vv(小文字のvを2個続ける)オプションを付けて実行すると記録されたSQL文をテキスト化できます。

この形式のバイナリログを利用した「行ベース」レプリケーションでは、行イメージをスレーブのリレーログに書き込み、その内容をスレーブのデータに反映します。行イメージをそのままコピーするため、どのような関数を利用しても問題がなく、スレーブでのロックが最小限となることがメリットです。

ただし、行イメージがバイナリログに記録されて転送されるため、「文ベース」と比較してサーバ間で渡されるデータ量が大きくなるケースがほとんどです。MySQL 5.6以降で利用可能なサーバオプションbinlog_row_imageをminimalに設定すると、主キーと変更のあった列のみをバイナリログに記録するため、「行ベース」レプリケーションで転送されるデータ量を抑えることができます。また、SQL文が実行されないので、「行ベース」のレプリケーションではスレーブ側でのトリガが実行されません。

OSのデータイメージを転送する物理方式

DRBD(Distributed Replicated Block Device)は、ハードディスクなどのブロックデバイスの上、ファイルシステムの下で稼働し、データブロックをLinuxサーバ間でミラーリングするオープンソースのソフトウェアです。Linuxカーネルの2.6.33からカーネルモジュールとしてパッケージされています。

DRBDでは特定のディレクトリやファイルを指定することはできず、/dev/sda1のように表現されるブロックデバイス、またはパーティション単位でのミラーリングを行います。そのため、MySQLなどのデータベースに限らず、ファイルシステムまたはRAWデバイス経由でディスクにデータを書き込もうとするあらゆるプログラムのデータが、ミラーリング対象になり得ます。

DRBDが設定された環境でデータをディスクに書き込もうとすると、DRBDのモジュールがリモートのサーバのDRBDモジュールに内容を転送し、それぞれのディスクに実際に書き込みにいきます。この際のデータの転送のタイミング、およびどこまで待機系の応答を待つかを決めるprotocolオプション(設定値はA、B、C)によって、耐障害性とアプリケーションへの応答性能が変わってきます。

デフォルトでは、完全同期レプリケーションのCとなっています。これは、両ノードのディスクに書き込みが完了してから応答を返すので、応答性能は最も低いものの、1台の障害ではデータを失うことがない最も耐障害性のある設定となっています。逆にAは、自機のディスクへの書き込みとコピーするデータを自機のTCPバッ

ファに載せた段階でアプリケーションに応答を返すため、応答性能は高いが、耐障害性が低い設定となります。Bはその間を採った設定です。

参考URL：http://www.drbd.jp/users-guide/s-replication-protocols.html

なお、データがコピーされるブロックデバイスは、アンマウントされている必要があります。MySQLのレプリケーションとは異なり、データのコピーを受け取る側ではMySQLサーバを稼働させることはできません。

データを自動的に冗長化する二重永続化方式

MySQL Clusterでは、SQL文の構文解析やデータの集約を行うSQLノードと、トランザクションの管理とデータの永続化を行うデータノードにプロセスが分かれています。データノードは通常2台1組でデータの二重永続化を行っています。

トランザクションを司るトランザクションコーディネータとして選ばれたデータノードに、SQLノードからトランザクション内容が転送されてきます。トランザクションコーディネータから該当するレコードを格納するデータノードに対して、書き込み準備の確認をそれぞれ行ってから実際にレコードを書き込みにいく2フェーズコミットを行っています。これによりMySQL Clusterは高い耐障害性を持つことができています。データとトランザクションログを書き込んで、チェックポイントのタイミングでディスクに書き出すことでレイテンシを抑えていますが、より高い性能のネットワークを利用することが重要です。

ハートビートパケットをノード間で相互に送ることによって障害の検知を行い、ノードに障害が発生すると自動的に構成から外され、残りのノードで処理を継続します。アプリケーションからSQLノードへの接続に関しては、JDBCドライバのConnector/Jなどの接続部品の機能で接続の切り替えが可能です。

なお、MySQL Cluster構成内のMySQLサーバのバイナリログにトランザクション情報を集約する「Binlog Injector」と呼ばれるしくみがあり、このバイナリログを使ってほかのMySQL Clusterに対して非同期レプリケーションを行うことができます。この非同期レプリケーションをデータセンター間で行うことで、ディザスタリカバリ構成とすることが可能で、大手の通信事業者の加入者データベースなどでも利用されています。さらに高い可用性が必要となる場合は、同じデータを持つデータノードを複数のデータセンターに配置して、レイテンシの極めて低いネットワークでつなぐことで1つのクラスタ構成とする方法もあります。

MySQL Clusterでは、データを各テーブルの主キーでパーティショニングを行って複数のデータノードに分散配置します。構成するサーバを追加することでスループットやデータ容量を柔軟に拡張できます。サーバ追加時には自動的にデータの再構成がバックグラウンドで行われます。さらに、レイテンシを最小限に抑えたC++やJavaなど複数のNoSQL APIによって高速なデータアクセスを実現しています。

まとめ

データベースのクラスタリング構成の中でも、データミラー型は共有ストレージやファイバーチャンネルハブなどのストレージ関連機器が不要なため、低コストで高可用性構成を実現できます。アプリケーションの要件に応じたデータミラーのタイミングおよび方式の選択が可能となっているのも大きな利点です。とくに同時多発的に大量のトランザクションが発生するアプリケーションや初期投資を抑えつつも将来的な利用者の増加に対応する必要があるシステム向けには、MySQL Clusterによる高い拡張性を備えた高可用性構成を活用できると思います。SD

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?

ベアメタルクラウド活用入門

IBMが満を持してサービスを開始したクラウドサービス「SoftLayer」を紹介します。全3回の予定で、まずはその使い方を入門的に紹介し、次回は少し応用的な例を、そして3回目ではMVCモデルに基づいたWebシステムの構築例を解説していきます。今回は、ベアメタルサーバの特徴をおもに紹介します。

Writer 常田 秀明(ときだ ひであき) 日本情報通信㈱ Hideaki_Tokida@NIandC.co.jp
北瀬 公彦(きたせ きみひこ) 日本アイ・ビー・エム㈱ kitasek@jp.ibm.com

SoftLayerが注目される理由

SoftLayerは、今もっとも注目されているIaaS型クラウドの1つです。IBMは2013年7月に、SoftLayer Technology社を買収し、それ以降IBMのクラウドサービスとして展開することになりました。2014年1月に追加投資を行い、新たにデータセンターを世界中に開設しています。最近では6月に香港、7月にロンドンデータセンターが完成し、日本でも2014年中にデータセンターが開設される予定です。

昨年まで「SoftLayer」を知らなかった方も多いのではないでしょうか。実は、海外では「Hostcabi.net」の人気ランキングでは常にトップ5入りするクラウドです。欧米ではslideshare.net、yelp、Citrix Systemsなどが、日本では、データホテル、東芝 クラウド&ソリューション、東急ケーブルネットワークなどがSoftLayerの顧客です。このSoftLayerについて、いくつかの特徴を挙げて解説します。

高速なグローバルネットワークが低コストに使える!

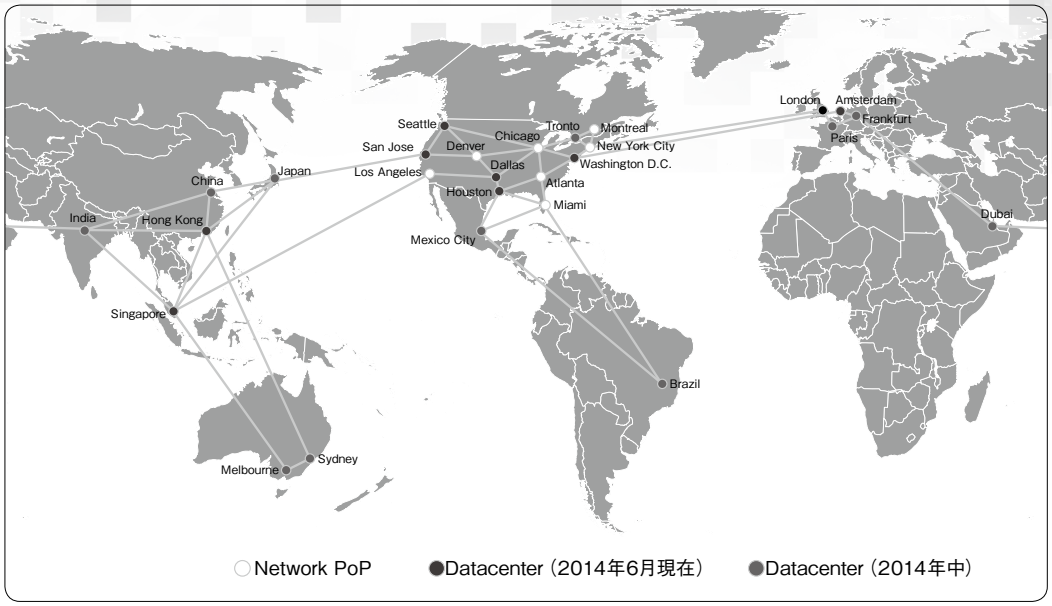
図1に示すようにSoftLayerは、全世界にある16のデータセンター、19のネットワーク拠点間で、高速で安定したグローバルネットワークを提供しています(2014年7月現在)。このグローバルネットワークは、複数のキャリアの10Gbpsリンクを冗長構成した高速なネットワークです。2014年中にさらに拡張される予定です。SoftLayerのユーザは、このグローバルネットワークを無料で使用できます。パブリックネットワークからのインバウンドトラフィックについても無料です。パブリックネットワークへのアウトバウンドトラフィックに関しては、仮想サーバを使用していれば、1仮想サーバにつき5TB/月まで無料です。物理サーバを利用していれば、1物理サーバにつき20TB/月まで無料で使えます。たとえば、アウトバウンドのネットワークトラフィックが大量に発生する動画転送システムなどが、とくにSoftLayerと相性が良さそうです。

ベアメアタルサーバでクラウドがもっと自由になる!

もう1つの特徴として、仮想サーバをクラウド上に作るのと同様の手続きで物理サーバを作成できるという点が挙げられます。シンプルな物理サーバならば小一時間で、複雑な構成のサーバでも数時間で作成できます。物理サーバをクラウド上に作成できるので、実現したいシステムのバリエーションが増えます。たとえば、サーバ仮想化やデスクトップ仮想化が挙げられます。そしてOpenStack、CloudStackなどを使ったプライベートクラウドもクラウド上に構築できるようになります。つまり、ユーザはオンプレミスのシステムをたやすく移行できるのです。

また、作成されたサーバは、パブリックとプライベートネットワークの2つのインターフェー

▼図1　SoftLayerのグローバルなネットワーク構成



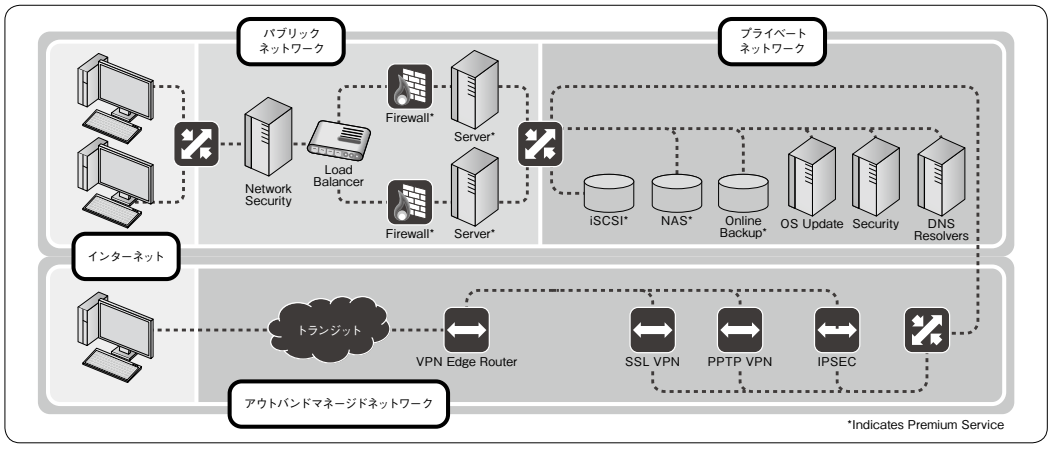
スを持ちます。これらはVLANで実現されます。パブリックネットワークでは、ファイアウォールやロードバランサなどのネットワークサービスを配置できますし、プライベートネットワークでは、iSCSI、NASなどのストレージを配置できます(**図2**)。

さらに、仮想サーバからイメージを作成し、そのイメージをもとに物理サーバを作成できます。開発環境では仮想サーバを使用し、本番環境で物理サーバを利用するなど利用シーンに応じて使い分けもできます。そして非常にたくさんのAPI(Application Programming Interface)も用意されています。このAPIにより、仮想、物理、両サーバを、スクリプトやアプリケーションからコントロールできます。

SoftLayerはインフラエンジニアの技が活かせる!

SoftLayerには、ベアメタルサーバを提供できるという大きな特徴があります。つまり、クラウド上であっても、インフラエンジニアがこ

▼図2　ベアメタルサーバを含むシンプルな構成



れまでに培ってきたスキルを使って、クラウド上にシステムを構築できることを意味します。新たに複雑なクラウドサービスメニューを覚える必要はありません。オンプレミスのシステムとほぼ同じアーキテクチャを使い、クラウド上にシステムを構築できます。その意味では、非常にわかりやすいクラウドだと言えます。

SoftLayerの使い方

【1st step】アカウントの作成

このサービスを利用するには、SoftLayerと契約し、アカウントを取得する必要があります。利用契約が完了すると、実際にIaaSを利用できるようになります。このアカウントを利用してSoftLayerのさまざまなサービスを利用することになります。

アカウントの発行は無料です。しかし、アカウントを作るためには必ず1台サーバを作成(購入)しなくてはなりません。「ハードルが高い！」と感じられるかもしれませんが、実際には無料トライアルを利用してアカウントを発行することになります(この無料トライアルに参加することにより、最小構成の仮想サーバが1台1ヵ月利用できます)。

このアカウントで、サーバの作成やネットワーク環境の構成以外にも支払精算の管理もする必要があります。使用するサービスにより、時間課金・月課金など課金のタイミングが違うものもあります。また支払い方法を変更することも可能ですので適切に設定しておきましょう。このアカウントがあれば何でもできてしまうので、くれぐれも紛失したり不用意に公開したりしないようにしましょう。この無料トライアルを使用して、SoftLayerの利用を始めましょう[注1]。

注1) アカウントが不要になった場合には、キチンとサーバを「削除」しておきましょう。

COLUMN キャンペーンをチェック

2014年7月14日に「ロンドンリージョン」が開設されました。その記念にIBMはDCオープンキャンペーンを行っています。登録後1ヵ月有効の$500分が無料になるプロモーションコードを配布しました。

【2nd step】サインアップからの物理サーバの購入方法

アカウントを作成し、サービスを開始する手順は次のようになります。

①キャンペーンを探す
②初期のサーバ購入(アカウントの登録)
③ポータルへのログイン、利用ユーザの登録
④SoftLayerのサービスを使う

最初の「キャンペーン」を探すのは重要です。アカウントを作成したあとでも、まれに掘り出し物がある可能性があります。

【3rd step】サーバ作成手順

実際にSoftLayerのサイトからサーバを作成する手順を示します。利用できるサーバはたくさんありますが、一般的な無料トライアル(https://www.softlayer.com/promo/freeCloud)で利用できるサーバは、仮想サーバの最小スペックモデル(CPU 1.2GHz Core/RAM 1GB/HDD 25GB/Public 5GB Bandwidth、1 IP Address、100Mbps)にあたります。 今回は特別なキャンペーン「ロンドンリージョン開設記念 500$クーポン」を利用してサーバを購入します。これで「Bare Metal Servers」と呼ばれる物理サーバを利用できます。もし、すでにアカウントを持っている場合にも、1回だけクーポンコードが利用できるので試してみましょう。

【注意】キャンペーンは最初の1ヵ月だけ有効です。それ以降課金されては困る方は、利用し終わったら管理ポータルから[Device]-[Device List]-[Action]-[Device Cancel]を選択して、サーバを削除してください。

①www.softLayer.comのキャンペーンバナーから[Order Now and Save $500]をクリックして進む。バナーが出ていない場合は、www.softlayer.com/info/london-hostingにアクセスする

②キャンペーンの説明がされているページから[Order Today]を選択する。コードは「500LN」となり2014年9月30日まで利用可能

③[Bare Metal Servers]の選択画面が出てくるので好きな構成を選び、[Buy now]をクリックする

④DataCenterは間違いなく[LON02]を選択し$500を超えないようにする。[Add Order]をクリックする

⑤右の[Promotion Code]に先ほどのコードを入力して[Apply]をするとProrated TotalとInitial Chargeが$0になる(忘れずに実施するように)。その後画面のクレジットカード情報などを入力してオーダーする

- 全角文字は使用できない
- 氏名、会社名などの情報は、クレジットカードに登録されているものと整合性が取れている必要がある
- 必ず英語で、かつアメリカの住所表記方法に準拠する必要がある
- メールアドレスはSoftLayerからの通知などに利用される
- 「Host Name」「Domain Name」はSoftLayerの管理画面上の表記であるので自由に決める(Domain Nameはプロジェクトやシステム名にしておく程度の意味合いでかまわない)

⑥オーダーを実施すると、登録したメールアドレスに件名「Your SoftLayer Technologies, Inc. Order # 2*** has been received」のメールが届く**

⑦30分程度でアカウント作成のメールが届く。記載されている用意されたマスターユーザ、パスワードを使用してSoftLayer Customer Portal：https://control.softlayer.com(以降管理ポータル)にログインする。場合によっては、確認のための電話があるが、簡単な確認事項なので落ち着いて対応する

COLUMN　物理サーバのオーダーについて

「物理サーバ」をオーダーする際に少し気をつけることがあります。SoftLayerでは要求されたハードウェアが用意できない場合に上位互換のハードウェアが提供されることがあります。たとえば2coreを注文したのに4coreであったりとかメモリが多くなる、ストレージのサイズが大きくなるなどが挙げられます。「ラッキー！」という場合は良いのですが、ライセンスの問題などもあり2coreなければ問題があるという場合などはサポートチケットにてその旨を伝えて調整をしてもらうようにしましょう。裏技的にはアップグレード前提で小さめに頼んでみるのも良いかもしれません。

【4th step】インスタンスにログインしてみよう

管理ポータルに接続します。SoftLayerを利用するには、管理ポータルからすべてを行うことになります。

メニューより[Devices]-[Device List]をクリックします。作成したサーバはここにリストされます。作成中は、サーバ名の左に時計のアイコンが表示され、カーソルを載せるとステータスが確認できます。サーバが作成されれば、時計アイコンが消えます。

作成したサーバにログインするために、接続先のIPアドレスとパスワードを確認してみます。

①[メニュー]→[Device]→[Device List]を選択

②デバイスリストの中から先ほど作成したサーバを選択してクリック

③IPアドレスは画面に、Public IP/Private IPとして表示される

④タブ[Passwords]をクリックして表示される表からrootパスワードを取得する

この、Public IPアドレスに対してログインをしてみましょう、利用したOSがLinux系の場合には、`ssh root@IP_Address`で接続がで

きるようになっています。

【5th step】SoftLayerのクラウド環境の特徴

クラウドサービスはWebホスティングと違い、1台のサーバがインターネット上に存在しているだけではありません。通常のオンプレミスと同様に「サーバ」、「ネットワーク」、「ストレージ」のリソースがクラウド上にも用意されています。先ほど作成した物理サーバは、仮想ネットワークの上に構築されています。

また、ObjectStorageのようなストレージサービスやファイアウォール、負荷分散装置といったネットワークサービスが利用可能です。このようにクラウド上に自分専用のシステム基盤環境があり、その中にサーバなどのリソースをレイアウトしてシステムを作成できるようになっています(**図3**)。

SoftLayerでは、大きく2つのネットワークがあります。「パブリックネットワーク」と「プライベートネットワーク」です。管理ポータルより、[Devices]-[Device List]をクリックし、作成したサーバを確認すると、それぞれのIPアドレスがわかります。「パブリックネットワーク」にはインターネットのトランジットと接続されている「Core Network」と、利用者のインターネット側のネットワークである「Frontend Customer Network」があります。「プライベートネットワーク」には利用者のプライベート側のネットワークの「Backend Customer Network」とSoftLayerの提供するサービス群が配置されている「Backend Service Network」があります。

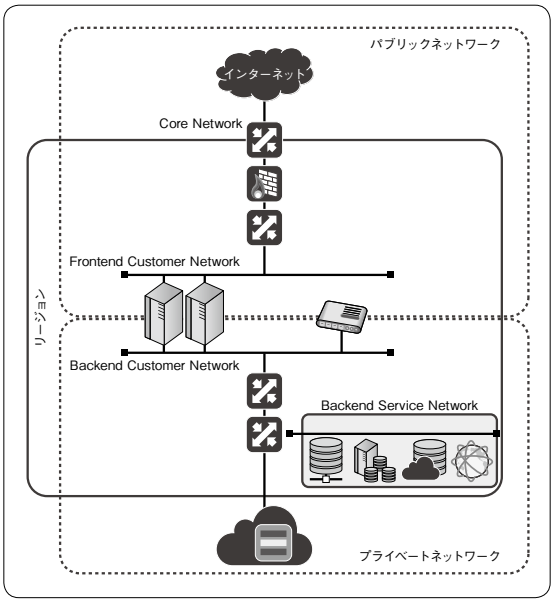
作成されたサーバは、「Frontend Customer Network」と、「Backend Customer Network」にそれぞれ所属しています。また、サーバの種類によっては「Management Network」が接続されていることもあります(これはKVMなどのメンテンナンス時に利用します)。それぞれわかりやすくするために「パブリックネットワーク」「プライベートネットワーク」と記載していきます。

先ほどのサーバの設定を見ていきます(**図4**)。「パブリックネットワーク」「プライベートネットワーク」に対してIPアドレスが振られている状態であることがわかります。

物理サーバの場合には、負荷分散や障害対応のために複数のNICをまとめて(ボンディング)使うようになっています。**bond1(eth4,eth6)**が「パブリックネットワーク」を示して、**bond0(eth5,eth7)**が「プライベートネットワーク」に該当します。

メンテナンス用のKVMなどを利用するための「Management Network」に接続されているIPについてはサーバによっては**eth***として表記される場合もありますし、今回のように表示されない場合もあります。 管理ポータルより、[Devices]-[Device List]をクリックし、作成したサーバをクリックします。「Remote Mgmt」をクリックすると「Management IP」が確認できます。このIPに対して、VPN経由でプライベートネットワークよりブラウザで接続すると、物理サーバのマザーボードへ接続して操作ができます。このあたりをいじることができるのも物理サーバの楽しさかと思います。購入時に構成したRAID構成を変更したい場合には、KVMで接続した後にBIOSのRAIDコントロー

▼図3　SoftLayerの基本的なネットワーク構成



▼図4 IPアドレスの状態確認

```
root@provisiontest1:~# ip a | grep -e ":.eth" -e ":.lo" -e ":.bond" -e "inet "
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default
    inet 127.0.0.1/8 scope host lo
2: eth0: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN group default qlen 1000
3: eth1: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN group default qlen 1000
4: eth2: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN group default qlen 1000
5: eth3: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN group default qlen 1000
6: eth4: <BROADCAST,MULTICAST,SLAVE,UP,LOWER_UP> mtu 1500 qdisc mq master bond0 state UP
group default qlen 1000
7: eth5: <BROADCAST,MULTICAST,SLAVE,UP,LOWER_UP> mtu 1500 qdisc mq master bond1 state UP
group default qlen 1000
8: eth6: <NO-CARRIER,BROADCAST,MULTICAST,SLAVE,UP> mtu 1500 qdisc mq master bond0 state
DOWN group default qlen 1000
9: eth7: <NO-CARRIER,BROADCAST,MULTICAST,SLAVE,UP> mtu 1500 qdisc mq master bond1 state
DOWN group default qlen 1000
10: bond0: <BROADCAST,MULTICAST,MASTER,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group
default
    inet 10.112.34.196/26 brd 10.112.34.255 scope global bond0
11: bond1: <BROADCAST,MULTICAST,MASTER,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group
default
    inet 5.10.107.228/28 brd 5.10.107.239 scope global bond1
```

▼図5 ルーティング構成の確認

```
root@provisiontest1:~# route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
default         5.10.107.225-st 0.0.0.0         UG    0      0        0 bond1
5.10.107.224    *               255.255.255.240 U     0      0        0 bond1
10.0.0.0        10.112.34.193   255.0.0.0       UG    0      0        0 bond0
10.112.34.192   *               255.255.255.192 U     0      0        0 bond0
```

ラ構成で行うことができます。

サーバの購入時に、「Private Network Only」と呼ばれるオプションを選択すると「プライベートネットワーク」には所属しない構成も取ることができます。名前からもわかるように「パブリックネットワーク」は直接インターネットと通信ができるグローバルアドレスが付与されますので、社内システムなどで外部との通信が不要な場合には設定する必要はありません。

図5のように、ルーティングを見てみると、**Private Network (10.0.0.0)**に対してはルーティングが切られており、デフォルトゲートウェイはパブリックネットワーク側であることがわかります。

SoftLayerのネットワークではリージョン、VLAN、サブネットという概念があります。リージョンとは、SoftLayerのサーバが置かれているデータセンターの場所のようなもので、サンノゼ、ダラス、シンガポールや香港といった地域ごとに作られています(早く東京ができてほしいところです。レイテンシが低く快適になります)。リージョンの中には、VLANが複数あります。VLANはプライベートネットワークにもパブリックネットワークにも存在しており、任意に追加できます。そしてVLANを構成するのは複数のサブネットです。サブネットはIPネットワークのサブネットであり、[**10.112.34.192/26**]のように表現されます。実際のサーバはこのサブネットに所属しています。

SoftLayerでは、VLANはブロードキャスト・ドメインであることを表しています。したがって同じVLANに所属しているサブネット間で相互に通信ができます。使うときには同じVLANであるかを気にしておけば問題ありません。

【6th step】セキュリティの設定

パブリックネットワークを利用している場合、すべてのポートがオープンしている状態のため非常に危険ですのでセキュリティの設定をする必要があります。一部の機能については次回で詳しく紹介をしていきたいと思いますが、ここでは最低限実施しておいてほしい内容を書いておきます。

システムの最新パッチの適応

プライベートネットワークだけの環境でもWindows Updateや各種Linuxのパッケージが導入可能なリポジトリが用意されています。導入時にリポジトリの参照先がSoftLayerのものに変更されていますので、とくに意識することなく各種OSのアップデートを利用できます。

SSL-VPNの設定

SSL-VPN接続では、プライベートネットワークに接続されます。この機能を利用して普段の作業についてはプライベートネットワーク側から利用することを勧めています。SSL-VPNの設定はポータルから行うことができます。簡単に流れを記載しておきます。また、ここでは記載しませんが子ユーザごとに設定ができます。

①メニューの[Account]から[VPN Access]をクリックする
②利用したいアカウントの[VPN Access]をクリックする
③ダイアログで[VPN Type]を選択する画面がでるので[SSL & PTP]を選択し、右下の[Save]をクリックする
④SSL-VPNを利用するパスワードを決める必要があるため[Account]→[User]から該当のユーザをクリックして選択する。[VPN Password]に適切な値を入れて[EditUser]をクリックする
⑤SSL-VPNクライアントは、Javaプラグインで動作する。Internet ExplorerかFirefoxを利用して管理ポータルメニューの[Support]→[Help]→[SSL-VPN login]をクリックしてログイン画面を表示する。そのあと、さきほどのパスワードでログインが可能になる

Firewallの適応

サーバを保護するためのFirewallは複数のサービスが用意されています。ここではまだ何も購入していませんのでOS上の機能で保護をしておきましょう。Linuxの場合にはiptablesなどを利用します。設定を簡単にするためにオプションでAdvanced Policy Firewall(APF)を利用することも可能です。上記のSSL-VPNを利用するように構成したうえでパブリックネットワークのインターフェースに対して必要最低限のサービスポート以外はすべて閉じておきましょう。SoftLayerで提供されているFirewallについては次回説明をしたいと思います。

バックアップ

いざというときに備えてバックアップを取得することは重要です。物理サーバを利用している場合には2つの選択肢があります。1つはRHELやWindowsサーバを選択している場合で、「Flex Image Backup」が利用可能です。これは物理と仮想を行き来することができるバックアップのしくみです。もう1つはそのほかのOSの場合に利用することになるEVaultバックアップサービスになります。仮想サーバの場合には、「Image Template」機能でバックアップが実施可能です。いずれの場合においてもシステムを維持運用するためにはバックアップの設計はしっかりしておきましょう。

【7th step】アカウント管理

SoftLayerのアカウント管理の特徴を紹介します。最初に発行されたユーザは「マスターユーザ」であり、名称の先頭にSLが付きます(例：SL012345678)。このSLユーザには、氏名や

住所、クレジットなどの請求情報が紐づきます。

そして1つの企業に対して基本的には1SL環境です。正確には、1つのクレジットカードあたり1つのSL環境です。したがってコーポレートカードなどを利用して復数の環境が欲しい場合は、別途SoftLayer側の調整が必要になります(ご注意!)。

この最初のアカウントを以降「マスターユーザ」と呼びます。これが管理者の権限を持つユーザです。このマスターユーザに紐づいた子ユーザをいくつでも作れます。実際にはこの子ユーザで運用します(以降、子ユーザは「ユーザ」とします。ユーザには任意の名前を付与できます)。

ユーザに関しては用途に応じてさまざまな権限をつけられますので、役割に応じて付与すると良いでしょう。またユーザ単位に公開する「サーバ」なども選択できますので、プロジェクト単位での管理面にも利用できます。API経由でSoftLayerの環境を利用する際にも、ユーザごとのAPIキーが発行されるので権限を制限して利用してください。

わからないことはサポートを利用

SoftLayerのサポートは残念ながら(?)英語だけのサービスですが、契約をすると無料で利用できます。このサポートでは、障害の対応や一般的な利用方法(仕様の確認)、また料金についてなど、さまざまな内容を問い合わせできます。しかも柔軟に回答してくれるので非常に便利です。使用していて、障害なのか仕様なのかわからない場合、どんどん「サポート」に質問したほうが早く原因がわかります。追加コストがかからないことはとても魅力的ですが、すべて英語ですので少しハードルが高いかもしれません。ただ、筆者の英語力は低いのですが、判別しにくいエラーメッセージの羅列を送っても親切に対応してくれます(日本語での支援が必要な場合には、日本SoftLayerユーザグループ:http://jslug.jp のメーリングリストに質問すれば、誰かが答えてくれるかもしれませんので、このメーリングリストに参加することをお勧めします。参加は、users-join@jslug.jp に空メールを送信し、返信メールのリンクから承認するだけです)。

サポートは「オンラインチャット」「サポートチケット」があります。チャットでは、すぐに質問できる内容やオーダー時に悩んだ点などを確認できます。ちなみに会話を長時間放置していても大丈夫ですので、翻訳する時間が十分あります[注2]。まれにチャットで相談していると、SoftLayerの運用側の方が代わりに操作してくれることもあります。チャットは終了時に履歴を保存できるので利用すると便利です[注3]。

実際にシステムを利用していくうえでは、チケットでの問い合わせが頻繁に行われます。これは掲示板のような画面で表示されます。問い合わせの履歴が見られ、エビデンスとしても便利なのでサポートチケットでの問い合わせを行い、補足としてオンラインチャットを使用する方法が良いでしょう。

障害が発生した際に、サポートは強力に利用者の力になってくれます。必要に応じて実際にサーバの中に入り確認をしてくれたりします。筆者もこれまで何回もサポート側で対応をしてもらって助かったことがあります。

まとめ

SoftLayerの物理サーバのインストール例と、その使い方の簡単な解説を行いました。次号では、ネットワークの構成例やNFVの利用の紹介、そしてサーバ構築時のTipsや簡単なAPIの使い方など、少し応用的な事例を紹介していきます。SD

注2) 海外にあるSoftLayerのサポート担当者と質問、依頼をやりとりすることになるため、書き込む時間によっては対応が遅くなるケースもあります。

注3) 質問をクローズする際、原則として質問者側がクローズします。なお、5日間更新がない場合はサポート担当者からクローズする旨の連絡が来るため、延長したい場合には質問者側で延長する旨を連絡する必要があります。

実力検証

NICをまとめて高速通信！

前編 リンク・アグリゲーションってなに？

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） ㈲オングス　代表取締役

ポートをまとめるチーミングとは ～よく使われるのはフェイルオーバー

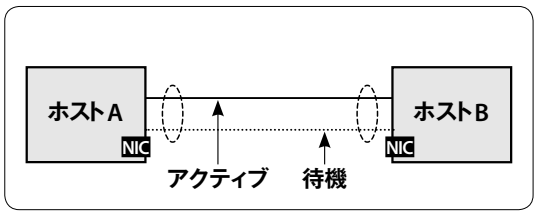
基幹システムの開発などにおいては、複数のNICのポートをまとめあげて1本の論理チャンネルとして利用することがあります。これはチーミング（teaming）と呼ばれ[注1]、おもにフェイルオーバーの目的で使われています。ホスト間を複数のネットワークケーブルで接続することで、ホストとホストを結んでいるケーブルやNICのポート、またはそのチップなどに故障が発生した場合でも、残りのポートとケーブルを使って通信を維持しようというわけです。

たとえば**図1**のように、ホストAとホストBをフェイルオーバー目的のチーミングで接続したとします。2本のLANケーブルをチーミングして1本の論理チャンネルのように扱います。この場合、1本がメイン回線（アクティブ）、もう1本が予備回線（待機）となります。普段使われるのはメイン回線です。

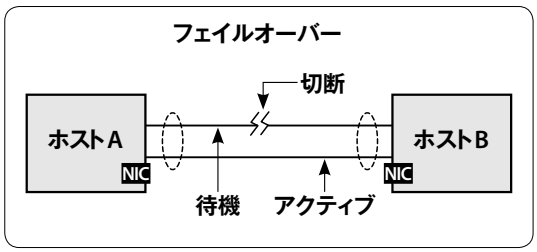
この状態でメインの回線を切断したり、メイン回線側のポートが故障するなどして通信が不可能になると、通信はもう1本の予備回線を使うように自動的に切り替わります（**図2**）。この処理はカーネルおよびドライバが担当しますので、通信を実施しているソフトウェアは何もする必要がありません。チーミング技術がよく使われるのはこの用途です。故障が発生してシステムが使用不可能になると、被る損害が大きい場合などにこうした構成が用いられます。

本稿では前後編に分け、この技術をフェイルオーバーの目的ではなく、「通信速度の向上」のために利用するというちょっと変わった方法を紹介します。前編となる今回は、チーミングを利用して通信速度を引き上げるその必要性や、どういった特徴をもった通信を実現できるのか

▼図1　フェイルオーバー目的のチーミング構成



▼図2　メイン回線が通信不能になった場合の動作



注1）チーミングは、Linuxでは使われているモジュールの名前から「ボンディング」と呼ばれることがあります。また、FreeBSDではlagg(4)インターフェースが使われることから「ラグ」といった呼ばれ方をされます。

を解説します。後編となる次回は、実際に8本のLANケーブルをチーミング(リンク・アグリゲーション)して、通信性能がどのように変化するのかを紹介する予定です。

高速通信の実現に使えるチーミング ～リンク・アグリゲーションとは

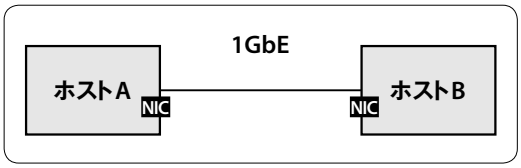
現在、エンタープライズシーンでもコンシューマシーンでも、もっとも広く普及している有線ネットワークは「1GbE」で構成されたネットワークです(**図3**)。このネットワークの通信速度では不十分なケースを考えます。MTU(Maximum Transmisson Unit)の値を変更することで多少の高速化は実現できますが、大幅な向上は難しいところがあります。簡単に思いつく方法は、1GbEのネットワークを10GbEのネットワーク、40GbEのネットワークといったより高速なネットワークへ置き換えることです(**図4**)。

これはシンプルでわかりやすい方法です。実現できるならもっともよい選択肢の1つといえます。しかし、いくつかの理由でこうした変更が困難、またはこれだけでは不十分な場合があります。たとえば次のようなケースです。

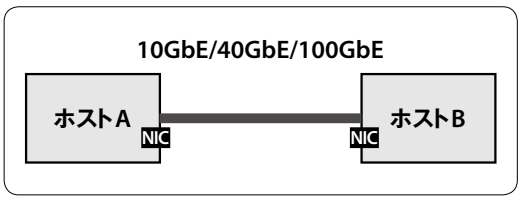
- 10GbEや40GbEに対応したネットワークアダプタやスイッチングハブといった製品の価格は下がってきてはいるが、それでも現行の1GbEの関連機材と比較するとかなり高価。予算的に導入することができない
- 10GbEや40GbEに対応したデバイスに置き換えることは予算的には十分可能だが、それでもまだ速度が足りない

既存のネットワークインフラストラクチャをすべて10GbEや40GbEに置き換えようとすれば、規模によってはかなりの予算が必要になります。1GbEの機材の価格帯から考えると、その必要性がかなり高い場合を除いてこの選択肢を取るのは難しいでしょう。10GbEや40GbEの関連機材の価格帯が現在の1GbEの価格帯まで下がってくるまで、1GbEの技術の枠の中で高速化を実現したいと考えます。この場合、1GbEを何本か組み合わせて通信の高速化を狙います。こうしたケースで使える技術がチーミングの中でもとくに「リンク・アグリゲーション」と呼ばれる技術です。

▼図3　1本の1GbEで接続されたホスト



▼図4　高速通信したいなら、もっと高速なハードウェアへ変更する

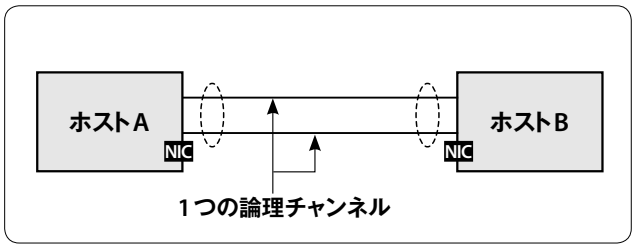


もう1つは10GbEや40GbEに変更しても、それでもまだ通信速度が足りないといったケースです。こうしたケースでは、10GbEや40GbEといったネットワークに切り替えたうえで、さらに高速化の手段を取る必要があります。こうしたケースでもリンク・アグリゲーションを使って通信速度の引き上げを実現することができます。

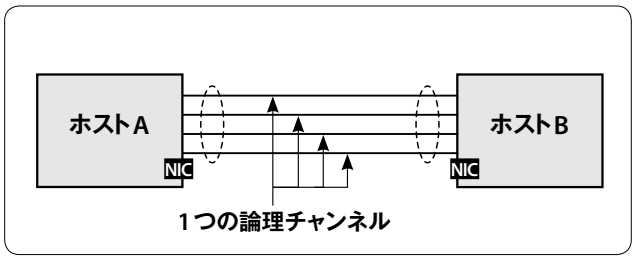
リンク・アグリゲーションでは複数のポートをまとめて1つの論理チャンネルのように扱います(**図5、6**)。オペレーティングシステムのレイヤから見ると、この処理はカーネルやリンク・アグリゲーションドライバが担当しますので、ユーザランドで動作しているソフトウェアには一切の変更が必要ありません。必要に応じてまとめ上げる本数を増やしたり、減らしたりといったことも、ソフトウェア側は変更することなく実現できます。

複数のポートをまとめて1つの論理的なチャンネルのように見せかけるためには、ポートやネットワークデバイスの間でそうした情報をや

▼図5　チーミング(リンク・アグリゲーション)
例1：2本を1つの論理チャンネルへ



▼図6　チーミング(リンク・アグリゲーション)
例2：4本を1つの論理チャンネルへ



りとりする必要があります。そのやり取りを規定したものが「IEEE 802.1AX Link Aggregation Control Protocol(LACP)」です。メーカーが独自に開発したプロトコルも存在しています。LACPを使い、どの物理ポートが1つの論理チャンネルとして設定されているかをネットワークデバイスが知ることで、複数の回線を利用した通信が実現されています。LACPという言葉が使われている場合、ここでの説明に使っているリンク・アグリゲーションのことだと考えてください。

1. 大量の観測データを処理する必要があるケース

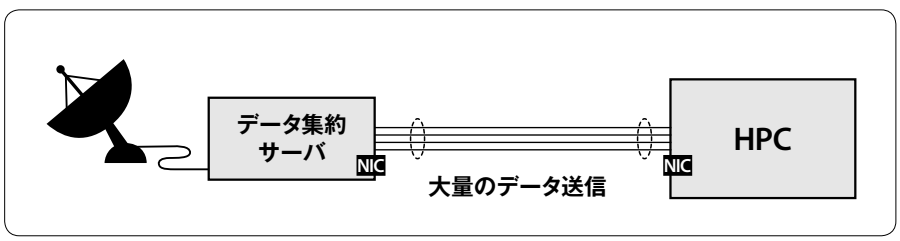
10GbEや40GbEでは通信速度が足りないといった顕著なケースを見てみましょう。これは天文学関連の観測データの処理などで見られるケースです。アンテナで受信される大量のデータを処理するためには、スーパーコンピュータに分類される高速なマシンで処理をこなす必要があります。こうしたマシンはアンテナの近くに設置することはできませんので、アンテナ付近からスーパーコンピュータまでデータを高速に送信する必要があります。

こうしたケースで活用できる技術の1つがリンク・アグリゲーションです(**図7**)。アンテナからデータを吸い出すデータ集約サーバは一切のデータをディスクに書き込みません。大量のメモリを搭載しておいて、データはいったんメモリ上に保持します。この状態からそのままリンク・アグリゲーションで構築されたネットワーク経由でスーパーコンピュータへデータを送ります。スーパーコンピュータ側では送られてきたデータを処理し、分析や集計などを実施したあとの必要なデータだけをストレージ領域に書き込みます(そうしないとデータ量が膨大過ぎるためです)。

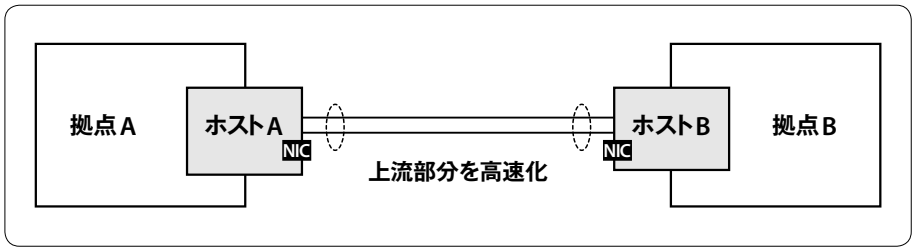
2. 上流だけは太く接続したいケース

上流部分だけ通信速度が高速にできればよいといった場合にもリンク・アグリゲーションを活用できます。たとえば拠点Aと拠点Bといっ

▼図7　リンク・アグリゲーションを利用してデータの高速転送



▼図8　上流部分だけといったように一部の区間だけを高速化したいケース



た異なるネットワークを接続する部分だけ高速にできればよい、といったケースです(図8)。10GbEの機材を購入しなくても、既存の1GbEのネットワークにいくらかの機材を追加するだけで対応できるので、コストを抑えながら高速通信も実現する必要があるといった場合に活用できます。

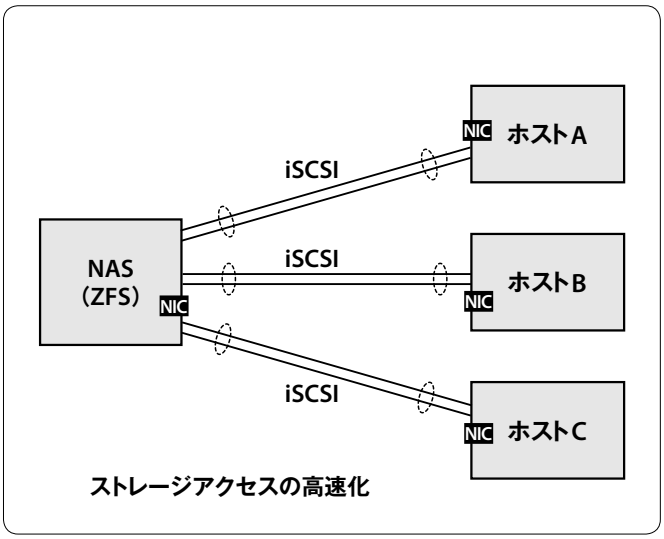
10GbEのネットワークアダプタもだいぶ値段がこなれてきていますので、こうしたケースでは10GbEのネットワークアダプタを購入して直接結線したほうが早いケースも多いかもしれません。ただし、10GbE以上のネットワークアダプタはコンシューマ市場にはあまり流通していないので、入手に時間がかかるという面はあります。

3. ストレージとの接続だけ太くしたいケース

ストレージとの接続部分だけ太くしたいといったケースもあります。ストレージはZFSで構築されたストレージデバイスに集約してあり、作業端末からiSCSI経由でディスクをマウントしたり、NFS経由でデータにアクセスするといった場合です。こういった場合、NASと各ホストをリンク・アグリゲーションで接続して通信の高速化をはかります(図9)。

システムのブートまで含めてストレージシステムからデータを持ってくるようなディスクレスシステムを構築することもできます。その場合、通信部分は高速になるならそれに越したことはありません

▼図9　ストレージとの通信を太くしたい



リンク・アグリゲーションの通信の特徴を知ろう！

ここまでリンク・アグリゲーションを見てきて、まるで束ねる本数を増やせば増やしただけ通信速度が高速化するような書き方をしてきましたが、実際にはそのような動作はしません。LACPで構築されるのはあくまで「1つの論理チャンネル」であって、回線1本1本の通信速度は変わらないからです。

つまり、1コネクション／1ストリームの通信速度の最大値は1本でも8本でも同じです。1本分の速度までしかでません(図10)。

しかし、同時に複数のストリームを使った場合、すべてを総合した通信速度は引き上がりま

す(**図11**)。物理的に複数の回線を使ってデータの転送が実施されるからです。理想的な状況になれば、8本分をまとめた回線では、8ストリームで8倍の速度が期待できます。

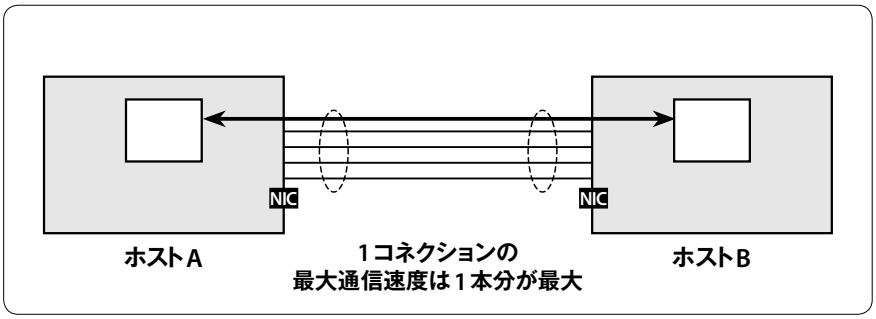
これは常に複数のコネクションが発生しているような用途で、性能の向上が期待できることを意味しています。単一のコネクション／ストリームで大量のデータを流すような用途には向いていません。その場合、複数のストリームを利用するようにソフトウェア側を書き換える必要があります。

また、リンク・アグリゲーションで性能の向上が期待できるかどうかは、カーネルの内部の実装にも依存していることに注意してください。ネットワークスタックが複数のポートや複数のコアに対してスケールしないカーネルでは、いくらリンク・アグリゲーションを実施しても通信性能の向上は期待できません。同じカーネルでもバージョンが古いと性能がでないなどの違いもありますので、実際に性能がでるかどうかは実機で対象のオペレーティングシステムを導入して実験してみる必要があります。

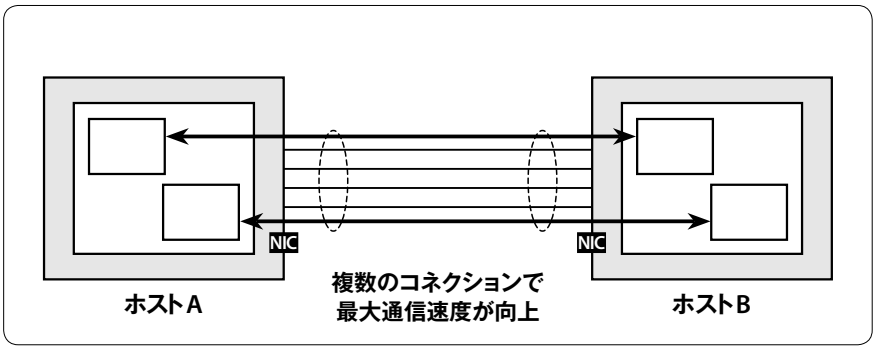
次は実験してみよう！

今回はチーミング(リンク・アグリゲーション)で通信速度を引き上げる必要があるシーンやケース、実際にどういったタイプの通信で速度の向上が実現できるのかなどを説明してきました。次回は実際に8ポートをチーミングして、どのように通信性能が変化するかを紹介します。また、フェイルオーバー目的での利用や、ロードバランス、ラウンドロビンなどで利用した場合の性能、MTUの値を変化させて通信速度を向上させる方法なども紹介します。SD

▼図10　1ストリームあたりの最大通信速度は同じ



▼図11　複数のストリーム全体で通信速度の向上が期待できる



BOOK no.1 Amazon Web Services 基礎からのネットワーク&サーバー構築

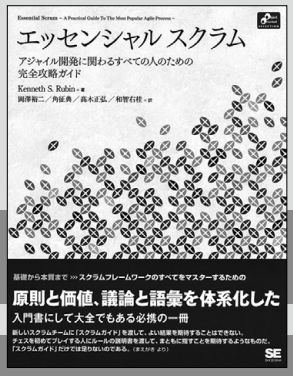
玉川 憲、片山 暁雄、今井 雄太 【著】
B5変形判、216ページ／価格＝2,700円＋税／発行＝日経BP社
ISBN＝978-4-8222-6296-9

インフラに詳しくない人が「自分でネットワークやサーバを構築できるようになる」ことを目指して書かれた本。Amazon Web Services（以下AWS）上でWordPressを使ったblogシステムを完成させることを目標に、リージョンの選択からVPC（Virtual Private Cloud）の作成、Webサーバ、DBサーバ、NATサーバの設定まで、各ステップを、AWSの設定画面を見せながら具体的に解説している。

AWS上ですべてを行うことで、ハードウェアを購入せずとも、インフラの構築について、実際に手を動かしながら学習できる。TCP/IPやWebサーバの基礎知識も紹介されており、初心者にも易しくわかる本となっている。

BOOK no.2 エッセンシャルスクラム
アジャイル開発に関わるすべての人のための完全攻略ガイド

Kenneth S.Rubin 【著】／岡澤 裕二、角 征典、髙木 正弘、和智 右桂 【訳】
B5変形判、448ページ／価格＝3,800円＋税／発行＝翔泳社
ISBN＝978-4-7981-3050-7

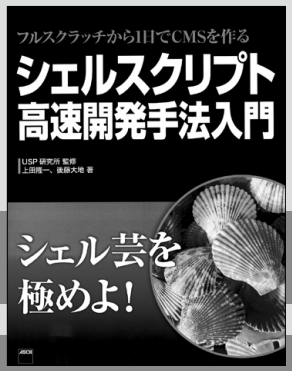


「スクラム」とは、アジャイルの中でもチーム開発に重きを置いた開発手法の1つ。本書ではそのスクラムについて、用語の説明からプロジェクトの流れまで、網羅的に解説している。プロダクトオーナー、スクラムマスタ、開発メンバ、マネージャー、それぞれの立場の人に向けて書かれた章があり、開発に関わるあらゆる人が読者対象だ。昨今徐々に広まりつつあるテスト駆動開発、継続的インテグレーションをスクラムの流れの中で学ぶこともできる。

スクラムについて膨大で緻密な情報が紹介されており、自分が今いる環境・立場に合わせて各章・各ステップから必要な知識を選び取り、実務に活かすのがいいだろう。

BOOK no.3 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

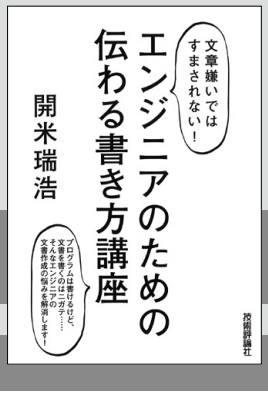
上田 隆一、後藤大地 【著】／USP研究所 【監修】
B5変形判、280ページ／価格＝2,600円＋税／発行＝KADOKAWA
ISBN＝978-4-04-866068-6



著者が所属するUSP友の会のWebサイトは、著者自らがシェルスクリプトで開発したもの。本書はその開発ノウハウを解説するのだが、開発を実況中継するかのような展開で話が進む。基本コマンドの説明などはなく、最初からサイトの構造やコンテンツの開発を順を追って説明していく。コマンドの説明などは新しいものが登場する都度、行われる。なぜそのコマンドなのか、なぜそのような設計なのかという理由も、開発を進める中で説明される。そのような展開のためか、必要な部分から作り、部品を積み上げるようにサイトを完成させていく様子がよくわかる。その開発の軽快さ、スピード感に注目して一読することを勧める。

BOOK no.4 文章嫌いではすまされない! エンジニアのための伝わる書き方講座

開米 瑞浩 【著】
A5判、200ページ／価格＝1,980円＋税／発行＝技術評論社
ISBN＝978-4-7741-6576-9



ITエンジニアの皆さんが説明しなければいけない事柄は、用語や知識の前提がある程度必要なため、非エンジニアの方には「わかりにくい」と感じてしまわれがち。本書ではわかりやすい文書にするためのコツを、例文を分解・再構築することで具体的に解説し、考え方や手法を身につけるための方法を紹介している。

・書く前が重要。6つのポイントを押さえる
・図解＋文章でわかりやすい「文書」になる
・構造を図解するための典型的な型を知る
・誰が読むのか。文書の役割を考える

ITエンジニアに馴染みのある例文から、求める文書に近いものを見つけて真似ることからはじめてみてほしい。

特別企画

オーケストレーションツール Serf・Consul入門 Serf編

クリエーションライン株式会社 Technology Evangelist 前佛 雅人(ぜんぶつ まさひと)
Twitter @zembutsu Web http://pocketstudio.jp/log3/

開発や運用の流れを最適化しようという機運が高まりつつある中で、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)」と、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)」が登場しました。本企画では2回にわたってそれぞれを解説します。今回は「Serf」について解説します。

オーケストレーションツール

ここ数年、クラウドコンピューティングの広がりにより、動的にサーバリソースやシステム規模の変更を容易に行う環境が整ってきました。海外のみならず、日本国内市場においても、クラウド事業者の参入や、それらを基盤として提供するサービス提供者も増えつつあります。

その結果、開発や運用を問わず、業務の流れが変化しつつあります。たとえば、構成管理ツールであるChefやPuppetを使い、システムの構成を自動化することが顕著です。構成管理を通して、コードとしてインフラを管理する考え(Infrastructure as a Code)も一般化しました。ほかにはCapistranoを使ったコンテンツのデプロイや、ServerSpecやJenkinsなどのように、自動テストやCI(Continuous Integration:継続的インテグレーション)ツールを利用する手法も広まりつつあります。さらにImmutable Infrastructureという概念も登場し、変化する環境に応じ、開発や運用の流れを最適化しようという機運が高まりつつあります。

このような自動化を推進する流れの中で、SerfやConsulは登場しました。どちらもVagrantを開発しているHashicorp社によるもので、オープンソースとして公開・開発が進められています。Serf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ツールとして活躍します。Consul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しくみを提供します。

本稿を通しSerfとConsulに対する理解を深め、開発・運用の現場で活用するためのヒントなり、きっかけになればと思います。

Serf

Serfとは

Serfは、クラスタのメンバ管理や、障害検知の機能を備えた、オーケストレーションを行うツールです。オーケストレーションには諸説ありますが、ここではクラスタ全体に対して、一斉に処理を行うことと定義します。たとえば、管弦楽のオーケストラで、指揮者が振るタイミングで、曲調を同時にコントロールするようなイメージです。一斉に行う対象が、それぞれの楽器ではなく、Serfの場合はSerfエージェントの入ったノードです。

このSerfを使い始めるのは、非常に簡単です。Serfはバイナリ1個で動作し、CLIとエージェントを兼用した"serf"コマンドを実行するだけで、クラスタを形成します。常駐メモリも少ないため、システムに対する影響を深く考慮する必要はありません。また、コマンドラインでの

操作が比較的わかりやすいです。

公式サイト[注1]には、次のような使用例が書かれています(抜粋)。

・Webサーバとロードバランサの自動連携
・memcachedやRedisクラスタのノード管理
・Serfを起点としてデプロイを行うシステム
・クラスタ状態に応じてDNSレコードの更新
・単純なコマンド実行による調査

機能

Serfにはオーケストレーションを行うための、おもに次の3つの機能があります。

♪ 1 メンバ管理

Serfエージェントは、相互に通信を行うクラスタを構成します。エージェント起動時に接続先を明示することで、クラスタに参加できます。いったんクラスタが形成されると、ゴシッププロトコル(後述)に基づき、相互に死活状態を監視し、メンバの稼働状況を監視できるようになります。また、新規メンバの参加や離脱は、リアルタイムにクラスタ内で情報共有されます。この情報のことを、Serfでは「イベント」と呼びます。

♪ 2 障害検知と復旧

あるノード上のSerfエージェント間で通信ができなくなると、対象のノードを障害が発生したとみなします。また、クラスタ全体に対して障害発生を通知し、対象ノードで問題が発生したという情報が伝わります。通信ができなくなっても、一定期間は既存ノードが定期的に接続を試みます。もしノードとの通信が復旧すると、エージェント起動時にとくに明示しなくても、自動的にクラスタに復旧する特長があります

なお、Serf公式サイトでは、これらをゾンビの例えで紹介しています。ある街の住人が、互いに人間かどうかを確認するような世界において、ゾンビが見つかったら、住人全員に対して「ゾンビがここにいるぞ!」と伝えるようなしくみです。

♪ 3 イベントとイベントハンドラ

イベントはSerfクラスタ上での何かの変化のことです。たとえば上記のメンバ参加や障害発生がイベントです。イベントは2種類あります。

・Serfのメンバ管理イベント(システムによる自動発行)
・ユーザによる任意イベント(随時に発行)

それぞれのイベント発生のタイミングで、任意のコマンドやスクリプトを実行できます。たとえば、ノード追加時に構成管理用のコマンドを実行したり、任意イベント発生のタイミングで、プロセスの再起動ができます。このしくみがイベントハンドラです。

アーキテクチャ

♪ 非中央集権型のクラスタ

Serfクラスタは、Serfエージェント間で構成されます。本体の**"serf"**バイナリは2つの役割を兼ね備えます。

・クラスタを維持するエージェント
・コマンドラインインターフェース(CLI)

Serfクラスタは、中心となるサーバが存在しない、非中央集権型です(**図1**)。Serfは一見するとクライアント/サーバ型に見えますが、クラスタ内のSerfエージェントは互いに通信し(TCPおよびUDPのPort 7946を使用)、全体として1つのクラスタを形成します。

それでは、クライアントの問い合わせ先は、どこでしょうか。答えは、クラスタ上のserfエージェントが動作しているノードであれば、どこでもかまいません。ノードに対して問い合わせをすると、結果が得られます。クラスタ内では情報が同期されているため、どこに問い合わせをしても、常に同じ結果を得られます。

なお、クライアントはクラスタに参加する必要はありません。クライアントは、標準のCLIを使

注1) http://www.serfdom.io/

うか、MsgPack over TCP(Port 7373)を用いてクラスタに接続し、RPCプロトコルで通信します。

♪ ゴシッププロトコル

このようなクラスタを構成できるのは、Serfが採用しているゴシッププロトコル[注2]のお陰です。ランダムに相互の死活監視を行い、クラスタを構成・維持します。また、後述するSerfのイベント情報を、クラスタ全体で瞬時に同期するしくみも提供します。

開発体制

開発主体はVagrantやPackerを開発しているHashicorp社です。コードにコミットしているのは、Mitchell Hashimoto氏ら、おもに社員の方ですが、開発体制はオープンです。GitHub[注3]やIRC[注4]、メーリングリスト[注5]を通して議論が行われ、ときおり新しい機能がコミュニティを通して取り込まれています。また、Go言語で書かれたすべてのソースコードは公開されており、オープンソース[注6]として利用できます。

ほかのツールとの比較

オーケストレーションを行うツールは、ほかにもありますが、Serfがほかのツールと決定的に違うのは、非常に簡単にクラスタを構成でき、手軽に使えるという点です。Serfは、1つのバイナリファイルを置き、いくつかのコマンドを実行するだけで、簡単にクラスタを構成できます。同時にコマンドを実行させるために必要な言語は何でもかまいません。シェルスクリプトでもかまいませんし、RubyやPerlでも自分なり現場なりで必要な言語を使って処理ができます。

また、多くのツールとは競合するものではなく、併用することもできます。たとえば、構成管理ツールとの連携として、設定投入のためのトリガとして機能させることができます。

環境構築

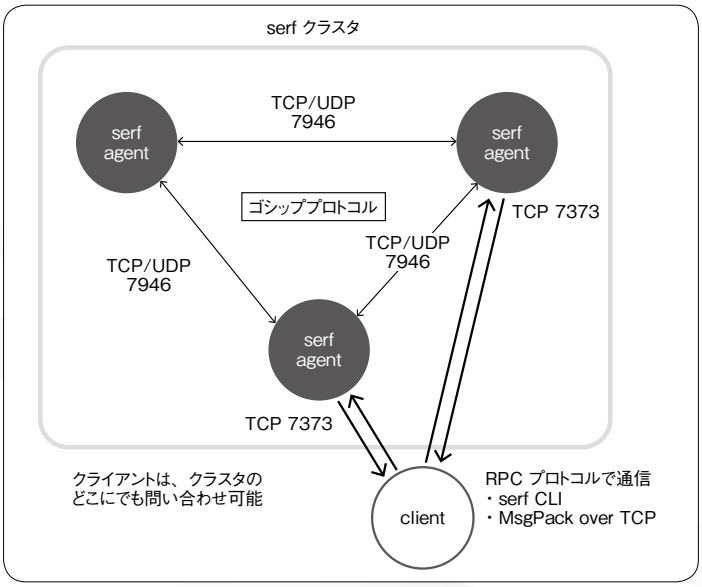
セットアップ

♪ 動作環境

Serfは幅広いOSに対応したバイナリが配付されています。バイナリファイルを1つ置くだけで動作するため、追加でほかのアプリケーションをセットアップする必要がありません。また、ソースコードはGitHubで公開されていますので、それをもとに構築することもできます。その場合は、各OS環境上で、Go言語の開発環境を用意してください。

注2) 正確には"SWIM：Scalable Weakly-consistent Infection-style Process Group membership Protocol"論文をもとに、伝播速度などの改良を加えたものです。計算では、100ノードに対して約2秒で情報が伝わります。
注3) https://github.com/hashicorp/serf/
注4) #serfdom(freenode.net)
注5) Serf Google Group: https://groups.google.com/group/serfdom/
注6) Mozilla Public license, version 2.0

▼図1 アーキテクチャ



♪ 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7から取得できます。現在のバージョン0.6.3(原稿執筆時)では、Linuxのほか、Mac OS X、Windows、FreeBSD、OpenBSDに対応しています。**図2**はwgetを用いた展開例です。Serfは頻繁にバージョンアップしていますので、最新版はダウンロードのページを確認してください。

ファイルを設置後は、"**serf -v**"と入力することでバージョンが表示されます(**図3**)。

ここで注目するのはバージョン番号"**Protocol:4**"です。Serfはバージョンが変わっても、2世代前までは互換性があります注8。しかし、一部機能に相違が出てくる可能性があります。これまでにあった大きな変更点は、"**role**"機能が"**tag**"に置き換えられたことです。常に最新の機能を使いたい場合、将来的なバージョンアップを計画してください。

なお、この例では"**/usr/bin/serf**"にserfを置きましたが、環境内のパスの通るところであれば、どこに置いてもかまいません。

初めてのSerfクラスタ構成

エージェントの起動

Serfでクラスタを構成するためには、エージェントを起動する必要があります。Serfの引数に"**agent**"を付けると、それだけで起動できます(**図4**)。ここでは動作確認のため、バックグラウンドでserf agentを起動しましょう。

これでエージェントが起動しました。バックグラウンドで動作させると、デバッグ用のログが画面に表示されます。このログは"**serf monitor**"と実行しても、同じ内容を確認できます。

次はメンバ情報を確認します。"**serf members**"とコマンドを実行すると、ホスト名やIPアドレスに加え、クラスタがどのような状況になっているかが表示されます。

```
$ serf members
node1  192.168.39.1:7946  alive
```

ここでは、自分自身のホスト情報が表示されます。ノードの状態は"**alive**"であり、正常に稼働中です。

クラスタの形成

次は、2台のサーバ上で動くSerfで、クラスタを構成します(**図5**)。ここでは、次のような2台のサーバが動いているものと想定します。

・node1(192.168.39.1)
・node2(192.168.39.2)

注7) http://www.serfdom.io/downloads.html
注8) http://www.serfdom.io/docs/compatibility.html

▼図2　wgetを用いた展開例

```
$ wget -O 0.6.3_linux_amd64.zip https://dl.bintray.com/mitchellh/serf/0.6.3_linux_amd64.zip
$ unzip ./0.6.3_linux_amd64.zip
# cp ./serf /usr/bin/serf
```

▼図3　Serfのバージョンを表示する

```
$ serf -v
Serf v0.6.3
Agent Protocol: 4 (Understands back to: 2)
```

▼図4　エージェントの起動

```
$ serf agent &
==> Starting Serf agent...
==> Starting Serf agent RPC...
==> Serf agent running!
         Node name: 'node1'
         Bind addr: '0.0.0.0:7946'
          RPC addr: '127.0.0.1:7373'
         Encrypted: false
          Snapshot: false
           Profile: lan

==> Log data will now stream in as it occurs:

    2014/07/06 21:17:43 [INFO] agent: Serf agent starting
    2014/07/06 21:17:43 [INFO] serf: EventMemberJoin: node1 192.168.39.1
```

サーバに複数のインターフェースが付いている場合は、エージェント起動時に明示しなくてはいけない場合があります。もし想定しているインターフェースが自動で使用されない場合、“-iface”タグを付ける必要があります。

・例：eth1を使用する場合

```
$ serf agent -iface=eth1
```

♪ serf join

次は、node1からnode2に接続を試みましょう。2台のサーバをつなぐには、“serf join”コマンドを使います。node1でコマンドを実行します。

```
$ serf join 192.168.39.1
```

コマンド実行後、ただちに双方がお互いをクラスタのメンバとして認識します。“serf members”コマンドを実行すると、どちらのサーバでも同様の結果が表示されます。

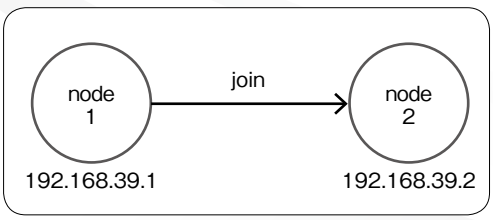
```
$ serf members
node1  192.168.39.1:7946  alive
node2  192.168.39.2:7946  alive
```

今回はnode1からnode2に接続を試みましたが、逆の接続もできます。node2上でnode1に対してjoinしても、結果的に2台で同一のクラスタを構成します。

♪ エージェント起動時にjoin

サーバ台数が増えてくると、その都度“serf join”を行うのはたいへんです。エージェント起動時に“-join”を使うと、自動的に指定したノードに対して接続を試みます。

▼図5　Serfクラスタのイメージ図




```
$ serf agent -join=node2
```

join先のサーバが不明な場合や、“-discover”オプションを使う方法があります。

```
$ serf agent -discover=serf
```

この例は、同一ネットワークの“serf”という名前のクラスタに対して、自動的にjoinするものです。あえてjoin先のホストを明示する必要はありません。ただし、この便利な方法が使えるのはマルチキャストDNS（mDNS）が利用可能なネットワークだけです。環境によっては通信が制限され、利用できないことがあるので注意してください。

クラスタ上でイベントを発行

Serfのクラスタを形成したあとは、すべてのノードで一斉にイベントを処理できるようになります。ここでは“test”という名前のイベントが発生すると、同時に実行されることを確認します。イベントは“serf event”コマンドを使って発行できます（図6）。

コマンドを実行すると、ノードが稼働している両方のサーバで“Received event”という情報が表示されます。これはSerfの持つイベントの機能のうち、ユーザが任意に発行できるものです。このほか、Serfシステムが自動的に使用する、“member-*”系のイベントがあります。

エージェントの停止

エージェントを停止するにはCtrl＋Cキーを押して処理を中断します。あるいは“kill”コマンドを使って、停止させることもできます。

イベントとイベントハンドラ

イベント

イベントは、Serfで何か処理を行う際のトリ

ガとなるものです。イベントが発生するタイミングで、都度、あらかじめ指定したコマンドを実行させることができます。この一連の処理のことをイベントハンドラと呼びます(**図7**)。また、Serfの特長として、特定のSerfノードで発生したイベントは、瞬時にクラスタ全体に伝わります。つまりSerfがあれば、クラスタ内の何らかのイベント発生をトリガとして、クラスタ全体に対して一斉に処理を行うことができます。このイベントハンドラのしくみこそが、Serfがオーケストレーションツールであると言えるゆえんなのです。

♪ イベントの種類

それでは、イベントの詳細を見ていきましょう。イベントには、システムが自動的に発行するものと、ユーザが任意に発行できるものと、2種類があります。

- メンバ管理：“member-*”という名称で、システムが自動発行するクラスタ管理用のイベント
- ユーザイベント：“event”と“query”はユーザが任意タイミングで発行可能なイベント

♪ メンバ管理

メンバ管理系のイベントは、クラスタへのメンバ参加や離脱(障害発生)をトリガとして、何かの処理を行うために使います(**図8**)。

・member-join

メンバがクラスタに参加したときに発生します。“serf join”コマンド実行時に発行されるイベントです。クラスタ全体に新しいノードが参加したことが伝わります。

・member-leave

メンバがクラスタから明示的に離脱したときに発生します。正常終了(明示的に離脱)したときに発生するイベントであり、この状態のノード上でSerfを起動しても、自動的にクラスタに復帰することはありません。

・member-failed

メンバとの通信が途絶したときに発生します。障害発生とみなすタイミングが、このイベント発生時です。定期的にSerfクラスタからの監視が継続しますので、エージェントとの通信が回復すると、自動的にクラスタに再参加できます。

・member-reap

メンバ情報をクラスタ上から抹消するときに発生します。デフォルトではmember-failed発生後、24時間後です[注9]。

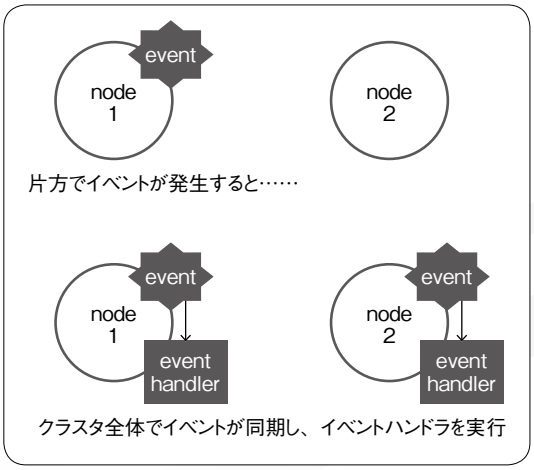
・member-update

メンバのタグ情報更新時に発生します。“serf tag”コマンドを使って、タグの変更を行うと、クラスタ全体に更新情報が通知されます。

イベントハンドラは、前述のとおり、イベントの発生をトリガとして任意の処理を行うことができます。Serfのノード状態を変化として、さまざまな処理に応用ができます。とくに、ロー

注9) 間隔は、Serf起動時に設定ファイルで明示することで、変更できます。

▼図7 イベントとイベントハンドラ



▼図6 すべてのノードで一斉にイベントを処理する

```
# serf event test
    2014/07/09 22:28:34 [INFO] agent: Received event: user-event: test
```

ドバランサや監視設定の追加削除に応用できるのではないでしょうか。

- member-join： 対象のノードをロードバランサに加えたり、監視設定を開始
- member-leave：ノードをロードバランサの設定から削除
- member-failed：ノードへの監視を一時的に停止
- member-reap： ノードへの監視設定の情報を削除

その他、タグの変化に応じてもイベントハンドラを処理できますので、roleに応じてアプリケーションをデプロイしたり、チェック用のコマンドを実行するなどの応用が考えられます。

♪ ユーザイベント

ユーザ側が任意のタイミングで発行できるイベントは2種類あります。

- event：任意の名称のイベントを発行する
- query：任意の名称のイベントと発行し、結果を取得する

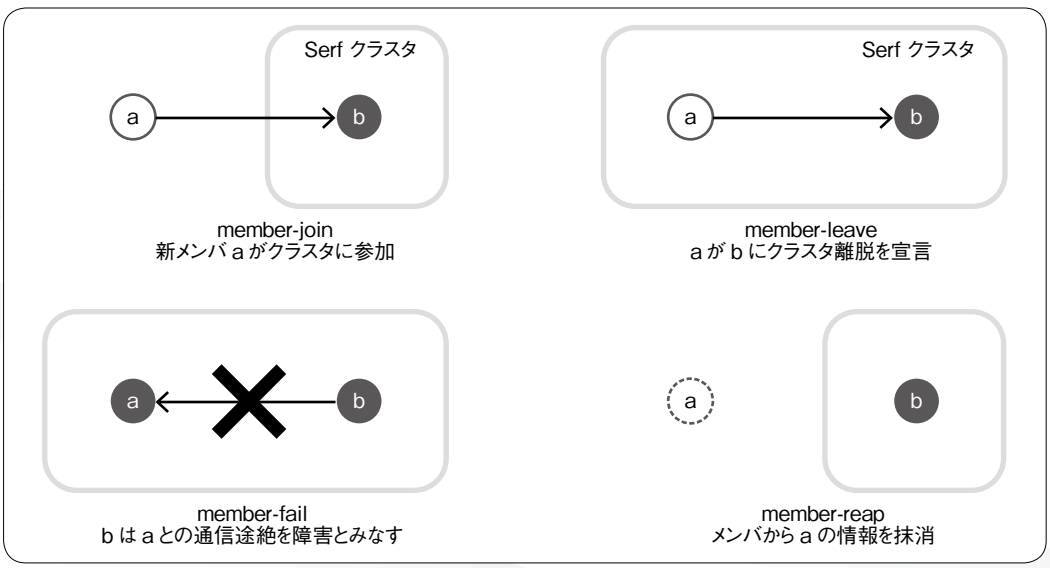
“event”と“query”の違いは、結果を取得するかしないかの違いです(**図9**)。これは、一方的にコマンドを実行するだけの“event”と、イベントを発行したノードに対して処理結果(標準出力の内容)を取得するかの違いです。たとえば、単純に“network restart”コマンドを実行したい場合は“event”を使いますが、その処理結果が正常かどうかを判別できるのは“query”です。

イベントハンドラと環境変数

イベントハンドラは、エージェント起動時に定義します。あらかじめ何のイベントに対して、どのような処理を行うかを明示します。あるいは、汎用的にイベントハンドラを指定し、イベントの種類を指定する環境変数“SERF_EVENT”を使い、条件付けを行うこともできます。

また、イベントハンドラで処理できるのは、コマンドライン上で扱えるものであれば何でもかまいません。一番簡単なものは、単純なコマンドの実行です。あるいは、コマンドを羅列したシェルスクリプトも利用できますし、RubyやPerlなど、自分が得意な任意のスクリプト言語に処理を引き渡すこともできます。また、環境変数が扱えないプログラムに対しても、あらかじめ条件を指定しておけば容易に利用できます。

▼図8 メンバ管理イベント



♪ イベントハンドラを指定するには

エージェント起動時、“-event-handler=”を指定します。

```
$ serf agent -event-handler=./event.sh
```

スクリプトでどのような環境変数を扱えるかは、**リスト1**のサンプルスクリプトを利用ください[注10]。

この状態で、“serf event”を実行すると、スクリプト側ではさまざまな環境変数を取得できていることがわかります。

♪ 環境変数“SERF_EVENT”

環境変数“SERF_EVENT”が、イベントを判別するために頻繁に使います。ここでは、“member-join”や“member-leave”のほか、“user”や“query”など、あらゆる種類のイベントで必ず使用されます。そのため、イベントに応じてさまざまな処理を行う汎用的なスクリプトを書くこともできます。

その他、環境変数の詳細は、公式サイトのドキュメント[注11]を参照してください。

♪ queryを取得する

イベントハンドラでqueryを指定することもできます。たとえば、クエリ名“check”時に“uptime”コマンドを実行させるには、**図10**のようにします。

コマンドラインからqueryを実行するには、**図11**のようにします。正常に応答があれば、各Serfクラスタでuptimeの実行結果が返ってきます。

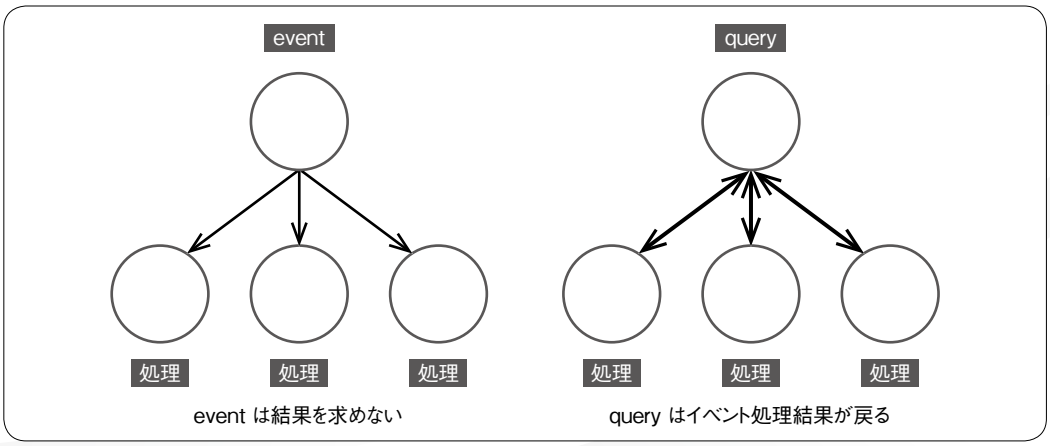
▼リスト1　サンプルスクリプト（event.sh）

```
#!/bin/sh

echo
echo "$0 triggered!"
echo
echo "SERF_EVENT is ${SERF_EVENT}"
echo "SERF_SELF_NAME is ${SERF_SELF_NAME}"
echo "SERF_SELF_ROLE is ${SERF_SELF_ROLE}"

echo "SERF_SELF_TAG is ${SERF_SELF_TAG}"
echo "SERF_TAG_ROLE is ${SERF_TAG_ROLE}"
echo "SERF_TAG_STATUS is ${SERF_TAG_STATUS}"
echo "SERF_USER_EVENT is ${SERF_USER_EVENT}"
echo "SERF_USER_LTIME is ${SERF_USER_LTIME}"
echo "SERF_QUERY_NAME is ${SERF_QUERY_NAME}"
echo "SERF_QUERY_LTIME is ${SERF_QUERY_
LTIME}"
echo
echo "BEGIN event data"
while read line; do
echo $line
done
echo "END event data"
echo "$0 finished!"
echo
```

▼図10　イベントハンドラでqueryを指定する

```
$ serf agent -event-handler=query:check=uptime
```

▼図9　eventとqueryの違い



注10）GitHubのissueに掲載されていたものに、新しい環境変数を追記しました：参照 https://github.com/hashicorp/serf/issues/54
注11）http://www.serfdom.io/docs/agent/event-handlers.html

LVSを使ったロードバランサへの応用

LVSとSerf

これまではSerfの概念や使い方が中心でした。次は、実際にSerfを使ったオーケストレーションを行いましょう。ここではLVS(Linux Virtual Server)のDSR(Direct Server Return)方式によるロードバランサに適用します。

LVSは軽量で手軽にバランシングを行うことができますが、コマンドラインでの管理が必要です。そのため、基本的なバランシング用のノードの追加や削除は、都度コマンドを実行します。この手作業で行うコマンド実行を、Serfのイベントハンドラに置き換えます。Serfがあれば、Serfのメンバ参加・離脱をトリガとして、バランシング対象に加えたり、削除できます。

本構成は、次のようにしました(**図12**)。なお、本環境はCentOS 6.5で検証を行いました。

・LVSサーバ(192.168.39.1)
　クライアントからのHTTPリクエストを受け止める
・LVSクライアント(192.168.39.2および3)
　バランシング先として、実際にクライアントに返す

LVSの準備

♪ LVSサーバ側

はじめにLVS管理用のツールipvsadmのセットアップを行います。

```
# yum -y install ipvsadm
```

必要があれば、サーバ再起動後、自動的に設定が反映されるようにします。

```
# /sbin/chkconfig ipvsadm on
$ /sbin/chkconfig --list ipvsadm
ipvsadm         0:off   1:off   2:on    ➥
3:on    4:on    5:on    6:off
```

ポートフォワーディングの有効化と、rp_filterの無効化を行います。

```
# sysctl -w net.ipv4.ip_forward=1
# sysctl -w net.ipv4.conf.default.rp_➥
filter=0
```

サーバ再起動後も有効にしたい場合は/etc/sysctl.confにも同様に記述を追加します。

♪ LVSクライアント

LVSサーバに届いたパケットを扱えるようにコマンドを実行します。

```
# iptables -t nat -A PREROUTING -d ➥
192.168.39.1 -j REDIRECT
# service iptables save
```

設定が有効かどうかは、iptablesコマンドを使います。

```
# iptables -t nat -nL
Chain PREROUTING (policy ACCEPT)
target     prot opt source               ➥
   destination
REDIRECT   all  --  0.0.0.0/0            ➥
   192.168.39.1
```

また、ブラウザからアクセスしたときに、どのホスト情報が表示されるかわからなくなるの

▼図11　コマンドラインからqueryを実行する

```
$ serf query check
Query 'check' dispatched
Ack from 'node1'
Response from 'node1':  23:56:38 up  5:12,  2 users,  load average: 0.00, 0.00, 0.00
Ack from 'node2'
Response from 'node2':  23:56:37 up  5:27,  2 users,  load average: 0.00, 0.00, 0.00
Total Acks: 2
Total Responses: 2
```

で、次のようにドキュメントルートに何らかのホストを識別できるファイルを置くと便利です。

```
# hostname > /var/www/html/hostname.html
```

LVSサーバ側のSerf設定

サーバ側では、イベントハンドラで呼び出されるスクリプト(**リスト2**)を設置します。

ファイルを設置したあとは、“**chmod +x ./lvs.sh**”のように、実行属性を与えます。あとは、エージェント起動時に、このスクリプトが呼び出されるように指定します。

```
$ serf agent -event-handler=./lvs.sh
```

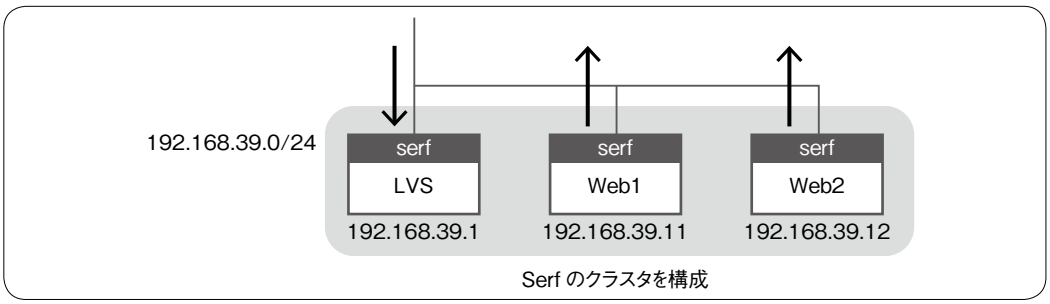
動作確認

クライアント側では、通常どおりエージェントを起動します。join先をLVSサーバ(192.168.39.1)にしつつ、タグで“**ROLE**”を指定します。これは、Serfクラスタすべてを LVSに登録するのではなく、ROLEに“**webapp**”の指定があるサーバのみを対象とします。

```
$ serf agent -join=192.168.39.1 -tag ROLE=webapp
```

同様のコマンドを、もう1台のノードでも実

▼図12　LVS(DSR)の構成



▼リスト2　イベントハンドラで呼び出されるスクリプト(lvs.sh)

```
#!/bin/sh

while read line
do
echo ${line}
    HOSTNAME=`echo ${line} | cut -d ' ' -f 1`
    ADDRESS=`echo ${line} | cut -d ' ' -f 2`
    ROLE=`echo ${line} | cut -d ' ' -f 3`

    case ${SERF_EVENT} in
    "member-join")
        if [ "${SERF_TAG_ROLE}" = "webapp" ] ; then
            ipvsadm -a -t 192.168.39.1:80 -r ${ADDRESS}:80 -g
        fi;;
    "member-leave" | "member-failed")
        if [ "${SERF_TAG_ROLE}" = "webapp" ] ; then
            ipvsadm -d -t 192.168.39.1:80 -r ${ADDRESS}:80
        fi;;
    ¥?)
      echo "other";;
    esac
break
done
exit 0
```

標準入力からホスト名 IPアドレス ロール名を取得

環境変数で判別 ${SERF_EVENT}

クラスタ参加時 ‘ipvsadm -a’ を実行しLVSに登録する

クラスタ離脱・障害時 ‘ipvsadm -d’ を実行しLVSから削除する

Serfが起動している間は、常に待ち受け

行します。

この状態でLVSサーバ側で確認コマンドを実行すると、各Serfノードが、自動的にバランシング設定が追加されていることがわかります(**図13**)。

各ノードのSerfエージェントを停止すると、自動的にバランシング設定は外れます。再びSerfを起動すると、再びバランシング対象に含まれます。

今回の例はLVSを扱いましたが、HAProxyに置き換えてもかまいません。コマンドを使う設定であれば、ほかの処理にも置き換えることができます。

▼図13　LVSサーバ側で確認コマンドを実行

```
# ipvsadm -Ln
IP Virtual Server version 1.2.1 (size=4096)
Prot LocalAddress:Port Scheduler Flags
  -> RemoteAddress:Port           Forward Weight ActiveConn InActConn
TCP  192.168.39.1:80 rr
  -> 192.168.39.11:80             Route   1      0          0
  -> 192.168.39.12:80             Route   1      0          0
```

▼表1　主要なオプション

オプション名(CLIオプション)	動作
node_name “-node”	ノード名称を指定。“serf members”などで参照
tags “-tag”	ノードのタグを指定。タグは“key=value”の形式
bind “-bind”	Serfエージェントが内部通信で使用するIPアドレス割り当て
interface “-iface”	複数のネットワークインターフェース混在時に明示可能
advertize “-advertize”	自身のノードのIPアドレスを明示。bindと違い、実IPアドレスの必要がない
discover “-discover”	mDNS(Multicast DNS)利用可能な環境で、クラスタを自動検出
encrypt_key “-encrypt”	暗号化秘密鍵を指定。鍵生成は“serf keygen”
leg_level “-log-level”	エージェントが表示するログレベルの指定。“trace”“debug”“info”“warn”“error”の5種
profle “profile”	Serfノード間の障害検出用に使用。チェックのタイミングを変更。通常デフォルトのまま
protocol “-protocol”	新旧のバージョンが混在する環境で、プロトコルのバージョンを明示
rpc_addr,rpc_aut “-rpc-addr”	通常はポート7373をバインド。変更可能
event_handlers “-event-handler”	イベント発生時に実行するコマンドを明示
start_join “-join”	agent起動時のジョイン先を明示。複数指定可
replay_on_join “-replay”	start_joinと同様の機能だが、過去のイベントもさかのぼって取得する
snapshot_path “-snapshot”	スナップショットは停止後の復旧時、過去イベントを重複受信しないようポインタを指定
leave_on_terminal	エージェントを“kill -TERM”か“kill -15”シグナルで停止したときの振る舞い。標準は“false”。このオプションを“true”にすると、停止時にmember-leave扱いとなり、障害発生時もステータスを“left”にする
skip_leave_on_interrupt	エージェント稼働時 Ctrl ＋ C で中断したときの振る舞い。デフォルトは“false”で、中断すると“left”になる。“true”にすると、中断時の処理は“failed”扱いになる
reconnect_interval	エージェントのfail検出後、何秒ごとに復帰したか確認する(デフォルト30秒)
reconnect_timeout	failedになってから、復帰を諦めるまでの時間(デフォルトは24時間)で、経過すると“member-reap”イベントが自動発行
tombstone_timeout	leftしてから情報を保持する時間。reconnect_timeoutと違い、復旧するかどうかのチェックは行わない
disable_name_resolution	名前解決を行わない

設定オプション

主要なオプション

主要なオプションは**表1**のとおりです。なお、オプションの詳細については、日本語での解説(拙作となりますが)もあります。

・Serf設定オプションまとめ¦Pocketstudio.jp log3
http://pocketstudio.jp/log3/2014/03/29/serf_configuration_quick_guide/

筆者の場合は、**リスト3**のようなテンプレートをあらかじめ作成しておき、汎用的に使い分けています。

まとめ

これまで見て来たとおり、Serfは非常にシンプルです。また、Serfを導入することにより、既存の作業手順を変更する必要はありません。日々のコマンド、むしろ日々の業務における手作業で面倒なところを、適切に処理してくれるツールであると言えるでしょう。

筆者の場合、現在は運用の現場から離れてしまいましたが、もし1年前にSerfがあれば、このように使っていました。

・メンテナンス時のhttpd一斉停止
・ネットワーク設定(gatewayなど)の切り替え
・新しい設定ファイルを反映させるための同時restart

このほか、イベントハンドラは任意に設定できますので、現場しだいでさまざまな応用ができるのではと思います。Serfは使えるようになるまでの時間がとても短いので、まずは手を動かして、試してはいかがでしょうか。SD

▼リスト3　筆者の作った汎用設定テンプレート

```
{
  "node_name": "miku3",          …… ノード名は「miku3」
  "tags": {                      …… ノードにタグを付ける
    "role": "develop",           …… タグ「role=develop」
    "network": "local"           …… タグ「network=local」
  },
  "interface": "eth1",           …… インターフェースは「eth1」
  "discover": "mikusan",         …… discoverするネットワークは「mikusan」

  "encrypt_key": "o6Md8LBVhwPi2UnbJBAwNA==", …… 暗号鍵の指定
  "log_level": "debug",          …… ログ出力は「debug」レベル

  "leave_on_terminate": true,    …… `killで停止しても「left」扱いにする
  "skip_leave_on_interrupt": false, …… Ctrl+Cで停止したら「left」扱いにする

  "reconnect_interval": "5s",    …… 再接続間隔を「5秒」
  "reconnect_timeout": "30m",    …… "failed"発生から「30分」復帰を待つ
  "tombstone_timeout": "30m",    …… "left"後、ノード情報を「30分」まで保持する

  "event_handlers": [            …… イベントハンドラの指定
    "./event.sh",                …… すべてのイベントで「./event.sh」を実行
    "user:deploy=./deploy.sh"    …… ユーザイベント"deploy"時に「./deploy.sh」を実行
  ]
}
```

サポートエンジニアのクラウドワークスタイル

新連載

Heroku女子の開発日記

第1回 Heroku事始め

今月号から始まる「Heroku女子の開発日記」は、PaaS製品である「Heroku」のユーザを、導入から開発・運用までサポートします。毎月、連載の終わりには、筆者が西海岸での仕事・生活をレポートする「サンフランシスコだより」も掲載します。第1回は、Herokuの概要について説明していきます。

Heroku　織田 敬子（おだ けいこ）

Heroku

みなさん、Herokuをご存じでしょうか。聞いたことはあるけど触ったことはない、という人も多いのではないかと思います。今回は連載を通じて、Herokuでできることを実例も混ぜて紹介していき、みなさんに実際に手を動かしてHerokuを体験してもらいたいと思っています。

Herokuは会社の名前であり、その製品の名前でもあります。ローマ字読みでそのまま「ヘロク」と読んでください。Herokuは2007年、ジェームス、アダム、オライオンの3人で設立したスタートアップが始まりで、2010年末にセールスフォース・ドットコムにより買収されています。日本風のデザインや商品名を多用しているので日本の会社と思われがちですが、本社はアメリカのサンフランシスコにあります。

Herokuとは

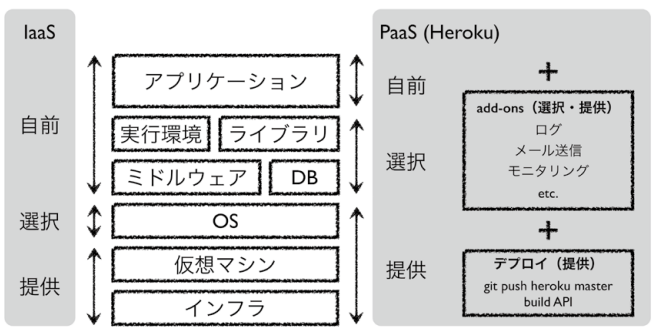
Herokuとは具体的に何をするサービスなのでしょうか。Herokuを知らない方、アプリケーション開発初心者の方には筆者は「Herokuは自分で作ったアプリケーションを全世界の人に見てもらえるように公開できるところ・サービス」と説明するようにしています。これだけだと少し具体性に欠けるので、ここでは一歩踏み込んだ形で説明したいと思います。

PaaSとIaaS

Herokuはいわゆる「クラウド」分野の製品で、その中でもPaaS(Platform as a Service)に分類されます(**図1**)。似たものとしてはIaaS(Infrastructure as a Service)としてAmazonが提供しているEC2があります。

簡単に言うとIaaSでは、空のコンピュータが提供されるので、OSなどを選択し、デプロイしたいアプリケーションに応じたミドルウェアや実行環境、ライブラリ群などを自分で用意し、そのうえにアプリケーションを自分でデプロイします。一方PaaSではアプリケーションが走るプラットフォーム、デプロイ方法ま

▼図1　Herokuの立ち位置



自前：深い知識・理解が必要　**選択**：最低限の知識は必要　**提供**：知識があるとよい

で用意するので、開発者はローカルで開発しているアプリケーションをただデプロイします。このように、開発者が実行環境などを意識する必要がなく、コアであるアプリケーションの開発に注力できるのはPaaSの大きな利点です。

逆に簡単にカスタマイズできず、自由度が低いというのはPaaSの欠点でもあります。とくにIaaSに慣れている人には、使っているライブラリが入っていない、バージョンが古いなど不都合に感じることもあるかもしれません。

Herokuでは現在、カスタマイズをしたい人が使いやすいようにするしくみに取り掛かっており(buildpackなどはその一例)、将来的にはPaaSの利点は残したままプラットフォーム部分のカスタマイズが容易にできるようになっていくでしょう。

Herokuの特徴

PaaSの特徴はHerokuの特徴でもありますが、ここではもう少し例を挙げたいと思います。

git push heroku master

Herokuでは、デプロイにGitを用います。Gitを用いたバージョン管理は現在のアプリケーション開発では主流となっていますので、開発者はデプロイ時に特別に何かを学習することなく、Herokuへのデプロイを行うことができます。

heroku ps:scale web=100

Heroku上のアプリケーションは、上記のようなコマンド1つで簡単にスケールアウト・スケールインすることができます。これにより、急なトラフィックの増加にもすぐ対応できます。

このほかにもHerokuにはアドオンや多言語サポートなどさまざまな特徴があります。これらについてはのちほど詳しく説明します。

もっとHerokuを知る

Herokuの創業者の1人であるアダムが書いた「The Twelve Factor App」[注1]には、Heroku上だけではなく、あらゆるクラウド上で走らせるアプリケーションの開発手法がどうあるべきかが書かれています。Herokuで走るアプリケーションはどのようにあるべきなのかを知るうえでもぜひ読んでみることをお勧めします。

Dev Center[注2]にはHerokuに関するすべてのドキュメンテーションがあります。Herokuで何かわからないことができたら、まずはここで検索してみましょう。この連載でも、Dev Centerへの引用を多用することになると思います。

Herokuではいくつかの独自に使っている用語があります。ここでは、その中でも代表的な3つを紹介します。

dyno(ダイノ)

Heroku上で走るインスタンスの単位。アプリケーションごとに数を自由に増やしたり減らしたりできます。サイズ(おもにメモリ・CPU性能の差)が設定でき、現在1X(シングル)、2X(ダブル)、PX(パフォーマンス)があります[注3]。

add-on(アドオン)

アプリケーションに追加できる拡張機能。おもにサードパーティ製です。データベースや、メール送信、ログ・モニタリングなどがあります[注4]。

注1) http://12factor.net/
注2) https://devcenter.heroku.com/
注3) https://devcenter.heroku.com/articles/dynos
注4) https://addons.heroku.com

buildpack（ビルドパック）

アプリケーションをデプロイしたときに走るスクリプト。ここでRubyをインスタンスに入れたり、依存関係を解決したりします。Herokuが提供・サポートしているものもあれば、自分で作ることもできます[注5]。

多言語サポート

Herokuで現在サポートされている言語はRuby、JVM系（Java、Scalaなど）、Node.js、Python、PHPなどがあります。これらの言語は、Herokuによってbuildpackが作成されメンテナンスされています。このような形でサポートしていない言語でも、buildpackを使うことによって対応できます。

サポートしている言語については、言語ごとにそれぞれ精通したエンジニアがおり、日々進歩するこの業界において新しい技術をすぐさまHeroku上で使えるようにしています。たとえば、Rubyなどは新しいバージョンがリリースされたらほとんどその日のうちにHeroku上でも使えるようになっています。

また、言語やその代表的なフレームワークごとに充実したgetting startedページ[注6]もあり、自分の得意な言語でHerokuを使い始めてみることができます。

料金体系

Herokuの課金は、使った分だけの従量課金となっています。ベースの金額については、dynoのサイズ（1X、2X、PX）によって異なります。以下、dynoのサイズごとの料金です。Webページ[注7]上で価格をシミュレーションしてみることもできます。

1X dynos : $0.05 / hour
2X dynos : $0.10 / hour
PX dynos : $0.80 / hour

アドオンについては各アドオン、またアドオン中でのプランごとにベースの金額が異なってきます。たとえば、Heroku postgresアドオンには無料のものから$6,000／月するものまでさまざまなプランが用意されています。詳しくはアドオンのページ[注8]を参考にしてください。

dynoについては時間単位、アドオンについては月単位での価格が記載されていますが、これらの課金はすべて秒単位で計算されています。

Herokuには、1つのアプリケーションにつき1ヵ月あたり1X dyno約1つ分が無料でついてきます。1X dynoが1つあれば、個人でHerokuを試してみるには十分です。この、無料から始められるというのはHerokuの大きな特徴でしょう。

Herokuの使用料の支払いは基本的にはクレジットカードで行われます。月々に使用した料金が翌月請求されます。Herokuはクレジットカード情報を登録しないと利用できないと思われている方も多いのですが、実はHerokuはクレジットカード情報を登録することなく始められます。アドオンを試してみる場合にはクレジットカード登録が必要です（Heroku postgres、pgbackupsアドオンを除く）。クレジットカード情報の登録なしでは作成できるアプリケーション数など利用できる範囲も限られてきますが、その枠内でも十分に試してみることができるで

注5）https://devcenter.heroku.com/articles/buildpacks
注6）https://devcenter.heroku.com/articles/quickstart
注7）https://www.heroku.com/pricing
注8）https://addons.heroku.com/

しょう。

請求書払いなども対応したエンタープライズ向けパッケージ[注9]での販売もしていますので、興味のある方はぜひお問い合わせください。

ここでカバーしたことも含めて料金の詳細は、Webページ[注10]を参考にしてください。

Sign Up

Herokuへのサインアップはとても簡単です。Herokuのホームページ[注11]へ行くと中央に大きく「Sign up for free」と書かれたボタンが見えます。メールアドレスを入力すると確認メールが届きますので、確認メール中のリンクよりパスワードを設定し、アカウントを作成してください。以降、**https://dashboard.heroku.com**があなたのHerokuダッシュボードになります。ここでアプリケーションやアカウントの管理ができます。

Heroku toolbelt

サインアップも終わったので、次はHeroku toolbeltのインストールにとりかかりましょう。ToolbeltはCLIツールであるHeroku client、アプリケーションをローカル上で動かすときに役立つForeman、バージョン管理ツールのGitの3つから構成されています。Toolbeltをページ[注12]からダウンロードし、インストールしてください。そのあと、Macの場合はターミナル、Windowsの場合はコマンドプロンプト[注13]を開き、次のとおりherokuコマンドを使ってログインをしてみましょう。

```
$ heroku login
Enter your Heroku credentials.
Email: adam@example.com
Password:
Could not find an existing public key.
Would you like to generate one? [Yn]
Generating new SSH public key.
Uploading ssh public key /Users/adam/.ssh/id_rsa.pub
```

今回はここまでです。ちょっと物足りないな、という方は**heroku help**コマンドを打って、どんなherokuコマンドがあるか見てみたり、Herokuダッシュボードでアカウント情報を見てみたりしましょう。

サンフランシスコだより

はじめまして。この連載を書いているKeikoと申します。HerokuではTechnical Support Engineerをやっています。2013年4月より働いており、今年8月からサンフランシスコ勤務になりました。各連載の終わりに、筆者の住んでいるサンフランシスコについてコラム形式で少し書いていきたいと思っています。サンフランシスコに来る予定のある方は、ぜひHerokuに遊びにきてください！

Heroku internal conference Shinzoku-kai

少し前の話になりますが、6月の末にサンフランシスコでHerokuのinternal conferenceであるShinzoku-kai（親族会）が行われました（**写真1**）。不思議なことに、Herokuは内部ツールやイベントの名前に日本語を使いたがります。社員の中にも、少し日本語ができる人がけっこういます。

注9） https://www.heroku.com/critical
注10） https://devcenter.heroku.com/articles/usage-and-billing
注11） https://www.heroku.com
注12） https://toolbelt.heroku.com
注13） Windowsでうまくいかない場合は、Heroku toolbeltをインストールした際に同時にインストールされたGit Bashを使ってコマンドを実行してみてください。

Heroku女子の開発日記

Herokuは、実は半数近くがリモート勤務のため、こういったinternal conferenceは皆が世界中から一堂に会するいい機会です。筆者のチームも世界各地に散らばっているのですが、普段チャットやGoogle hangoutでしか話さないみんなと実際に顔を合わせて話すととてもうれしくなりますし、あとあとの仕事にもいい影響を与えます。

▼ 写真1　Shinzoku-kaiの様子（Photo by Yannick）

サンフランシスコの宿泊と交通

Shinzoku-kaiの日程がGoogle I/Oと被ったということもありホテル価格も高騰していたため、筆者は同僚と一緒にAirbnb[注14]を利用しました。夜にリビングなどで話す機会があり、なんだか合宿のようでとてもよかったです。サンフランシスコはAirbnbの本社がある土地ということもありたくさんの候補地があるので、機会があればぜひ利用してみてください。

移動はよくUber[注15]を使いました。Uberは普通黒塗りの車なのですが、その1つランクが下のUber X（普通の乗用車）を使うと、タクシーよりたいてい安いです。複数人で移動するととても経済的ですし、降りるときにチップの計算もいらず財布も出さなくても良いのでとても便利です。サンフランシスコでは空港はもちろん市内いたるところにUberがはびこっていますので、こちらもサンフランシスコへ来たときには利用されることをお勧めします。

Heroku Office

HerokuのOfficeはSoMa（South of Market）と呼ばれる地区にあります。サンフランシスコとサンノゼをつなぐCaltrainと呼ばれる鉄道のSan Francisco駅の近くになります。このSoMa地区にはたくさんのIT系企業のオフィスがあります。2013年の夏にこのオフィスに引っ越しをしたのですが、筆者はこのオフィスがとても気に入っています！　広くて、たくさんの遊び心のあるスペースがあり、こだわりを感じ、それでいてとても居心地が良い。みなさんも遊びに来たらきっとHerokuで働きたくなることでしょう（**写真2**）。YouTubeにオフィスの紹介ビデオ[注16]がありますので、ぜひご覧ください。SD

▼ 写真2　Herokuオフィス（Photo by @seaofclouds）

注14） https://www.airbnb.com/（宿泊施設を貸し出せる、借りられるWebサイト）
注15） https://www.uber.com/（携帯アプリケーションから利用できる配車サービス）
注16） https://www.youtube.com/watch?v=WLGLTX4yqSA

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）

2014年3月号

第1特集
データベースの諸問題
RDBとNoSQLどちらを選びますか?

第2特集
ネットワークエンジニアのための
プロキシサーバの教科書

一般記事
・さらに踏み込む、Mac OS Xと仮想デスクトップ #1

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。

サーバーワークスの瑞雲吉兆仕事術

第2回　Google Apps、SalesforceそしてAWS

クラウドの登場によって、情報システムのありかた、そしてエンジニアのキャリアそのものが大きく変わろうとしています。本連載では、すでに起こりつつある変化を読み取ることで、みなさんが自分のキャリアを考えるための材料を提供します。

Writer ㈱サーバーワークス　代表取締役　大石 良（おおいし りょう）／ http://blog.serverworks.co.jp/ceo/

クラウド導入の失敗と成功

前回は「クラウド前夜」と題して、国内にスマートフォンが入ってくる流れについて話しました。今回は、筆者たちがこの3つのメジャーなクラウドサービスをどのように使っていき、どんなふうに付き合うことにしたのか、その流れを紹介させていただくとともに、そこから筆者たちが何を学び、将来のエンジニア像をどのようにイメージすることになったのか、筆者たちの事例を紹介します。

Google Apps導入の理由とは

前回、筆者たちが「モバイルが起点になって、Google Appsを使うことになった」という話をしました。「iPhoneから業務をやりやすくする」ことを中心に据えるようになると、いろいろと会社のツールが合わないケースが出てきます。もっともその影響を受けたのは、SharePointでした。筆者たちは、2006年からMicrosoft Office SharePoint Server（MOSS）をグループウェアとして使っていましたが、「Internet Explorerからでないと使えない」ことが致命傷になっていました。今となっては信じられないことですが、2006～2007年当時は、筆者たちも社内のPCとブラウザを「Windows + Internet Explorer」で決め打ちにしていたのです！

個人的にSharePointはとても気に入っていたのですが、「iPhoneから使えないのではしかたない」と考えるようになり、スケジュール・カレンダーはGoogle Appsに、ファイル共有はローカルのファイルサーバに、と分割して機能させることにしました。こうして、オンプレミス環境だったグループウェアがGoogle Appsというクラウドサービスに取って代わりました。

増え続けるサーバへの対策

そして、ほどなくしてAWS（Amazon Web Services）と出会います。当時は「EC2」として認知されるケースがほとんどでしたので、筆者たちも、AWSというよりは「EC2と出会った」という感覚でした。もともと大学向け合否照会システムの運用をしていた筆者たちは、「2月の特定の日の、午前10時から15分間だけに発生する極端なスパイクトラフィック」を何とかして解決しようと、さまざまなソリューションを検討していました。

初期のころはハードウェアからのアプローチが中心でした。ブレードサーバによるスペース集約などはかなり早くから実施し、一定の効果を上げましたが、ハードウェアのアプローチでもっとも効果的だったのがCitrix NetScalerの導入でした。今はロードバランサやアプリケー

ションデリバリーコントローラと呼ばれるケースが多いようですが、当時は「Webアクセラレータ」というカテゴリの製品でした。

当時はまだ低速なモバイル端末からのアクセスが非常に多かったのですが、低速な回線から大量にアクセスが集中すると、Apacheのプロセスが大量に立ち上がってしまい、メモリを圧迫するという問題に悩まされていました。NetScalerは非常にインテリジェントで、Webサーバの前に配置することで、リバースプロキシとして動作することに加え、HTTP1.0のアクセスもHTTP1.1にしてWebサーバに転送してくれるため、回線が遅いときに全部のコンテンツを返し終わるまでApacheがメモリに滞留してしまうという問題が解決されたのです。これはとても効果的で、アクセスが増え続ける中でもサーバ台数を増やさなくても対応できるまでになりました。サーバ台数が、アクセス数に応じてリニアに増え続けるという悪夢を、未然に防ぐことに成功したのです。

ですが、ハードウェア自体が相応の価格になることに加え、結局こうしたハードウェアの保守・メンテナンスも追加で行う必要が発生するようになり、「現状維持はできるようになったが、運用のための負荷が下がったわけではない」という状況に変化はありませんでした。

AWSとの出会い

こうした中で、当社のエンジニアが「これはスゴいですよ」と見せてくれたのが、EC2でした。筆者も最初は「なぜあのAmazonが？」と思いました。しかも、今と違いEC2のタイプも「インスタンスストアタイプ」と呼ばれる「インスタンスを停止させると、その仮想サーバの中身もまるごと消える」というタイプからしか選択できなかったのです。今でこそ「immutable infrastructure」という概念がよく知られていますが、当時はSSHでログインしてhttpd.confをゴリゴリ書き換えるのが当たり前でした。「止めたら中身が消える仮想サーバをいったい何に使うんだ！？」という反応が、正直な第一印象でした。

それでも、よく調べてみると「EC2以外のところにすごみがある」ことがわかってきました。とくにAmazon S3のインパクトは強烈で、当時の情報ではいったいどうやって実装しているのかまったく想像もつかないレベルでした。「これは何かとてつもないことが起きるかもしれない」と予感した筆者たちは、2008年の正月に「社内サーバ購入禁止令」を出します。まずは開発サーバや自社サーバなどから強制的にEC2を使い始めてみて、最初の予感が本物かどうか確かめるためです。そして「これはいける」と判断しました。2008年から外販を始め、2009年からは「事業の主軸をAWSにする」と決断するにいたりました。

余談ですが、インスタンスストアタイプしかなかったころは、みんなよってたかって「EC2は止めたらデータが消えるから使えない」と文句を言っていましたが、今になって「サーバの内部にデータを残すなんてイケてない！　これからはimmutable infrastructureだ！」という風潮が高まっており、AWSの中の人は「それみたことか」と思っているのではないかとニヤニヤ観察しています(笑)。

Salesforceの導入に失敗!

さて、このような理由で筆者たちはAWSに焦点を当てることにしたのですが、事業推進のためには営業が大切なので、営業支援ツールを本格的に探すことにしました。いくつか選択肢はあったものの——確実に違いないと考え——Salesforceのアカウントをごく少数契約し、社内への導入を図ってみました。ところが見事に頓挫しました。まず営業からはクレームが頻発しました。「画面がイケてない。使い勝手が悪い。自分たちのやりたいことが実現できない」などなど、細かい要請が積み重なります。会社としても、現場で使われないのにこの金額では高過ぎる、ということになり結局導入に失敗しました。

内製ツールでしのごうとするも失敗!

この失敗を「Salesforce固有の問題ではない。自分たちの業務に合うモノは自分たちで作らないと満足できるものは作れない」と考えた筆者たちは、営業支援ツールを内製することにしました。幸い自社に開発者もいますし、こういったツールを作る経験はそれなりにしてきましたので、特有の業務要件はあったものの構築自体はそれほど高いハードルではありませんでした。こうして内製のツールが完成します。

「めでたしめでたし」に、なるハズでした。ところが、また別な問題がわき上がりました。業務が少しずつ変わってきたにもかかわらず、ツールを修正するための人手が割けなかったのです。これは開発者を抱える会社が陥りがちな「罠」ですが、「開発者がいれば、いつでも修正できる」と錯覚してしまうのです。開発者のリソースが空いていれば、内製システムを作ったり修正したりといったことができるかもしれませんが、実際にそんなに人手が空いていません。SIをやっている会社が「SIをやりながらパッケージやサービスを作ろう」としてもうまくいかないのも、同じ理由です。結局、筆者たちが内製したツールも「いつまでたっても問題が修正されない」状態が長期にわたって放置されることになってしまい、結局「Salesforceを使っていたときよりも営業からのクレームが大きくなる」という逆転現象を招いてしまったのです。

再度Salesforceへ

このような失敗を経て、筆者たちは「システムは作った瞬間よりも、継続して運用できるかどうかのほうが大切だ」というごく当たり前のことを、身をもって体験しました。そして同時に「開発者がいるからといって、いつでも自社ツールのために時間を使えるわけでもないし、ベストなものが作れるわけでもない」ということも学習しました。でも、こうした課題を解決するためには「やはりSalesforceが良いかもしれない」と考えるようになったのです。ただし、前回と同じことをやっては同じ結果を招くだけです。「今度こそ絶対に営業支援ツールの導入を成功させる」という意気込みのもと、次のアクションを設定しました。

・まず、経営者である筆者がSalesforceの中身／価値をよく理解すること

・営業全員に、全お客様情報、全商談を必ず入力させること
・営業が自分でレポートなどを作れるようにして、自分たちで営業成績をあげるための使い方ができるようにサポートすること

こうしたアクションを経て、ようやくSalesforceが社内に定着するに至りました。

自社開発の失敗で思うこと

当社の事例は、このように2回の失敗を経て結局Salesforceで成功を迎えましたが、このことは筆者たちにとっても非常に大きなインパクトがありました。

前号の記事で「自分たちはメールサーバを立てて運用する能力があると思っていたが、実際にはGoogleを使ったほうがずっとよかった」と紹介しましたが、これはインフラの話でした。ところが、アプリケーション開発の分野でも「自分たちで作ることができると思っていたが、Salesforceを使ったほうがずっとよかった」という事実を突きつけられたのです。

これは恐怖でした。SIerといえば、アプリケーションを開発して、インフラを運用して、そしてお金をいただく商売です。それが「アプリケーションの開発もインフラの運用も、どちらもクラウド事業者のほうが優れている」という事実を突きつけられたのです。この経験が「作らないSI」という筆者たちの方向性につながっていくのです。

実は「作らないSI」という方向性と、「作らないSI時代のエンジニア像」こそが、筆者がこの連載でもっとも伝えたいことなのです。

「クラウドありき」ではない

さて、著者の会社は「AWS専業インテグレータ」と称してはいるものの、決して2010年当時から「すべてのシステムをAWSなどのクラウドにするのがベストだ」と考えていたわけではないことが理解いただけたのではないでしょうか。

Google Appsは「モバイル業務を進めるため」、AWSは「突発的なトラフィックをさばくため」、そしてSalesforceは「自分たちの業務を円滑にするため」という目的でした。それぞれ達成するための解決策を探していたら、結果としてクラウドになった――のです。

筆者たちはクラウドインテグレータを標榜していますが、決してクラウドを「目的」にしているわけではありません。どこまでいってもコンピュータは人間の「道具」であり、クラウドそのものが目的になることはないと思います。しかし、試行錯誤で最適解を見つける努力をしているうちに、結果として「これはクラウドのほうがいいよね」という領域が増えていった、ということなのです。そして「作らないSI時代のエンジニア像がある」ことがわかってきたのです。

次稿では、「作らないSI」の流れと社内システムの未来像についてもう少し詳しく掘り下ます。SD

㈱サーバーワークス
代表取締役
大石 良（おおいし りょう）

・昭和48年7月20日新潟市生まれ
・コンピュータとの出会いは10歳の頃
・当時はPC-8001にベーマガのプログラムを入力する日々
・コンピュータの購入は11歳／SHARP X1
・中2の時に初めてプログラムが書籍に掲載
・高校入学記念にX68000を購入
・大学生の時にパソコン通信開始。本格的にシェアウェアを販売
・総合商社でインターネットサービスプロバイダー事業に携わる
・2000年にECのASPを立ち上げるべく起業

思考をカタチにするエディタの使い方

るびきち流 Emacs超入門

Writer | るびきち twitter@rubikitch | http://rubikitch.com/

第5回 カーソル移動と入力支援でスピードアップ!

エディタ選びで最初に注目するのは、やはり「どれだけ効率的に文章・コードを書けるか」ですよね。今回は「isearch+Migemo(+ace-jump-mode)」を使った高速なカーソル移動、「dabbrev-expand」、「hippie-expand」を使った強力な入力支援を紹介します。

ども、Emacsと熱愛状態にあるるびきちです。「Emacsの恩恵は文字入力の一元化にある」という路線で本連載は進めていっています。今回はテキストエディタの本質であるカーソル移動と効率的な文字入力について触れていきます。本記事を読んで設定していけば、Emacs以外で文章を書くことが本当に馬鹿らしくなってきますので楽しみにしててください。

カーソル移動

基本

まずはカーソル移動についてです。カーソル移動の最小単位は文字・行です。C-b(←)、C-f(→)、C-p(↑)、C-n(↓)はもう無意識で使えていますか？　マウスやカーソルキーを使っていない時点でメモ帳やブラウザのtextareaなどただのおもちゃレベルで、相手になりません。

その次は単語単位の移動になります。M-b(←)、M-f(→)というふうに、Ctrlがmetaに変わっただけですが、少し離れた位置にカーソルを移動させる場合に重宝します。また、C-a(行頭)、C-e(行末)も大事なコマンドです。

プログラミング言語を編集する場合は、単語とは別に「まとまり」単位で動いてほしいことがあります。変数名や関数名といったシンボル、文字列リテラル、対応する括弧の末尾など、自然言語における「単語」とは違ってプログラミング言語的に意味する構成要素があります。たとえば、Emacs Lispで「find-file」という文字列がある場合、単語としては2語ですが、Emacs Lispとしては1つのシンボル(名前)です。その構成要素を「S式」といいます。S式単位の移動はC-M-b(←)、C-M-f(→)です。

S式はLisp由来の概念で、Emacs Lispにはとくによくなじんでいます。ほかの(非Lisp系でさえも)言語でもシンボルや括弧に囲まれたものはS式とみなされます。S式という概念を詳しく説明するのは本稿から外れてしまうのでこれくらいにしておきますが、コーディングにおいてC-M-bとC-M-fは基本的なカーソル移動手段なので感覚を身につけておいてください。

インクリメンタルサーチ

Emacsでのカーソル移動コマンドは強力ですが、到達したい場所が離れている場合は、それだけでは力不足です。離れた場所にでもすぐに到達できるようにするためには、移動したい場所の文字列を指定して、エディタに探してもらう「検索」という機能が必要となります。

Emacsでの検索はインクリメンタルサーチ(isearch)が採用されています。普通の検索は検索文字列を入力したあとに検索を始めますが、

isearchは検索文字列をタイプするたびに検索します。そのおかげで最小限のタイプ数で到達したい場所へとカーソルを移動してくれます。だからこそisearchは重要なカーソル移動コマンドです。少し離れた場所に移動する場合はisearch、これは鉄板です。邪魔なダイアログボックスが現れないこともありがたいです。

isearchは前方(カーソル以降)にはC-s、後方(カーソル以前)にはC-rを使います。1ストロークの押しやすいキーに割り当ててあることから、多用されるべき機能であることが見て取れます。

isearchをしていて目的の場所に到達したときは RET を押して終了します。C-fなど、ほかのカーソル移動コマンドでもisearchを抜けられます。

isearch一発で目的の場所に到達するとは限りません。その場合に採れる選択肢は2つです。さらに検索文字列を入力するか、同じ文字列を検索するかです。同じ文字列を検索するにはisearch中にC-s(前方)かC-r(後方)を使います。

isearch終了後に最後に検索した文字列を再検索するにはC-s C-sあるいはC-r C-rを使います。

C-uを前置すると正規表現isearchになります。つまり、前方はC-u C-s、後方はC-u C-rです。正規表現とは文字列のパターンを記述するミニ言語で、あらゆる分野で使われています。正規表現についてはかなり奥が深く、それだけで1冊の本になっているほどです。ですが、最小限の正規表現を知っておくだけでEmacsでの日常的な編集はまかなえます。

正規表現で使われる文字は「メタ文字」(**表1**)と「それ以外の文字」に分かれます。メタ文字とは、パターンを表現するために特別な意味を持つ文字です。メタ文字以外の文字はその文字そのものにマッチします。Emacsの正規表現はややクセがあり、「(」と「)」と「|」はそれぞれバックスラッシュを付けて「\(」、「\)」、「\|」とします。

たとえば、後方にある「C-x」と「C-f」を検索するときには正規表現isearchを使います。C-u C-rのあとに文字どおり「C-.」を入力します。メタ文字「.」は改行以外の文字にマッチするので、この場合xとfにマッチします。厳密には「C-[xf]」を指定すべきですが、「.」を使ったほうが手軽です。その分余計な文字にもマッチしてしまうので、臨機応変に対処してください。

▼表1　主な正規表現のメタ文字

メタ文字	意味
.	改行以外のすべての文字
*	直前の表現が0回以上
+	直前の表現が1回以上
?	直前の表現が0回か1回
[...]	文字クラス(...の文字に一致)
[^ ...]	否定文字クラス(...の文字に一致しない)
$	行末
|	|で区切られた表現のうちどれか
\(... \)	グルーピング

Migemoを使って、ローマ字で日本語をisearch

isearchはカーソル移動においては不可欠ですが、日本語とは相性がよくありません。なぜなら、日本語文字列を検索するには、わざわざ漢字変換をする必要があるからです。漢字変換するくらいならカーソル移動コマンドをそのまま使ったほうが早いくらいです。漢直入力を使わない限り、日本語と漢字変換は切っても切れない関係であり、大きなハンデとなっています。

漢字変換なしでローマ字で日本語文字列をisearchできたらいいなと思いませんか？　その願望を叶えてくれる神ツールがMigemoです。Migemoはもともと、内部処理を担当するRubyスクリプトmigemoとEmacsインターフェースのmigemo.elの2つで構成されています。今のRubyではmigemoは動かないので、内部処理はC言語で書かれたcmigemoを使います。よって、必要なのはcmigemoとmigemo.elです。

Debian系列のGNU/Linuxならば両者ともパッケージ化されているのでインストールは簡単です。「sudo apt-get install cmigemo migemo-el」を実行するだけで、初期設定までしてくれて、

▼リスト1　非公式Emacs Lispパッケージ（MELPA・Marmalade）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)
```

▼リスト2　migemo.elからcmigemoを使う初期設定

```
(when (locate-library "migemo")
  (setq migemo-command "/usr/local/bin/cmigemo") ; HERE cmigemoバイナリ
  (setq migemo-options '("-q" "--emacs"))
  (setq migemo-dictionary "/usr/local/share/migemo/utf-8/migemo-dict") ; HERE Migemo辞書
  (setq migemo-user-dictionary nil)
  (setq migemo-regex-dictionary nil)
  (setq migemo-coding-system 'utf-8-unix)
  (load-library "migemo")
  (migemo-init))
```

そのまま使えます。多くのGNU/Linuxはパッケージシステムがあるため、インストール・設定・管理がとても楽です。パッケージを使うためには初期設定が必要となります（**リスト1**）。

OS側でパッケージ化されていない場合はcmigemoとmigemo.elは別個でインストールし、初期設定も行う必要があります。Macは「brew install cmigemo」で、Windowsは@kaoriya氏のサイト[注1]からcmigemoのバイナリを取ってきます。

migemo.elはMELPA（Milkypostman's Emacs Lisp Package Archive）に登録されています。初期設定でHEREと書かれた部分は環境に合わせて書き換えてください（**リスト2**）。次のようにしてインストールが終われば、init.elに設定を書き加えてください。

```
M-x package-refresh-contents
M-x package-install migemo
```

Migemoをインストールしたら、Emacsを再起動してローマ字で日本語文字列を検索してください。たとえばC-s nihoで「日本語」に到達できるようになります。実際に使ってみれば感動すること請け合いです。

注1）URL http://www.kaoriya.net/software/cmigemo/

ace-jump-mode

isearchは強力なカーソル移動方法ですが、もう1つ便利なカーソル移動コマンドを紹介しておきます。M-x ace-jump-word-modeは画面内の任意の単語開始位置に3ストローク以内でジャンプするコマンドです。isearchはカレントバッファ全体が走査対象ですが、ace-jump-word-modeは画面内の移動に特化しています。isearchでは検索文字列が多数マッチしたときに何度もC-sやC-rを押す必要があって手間がかかりますが、ace-jump-word-modeはそんな問題とはおさらばできます。また、ウィンドウが分割されていたとしても、画面に表示されているのであればウィンドウ切り替えなしで即ジャンプできます。筆者は導入後あっさり魅了されました。

リスト1でMELPAを使用可能にして、

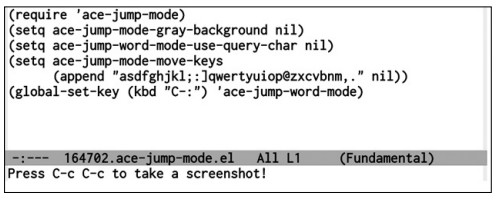
```
M-x package-install ace-jump-mode
```

でインストールし、**リスト3**のように初期設定を行います。筆者はC-oに割り当てていますが、

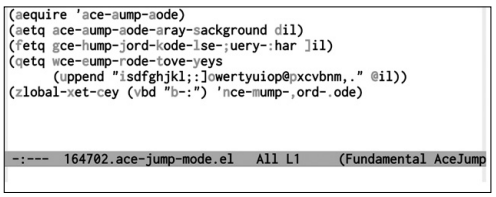
▼リスト3　ace-jump-word-modeの初期設定

```
(require 'ace-jump-mode)
(setq ace-jump-mode-gray-background nil)
(setq ace-jump-word-mode-use-query-char nil)
(setq ace-jump-mode-move-keys
     (append "asdfghjkl;:]qwertyuiop@zxcvbnm,." nil))
(global-set-key (kbd "C-:") 'ace-jump-word-mode)
```

▼図1　ace-jump-mode実行前



▼図2　ace-jump-mode実行後



ここではC-:に割り当てています。

使い方は簡単です。C-:を押したら単語の先頭に別色で文字[注2]が表示されるので(**図1→図2**)、移動したい場所の色文字をタイプしてください。

入力支援機能

後半はEmacsの強力な入力支援機能をいくつか紹介します。とくにdabbrevは、これなしでは生きていけないほど超強力です。

dabbrev-expand

入力支援機能の内、真っ先に紹介しておきたいのがこのdabbrev機能です。dabbrevとは動的略語展開のことで、長い文字列を補ってくれるものです。

Emacsを使っていると、どうしても同じ単語を何度も打ち込むことが多くなりますね。でも、毎回馬鹿正直にタイプすると、時間がかかる上に、タイプミスが起こりやすくなってしまいます。

そこで、長い単語を入力するときは先頭の数文字をタイプしてからM-/を押してみましょう。たとえばinterのあとにM-/を押してみると、internet、interesting、interactive、intervalなどの単語に補完されます。もし望みの単語でなければ繰り返しM-/を押してください。もし、行き過ぎてしまったらC-/で戻してください。

M-/がどのように補完されるかは状況に依存します。連続で押していくと、次の場所から探索されます。

❶カレントバッファのカーソル位置に一番近い単語

❷カーソル位置から離れた単語

❸ほかのバッファ

これは言葉で説明するよりも、実際に手を動かしてください。これを知ると本当に世界が変わります。初めて使ったときには、まるで魔法でもかかっているかのように適切に補完してくれることに驚くことでしょう。文章入力でもプログラミングでもあらゆる局面で使えます。

プログラミングにおいては言語固有の補完を使うのが普通ですが、一部の言語では正確な補完候補を求めるのが困難なケースがあります。たとえばRubyプログラミングで補完すべきメソッド名がわかっている場合は補完コマンドを使うのではなくてM-/で済ませてしまいます。

ただ、日本語においては単語の間に空白を入れないために相性が悪いです。お使いの日本語入力システムでカバーしてください。SKK(第4回で紹介)は過去に入力した見出し語を補完できるので便利です。dabbrevは英文やコーディングに絶大な威力を発揮します。

M-/を多用するようになると、そのうち打ちづらく感じてくることでしょう。押しやすいキーに各自割り当てると快適になります。次の設定例ではC-@に割り当てています。

```
(global-set-key (kbd "C-@") 'dabbrev-expand)
```

hippie-expand

dabbrev-expandは極めて強力なコマンドですが、この進化形と言えるコマンドが標準で存在

注2) 図2中、他より淡い色の文字がace-jump-modeの表示です。

▼リスト4　hippie-expandのデフォルト設定

```
(setq hippie-expand-try-functions-list
      '(try-complete-file-name-partially   ; ファイル名の一部
        try-complete-file-name             ; ファイル名全体
        try-expand-all-abbrevs             ; 略語展開(より良い方法があり、今はあまり使われない)
        try-expand-list                    ; 括弧に囲まれた内容(役立たない)
        try-expand-line                    ; 行そのもの(役立たない)
        try-expand-dabbrev                 ; カレントバッファでdabbrev
        try-expand-dabbrev-all-buffers     ; すべてのバッファでdabbrev
        try-expand-dabbrev-from-kill       ; キルリングの中からdabbrev
        try-complete-lisp-symbol-partially ; Emacs Lispシンボルの一部(役立たない)
        try-complete-lisp-symbol))         ; Emacs Lispシンボル全体(役立たない)
```

します。M-x hippie-expandです。M-/では開かれているバッファの中が補完候補になりますが、hippie-expandでは入力中のファイル名だったり、Emacs Lispのシンボルだったり、キルリングの中身からも走査してくれます。

hippie-expandはとても賢く、入力の状況に応じて空気を読んでくれます。とくにファイル名を入力しているときには先頭数文字だけ入力すれば適切な補完をしてくれるありがたいコマンドです。

使い方はM-/と同じで、数文字タイプしてから実行し、望みの結果と異なるときには再実行します。本稿では次のように、dabbrev-expandと同じC-@に割り当てていますが、dabbrev-expandに慣れてからhippie-expandに乗り換えればいいです。

```
(global-set-key (kbd "C-@") 'hippie-expand)
```

変数hippie-expand-try-functions-listはhippie-expandでどのように補完するかを細かく指定できます。補完の方法はtryから始まる関数で指定してあり、デフォルトでは**リスト4**のような設定になっています。上から順番に関数を実行していき、実際に補完が行われたときにhippie-expandの実行は終了します。好みに応じて削除したり順番を入れ替えたりできます。hippie-expandを実行すると「Using try-expand-dabbrev」などと表示されますが、どの補完が働いたのかを示しています。

ファイル名入力中にhippie-expandを実行すると、最初にtry-complete-file-name-partiallyが働き、最低限の補完が行われます。そして、再実行するとtry-complete-file-nameが働いて、存在するファイル名が実行するたびに出てくるようになります。たとえば、SD1404.pdfとSD1405.pdfが存在するときは、SDのあとにhippie-expandを実行するとSD140と補完されます。そのあとはファイル名の一部を入力するか、再びhippie-expandを実行するかで挙動が変わってきます(**図3**)。

try-expand-lineは同じ行を入力しようとします。たとえば「ab cd」という行が存在したときに「a」のあとにhippie-expandを実行すると「ab cd」が出てきます。

try-expand-listは括弧に囲まれた内容を入力しようとします。たとえば「(a b)」とどこかに書いてあるときに「(a」が「(a b)」になります。

hippie-expandはdabbrevにファイル名補完が付いたものとして使われることが多いと思われます。筆者の経験上、try-expand-lineとtry-expand-listのお世話になったことがありません。しかもdabbrevよりも先に実行されるのでdabbrevのつもりでhippie-expandを実行したら思わぬ結果に戸惑ってしまいます。

また、dabbrevの下にEmacs Lispシンボル補完が設定されていますが、Emacs Lispファイル以外でEmacs Lispシンボルを入力することはめったにありません。ChangeLogなどでシンボルを入力することはありますが、そのシンボルはdabbrevの時点で補完できます。なぜなら、そ

▼図3　hippie-expandの実行例

```
SD
↓ hippie-expand
SD140
↓ 5を入力してhippie-expand
SD1405.pdf

SD
↓ hippie-expand
SD140
↓ hippie-expand
SD1404.pdf
↓ hippie-expand
SD1405.pdf
```

のシンボルが書かれているEmacs Lispファイルをすでに開いているからです。

一般に、多機能ということには余計なものが含まれていたり、想定とは異なる挙動をしてしまうことが多々あります。自分の理解を超えた機能というのは必要ありません。筆者はhippie-expandのデフォルトの設定は複雑過ぎると考えています。ファイル名補完＋dabbrevで十分でしょう(**リスト5**)。

SKKを使う

dabbrev-expandやその進化形のhippie-expandは強力ですが日本語を苦手としています。日本語入力においては独自のノウハウがあります。

第4回で紹介した日本語入力システムSKKは単語変換だからこそ独自のメリットが存在します。特筆すべきはすぐに使える単語登録と英字変換です。SKK辞書は品詞情報がないため、見出し語と変換結果のテーブルにすぎません。つまり、好き勝手に単語登録しまくれるのです。

そして、英字変換は英数字を見出し語にできることです。たとえば、「code→コード」のような変換が行えます。「/code」を変換すればコードと出てきます。

この2つを組み合わせれば現在書いている文章に頻出する単語を登録してすぐに出せるようになります。たとえば、「日本語入力」という単語が頻出する場合には「/n」で出せるように単語

▼リスト5　hippie-expandの推奨設定

```
(setq hippie-expand-try-functions-list
      '(try-complete-file-name-partially
        try-complete-file-name
        try-expand-dabbrev
        try-expand-dabbrev-all-buffers
        try-expand-dabbrev-from-kill))
```

登録してしまえばいいのです。SKKでは真っ先に変換候補として出るのが最近使った単語なので、こういう単語登録は邪魔にはなりません。使わなくなった単語登録は候補の後ろへ追いやられるだけです。一時的に使う入力短縮のために積極的に単語登録できるのがSKKの強みの1つです。もちろんURLやメールアドレスを単語登録することだってできます。

さらに次の設定を加えると、見出し語入力中に最後に入力した見出し語が表示されるようになります。

```
(setq skk-dcomp-activate t)
```

最後に「日本語」と入力したときに再び「Ni」と入力すると、別の色で「にほんご」と出てきます。そのまま変換したいときはM-SPCを押します。これを知っているだけでもかなりタイプ数を減らせます。

終わりに

今回はカーソル移動と文字入力というテキストエディタの基本機能を快適に使うお話でした。少し難し過ぎましたか？　入力支援機能は、もっともっと高度で便利なものが存在しますが、今回はシンプルなものを取り上げました。それでもMigemoとdabbrevには驚かれるかと思います。

筆者はEmacsの週刊メルマガ[注3]を書いています。Emacsをもっともっと便利に使いたい、将来的にはEmacsの達人になりたいのならば登録お願いします。Happy Emacsing！SD

注3)　URL http://www.mag2.com/m/0001373131.html

シェルスクリプトではじめるAWS入門

――AWS APIの活用と実践

第6回　AWS利用環境の構築(補足:Billing関連IAMユーザの作成)

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

現在世界に8つあるリージョンのすべてでCloudTrailを利用できるようになったことが6月30日に公表されました[注1]。これにより、ようやく東京リージョンでもCloudTrailが使えるようになっただけでなく、どこのリージョンであるかを問わずAWSの操作ログを保存することが可能になりました。特に複数人で同一のAWS環境を運用している場合には、なるべくCloudTrailの設定をしておくようにしましょう。

設定はCloudTrailマネジメントコンソール[注2]からリージョンごとに行います。このときに次の2つの選択肢があります。

1. 全リージョンのCloudTrailログを1つのS3バケットに保存
2. 各リージョンごとにそれぞれ保存

1の場合は、CloudTrail設定画面の[Create a new S3 bucket?]欄で[No]を選択し、[S3 Bucket]のプルダウンメニューから既存のS3バケットを選択します。2の場合は、[Create a new S3 bucket?]欄では[Yes]を選択し、[S3 Bucket]欄に新しいS3バケット名を入力します。この場合、S3バケットはそのリージョンに作成されます。

検証程度にAWSを利用している場合は、API操作ログが一個所に保存されるため利便性を優先して1の方法を取ることが考えられます。本格的にAWSを利用しているのであれば、リージョンまたぎによる転送遅延やS3バケット共有による容量圧迫リスクを考慮して、2の方法を取るほうが良い場合もあるでしょう。

今回のテーマ

前々回までで、AWSアカウントでなければできないことは請求情報の変更や請求レポートの閲覧以外ほぼなくなりました。前回の設定でAWSアカウントのセキュリティについても強化が完了しました。さて、今回からは実際にAWS APIの話に……と思っていたのですが、本記事を執筆中の7月7日に「AWS Billingコンソールに対してIAMによる制御が可能になった」旨のアナウンス[注3]がAmazonからありました。

これにより「AWSアカウントでなければできないこと」がほぼなくなるため、今回は急遽予定を変更して、次の「IAMグループとユーザの作成」について解説します。

注1) URL http://aws.typepad.com/aws_japan/2014/07/cloudtrail-expands-again.html

注2) URL https://console.aws.amazon.com/cloudtrail/home

注3) URL http://blogs.aws.amazon.com/security/post/Tx2154FGFDNMQNP/Enhanced-IAM-Capabilities-for-the-AWS-Billing-Console

・請求情報だけにアクセスするためのIAMグループ(マネージャ向け。以下「請求レポートグループ」)
・支払い情報だけにアクセスするためのIAMグループ(経理担当者向け。以下「支払グループ」)

これに加えて、「サポート」にAWSマネジメントコンソールからアクセスするためのIAMグループの作成についても解説します。

請求情報に対するIAMアクセスのアクティブ化

請求情報および支払情報にIAMからアクセスするためには、AWSアカウントの設定で「請求情報に対するIAMユーザーアクセス」を許可する必要があります。

①言語設定を日本語に変更

AWSアカウントでログインし、アカウント設定画面[注4]にアクセスします。画面右下に言語選択のプルダウンメニューがあるので「日本語」を選択します。

②[請求情報に対するIAMユーザーアクセス]の編集

画面中段に[請求情報に対するIAMユーザーアクセス]と表示されている欄があるので、「編集」リンクをクリックします(**図1**)。

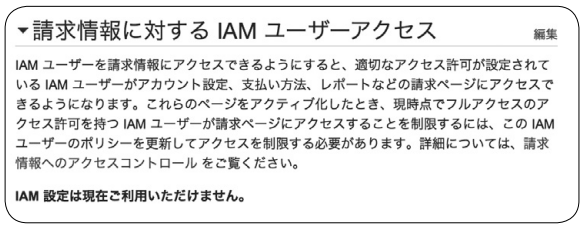
③情報の更新

[IAMアクセスのアクティブ化]にチェックを入れ、[更新]ボタンをクリックします(**図2**)。

④IAMユーザーアクセスのアクティブ化

更新すると「IAM設定は現在ご利用いただけません」と表記されていた一文が「請求情報に対するIAMユーザーアクセスがアクティブ化されます。」と表示が変わります。

▼図1　[請求情報に対するIAMユーザーアクセス]



▼図2　情報の更新

ここまでではAWSマネジメントコンソールでのみ可能な作業は完了です。これ以降はAWS CLIで作業を進めていきます(AWS CLIの設定については連載第3回の記事を参考にしてください)。

IAM全体の設定

連載第2回の解説では言及しませんでしたが、IAMを日常的に利用するうえで欠かせない次の2つの設定について、ここでは解説します。

・パスワードポリシー
……IAMユーザのパスワードに強度が弱い文字列を設定不可にする。

・アカウントエイリアス
……IAMユーザのログイン用ページのURLをわかりやすいものにする。

①パスワードポリシー設定

IAMユーザのパスワードポリシーを設定します。次のコマンド実行例ではIAMが提供する次のすべてのポリシーを適用し、さらにパスワード文字列を9文字以上とします。

注4) URL https://console.aws.amazon.com/billing/home?#/account

・記号(require-symbols)
・数字(require-numbers)
・大文字(require-uppercase-characters)
・小文字(require-lowercase-characters)
・ユーザ自身のパスワード変更許可(allow-users-to-change-password)

実際には次のようになります。

・コマンド:

```
$ aws iam update-account-password-policy --minimum-password-length 9 --require-symbols --require-numbers --require-uppercase-characters --require-lowercase-characters --allow-users-to-change-password
```

②アカウントエイリアス設定

アカウントのエイリアスを設定します。これは、IAMユーザのログインURLである"https://XXX.signin.aws.amazon.com/console"のXXXの部分についてわかりやすい文字列を指定できる機能です。標準では12桁のアカウントIDになっているので、利便性向上のためエイリアス設定をしておきましょう。

エイリアス名は世界でユニークである必要があります。ここでは例としてexampleをアカウントエイリアスとして設定します。

・コマンド:

```
$ aws iam create-account-alias --account-alias example
```

これで、"https://example.signin.aws.amazon.com/console"からIAMログインすることが可能になります。

請求レポートグループ/支払グループの作成

まず、請求情報にアクセスする2つのグループを作成しましょう。

2つのグループは、次の項目が異なるだけで作成手順は同じです。

・グループ名
・ポリシードキュメント

IAMグループの作成手順

①グループ名の決定

最初に、それぞれのグループ名を決めましょう。ここでは、次のとおりとします。

・請求レポートグループ:BillingReport
・支払グループ:Payment

②ポリシードキュメントの作成

次に、各グループのアクセス権限を定義するポリシードキュメントを作成します(**リスト1**、**リスト2**)。

> **NOTE**
> 本記事のポリシードキュメントでは、アクセスを許可する"Allow"定義のみサンプルとして掲載しています。AWSのポリシードキュメントでは、明示的なDeny、明示的なAllow、未定義(暗黙的なDeny)の順で適用されるため、確実にアクセスを拒否したい場合はDeny定義をポリシードキュメントに記述するようにしてください。前述のblog記事のサンプルにはDeny定義についても記述されているので参考にしてください。

IAMグループの作成とポリシー設定

①IAMグループの作成

IAMグループの作成は次のように行います。

・コマンド:

```
$ aws iam create-group --group-name BillingReport
$ aws iam create-group --group-name Payment
```

実行結果は、次に示すような形式で生成されます。

▼リスト1　請求レポートグループ用ポリシードキュメント(policy-BillingReport.json)

```
{
    "Statement": [
        {
            "Action": [
                "aws-portal:ViewUsage",
                "aws-portal:ViewBilling",
                "aws-portal:ModifyBilling",
                "ec2-reports:ViewInstanceUsageReport",
                "ec2-reports:ViewReservedInstanceUtilizationReport"
            ],
            "Resource": "*",
            "Effect": "Allow"
        }
    ]
}
```

・結果(BillingReportの例):

```
{
    "Group": {
        "Path": "/",
        "CreateDate": "2014-07-10T03:41:06.651Z",
        "GroupId": "AGPAJXXXXXXXXXXXXXXX",
        "Arn": "arn:aws:iam::XXXXXXXXXXX:group/BillingReport",
        "GroupName": "BillingReport"
    }
}
```

②IAMグループにポリシーを適用

同様に次のようにコマンド入力します。

・コマンド:

```
$ aws iam put-group-policy --group-name BillingReport --policy-name policyBillingReport --policy-document file://policy-BillingReport.json
$ aws iam put-group-policy --group-name Payment --policy-name policyPayment --policy-document file://policy-Payment.json
```

・結果:

```
(戻り値なし)
```

ここまでで、グループの作成は完了です。

▼リスト2　支払グループ用ポリシードキュメント(policy-Payment.json)

```
{
    "Statement": [
        {
            "Action": [
                "aws-portal:ViewBilling",
                "aws-portal:ViewPaymentMethods",
                "aws-portal:ModifyPaymentMethods"
            ],
            "Resource": "*",
            "Effect": "Allow"
        }
    ]
}
```

IAMユーザの作成

次にIAMユーザを作成します。ここでは各IAMグループに個別にIAMユーザを1つずつ登録していきます。bilユーザをBillingReportグループに、payユーザをPaymentグループに登録します。

①IAMユーザの作成

同様に次のようにコマンド入力します。

・コマンド:

```
$ aws iam create-user --user-name bil
$ aws iam create-user --user-name pay
```

次のような出力が得られます。

・結果(bilの例):

```
{
    "User": {
        "UserName": "bil",
        "Path": "/",
        "CreateDate": "2014-07-10T03:49:05.462Z",
        "UserId": "AIDAJXXXXXXXXXXXXXXX",
        "Arn": "arn:aws:iam::XXXXXXXXXXX:user/bil"
    }
}
```

②各IAMユーザのパスワードの設定

次のように入力し作成します。

・コマンド：

```
$ aws iam create-login-profile --user
-name bil --password 'XXXXXXXX'
$ aws iam create-login-profile --user
-name pay --password 'XXXXXXXX'
```

> **Note**
> XXXXXXXXの文字列は、さきほど設定したパスワードポリシーに従ったものにしてください。

bilを例とした場合、次のような出力が得られます。

・結果(bilの例)：

```
{
  "LoginProfile": {
    "UserName": "bil",
    "CreateDate": "2014-07-10T04:
04:39.068Z"
  }
}
```

IAMユーザのIAMグループへの登録

各IAMグループ、IAMユーザの作成が完了したので、次に各IAMユーザをそれぞれIAMグループに登録していきます。

入力するコマンドは次のようになります。

・コマンド：

```
$ aws iam add-user-to-group --group
-name BillingReport --user-name bil
$ aws iam add-user-to-group --group
-name Payment --user-name pay
```

ログイン確認

さきほど作成したURL[注5]からAWSマネジメントコンソールにログインします。

ログイン時に必要な情報は下記のとおりです。

・Account：アカウントエイリアス(あらかじめ値が入っている)

・User Name：ログインするIAMユーザ名

・Password：ログインするIAMユーザのパスワード

各ユーザがそれぞれ下記ページの情報を閲覧できることを確認します。

・bilユーザ：

-Billingダッシュボード(https://console.aws.amazon.com/billing/home?#/)

-レポート(https://console.aws.amazon.com/billing/home?#/reports)など

・payユーザ：

-Billingダッシュボード(https://console.aws.amazon.com/billing/home?#/)

-お支払方法(https://console.aws.amazon.com/billing/home?#/paymentmethods)

以上で「請求レポートグループ」「支払グループ」とそれぞれのIAMユーザの作成は完了です。各担当者(上司や経理担当者)にIAMユーザのログイン情報を伝えてさっそく利用してもらいましょう。

Billingマネジメントコンソールでの権限設定については、公式ドキュメント[注6]にいくつか事例が掲載されていますので参照ください。

サポート利用グループの作成

AWSマネジメントコンソールの「サポート」[注7]はAWSサポートチームに対する問い合わせのためのWebユーザインターフェースを提供しています。サポートレベルがビジネス以上の場合はTrusted Advisorというユーザ設定に対する自動チェックツールにもアクセスが可能です。

これらはAWS CLIからのアクセスも可能ですが、通常運用で利用するには作り込みが必要

注5) 例：https://example.signin.aws.amazon.com/console

注6) URL http://docs.aws.amazon.com/ja_jp/awsaccountbilling/latest/about/ControllingAccessWebsite.html

注7) URL https://aws.amazon.com/support/

となるので、24時間監視チームなど、「AWSプロダクトの操作はしないがAWSサポートチームへの問い合わせは行なうことがある」という立場の人達(以下「サポート利用グループ」)を想定したIAMグループを作成します。

①サポート利用グループ名の決定

最初に、サポート利用グループのグループ名を決めましょう。ここでは、"Support"とします。

②ポリシードキュメントの作成

次にサポート利用グループのアクセス権限を定義するポリシードキュメントを作成します(**リスト3**)。

③ログインして確認

これ以降のIAMグループ作成、IAMユーザ作成手順は前項の「請求レポートグループ」「支払グループ」と同じです。「サポート利用グループ」のメンバーとして例えば"sup"というIAMユーザを作成して、ログイン確認してみましょう。

④サポートセンター利用で注意すべき点

次の2点が挙げられます。

1.言語選択

日本語で問い合わせを行う場合は、必ず「言語選択」で「日本語」を指定してください。言語選択が英語になっていると、英語圏のサポートセンターにケースが送付されるらしく、日本語で再度起票ということになってしまいます。

2.メールアドレス

IAMユーザでケースを作成する場合は、更新通知するためのメールアドレスが必須事項になっています。あらかじめ通知先のメールアドレスを決めておきましょう。ケースに対して更新が行なわれるたびにメールで通知されます。

以上で「サポート利用グループ」とその所属IAMユーザの作成は完了です。24時間監視チームなどのメンバーにIAMユーザのログイン情報を伝えてさっそく利用してもらいます。

▼リスト3　サポート利用グループ用ポリシードキュメント(policy-Support.json)

```
{
    "Statement": [
        {
            "Action": [
                "support:*",
                "ec2:DescribeInstance*",
                "ec2:DescribeVolume*",
                "ec2:DescribeRegions"
            ],
            "Resource": "*",
            "Effect": "Allow"
        }
    ]
}
```

Note

本記事では、説明の便宜上各IAMグループに1つずつIAMユーザを作成しましたが、実運用ではIAMユーザは必ず担当者個々人に対して作成し、同一IAMユーザを複数人で使いまわすような運用はしないようにしてください。

次回は

今回までの設定で、組織上の経理(支払い管理)、マネジメント(請求管理)、実作業(Administrator)、サポート利用(24時間監視チーム)それぞれに必要なIAMユーザを作成しました。これによってAWSサービスにおけるrootアカウントとも言うべき「AWSアカウント」を使う機会がなくなったと言ってよいと思います。

次回からは、実際にAWS APIをシェルスクリプトで利用する方法の解説に進みます。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22回

bhyveにおける仮想シリアルポートの実装（その2）

Writer　浅田 拓也（あさだ たくや）　Twitter @syuu1228

はじめに

前回の記事では、PCアーキテクチャにおけるシリアルポートの仕様について解説しました。今回は、bhyveにおける仮想シリアルポートの実装について解説します。

bhyve における仮想シリアルポートの実装

bhyveに実装されている仮想シリアルポートは、PCI-LPCブリッジのエミュレーション（pci_lpc.c）と16550A UARTコントローラのエミュレーション（uart_emul.c）に分かれています。

これは、LPCバス接続のレガシシリアルデバイスのほかにPC接続のシリアルデバイスもサポートしているのでコードの重複を避けてUARTコントローラのコードだけを切り離しているためですが、今回はLPCバス接続のものだけを解説します。

PCI-LPC ブリッジのエミュレーションコード

PCI-LPCブリッジのエミュレーションコードは、次の2つの役割を担っています。

- PCIデバイスの1つとしてPCI-LPCブリッジを登録、OSからブリッジが発見できるようにする
- PCIデバイスエミュレーション経由でIOAPIC経由割り込みとI/Oポートハンドラを登録、UARTエミュレータが割り込みとI/Oポートを使えるようにする

では、PCI-LPCブリッジのコードをみていきましょう。

リスト1で紹介するpci_lpc_initは、PCI-LPCブ

▼リスト1　pci_lpc_init

```
static int
pci_lpc_init(struct vmctx *ctx, struct pci_devinst *pi, char *opts)
{
  if (lpc_bridge != NULL)
    return (-1);

  if (lpc_init() != 0)
    return (-1);
  /* initialize config space */
  pci_set_cfgdata16(pi, PCIR_DEVICE, LPC_DEV);
  pci_set_cfgdata16(pi, PCIR_VENDOR, LPC_VENDOR);
  pci_set_cfgdata8(pi, PCIR_CLASS, PCIC_BRIDGE);
  pci_set_cfgdata8(pi, PCIR_SUBCLASS, PCIS_BRIDGE_ISA);
  lpc_bridge = pi;
  return (0);
}
```

初期化コードの本体であるlpc_initを呼ぶ（後述）

PCIコンフィグレーション空間にデバイスID、ベンダID、クラスコードを書き込む。クラスコードとしてPCI-ISAブリッジの値を使用する

▼リスト2　lpc_init

```
static int
lpc_init(void)
{
  struct lpc_uart_softc *sc;
  struct inout_port iop;
  const char *name;
  int unit, error;

  /* COM1 and COM2 */
  for (unit = 0; unit < LPC_UART_NUM; unit++) {
    sc = &lpc_uart_softc[unit];
    name = lpc_uart_names[unit];

    if (uart_legacy_alloc(unit, &sc->iobase, &sc->irq) != 0) {
      fprintf(stderr, "Unable to allocate resources for "
          "LPC device %s¥n", name);
      return (-1);
    }

    sc->uart_softc = uart_init(lpc_uart_intr_assert,
          lpc_uart_intr_deassert, sc);

    if (uart_set_backend(sc->uart_softc, sc->opts) != 0) {
      fprintf(stderr, "Unable to initialize backend '%s' "
          "for LPC device %s¥n", sc->opts, name);
      return (-1);
    }

    bzero(&iop, sizeof(struct inout_port));
    iop.name = name;
    iop.port = sc->iobase;
    iop.size = UART_IO_BAR_SIZE;
    iop.flags = IOPORT_F_INOUT;
    iop.handler = lpc_uart_io_handler;
    iop.arg = sc;

    error = register_inout(&iop);
    assert(error == 0);
  }

  return (0);
}
```

COM1とCOM2の両方を初期化する

I/Oポート番号とIRQ番号をUARTエミュレータに通知する

割り込み通知（**lpc_uart_intr_assert**）／割り込み解除コールバック関数（**lpc_uart_intr_deassert**）を登録してUARTエミュレータを初期化する。**lpc_uart_intr_assert**は**ioapic_assert_pin**を呼び出して、割り込みをアサートする。そして**lpic_uart_intr_deassert**は**ioapic_deassert_pin**を呼び出して割り込みのアサートを解除する。どちらもUARTエミュレータから関数ポインタとしてコールされる

UARTのバックエンドデバイスを設定している。bhyveではUARTの接続先として標準入出力かttyデバイスを選択できる[注1]が、ここではbhyveの引数に渡された値に応じて接続先を切り替えている

UARTのI/Oポートをハンドルするようにbhyveに登録している。ハンドラとして**lpc_uart_io_handler**を指定している。このハンドラ関数はI/Oポートアクセスの方向に応じてUARTエミュレータの**uart_read/uart_write**を呼び出す

リッジがbhyve上で初期化され、メモリ空間・I/O空間に接続されるときに呼び出されるコードです。**リスト2**では、初期化コードの本体であるlpc_initを示しています。

UARTエミュレーションのコード

UARTレジスタの書き込みは、**リスト3**のようになります。

注1） 以降「ttyデバイス」として解説しますが、標準出力またはttyデバイスのどちらかが使用されます。

▼リスト3　uart_write

```
void
uart_write(struct uart_softc *sc, int offset, uint8_t value)
{
  int fifosz;
  uint8_t msr;

  pthread_mutex_lock(&sc->mtx);

  if ((sc->lcr & LCR_DLAB) != 0) {
    if (offset == REG_DLL) {
      sc->dll = value;
      goto done;
    }

    if (offset == REG_DLH) {
      sc->dlh = value;
      goto done;
    }
  }

switch (offset) {
  case REG_DATA:
    if (sc->mcr & MCR_LOOPBACK) {
      if (fifo_putchar(&sc->rxfifo, value) != 0)
        sc->lsr |= LSR_OE;
    } else if (sc->tty.opened) {
      ttywrite(&sc->tty, value);
    } /* else drop on floor */
    sc->thre_int_pending = true;
    break;
  case REG_IER:
    /*
     * Apply mask so that bits 4-7 are 0
     * Also enables bits 0-3 only if they're 1
     */
    sc->ier = value & 0x0F;
    break;
    case REG_FCR:

      if ((sc->fcr & FCR_ENABLE) ^ (value & FCR_ENABLE)) {
        fifosz = (value & FCR_ENABLE) ? FIFOSZ : 1;
        fifo_reset(&sc->rxfifo, fifosz);
      }

      if ((value & FCR_ENABLE) == 0) {
        sc->fcr = 0;
      } else {
        if ((value & FCR_RCV_RST) != 0)
          fifo_reset(&sc->rxfifo, FIFOSZ);

        sc->fcr = value &
           (FCR_ENABLE | FCR_DMA | FCR_RX_MASK);
      }
      break;
    case REG_LCR:
      sc->lcr = value;
      break;
    case REG_MCR:
      /* Apply mask so that bits 5-7 are 0 */
      sc->mcr = value & 0x1F;
```

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。書き込み値はuart_softc構造体のdllメンバ、dlmメンバへ書き込まれる

MCRレジスタにループバックビットが立っているときだけRX FIFOへデータが書き込まれる[注2]。それ以外の場合は**ttywrite**を呼び出して初期化時に指定されたttyデバイスへ1文字書き込む。**sc->thre_int_pending**に**true**をセットして、送信可能割り込みを有効にする

sc->ierにIERレジスタへの書き込みを保存

FIFOが有効から無効、無効から有効へ移行する場合は**fifo_reset**でRX FIFOをゼロクリアして状態をリセットする。この時に一緒に書き込まれた6～7ビットの設定でFIFO長が設定される。それ以外の場合で、FIFO有効ビットが0ならば**sc->fcr**をゼロクリアする。FIFO有効化ビットが1ならば**sc->fcr**に書き込まれた値を代入する。FIFO有効化ビットが1かつRX FIFOリセットビットが1の場合には、ステートが移行する場合と同様に**fifo_reset**でRX FIFOをゼロクリアして状態をリセットする

sc->lcrにLCRレジスタへの書き込みを保存

sc->mcrにMCRレジスタへの書き込みを保存し、**sc->msr**へMSRレジスタの更新された値を構築する

次ページに続く

注2）書き込んだデータが読み込みで取り出せるようになる→ループバック

リスト3の続き

```
      msr = 0;
      if (sc->mcr & MCR_LOOPBACK) {
        /*
         * In the loopback mode certain bits from the
         * MCR are reflected back into MSR
         */
        if (sc->mcr & MCR_RTS)
          msr |= MSR_CTS;
        if (sc->mcr & MCR_DTR)
          msr |= MSR_DSR;
        if (sc->mcr & MCR_OUT1)
          msr |= MSR_RI;
        if (sc->mcr & MCR_OUT2)
          msr |= MSR_DCD;
      }


      if ((msr & MSR_CTS) ^ (sc->msr & MSR_CTS))
        sc->msr |= MSR_DCTS;
      if ((msr & MSR_DSR) ^ (sc->msr & MSR_DSR))
        sc->msr |= MSR_DDSR;
      if ((msr & MSR_DCD) ^ (sc->msr & MSR_DCD))
        sc->msr |= MSR_DDCD;
      if ((sc->msr & MSR_RI) != 0 && (msr & MSR_RI) == 0)
        sc->msr |= MSR_TERI;


      sc->msr &= MSR_DELTA_MASK;
      sc->msr |= msr;
      break;
    case REG_LSR:
      break;
    case REG_MSR:
      break;
    case REG_SCR:
      sc->scr = value;
      break;
    default:
      break;
  }

done:
  uart_toggle_intr(sc);
  pthread_mutex_unlock(&sc->mtx);
}
```

LSRは読み込み専用レジスタなので何もしない

sc->scrにSCRレジスタへの書き込みを保存

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デアサートする。割り込みのアサート／アサート解除にはUARTエミュレータ初期化時に渡されたlpc_uart_intr_assert、lpc_uart_intr_deassertを使用する

tty デバイスへの文字列出力

ttyデバイスへの文字列出力は、DATAレジスタへの1文字書き込みからttywriteを経由し、ttyデバイスへwirteで直接書き込まれることによって行われます。

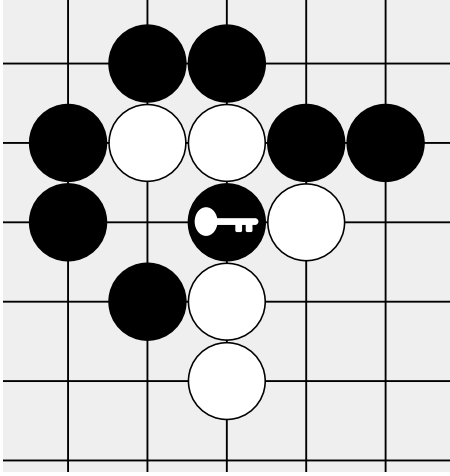
まとめ

今回はbhyveの仮想シリアルポートのソースコードのうち、LPC-PCIブリッジとUARTコントローラのレジスタへの書き込みハンドラを解説しました。

次回はUARTコントローラのレジスタからの読み込みハンドラとttyのイベントポーリングハンドラを解説します。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

【第十三回】動的メモリアロケーションの落とし穴

ここ2回に渡り特別編として、OpenSSLの脆弱性「OpenSSL Heartbeat Buffer Overread」の話題を取り上げました。今回は、その問題の背景の1つであった動的メモリアロケーションについて、プログラムを書く立場から解説を行います。

前提

動的メモリアロケーションはセキュリティだけではなく、もともとC言語プログラミングで最もやっかいなバグを発生しやすい部分です。説明されてもわかりづらいことが、さらにバグの温床となりやすいという悪循環を生んでいます。

しかし、OpenSSL Heartbeat Buffer Overreadのようなコードが生み出されたことを考えると、「難しいから」「わかりづらいから」といって、ここを避けて通っても良い結果を生まないことは容易に想像できます。今回は良い機会ですので、動的メモリアロケーションの落とし穴を取り上げたいと思います。

なお、本文でのmalloc()は、GNU/Linux上においてデフォルトで使われているglibcのmalloc()を前提として説明を進めていきます。

バグと脆弱性との境界線

まず議論を始める前に、ただの「バグ」と「脆弱性」の違いを考えてみたいと思います。

本稿で扱う「バグ」とは、プログラム内のミスにより、本来の意図した処理を行わない、つまり、プログラムが正常に動かないこと、またその原因です。

また、「脆弱性」とは、第三者が意図的にそのバグを発現させることができ、それによりシステムが機密性を失ったり、あるいは可用性を失ったりしてしまうこと、またその原因です。

たとえば、サーバプログラムがバグでクラッシュして停止してしまっても、そのバグを意図的に発現させられないなら、単純に「品質の悪いソフトウェア」というカテゴリに入ってしまいます。しかし、このバグを、意図的に発現させることができるならば、その行為は「サービス不能攻撃」であり、そのプログラムは「脆弱性を持つソフトウェア」となります。広く使われているものであれば、JVN(Japan Vulnerability Notes)のような脆弱性を管理する枠組みに組み入れて、対処する必要が出てきます。

セキュアコーディング

情報セキュリティのチームとして世界で最も古い歴史を持つチームの1つで、かつ先端的な活動をしているカーネギーメロン大学ソフトウェア工学研究所のCERT/CCが、安全なコーディング基準「セキュアコーディング」[注1]を広めようとしています。セキュアコーディングのチェックすべきポイントとして、メモリ管理の項目もあり、動的メモリアロケーションについて言及しています。

日本版もJPCERT/CCからオンラインで公開されているので、プログラマであれば動的メモリアロ

注1) CERT/CC、"CERT C Coding Standard" 08.Memory Management(MEM) **https://www.securecoding.cert.org/confluence/pages/viewpage.action?pageId=437**
日本語版はJPCERT/CCのサイトで読むことができる。「CERT C セキュアコーディングスタンダード」 **https://www.jpcert.or.jp/sc-rules/#c08**

ケーションだけでなく、ぜひ、全体に目を通してみてください。

UNIXとC言語とライブラリ

最初、アセンブラで書かれていたUNIXカーネルを書き換えるために、C言語は設計されました。コンパイラが作りやすいシンプルな構造を持った言語です。C言語の設計者であるDennis M. Ritchie氏が書いた“The Development of the C Language”という文章の中には、「FORTRAN、PL/IやAlgol 68も検討したが、仕様や必要とするリソースが大き過ぎる問題があった」といったことが書かれています。

さて、C言語にはUNIXの機能をフルに使うためのプログラミングライブラリが用意されています。その1つが動的メモリアロケーション関数malloc()です。これは現在では、IEEE Std 1003.1-2001で定義されています。ここには動的メモリアロケーション関数として**リスト1**の4つの関数が定義されています。各関数の機能は次のとおりです。

- **malloc()：メモリ領域をアロケーションする関数**
- **free()：不必要なメモリアロケーションを再利用できるように解放する関数**
- **calloc()：ゼロクリアしたメモリ領域をアロケーションする関数**
- **realloc()：すでにアロケーションされたメモリ領域を拡張する関数**

今回は話を絞り、malloc()とfree()の2つについて説明します。malloc()は、ほしいメモリのサイズを与えると、その領域を確保して、そのポインタを戻します。free()は、malloc()で確保した領域のポインタを与えると、その領域を解放します。

中身を理解しなくても、APIや動作だけを知っていれば、とりあえずプログラムは書けると思います。しかし、セキュリティのことを考えた場合、一歩踏み込んで理解すべき点（あるいは、疑問点と言ってもいいのかもしれません）があります。まず、次の2点を考えるところからスタートしてみます。

- **malloc()領域はどこからやってくるのか**
- **free()の「解放」とはどういう意味か**

◆リスト1　IEEE Std 1003.1-2001で定義されている動的メモリアロケーション関数

```
#include <stdlib.h>

void *malloc(size_t size);
void free(void *ptr);
void *calloc(size_t nmemb, size_t size);
void *realloc(void *ptr, size_t size);
```

メモリ領域の確保

ここではmalloc()の基本的なモデルを説明します。malloc()のメモリ領域は、要求されたサイズがすでにユーザ領域として確保しているメモリ領域から割り当てることができれば、そこから切り出されます。足りなければ、システムから新しいメモリ領域を割り当ててもらうために、システムコールを呼び出し領域を確保します。

足りない場合、古典的なUNIXではbrk/sbrkというシステムコールを呼び出します。これはヒープエリアを拡張するシステムコールです。しかし、システムコールbrk/sbrkは標準規格であるIEEE Std 1003.1-2001からすでに外されています。今日では、GNU/Linuxとglibcのmalloc()の組み合わせでは、メモリ領域はmmap()を使って新たに確保していると理解しておいたほうが良いでしょう。

リスト2は、システムコールmmap()を使って8KBのメモリ領域を確保している例です。mmap()は、ファイルをメモリのようにマップするためのシステムコールです。これによりメモリもファイルも同一のアクセス方式で処理できるようになります。

この考え方は単一レベル記憶と呼ばれ、アイデアはUNIXより以前に設計されたMulticsというOSにすでに取り入れられています。IBMのミニコンOSでは古くから用意されていた機能ですが、UNIXでは4.3BSD以降に取り入れられました。

リスト2では、引数でMAP_ANONYMOUSを指

◆リスト2　mmapでメモリ領域を確保する例

```
char *mm;
mm = (char *)mmap(0, 8096, (PROT_READ | PROT_WRITE), ➥
MAP_ANONYMOUS | MAP_PRIVATE, -1, 0);
```

定しているため、具体的なファイルにはマップせず、仮想記憶の空間からメモリ領域を用意します（コラム参照）。また、MAP_PRIVATEを指定しているので、自分しかアクセスできません。

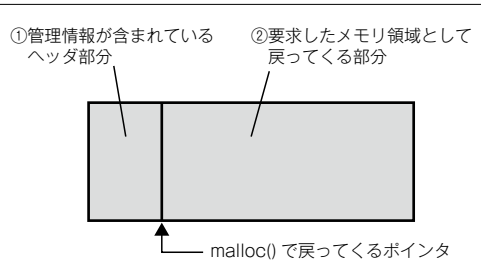
malloc()の役割

malloc()は、このような形で内部に確保された大きなメモリ領域から、要求に応じて細分化し提供します。malloc()で得られたメモリ領域は、より大きいユーザのメモリ領域の一部分です。これは、得られたメモリ領域を越えて、内部で確保したメモリ領域のどの部分でもアクセスできることを意味します。

次に重要な点は、malloc()が与えるメモリ領域には内部的に、管理情報が含まれたヘッダが付けられています（**図1**）。これはメモリ領域が不必要になったときに、free()で解放できるようにするためのものです。ヘッダ部分にはmalloc()で切り出した複数のメモリ領域を管理するための情報が入っており、内部的に整合性を持たせています。

もし、この部分を壊すとmalloc()全体の整合性が取れなくなり、指し示しているアドレスなども誤ったものになり、結果としてプログラム自体のクラッシュを引き起こす可能性があります。いつクラッシュしてもおかしくないという意味では、「可能性」という表現では弱く、本質的には「バグ」と呼ぶべきかもしれません。これを外部から発現させられるなら、その用法は「サービス不能攻撃」と呼ばれ、そのバグは「脆弱性」と呼ばれることになります。

◆**図1　malloc()で得られるメモリ領域の内訳**



◉mmap()で仮想記憶を利用するときの注意点

mmap()で仮想記憶を利用するときに、セキュリティ的に1つ注意してほしいのは、仮想記憶である以上、何かの拍子にスワップファイル[注2]に秘密情報を書き出す可能性があるということです。最近のディストリビューションであれば（GNU/Linuxだとディストリビューションが多過ぎて筆者が知らないものも多々ありますが）、インストール時にスワップファイルを暗号化するオプションがあったり、デフォルトで暗号化するようになっていたりするはずですので、それを有効に活用してください。

free()の役割

free()の役割は、多くの場合「メモリの解放」という表現で説明されます。ですが、より正確に表現すると、「不必要になったメモリ領域を再利用するために、再利用リストに登録する」ということになります。そして次にmalloc()をするときに、その再利用リストに適当なものがあれば、それを使います。なければ、新たに内部保留したメモリ領域から必要なサイズの領域を切り出します。もし、内部保留している領域も足りなくなったら、mmap()を使ってさらにメモリ領域を確保し、その中から用意します。

free()したとき、中身をクリアする、といったことはしません。ですからmalloc()をしてメモリ領域に何かを書き込み、そののちにfree()をしても、メモリ領域の中身はそのまま残っています。

malloc()で起こりがちなバグとその対処法

malloc()を使って、誰でも一度はやってしまった経験のあるバグは、**リスト3**のようなものでしょう。

まず、7行目でlの値は14となります。8行目でmmの領域は14バイト確保されています。9行目でmm[14]の場所に値0をセットします。そして10行

注2）仮想記憶において、メインメモリの容量が不足したときに、ハードディスク上に割り当てられる領域のこと。この領域をメモリに見立てて使用する。

目でmmを解放しています。どこが間違いかわかるでしょうか？

C言語の配列はゼロオリジンです。つまり、本来mmは、mm[0]～mm[13]の範囲しか割り当てられていません。ですから、mm[14]の場所に値0を設定しているのは不正です。ですが、C言語の配列は（ポインタも）、あるアドレスを指し示しているだけですので、チェックせず、そのまま値を書き込めてしまいます。

問題を見つけるのは難しい

malloc()とfree()を繰り返していると、内部で確保していた大きなメモリ領域の中で、利用、再生が繰り返されます。フラグメント（断片化）していき、どんどん虫食い状態になっていきます。そんな状態だと、次にmalloc()をしたとき、どこのメモリ領域が使われるかは、誰も予測がつかなくなります。

先ほどのような領域を侵害して情報を破壊している個所を、ソースコードから追いかけていくのは、至難の技です。まだUNIXが定着していなかった80年代後半に、研究会レベルの論文ではありますが、「いくらソースコードをチェックしてもmalloc()の問題は見つからなかった。malloc()のライブラリの問題と結論づけられる。UNIXのライブラリは安定していない」と書かれている論文を、筆者は目にしたことがあります。

GNU/Linuxの環境だと、簡易版のチェックで良ければ、mcheck()というmalloc()の状況をチェックする関数があります。わかりやすいように、先ほどのコードの問題個所でmcheck_check_all()を呼び出してみます。

◆リスト3　malloc()に関連する典型的なバグ

```
 1 #include <stdlib.h>
 2 #include <string.h>
 3 main() {
 4   char sdstr[]="SoftwareDesign";
 5   char *mm;
 6   int l;
 7   l = strlen(sdstr);
 8   mm = (char *)malloc(l);
 9   mm[l]='\0';
10   free(mm);
11 }
```

```
（... 省略 ...）
 9  mm[l]='\0';
10  mcheck_check_all();  ←追加
（... 省略 ...）
```

コンパイル時に-lmcheckを付けます。できあがった実行ファイルa.outを実行すると、**図2**のようなエラーメッセージが出て停止します。誤った値を入れたあとのmcheck_check_all()を呼び出したところでプログラムがストップしてしまいます。

実際のgdb（The GNU Project Debugger）を使ったデバッキングでは、ソースコードの6～7行目の部分にmcheck_check_all()の1行を加えます（任意の場所で呼び出す準備のため）。

次にgdbでステップ実行していき、mm[l]='\0'の行を実行したのちに、gdbのコマンドcallで関数mcheck_check_all()を呼び出します。そうするとSIGABRTをキャッチしプログラムが停止し、先ほどのメッセージが現れます（**図3**）。こうすることでデバッグ文をいちいちコードの中に用意せずとも、任意の行でmcheck_check_all()を実行し、チェックすることができます。

mcheck()の利用方法は、詳しくはマニュアルに譲るとして、このような関数を活用すればソースコードを目で追ってチェックするよりも、ずいぶんと良い結果を生むと思います。

ただ、この場合も、値を挿入するならばチェックできますが、OpenSSLのHeartbeat Buffer Overreadのように値を参照するだけの場合はチェックできません。

ここではmcheck()を紹介しましたが、もちろん

◆図2　mcheck()を付けて実行した例

```
$ cc -lmcheck -g foo.c
$ ./a.out
memory clobbered past end of allocated block
Aborted (core dumped)
```

◆図3　gdbを使った際のエラーメッセージ

```
(gdb) call mcheck_check_all()
memory clobbered past end of allocated block
Program received signal SIGABRT, Aborted.
（... 省略 ...）
```

商用のRational PurifyPlus[注3]のような多機能で使い勝手の良いメモリアロケーション専用のデバッグ環境を使うのも良い選択だと思います。

free()前に機密情報はクリアする

malloc()で獲得したメモリ領域を使い終わったら、free()を使って解放します。前述のとおり、動作としては、そのメモリ領域を再利用リストに戻します。次にmalloc()が呼ばれたとき、再利用リストを確認し、そこに利用可能なものがあれば再利用リストから取り出し、そのメモリ領域を使います。このとき、メモリ領域には以前の古いデータが、まったく手つかずのままで残っています。

一般的には、「malloc()で得たメモリ領域の値は不定である。ゼロクリアしたメモリ領域を利用するには、calloc()を使う」と説明されていると思います。これは、見方を変えれば、「malloc()で得たメモリ領域には以前に利用したデータが含まれている」ということです。また、free()したのち、再度、malloc()で使われるまでは、メモリの中に秘密情報を保持したまま再利用リストに登録され続けている、ということでもあります。

たとえば、パスワードや秘密鍵、あるいは復号処理のときに使う各種パラメータなどが、すでに不必要になっているのに保持されている、ということです。これらのデータが何らかの拍子で外部に漏れる可能性は否定できません。

そこで、秘密情報を扱ったメモリ領域は不必要になった時点でクリアし、それからfree()するといったプログラミングスタイルが必要になります。

動的メモリ確保のライフサイクル

malloc()で動的にメモリ領域を取得し、そこを利用し、不必要になればfree()で解放する、というのがメモリ領域のライフサイクルです。不必要になったにもかかわらず、free()をせずにそのままにしていると、メモリ領域が再利用されず、新しいメモリ領域がどんどん使われていき、無駄にメモリを消費してしまいます。このことを一般に「メモリリーク」と呼んでいます。しかし、「漏れる(leak)」というよりは「無駄にしてしまう／浪費してしまう(waste)」といったほうが、適切な表現だと思います。

その一方で、まだ使っているメモリ領域をfree()してしまうというのもありがちなバグです。C言語のポインタは単純にアドレスを指し示すだけですので、そのポインタが示している領域が有効であるかどうかは自明ではなく、自分のプログラム側で注意深く設計し、実装しなければなりません。

リスト4では、10～11行目でmalloc()で獲得したメモリ領域mmの(1オリジンで数えて)5バイト目と10バイト目を2つのポインタに入れています。pとqがまだ使っているにもかかわらず、13行目でmmをfree()してしまいます。そして、14～15行目でpとqにまたアクセスします。もちろん、14～15行目はバグです。

この小さなプログラムだと簡単にバグだとわかりますが、見通しの悪い大きなプログラムでメモリ領域をあちこちから参照している場合、どこで、どのタイミングで、どういう具合に使われているかを確実に把握するのは、たいへんに難しいと言えます。

その状況で、まだ利用しているエリアをfree()してしまえば、もちろんそれはバグです。free()したあとにmalloc()を行い、そのメモリ領域が再利用さ

◆リスト4　解放後のメモリ領域を使うバグ

```
 1 #include <stdlib.h>
 2 #include <string.h>
 3 #include <stdio.h>
 4 main() {
 5   char sdstr[]="Software Design";
 6   char *mm;
 7   char *p,*q;
 8   mm = (char *)malloc(strlen(sdstr));
 9   strncpy(mm,sdstr,strlen(sdstr));
10   p = &mm[4];
11   q = &mm[9];
12   printf("%s\n",mm);
13   free(mm);
14   printf("%s\n",p);
15   printf("%s\n",q);
16 }
```

注3) Rational PurifyPlus　http://www-06.ibm.com/software/jp/rational/products/purifyp/

れてしまうと、今度は、2つ、あるいはそれ以上の意味の違うポインタが同じメモリ領域を指し示し、そこのデータを参照したり、書き換えたりするわけですから、これはもう、どんな副作用が出るかは予測がつきません。また、どこで書き換えているかといったことを探すのは、容易ではありません。

malloc()の失敗

ここまでのmalloc()のサンプルコードでは、説明を簡略化するために、malloc()が失敗したときのコードはいっさい入れていません。malloc()を呼び出して失敗する確率は低いですが、それでもmalloc()が失敗しないわけではありません(その場合、NULLポインタを返します)。まれであっても何らかの理由で発生する可能性はあるので、その際に必要な適切なエラー処理をきちんと入れるべきです。

動的メモリアロケーションライブラリのバリエーション

ここまでGNU/Linuxのデフォルトライブラリであるglibcのmalloc()を前提に説明してきましたが、オープンソースの動的メモリアロケーションライブラリとして、ほかのものを使うこともできます。たとえば、Googleはtcmallocを公開していますし、FreeBSDはjemallocを採用しています。これらは「スレッド性能が良い」「より効率的にメモリ領域を利用する」「デバッキングやチューニングがより楽である」といったアドバンテージがあります。

プログラミングの面では、malloc()と引数などは同じに作ってあり、代替の動的メモリアロケーションとして、あとからリンクするライブラリを変更することも可能です。もちろん、これらはglibcのmallocとは内部データ構造も実装もまったく違うものです。

アプリケーションは独自にこれらの動的メモリアロケーションを使うことが可能ですし、実際に使われています。たとえば、Google Chromeはtcmallocを利用し、Mozilla Firefoxはjemallocを利用しています。Google Chromeで"chrome://tcmalloc"のURLにアクセスすると、Google Chromeのmalloc()の利用状況が表示されます(**図4**)。

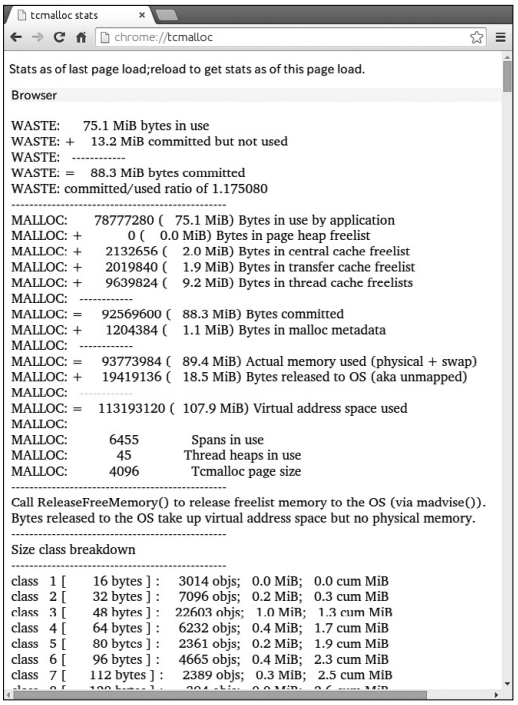
必要に応じてglibcのmalloc()ではなくtcmallocやjemallocを組み入れるという選択肢も考慮に入れるべきではないかと思います。

セキュアプログラミングの本質とは

malloc()の例を見て、すでにお気づきかと思いますが、セキュリティの問題を抱えるというのは、ソフトウェア品質として問題点を抱えているということです。セキュリティの問題を解決するとは、ソフトウェア品質を向上させるということです。

「抜けのないロバストな(しっかりした)プログラミングコードにすること」「正しい動作を行うプログラムを作成すること」「バグのないプログラムを作ること」といった当たり前のことを、当たり前にすることが、セキュアプログラミングの本質なのではないでしょうか。SD

◆**図4　Chromeの内部でtcmalloc()の状態を表示する**



第50回 省電力なアプリ開発のために知っておきたいこと

モバイルデバイス初のオープンソースプラットフォームとして、エンジニアから高い関心を集めるGoogle Android。いち早くそのノウハウを蓄積したAndroidエンジニアたちが展開するテクニックや情報を参考にして、大きく開かれたAndroidの世界へふみだそう！

神山 剛　KAMIYAMA Takeshi
㈱NTTドコモ 先進技術研究所

はじめに

Android端末をはじめとするスマートフォンの販売開始以降、依然として「電池の持ち」は、エンドユーザのスマートフォン利用における満足度を決定づける重要な一要因にあります。フィーチャーフォン（いわゆるガラケー）とは異なり、スマートフォンを構成するハードウェアコンポーネントは高性能化・多機能化が進み、さらにアプリはこれらのリソースをたいへん自由度高く利用できるようになりました。一方で、電池というリソースには限りがあることから、電池持ちの観点をふまえてアプリを作る・動かすことが重要視されるようになっています。

本稿では、電池にやさしい省電力なアプリを作るためのヒントとして、とくに、アプリ開発者目線では見えないスマートフォンの特徴・挙動と電力消費について解説します。アプリを設計される際の参考情報となれば幸いです。

何が電池を食うのですか?

筆者がこれまでスマートフォンの省電力化に関する仕事をするなかで、本当によく聞かれる質問が、「何が一番電池を食うのですか？」でした。正直、「これです」と端的に答えることができなくて本当に困ります。アプリや使い方で大きく変わるのでなんとも、としか言いようがありません。

図1は、あるスマートフォンを構成する各ハードウェアコンポーネントがそれぞれフル稼働したときの消費電力です。ご覧のとおり、現在のスマートフォンでは常にもっとも支配的なものというのはなく、CPU、無線、GPU、GPS……等々（とうとう）、消費の大きなコンポーネントが複数存在することがわかります。前述したように、これらは各コンポーネントの最大消費電力ですので、実際には稼働状態に応じて変動するわけですが、つまり、これらの組み合わせや度合いを最終的に決定づけるのが、アプリとその使い方なのです。

では、アプリによって電池消費の主要因が異なる例を示したいと思いますが、その前に、ご存じの方には不要ですが、電池の消費量の考え方について説明します。

スマートフォン用に限らず、電池にはそこから利用可能なエネルギー量を示す値として○Ah（電流量）、または○Wh（電力量）といった表示がされています。一方、電流（A）または電力（W）は、簡単にいうと消費される瞬間のエネルギーの大きさを示す値になりますので、消費された時間をかけることで電流量または電力量が求められます（**図2**）。ここでご理解いただきたいのは、アプリによる電池消費への影響を考える際、単に“消費電力の大きなコンポーネントを使うか否か”というだけでなく、想定するアプリの利用シナリオを通した“期間での消費電力量”が重要

だということです。

図3はいくつか実際のアプリを所定の操作シナリオで動作させたときの電流量を、コンポーネントごとの内訳と共に示したものです。「Offset」とあるのは後述しますのでここでは除外して見てください。前述したように、特定のコンポーネント分が際立っているケースもあれば、消費量の組み合わせが微妙に異なるパターンもあり、電池を食う要因はアプリとその使い方によってケースバイケースであることがわかります。

省電力なアプリ開発のためのヒント

省電力なアプリ開発のためのヒントとして、とくにアプリ開発者の立場では見えにくいスマートフォンの電力消費の特徴についてご紹介します。

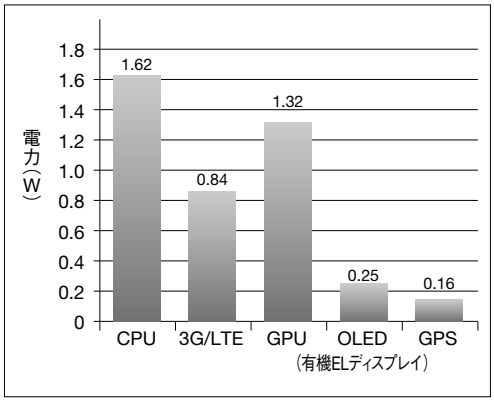
モバイル無線IF（3G/LTE）の電力消費

市場にあるアプリの大部分はモバイル環境でのデータ送受信を伴うものであると思いますが、ここでは3GやLTEといったモバイル無線IFを使用したデータ送受信の電力消費の特徴について解説します。

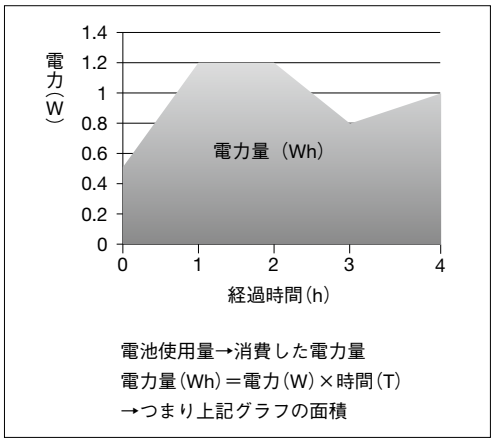
通常のアプリ開発では、データ送受信部分を設計する際、皆さんどこまでのプロトコルレイヤを意識しているでしょうか？

たいていはあらかじめ用意されたAPIを利用してhttpなどでデータ送受信部を実装するでしょうし、サーバを含めたシス

▼図1　ハードウェアコンポーネントの最大電力

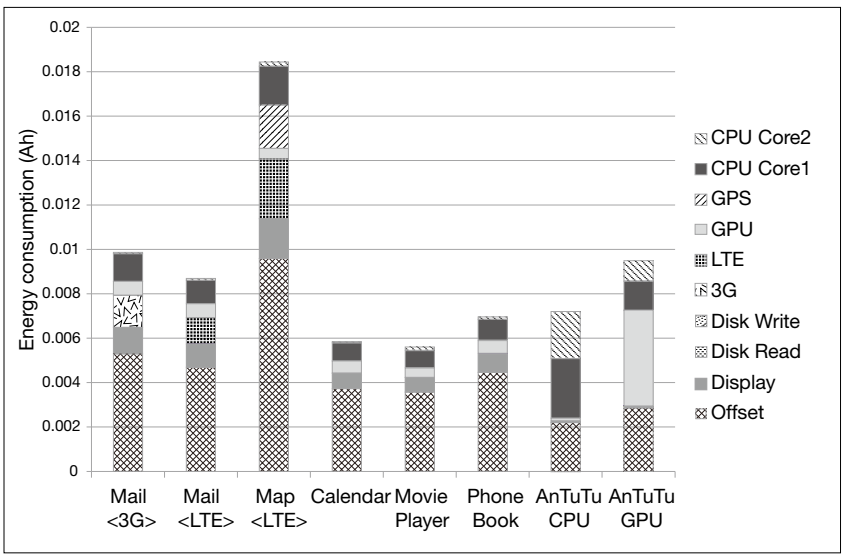


▼図2　電力と電力量の考え方



電池使用量→消費した電力量
電力量（Wh）＝電力（W）×時間（T）
→つまり上記グラフの面積

▼図3　アプリ使用時の電力量（電流量）



テム全体の設計でもTCP/IPあたりまでを考慮されているのではないかと思います。電力消費の観点からも、省電力を意識して、「送受信データを圧縮する」「キャッシュを活用する」などして送受信量を減らすといった工夫をされている方もいらっしゃると思います。

しかし、実はここにまだ落とし穴があるのです。それは、モバイル無線環境でデータ通信を確立するために必要な、3G/LTE特有のプロトコルの特徴にあります。

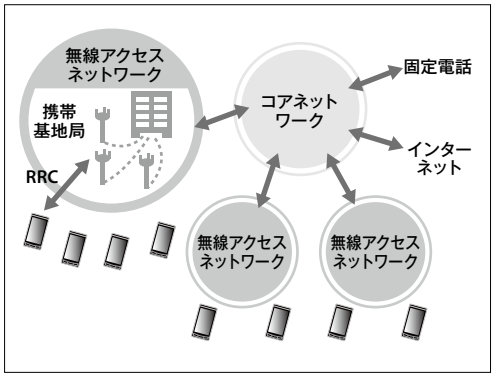
RRC (Radio Resource Control)

まず、できるだけ簡単に、この特徴について解説していきます。モバイル無線通信では、データ通信を確立するためにRRC(Radio Resource Control)と呼ばれるプロトコルが3GPPで規定されています。

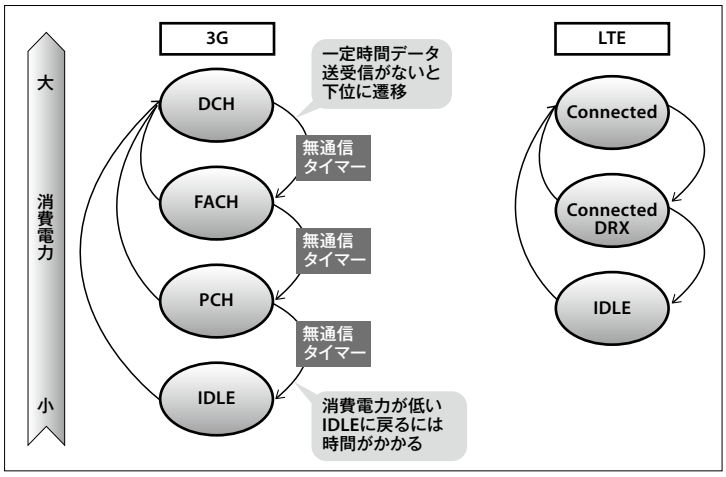
自宅やオフィスでの光回線、カフェなどの公衆無線LANなど、一定の閉じた空間でのインターネット接続とは異なり、モバイル環境での利用は広範囲な移動を伴うことが前提となっていますので、**図4**のように、端末は複数の基地局との無線接続を切り替えながら、データ送受信を維持しなければなりません。RRCは端末と基地局間の無線通信の確立・開放など、無線リソースを制御するためのプロトコルとして動作しています。誌面に限りがありますので、RRCについて詳細な説明は省略しますが、ここで注目いただきたいのは、データ通信の際の端末・基地局間での接続状態(RRC state)についてです。

RRCでは、端末・基地局間の無線リソースの制御に複数のモードがあり、**図5**のように各モードに対応したRRC stateが定義されています。たとえば3Gの場合、何もデータ送受信なく基地局とのコネクションが解放された状態がIDLEにあたりますが、アプリなどからのデータ通信要求に応じて、データ送受信を行うためのDCH(Dedicated Channel)に遷移します。なんらかのデータ送受信が終わり、一定時間、無通信状態が継続すると、FACH(Forward Access Channel)など下位のstateに遷移し、最終的にはIDLEに戻るというのが基本的な流れです。なお、下位のstateへの遷移契機となる無通信時間は、stateごとにタイマーとして個別の値(だいたい数秒〜数十秒)が設定されており、実際にはキャリアによって独自の値が設定されています。

▼図4　モバイル無線通信システム



▼図5　RRC state遷移



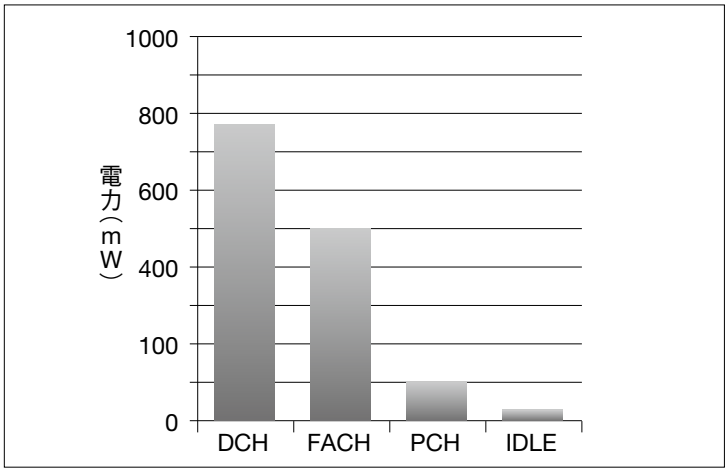
モバイル無線IFの消費電力

モバイル無線IFの消費電力は、**図6**の例のように、DCH、FACHといった上位のstateほど大きな電力を消

費する傾向にあり、IDLEではほぼゼロに近い値になります。

ここでのポイントは、モバイル無線IFの消費電力は、どのRRC stateにいるかでほぼ決まってくる、ということです(もちろん、実際には電波環境なども影響します)。これは言い換えると、アプリ的には何もデータ送受信していなくても、DCH、FACHなどのstateにいる限り、無通信タイマーが満了するまでデータ送受信中とほぼ同等の電力を消費し続けてしまうということです。「何もデータを送ってないのに、なぜ電力を食うんだ?」と違和感を感じられる方もいるでしょう。これは、アプリが扱うデータ送受信とは別の、RRCによる無線リソース制御のための制御信号のやり取りなどが、各stateで行われているためです。

▼図6　RRC stateごとの消費電力



▼図7　無通信タイマーによる状態遷移のタイミングと消費電力

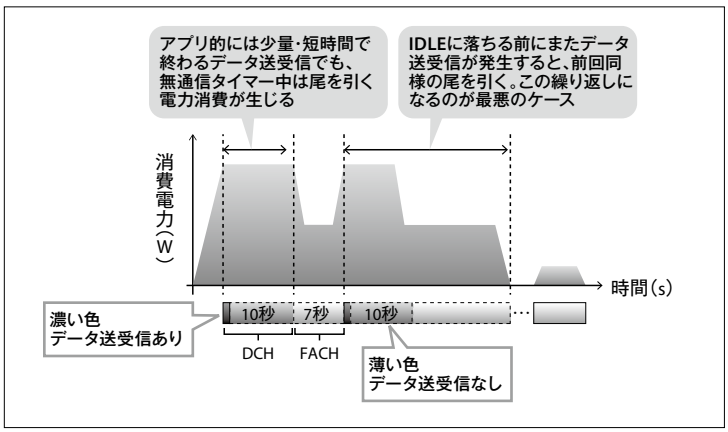


図7のイメージ図を例に説明します。なお、ここでは説明上、DCH→FACH遷移とFACH→PCH(Paging Channel)遷移のための無通信タイマーの値をそれぞれ10秒、20秒とします。まず、アプリ側からのデータ通信要求に基づき、DCHに遷移し、データ送受信が1秒間行われたとします。次に、無通信状態が続き、DCH→FACHの無通信タイマー10秒がカウントされ、FACHに遷移します。この時点ですぐお気づきの方もいるかもしれませんが、たった1秒程度のデータ送受信を行うのに、無通信タイマー動作中は尾を引く形で無駄な電力を要してしまうのです。**図7**の例に戻ります。FACHに遷移した後、データ通信要求がないので20秒の無通信タイマーがカウントされていきます。そのまま20秒経過すればさらに下位のPCHに遷移するところですが、7秒後にデータ通信要求が起こったためDCHにふたたび遷移します。そこから先は、前述のとおりのロジックです。順調にIDLEまで落ちていけばいいですが、なかなか落ちずにDCHとの行き来を延々と繰り返すとなると最悪です。

このように、十数秒おきに比較的少量のデータ送受信を繰り返すようなパターンは、実際のアプリ利用で結構あるのではないかと思います。ニュースリーダやTwitterクライアントなど、読んでは次をダウンロード……を繰り返すようなアプリはわかりやすい例です。または、サーバ

とのデータ同期などなんらかの目的で周期的なバックグラウンドタスクとしてデータ送受信を行うアプリもあるでしょう。具体的な事例は**参考文献**[1]に多く紹介されていますので、興味があればご覧ください。

省電力化のためのヒント

さて、では効率的なデータ送受信となるためにどうアプリを設計したらよいか、まとめて本項を締めたいと思います。

❶ データ圧縮・キャッシュ利用などで1回の送受信量を減らす工夫をする
❷ プリフェッチなど1回の受信で充分なデータをあらかじめ取得しておき、データ送受信の頻度・間隔を最適化する
❸ 少なくとも5〜10分程度の間隔をあける形で、データ送受信の頻度を下げる

❶については、ブラウザでは従来より一般的に考慮されているポイントですが、ニュースリーダなど、テキストだけでなく画像なども含むコンテンツを扱うアプリではとくに有効かと思います。また同時に、❷のように、コンテンツの一覧表示画面をもつアプリの場合は、小分けに複数回ダウンロードするのではなく、一度にダウンロードする記事件数を最適化することで頻度を下げることができます。実際、とあるTwitterクライアントでそのような改善を施したところ、大幅な省電力効果がありました。最後に❸ですが、これは本項で取り上げたとおりです。もちろん、先に挙げたRRC stateの遷移条件を厳密に考慮する必要はまったくありません。タイマーの値などキャリアによって仕様が異なりますので現実的に不可能です。ただし、どうしても無駄が出やすいポイントであることをご理解ください。

バックグラウンドタスクによる影響

ここからは省電力化のアプローチを変え、ユーザの目に見えないバックグラウンドタスク(以下、BGタスク)をもつアプリの設計に参考になるポイントを解説します。ここでいうBGタスクとは、端末のディスプレイがOFFで、ユーザによるアプリ操作のない状態において、AlarmManagerやBroadcastReceiverなどを契機に実行されるタスクのことを指します。前述しましたモバイル無線IFの省電力化のポイントと一緒に考慮いただくとより効果的です。

省電力化の観点から、BGタスクを頻繁に実行させないようにするべきという考え方は一般的に知られていることだと思いますが、それはなぜでしょうか。あまり大きなCPU負荷のあるタスクを頻発すべきではない、前項で述べたように無駄に通信すべきではないなど、個々のハードウェアコンポーネントのリソース消費を抑えるべきという考え方がその理由なのではないでしょうか。

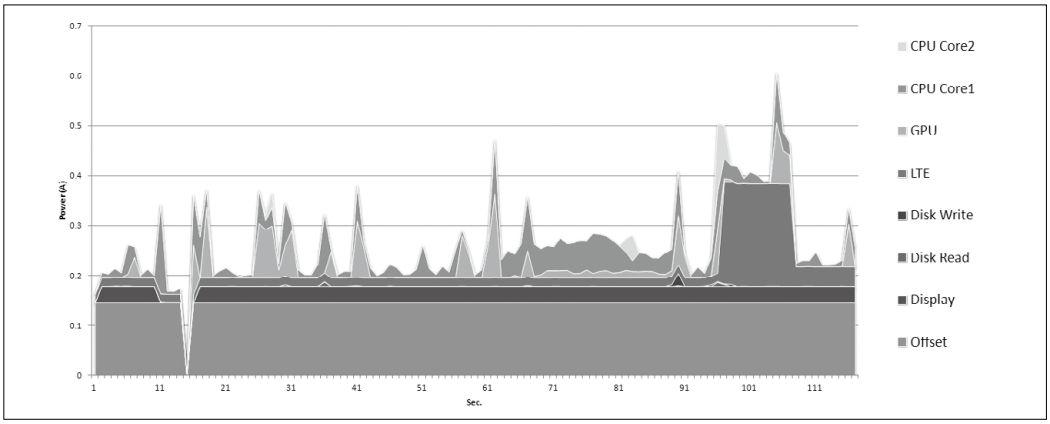
もちろんそれは正解です。しかし、本項の内容は、それらとは異なる部分にフォーカスしたものです。本記事の冒頭で、いくつかのアプリごとの消費電流量とその内訳(**図3**)について触れましたが、そのうちOffsetと記された部分については省略しました。本項で解説するポイントは、まさにそのOffset部分に関するものです。これがBGタスクとどう関係するか、解説していきます。

Offset電力とは

まず、Offset電力について説明します。なお、以降も記事中においてOffsetと呼んでいきますが、これは一般的に知られている用語ではなく、あくまで筆者を含む研究チーム内で勝手に命名したものになりますのでご注意ください。

図8は、ある端末でメールアプリを使用したときの消費電流です。CPUやLTEなどの内訳があるなか、一番下におよそ0.15Aくらい常に消費している成分があります。これがOffset電力です。メールアプリ使用期間中において、消費電流全体にOffsetが占める割合が大きいこと

▼図8　メールアプリ使用時の消費電流



がわかります。

Offset電力は、端末上でアプリなどのソフトウェアが動作可能な状態においては、CPUなどコンポーネントの稼働状態にかかわらず必ず消費する消費電力を指しています。少し言い換えると、ディスプレイはOFFだが、CPUをはじめ端末全体がSleepしておらず、各コンポーネントがIdle状態であるときの端末全体の消費電力です。フィーチャーフォン含め基本的にすべての携帯電話は、ユーザが使用していないとき、いわゆる待受状態の電池消費を極力抑えるため、ディスプレイをOFFにし、CPUなどチップセット上の各コンポーネントを休眠状態（Sleep）に移行させます。言うまでもなくアプリはこの状態のままでは動作できませんので、ユーザ操作によるディスプレイONか、ディスプレイOFFのままでもOSなどシステム側の制御を契機にSleep状態から復帰したうえで動作します。

つまり、Offsetはアプリが動作中は必ず消費してしまう電力ですので、アプリがいくらハードウェアコンポーネントの使用を抑えても、結局削減できない大きな電力成分が残っているということです。とくに、ディスプレイOFFの状態でBGタスクを実行させるときには、よりOffsetが占める割合が高くなりますし、バックグラウンド状態での動作は目に見えないため、アプリ開発者でもなかなか気づきにくいポイントだと思います。また、Sleep状態から時刻指定などでBGタスク起動する際は、基本的にそのアプリのためだけにSleep状態からの復帰とOffsetの消費を伴うことになります。そのため、多くのアプリがインストールされた端末全体で見たとき、各アプリから個別にタスク起動が頻発すると電池消費への影響は非常に大きくなるといえます。

省電力化のためのヒント

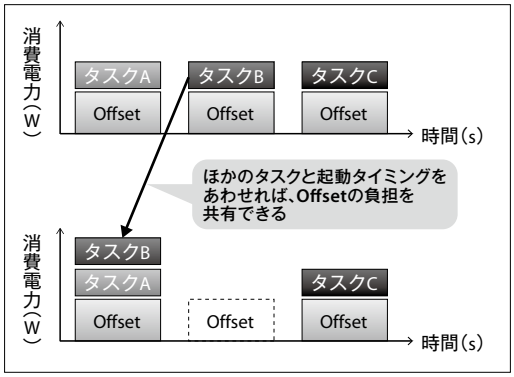
すでにお気づきの方もいらっしゃるかもしれませんが、アプリ設計からOffset分の「電力」を削減することはできませんが、「電力量」を削減することはできます。それはBGタスクの動作時間を短縮することです。前述したとおり、電力量は電力×時間ですので、時間を短縮することで結果的に電力量を削減することができます。

そのためには、まず、BGタスクの起動タイミングや回数が本当に適切か、機能要件とあわせてご検討ください。たとえば、目覚まし時計のように必ず朝7時に起動しなければいけないといった実時間制約があるか、または多少の遅れが許されるか、などが挙げられます。

そのうえで、さらに動作時間を削減するためには、**図9**のように、他のBGタスクと起動タイミングを集約することができれば、その分端末のSleep時間を確保でき、Offset分の電力量を1回に抑えることができます。

起動タイミングの集約方法をご紹介します。

▼図9　BGタスクのタイミング調整によるOffset電力の削減



基本的に、実時間制約のないBGタスクが、サービス性の観点から集約の対象になります。具体的には、AlarmManagerにBGタスクの起動スケジュールを設定する際のポイントになります。ご自身のアプリだけでなく、ほかのアプリのBGタスクも含めて、OS側のしくみ、または制御によって、自動的に集約することが可能です。

❶ スケジュール設定ではWAKEUP指定をしない
❷ set()の代わりにsetWindow()を使う
❸ 繰り返し処理のスケジュール設定ではsetInexactRepeating()を使う

AlarmManager関連のメソッドはAPIレベルによってその有無や挙動が異なりますので、詳しくはAPIドキュメントなどをご参照ください。

❶について、スケジュール設定を行うメソッド（set()やsetRepeating()など）の引数Typeに、ELAPSED_REALTIME_WAKEUPまたはRTC_WAKEUPを指定すると、指定されたタイミングで端末がSleep状態であればそれを解除して、必ずタスクが起動されます。起動タイミングの遅延が許容されるタスクに対しては、WAKEUPのないTypeを指定しておけば、他のタスクなどによるSleep状態の解除を契機に起動され、結果的に複数タスクの起動が同じタイミングになります。

ただ、これだといつタスクが起動されるのかわからず不安になるかもしれません。その場合、❷のとおり、APIレベル19から新たに加わったメソッドの1つ、setWindow()を用いると良いでしょう。引数に期間を指定する形で、その期間内にタスクが起動されるようOSが制御してくれます。このとき期間内に起動する予定のタスクがほかにあれば、タイミングが集約されます。また、周期的な繰り返し処理を行う場合には、setInexactRepeating()でスケジュール設定を行うと、周期的に同じ傾向をもつほかのタスクがあれば、同じタイミングで起動するようにOSが制御してくれます。

おわりに

最後になりますが、筆者自身、アプリの省電力化を図るからといって、結果的にサービスの品質レベルやユーザ体験を損なってはいけない、それらなしに省電力化を優先すべきではないと思っています。また、品質レベルを担保するのに必要最小限のエネルギーであるなら、それは「無駄」ではないと思います。ユーザにアプリがどう使われるかよく考えたうえで、アプリ品質のイチ指標として省電力化観点を考えていただけると幸いです。SD

参考文献

[1] F. Quin, et al, Profiling Resource Usage for Mobile Applications: A Cross-layer Approach, MobiSys' 11, June 28-July 1, 2011, Bethesda, Maryland, USA.
[2] 小西 哲平, 稲村 浩, 川崎 仁嗣, 神山 剛, 大久保 信三, 太田 賢, "画面オフ状態におけるバックグラウンドタスク同時実行によるAndroid端末の省電力化", 情報処理学会論文誌,55(2),587-597 (2014-02-15)
[3] Takeshi Kamiyama, Hiroshi Inamura, Ken Ohta, "A Model-based Energy Profiler using Online Logging for Android Applications", Proc. of The seventh International Conference on Mobile Computing and Ubiquitous Networking (ICMU2014), pp.7-13, January 2014.

Column

Docomo Application Profilerのご紹介

Docomo Application Profiler(以下、DAP)とは、省電力なアプリ開発を支援するためのアプリ評価ツールです(**図A**)。後述のURLからアカウント登録いただければ、どなたでも無料でご使用いただけます。DAPは、実際の端末・使用環境において開発中アプリの動作ログを収集し、サーバ上で解析を行います。主な特徴は次の3点です。

1消費電力の可視化

アプリ動作中の電力消費をシミュレーションする技術を用いることで、測定器を用意しなくても簡単にアプリの消費電力を評価できます。**図8**のように、CPUなどハードウェアコンポーネントごとの内訳も確認できます。

2アプリ改善個所の可視化

Activity、ServiceといったAndroidアプリを構成するソフトウェアモジュール単位でアプリ挙動と消費電力をとらえ、可視化することで、電力消費量の高い、優先的に改善すべきモジュールの特定を可能にします。たとえば**図B**のようなグラフで、モジュール間の呼び出し関係とモジュールごとの電力量の合計を可視化することで、もっとも消費量の大きいモジュールを特定でき、優先的に改善することで効果的にアプリの省電力化を図ることができます。

▼図A　DAPのロゴ

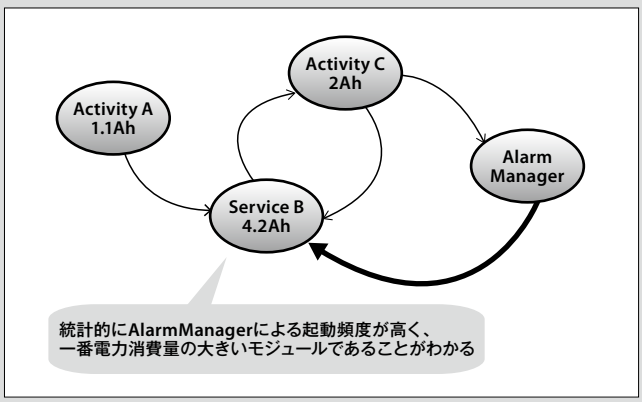
3アプリの実利用動向の把握

テストユーザやテストシナリオを増やすことで、開発時に想定しにくいアプリの使われ方や、利用環境の違いを把握し、アプリの評価や改善ができます。

詳しくは、以下、Docomo Application Profiler Webサイトよりご覧ください。

URL https://dap.dev.smt.docomo.ne.jp/

▼図B　モジュール遷移グラフ



神山 剛(かみやま たけし)　㈱NTTドコモ 先進技術研究所

大学時代の零細IT企業代表を経て、現在はNTTドコモ 先進技術研究所に勤務。入社以来、モバイルコンピューティング、とくに端末ソフトウェアの省電力化や分散システムに関する研究開発に従事。釣りなどアウトドアライフをこよなく愛し、いつかはITとは無縁の世界で生きていきたいと願っている。

Red Hat Enterprise Linuxを
極める・使いこなすヒント

.SPECS

ドット・
スペックス

第5回 Red Hat Enterprise Linux 7とDockerに触れてみよう

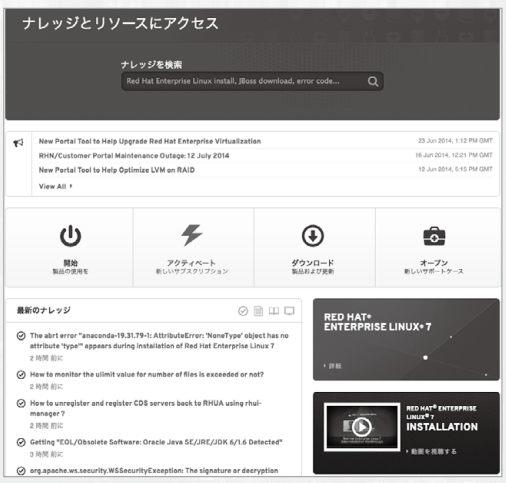
前回はRed Hat Enterprise Linuxの新バージョンである7の新機能を中心に紹介しました。今回は「お試し方法」を中心に紹介します。話題となっているDockerの利用方法についても紹介しますので、本稿を参考にぜひ触れてみてください。

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)

RHEL7の試用版の入手方法

製品として利用するにはサブスクリプションを購入する必要がありますが、レッドハットでは試用・評価する方法も用意しています。試用版(Evaluation Subscription)は30日間利用可能となっており、製品版との技術的な違いはありませんし、評価終了後にサブスクリプション契約を結べば、そのまま本番環境でも利用可能です。評価版の入手方法を次で説明しましょう。

▼図1　[RED HAT ENTERPRISE LINUX 7(https://access.redhat.com/home)



●Step1

Red Hat Customer Portal(https://access.redhat.com/home)にアクセスし、[RED HAT ENTERPRISE LINUX 7]バナーをクリックします(**図1**)。

●Step2

“INTRODUCING RED HAT ENTERPRISE LINUX 7”のページにおいて、“TRY IT FREE FOR 30DAYS”の[Request an Evaluation]のリンクをクリックします。

●Step3

“Important Evaluation Terms”が表示されるので熟読し、内容に合意できる場合、[By proceeding, ...]チェックボックスをチェックし、[Continue]ボタンをクリックします(**図2**)。

●Step4

Red Hatログインページでアカウント情報を入力してログインします(**図3**)。アカウントを持っていない場合は、[登録]ボタンをクリックして作成します。アカウントの作成は無償です。

●Step5

アカウントへの評価版の追加登録が完了すると“Thank you!”ページが表示され、数分後にはメールが届くので、Webページあるいはメールの[Explore Your Evaluation]ボタンをクリックします。

●Step6

Step2のページが表示されるので、“DOWNLOAD RED HAT ENTERPRISE LINUX 7”の[Get Installation Media]のリンクをクリックします。

●Step7

評価版としてダウンロードできる製品の一覧が表示されるので、[Red Hat Enterprise Linux]のリンクをクリックします。

●Step8

[RHEL 7.0 Binary DVD](3.49GB)をダウンロードします。“Supplementary”はIBMのJavaランタイムなどが必要な場合にダウンロードします。

●Step9

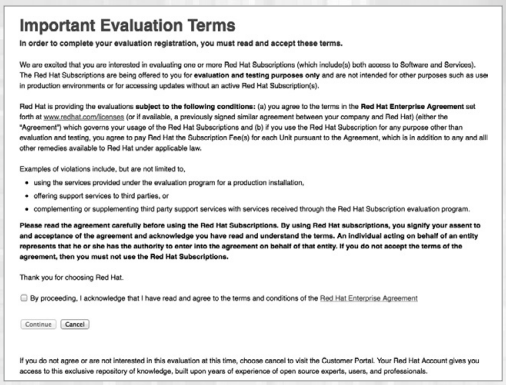
ダウンロードが完了したらチェックサムをチェック[注1]します。

以上の手順でインストーラのDVDイメージが入手できたので、DVD-Rに焼いて物理マシンか、ISOイメージを仮想DVDドライブを通じて仮想マシンに接続してインストールします。

RHEL7の登録

前回説明したようにRHEL7のインストーラはハブ&モジュール形式になっており、変更したい個所だけを任意の値に設定すればインストールが可能です。またサブスクリプションの管理についてはRed Hat Satellite[注2]のユーザを除きrhn_registerが廃止され、subscription-managerのみが利用可能となったことも説明しました。具体的な手順については触れませんでしたので、推奨されるインストール時設定のパッケージグループのMinimal(日本語では「最小限のインストール」)を選択してインストールした前提で、CUIあるいはターミナルを起動してコマンドを実行する手順を説明します。

▼図2 [Continue]で先に進める



▼図3 アカウント作成

●Step1

サブスクリプションマネージャでRed Hatに登録します。評価版の入手時の「Red Hatアカウント」が「ユーザー名」です(**図4**)。

▼図4 サブスクリプションマネージャで登録

```
# subscription-manager register
ユーザー名: rfujita
パスワード:
システムは ID で登録されています: af5261a5-efe8-45ae-b23c-d76be374e834
```

注1) RHEL 7.0 Binary DVDであれば“85a9fedc2bf0fc825cc7817056aa00b3ea87d7e111e0cf8de77d3ba643f8646c”。Linuxであればsha256sumコマンドが、Mac OS Xであれば“shasum -a 256”コマンドを実行し比較する。

注2) Red Hat Satellite(旧称・Red Hat Network Satellite)は、RHELを大量に利用する場合やインターネットに接続できない環境でRPMパッケージの更新やシステムの管理を行うための製品。システムの標準化などにも用いられ、Red Hat製品のレポジトリとして利用可能。

▼図5　利用可能なプール一覧の取得

```
# subscription-manager list --available
+-------------------------------------------+
    利用可能なサブスクリプション
+-------------------------------------------+
サブスクリプション名:     30 Day Self-Supported Red Hat Enterprise Linux Server, (2 sockets) (Up to 1 guest)
Evaluation
提供:                     Red Hat Beta
......
プール ID:                8a85f98146f719190146fb3c2bae1859
......
```

▼図6　システムへのプール紐づけ

```
# subscription-manager attach --pool=8a85f98146f719190146fb3c2bae1859
サブスクリプションが正しく割り当てられました: 30 Day Self-Supported Red Hat Enterprise Linux Server, (2 sockets)
(Up to 1 guest) Evaluation
```

▼図7　yumでアップデートしておく

```
# yum repolist
読み込んだプラグイン:product-id, subscription-manager
rhel-7-server-rpms                                       | 3.7 kB     00:00
......
リポジトリー ID                  リポジトリー名                              状態
rhel-7-server-rpms/7Server/x86_64 Red Hat Enterprise Linux 7 Server (RPMs) 4,480
repolist: 4,480
# yum -y update
......
```

▼図8　レポジトリの追加

```
# subscription-manager repos --enable=rhel-7-server-extras-rpms
リポジトリ 'rhel-7-server-extras-rpms' はこのシステムに対して有効になっています。
# subscription-manager repos --enable=rhel-7-server-optional-rpms
リポジトリ 'rhel-7-server-optional-rpms' はこのシステムに対して有効になっています。
```

●Step2

利用可能なプールの一覧を取得します。ここで表示される「プールID」をコピーしておきます(**図5**)。

●Step3

システムにプールを紐付けます(**図6**)。

●Step4

yumコマンドで登録されたレポジトリを確認します。確認後、最新のパッケージにアップデートします(**図7**)。

yumコマンドでのアップデートが完了したら、kernelを最新のバージョン[注3]に置き換えるために再起動しましょう。

RHEL7でDockerしよう!

前回の記事でRHEL7にはDockerが同梱され、extrasチャネルを追加すれば利用可能になることをご紹介しました。RHEL7のインストールが完了すれば、Dockerの利用は非常に簡単です。次の手順でDockerをインストールします。

●Step1

レポジトリを追加します(**図8**)。

●Step2

Dockerを追加インストールします。

注3) 執筆時点ではkernel-3.10.0-123.4.2.el7が最新。

▼図9 Dockerサービスの起動と有効化

```
# systemctl start docker
# systemctl enable docker
ln -s '/usr/lib/systemd/system/docker.service' '/etc/systemd/system/multi-user.target.wants/docker.service'
# systemctl status docker
docker.service - Docker Application Container Engine
   Loaded: loaded (/usr/lib/systemd/system/docker.service; enabled)
   Active: active (running) since Mon 2014-07-07 16:08:47 JST; 1min 25s ago
......
```

▼図10 Dockerのイメージのダウンロード

```
# docker pull registry.access.redhat.com/rhel
Pulling repository registry.access.redhat.com/rhel
e1f5733f050b: Pulling image (latest) from registry.access.redhat.com/rhel, endpo
e1f5733f050b: Download complete
```

▼図11 Dockerイメージの確認

```
# docker images
REPOSITORY                      TAG                 IMAGE ID            CREATED             VIRTUAL SIZE
registry.access.redhat.com/rhel latest              e1f5733f050b        4 weeks ago         140.2 MB
```

▼図12 Dockerイメージの稼働

```
# docker run -it registry.access.redhat.com/rhel /bin/bash
bash-4.2# id
uid=0(root) gid=0(root) groups=0(root)
bash-4.2#
```

```
# yum -y install docker
```

これでDockerのインストールは完了です。次にDockerサービスを起動し有効化します(**図9**)。

執筆時点ではDockerはfirewalldとコンフリクトするためfirewalldを停止します。

```
# systemctl stop firewalld
# systemctl disable firewalld
```

以上でDockerを利用する準備が整いました。レッドハットのレジストリ[注4]からイメージをダウンロードしましょう(**図10**)。

ダウンロードが完了したら、イメージを確認しましょう(**図11**)。

これで準備が整いました。早速、このイメージを動かしてみましょう(**図12**)。

これでコンテナ内でbashシェルを実行できました。これだけではとても実用的とは言えませんが、レッドハットのドキュメント[注5]や同僚の中井氏のスライド[注6]にサーバを動かす例などがあるので、それらを参考にしてみてください。

次号は!

米国時間の2014年7月1日にRed Hat Satellite 6のパブリックベータ版が公開されました。RHEL7を含む数十台～数千台のRHELサーバの管理を可能にするSatelliteのしくみと機能について紹介する予定です。SD

注4) Dockerではコンテナに割り当てるファイル等のデータをイメージと呼び、レジストリにイメージを登録して用いる。レジストリを独自に構築することも可能だが、本稿では説明の簡略のためレッドハットのレジストリを利用。

注5) http://red.ht/1zjfOPU

注6) http://www.slideshare.net/enakai/docker-34526343

Be familiar with FreeBSD.

チャーリー・ルートからの手紙

第11回 ◇ FreeBSD 10.0新機能紹介 ～iSCSIストレージの作りかた～

◉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。

カーネルネイティブiSCSIターゲット

FreeBSD 10.0-RELEASEで追加された新機能のうち、業務向けの機能として最たるものの1つがカーネルネイティブに動作するiSCSIターゲット／イニシエータ[注1]の機能です。FreeBSD 10.0-RELEASEからはソフトウェアを追加することなく、iSCSIのターゲットやイニシエータとして使うことができます。

FreeBSDはクライアントとして使うよりはサーバとして使うことのほうが多いオペレーティングシステム（OS）ですので、シーンとしてはiSCSIターゲットとして使われることのほうが多いでしょう。ZFSで対象となるボリュームを作成し、iSCSIターゲットデーモンを通じてiSCSIターゲットとして機能させます。

セットアップも簡単にできますので、今回はiSCSIターゲットを作成してWindowsクライアント向けにボリュームを提供する方法を紹介します。

ZFSでボリューム管理

NAS（Network Attached Storage）やiSCSIターゲットなど、ストレージシステムを構築する場合に力を発揮するファイルシステムでありボリューム管理システムであるのがZFSです。/boot/loader.confファイルに次の設定を追加してZFSを有効化します。

```
zfs_load="YES"
```

続いてZFSでボリュームを作成して、そのボリュームをiSCSIターゲットとして使用することにします。

次にZFSで利用するプールを作成します。次のようにコマンドを実行すると、SATA接続された3つ目のディスク（ada2）と4つ目のディスク（ada3）をRAID1構成（ミラーリング）にして、プール「z1」が作成されます。

```
# zpool create z1 mirror ada2 ada3
```

zpool listで確認すると1.81TBのプールが作成されていることがわかります（**図1**）。

zpool statusでより詳しい状態を確認できます（**図2**）。

プールを作成したら次はボリュームを作成します。次のようにzfs createを実行すると50GBの「z1/disk」というボリュームが作成されます。

```
# zfs create -V 50G z1/disk
```

作成したボリュームは**図3**のようになっていることが確認できます。このボリュームは/dev/zvol/z1/diskというブロックデバイスとして認識されます（**図4**）。これでiSCSIターゲットで使用するボリュームの準備は完了です。

注1）iSCSIでは、レスポンスを返すサーバ側のことをターゲット、コマンドを発行するクライアント側のことをイニシエータと呼びます。

各コマンドのうしろについている括弧書きの数字は、manコマンドで見ることができるマニュアルに記載されている章番号を表しています。

▼図1　1.81TBのプールz1

```
# zpool list z1
NAME   SIZE  ALLOC   FREE    CAP  DEDUP  HEALTH  ALTROOT
z1    1.81T   603G  1.22T    32%  1.00x  ONLINE  -
#
```

▼図2　z1プールのより詳しい情報

```
# zpool status z1
  pool: z1
 state: ONLINE
  scan: resilvered 479G in 3h1m with 0 errors on Thu Dec 13 15:33:30 2012
config:

        NAME        STATE     READ WRITE CKSUM
        z1          ONLINE       0     0     0
          mirror-0  ONLINE       0     0     0
            ada2    ONLINE       0     0     0
            ada3    ONLINE       0     0     0

errors: No known data errors
#
```

▼図3　zfs listで作成したボリュームを確認

```
# zfs list z1/disk
NAME      USED  AVAIL  REFER  MOUNTPOINT
z1/disk  51.6G  1.19T    16K  -
#
```

▼図4　ボリュームはブロックデバイスとして認識される

```
# ls -l /dev/zvol/z1/
total 0
crw-r-----  1 root  operator  0x9e Jul  5 10:08 disk
#
```

なお余談になりますが、ZFSは単一のディスクでプールを構築してもあまり意味がありません。ここで示したように2台でRAID1、または4台以上でRAID1＋0、複数台でRAID-Z（RAID5やRAID6に相当）を構成する方法が推奨されます。

iSCSIターゲット管理

iSCSIターゲットとしての機能は**ctld**(8)デーモンが担当します。次の設定を/etc/rc.confに追加して、ctld(8)デーモンを利用できるようにします。

```
ctld_enable="YES"
```

どのボリュームを誰に対して提供するのか、アクセスするユーザの認証はどうするのか、といった設定は/etc/ctl.confファイルに記述します。

たとえば**リスト1**のような/etc/ctl.confファイルを作成します。

設定されている内容の意味は**表1**のとおりです。

▼リスト1　iSCSIターゲット設定を記述した/etc/ctl.conf

```
portal-group pg0 {
        discovery-auth-group no-authentication
        listen 0.0.0.0
}

target iqn.2002-05.jp.co.ongs:disk {
        portal-group pg0
        chap daichi NH6dcqNu9T0eQ
        lun 0 {
                path /dev/zvol/z1/disk
                size 50G
        }
}
```

iSCSIイニシエータ側では「iqn.2002-05.jp.co.ongs:disk」というターゲットを見つけて、ユーザ名に「daichi」、パスワードに「NH6dcqNu9T0eQ」で接続すればこのターゲットを利用できます。

設定ファイルを作成したらctld(8)デーモンを次のように起動します。

▼表1　/etc/ctl.confの設定内容の意味

大枠	項目	内容
portal-group pg0		ポータルグループの設定。iSCSIターゲットの探索を許可する範囲を設定。ここでは誰に対しても探索を許可している
	discovery-auth-group no-authentication	探索に関しては認証なし
	listen 0.0.0.0	すべての範囲を対象
target iqn.2002-05.jp.co.ongs:disk		ターゲットの名称。disk部分は任意の文字列。iqn.2002-05.jp.co.ongsは「iqn.ドメインを取得した年-ドメインを取得した月.ドメイン名の逆さ順」を指定
	portal-group pg0	ポータルグループ（探索範囲）を指定
	chap daichi NH6dcqNu9T0eQ	このターゲットにアクセスするための認証設定（CHAP認証、ユーザ名daichi、パスワードNH6dcqNu9T0eQ）
	path /dev/zvol/z1/disk	提供するボリューム
	size 50G	提供するボリュームのサイズ

```
# service ctld start
```

設定がちゃんと機能しているかは**ctladm devlist**を使って確認できます（**図5**）。作成したボリュームがターゲットとして提供されていることがわかります。

/etc/rc.confに設定を追加しましたので、OSを再起動してもctld(8)は自動的に起動してくれます。

Windows 7から使ってみよう

それではWindows 7から使ってみましょう。「iSCSIイニシエーター」を起動すると、設定したターゲットがリストに掲載されていることを確認できます（**図6**）。

ターゲットを選択して詳細設定のダイアログを表示させます。「名前」部分がユーザ名、「ターゲットシークレット」部分がパスワードです。ここに設定した値を入力します（**図7**）。

ターゲットに接続したら「ディスクの管理」を起動します。接続したターゲットが認識されますので、フォーマットして利用できる状態にします（**図8**）。

フォーマット後に49.87GBのNTFSでフォーマットされたボリューム「E:」が作成されたことを確認できます（**図9、10**）。

このようにちょっとした作業ですぐに実用的なiSCSIストレージシステムを構築できます。

活用はアイディア次第

簡単に利用できるのであっという間に作業は終わってしまいますが、これはとても強力な機能なので、さまざまなシーンでさまざまな使い方ができます。業務システムのクライアントディスクを集約してディスクレスシステムを構築するといったことから、プロジェクトごとに必要になるストレージの提供など業務的な使い方もできますし、個人のツールとしても活用できます。

仮想環境との組み合わせも便利です。業務で利用するデータはiSCSIターゲット側で保持して、仮想環境には最低限のデータだけを保持するといった組み合わせで、仮想環境側のリソースを抑えながら利用するストレージを大きく持ってお

▼図5　設定の確認

```
# ctladm devlist -v
LUN Backend       Size (Blocks)   BS Serial Number    Device ID
  0 block            104857600  512 MYSERIAL   0     MYDEVID   0
      lun_type=0
      num_threads=14
      file=/dev/zvol/z1/disk
      cfiscsi_target=iqn.2002-05.jp.co.ongs:disk
      cfiscsi_lun=0
#
```

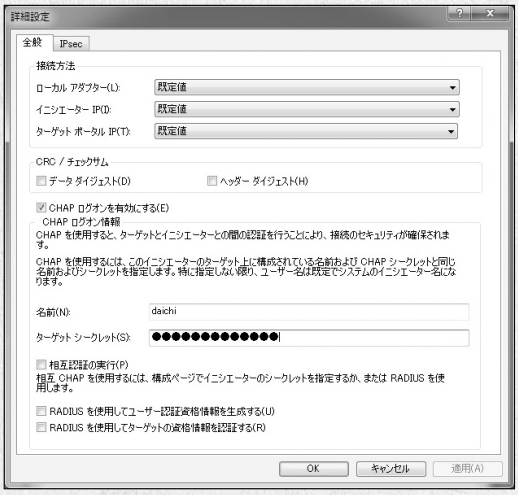
くことができます。アイディア次第で多くのことができます。

ストレージシステムをつくってみよう

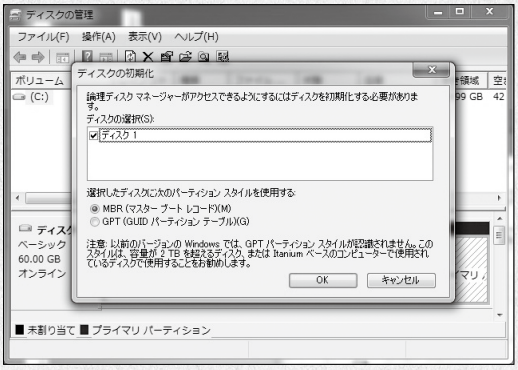
FreeBSDは複数社のエンタープライズ向けのストレージアプライアンスで採用されています。このため、こうした機能はすでにハードウェアベンダは独自の実装として持っていました。今回、デフォルトの機能としてiSCSIターゲット／イニシエータの機能が導入されたことで、ストレージシステムを構築するためのOSとして今までよりも便利になりました。SD

▼図6　設定したターゲットを確認

▼図7　CHAP認証のユーザとパスワードを設定



▼図8　認識される新しいボリューム



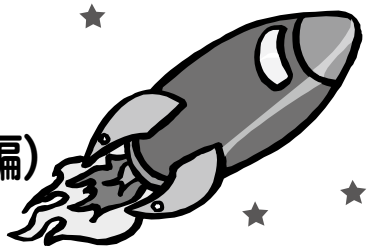
▼図9　新しいボリューム「E:」

▼図10　ボリュームのプロパティをチェック

18

Debian Developer　やまねひでき　henrich@debian.org

設定ファイルの読み方・書き方でわかるLinuxのしくみ(Debian編)



Debian Hot Topics

Debian 7.6/6.0.10がリリース

7月には、Debian 7.6とDebian 6.0.10が、アップデートとしてリリースされました。Debian 6はLTSとして、一部のパッケージのセキュリティ修正は行われるはずですが、アップデートリリースは6.0.10が最後になります。そして、Debian 8の開発スケジュールは変わらずフリーズが着々と近づいており、作業に手のついていない筆者にはプレッシャーとなっています。

RHELの流儀だけがLinuxではありません

本誌2014年6月号の第1特集は「設定ファイルの読み方・書き方でわかるLinuxのしくみ」と題した、レッドハット㈱の中井悦司氏による記事でした。当然のことながら、内容はRHEL 6(Red Hat Enterprise Linux 6)を前提にした説明となっていました。しかし、RHELとDebianでは若干の差異があります。Debianユーザで、この情報を参考にしている方がつまずかないよう、「設定ファイルの読み方・書き方でわかるLinuxのしくみ(Debian編)」をお送りしたいと思います(なお、本誌6月号が手元にあることが前提となっている点が多いのは、ご容赦ください)。

/tmpは定期的に空に……されません

DebianはRHELとは違ってtmpwatchは用意されていませんので、/tmpに置いたファイルが勝手に消えることは、基本的にありません。「基本的に」と述べたのは、デフォルトでは/tmpはtmpfsというメモリファイルシステム上に置かれるようになっており[注1]、再起動するとメモリはクリアされますから、それに伴って/tmpの内容も空になるからです。「再起動しなければ/tmpの中身は消えない」と覚えてください。

viエディタ……なの?

Debianの最小構成ではvi(あるいはvim)はインストールされません(当然、Emacsもですよ、念のため)。代わりのエディタとして「nano(なの)」が用意されています。

visudoコマンドでsudoユーザの編集をしようとすると、nanoが起動され、「なんじゃこりゃぁ!?」と叫ぶユーザもいることでしょう[注2]。edじゃないことに感謝してください……というのは冗談ですが、vi(vim)を使いたい場合は、適宜、`apt-get install vim`などとしてパッケージをインストールしてください[注3]。

注1) RHEL 7では「systemctl enable tmp.mount」とすることで、Debianと同様に/tmpをtmpfsにできるようになっています。

注2) visudoでは、/usr/bin/editorが呼ばれます。これが/etc/alternatives/editorを参照していて、alternativesというしくみでエディタを切り替えて使うことになっています。alternativesは、Debian由来でRHELにも採用されている機構です。詳細はupdate-alternatives(8)を参照してください。

注3) vimのパッケージでも、最小限の機能を持ったvim-tinyパッケージと、機能を網羅しているvimパッケージがそれぞれあります。「vimを入れたのに、矢印キーでカーソルを移動できない」という相談を受けたことがありますが、そのような場合は、「hjklで移動しろ」ではなく、vim-tinyパッケージが入っているので、vimパッケージを入れれば解決します。

cronは「.」を含まないファイル名で指定せよ

「Debianを新しく設定したので、cronでシェルスクリプトを定期的に実行しよう。/etc/cron.dにファイルを置いてと……あれ、実行されていない？　権限も問題ないはずだし、前に設定したCentOSでは動いてるんだけど？」という人、実はちょっとした罠があるのです。

man cronすると「DEBIAN SPECIFIC」という項目があり、そこにはこんな説明があります。

Files in this directory have to be owned by root, do not need to be executable (they are configuration files, just like /etc/crontab) and must conform to the same naming convention as used by run-parts(8): they must consist solely of upper- and lower-case letters, digits, underscores, and hyphens. This means that they cannot contain any dots.
(このディレクトリ以下に置くファイルの所有者は、rootでなければならず(/etc/crontabのように設定ファイルですので)、実行権限は不要です。そして、run-partsで使われている命名規則に従わねばなりません。ファイル名は大文字、小文字、数字、アンダースコア(_)、ハイフン(-)で構成する必要があります。これはドット(.)を含んではいけないということです)

そう、実行しようとしているファイル名がhoge.shというように「.」を含んでいませんか？その場合、この制限に引っかかって動作しません(「いや、なんでそんな制限あるの」というのはごもっとも……はて、なんでですかね？)。これは/etc/cron.hourly (daily, weekly, monthly)に配置するファイルも同様です。

ユーザの作成とユーザID

RHELと同じくDebianにも、ユーザ／グループを作成するuseradd/groupaddコマンドが、用意されています。ですが、それとは「別に」[注4]Debianには「adduser」と「addgroup」というコマンドもあります……超混乱しますね。

「では、どっちを使うのがいいのか？」というと、答えはuseraddのmanpageに記載されていて、*"useradd is a low level utility for adding users. On Debian, administrators should usually use adduser(8) instead."* (*Debianでは、管理者の方は通常はadduserを使ってください*)とあります。スクリプトで作業するときにはuseradd/groupaddを使うのですが、ターミナルを使って作業する場合は、インタラクティブに指定ができるadduser/addgroupのほうが良いでしょう。

そして、Debianの場合、作成されたユーザのユーザIDは「1000」から始まります[注5](**表1**)。500から始まるRHELとは違いますので、NFSやほかのシステムからの移行時にはお気をつけください。

ランレベルと各デーモンのinitスクリプト制御

RHELだとランレベルを変えて動作を変更

注4) RHELではadduser/addgroupは、useradd/groupaddへのシンボリックリンクです。

注5) この指定については/etc/adduser.confを参照。

▼表1　ユーザID(UID)／グループID(GID)の割り当て範囲

ユーザ／グループの種類	RHEL 6まで	Debian
root	0	0
システム	1～499	1～99(割り当て済み)、100～999(インストール時に割り当て)
個人	500～60000(RHEL 7では1000～)	1000～59999※

※ Debian Policyでは59999までとなっていますが、/etc/login.defでの最大UID指定は60000です。/etc/adduser.confで29999になっているので、adduserコマンドを使った場合は、29999までのUID……とそれぞれ微妙に異なっています。この辺になぜ差異があるのか、問題ないのかは、もう少し調べてみたいところです。

するという概念があります。Debianではランレベル2〜5には、動作の違いがありません[注6](**表2**)。ですので、GUIを自動起動したくない場合は、ランレベルを変更して……ということができません。

「では、どうするのか?」というと、GDM3/KDM/lightDMなどのディスプレイマネージャーを入れないか、またはディスプレイマネージャーの動作を停止しておくことになります。

ですが、chkconfigでディスプレイマネージャーを停止しようとしても、Debianにはchkconfigはデフォルトインストールでは存在していません。chkconfigパッケージ自体は存在していますので入れることも可能ですが、デフォルトではinsservコマンド(**図1**)、あるいは原始的にupdate-rc.dコマンドを使って停止することになります(ほかの方法としてはrcconfパッケージやsysv-rc-confパッケージを入れる方法もあります)。

このあたりは、systemd採用で差異が吸収されていくものと推測しています。systemdについては、いち早く採用を行ったRHEL 7やCentOS 7、Fedoraなどの情報を参考にしてください。

注6) なぜ、ないんでしょうね? 筆者も不勉強でわからないので、おわかりになる識者の方ご連絡ください。

grub.confは直接いじるな

6月号の特集記事では、例としてgrub.confが取り上げられています。Debianの場合は、このgrub.confの中心要素を抜粋したものである/etc/default/grubを編集し、update-grubコマンドで変更をgrub.confに反映します[注7]。

ネットワーク設定ファイルの場所が異なる

ネットワークの設定はRHELとは違い、Debianでは/etc/network/interfacesにまとめて書かれます[注8]。設定の詳細は`man interfaces`で、マニュアルを参照してみてください。

注7) RHEL 7からは同様に/etc/default/grubを編集します。ですが、変更反映のコマンドはgrub2-mkconfigです。

注8) RHELも6と7で大幅に変わっているようで、RHEL 7からはNetworkManagerを使った設定が推奨です。デスクトップ向けツールの印象が強いNetworkManagerですが、コマンドラインからの設定やキャラクタベースのインターフェースなども追加されました。さらに、サーバで使うために、未設定のインターフェースでの自動設定を抑止するオプションと、リンクがなくてもstatic IPの設定を行うオプション設定が可能なNetworkManager-config-serverパッケージも追加されています。
Debianのnetwork-managerパッケージも、コマンドラインからの設定ツール「nmcli」に加え、対話式で設定可能な「nmtui」が含まれた最新のNetworkManagerが、先日、利用可能になりました(余談ですが、Fedora 20にもまだnmtuiは含まれていませんでした。RHELのほうがFedoraよりもソフトウェアのバージョンが新しいことがあるのですね)。

▼表2 RHEL(6まで)とDebianのランレベル

ランレベル	RHEL 6まで	Debian
1	シングルユーザ	シングルユーザ
2	マルチユーザ(ネットワーク接続なし)	マルチユーザ、X11が動作する
3	マルチユーザ	
4	(未使用)	
5	X11が動作する	
6	再起動	再起動

▼図1 insservを使ってGDM3を制御する

```
$ sudo insserv -r gdm3      ←GDM3が起動しないように設定
$ sudo service gdm3 stop    ←現在動いているGDM3を停止
```

▼図2 ufwを使ったHTTPサーバのアクセス権の設定例

```
$ sudo apt-get install ufw
$ sudo ufw enable
$ sudo ufw allow OpenSSH    ←SSHで接続している場合は遮断されないようにする
$ sudo ufw allow WWW        ←外部からのHTTP接続を許可する
$ sudo ufw default deny     ←上記以外、デフォルトでは遮断にする
必要に応じてufwコマンドを使って調整
```

手軽にiptablesを設定

iptablesを手動設定するのも理解が深まりますが、「ufw」というフロントエンドを使えば、**図2**のようにHTTPサーバへのアクセスを許可するような典型的な設定は間違いなく済ませられます。

SELinuxは?

DebianでもSELinuxは存在するものの、デフォルトでは設定がまったく入りません[注9]。手動でインストールしたとしても、パッケージ側では設定情報があまり整っていません。利用できるようになるまでには、茨の道が約束されています。「安全のためデフォルトでSELinuxをenforcing設定にしたい!」という人は、パッケージの修正のほうで未来のバージョンでのサポートに向けて一緒にがんばりましょう。

パッケージといったらaptだろ!

パッケージ管理ツールというと、RHELではrpm/yumですが、Debianはdpkg/apt(apt-get/apt-cache)です。yumは先発であるaptを意識して作られているためでしょうか、使い方は似ており、ものによってはオプションも同一です。

普段の使い方で気がつくであろうところは、yumは「update」オプションでパッケージ自体の更新が実行されますが、aptの「update」オプションではパッケージ自体はアップデートされず、パッケージデータベースの情報だけが更新されます。aptでパッケージ自体のアップデートをするには、別途「upgrade」オプションを使う、ということを覚えておくと良いでしょう。

DebianでもApacheを設定してみるか……

RHELやCentOSを利用してきたユーザがDebianを使ってみてよく言うのが、「Apacheの設定ファイル構成がよくわからない」というものです。Debianの場合は「機能ごとに細切れにするのが方針」ですので、次の点に留意して設定しましょう

- パッケージ名はhttpdではなくapache2。ディレクトリも/etc/apache2となる
- httpd.confは存在するが、ここにまとめて記述するのではなく、設定するサイトごとにファイルを分けて/etc/apache2/site-available以下に置くのが推奨(例:/etc/apache2/site-available/www.example.comなど)
- サイトを利用する場合はa2ensiteコマンド、無効にする場合はa2dissiteコマンドを使う[注10]
- 全サイトに対して設定を行う場合は、機能ごとに/etc/apache2/conf-availableにファイルを置き、a2enconf/a2disconfで切り替える
- 機能モジュールについても、同様にa2enmod/a2dismodでモジュールの有効/無効を切り替える
- 設定を変えたらサービスを再起動するなどして反映する

なぜ違いがあるの?

一部からは「違いを把握するなんてめんどくさいなー」という声が聞こえてきそうですが、問題に対するアプローチと解き方は、1つとは決まってはいません。筆者は上記のような違いについて「覚え」なくても「RHELだとこういう考え方だからこういうアプローチで、Debianだとこういうアプローチ」という「背景にある考え方の理解」をすることが重要だと考えています[注11]。SD

注9) UbuntuだとAppArmorという別のセキュリティ実装が採用されています。また、ほかの実装としてはTOMOYO Linuxがtomoyo-toolsパッケージとして存在しています。

注10) 実はこの辺、単にsymlinkを張っているだけです。ls -al /etc/apache2/sites-enabledなどと、実行してみるとわかります。モジュールについても同様です。結構簡単な作りなのですね。

注11) そうでないと、資格試験のために単に暗記しただけで応用が効かない……という残念なことになります。そもそも世の中いろいろな事柄についてさまざまな種類のものが共存しているというのに、Linuxについては1種類になるべきというのは暴論でしょう(面倒なのは理解できますが)。

Upstream First!

岩尾 はるか
IWAO Haruka
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
ストレージソリューションアーキテクト

はじめに

こんにちは。2014年5月からレッドハットで働いている、岩尾はるかです。肩書は「ストレージソリューションアーキテクト」と少し長いですが、簡単に言うと、ストレージが専門のプリセールスエンジニアです。入社前は、家電メーカーやWebサービス会社で働いていました。

入社して間もなくこの記事を依頼され、何を書いていいのかと過去記事を読み返したところ、自分なりの言葉で会社のカルチャーを伝えればいいとの考えに至りました。そこで、これまで働いていた環境との違いを中心にまとめました。

メールが基本

過去に働いたWebサービス会社では、SkypeやHipChatなどのチャットが主要なコミュニケーションの道具でした。ところが、レッドハットではメールとメーリングリストがもっとも活発な議論の場所になっています。これには大きく2つの理由が挙げられます。1つ目は拠点が世界各国にあるので、メールのほうが時差を吸収しやすいこと。2つ目は、レッドハットの各製品にはupstream（アップストリーム）と呼ばれるオープンソースのプロジェクトがあり、これらの主な議論の場がメーリングリストであること。とくに2つ目の理由は、レッドハット独特だと感じました。

私がおもに担当しているストレージ製品も、upstreamとなるGlusterFSやCephといったオープンソースプロジェクトが存在しています。これらのプロジェクトのメーリングリストをのぞいてみると、レッドハットのメールアドレスからの投稿が多く見られます。社外の開発者やユーザとのやりとりだけではなく、時には社員同士で、公開のメーリングリストで議論することもあります。つまり、ソフトウェアの開発が100%オープンソースの世界で完結していて、途中過程も含めた意思決定を、コミュニティを中心に行っているということです。通常のクローズドな製品を開発していると、普段のメールの相手は同僚のエンジニアばかりですから、これは実際に体験すると新鮮な驚きでした。

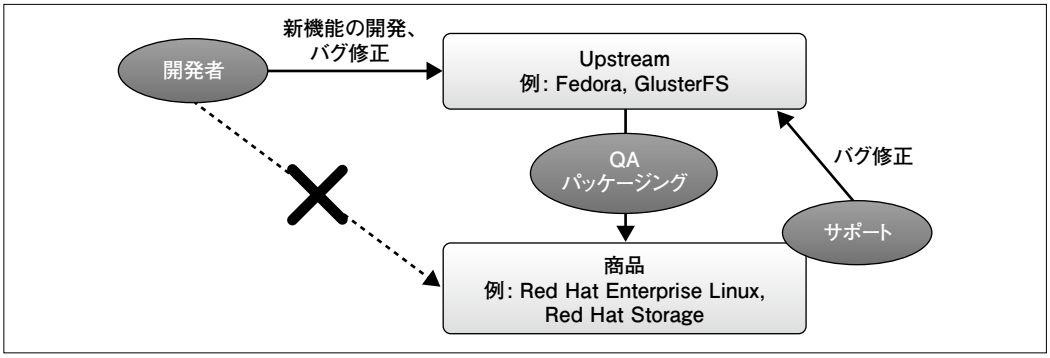
もちろんリアルタイムに会話をしたい場合には、IRCや電話会議などを使うこともあります。それでも製品開発の議論が、公開のメーリングリストを中心に行われているのは、他の企業と違う大きな特徴だと思いました。

Upstream First

レッドハットの製品開発は「Upstream First」という考え方に沿って行われています（**図1**）。メーリングリスト文化とも重なりますが、新機能の開発、バグ修正などを含めたすべての開発業務を、「Upstream」つまり商品の元となるオープンソースプロジェクトで行うという意味です。

社内で開発したライブラリなどをオープンソースとして出すことはあっても、コードの変更はまず社内版で行ってから後でまとめてリリースしたり、新機能の議論は社内で行われていることが、よくあるのではないでしょうか。また、

▼図1 「Upstream First」



社内版だけに存在するバグ修正がオープンソース版に取り込まれず、公開しているバージョンと使っている版に差があったりすることもあるかもしれません。

レッドハットでは「Upstream First」の考え方を徹底し、原則オープンソース版に取り込まれないコードは製品にも取り込まない、というポリシーを貫いています。また、自社製品で見つかったバグは必ずオープンソース版に取り込んでいます。こうすることで、オープンソース版と製品版の差を縮め、機動的な開発を可能にしています。

この「Upstream First」には違った側面もあり、それは「製品が売れることも大事だけど、オープンソース版が広く利用され、コミュニティとして活発で健全であることも同じぐらい大事」と思っている社員が多いことです。そのため、LinuxConをはじめとするオープンソースプロジェクトの会議で発表する開発者が多くいますし、コミュニティのメーリングリストに投稿してきた質問に対してボランティアとして答えることもあります(無保証です。サポートが必要な場合は製品版をご利用ください)。日本国内でも勉強会で発表したり、オープンソース版を元にした書籍を書いたりしている社員がいます。

私自身、製品を売る立場としては、オープンソース版にはない付加価値をどんどん提供していかなければならないので、苦しいこともある反面、コミュニティから学べることも多く、非常に良い刺激になっています。個人的にも、どんどんコミュニティ活動をしていきたいと思っています。

英語

外資系企業に勤めるのは初めてなので、仕事でどの程度英語を使うのかなあと、入社前は少し心配していました。ふたを開けてみると、恵比寿オフィスで働いている人は日本語が話せる人がほとんどで、オフィスでの会話はほとんど日本語でした。一方で、upstreamのコミュニティや、アメリカのエンジニアとやりとりする機会が毎日のようにあり、そちらでは英語を使っています。これは日系企業であってもソフトウェアエンジニアであれば、あまり変わらないかなと思います。人や部署によっては上司が外国のオフィスで働いている場合があり、その場合は上司との会話はすべて英語になるので、私の部署よりも英語を使う機会が多いです。

まとめ

入社する前から「Upstream First」やオープンソースコミュニティへの投資をしている会社だという印象はありましたが、入ってみてきちんと実践しているなと思いました。私もオープンソースと、ストレージの世界を盛り上げていくべく、精一杯がんばります。SD

LibreOffice 4.3の新機能

Ubuntu Japanese Team
あわしろいくや　ikuya@fruitsbasket.info

今回は恒例となったLibreOfficeの最新版である4.3の新機能や仕様の変更点をまとめます。

はじめに

7月下旬にLibreOffice（LibO）の最新版である4.3の最初のリリースが行われる予定です[注1]。4.3は4.2とは異なってCalcのコアを差し替えるといった大胆な変更はありませんが、確実に使い勝手は向上しています。今回はとくにこれまでの仕様と異なる部分を重点的にピックアップしてみました。

Writer

段落あたりの文字数の増加

LibOとその前身であるOpenOffice.org（OOo）には、1段落あたり65,535文字までという制限がありました。この数字を見たらピンとくると思うのですが、16bitという制限があったということです。4.3では、これが32bit（約21億文字）まで拡大されました。OOoのバグ報告[注2]を見ると、最初に報告があったのは2003年7月21日と、修正に約11年かかったことになります[注3]。現実的には1段落に6万文字も書くことはほとんどないと思いますが、内部的な仕様を突破するには、別プロジェクトになって数年の時間が必要であったというところが興味深いです。

ナビゲーションボタンの移動

4.2まではスクロールバーの下端にナビゲーションボタンがありました。しかし、これを使ったことがある人はほとんどいないでしょう。移動先は検索バーですが、サイドバーにも表示されるのでやはりあまり使われることはないように思います。それよりもスクロールバーのスペースが広くなり、利便性が向上したと言っていいでしょう。

文字の拡大／縮小の制限撤廃

拡大／縮小したい文字を選択して[Ctrl]+[]]キー（拡大の場合）／[Ctrl]+[[]キー（縮小の場合）を押すか、サイドバーを表示してプロパティパネルを表示し、文字の拡大／縮小ボタンを押すと、4.2までは96ポイントまでプルダウンメニューと同じ間隔で文字の拡大／縮小が可能でした[注4]。しかし、4.3からは2ポイント単位での拡大縮小が可能になり、また96ポイントまでという制限も撤廃されました。

コメントの入れ子をインポート／エクスポート可能に

これは表題のとおりですが、ODFや各種Word用

注1） いつもであればほぼ正確な日時が書けるのですが、今回は大人の事情で不可能でした。しかし、いずれにせよ本誌が店頭に並ぶころにはリリースされているはずです。

注2） 現在は事実上の後継プロジェクトであるApache OpenOffice（AOO）のバグ報告ですが。

注3） ちなみにAOOでは修正されていません。

注4） リリースノートでは72ポイントまでになっていますが、どう確認しても96ポイントまでは拡大できました。

のフォーマットでコメントの入れ子がインポート／エクスポートできるようになりました。もともとApache OpenOffice 4.1での機能だったのですが、LibOにも取り込まれました。

画像の拡大／縮小の仕様変更

4.2までは画像の拡大縮小が自由にできていましたが、やはりこれは不便で、縦横の比率は固定であったほうがうれしく、サイズ変更する場合は Shift キーを押しながらしていたと思います。4.3ではこの挙動が逆になり、普通にサイズを変更すると縦横の比率が固定され、Shift キーを押しながらだと自由に変更するようになりました。

テキスト枠で相対サイズの指定がページ全体に関しても可能に

これはスクリーンショットを見ていただくほうが早いですが(**図1**)、これまでも[挿入]-[枠]のテキスト枠では相対サイズの指定は可能でしたが、何に対する比率なのかがわかりくかったです。4.3からは、上下左右の余白を含むかどうかを選択できるようになりました。[段落範囲]は余白を含まず、[ページ全体]は含みます。すなわち、同じ比率でも[段落範囲]にするとテキスト枠は小さくなり、[ページ全体]にすると大きくなる、ということです。

テキスト枠の内容の配置位置を指定

テキスト枠内の文章を上(デフォルト)、中央、下に配置できるようになりました。上と中央は使いどころがありますが、下はどんなときに使ったらいいのか直感的には思いつきません。しかし、いずれにせよテキスト枠の用途が広まったように思います。

コメントを余白に印刷

これまでもコメントを印刷する方法はいくつかありましたが、すぐ横の余白に印刷することはできませんでした。しかし、4.3ではこれができるようになりました。方法は印刷タブで[マージン内の位置]を選択するだけです(**図2**)。[ツール]-[オプション]-[LibreOffice Writer]-[印刷]の[コメント]で既定値を変更することもできます。

コメントの書式が設定可能に

これまでコメントの書式を一括で変更する方法はありませんでしたが、コメントの削除と同じ方法[注5]で一括変更が可能になりました。

Calc

ステータスバーに選択している列と行の数を表示

ステータスバーに選択している列と行の数を表示

注5) コメントの枠内の右下にある▼ボタンをクリックするとサブメニューを表示します。

図1　相対サイズの基準を選択できるようになった

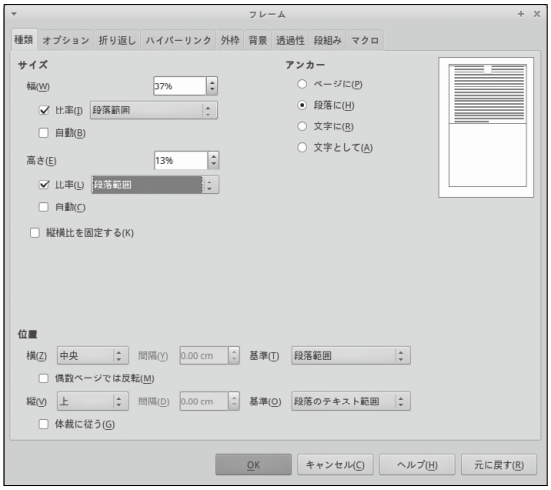


図2　左下の[コメント]を[マージン内の位置]にすると、コメントを余白に印刷できる

するようになりました。

アクティブセルの1つ上の内容をアクティブセルにコピーする機能の追加

Excelをお使いの方は Ctrl + Shift + ' (シングルクォーテーション)の機能、というとわかりやすいかもしれませんが、アクティブなセルの1つ上のセルにある内容をアクティブなセルにコピーします。複数にコピーしたい場合はオートフィルが便利かもしれませんが、1つの場合はこの機能を使うのが便利でしょう。

ただしCalcのデフォルトでは Ctrl + ' キーに割り当てられており、手元のUbuntuでは機能しませんでした。ですので、まずはこのキーバインドをExcelと同じく Ctrl + Shift + ' キーに割り当てましょう。

[ツール]-[カスタマイズ]-[キーボード]タブを表示し、[分類]を[編集]に、[機能]を[Fill Single Edit]にし、[ショートカットキー]の[Ctrl+Space+']を選択した状態で[変更]をクリックすると[Ctrl+Space+']が追加されます(**図3**)。[Fill Single Edit]は未訳ですので今後訳されるかもしれませんが、その場合は[Ctrl+']に割り当てられた機能を見て、どのように訳されたのか確認してください。

あとはコピーしたい内容のあるセルの下をアクティブなセルにし、Ctrl + スペース + 'を押すとその内容がコピーされた状態で編集ができます。これは知っているのと知らないとでは完全に作業効率が変わるほどの便利な機能だと思います。

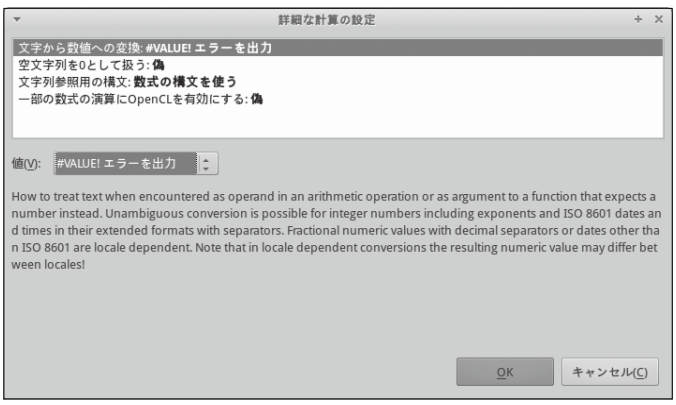
図3 [Fill Single Edit]を Ctrl + Shift + ' に割り当てます

テキストを数値に変換する場合のルールを選択可能に

やや難解ですが、[ツール]-[オプション]-[LibreOffice Calc]-[数式]-[ユーザー定義]-[詳細]に[文字から数値への変換]という項目が追加されました(**図4**)。セルの中で「"」(ダブルクォーテーション)でくくられている場合、数字であっても文字として扱われますが、これを数字として扱い、計算できるようにするためのオプションです。

[#VALUE!エラーを出力]は、そのような計算はエラーを返します。すなわち、[1+"1"]はエラーになります。[0として扱う]は、文字部分を0にします。すなわち[1+"1"]は1になります。[一義的な場合のみ変換]は、[1+"1"]は2になりますが[1+"1.000"]はエラーになります。なぜかというと、ロケールによっては千の単位での区切りに「,」ではなく「.」を使う場合もあるからです。[ロケールに依存して変換]は、千の単位の区切りが「.」の場合は[1+"1.000"]を1.001に、日本を含むそうでない場合は2になります。

図4 [文字から数値への変換]のルールを4つから選択します



追加された関数

4.3でもたくさんの関数が追加されました。**表1**にまとめましたのでご覧ください。なお、対応するExcelのバージョンはすべて2010です。

ページ総数のカウント方法の変更

[挿入]-[フィールド]-[ページ総数]でページ総数を挿入できますが、4.2までは非表示スライドもカウントしていました。すなわち、スライドのページ数が3枚で1枚非表示の場合、[3]が挿入されていました。しかし、4.3からは[2]が挿入されるようになり、実態に即したページ数となりました。

スライドをウィンドウにフィット

ステータスバーのズームを変更するスライダーの左側に[+]に似たアイコンのボタンが追加されました。これをクリックするとスライドをウィンドウにフィットできます。表示倍率の数字を右クリックして[ページ全体]をクリックしてもできましたが、ワンクリックでできるようになったのは便利です。

選択していないスライドをわかりやすく

たとえば3枚のスライドがあり、これを全部選択して真ん中だけ選択解除するには、Ctrlキーを押しながらクリックします。これが4.2ではマウスを動かさないと選択解除したかどうかがよくわかりませんでした。しかし、4.3では即座に色が変わるので選択解除がはっきりとわかるようになりました。

Drawでプロパティパネルの削除

Drawでサイドバーを有効にし、プロパティパネルを開くと4.2ではシェイプの挿入ができていました。しかし4.3ではこの機能が削除されました。「プロパティパネルは現在のプロパティを変更するためにあるので、新たにシェイプを追加するのは役割として違う」ということになったようですが、あったらあったで便利だったのでちょっと不便になったような気はします。

3Dモデルの追加

Impressのみ[注6]であり、アーキテクチャもLinuxとWindowsのみという制限はありますが、[挿入]-[オブジェクト]-[3Dモデル]が追加されました。JSON(GL Transmission Format)とDAE(COLLADA)とKMZ(Keyhole Markup language Zipped)形式の3Dモデルを挿入できます。サンプルファイル[注7]で試してみたところ、オブジェクトを選択し、編集状態だと再生ボタン[注8]を押した状態で、スライドショー実行時はとくに何もしないで、3Dモデルのwキーを押すと拡大、sキーを押すと縮小、aキーを押すと右に移動、dキーを押すと左に移動、mキーを押したあとマウスをクリックしたまま移動すると3Dモデルを回転することができます。ただ、筆者が試した限りではお世辞にも安定しているとは言えず、何度も強制終了してしま

注6) Drawでは非対応という意味です。
注7) https://wiki.documentfoundation.org/images/e/e1/Duck_gltf_model.odp
注8) 動画ファイルとして読み込まれているのか、再生ボタンや停止ボタンが表示されています。

表1　4.3で追加された関数

関数	ジャンル
GAMMA.DIST, GAMMA.INV, GAMMALN.PRECISE	統計
LOGNORM.DIST, LOGNORM.INV, NORM.DIST, NORM.INV, NORM.S.DIST, NORM.S.INV	統計
T.DIST, T.DIST.2T, T.DIST.RT, T.INV, T.INV.2T, T.TEST	統計
PERCENTILE.EXC, PERCENTILE.INC, PERCENTRANK.EXC, PERCENTRANK.INC, QUARTILE.EXC, QUARTILE.INC, RANK.EQ, RANK.AVG	統計
MODE.SNGL, MODE.MULT, NEGBINOM.DIST, Z.TEST	統計
FLOOR.PRECISE, CEILING.PRECISE, ISO.CEILING	数学
NETWORKDAYS.INTL, WORKDAY.INTL	日付と時刻
ERF.PRECISE, ERFC.PRECISE	統計

いました。正直なところどういうときに役に立つのかはよくわかりませんが、今後のバージョンアップによる機能追加に期待したいところです。

その他

その他の追加された機能は次のものです。

- FAX送信の方法が変更になりました。そしてプリンタ設定ツールであるspadminがなくなりました
- OOXMLサポートが改善しました
- PDFインポートが改善しました
- 色の選択がやりやすくなりました
- スタートセンターが改善しました
- HiDPIサポートを開始しました

いくつか補足します。まず、FAXは印刷と同じような方法で送信できるようになりました。しかし、これには[fax4CUPS]というパッケージが必要なのですが、Ubuntuのリポジトリにはありません。HiDPIサポートは、今のところなんの苦労もなく動作するのはWindows 8.1とGNOMEのみで、Unityでも問題があるようです。

翻訳状況

ユーザインターフェースの翻訳対象語は99,951語(4.3.0)であり、4.2の97,426語よりもだいぶ増えました。しかし、翻訳率は99%であり、未訳語を目にする機会はあまりないレベルになりました。とはいえ、品質を考えると多くの人の「目」と「手」が足りない状態であるのはまったく変わらないので、興味がある方はLibreOfficeの日本語メーリングリスト[注9]に投稿していただけると幸いです。

「安定版」の意義の変化

LibOはどのバージョンが安定版なのか極めてわかりにくいという問題がありました。何を以って「安定版」とするかですが、LibOの母体であるThe Document Foundation(TDF)では伝統的に企業での大量導入に勧められるかどうかで安定版か否かを区別していました。具体的には4.1だと4.1.3以降を安定版としています。これはTDFのリリースアナウンスを見ないとわかりませんでした。しかし、4.2の途中からは4.1を安定版、4.2を最新版[注10]としました。大規模導入は4.2.5からお勧めとなり、安定版と大量導入できるかどうかを切り離すことにしたようです。

ただし気を付けなくてはいけないのは、4.1は5月28日でサポートが切れています。それじゃあ大量導入なんてできないじゃないかというツッコミはそのとおりで、「そのような場合にはTDFはL3サポートを提供している企業とサポート契約を結んでください」という立場をとっています。L3サポートとは、簡単に言えば使い方とかだけではなくソースコードレベルでのサポートのことです。大量導入したいけどサポート契約は結びたくないという場合は、X.Y.5か6を半年ごとに更新していけばいいということになります。もっとも、Ubuntuの場合はそのリリースに含まれているものを継続使用しないといけないことになりますので、安定版云々は関係ありません。当然のことではあるのですが、具体的にはUbuntu 12.04では3.5、14.04では4.2です。3.5は公式には2012年11月にサポートが切れていますが、12.04では2017年4月までは継続されます。

4.3を使うには

実のところ最新のLibOを試すには、UbuntuではなくXubuntuやLubuntuと行ったUbuntuフレーバーをインストールし、そこにダウンロード[注11]したオフィシャルビルドをインストールするのが簡単です。UbuntuにインストールされているLibOをアップデートする場合は、PPA[注12]から行ってください。Ubuntu 14.10には4.3.2あたりが搭載されているでしょうから、急がない場合はそれを待つのもいいでしょう。SD

注9) http://ja.libreoffice.org/get-help/mailing-lists/ discuss@でもusers@でもどちらでもけっこうです。

注10) 原文は"Fresh"。

注11) http://ja.libreoffice.org/download/libreoffice-fresh/

注12) https://launchpad.net/~libreoffice

Dで SF小飼弾

第9回

『ファウンデーション――銀河帝国興亡史〈1〉』（アイザック・アシモフ／早川書房）

SFといえば未来。未来といえば予測。そして予測といえば、『ファウンデーション』は外せません。Mac OS XやiOSでアプリを作るときに必ずimportするアレ？　いえいえ。日本では『銀河帝国の興亡』として知られる、アイザック・アシモフの二大シリーズの1つのほうです。

「ガスの分子は多すぎて、ひとつひとつ追跡はできないけれども、統計的にまとめることでその振る舞いを理解し予測できる。ならば人間ひとりひとりは予測できなくとも、人類全体ならば同様に理解し予測できないか？」――ハリ・セルダンはそう仮説し、そしてついにその仮説を理論にすることに成功しました。これが、[心理歴史学]（サイコヒストリー）。セルダンは理論にとどまらず、その理論を彼の住む人類世界、銀河帝国に適用してみました。その結果得られたのは、銀河帝国の崩壊は避けられず、そしてその混乱は一万年に及ぶというものでした。自らの予言に戦慄しつつも、人類の将来を案じたセルダンは、一万年の暗黒時代をわずか一千年に縮める方法を見つけます。

それが、原題ともなった[基盤]（ファウンデーション）。これまでの人類の叡智を、戦乱の及ばぬ辺境の星にまとめ、それを[銀河百科事典]（エンサイクロペディア・ギャラクティカ）としてまとめるのです……。ローマ人もびっくりのスケールですね。ノーベル経済学賞を受賞したポール・クルーグマンは、心理歴史学者になりたくて経済学を志したそうですが、このスケールがどれほどの読者の心をわしづかみにしたのかは、一読すればわかります。

とはいうものの、20世紀の今読み返すと、経年劣化も否めないのが正直なところでもあります。はるか未来の登場人物たちなのに、タバコふかしまくってるわ、単位系がヤードポンド法だわといったレトロさは愛嬌としても、「個はわからずとも全ならわかる」という着想そのものが現実に追い抜かれている感もあります。今や流体シミュレーションでは空気をガスではなく分子の集まりとしてモデル化してますし、銀河百科事典はWikipediaとして実現した感がありますし、ネットの巨人たちはビッグデータで個人を追っています。「あなたの本の読者は、こんな本も読んでいます」とAmazonに言われたらアシモフはどう反応したのか、見てみたかったなあ……。

このように「偉大なる失敗」としても読める同作ですが、実はこのあとさらに大きな失敗が待っています。次回はより大きなその失敗について語ることにしましょう。SD

題字・イラスト／aico

第30回

Linux 3.15の新機能
fallocate、cross rename、VMA cache

Text：青田 直大　AOTA Naohiro

今月もひき続きLinux 3.15について解説していきます。Linux 3.15はコミット数が多かったため紹介する機能追加も多くなっています。今回はファイルシステムに関連する2つの変更、「fallocateシステムコール」の新たなフラグとrenameの新機能「cross rename」、そして「VMA cache」について取り上げていきます。

fallocateシステムコールの進化

まずは「fallocateシステムコール」について解説します。このシステムコールはもともと、その名前のとおり「ファイルの指定した領域に対応するディスク領域を確保する」ためのシステムコールでした。すなわち、ファイルに書き込みを行う前にディスク領域を確保しておくことで、確保した区間への書き込みはファイルシステム上の容量が足りないという理由で失敗することがなくなります。また、事前に領域が確保されることで、連続した領域がディスク上でも連続して確保されやすくなり、パフォーマンスの改善にもつながります。

このfallocateが、ファイル領域に関する操作をより効率に行うためのシステムコールとして近年進化を続けています(**図1**)。たとえば、Linux 2.6.38からはFALLOC_FL_PUNCH_HOLEフラグを使った"punch holing"が実装されました。これは元のfallocateとは逆に、ファイルのディスク上に割り当てられている領域に「パンチ穴を開けて」ディスク上の領域を解放する処理を行います。この「穴を開けられた」領域は、アプリケーションには0が連続した領域に見えます。0で埋められた領域を解放することでディスクの節約につながります。

▼リスト1　hole-write.c

```
 1  #include <unistd.h>
 2  #include <sys/types.h>
 3  #include <sys/stat.h>
 4  #include <fcntl.h>
 5  #include <stdio.h>
 6
 7  #define BUF_SIZE 4096
 8  static char buf[BUF_SIZE];
 9
10  int main()
11  {
12    int i;
13    int fd = open("file.img", O_RDWR);
14    lseek(fd, 1024*1024*1024, SEEK_SET);
15    for(i = 0; i < 512*1024; ++i) {
16      int r = write(fd, buf, BUF_SIZE);
17      if (r != BUF_SIZE) {
18        perror("write");
19        return 1;
20      }
21    }
22    fsync(fd);
23    close(fd);
24    return 0;
25  }
```

Linux 3.14からはさらにFALLOC_FL_ZERO_RANGEとして、指定した領域をI/Oなしに0埋めする機能が追加されました。一見、punch holingと似ていますがパンチホールではその領域に相当する部分がディスク上から解放されてしまうのに対して、zero rangeではディスク領域が解放されないことが異なっています。

そして、Linux 3.15ではFALLOC_FL_COLLAPSE_RANGEというフラグが追加されました。このフラグを使うと、指定した領域が削除され、さらにその領域よりも後ろの部分が削除された分だけで前につめられます。

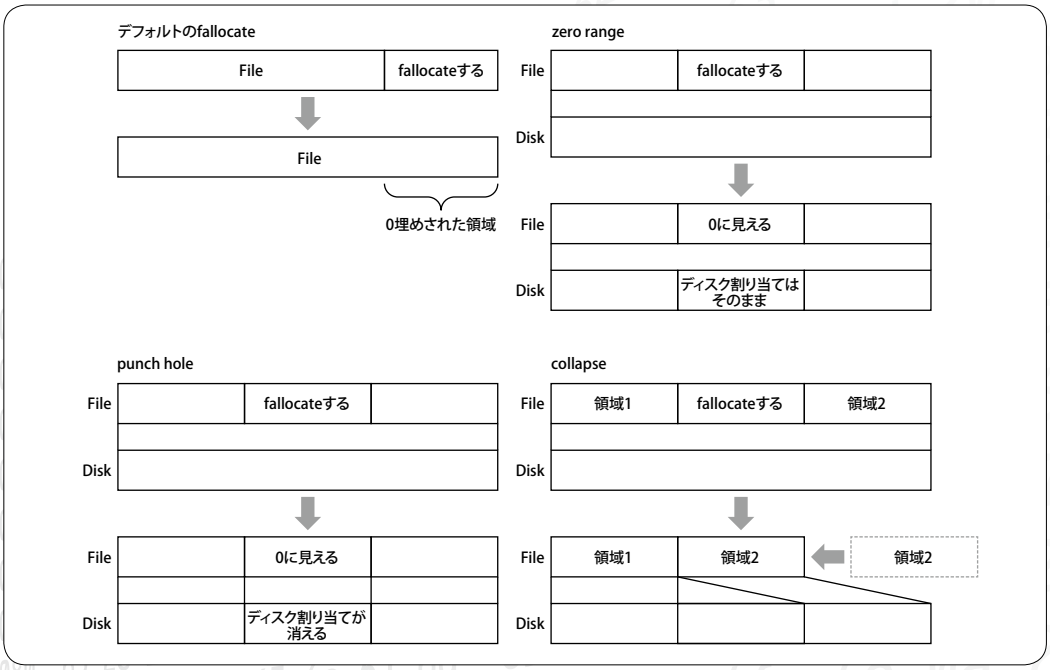
それではfallocateを実際に使ってみましょう。4GBのファイルを作成し、その中の中間の2GBを0埋めするという作業をwriteとfallocateでやってみましょう。writeを使うプログラムとしてhole-write.c(**リスト1**)を、fallocateを使うプログラムとしてhole-{punch,zero,collapse}.c(**リスト2**)を作成します。ここではhole-write.cとhole-punch.cだけを示していますが、hole-{zero,collapse}.cはhole-punch.c(**リスト2**)の12行目のfallocateのフラグが**表1**に示したように書き換わっているだけです。

実行してみると、writeの場合に比べてfallocateの場合が格段に速く効率が良いことを確認できます(**図2**)。また、ファイルサイズに注目するとcollapseの場合には、collapseが指定範囲を抜き取る操作であることからファイルサイズが2GBになっていることが確認できます。

▼リスト2　hole-punch.c

```
 1  #define _GNU_SOURCE
 2  #include <unistd.h>
 3  #include <sys/types.h>
 4  #include <sys/stat.h>
 5  #include <fcntl.h>
 6  #include <stdio.h>
 7  #include <linux/falloc.h>
 8
 9  int main()
10  {
11    int fd = open("file.img", O_RDWR);
12    int r = fallocate(fd, FALLOC_FL_➥
PUNCH_HOLE | FALLOC_FL_KEEP_SIZE,
13        1024*1024*1024, (off_t)➥
2*1024*1024*1024);
14    if(r<0)
15      perror("fallocate");
16    fsync(fd);
17    close(fd);
18    return 0;
19  }
```

▼図1　fallocateの進化



同様にディスク上のサイズはpunchとcollapseの場合に元より減って2.1GBになっていることがわかります。最後にsha1sumの結果を見てみましょう。write、punch、zero rangeの場合はどれも読み込み側からは指定した範囲が0で埋められているように見えてsha1sumの結果が一致しています。一方でcollapseではファイルの内容がほかのものとは変わっていて、sha1sumの結果も異なっています。

cross rename

次にLinux 3.15から追加された、renameの拡張である「cross rename」を解説します。renameはすなわちmvコマンドで使われているファイル名を変更するシステムコールです。mvコマンドを使ったことがあれば、すぐにわかるように、もしも変更先に指定した名前と同じファイルが存在していた場合、そのファイルはアトミックに変更元のファイルに置き換えられます。

これに対して、今回追加された「cross rename」はアトミックに変更元と変更先のファイル名を入れかえます。この機能を使うことでたとえば、随時openされて書き込みがあるようなログファイルと空ファイルを入れ替えて、アーカイブを作るといったようなことができるようになります。そのほかにも、実はこの機能はoverlayfsでさまざまな操作をアトミックに行えるように活用することが予定されています。

overlayfsとは2つの任意のディレクトリツリーを「上下に重ねて」新しいファイルシステムツリーを見せることができるファイルシステムです(**図3**)。2つのディレクトリツリーに同じ名前のファイルがあれば上のツリーのファイルが見え、下のファイルは隠蔽されます。また、上下に同じ名前のディレクトリがある場合はそのディレクトリは上のディレクトリのファイルも下のディレクトリのファイルも持っているように見えます。たとえば、これをLiveCDの読み込み専用のルートディレクトリを下に、USBメモリ上の書き込み可能なファイルシステム上のディレクトリを上に重ねれば、LiveCDにもともとあるファイルを使いつつ書き込みはUSBメモリ上に行われ、通常であれば

▼図2　fallocateのテスト実行結果

```
$ dd if=/dev/urandom of=file.img bs=4096 count=1048576
$ cp file.img file.img.bak

$ cp file.img.bak file.img
$ time ./hole-write; sha1sum file.img; ls -lhs file.img
./hole-write  0.02s user 1.22s system 3% cpu 32.320 total
ab94270ef0c71e4e20e5eeda884879fa28d255cb  file.img
4.1G -rw------- 1 naota naota 4.0G Jul 15 14:22 file.img

$ cp file.img.bak file.img
$ time ./hole-punch; sha1sum file.img; ls -lhs file.img
./hole-punch  0.00s user 0.39s system 95% cpu 0.405 total
ab94270ef0c71e4e20e5eeda884879fa28d255cb  file.img
2.1G -rw------- 1 naota naota 4.0G Jul 15 14:25 file.img

$ cp file.img.bak file.img
$ time ./hole-zero; sha1sum file.img; ls -lhs file.img
./hole-zero  0.00s user 0.39s system 93% cpu 0.411 total
ab94270ef0c71e4e20e5eeda884879fa28d255cb  file.img
4.1G -rw------- 1 naota naota 4.0G Jul 15 14:29 file.img

$ cp file.img.bak file.img
$ time ./hole-collapse; sha1sum file.img; ls -lhs file.img
./hole-collapse  0.00s user 0.57s system 95% cpu 0.603 total
1270b3a98e83a1ba84c464b0624d73348d681498  file.img
sha1sum file.img  9.21s user 0.61s system 49% cpu 19.993 total
2.1G -rw------- 1 naota naota 2.0G Jul 15 14:33 file.img
```

▼表1　実行結果のまとめ

方法	時間(秒)	ファイルサイズ	ディスク上のサイズ	fallocateのフラグ
write	32.320	4.0GB	4.1GB	-
punch	0.405	4.0GB	2.1GB	FALLOC_FL_PUNCH_HOLE,FALLOC_FL_KEEP_SIZE
zero range	0.411	4.0GB	4.1GB	FALLOC_FL_ZERO_RANGE,FALLOC_FL_KEEP_SIZE
collapse	0.603	2.0GB	2.1GB	FALLOC_FL_COLLAPSE_RANGE

読み込み専用のLiveCD上のシステムをあたかも書き込みもできるシステムのように使うことができるようになります。

overlayfsで問題になるのは下のディレクトリツリーのファイル削除です。下のほうには書き込みができないので、上のディレクトリツリーに「削除した」という意味のファイルやディレクトリを作らなければいけません。たとえば、ファイルを削除するには、major/minor=0/0であるキャラクタデバイスファイル(whiteoutと呼ばれます)を同名で上のツリーに作成し、ディレクトリを削除するにはxattr(拡張ファイル属性)の"trusted.overlay.opaque"を"y"に設定したディレクトリ(opaqueディレクトリと呼ばれます)を作成します。上のツリーでwhiteoutやopaqueディレクトリを見つけると、overlayfsは下のツリーの同名のファイルやディレクトリを無視し、whiteoutやopaqueファイル自体も無視してoverlayfsのユーザに見えないようにします。

このように削除をwhiteoutやopaqueディレクトリによって実現するために、本来であればアトミックに実現できていた操作がoverlayfsではアトミックではなくなる(そのために特殊な処理を上のツリーのファイルシステムに導入する必要がある)という問題がありました。具体的に下のツリーに存在するfooというファイルを削除し、同名のディレクトリを作成するという2つの操作を連続して行う場合を考えてみましょう。一般のファイルシステムであれば、

- fooの削除 → unlink foo
- ディレクトリfooの作成 → mkdir foo

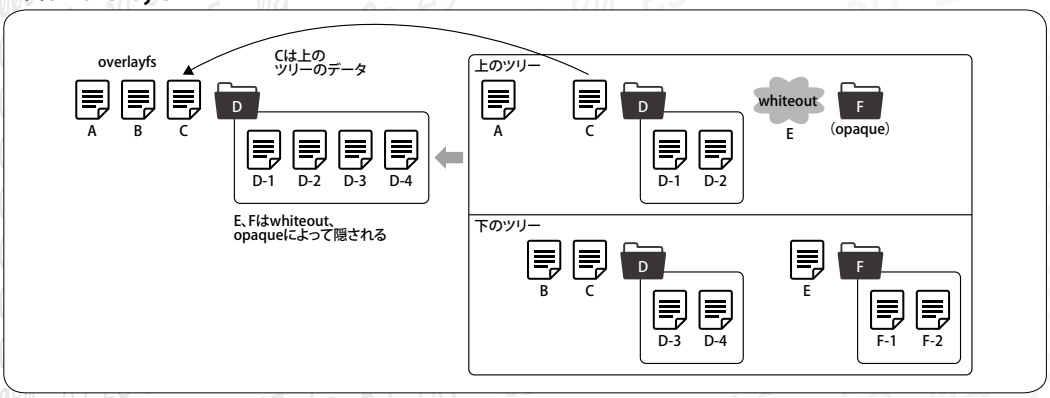
という操作が行われ、unlinkの前には「fooというファイルが存在する状態」であり、unlinkのあとからmkdirの前までは「fooというファイルもディレクトリも存在しない状態」であり、mkdirのあとは「fooというディレクトリが存在する状態」というように推移します。一方で、cross renameがない場合のoverlayfsの動きについて考えてみましょう。workdirはoverlayfsによって使われる作業ディレクトリで、読み込み時にはoverlayfsによって見えなくされる特殊なディレクトリとします。そうするとoverlayfsでは次のような操作が必要になります。

- fooの削除
  whiteout foo(fooという名のwhiteoutを作成)
- ディレクトリfooの作成
  rename foo workdir/foo.X(fooというwhiteoutを作業ディレクトリの中にどかす)

```
mkdir foo
unlink workdir/foo.X
```

ひとつひとつfooの状態について考えてみましょう。whiteoutの前までは「fooというファイルが存在する状態」です。whiteoutから"rename foo workdir/foo.X"までは「fooというファイルもディレクトリも存在しない状態」です。その後"mkdir foo"までの期間にもう一度「fooという

▼図3 overlayfs

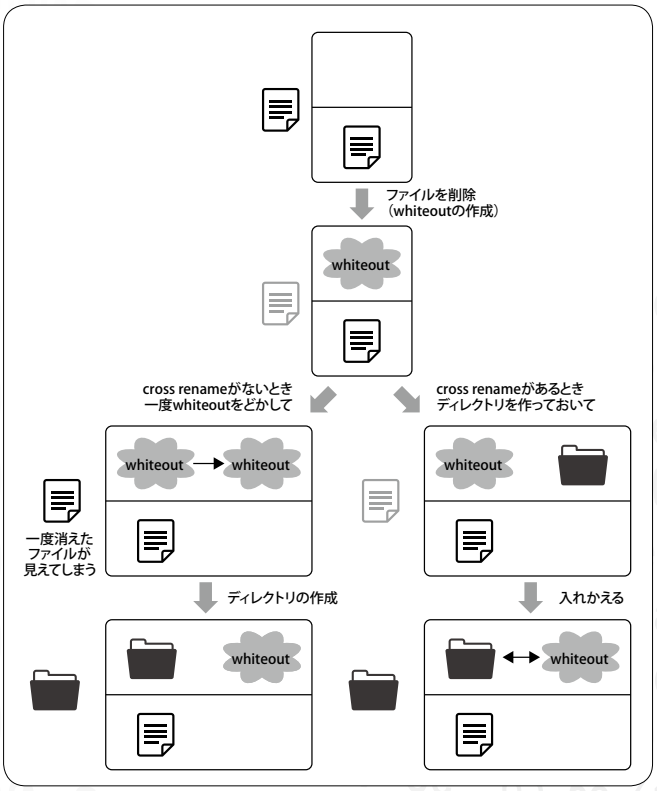


ファイルが存在する状態」が起きます。そして、“mkdir foo”が終われば「fooというディレクトリが存在する状態」となります。このようにcross renameがなく特殊な操作を使わないと、“fooの削除”を行ったあとに、再びfooが存在する瞬間ができてしまいます。ここでもしcross renameを使うことができれば、ディレクトリfooの作成を次のように行うことができます。

- mkdir workdir/foo(作業ディレクトリの中にfooを作成)
- cross-rename workdir/foo foo
- unlink workdir/foo

こうするとcroos-renameの前までは「fooというファイルもディレクトリも存在しない状態」であり、cross-renameのあとには「fooというディレクトリが存在する状態」となり、一般のファイルシステムでの状態遷移との一貫性をとることができるようになります(**図4**)。

▼図4　cross renameの有無による動作



このほかにも、上のツリーに存在するディレクトリを削除し、下のツリーのファイルを見せないために、ディレクトリを削除すると同時にwhiteoutを作成しなければいけない場合や、下のツリーに存在するファイルと同名の上のツリーのファイルをリネームする場合(上のツリーで元ファイルをwhiteoutしなければいけない)に、このcross renameを用いることで、これまではアトミックに行うことができなかったこれらの操作をアトミックに行うことができます。

VMA cacheの改善

次に紹介するのはVMAキャッシュの改善です。各プロセスはそれぞれに固有の仮想アドレス空間を持っています。この仮想アドレスの、どの領域がどのようにファイルやスタックに割り当てられているかは“cat /proc/<PID>/maps”するとみることができます。プロセスが自分の仮想アドレス空間にアクセスすると、アクセスしたアドレスにちゃんと物理メモリが割り当てられていればプロセスは何ごともなくそのアドレスにアクセスすることになります。

しかし、物理メモリの割り当てはシステムの状態によって、swapされたりキャッシュから落とされたりと変わっていくので、アクセスしてみても物理メモリが割り当てられていない場合(ページフォルト)もあります。そういうときには、カーネルが物理メモリの割り当ての作業を行わなければいけません。

では、どこからそのアドレスにあるべきデータを読み込むのか、あるいはそもそもそのアドレスは有効なのかといった情報を管理するために、カーネルはVMA(virtual memory area)を管理しています。VMAには仮想アドレス空間のどこからどこが

どのようなアクセス権限で割り当てられているのか、どのファイルのどこかから割り当てられているのかといった情報が含まれています(**図5**)。

さて、このようにカーネルはページフォルトのたびに、そのアドレスを管理するVMAの情報を取得しなければいけません。この作業を効率的に行うために、VMAを赤黒木[注1]で管理し、また最後にアクセスしたVMAの情報をキャッシュしておくという方法を使い赤黒木の探索の手間さえも省略しようとしていました。

Linux 3.15ではこのVMAのキャッシュ方法が改善されています。具体的にはスレッドごとに最大4つのVMAをキャッシュすることにしました。まず、当たり前ですがキャッシュ数を増やしたことでキャッシュのヒット率が改善されます。さらに、プロセスごとのキャッシュをスレッドごとにしたことで同じアドレス空間を共有するものの、別々のアドレスに動作することが多いマルチスレッドアプリケーションでのキャッシュヒットが改善されています。たとえばパッチのメール[注2]によれば、システムの起動時のキャッシュヒット率が50%から73%に改善され、必要なCPUサイクル数も199億サイクルから136億サイクルに減少しているということが言われています。

Eudyptula Challenge

最後にEudyptula Challenge[注3]について紹介します。これはLinuxカーネルのプログラミングについて、カーネルモジュールのHello WorldからLinuxカーネルのメインツリーにpatchを受け入れてもらうレベルまで学ぶことができるインターネット上のコースです。このコースでの学習はすべてE-mailを使って行われます。Webサイトに記載してあるメールアドレスに参加したい旨を送ると、最初の問題とどのようにこのコースが行われるかのメールが返信されてきます。そして問題を解いて、また次の問題が送られてくる……という過程がすべてE-mailを通じて行われます。E-mailを使うのはLinuxカーネルの開発も、すべてがE-mailを使って行われるからだそうです。LinuxカーネルHackを始めてみたいけれど、何をどうしたらいいのかわからないという方はこのEudyptula Challengeから始めてみてはいかがでしょうか。SD

注1) 2色木(red-black tree)。マップ(連想配列)の実装の1つ。

注2) http://thread.gmane.org/gmane.linux.kernel.mm/114050

注3) http://eudyptula-challenge.org/

▼図5 VMAの割り当て

```
$ cat /proc/self/maps
00400000-0040b000 r-xp 00000000 00:10 5926                               /bin/cat
0060b000-0060c000 r--p 0000b000 00:10 5926                               /bin/cat
0060c000-0060d000 rw-p 0000c000 00:10 5926                               /bin/cat
0222c000-0224d000 rw-p 00000000 00:00 0                                  [heap]
7fbf67651000-7fbf6db7c000 r--p 00000000 00:10 498149                     /usr/lib64/locale/
locale-archive
7fbf6db7c000-7fbf6dd19000 r-xp 00000000 00:10 498065                     /lib64/libc-2.19.so
7fbf6dd19000-7fbf6df18000 ---p 0019d000 00:10 498065                     /lib64/libc-2.19.so
7fbf6df18000-7fbf6df1c000 r--p 0019c000 00:10 498065                     /lib64/libc-2.19.so
7fbf6df1c000-7fbf6df1e000 rw-p 001a0000 00:10 498065                     /lib64/libc-2.19.so
7fbf6df1e000-7fbf6df22000 rw-p 00000000 00:00 0
7fbf6df22000-7fbf6df43000 r-xp 00000000 00:10 498073                     /lib64/ld-2.19.so
7fbf6e128000-7fbf6e12b000 rw-p 00000000 00:00 0
7fbf6e141000-7fbf6e142000 rw-p 00000000 00:00 0
7fbf6e142000-7fbf6e143000 r--p 00020000 00:10 498073                     /lib64/ld-2.19.so
7fbf6e143000-7fbf6e144000 rw-p 00021000 00:10 498073                     /lib64/ld-2.19.so
7fbf6e144000-7fbf6e145000 rw-p 00000000 00:00 0
7fffa884f000-7fffa8870000 rw-p 00000000 00:00 0                          [stack]
7fffa89fe000-7fffa8a00000 r-xp 00000000 00:00 0                          [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0                  [vsyscall]
```

September 2014

NO.35

Monthly News from jus

Japan UNIX Society

日本UNIXユーザ会　http://www.jus.or.jp/
内山 千晶　UCHIYAMA Chiaki　chiaki@jus.or.jp
法林 浩之　HOURIN Hiroyuki　hourin@suplex.gr.jp

同日開催! 大阪vs札幌!

以前に比べるとITコミュニティが主催するイベントが激増し、開催日が重なることも珍しくなくなりました。しかし、jusはなぜか団体内でも行事が重なってしまうことがあります。今回は6月14日に行った2つのイベントを報告します。

2014年6月jus勉強会（大阪）

■jus & USP友の会共催
シェルワンライナー勉強会＠関西
【講師】上田 隆一（USP友の会／産業技術大学院大学）
【司会】榎 真治（日本UNIXユーザ会）、
齊藤 明紀（日本UNIXユーザ会）
【日時】2014年6月14日（土）13:30～18:00
【会場】ECCコンピュータ専門学校 1号館4階

大阪で行った勉強会は、関西地区初開催のシェル芸勉強会。参加者は学生から一般まで18人、会場はほぼ満員となりました。

シェル芸とは、UNIXシェル（おもにbash）のワンライナーを駆使して文字列加工を自由自在に操る技です。シェル芸を磨くことで文字列の整形や数値の集計、特定文字の抜き出しなど、データの加工がすばやく、大量に実行できるようになります。

勉強会はまず講師の自己紹介から始まりました。上田さんは日々、シェル芸の普及活動にあたられていますが、専門はロボット工学です。本業は産業技術大学院大学で教鞭を取っていらっしゃいます。また、民間企業のアドバザリーフェローでもあり、活躍場所は多岐に渡っています。

勉強会の前半は座学です。「ソフトウェアツールとAWK・sedについて」という表題でUNIXの歴史、テキストを操作するコマンド、AWK、sedについての説明がありました。理解が深まったところで4問の問題を解くことになりました。

お題の例
問題1：次のechoの出力にパイプをつなげて、カンマで区切られた数を足し算してください。

```
$echo -12,135,123,135,123
```

読者のみなさんはこのお題、どのように取り組まれるでしょうか。回答例として、まずカンマをsedの正規表現かtrで取り除く。その次にawkで加算、または＋を正規表現で挿入し結果をbcに渡す、などの方法がありました。

後半は初学者と上級者がペアとなって実践問題に取り組みました。提示された問題は次のとおり。

問題5：CSVに保存されている数字の加算（桁がカンマで区切られダブルクォーテーションでくくってある数字の文字列あり）
問題6：3次正方行列の転置
問題7：IPv6のアドレスから省略した0を復元
問題8：問題7の応用

シェル芸勉強会の特徴は、誰もが手を動かして参加することにあります。後半の問題は初学者には難しいですが、ペアとなった上級者の取り組みを横で見たり、語句の説明を受けたりするうちに慣れ親し

むようになります。上級者もほかの回答で新たな発見を得られるようです。また、Twitter経由の回答を見るのも醍醐味の1つ。遠隔からの達人たちのすばやい回答でも会場がにぎわいました。中にはピュアシェル芸と称してawk、sedを使わず解く回答もあり、どよめきが聞こえてくることも。上田さんのユーモア溢れる講評とともに終始笑いに包まれながら、勉強会は盛況のうちに終わりました。

jus研究会札幌大会

■元IT技術者が田舎暮らしを始めたら
～田舎のネットワーク事情と様々な工夫～
【講師】岡 善博
【司会】法林 浩之（日本UNIXユーザ会）
【日時】2014年6月14日（土）15:15～16:00
【会場】札幌市産業振興センター 技能訓練棟3階

講師の岡さんは、以前はjus関西UNIX研究会などによく参加されていましたが、現在は北海道・十勝地方の中札内村にお住まいです。都会を離れた土地におけるIT事情について話をしていただきました。39人という多くの方の参加がありました。

はじめは岡さんの経歴紹介です。まずPET 2001、VIC-1001、PC-9801、PC386LSなどのコンピュータと出会い、次に職場（松下電器）のUNIXマシンにアカウントをもらってインターネットと出会いました。当時はNetNewsやメーリングリストがさかんで、そこで世界が広がりました。fj.forsaleでSun Microsystems社（以下、Sun）のワークステーションが売りに出ていたので購入し、当時は商用インターネットがまだなかったのでJuiceに接続しました。それからFreeBSDとの出会いも重要で、ノートPCはWindowsとFreeBSDのデュアルブートにし、自宅や職場のサーバでも利用しました。

続いては北海道への移住の話です。もともと北海道が好きで、定年後は移住したいと考えていたところ、中札内で村営分譲宅地が売られていたのを気に入り、購入して自宅を建てました。定年後に建てたのでは遅いので、定年前に建て、定年までは大阪と往復しながら生活しました。自宅にはISDN、LAN、電話、電源などの口を備えたコンセントを導入。建築当初はISDNでの接続でしたが、その後はADSLを経て現在はBフレッツで接続しています。建築当時は中札内には光ファイバーは来ないかもしれないと思っていたそうですが、一応設備だけは用意しておいたのが後年役に立ちました。

中札内あたりのIT事情としては、市街地ではBフレッツが使えますが、ADSLのままの人も多いとのこと。理由として、局舎に近い場所ではある程度の速度が出ることと、Bフレッツは集合住宅では安いですが一戸建てでは高いというのがあります。また、市街地から離れるとADSLは厳しく、ワイコムのAir5Gという無線インターネットがよく使われます。帯広市内では、固定回線を捨ててWiMAX一本の人もいますが、中札内にはWiMAXは来ていません。道央の長沼町では自治体で補助金を得て街全体に光ファイバーを敷設し、郊外は無線で接続できるようにするという取り組みを行っています。

さらに自宅サーバの遍歴が紹介されました。最初はSunやLunaなどワークステーションを使っていましたが、その後はDigital HiNote UltraやLet's noteなどのノートPCにFreeBSDをインストールして運用しています。現在はGoogle Appsも併用しています。自分でnakasatsunai.jpドメインも運用していて、DNSの設定なども見せていただきました。最近はiPadを手に入れたのを契機に自宅サーバをUTF-8化し、iPadでサーバにログインしてEmacsとMewでメールを読み書きしています。

最後に、田舎暮らしの経験から得られたこととして、買い物は通販があるので困らない、ITの仕事はない、IT事情を考慮するなら田舎でも市街地に住むほうが良い、といったコメントで講演を締めくくりました。

都会のIT事情を聴ける機会はあっても、田舎のIT事情を聴ける機会はさほど多くありません。とくに自宅を建てる際の考慮点は非常に参考になるものであり、貴重な研究会であったと思います。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

第33回

Race for Resilienceハッカソン

●Hack For Japanスタッフ
及川 卓也　OIKAWA Takuya
Twitter @takoratta
関 治之　Hal Seki
Twitter @hal_sk
高橋 憲一　TAKAHASHI Kenichi
Twitter @ken1_taka

“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「発展途上国×防災・減災」をテーマにしたグローバルハッカソンの報告です。

Race for Resilienceとは

Race for Resilieneは「発展途上国×防災・減災」をテーマにしたグローバルハッカソンです。実際にNPOやNGO、世銀関連機関の発展途上国の防災・減災活動に取り入れられるようなソフトウェア、ハードウェアをつくりあげ、自然災害に対してしなやかな社会をつくることを目指して開催されました。世界銀行の主催でアジア数ヵ国およびハイチ、ロンドンでのハッカソンの後、各国の優秀プロダクトはグローバル審査に進み、6月30日にロンドンでグローバルアワード表彰式が行われました。日本では2014年2月8日と9日に石巻、東京、名古屋の3ヵ所でハッカソンが開催されました。

雪の中でのハッカソン

ハッカソンが行われた2月8日と9日は、石巻と東京の会場ではちょうど大雪に見舞われた日で、とくに石巻では91年ぶりと言われるほど大量の積雪があり、1日目の夜には**写真1**のようなかまくらを作って、その中で開発ができるほどでした。東北とはいえ太平洋側沿岸に位置する石巻では、これほどの雪が降ることは滅多にないため交通機関もストップし、東京から参加していたHack For Japanスタッフの及川と高橋は1日延泊を余儀なくされるという状況でした。

グローバル審査

グローバル審査に進むことになったプロジェクトはハッカソン終了後も引き続き開発が続けられ、最終提出期限が迫っていたことから4月末から5月にかけてのゴールデンウィークも返上で各プロジェクトのメンバーが頑張っていました。

Hack For Japanの取り組みでもたびたび課題として挙がるのですが、ハッカソンでは1日、もしくは2日間のイベント終了後はプロジェクトが継続されないことも多いなか、今回は各会場で、国内、グローバルという複数段階での審査が、ある程度の期間をあけて設けられていたこともあって、複数のプロジェクトで継続して開発が進められたことも特筆できる点だと思います。

グローバルアワードの最終発表と表彰式は、6月30日にロンドンで行われた「Understanding Risk Forum[注1]」の中で行われ、ファイナリストとして残った10のプロジェクト[注2]の紹介もされました。グランドプライズ[注3]には、インドネシアからの

◆写真1　かまくらの中で開発

注1　https://www.understandrisk.org/UR2014
注2　http://www.codeforresilience.org/blog/announcing-code-resilience-finalists
注3　http://www.worldbank.org/en/news/feature/2014/06/30/innovative-apps-for-disaster-risk-reduction-win-global-attention

「Jakarta Flood Alert」「Quick Disaster」という2つのプロジェクト、日本からの「逃げ地図」プロジェクトが選出されました。

各プロジェクトについて

グランドプライズは逃してしまいましたが、ここではHack For Japanスタッフがかかわった3つのプロジェクトの解説をしたいと思います。

Survival Toolboxプロジェクト

Survival Toolboxプロジェクトは、石巻在住の中塩成海さんが自身の被災体験を基に2月のRace for Resilience石巻会場でのアイデアピッチで提案したプロジェクトです。

中塩さんは東日本大震災のときに、身近なものを組み合わせるなど創意工夫して困難な状況を乗り切りました。例として自転車を使った発電のアイデアなどを披露し、このようなアイデアは日頃から考えて試してみるのが良いのではないかということで、アイデアをレシピとして公開し共有するプラットフォームを作りたいと仲間を募りました。

その提案に賛同して、中塩さんの妹さんとその友人(お2人とも石巻在住の女子高生です)、東京在住のソーシャルメディアの研究者、仙台在住のフリーランスエンジニア、ドイツ人ITコンサルタントがメンバーとして加わりました。さらには、現地で参加はできなかったのですが、横浜からはデザイナーも加わって開発が進みました。

結果、現地での審査では見事2位になり、ロンドンを目指すことになります。

その後、関西在住のフリーランスエンジニアも加わり、プロジェクトはハングアウトとFacebookグループを介したオンラインでの開発をベースとして進みます。学業を優先させる中塩さんの事情などもあり、開発は必ずしもスムーズには進みませんでしたが、審査に必要なサイトやオープンソースのレポジトリ、Code for Resilienceへの登録などをメンバーで手分けして行いました。グローバル審査では残念ながらファイナリストには選ばれなかったのですが、現在もサービス開始に向けて開発を進めています。

このプロジェクトの特徴は平時と発災時においてインターフェースを変化させる点です。防災・減災用のアプリケーションやサービスには、普段使っていないものをいかに災害時に使えるようにするかという課題があります。平時にはアイデアを競い合うような楽しさを取り入れ、発災時には共有されているアイデアをその災害の内容や段階に応じて、積極的に提示するという形にしようと検討しています。

また、利用者の分析も行いました。災害発生後の状況では、被災者自身がスマートフォンやWebを使ってアイデアを検索する余裕があるとは考えづらいこともあり、このサービスはもっぱら支援者を対象としています。支援者が被災者に必要と思われる情報を伝達するために、アウトプットとしては紙も想定しており、その紙を印刷したり、FAXで送信したりすることを考えています。

グローバル審査ではAndroidアプリケーションとして動作するものを提出しましたが、これはあくまでも短期間のプロジェクトの成果としてのプロトタイプのようなものであり、アイデアを共有する部分を含むバックエンドやほかのインターフェース(Webや紙)での入出力はまだまったく手を付けられていません。

本プロジェクトに賛同する方はぜひともプロジェクトメンバーにコンタクトしてみてください。プロジェクトのサイトはこちらになります。

URL http://survivalpad.net

Flood ARプロジェクト

Flood ARは2月のハッカソンの石巻会場で1位となり、グローバル審査に進んだプロジェクトです。その後ファイナリストには残ったものの、惜しくもロンドンへ行くチャンスは逃してしまいました。

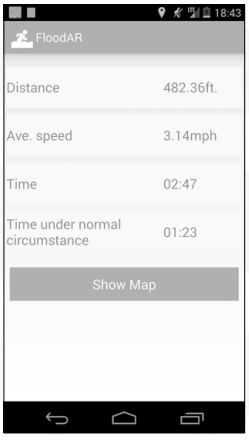
これはAndroid端末用のアプリケーションで、津波の発生した状況下での避難をシミュレートするものです。このアプリケーションの着想は、事前に行ったアイデアソンで石巻の高校生が話してくれた

被災時の体験が基になっており、たとえ膝の高さ程度の浸水でも歩行には困難を伴うこと、また実際の水の色はほぼ黒で地面は見えず、足下に何があるかわからないという危険があることをAR（拡張現実）を用いて画面上に表現します。今までの津波対策の避難訓練は津波が来る前に逃げることを想定したものでしたが、実際に東日本大震災では警報が届かなかったり、警報があっても逃げ遅れてしまったなどで多くの被害者が出ました。このアプリケーションを使うことで、津波の脅威をより身近に感じてもらうことができます。

起動すると、性別、年齢、身長、そして避難先の位置をマップ上で指定して入力します。Startボタンでシミュレーションを開始するとAR画面になり、性別に応じた自分自身を示すアバターが表示され、身長に合わせた高さの浸水が表現されます（**図1**）。このアバターはアプリの使用者の状態に応じてアニメーションするようになっており、止まっているときは周囲を見渡す動作、ほかに歩く、走る際はそのスピードに合わせたアニメーションをします。

◆**図1　AR画面**

◆**図2　結果表示**



シミュレーション実行中は定期的に自分の現在地をGPSから取得して移動経路を記録し、移動速度を算出しており、浸水している状況下での歩きにくさや、通常のスピードで歩くことができないことを示すため、想定より速い速度で歩いていると判定された場合は、もっとゆっくり歩くように促すメッセージが表示されます。そして、避難先として設定した地点に到着すると結果表示画面に切り替わり、実際にここまで移動するのにかかった時間と想定時間、実際の歩いてきた経路をマップ上で確認することもできます（**図2**）。

●実装裏話

今の潮流ではこのような3Dグラフィクスのアニメーションを活用したアプリケーションを作る場合はUnityを使うケースが多いと思うのですが、今回は標準のAndroid SDKにてJavaで開発を行いました。開発にかかわったメンバーが通常のJavaでのAndroidアプリ開発に慣れていることと、現在地情報の取得やGoogle Maps APIの活用を考慮するとJavaから3Dグラフィクスをコントロールしやすいようにしたほうが良いと判断したためです。とはいえ、OpenGL ESを直接使用して人型のアバターにアニメーションをさせるにはかなりの開発時間を必要とし、現実的ではありません。

そこで今回は、オープンソースで公開されているライブラリを活用することにしました。当初のバージョンではJavaのみで書かれたライブラリを使用していたのですが、扱うことのできる3Dモデル、およびアニメーションデータの種類に難があり、Unityがデータをインポートする際にも使われているfbx形式のデータを扱えるよう、最終的にはC++で書かれたライブラリをNDKを用いてJavaからコントロールできるようにして実現しました。

また、ハッカソン直後の最初のバージョンでは、水の表現も単に青色の半透明の板を置いていただけにとどまっていたのですが、実際の浸水時の水の色はもっと黒に近い色に見えるため、そのような色の水のテクスチャを用意してより実際のものに近い表

現にしました。このようにしてアバターと水面の表現をカメラからのプレビュー画像の上に重ねて、歩いている自分のイメージを画面上に投影できるようにしています。

このプロジェクトの説明と動画はこちらから参照できます。

URL http://www.codeforresilience.org/app/flood-ar-0

すごい避難訓練プロジェクト

すごい避難訓練プロジェクトは、Race for Resilienceの名古屋会場で1位を獲得したプロジェクトです。従来型の避難訓練は、単なる「避難経路の確認」を超えていないという問題意識から、本当に役に立つ避難訓練のあり方を提示する目的で作られました。この記事を書いているCode for Japanの関と、Code for Nagoyaの河合さんのアイデアを元に、デザイナーや大学の先生、ITエンジニアなどが参加したプロジェクトです。その後、すごろくチームをやっていた西村さんが参加し、プロジェクトが続いています。

実際の災害時には、あらかじめ決められた避難経路をたどっていくようなことよりも、さまざまなアクシデントへの対応を求められる場合が多いと思います。たとえば、お年寄りに遭遇する、一緒に避難しているメンバーが怪我をしてしまった、通れるはずの道が火災で通れない、などなどです。このアプリケーションでは、そういったリアルな災害状況をシナリオに沿って体験することで、避難訓練自体を自発的に楽しみながらできるようになっています。

すごい避難訓練はシナリオゲーム型避難訓練になっており、チームリーダーがスマートフォンを持ち、数名のメンバーと共に避難訓練を行います。メンバーはロールカードを持っており、「お年寄り」「子ども」などといった仮想の役割が割り当てられます。避難訓練中にはスマートフォンにイベントが届くようになっており、状況が刻々と変化します(**図3**)。ゲーム参加者は、発生イベントとロールの制約の中で避難を行わなくてはいけません。

発生するイベントには、「火災が発生し、○○通りが通行止め」というものや、「子どもがパニックになり、家に帰りたいとぐずる」など、ロールカードを持っている人に対する指令なども存在します。管理者(ゲームマスター)は、各チームに対して、学習効果が高まるようにさまざまな指令を出していきます。

◆**図3 スマートフォンに送信されるイベント例**

ゲーム終了後には戻ってきて振り返りのミーティングを行うことで、災害時に起きうることや心構えなどについて気づいてもらって終了します。

Webサイトには、今も当時のプロトタイプアプリケーションとプレゼンスライドがアップされていますので興味のある方はご覧ください。

URL http://race2s.opendata-tokai.jp/

残念ながら、このプロジェクトはCode for Resilienceの世界大会での選考からは漏れてしまいましたが、その後、災害救援ボランティア推進委員会の宮崎さんがいる明治大学での避難訓練で、トライアル的に利用をしていただき、好評を得ました。

また、8月に浦安市で行われる予定の、浦安市「立志塾」宿泊型防災研修プログラム(DECO)の中の企画として、バージョンアップした「すごい災害対応訓練」を提供、実施する予定です。コンセプトはそのままですが、機能面については実際のユースケースに合わせシンプル化し、管理画面などを実装することで本番運用に耐えられるようにしています。また、市が提供するハザードマップの組み込みなども行う予定です。本誌が発売されるころには、結果がWebサイトにも公開されていると思います。良ければ探してみていただければと思います。SD

温故知新
IT むかしばなし

第37回

画面表示あれこれ

たけおかしょうぞう　TAKEOKA Shouzou

VRAMとフレームバッファ

現在のPCなどの画面表示は、「フレームバッファ」というものが一般的です。フレームバッファは、1980年代は「VRAM（ブイラム）」と呼ぶのが一般的で、「Video RAM」の略でした。

VRAM（フレームバッファ）は、メモリ空間中のRAM領域に、CPUから情報を書き込むと、それが自動的に画面に表示されるものです。VRAM方式では、基本的にDMA（Direct Memory Access）技術を使用して、表示回路はCPUの動作と無関係にRAMを読み出します。しかし、CPUを止めるDMA方式は、CPUが動作する時間を奪われ、速度が低下します。そのためアクセスが簡単で比較的高速なスタティックRAM（SRAM）をメインメモリ（DRAM）とは別に用意し、通常は表示回路だけがそのSRAMをアクセスし、VRAMの情報の読み書きのときだけ、CPUがそのSRAMをアクセスするような回路を設け、SRAMを実質的に2ポート化していました。

帰線時間の利用

通常、CPUからのアクセスは、表示回路より優先され、表示回路がデータの読み出し中にCPUがアクセスすると、画面表示が乱れました。そこで、その対策として、各社はいろんな工夫をしていました。一般的にアナログテレビの場合、表示の帰線時間（表示回路からのSRAMアクセスがない期間、V-Sync（垂直帰線時間）、H- Sync（水平帰線時間）と呼ばれる）をCPUから検知できるようにして、帰線時間だけCPUからアクセスさせるようにすることが多かったです。タンディラジオシャックのTRS-80 Model-1[注1]（1977年リリースと言われるが、日本上陸はApple IIよりも遅かった、Z80 CPU搭載）や、シャープのMZ-80K（1978年発売、Z80 CPU搭載）はこういう回路を採用しています。

サイクルスチール

いわゆるメジャーパソコンとして1977年、最初に登場したApple IIは、前述の回路とはまったく異なる方式を採用しています。6502は6800系のいわゆる「同期式バス」というもので、システム全体が1つの基本タイミングで動作していました。CPUはその基本タイミングの「表」を使用してメモリアクセスを行い、表示回路はCPUの「裏」のタイミングでメモリアクセスを行うのです。それによって、CPUと表示回路が競合することは絶対になく、CPUは常にフルスピードで走行し、いつでもVRAM領域にもアクセスできました。こういう方式を「サイクルスチール」と呼びました。Apple IIは、280×192ドット×6色の高解像度グラフィックスを実現し、VRAMとして、当時としては多い約8KBのメモリを使用しました（しかも、2組を使用できた）。Apple IIでは、VRAMは通常のメインメモリと、何の差もないので、テキスト表示のみのアプリケーションでは、グラフィックメモリを通常のRAMとして使用しました。

CPUを止めての高速化

時代を遡（さかのぼ）ると、NECのTK-80

注1） TRS80 Mode-1 回路図：http://incolor.inetnebr.com/bill_r/trs80_schematics.htm

(8080 CPU搭載)の8桁LEDもDMAをしていました。これは、メインメモリ(512B:1,024bitのSRAM 4個で構成)の後方を、VRAMとしてシンプルな回路が、CPUを止め、RAMをDMAして表示していました。TK-80も、DMAを止めると実行速度が上がりました。

日本で1979年に発売されたNECのPC-8001(Z80 CPU搭載)は、低解像度のカラーグラフィックを備えたパソコンです。このマシンは、表示のためにμPD3301を搭載していましたが、表示のためにCPUを止めるDMAを行っていました。PC-8001では、画面表示をやめ、DMAを止めると30%以上も高速になったと言われています。このアーキテクチャはPC-8801にも受け継がれました。

キャラクタジェネレータ

8bit、16bit時代のパソコンは、文字表示用に、「キャラクタジェネレータ(略称:キャラジェネ/キャラゼネ)」という、ハードウェアのテーブルを持っていました。CPUは、VRAMには「文字コード」を書き込みます。すると表示回路は、DMAで読み出した文字コードでキャラクタジェネレータから文字パターンを呼び出します。そして、文字を構成するドット(横1列分6〜8ドット)が得られます。それを、1ドットずつ、適切なタイミングでアナログ信号にすると、ブラウン管に点が表示されていく、というしくみです。

8bit時代は、文字表示は40文字×25行程度だったので、VRAM領域は1KB、16bit漢字VRAM時代は、80文字(漢字40文字)×25行だったので、2K〜4KB程度のサイズでした。

PCGとスプライト

PC-8001、MZ-80Kには、「プログラマブルキャラジェネ(PCG)」というものが販売されていました。前述のキャラジェネは、パソコン内ではROMで実現されていましたが、それを高速なSRAMに取り替え、そのSRAMの内容をCPUから自由に書き換えられるようにしたものです。これはその後、シャープX-1(1982年)やMSX(1983年)などで標準装備されるようになった「スプライト」の初期の形です。スプライトは、表示できる座標が文字単位の座標指定ではなく、もっと自由であることが通常です。また、カラーで、その名前の由来である「透明色」を持ち、スプライト同士が重なったときの優先度指定などもあります。

VRAMなしの機種も

VRAMを持たない機械もたくさん存在しています。たとえば、ファミコンなどの古いゲーム機はVRAMなしで動作しています。前述の「スプライト」と「背景」は、少量のRAMか、カセットのROM上にありました。背景は、小さなパターンの繰り返しでROM上にあり、RAMは少ししかありません。背景のスクロールなどはハードウェアで行えます(回路として非常に簡単)。プログラムは、スプライトの情報や背景の表示位置を計算し、V-Syncのうちにそれらの値をハードウェアにセットしなければなりません。それに失敗すると表示が乱れてバグとして見えたり、処理落ちになったりします。

シャープX-1、X68000(1987年)は、ビットマップのVRAMもあり、スプライトもあるので、それらの組み合わせでいろんな解決方法を考えられたという、たいへんに贅沢(ぜいたく)な機械で、しかも、CPUもほぼノーウェイトで走行していた、いい機械だったと思います。

ワークステーションの原型

ビットマップディスプレイを備えた、初期のAI用「ワークステーション」の原型であるXerox Alto[注2](1973年)は、606×808ドット(縦長)の高解像度白黒ディスプレイの表示を行うのに、16WordのバッファにCPUがデータを送り込み、表示ハードウェアはそれを1ドットずつシフトしながらブラウン管に表示しました。16Wordのバッファへデータを送るのは、マイクロコードで行いました。必要最小限のハードウェアで高解像度ディスプレイを実現し、AI開発などを行うマシンとしているのが興味深いところです。SD

注2) Xerox Altoハードウェアマニュアル:http://bitsavers.informatik.uni-stuttgart.de/pdf/xerox/alto/Alto_Hardware_Manual_Aug76.pdf

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9回 パスワード管理

to be Continued

パスワード管理って面倒くさいですよね。パスワード管理ツールを利用すると、そのツールを開くパスワード1つ覚えるだけで楽です。サービスごとに別パスワードを使うのにも便利です。ですが、世の中には各サイト別パスワードに対応可能な人が実はいて驚きます。本当に全パスワードを覚えてる人、サービス名から連想できる仮面ライダー名をつけるという人(Amazon利用時のパスワードを予想できそうですよね)。あと頭文字にサービス名「tw」とか「ggl」とか用意してその後は共通の複雑文字列を覚えるってアイデアを聞いた時は膝を叩きましたね。管理ツールを利用する場合はそのデータをどう管理するかでまた悩むんですけどね。ちなみに、この妖怪との戦い話は続きません。

夏休みは海の家で暮らしているノマドワーカーのくつな先生に愛のお手紙を！（物理メール限定）

Software

レッドハット、
Red Hat Enterprise Linux 7を発表

7月10日、レッドハット㈱は「Red Hat Enterprise Linux 7（以下RHEL 7）」を提供することを発表した。

動作環境は、x86_64、IBM POWER、IBM System z。インストールアーキテクチャは64bitのみとなっている（32bitアプリケーション用ライブラリを同梱）。今回、次のような新機能が追加された。

- Linuxコンテナにより、アプリケーションのデプロイを簡素化／迅速化。cGroups、ネームスペース、SELinuxなどの技術を用いてセキュアなマルチテナントを実現
- デフォルトのファイルシステムとしてXFSを採用。500TBまで拡張可能
- Microsoft Active Directoryとの連携を強化。Microsoft WindowsドメインとRHELドメインの混在環境において、セキュアなアクセスを実現できる

そのほか、起動処理をsystemdへ変更、システム管理インターフェースOpenLMIを導入するなどの変更があった。

CONTACT レッドハット㈱
URL http://jp.redhat.com/

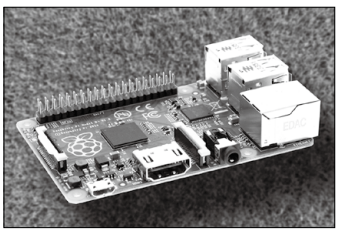
Hardware

アールエスコンポーネンツ、
「Raspberry Pi Model B+」発売

アールエスコンポーネンツ㈱は、名刺サイズのワンボードPC「Raspberry Pi」の新モデル「Raspberry Pi Model B+」を7月16日に発売した。

本製品は、既存商品「Raspberry Pi Model B」と同じ価格（3,940円（税抜き））のまま、使い勝手のよさを追求した新モデル。主な特長は次のとおり。

- リニア電源からスイッチング・レギュレータに変更し、消費電力を約30%削減
- USBポートを4基備え、HUBによるカスケード接続を使用することなく、各デバイスの同時接続が可能
- GPIOのピンを26ピンから40ピンに増加
- 音声の出力品質や側面のコネクタの配置を改善
- ソケットをSDからmicro SDに変更


▲Raspberry Pi Model B+

CONTACT アールエスコンポーネンツ㈱
URL http://rs-components.jp/

Service

Skeed、
自律分散ネットワーク技術の事業化構想を発表

㈱Skeedは昨年 7月6日に逝去した同社創業者兼前CINO（Chief Innovation Officer）の故金子勇の一周忌を迎え、IOT（Internet of Things）市場に対して、同社が培ってきたP2P自律分散ネットワーク技術を核とした新事業に着手することを7月9日に発表した。

同社はIOT事業開発室を6月1日に設置しており、WinnyのP2Pアーキテクチャの「多段転送」および「マルチパス」に加えて「自律分散ストレージ」、「自律分散処理」の技術を基にエッジコンピューティング（PCやスマートフォンなど端末側のみでデータを生成・伝送・処理・保管するシステム）の実現に取り組んでいる。これにより、クラウドなどデータセンタへの一極集中による単一障害ポイントや、データトラフィック増大に伴う輻輳（ふくそう）などの課題を解消する。

具体的な事業としては10月1日から次の3つの製品・サービスの提供を開始する予定だ。

① IOT端末開発者向けモジュール（ライブラリ）
② サービス提供者向けソフトウェアプロダクト
③ P2Pアーキテクチャ活用サービス事業

CONTACT ㈱Skeed
URL http://skeed.jp/

Report

日本アイ・ビー・エム、「BlueMix女子会」開催

7月8日、日本アイ・ビー・エム㈱は、クラウド統合環境「BlueMix」を体験・実習できる女性限定のイベント「BlueMix女子会」を渋谷の21cafe（運営：ギークス㈱）にて開催した。

BlueMixはIBMが提供するPaaS（Platform as a Service）製品。クラウド・アプリケーションの作成、デプロイ、管理を迅速に行える特長がある。

本イベントでは、冒頭にBlueMixの概要・機能についての紹介があり、次いで、BluMixでアプリケーションを開発する実習の時間が設けられた。内容は、Javaアプリ・PHPアプリのデプロイや、Boilerplate（テンプレート）を使ったIOTアプリの開発など実践的なプログラムが多く、講師が参加者一人一人にレクチャーを行った。本イベントの参加者は、半数が女子大生、もう半数が企業で開発に携わる女性社員だった。


▲会場の様子

CONTACT 日本アイ・ビー・エム㈱
URL http://www.ibm.com/jp/ja/

Report

AWS Summit Toyo 2014、開催

7月17、18日に品川のグランドプリンスホテル新高輪で、「AWS Summit Tokyo 2014」が開催された。

AWS Summitは世界各地で開催される、Amazon Web Services（以下AWS）に関する最大規模のカンファレンス。東京での開催は今年で3回目。Amazon.comのCTO,Werner Vogels氏、アマゾンデータサービスジャパン㈱代表取締役社長の長崎忠雄氏による基調講演に加え、2日で合計72にもおよぶセッションが行われた。

グラニセッション～AWS＋C#でWebアプリケーション開発

ソーシャルゲームの開発を行う㈱グラニの取締役CTOの河合宜文氏は、AWSとC#で開発するWebソーシャルゲームに関するセッションを行った。

グラニではAmazon Elastic Compute Cloud（以下EC2）でWindows Server 2012とRedis（key-valueデータストアソフトウェア）を、Amazon Relational Database Service（以下RDS）でMySQLを稼働させ、ソーシャルゲームの開発を行っている。


▲㈱グラニ　取締役CTO
河合宜文氏

セッションの中で同氏は、スケーリングが容易でサポートが豊富なRDS、非同期処理を得意とするC#の利点を強調し、高負荷になりがちなWebソーシャルゲームでそれらが性能を発揮すると語った。また今後の展開として、Amazon RedShiftを使ったアプリケーション分析も模索しているとのことだ。最後には、「言語（C#）も環境（AWS）も常に進化し続ける。最新の状態を維持することは、難しいが、重要なこと」と締めくくった。

NTTデータセッション～AWS＋Hinemosでクラウド運用自動化

㈱NTTデータのセッションでは、2名のスピーカーがインフラの自動化について講演した。錦織真介氏は日本における基盤自動化について、Chefを使った厳密な管理、Gitを使った海外のエンジニアとのチーム開発、本番環境に手を加えない「Immutable Infrastructure」の3点が重要だと語った。

また、長妻賢氏はオープンソースの統合運用管理ツール「Hinemos」について語った。同ツールにより、クラウドにおける構成管理への自動追随、インスタンスの各種制御、課金情報の可視化・監視を容易に行うことができる。また同氏は、HinemosをAWS上でクラスタリングできるオプション「Hinemos HA on AWS」を9月にリリースすることを、本イベントで初めて発表した。


▲㈱NTTデータ　錦織真介氏


▲㈱NTTデータ　長妻賢氏

CONTACT アマゾンデータサービスジャパン㈱
URL http://aws.amazon.com/jp/

Letters from Readers

ITで変わる？　花火大会

夏の風物詩と言えば花火ですが、今の花火の打ち上げはコンピュータで制御されており、秒単位で点火のタイミングを調整できるそうです。花火大会の打ち上げの流れは最初からプログラミングされており、スタートの指示を出すだけで、あとはすべてそのとおりに打ち上げられるとのこと。そのデータがあれば、同じ花火大会を再現したり、PC上でシミュレーションしたりできるのですかねえ。

2014年7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 Nginx移行を考えているあなたへ

Nginxは、従来のApacheに比べて、高速で高負荷にも耐えられると評判のWebサーバです。とはいえ、すべてのWebサーバをApacheからNginxに移行して良いものでしょうか？　本特集では両ソフトウェアの長所短所を整理したうえで、効果的な移行方法を紹介しました。

Apacheから単純に置き換えるのではなく、それぞれの特徴を活かした使い方が重要なんですね。個人的にはNginxの少ないメモリで動作する点に興味があるので、機会があればそのような特徴を発揮するための設定などを具体的に取り上げてくださるとうれしいです。

大阪府／出玉のタマさん

アーキテクチャの比較、移行前のチェックポイント洗い出しの解説が参考になった。

岩手県／隼さん

今までApacheしか触ってこなかったが、必要に応じてNginxにすることも考えたほうがいいかもしれない。

東京都／tomato360さん

細かく比べると「Apacheにはあるが、Nginxにはない」機能もいくつかありました。移行する際は、そういった点を十分に考慮しなければなりませんね。

第2特集 DHCPサーバの教科書

IPアドレスを自動で割り振るサーバがDHCPサーバです。本特集では、あらためてDHCPの役割、動作原理を解説しました。

基本的なところからクラウド時代の今どきの話まであってよかった。

東京都／松本さん

冒頭の「固定IPをExcelで管理」というところに懐かしさを感じました。当時、DHCPを提案したら「管理できなくなるだろ!」と一蹴されたのを思い出します。

神奈川県／ewiad420さん

今や、百近いコンピュータがつながるLANも珍しくありません。そんな環境では、もはやDHCPサーバなしでの運用は考えられませんね。

一般記事　OpenSSLの脆弱性“Heartbleed”の教訓（前編）

世間でも大きな問題となったHeartbleed問題。原因は単純なプログラムミスですが、それがどのように作りこまれたか、その経緯を解説しました。

TVなどでも特集が組まれたりしていたが、本記事はわかりやすくていいと思う。後編が楽しみです。

静岡県／ももんがさん

OpenSSLの脆弱性が多くて困りますね。セミナーとかでも話題に挙がることが多いです。

京都府／クラウドマンさん

十分に枯れた技術と思っていたOpenSSLに脆弱性があったことには、驚きましたが、オープンな議論を経て取り込まれた機能にもかかわらず、単純なミスが誰にも発見されなかったという事実は、さらに衝撃的でした。

短期集中連載　Web標準技術で行うWebアプリのパフォーマンス改善

最終回のテーマは、パフォーマンス計測技術。より実態に近い値を計測できるRUMという技術を取り上げました。

ブラウザで計測するのはいいと思った。

東京都／binaさん

今回のNavigation Timingは利用者目線のQoSを改善するうえで非常にありがたかったです。

愛知県／NGC2068さん

本記事の著者の川田寛氏が作ったhtml5jパフォーマンス部のWebサイト（http://perf.html5j.org/）では、実際にRUMで計測できるようになっています。ぜひ一度、ご覧ください。

複雑化するサーバ環境の監視を変える「OpenTSDB」後編

サーバ監視ツール「OpenTSDB」記事の後編です。効果的に分析するために、どんなデータを、どのように記録するか、どうやってグラフ化するかなど、実践的な内容を取り扱いました。

サーバ環境の複雑化は現代社会の重要課題だと思う。

長崎県／システム・ナルシストさん

OpenTSDBをまったく知らなかったが、使ってみようと思う。

東京都／blackbirdさん

今回は、ツールの使い方だけではなく、測定するデータの意味や特徴についても言及していました。OpenTSDBに限らず、それらをきちんと理解しておかないと、効果的な監視はできませんね。

短期集中連載　Rettyのサービス拡大を支えた“たたき上げDevOps”

最終回は、今流行のImmutable Infrastructureについて、Rettyではどのような形で実現しているのかを紹介しました。

文体が読みやすくて好感が持てました。記事中の「ここ最近のインフラ周りの進化には目を見張るものがあってとてもテンションが上がります」に同感です。

東京都／山下さん

1、2回目を読み飛ばしていたが、現在の業務改善に役立ちそうなヒントがあったので、連載全体を読み返そうと思った。短期連載で取り上げるのに、うってつけの内容だったと思う。

千葉県／若山さん

この回の記事タイトルは、「やっぱり楽しい！　トレンドに乗ったインフラ改善」でした。これだけで、著者の前向きな姿勢が伝わってきますね。ともすると、「システムのお守り」というふうにマイナス思考に陥りがちなインフラ業務ですが、やるからには、このように積極的に取り組みたいものですね。

エンジニアの能率を高める一品

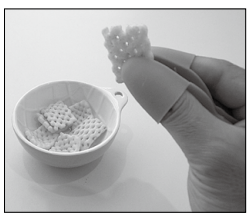
仕事の能率を高める道具は、ソフトウェアやスマートフォンのようなデジタルなものだけではありません。
このコーナーではエンジニアの能率アップ心をくすぐるアナログな文具、雑貨、小物を紹介していきます。

ゆびトング付き菓子ボウル

380円（税別）／ゲットクラブ　http://www.get-club.net/

◀写真1　ゆびトング

ずっとPCに向かって仕事をしていると、ジャンクなお菓子の1つもつまみたくなります。しかし、ポテトチップスのような脂っこいものや、チョコレートのような溶けるお菓子は手が汚れます。汚れた手でキーボードはさわれないので、タイピングの手を止めずにちょっとお菓子をつまむということができません。本製品はそんな悩みを解決します。ゆびサックのようなトングを指にはめてお菓子をつまめば、指が汚れません（**写真1、2**）。片手だけで着脱できるので、キータイピングの合間に、サッとトングをはめて、ひとつまみすることも可能です。「トングだけあれば、ボウルはいらないのでは?」と言われそうですが、お菓子が袋に入ったままだと、袋に手を突っ込んだときに、手のほかの部分が汚れたり、上手くつまめなかったりするので、最初からお菓子をボウルなどに出したほうが食べやすいですよ。ただ、本製品のボウルではやや小さいかも。



▲写真2　ゆびトング装着

祝　7月号のプレゼント当選者は、次の皆さまです

①iPad Air対応 QODE Thin Typeキーボード ..... 東京都　松本直樹様
②mouse fit ..... 宮城県　伊勢雅博様

★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
October 2014

2014年10月号
定価（本体1,220円+税）
176ページ

9月18日発売

［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年に最初のバージョンが公開されて以来、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もリリースされました。Javaユーザのみんながみんな、追加されてきた新機能を使いこなせているのでしょうか? 初心者だからって、くり返し、条件文、配列など、基本的な文法だけでコーディングしていませんか? Java経験が長いからって、昔ながらの言語仕様のままで、ずっとコーディングしてきていませんか? 今のJavaなら、もっと効率的で品質の高い書き方ができます!

［第2特集］進化したアーキテクチャを一瞥で分析

サーバの目利きになる方法［前編］

オンプレミスを制するものはクラウドを制する

絶え間なく進化するサーバマシンの「目利き」になる方法を前後編に分けて解説。10G/100Gbps高速通信ネットワーク対応、サーバサイドフラッシュ、大容量HDD、マルチコアCPUなどなど、最新サーバマシンを支える技術をわかりやすく紹介!

■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

※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。

SD Staff Room

●『システム管理者の眠れない夜』の著者である柳原秀基さんが7月4日に急逝されました。この場を借りて謹んでお悔やみ申し上げます。ちょうど前号の入稿と重なり大阪での葬儀に行けませんでしたが、FBなどのTLを見て彼のことを悼みました。きっと天国でも泡盛片手に技術論を語っているんだろうな!（本）

●腐敗した鶏肉や牛肉などを原料とした加工食品が流通しているようだ。しかし、下水道から油を取り出したり、腐ったりカビが生えた原料から食べてもわからないレベルのものを作れる技術ってすごいなと思う。そういえばどっかで聞いたことがある。「ただちに健康に影響はない……」（自家水耕栽培お勧め幕）

●娘のバレエ発表会がありました。今回は初めてトウシューズを履いての舞台。なぜつま先立ちをしたまま踊れるのか不思議で仕方ありません。それはそうと、このバレエ発表会での楽しみは毎回ゲストで呼ばれる一人の男性ダンサー。ダンスはもちろん、立ち居振る舞いでの演技力にいつも惚れ惚れします。（キ）

●夏になると不思議に思うのは、アスファルトだらけのオフィス街でも、多くのセミが鳴いていること。彼らはどこから来るのでしょう? 街路樹のところにある、わずかな土の中から這い出てくるのでしょうか? だとすると、街路樹の根元には、相当数のセミの幼虫がギッシリつまっていることに……!（よし）

●都電荒川線に乗ってみました。東京に来てから初めてその存在を知り、驚いたのを覚えています。休日の夕方に乗ったのであまり混んでおらず、情緒ある雰囲気も相まってのんびりできました。路地の間ギリギリを通る区間もあり、おもしろかったです。ただ、けっこう揺れるんですよね! 酔いました。（な）

●年に数回、風邪をこじらせて通院するのですが、今月だけですでに二回。冷房が苦手で、実家にいるときはほとんど使わなかったのですが、さすがにそうもいかず、設定温度高めでお互い妥協したのですが、寝ているときは防寒対策も効果なし。夏の間は家庭内別居が平和に落ち着くのかな（笑）（自然派）

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9月号

発行日
2014年9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㈱

特別付録

3分間ネットワーク基礎講座

網野衛二 著
EIJI AMINO

特別編

まるごとルーティング基礎マスター

某所の某大学にて、情報処理を教える博士。専門はネットワーク。たった 1 人しかいないゼミ生であるネット君をこき使う。

インター博士のただ 1 人のゼミ生。ネットワークについてはまったくの素人。

◎ルーティングの基礎の基礎を押さえる！
◎本格的なルーティング技術も併せて紹介！
◎ネットワークの理解が進む珠玉の 48 ページ

技術評論社

第1回

アドレスと経路

● IP アドレスと MAC アドレス

さて、レイヤー 3 の話を『改訂新版ネットワーク基礎講座』の第 3 章からしているわけだ。レイヤー 3 は「インターネットワーク」を実現する役割、を担っていたわけだ。

「ネットワーク間でのデータの転送」がインターネットワーク、でしたよね。レイヤー 1 がケーブルで信号を運ぶ、レイヤー 2 がケーブルでつながっている範囲としてのネットワーク内でのデータ転送。そしてレイヤー 3 がネットワーク間、ですね。

そういうことだ。各レイヤーが、それぞれの範囲でのデータのやり取りのしくみを決めているわけだな。レイヤー 3 は「アドレッシング」と「ルーティング」でインターネットワークを行う。そのうち、前掲書の第 3 章ではアドレッシングを説明したわけだ。

アドレッシング、IP アドレスですね！！　「どこのネットワークにある」「どのコンピューター」という情報が IP アドレスでしたよね。IP アドレスで場所を特定して、「ルーティング」で道筋を決定するんでしたっけ。

そういうことだ。IP アドレスはあて先を特定する役割だった。で、ネット君。この「あて先」というのはなにかね？

え？　データのあて先のコンピューターでしょ？

そうだな。「論理アドレス」である IP アドレスはデータのあて先のコンピューターを指定する。じゃあ、MAC アドレスは？

MAC アドレス……、イーサネットや無線 LAN で使用されている、LAN インターフェースに付属の「物理アドレス」ですよね。MAC アドレスは、あて先のコンピューターを示して……、あれ？

IP アドレスも MAC アドレスも、どちらもあて先を特定する。ではどこが異なるかというと、**MAC アドレスは同じネットワーク内でのあて先**を特定する。それに対して**IP アドレスはあて先のコンピューター**を特定する。つまり、IP アドレスが最終的なデータの届け先、MAC アドレスが次の届け先、ということになる。

MAC アドレスが次の届け先？　IP アドレスが最終的なアドレス？　どういうことですか？

インターネットワークでは、データは複数のネットワークを経由して、最終的なあて先のコンピューターまで届く。経由していくのはいいのだが、「誰に経由してもらう」かがわからないと困るだろう？　例えば、とあるネットワーク A から、ネットワーク B を経由してネットワーク C に行くとする。A から出たデータグラムは B に入ってその次に C へ行くわけだが、B に入った時点で「C へ中継してくれる誰か」のところへ行かなければいけない。それを指定するのが MAC アドレス、「次に送る場所」の決定というわけだ。**(図 1-1)**

ん〜、MAC アドレスで「次に送る場所」を指定する。そこに届いたら、また MAC アドレスで「次に送る場所」を指定する。それを繰り返して、最終的なあて先へ届く、ということですか。……ということは、あて先となる MAC アドレスはころころ変わるんですね？

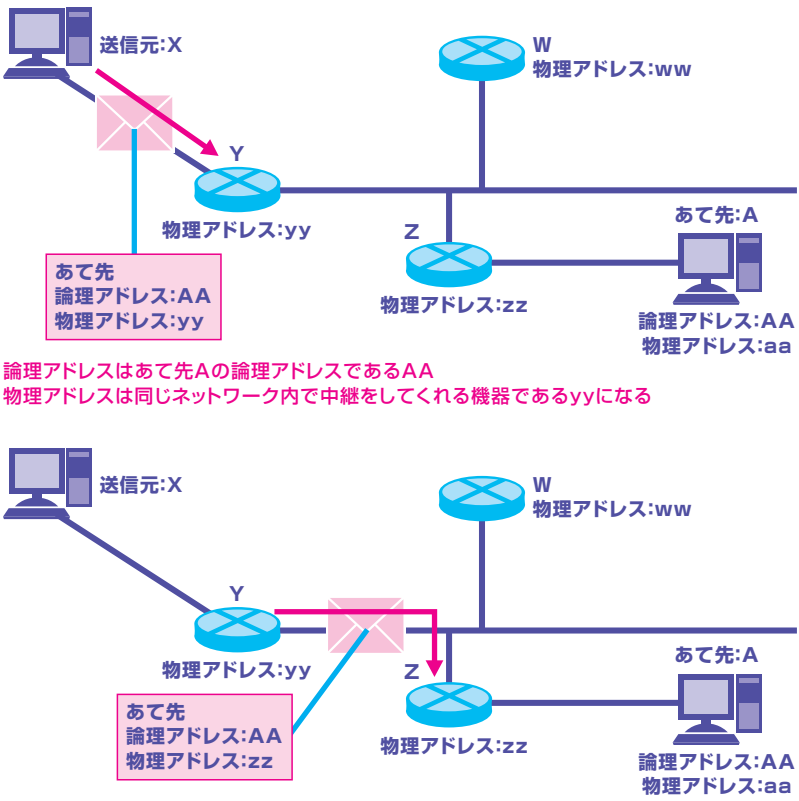
そういうことになる。MAC アドレスは「次のあて先」だから、そのコンピューターに届いたら、さらに次を指定する必要がある。よって、MAC アドレスは変更される。一方で IP アドレスは変更されない。だから、IP アドレスが最終的なデータの届け先、MAC アドレスが次の届け先、というわけだ。

●経路

さて、MAC アドレスにより「次のあて先」を、IP アドレスで「最終的なあて先」を指定するわけだ。それにより、**あて先までの経路**ができる。

経路……、つまりアレですね、まず送信元から次のあて先が MAC アドレスで指定されて、そこへ届いたら次のあて先がまた MAC アドレスで指定されて……、を繰り返して、最終的に IP アドレスで指定される本来のあて先に届く、と。その動きをつなぐと、確かにあて先までの「道」ができますね。

図 1-1　IP アドレスと MAC アドレス



そうだ、それが「あて先への経路」となる。そして、この**経路を決定する役割を持つ機器がルーター**だ。このルーターが行う経路の決定だが、実は「経路を決定」しているわけではない。

え？　じゃあなにを決定してるんですか？　ルーティングするのがルーターで、ルーティングであて先までどうやって行くか、つまり経路を決定しているんじゃないんですか？

まぁ、確かにその通り。ただ、基本的にルーターが決定しているのは「経路の一部」にすぎない。つまり、自分の場所からあて先まで行くために、**次にどこへ送ればよいか**を決定しているだけなのだよ。「送信元からあて先」までのすべての道筋を理解しているわけではない、ということだ。

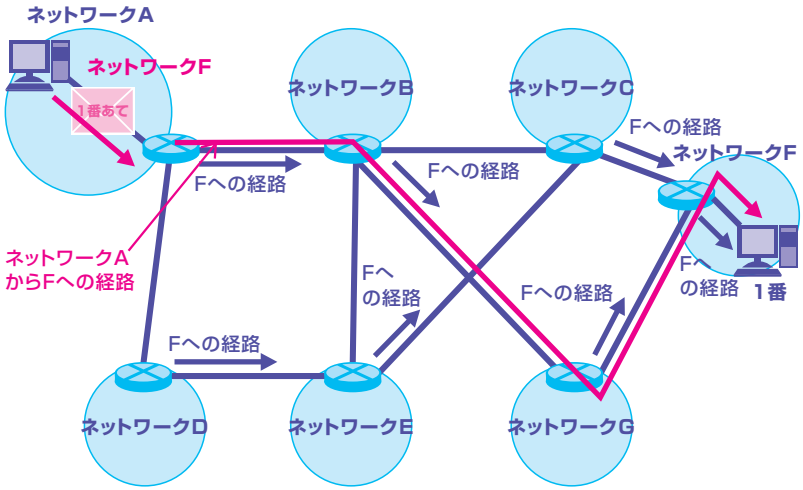
ふむふむ、次の場所は知っているけど、その次からどうやって行くか、までは理解していないということですかね。なんていうか、「方向指示機」みたいな感じですね。あそこへ行きたいならこっちへ行け、でそこまでついたらまた指示をあおげ、みたいに？

うむ、その考え方は間違っていない。このように、順繰りに「次への道」を指示していく方式を**ホップバイホップ**［Hop-by-Hop］と呼ぶ。ホップとは、ルーティングではルーターを指す言葉だ。**(図 1-2)**
このように「次の」あて先を示すことを、複数のルーターが繰り返すことによって、全体の「経路」ができていくわけだ。そしてルーターは**ネットワークの境界上に配置**され、受け取ったデータグラムをルーティングし、次のあて先を決定する。

う〜ん、その次のあて先ってのは具体的になんですか？

図 1-2　ホップバイホップ

ルーターは次のあて先を決め、それがつながって経路になる



基本的には**あて先へ行くための次のルーター**だ。ただし、あて先がそのルーターに接したネットワーク内にある場合は、あて先のコンピューターそのものが次のあて先になる。
ここで重要なのは、**ルーターがなければ別のネットワークへデータグラムを送ることはできない**ということだ。これは絶対のルールだ。ルーターがルーティングすることにより、別のネットワークへの「経路」ができる。この「経路」ができなければ別のネットワークへは届かない、ということだ。

絶対ですか？　例えば、直接つながっていたとしても？　ネットワークって、コンピューターのグループですよね。同じマルチアクセスネットワークに、違うネットワークのコンピューターがつながっていたらどうなるんですか？

それでもダメだ。例えば、同じハブに 2 台のコンピューターがつながっていて、その 2 台が別のネットワークに所属するように設定されていたとする。この状態の場合でも、別のネットワークのコンピューター、つまり同じハブにつながっているもう 1 台にはデータグラムは届かない。

え？だって、データグラムを送ってしまえば、ハブは受信したインターフェース以外のすべてのインターフェースから送信する「フラッディング」をするんですよね。であるなら、別のネットワークだろうがなんだろうが、信号は送られるわけですから、あて先のコンピューターには届きませんか？

いや、届かない。なぜならば、コンピューターは「あて先が別のネットワークにある場合はルーターへ送信する」「あて先が同じネットワークにある場合は直接あて先へ送信」する、というルールで動いている。よって、ルーターがないと別ネットワークへは送れない。**（図 1-3）**

へー、もしコンピューターに「ルーターが設定されていない」ならば、別のネットワークへのデータの送信自体が行われないんですね。

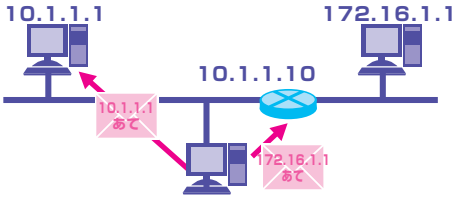
そういうことだ。このとき、コンピューターが指定するルーターのことを**デフォルトゲートウェイ**［Default Gateway］と呼ぶ。これについては先で説明する（P.13 参照）。とりあえず、今回はここまでとしておこう。

はい。3 分間ネットワーク基礎講座でした～♪

図 1-3　コンピューターが送信するルール

別ネットワークあてならばルーター（デフォルトゲートウェイ）へ、同一ネットワークあてなら直接通信する

①ルーター（デフォルトゲートウェイ）が設定されている

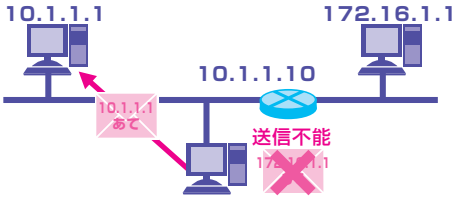


10.1.1.2
ルーター（デフォルトゲートウェイ）：10.1.1.10

・同一ネットワークあてならば直接あて先へ送信する

・別ネットワークあてならばデフォルトゲートウェイに設定されたルーターへ送信する

②ルーター（デフォルトゲートウェイ）が設定されていない



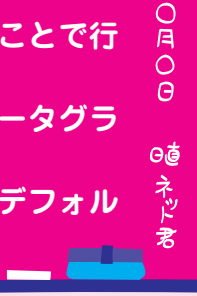
10.1.1.2
ルーター（デフォルトゲートウェイ）：———

・同一ネットワークあてならば直接あて先へ送信する

・別ネットワークあてならば送信不能とする

ネット君の今日のポイント

- IP アドレスは「最終的なあて先」、MAC アドレスは「次のあて先」を決定する。
- ルーティングは次のあて先を指定していくことで行うホップバイホップである。
- ルーターがなければ別のネットワークへデータグラムを送ることはできない。
- コンピューターに設定しているルーターはデフォルトゲートウェイと呼ぶ。



第2回

ルーター

●ルーターとは

さて、ルーターがなければ別のネットワークへデータグラムを送信することはできない。つまり、インターネットワークを実現する機器がルーターなのだということだ。

インターネットワークは、「ネットワーク間でのデータの転送」でしたよね（『改訂新版ネットワーク基礎講座』の第3章 P.123）。ルーターがなければ別ネットワークへデータグラムを送信できない。つまり、ルーターがなければネットワーク間でのやり取りができない、ですね。

そういうことだ。ルーターはルーティングを行い、別のネットワークへのデータ転送を行う。つまり**ルーターこそがインターネットワークの最重要機器**ということになる。それで、だ。ルータールーターと何回か言葉が出てきたが、実際どのような機器か説明していない。

え〜っと、ルーティングする、ネットワークの境界上に配置される？……、えっと他になにかありましたっけ？

まず、ネットワークの境界上に配置される、から説明しようか。ルーターはとあるネットワークから別のネットワークへデータグラムを送り出すという役割上、「ネットワークとネットワークの境界上」に配置される。つまり、**複数のインターフェースを持つ**。

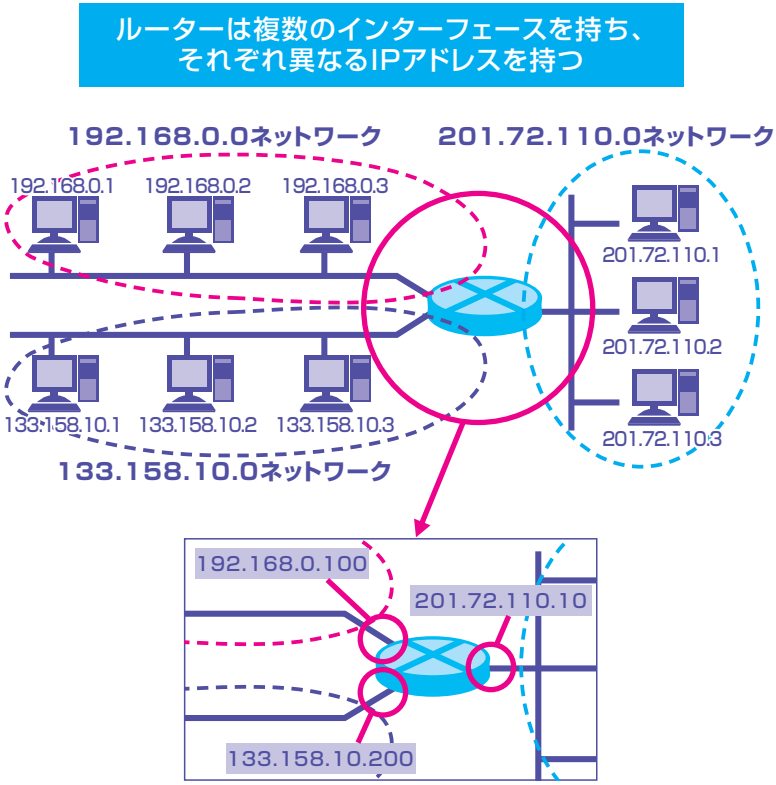
複数のインターフェースを持っていて、そこから複数のネットワークにつながっているってことですね。それが、ネットワークの境界上にあるということの意味ですか。

そういうことだな。ルーターのインターフェースには論理アドレス、つまりIPアドレスが設定されている。つまり、ルーターの各インターフェースは**それぞれのネットワークに所属している**形になるわけだ。(図2-1)

その結果、ルーターは単独ではなく、複数のネットワークに所属するわけですね。

そして、ルーティングする。ルーティングとは簡単に言えば、**データグラムのあて先IPアドレスを元に、次に送信するルーターを決定する**ことだ。これを行うことで「経路」が決定される、というわけだ。

図2-1　ルーターの接続



ふむふむ、経路を決定する、と。ルーターは「ネットワークの境界上に配置」されて「ルーティング」する。他の役割はないんですか？

そうだな。まず、ルーターはネットワークの境界線上にあるため、**複数のネットワーク同士をつなぐ**役割を持っている。この「複数のネットワーク」は、LAN で使われているイーサネットの場合もあるし、WAN で使われている回線の場合もある。これら違う種類のネットワークを「つなぐ」のもルーターの役割だ。

こっちはイーサネット、こっちは WAN の回線。違う種類のネットワークの間にいて、データの「中継」をするってことですね。

そういうことだ。あとは、流れてきたデータグラムに対し、条件をつけてそのデータグラムを破棄してしまう**フィルタリング**［Filtering］という処理を行ったりもする。例えば、大学のコンピューター実習室のネットワークからは、学生のデータが入っている事務室のネットワークへはデータを流さない、とかだな。

なるほど、確かに学生のみんなが使うコンピューター実習室から、事務室へデータがやり取りできちゃうと悪いことを考えちゃいそうです。僕の成績証明書を……。

もちろん、ウチの学校はそんなことはできないぞ。では、まとめてみると、だ。ルーターは「ネットワークの境界上にあり」「複数のネットワーク同士をつなぎ」？

「ルーティングにより次のルーターを指定して経路を作る」？

さらに「フィルタリングによりルーティングするデータを仕分けできる」、ということだ。

●ルーターの動作

では実際のルーターの動作を説明しておこう。まず、ルーターは**ルーティングテーブル**というものを持っている。

るーてぃんぐてーぶる？

うむ。**最適な経路の地図**だと思ってもらうとわかりやすい。つまり、ルーターが受け取ったパケットのあて先までの最適な経路が載っている地図だ。

最適な経路が載っている地図。それを見て、ルーターはあて先ネットワークまでの経路を決定するんですか？

そうだ。この地図には、**あて先ネットワークまでの距離、次に届けるルーター、そのルーターにつながっている自分のインターフェース**などが載っている。ルーターはこのテーブルに従って、受け取ったパケットをあて先まで送る。つまり、**ルーティングテーブルこそがルーターの要**なのだよ。**(図2-2)**

ふむふむ。ルーティングテーブルから、あて先のネットワークを見つけて、次に届けるルーターを決定して、インターフェースから送信するってことですね。

そういうことだ。ではポイントを説明しよう。まず、前にも説明したようにルーターが決定するのは「次のあて先」だ(P.4 参照)。ルーティングテーブルにも「次のあて先になるルーター」が記載されている。

でしたよね。ホップバイホップで、次のルーター、次のルーターって順番に届いていくわけですよね。

そうだ。ルーティングテーブルから次のあて先を探し出すわけだが、どうやって探しているかというと、これは**最長一致ルール**と呼ばれるルールで決められている。英語で言うと、ロンゲストマッチ［Longest Match］という。

ろんげすとまっち？　最長一致って、なにが最長一致なんですか？

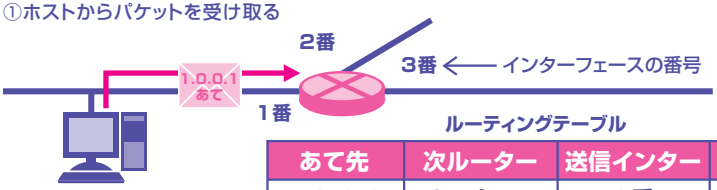
実際のデータグラムのあて先 IP アドレスから、ルーティングテーブルのあて先ネットワークアドレスを決定して、次のルーターや送信インターフェースを決定する。そこで、「あて先 IP アドレス」と「あて先ネットワークアドレス」を比較するときのルールが最長一致のルールだ。**(図2-3)**

IP アドレスのビット列と、ネットワークアドレスのビット列を先頭から順番に比較していって、一番多く一致するものから選ぶんですね。だから、最長一致？

図 2-2　ルーターの動作

ルーティングテーブルから次のあて先となるルーター、
送信するインターフェースを決定する

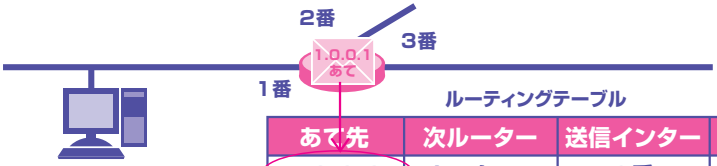
①ホストからパケットを受け取る



ルーティングテーブル

あて先	次ルーター	送信インター	距離
1.0.0.0	ルーターX	3番	5
2.0.0.0	ルーターY	2番	3

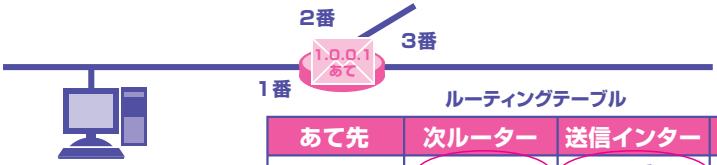
②あて先IPアドレスからあて先ネットワークを決定する



ルーティングテーブル

あて先	次ルーター	送信インター	距離
1.0.0.0	ルーターX	3番	5
2.0.0.0	ルーターY	2番	3

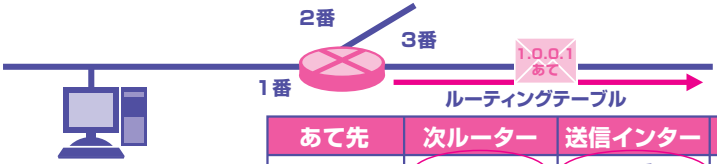
③ルーティングテーブルから、次に中継するルーター、送信するインターフェースが決定される



ルーティングテーブル

あて先	次ルーター	送信インター	距離
1.0.0.0	ルーターX	3番	5
2.0.0.0	ルーターY	2番	3

④決定したインターフェースからパケットを送信する

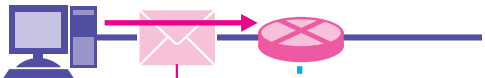


ルーティングテーブル

あて先	次ルーター	送信インター	距離
1.0.0.0	ルーターX	3番	5
2.0.0.0	ルーターY	2番	3

図2-3　最長一致のルール

あて先IPアドレスともっともビットが一致するものを選ぶ



あて先IPアドレス
192.168.12.5

ルーティングテーブル

IPアドレス/プレフィックス長	次のルーター	送信インター	メトリック
192.168.0.0/16	172.18.5.2	0番	2
192.168.12.0/24	172.16.10.2	1番	3
192.168.10.0/24	172.17.22.2	2番	8

プレフィックス長まで一致したなかでもっともプレフィックス長が長いこの経路を使用する

あて先IPアドレスとルーティングテーブルのエントリの比較

あて先 192.168.12.5	1100 0000	1010 1000	0000 1010	0000 0101	
192.168.0.0/16	1100 0000	1010 1000	0000 0000	0000 0000	プレフィックス長の16ビットまで一致
192.168.12.0/24	1100 0000	1010 1000	0000 1010	0000 0000	プレフィックス長の24ビットまで一致
192.168.10.0/24	1100 0000	1010 1000	0000 1000	0000 0000	プレフィックス長の24ビットまで一致せず

そういうことだ。ルーターは非常に重要なので、しばらくルーターの話を続ける。ではまた次回としよう。

了解。3分間ネットワーク基礎講座でした～♪

ネット君の今日のポイント

- ルーターが経路選択を行う。
- ルーターのインターフェースは、IPアドレスを持つ。
- ルーターは経路を選択するためにルーティングテーブルを持つ。

○月○日　ネット君

第3回

デフォルトゲートウェイ

●ブロードキャストドメイン

前回はルーターの話だったな。ルーターはネットワークの境界上に配置され、ルーティングを行う。それにより経路が設定され、あて先までデータグラムが届くようになる。

ルーティングテーブルを持っていて、次に届けるルーター、送信するインターフェースを決定するんでしたね。

うむうむ。さて今回は、ルーターの役割、というか機能の話をしよう。**ルーターを越えてブロードキャストは流れない**という話を以前したな。

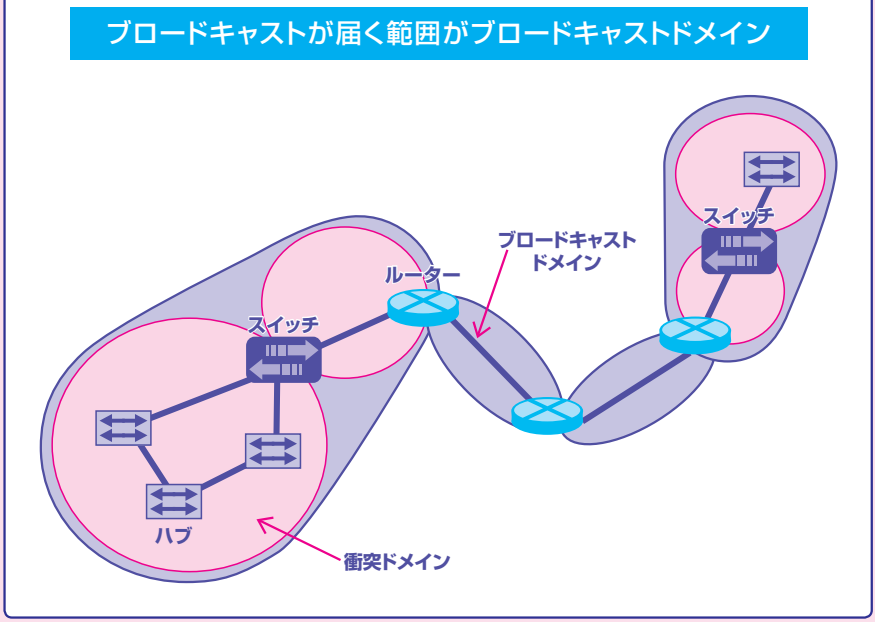
え〜っと、ネットワークを分断することで、ブロードキャストが流れる量が減るって話をしましたよね（『改訂新版ネットワーク基礎講座』の第3章 P.125）。

そうだ。つまり**ルーターがネットワークを分断することで、ブロードキャストが他のネットワークに流れない**ようにしている。このブロードキャストが届く範囲のことを、**ブロードキャストドメイン**［Broadcast Domain］というが、ルーターはブロードキャストドメインを分けることができるのだよ。

ぶろーどきゃすとどめいん？　え〜っと、前に出てきた、ほら、信号を送ると衝突が起きるかもしれない範囲のことを指す、えっと、そう、衝突ドメインってのに似てますね。

うむ。考え方は同じだ。衝突の影響が及ぶ範囲が、衝突ドメイン。ブロードキャストが及ぶ範囲が、ブロードキャストドメイン。**衝突ドメインはスイッチが区分けし、ブロードキャストドメインはルーターが区分けする。**（図3-1）

図 3-1　ブロードキャストドメイン



ハブはどちらにも影響を及ぼさないですね。

そういうことだ。ルーターはブロードキャストドメインを分割する、つまりルーターがブロードキャストドメインの境界になる、ということになるわけだ。よって？

よって？　……あれ？　ルーターはネットワークの境界上にもあるんですよね。ブロードキャストドメインとネットワークってどう違うんですか？

うむ、基本的に違いはない。ブロードキャストドメイン＝ネットワーク、と考えて問題はないのだよ。

● ARP とルーター

ルーターは、ブロードキャストを他に転送しない。このことで考えなければいけないことがある。ネット君、あて先の MAC アドレスを知る方法はなんだった？

え〜っと。ARP です。

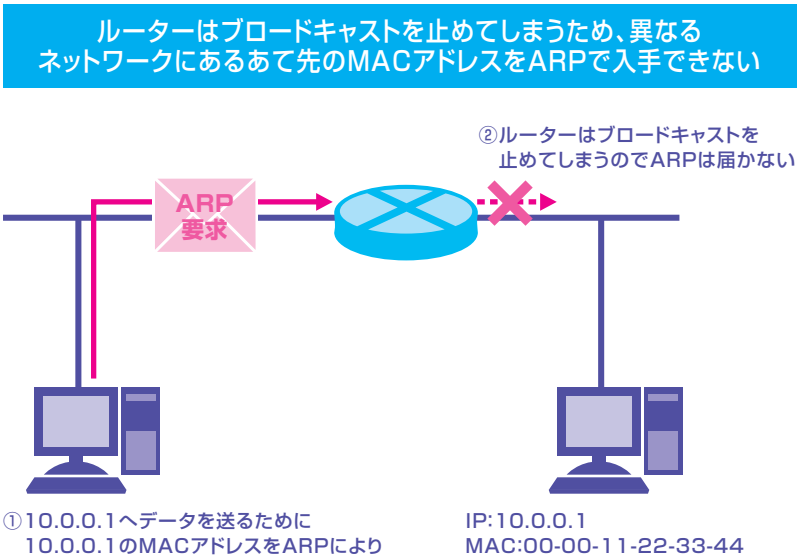
そう、あて先の IP アドレスからあて先の MAC アドレスを調べるのには ARP（Address Resolution Protoco）を使う。この ARP であて先 MAC アドレスを調べるための「ARP 要求」がブロードキャストを使用するというのが問題だ。

ARP はブロードキャスト。ということは、ルーターは ARP を他のネットワークに流さない。ということは、他のネットワークにあるコンピューターの MAC アドレスはどうやって知るんですか？

ルーターがブロードキャストを他のネットワークに流さないので、**ARP はあて先まで届かない**。あて先まで届かないと、もちろんあて先 MAC アドレスがわからないことになる。**（図 3-2）**

そうなると。4 つのアドレスがそろわないから、データ送信ができないってことになりますよね。ネットワークを接続するのがルーターの役割なのに、ルーターが ARP を止めてしまうせいでネットワーク間のデータ転送ができなくなってしまうんですね。

図 3-2　ARP とルーター



そういうことだ。そこで、デフォルトゲートウェイという言葉を思い出してもらおう。デフォルトゲートウェイは、コンピューターが次に送るルーターだったな（P.5 参照）。ルーターはネットワークの境界上にあり、他のネットワークへのデータグラムをルーティングする。つまり、**デフォルトゲートウェイがネットワークの出入り口**となる。

コンピューターは別のネットワークにデータグラムを送信する場合、必ずデフォルトゲートウェイに送信する、でしたよね。そう考えれば、「ネットワークの出入り口」とも言えますね。

●デフォルトゲートウェイ

他のネットワークへデータ転送を希望するホストは、一度**デフォルトゲートウェイにデータを送り、他ネットワークへ転送してもらう**。つまり、**コンピューターが最初にデータを送るあて先はデフォルトゲートウェイ**ということになる。

最初にデータを送るあて先がデフォルトゲートウェイ……。ということは？

ということは、だ。IP アドレスと MAC アドレスはあて先の意味にどんな違いがあるのだったかね、ネット君？

IP アドレスは「最終的なあて先」を、MAC アドレスは「次のあて先」を指定するんでしたよね。ということは、コンピューターが別のネットワークへデータを送信する場合のあて先は、必ずデフォルトゲートウェイの MAC アドレスになる？

そういうことだ。つまり、**コンピューターは別のネットワークへデータを送信したい場合、デフォルトゲートウェイに対して ARP を行う**。**(図 3-3)**

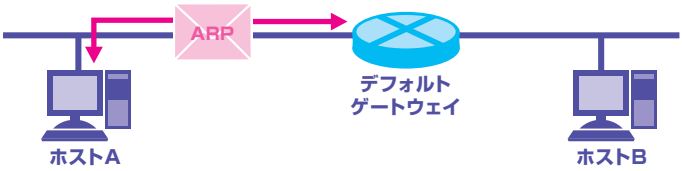
ははぁ。そうすれば ARP により、デフォルトゲートウェイの MAC アドレスが入手できるわけですね。えっと、まとめるとコンピューターはあて先を決定すると、あて先が同じネットワークかどうかを調べて？

同じネットワークならば、あて先 IP アドレスあてに ARP を行い、そのコンピューターの MAC アドレスを入手する。そうでない、つまり別ネットワークがあて先ならば？

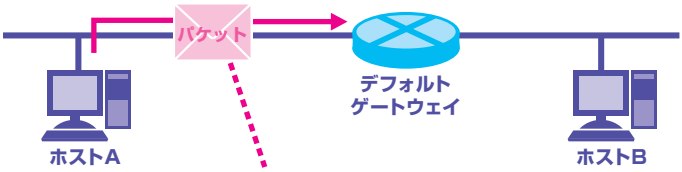
図3-3　デフォルトゲートウェイ

コンピューターは別ネットワークがあて先の場合はデフォルトゲートウェイにARPを行う

①他のネットワークへデータを送りたい場合、ホストはデフォルトゲートウェイにARPを行い、デフォルトゲートウェイのMACアドレスを入手する



②ホストはあて先MACアドレスをデフォルトゲートウェイに、　あて先IPアドレスをあて先ホストにしてパケットを送る



あて先MAC	送信MAC	送信元IP	あて先IP	ペイロード
デフォルトゲートウェイ	ホストA	ホストA	ホストB	

③受け取ったデフォルトゲートウェイはルーティングを行い、中継ルーター、送信ポートを決定し、次に受け取る相手（中継ルーターまたはあて先）にARPを行う

④ARPにより入手したMACアドレスをあて先MACアドレスに、自分自身のMACアドレスを送信元MACアドレスに書き換えて送信する。IPアドレスは変更されない

あて先MAC	送信MAC	送信元IP	あて先IP	ペイロード
ホストB	デフォルトゲートウェイ	ホストA	ホストB	

デフォルトゲートウェイに ARP して、デフォルトゲートウェイの MAC アドレスを入手するわけですね。ということはデフォルトゲートウェイの IP アドレスを知らないといけませんね？

うむ。なので**コンピューターにはデフォルトゲートウェイの IP アドレスをあらかじめ設定しておく**。その方法としては、手動で設定するか、DHCP により設定するかのどちらかだ。

そういえば、DHCP は「IP アドレスの配布」を行う際に、サブネットマスクとかも一緒に配布できるという話でしたよね。この時にデフォルトゲートウェイの IP アドレスも配布することができるんですね。

そうだ。DHCP では IP アドレス以外に、サブネットマスクやデフォルトゲートウェイ、DNS サーバーのアドレスを配布する事が可能になっている。そして、デフォルトゲートウェイが設定されていなかった場合、コンピューターは別ネットワークにデータを送信できないからな。これは前も話したな。

でした。

さて、今回はここまでとしよう。次回もルーターの話をする。

了解。3 分間ネットワーク基礎講座でした〜♪

ネット君の今日のポイント

- **ルーターはブロードキャストを他のネットワークに流さない。**
- **ブロードキャストが届く範囲をブロードキャストドメインという。**
- **コンピューターは、異なるネットワークへデータを転送したい場合、デフォルトゲートウェイに送る。**
- **そのときはデフォルトゲートウェイに ARP を送信し、デフォルトゲートウェイの MAC アドレスをあて先 MAC アドレスにする。**

○月○日 ネット君

第4回

ルーティング

●ルーティングテーブル

さて、ネット君。ルーターはルーティングを行う機器で、コンピューターのデフォルトゲートウェイになる。そして、ルーターがルーティングを行うためには**ルーティングテーブル**を持つ、という話をしたな。

ルーティングテーブルから、あて先のネットワークを調べて、次のルーターを決定するんでしたよね。

そうだ。そのルーティングテーブルだが、簡単に言うと**あて先ネットワーク**と、**中継地点**と、**メトリック**と、**あて先への出口**が載っている表だ。メトリックについてはあとで説明する。**(図4-1)**
以前も話した通り、ルーターは**あて先ネットワーク**を決定する（P9参照）。いちいち何番ネットワークの何番コンピューターというところまでは考えない。ルーターは**あて先ネットワークアドレスとルーティングテーブルを比較して、経路を探し出す。**

最長一致のルール、でしたよね。そういえば、ルーティングテーブルにあて先ネットワークがない場合、どうなるんです？

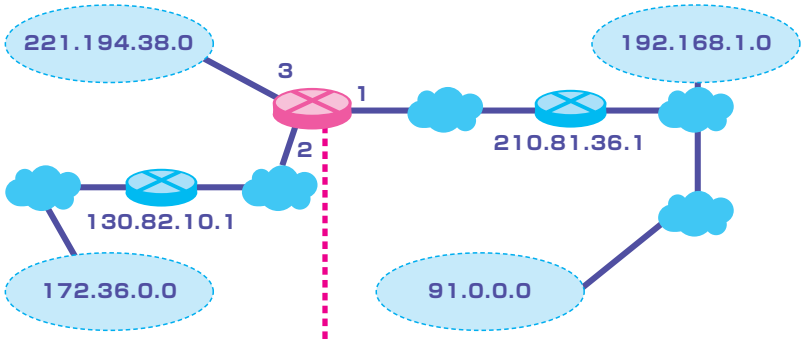
その場合、**あて先不明としてデータグラムを破棄する。**スイッチは、あて先がわからない場合フラッディングするが、ルーターは破棄してしまう。

●２つのルーティング

ではネット君、質問だ。ルーターは**あて先のネットワークへの最適な経路をどうやって見つけるのか？**　そもそもルーティングテーブルにはあて先への経路が載っているわけだが、どうやって知ったのだ？

図4-1　ルーティングテーブル

あて先のネットワーク、次のルーター、メトリック、送信インターフェースが書かれている



あて先ネットワーク	次のルーター	メトリック	インターフェース
192.168.1.0	210.81.36.1	3	1番
91.0.0.0	210.81.36.1	6	1番
172.36.0.0	130.82.10.1	2	2番
221.194.38.0	なし	0	3番

え～～～～っと。

ま、ネット君の考えを待っている間に日が暮れてしまうので、先へ行こう。ルーターは**最適な経路を見つけるために、他のネットワークへの経路をすべて知る必要がある**。

比べてみないとどこが最適な経路かわからないわけですから、そうなりますよね。

そして、**知った経路の中から、最適なものを選んでルーティングテーブルを作成する**。どうやって他のネットワークへのすべての経路を知るかというと、方法は2種類ある。**静的ルーティング**と**動的ルーティング**だ。

静的と動的ですか。IP アドレスの設定にも出てきた言葉ですよね。静的な設定は「手動で入力する」、動的な設定は「自動で設定される」でしたっけ。

うむ、そうだな。「静的」が手動、「動的」が自動の意味だ。まず、**静的ルーティングは管理者が手動でルートを入力する**。「このネットワークへは、この経路を使いなさい」とな。

ルーターにですか？

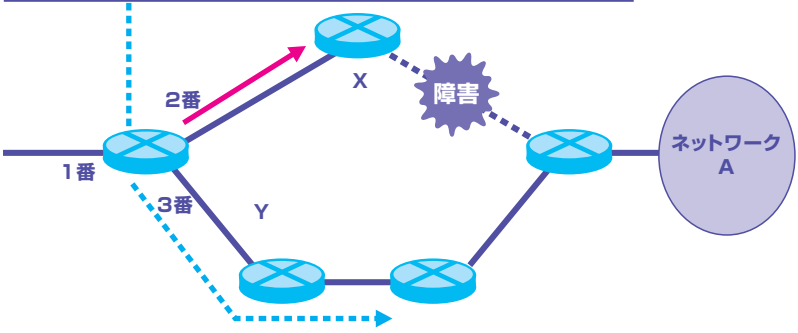
もちろん、ルーターにだ。だが、静的ルーティングは大きな欠点を持っている。迂回路の問題だ。**手動で入力した経路が使えなくなってしまう**ことが起こりえる。(図 4-2)

なるほど。でもこの場合なら、下の経路を使えばいいじゃないですか。

図 4-2　静的ルーティングと迂回路

静的ルーティングでは迂回路への切り替えは
手動で行う必要がある

あて先	次ルーター	送信インター	メトリック
ネットワークA	ルーターX	2番	1



障害のため、ルーターX経由はネットワークAまで届かないが、静的に書かれたテーブルがルーターX経由になっているため、ルーターXを経由して送ろうとする。迂回路であるルーターY経由は使われない。

うむ。確かにその通りだ。だがその場合、**管理者が手動で経路を書き換える必要がある**な。だがこの方法では、いつ起こるかわからない障害が起きたときのために、管理者をルーターの前に張りつかせておく必要がある。これは大変だ。よって自動化する。

自動化？　ということは、ルーターが障害を見つけて、勝手にルートを変更するようにですか？

その通り、それが**動的ルーティング**だ。**ルーターが自動で情報を交換し合い経路を知る**方法だ。

なるほど。ルーター同士で情報を交換して経路を知るんですね。

そして、**すべての経路の中から自動で最適なものを選び、ルーティングテーブルを作成する**。これが動的ルーティングだ。

障害があった場合は、それは最適な経路ではなくなるので、新たな最適経路にルーティングテーブルを書き換えるんですね。はは〜、うまく考えてますね。

ただし、弱点も持つ。1つ目は、ルーター同士が情報を交換し合うということは、データを送り合うということだ。つまり、その分の回線の転送を圧迫する。データ転送に使われる分が減ってしまうのだ。

む〜。それは駄目なんじゃないんですか？

うむ。正直あまりよくはない。特に**低速な回線を使用している場合には、注意が必要**だ。だが、障害によってデータ転送ができなくなるよりは、こちらの方がまだましなのだよ。さらに、2つ目の弱点として、**交換し合った情報から、最適な経路を計算する必要がある**という点だ。その分のルーターの処理能力が必要だ。能力の低いルーターだと、経路計算に処理能力がとられて、データグラムの転送の処理が遅れてしまうことがある。

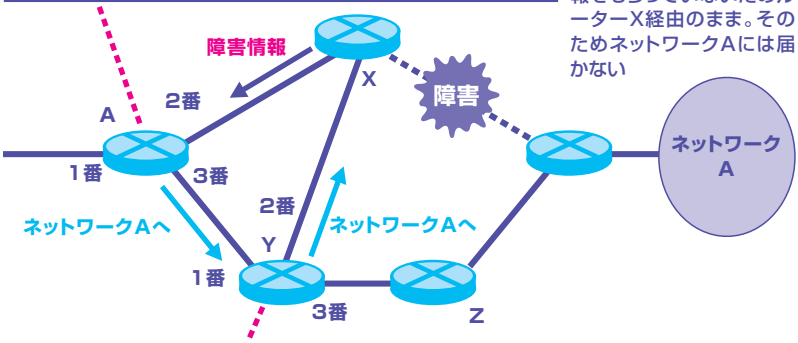
それも全然よくないことじゃないですか？

図4-3　コンバージェンス

ネットワーク内のすべてのルーターが
正しい経路情報を持たないと、あて先へ正しく届かない

あて先	次ルーター	送信ポート	距離
~~ネットワークA~~	~~ルーターX~~	~~2番~~	~~1~~
ネットワークA	ルーターY	3番	2

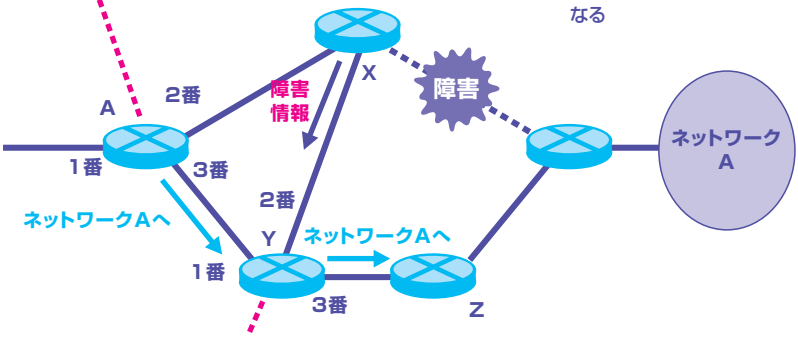
ルーターAはXより障害情報を入手したため、ルーターY経由に切り替えた。しかしルーターYはまだ障害情報をもらっていないためルーターX経由のまま。そのためネットワークAには届かない



あて先	次ルーター	送信ポート	距離
ネットワークA	ルーターX	2番	2

あて先	次ルーター	送信ポート	距離
~~ネットワークA~~	~~ルーターX~~	~~2番~~	~~1~~
ネットワークA	ルーターY	3番	2

ルーターYにも障害情報が届くことにより、ルーターYも経路を切り替え、ルーターZ経由にした。これによりネットワークAへ届くようになる



あて先	次ルーター	送信ポート	距離
~~ネットワークA~~	~~ルーターX~~	~~2番~~	~~2~~
ネットワークA	ルーターZ	3番	2

そうだな。そして、最後にして最大の弱点は、**すべてのルーターが同一の情報を持つ必要がある**という点だ。すべてのルーターが同一の情報を持っている状態のことを**コンバージェンス**［Convergence］というが、ネットワークのルーター達は**コンバージェンスになっている必要がある**。(図4-3)

こんばーじぇんす。「ここに障害があるよ」とか「新しいネットワークができたよ」とかいう情報を全部のルーターが持ってないとダメってことですね。それが「同一の情報」を持つってことですか。ん～なんか弱点が多いので、大変ですね。

確かにいろいろと面倒な部分が多い。だが、**自動で障害を切り離せるというのはそれだけ重要**なのだよ。

確かにそうかも。障害を切り離せなかったら、パケットが届かないんですものねぇ。

というわけで、もうちょっとルーティングについて話そう。ではまた次回。

いぇっさー。3分間ネットワーク基礎講座でした～♪

ネット君の今日のポイント

- ルーターはルーティングテーブルを参照し、あて先へのルートを決定する。
- ルーティングテーブルには、あて先ネットワーク、次の中継ルーター、距離、送信インターフェースが記載されている。
- ルーティングテーブルを作るため、ルーターは他ネットワークへのルートを知る必要がある。
- 知る方法は、静的ルーティングと動的ルーティングがある。
- コンバージェンスである必要がある。

○月○日 晴れ ネット君

第5回

ルーティングプロトコル

●自律システム

前回、ルーターが使うルーティングには２種類あるという話をしたな。ルーティングは、**静的ルーティング**と**動的ルーティング**の２つの方法があるのだった。

静的ルーティングは管理者が手動入力。動的ルーティングは**ルーターが自動でルートを決定**する、と。

そうだ。なにか障害が発生した際、迂回路を作るなどの**冗長性を維持するため**（*1）、動的ルーティングを使うことが多い。特に大規模ネットワークでは必須と言っていい。

障害が発生したら、パケットが届かなくなってしまいますもんね。迂回路が作れないと困りますよね。

うむ。そのためルーターは、動的ルーティングを実現する**ルーティングプロトコル**［Routing Protocol］が利用できるようになっている。ルーティングプロトコルは、**近接ルーターとの間でネットワークの情報を交換し合う**ためのルールだ。

ネットワークの情報を交換し合う？　それでどうするんですか？

そして、**交換した情報を元にしてルーティングテーブルを変更する**。この２つが、ルーティングプロトコルの機能だ。

（*1）冗長性　［Redundancy］。余分や重複があること。ネットワークでは予備を持つことで障害などに対応できることを指す。

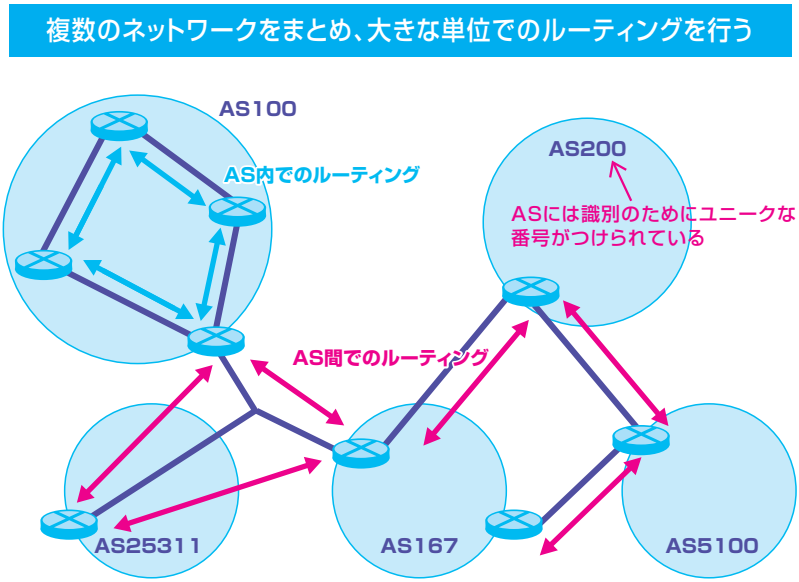
情報を交換して、それによってルーティングテーブルを変更する。なんか簡単そうなんですけど。

そうでもない。これがなかなか奥深い。まず、ルーティングプロトコルを説明する前に説明しておくことがある。それは**自律システム**(*2)［Autonomous System］と呼ばれるものだ。通常は頭文字をとって、**AS**と呼ばれる。

自律しすてむ？　えーえす？

うむ、エーエスと読む。**1つの管理団体によって管理されるネットワークの集合体**だ。ルーティングでは、ASは1つの範囲として扱われる。インターネットには、あまりに多くのネットワークが存在する。なので、インターネットでは同じ組織が管理する複数のネットワークをASとしてまとめてしまうのだ。
あて先のASへ届けるルーティングを行い、次にAS内部で各ネットワークへ届けるルーティングを行う、という形になる。(図5-1)

図5-1　AS



●ルーティングプロトコルの種類

なぜ AS の話をしたかというと。ルーティングプロトコルは大きく分けて**2種類**ある、という説明をしたかったからなのだ。**AS 間のルーティング用**と**AS 内部のルーティング用**の 2 種類だ。それぞれ、**EGP**［Exterior Gateway Protocol］、と **IGP**［Interior Gateway Protocol］と呼ばれる。

2 種類、EGP と IGP、ですね。

うむ。ただし、EGP、IGP はあくまでルーティングプロトコルの種類を表す言葉だ。そういう名前のルーティングプロトコルを使用するわけではない。例えば EGP では、BGP と呼ばれるプロトコルがスタンダードだし、一方、IGP では、それぞれの AS でそこの管理者が AS の状態に合わせてプロトコルを選ぶ。**(図 5-2)**

図 5-2　ルーティングプロトコルの種類

ルーティングを行う規模や動作によって
ルーティングプロトコルは種類がある

種別	ルーティングプロトコル	動作
EGP	EGP (Exterior Gateway Protocol)	ディスタンスベクター
	BGP (Border Gateway Protocol)	パスベクター
IGP	RIP (Routing Information Protocol)	ディスタンスベクター
	OSPF (Open Shortest Path First)	リンクステート
	IS-IS (IntermediateSystem to IntermediateSystem)	リンクステート
	EIGRP (Enhanced Interior Gateway Routing Protocol)	ハイブリッド

(*2) 自律システム　［Autonomous System］。経路ドメイン、経路制御ドメインとも呼ばれる。

なんすか、この、ディスタンスベクターとか。リンクステートとか？

ルーティングプロトコルの動作の方式によって種類があるのだよ。つまり、プロトコルを使う場所によって、EGP と IGP の 2 種類に。動作の方式によって、4 種類に分けることができる、ということだ。

●ルーティングプロトコルが行うこと

では、ルーティングプロトコルはなにをするか、という点を話そう。先ほども言った通り、ルーターは**近接するルーター間でネットワークの情報を交換し合う**。このネットワークとつながっていますよ、あのネットワークを知っていますよ、あっちのネットワークは障害でつながりませんよ、という情報だな。
そして、この情報交換を**いつ行うか**、**どうやって行うか**、**誰に送るのか**、**どのような情報を送るのか**ということなどをルーティングプロトコルが決定している。

いつ、どうやって、誰に、どんな情報を送るのか、を決定するのがルーティングプロトコル、と。

そして、ルーティングプロトコルによって決定された手段により、情報を交換し合い、**コンバージェンス**に達するわけだ。**（図 5-3）**

こんばーじぇんす。**すべてのルーターが同一の経路情報を持つこと**でしたっけ（P24 参照）。

そうだ。前回も出てきた通り、持っている経路情報に食い違いがあると、正しく届かない可能性があるからな。

でしたよね。障害があるとか、新しく追加されたとかの情報をみんなで共有するんですね。

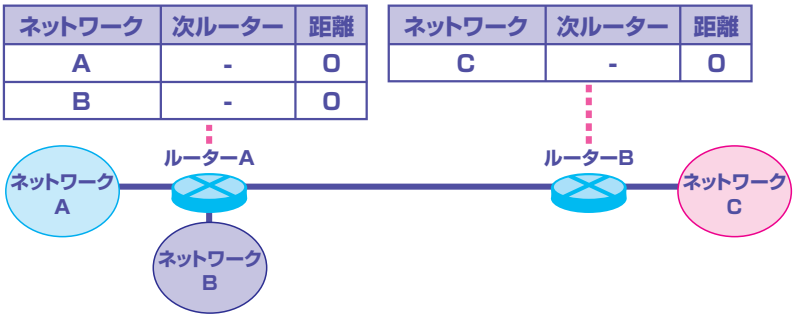
この交換し合ったルート情報を元に、**最適な経路をルーティングテーブルに載せる**。これにより、**常に最適な経路が使用可能**ということになる。

なるほど。ルーティングプロトコルによりルーティングテーブルが作られる、ということですね。

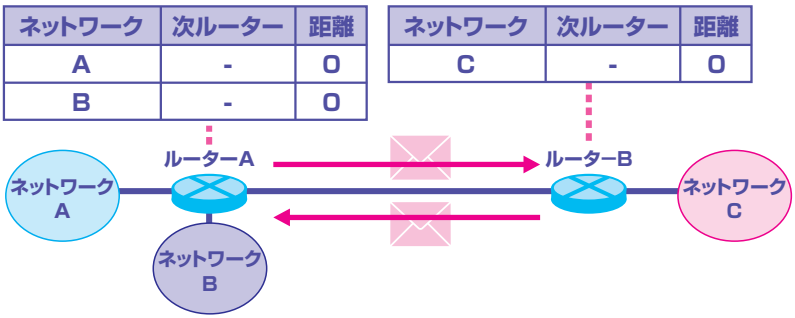
図5-3　ルーティングプロトコルが行うこと

経路の情報をいつ、どうやって、誰に、どんな情報を送るのかを決定する

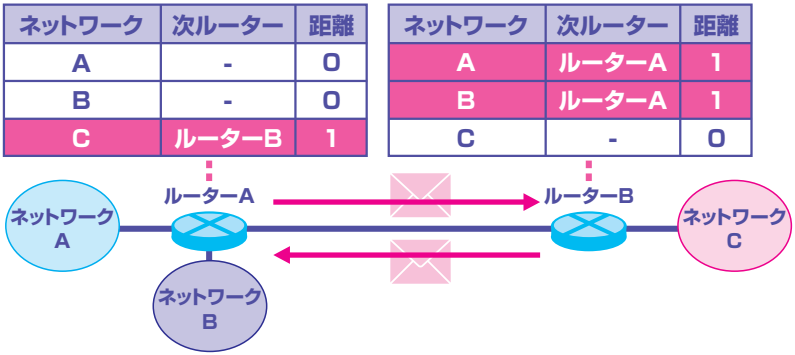
①ルーターは自分が接しているネットワークをテーブルに保持している



②ルーティングプロトコルを使って、持っているネットワークの情報を交換しあう



③交換した情報を使って、ルーティングテーブルを更新する



そういうことだ。次回は、ルーティングプロトコルの 1 つである RIP を例にして、ルーティングプロトコルをもう少し詳しく説明する。

了解ッス。3 分間ネットワーク基礎講座でした〜♪

ネット君の今日のポイント

- 動的ルーティングはルーティングプロトコルで実現される。
- ルーティングプロトコルは方法と場所によって複数種類がある。
- ルーティングプロトコルによって、近接ルーター間でネットワークの情報を交換し合う。
- いつ、どうやって、誰に、なにを送るのかをルーティングプロトコルが決定する。

○月○日 ネット君

「BGP(Border Gateway Protocol)をご存じですか?」

BGP は、現在使用されているバージョンは 4 のため、BGP4 とも書かれる。AS 間のルーティングを担うルーティングプロトコルである。数十万〜百万と言われるインターネット上のネットワークの経路情報をやりとりするため、信頼性と大量の情報をやりとりする仕組みを持つ。また、ポリシーが異なる AS 間で使用されるため、パス属性という値によって最適経路を決定する柔軟さを持つ。現在では拡大するインターネットとそのサービスのため、IPv6 や MPLS への対応（BGP+ や MBGP）、経路情報の信頼性、障害の検出など様々な拡張が行われている。

第6回

RIP(Routing Information Protocol)

●メトリック

さて、ネット君。ルーティングの話が続いているが。これまで何回か、「最適な経路」という言葉を使ったが、「最適」とは曖昧な言葉じゃないかね？「最短」でも「最速」でもなく、「最適」という言葉なのはなぜかね？

え？　いや、あの。最短＝最適だと思ってたんですが。違うんですか？

違う。最短な経路が、必ずしも最適な経路は限らない。昔からよく言うだろう、「急がば走れ」と。

いや、それはそれであってますけど。それを言うなら、「急がば回れ」でしょ。

ああ、それだ。つまりだ。渋滞中の高速道路より、すいている一般道の方が早いことがあるよな？　山を直接横断する山道よりも、迂回する高速道路の方が早かったり。

あ〜、確かにそういうことってありますよね。でも、どっちを使うかは人によって違うんじゃないんですか？　早く着く方がいい人もいれば、短い距離の方がいい人もいたり。

そう、「最短」が最適なのか、「最速」が最適なのかを判断する必要があるわけだな。つまり、「最適」を判断する基準があって、はじめて「最適」と決定されるというわけだ。この**最適な経路を決定する際の判断基準**のことを、**メトリック**［Metric］という。

めとりっく。最適を決めるための値、という意味になるんですかね？

そうだ。中継するルーターの数や、回線のスピード、混み具合、エラー発生率などの判断基準からルーティングプロトコルによって決められた値を計算して、その中で**最小の値を持つものを最適な経路**とするのだよ。（図 6-1）

● RIP（Routing Information Protocol）

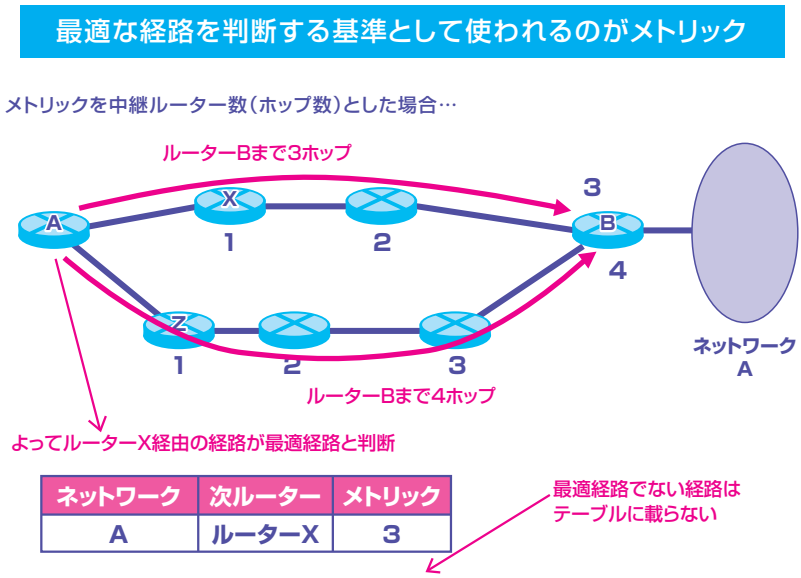
さて、実際のルーティングプロトコルの話で、ルーティングプロトコルの動作をわかってもらおうか。今回説明するのは RIP だ。

えっと、前回ではディスタンスベクターとかなんとかいう動作だという説明でしたよね。

そうだ、ディスタンスは「距離」、ベクターは「方向」、つまりディスタンスベクターとは、**距離と方向**のことだ。

距離と方向？　なんか変な名前ですね。

図 6-1　メトリック



RIP の動作の中心にあるのが「距離と方向」だから、この名前がついている。ルーターが他のルーターと交換する情報のことを、RIP では**ルーティングアップデート**［Routing Update］という。

るーてぃんぐあっぷでーと？　アップデートって、「更新」ですよね。

そうだ。「経路更新情報」とでも言うべきものかな。RIP ではこれを交換し合う。ではこの情報交換で、どのような情報を交換するかというと、**ルーティングテーブルをそのまま交換し合う**。

ルーティングテーブルをそのまま？

そのままだ。これを、**30 秒に 1 回**送る。それにより、ルーティングテーブルの新しい情報を交換し合うわけだ。

なるほどなるほど。定期的にやり取りすることによって、新しい情報を常に入手できるわけですね。

そして、このアップデートを**６回受け取らなかったら、そのルーターにはなんらかの障害が発生したとみなす**。この場合、**そのルーターを使うルートを消してしまう**。

返事がないので、もういないものとみなすわけですね。

そういうことだ。次の図が、RIP の動作を説明した図だ。あまり複雑な形を説明しても長くなるだけなので、3 つのルーターで説明しよう。**(図 6-2)**

はー、ルーティングテーブルの情報を送ってもらって、知らない情報を追加していく、と。こうやってルーティングテーブルを更新するわけですね。

そうだ。そして、RIP は**メトリックとしてホップ数を使う**。ホップ数とは、あて先ネットワークまでに**通過するルーターの数**だ。

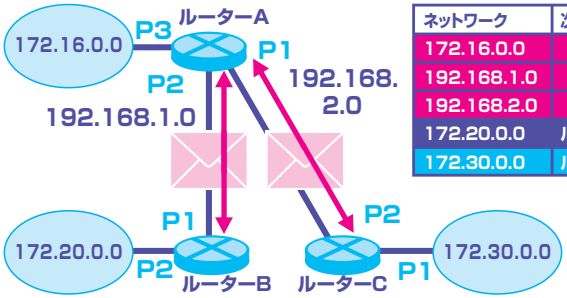
だから、直接接続されているネットワークのメトリックが 0 なんですね。

うむ。RIP は簡単に言うと、アップデートを受け取ったら、**自分の知ら**

図6-2　RIPの動作1

ルーティングアップデートを交換し、知らない経路情報を入手する

①自分に接続しているネットワークに加え、隣接しているルーターの情報（ルーターAが持つ情報はオレンジ、ルーターBが持つ情報が黒、ルーターCが持つ情報が緑）がRIPによりルーティングテーブルに追加される

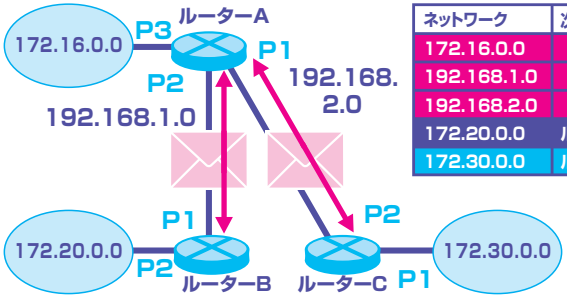


ネットワーク	次ルーター	メトリック	ポート
172.16.0.0		0	P3
192.168.1.0		0	P2
192.168.2.0		0	P1
172.20.0.0	ルーターB	1	P2
172.30.0.0	ルーターC	1	P1

ネットワーク	次ルーター	メトリック	ポート
192.168.1.0		0	P1
172.20.0.0		0	P2
192.168.2.0	ルーターA	1	P1
172.16.0.0	ルーターA	1	P1

ネットワーク	次ルーター	メトリック	ポート
192.168.2.0		0	P2
172.30.0.0		0	P1
192.168.1.0	ルーターA	1	P2
172.16.0.0	ルーターA	1	P2

②さらに次の更新で、先ほど更新された情報がやり取りされる。
（ルーターBにルーターCの情報が届き、ルーターCにルーターBの情報がAを経由して届いている）これでコンバージェンスになる



ネットワーク	次ルーター	メトリック	ポート
172.16.0.0		0	P3
192.168.1.0		0	P2
192.168.2.0		0	P1
172.20.0.0	ルーターB	1	P2
172.30.0.0	ルーターC	1	P1

ネットワーク	次ルーター	メトリック	ポート
192.168.1.0		0	P1
172.20.0.0		0	P2
192.168.2.0	ルーターA	1	P1
172.16.0.0	ルーターA	1	P1
172.30.0.0	ルーターA	2	P1

ネットワーク	次ルーター	メトリック	ポート
192.168.2.0		0	P2
172.30.0.0		0	P1
192.168.1.0	ルーターA	1	P2
172.16.0.0	ルーターA	1	P2
172.16.20.0	ルーターA	2	P2

ないネットワークをテーブルに追加する。その際、**アップデートを送ってきたルーターを、その先のネットワークへの中継ルーターに、アップデートを受け取ったインターフェースを、その先のネットワークへの送信インターフェースに**することを行う。

これで、隣のルーターが知っているネットワークの情報が入手されるわけですね。

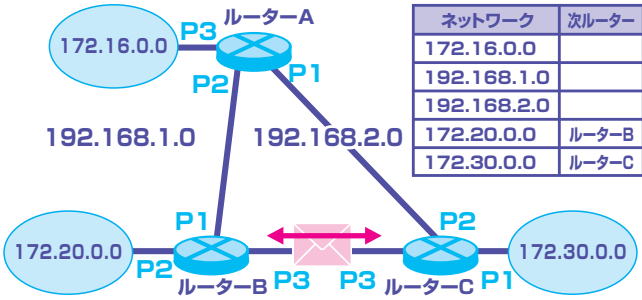
そうだ。教えてくれたルーターがいる「方向」が、あて先ネットワークへの経路となる。
さて次は、知っているネットワークの情報をアップデートで教えてもらった場合の例だ。先ほどの例のルーターＢとルーターＣをつないだ形の例だ。**（図６−３）**

メトリックによって経路を比較して、メトリックが小さい方をルーティングテーブルに載せる、ですか？

図 33−3　RIP の動作 2

知っている経路の情報を入手した場合、メトリックで判断する

ルーターBとルーターCが直結したことにより、ルーターA経由の経路（メトリック2）より直接届いた経路（メトリック1）の方がよいため、ルーティングテーブルが更新されている



ネットワーク	次ルーター	メトリック	ポート
172.16.0.0		0	P3
192.168.1.0		0	P2
192.168.2.0		0	P1
172.20.0.0	ルーターB	1	P2
172.30.0.0	ルーターC	1	P1

ネットワーク	次ルーター	メトリック	ポート
192.168.1.0		0	P1
172.20.0.0		0	P2
192.168.2.0	ルーターA	1	P1
172.16.0.0	ルーターA	1	P1
172.30.0.0	ルーターC	1	P3

ネットワーク	次ルーター	メトリック	ポート
192.168.2.0		0	P2
172.30.0.0		0	P1
192.168.1.0	ルーターA	1	P2
172.16.0.0	ルーターA	1	P2
172.16.20.0	ルーターB	1	P3

そうだ。RIP では、**すでにテーブルに存在するネットワークについて、新たな情報がアップデートで来て、新しい経路の方がメトリックが小さい場合、そちらをテーブルに載せる。**新しい経路の方がメトリックが大きいならば、それは無視する。
メトリックという経路を評価する基準は、RIP だとホップ数になる。ホップ数とは、経由するルーターの数のことだから、イメージ的には「距離」と言ってもいいだろう。

教えてもらったルーターがある「方向」、ホップ数という「距離」。だから距離と方向でディスタンスベクター？

そういうことだ。このディスタンスベクターの RIP は簡単でわかりやすいルーティングプロトコルとして今回例に出したが、結構古いプロトコルなので欠点を持つ。例えば、ホップ数という単純な評価で経路を決定するので、「経由ルーター数が多いが高速な回線」「経由ルーター数が少ないが低速な回線」では、低速な回線を選んでしまったり。また、「ルーティングループ」と呼ばれる経路がループ状になってしまい、あて先まで到達不能になってしまう現象が発生したり。他にもアップデートに対するセキュリティが全くないので、改ざんなどの攻撃に対処できなかったり、だな。これらの問題のいくつかに対処するために、RIP の新バージョンである RIP2 や、その他前に名前をあげた OSPF などの他のルーティングプロトコルが使われる。ま、これらは RIP に比べるとかなり複雑なので別の本「3 分間ルーティング基礎講座」などで勉強してくれたまえ。今回はこれでおしまいとしよう。

ネット君の今日のポイント

- ホップ数、回線のスピード、混み具合、エラー率などが基準となる。
- RIP はディスタンスベクター型ルーティングプロトコルである。
- ルーティングアップデートという情報を交換する。
- ルーティングテーブルをアップデートとして送る。
- 30 秒に 1 回、アップデートを送る。
- RIP のメトリックはホップ数である。
- アップデートで送られてきたルーティングテーブルを自分のものと比べて、テーブルを更新する。

〇月〇日 ネット君

第7回

ICMP

● ICMP

さて、ネット君。ここまで「アドレッシング」と「ルーティング」を説明してきたわけだ。この２つは、レイヤー３の「インターネットワークを実現する」という役割のために必要不可欠な機能だったわけだ。

はい。「アドレッシング」でアドレスのつけ方が決まって、「ルーティング」であて先までの経路を決定する、でしたよね。

よしよし、ちゃんと覚えているな。さて、レイヤー３のプロトコルとしては、IP がもちろん最重要であるのは間違いない。だが、レイヤー３のプロトコルは IP 以外にも存在する。

そうなんですか？　レイヤー３は「インターネットワークを実現」するのが目的で、IP がそれを行うんですよね。他にどんなプロトコルがあるんですか？

それは **ICMP**［Internet Control Message Protocol］というプロトコルだ。

あいしーえむぴー。インターネットをコントロールするメッセージのプロトコル？

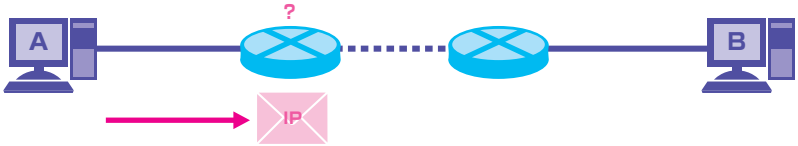
そうだな、直訳すれば**インターネット制御メッセージプロトコル**。役割的に翻訳すると、**エラー報告プロトコル**ってところだな。
例えばこんな風に使われる。**（図 7-1）**

「送信不能メッセージ」を受け取ったホストはどうするんです？

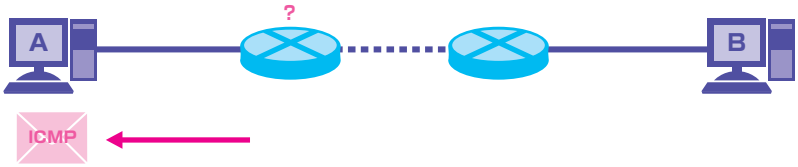
図 7-1　ICMP

ネットワークの制御・管理などを行うプロトコル

①ホストAからホストBへIPパケットを送信したが、ルーターはホストBのあるネットワークへの経路を知らなかった



②ルーターは宛先へ到達できないことを示す送信不能メッセージをICMPでホストAに通知する



それは個々のアプリケーションによって対応が異なる。このように、**ネットワークの制御・管理**に使用されるのがICMPなのだよ。さて、どのようなデータをやり取りするかというと、IPデータグラムにICMPメッセージを入れる。

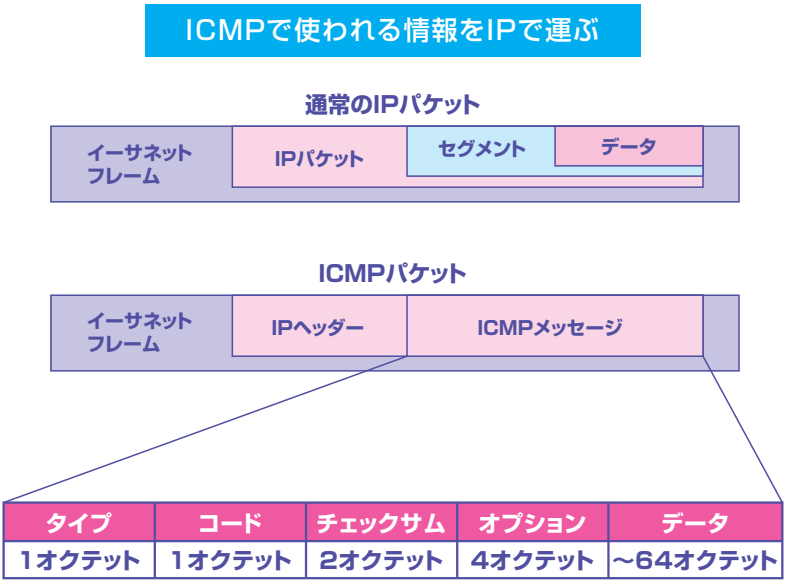
ICMPメッセージ？　それはいったいどんなものなんですか？

ICMPで使われる情報だな。この情報を、IPデータグラムのペイロード（P132の図参照）に入れる。通常、IPデータグラムのペイロードにはTCPセグメントかUDPデータグラムが入るのだが、これらの代わりにICMPメッセージを入れて送るのだよ。**（図 7-2）**

IPヘッダー＋ICMPメッセージの形になるわけですね。で、ICMPメッセージはタイプと、コードと……？

まぁ、他にも項目はあるが、重要なのは「タイプ」とそれに付随する「コード」になる。**タイプはICMPの種類**、コードはその詳細だな。

図 7-2　ICMP メッセージ



● ICMP の種類

さて。ICMP には大きく分けて、2 種類のメッセージがある。**Query メッセージ**と **Error メッセージ**だ。
Query は**状態を調査するため**に使用されるメッセージ。Error はそのまま**エラーを通知するため**のメッセージだ。

ははぁ、ICMP は状態の調査にも使用するんですね。さすが Internet Control のプロトコルですね。

うむ。この 2 種類に、それぞれ**複数のタイプ**が存在する。それは、先ほどのメッセージの中の「タイプ」に数字で表されている。タイプは 11 種類（*1）存在する。(図 7-3)

1 番とか 2 番とか、7 番が抜けているのはなにか意味があるんですか？

(*1) **11 種類**　最初の ICMP の規定で決められているのが 11 種類だが、拡張として他の種類を使うこともある。

ふむ。それらはもともと定義されていない番号だ。私が省略したわけではない。この中で頻繁に利用されるのは 0、3、5、8、11 だな。そうだな、例として 3 番 Destination Unreachable はこうなる。**(図 7-4)**

ははぁ、あて先［Destination］へ届かない［Unreachable］、ですね。

● TTL

さて、ICMP のその他の代表的なメッセージを説明するが、その前に IP ヘッダーの項目の説明をしておこう。

えっと、IP ヘッダーってことは。IP によって TCP セグメントや UDP データグラムをカプセル化して IP データグラムにする際に、付加される IP の制御情報ですよね。IP ヘッダーは 20 オクテットで、送信元 IP アドレス・あて先 IP アドレス、IP データグラムを分割する際に使用するフラグやオフセット、ヘッダーチェックサムなんかが IP ヘッダーに含まれていましたよね。

図 7-3　ICMP タイプ

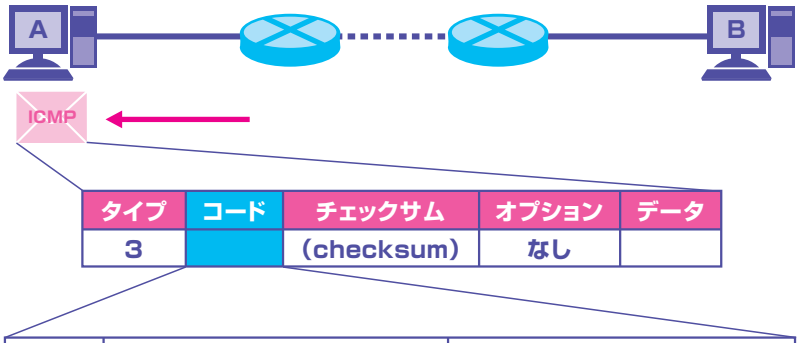
ICMPメッセージにはQueryとErrorの2種類が存在する

タイプ	説明	意味	種類
0	Echo Reply	Echo応答	Query
3	Destinaiton Unreachable	あて先到達不能	Error
4	Source Quench	転送抑制指示	Error
5	Redirect	最適経路通知	Error
8	Echo Request	要求	Query
11	Time Exceeded	時間超過による パケット破棄	Error
12	Parameter Problem	誤ったパラメータ によるエラー	Error
13	Timestamp Request	タイムスタンプ要求	Query
14	Timestamp Reply	タイムスタンプ応答	Query
15	Information Request	（未使用）	Query
16	Information Reply	（未使用）	Query

図 7-4　Destination Unreachable

あて先に届かない理由を通達する

あて先に到達できない場合、ルーターまたはホストがタイプ3
Destination Unreachableを返す
その際、到達できない理由をコードに入れる



タイプ	コード	チェックサム	オプション	データ
3		(checksum)	なし	

コード	説明	意味
0	Net Unreachable	ネットワークへ到達不能
1	Host Unreachable	ホストへ到達不能
2	Protocol Unreachable	そのプロトコルは使用できない
3	Port Unreachable	対象ポートが閉じている
4	Fragmentation Needed and DF Set	IPパケットを分割したいが、分割が不可になっている

※他にもコードはある

うむ。その際に説明しなかった項目で、ICMP と非常に関連がある項目があるのだよ。それは **TTL**［Time To Live］だ。日本語に訳せば「生存時間」だな。

生存時間？　なんの生存時間ですか？

IP データグラムのだ。**IP データグラムの TTL はルーターを経由するたびに 1 ずつ減っていき、0 になるとそのデータグラムは破棄される。**まさしく、IP データグラムの「生存時間」というわけだ。

ルーターを経由するたびに 1 ずつ減って、0 になったら破棄。死へのカウントダウンですね。でも、なんでそんな項目があるんですか？

ルーティングで経路情報にミスが起きる、例えば静的ルーティングで手動で経路を入力する際に、中継ルーターを間違えてしまうとかだな。本来はそうすると、あて先へ届かずにどこかへ行ってしまったり、挙句の果てに同じ場所をぐるぐる回り続けてしまったりすることがある。そうなるとそのデータグラムは永遠にネットワーク内を循環し続ける。このようなデータグラムは邪魔なだけだ。

ふむふむ。ルーティングの方向指示がミスって、あて先に届かなくなるんですね。そうなると確かに邪魔になりますね。

なので、一定時間が経ったら破棄する。実際は時間ではなく、経由ルーター数で判断するわけだがな。

邪魔になるから、破棄するわけですね。で、実際はどれぐらいルーターを経由すると破棄されるんですか？

Linux では 64、Windows では 128 が多いな。インターネットでは世界の裏側に送ってもルーターは 30 個を越えるぐらいしか経由せずに届くので、64 とか 128 経由することはあきらかにルーティングを失敗しているのだよ。さて、今回はここまでとしておこう。

いぇっさー。3 分間ネットワーク基礎講座でした〜♪

ネット君の今日のポイント

- エラーメッセージなどを転送するプロトコルが ICMP。
- ICMP では IP ヘッダー +ICMP メッセージを送信する。
- ICMP には Query と Error の 2 種類のメッセージがある。
- タイプ 3 の ICMP はあて先へ届かないことを通知する。

○月○日 ネット君

第8回

Echo と Time Exceeded

● Echo

さて、前回からレイヤー 3 のプロトコルの 1 つ、ICMP を説明しているわけだ。ICMP エラーを通知したり、通信状態を確認したりするプロトコルが、ICMP だ。

インターネット制御メッセージプロトコル、でしたよね。前回は Destination Unreachable、あて先到達不能を通知するメッセージの説明がありました。

うむ、タイプ 3 だな。今回はタイプ 0、8、11 の 3 種類を説明する。まず Echo、タイプ 0 と 8 の話をしよう。

えこー？　エコーっていうと、あれですか。反響音というか、こだまというか。

そうだ、あとは山彦だな。ともかく、その「Echo」、タイプ 0 の**エコー応答**［Echo Reply］とタイプ 8 の**エコー要求**［Echo Request］だ。

「要求」と「応答」…。なにを「要求」して、なにを「応答」するんですか？

なにを、と言われても困るな。エコーを要求して、エコーを応答するんだ。つまり**送信側はエコー要求を送り、それを受け取ったコンピューターはエコー応答を返す**というしくみだ。**（図 8-1）**

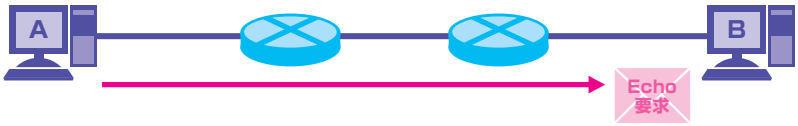
ははぁ。まさしく、「エコー」なんですね。要求すると、返答する。「やっほー」といえば「やっほー」と帰ってくる。

そうだ。たったコレだけのしくみだ。

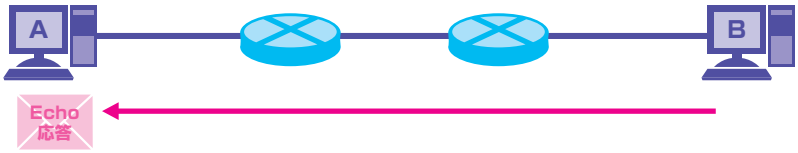
図 8-1　Echo Request と Echo Reply

エコー要求を受信すると、エコー応答を送り返す

①ホストAからホストBへ、Echo Request（エコー要求）を送信すると…



②受信したホストBは送信元（ホストA）へEcho Reply（エコー応答）を送り返す



はぁ。これ、なんかの役に立つんですか？

この「Echo」のしくみを利用したもので、**ping**という**任意のあて先へエコー要求を送りつける**ソフトウェアがある。このソフトウェアは**ネットワーク管理者御用達。このコマンドを使わない管理者はいない**とまで断言できるシロモノだ。

任意のあて先へエコー要求を送りつける？　エコー要求を送りつけると、それを受け取ったあて先はエコー応答を返してきますよね。

その通り。エコー要求を受け取ったあて先は、エコー応答を返してくる。その結果、エコー要求を送った送信元はエコー応答を受け取ることになる。これはつまり、エコーの要求と応答がやり取りされる、つまり**送信元とあて先間でデータが送受信される**という意味だ。

あ〜、なるほど。ICMP パケットが送信元とあて先の間を行き来できるんですね。もしエコー要求を送って、エコー応答が返ってこなかったら、それは行きか帰りのどちらかに問題があってやり取りできない、ってことですよね。

そうだ。さらに、エコーの要求と応答にかかる時間を計ることにより、**ネットワークの状態を調べることもできる**。他にも分割不可にして、データサイズを変えることにより回線の MTU を調べることができたりもする。ネットワークの基本的な状態を確認する、非常に有益なソフトウェアということだな。

● Time Exceeded

次はタイプ 11 の Time Exceeded メッセージだな。これは、「時間超過によるパケット破棄」とでもいうメッセージだ。

「時間超過」ってなんです？

「時間超過」は、前回説明した **TTL が関係している**（P217 参照）。TTL が切れたパケットは破棄される。このとき、**破棄したことを通知するメッセージが Time Exceeded** だ。**(図 8-2)**

なるほど。TTL という生存時間が切れたから、Time Exceeded で「時間超過」なんですね。

そういうことだ。このタイプ 11 を使ったネットワークのチェック用のソフトウェアがある **traceroute** というソフトウェアだ。Windows では、「**tracert**」、Linux では「**traceroute**」というコマンド（シェル）という形で実装されているな。ただ、普通に使用すると結構時間がかかるので、タイムアウト時間を短めにとったりしたほうがいいかな。

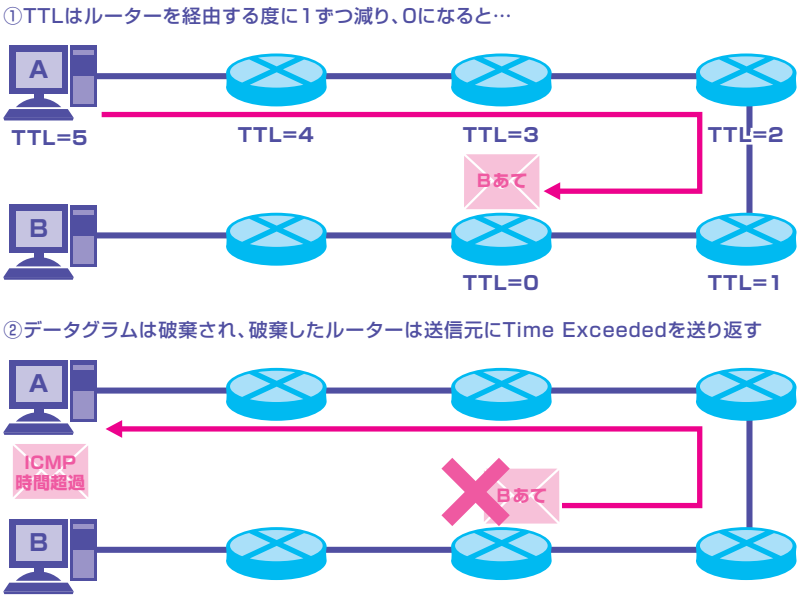
とれーするーと？　ルートをたどる？

うむ、その通り**あて先までのルートを教えてくれるソフトウェア**だ。正確には、**あて先に届くまでに経由するルーター**を教えてくれる。**(図 8-3)**

ははぁ。あて先に届くまでに、どのルーターを通っていくか、を教えてくれるんですね。意図的にエラーメッセージをもらい、それを表にしていくんですね。うまくできてるなぁ。

図 8-2　Time Exceeded

TTLが切れてデータグラムが破棄されると、
破棄したルーターはTime Exceededを送信する



うむ。これにより**どのルートをたどっていったかがわかる**わけだ。これらの ping や traceroute は非常に便利なコマンドであるし、使って手に入った情報も非常に有益だ。

ですね。ping ならあて先に届くかどうか確認できますし、traceroute は途中のどういう経路を通っていくかを調べることができますよね。

だが、これらの情報は、**クラッカー**［Cracker］**（*1）**の攻撃にも役に立つ、ということも事実だ。

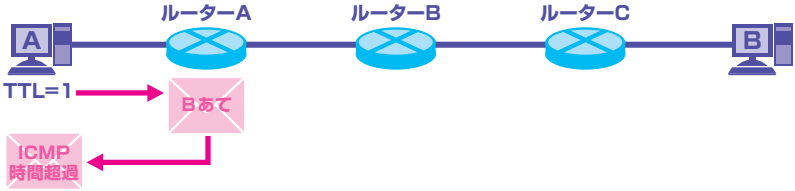
くらっかーの攻撃……？　**悪用される**ってことですか？

（*1）クラッカー　システムのセキュリティを破り、不正にコンピューターに侵入して悪意のある行動を行う人のこと。一般的にはハッカー［Hacker］と混同されている。

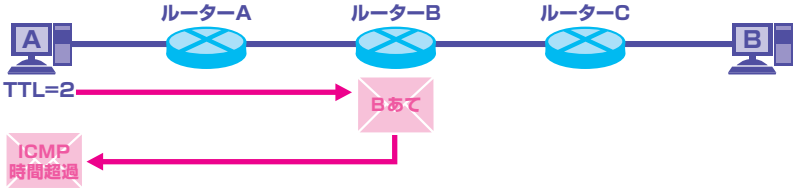
図8-3　traceroute

あて先まで経由するルーターを調べることができる

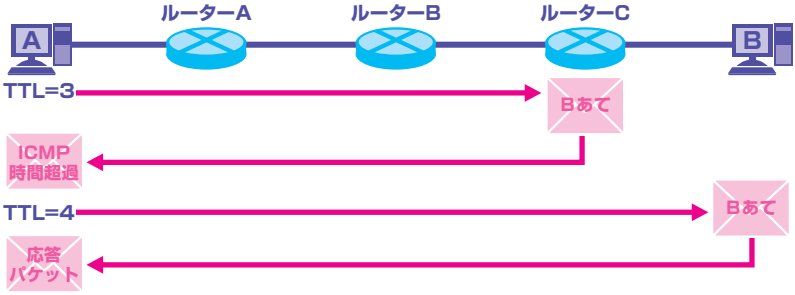
①TTL=1であて先へパケットを送ると、1つ目のルーターでTime Exceededが返ってくる



②次はTTL=2であて先へパケットを送り、2つ目のルーターでTime Exceededが返ってくる



③以後、TTLを1ずつ増やしていき、あて先までパケットを届ける



④あて先までデータが届き、応答パケットを受信したら、いままでTime Exceededを送り返してきたルーターを表示する。それがあて先までの経由ルーターの一覧になる

ホストBまでの経由ルーター表

ルーターA	ルーターB	ルーターC	ホストB

そうだ。なので、**ルーターの管理者は、ICMP の運用に注意する必要がある。**特に traceroute で使われる Time Exceeded は注意が必要だな。
traceroute により、途中のルーターの IP アドレスを調べることができる。ルーターの IP アドレスがわかると、そこに攻撃をしかけることができる。ルーターが攻撃によりおかしくなってしまうと、ネットワークの広範囲に影響がでてしまうのだよ。

あぁ、ルーターはネットワークの最重要機器、でしたっけ。そこが攻撃されると困りますよねぇ。

その通りだ。さて、今回はここらでおしまいにしよう。

ネット君の今日のポイント

- ICMP タイプ 8 と 0 はエコー要求とエコー応答。
- エコー要求を受け取ったコンピューターは、エコー応答を返す。
- エコー要求に対し、エコー応答が返ってくれば、その相手とはデータの送受信が可能なことを示す。
- エコー要求は ping を使って実行できる。
- TTL によりパケットを破棄したルーターは、送信元に Time Exceeded を送り返す。
- Time Exceeded を使い、あて先までのルートを調べるコマンドが traceroute。

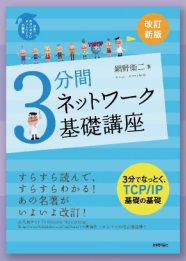
〇月〇日 ネット君

「筆者からのメッセージ」

ネットワークエンジニアは、時代の変化とともに覚えなければいけないことが増えていく傾向にあります。今ですと、クラウド、仮想化、SDN などがありますが、今後またあらたに増えていくことでしょう。ですが、そういう新しい技術を追っかけるためには、しっかりとした足元、基礎知識が重要だと思います。わかっているつもりでも、また勉強し直すと別の発見があるかもしれませんよ。

ネットワーク技術を身につけ極めよう！

[3 minutes networking seminar]



改訂新版 3 分間ネットワーク基礎講座

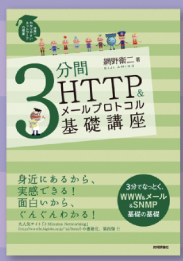
ISBN978-4-7741-4373-6
定価（本体 1780 円＋税）

改訂新版 3 分間ルーティング講座

ISBN978-4-7741-5737-5
定価（本体 1980 円＋税）

3 分間 DNS 基礎講座

ISBN978-4-7741-3863-3
定価（本体 2280 円＋税）



3 分間 HTTP&メールプロトコル基礎講座

ISBN978-4-7741-4081-0
定価（本体 2280 円＋税）

SoftwareDesign 2014 年 9 月号 特別付録小冊子

著者／網野衛二（『改訂新版 3 分間ネットワーク基礎講座』より）
カバー＆レイアウトデザイン／トップスタジオ
発行所／株式会社技術評論社 〒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。