special feature 1 楽しくネットワーク技術を学ぶ

special feature 2 UNIX使い養成

Software Design

2014年5月18日発行
毎月1回18日発行
通巻349号
（発刊283号）
1985年7月1日
第三種郵便物認可
ISSN 0916-6297
定価 本体 1,220円 +税

2014 5

新人歓迎
ITの基礎知識をインストール！

［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

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！

special feature 1

special feature 2

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

短期集中連載

Rettyのサービス拡大を支えた
「たたき上げ」DevOps

Web標準技術で行う
Webアプリの
パフォーマンス改善

新連載

るびきち流
「Emacs 超入門」

Red Hat Enterprise Linuxを使いこなせ！
.SPECS

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65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。

Infrastructure as Code

ITインフラ管理を変える「Infrastructure as Code」

近年、インフラの構築や運用管理の現場で「Infrastructure as Code」と呼ばれる考え方が注目を集め始めています。Infrastructure as Codeとは、ITシステムのインフラの構築や運用をプログラミングのようにコードを書くことで実現しようというアプローチです。

従来、インフラの構成管理は構築手順書を元に手動で行うことが一般的でした。Infrastructure as Codeでは、構築手順書の代わりにソースコードによってインフラ構築時に行うべき作業を定義し、それを実行することで実際の環境に反映させます。たとえば、サーバにOSをインストールし、ミドルウェアを導入し、各種設定を行い、アプリケーションのデプロイなどを行う。この一連の作業をプログラムとして記述するわけです。このようなコード化によって、次のようなメリットが生まれます。

- 作業ミスなどのヒューマンエラーを減らせる
- 複数台のサーバを効率よく構築できる
- 作業手順の曖昧さを回避できる
- コードがバージョン管理できるため、複数世代の構成を保持することが可能
- コードレビューによって品質向上や情報共有が行える
- 継続的インテグレーションが実現できる

Infrastructure as Codeを考えるうえで重要なのは、これが単にインフラ構築を自動化するという話ではなく、インフラをソフトウェアのように扱えるという点に着目することです。コード化することと合わせて、ソフトウェアの世界で培われてきたノウハウを適用することによって、インフラ構築の迅速化やコスト削減、品質の向上につなげようというわけです。

変わり続けるITシステムを実現する

ビジネス市場の急速な変化に対応していくためには、常に最新の動向を取り入れた新しいITサービスを迅速にリリースし続ける必要があります。そのためにはソフトウェアの継続的な更新やデリバリーはもちろんのこと、システムを支えるインフラにも常に手を加えていかなければなりません。

最近では仮想化技術の発達やクラウドの普及によって、サーバやネットワークといったインフラをソフトウェア的に構成し、拡張性や柔軟性が極めて高いシステムを構築できます。ハードウェアへの依存度が下がったことで、インフラの再構築にかかるリスクも以前に比べて大幅に小さくなりました。Infrastructure as Codeという考え方は、その柔軟性をさらに高め、継続的にビジネス価値を創出し続けることを目的として誕生しました。

Infrastructure as Codeが実現すれば、ベースとなるインフラのライフサイクルと、そのインフラの上で提供されるITサービスのライフサイクルを同期させることができるようになります。両者のライフサイクルの違いは、長らく企業のITシステムの柔軟性を阻害する要因のひとつとなっていました。Infrastructure as Codeには、その隔たりを解消する手段として大きな期待が寄せられています。

サポートするツール群

Infrastructure as Codeをサポートする代表的なツールには次のようなものがあります。

- **Puppet**…サーバの環境構築をコードに基づいて自動化する設定管理ツール。言語は独自DSL(Domain Specific Language)を使用
- **Chef**…Puppetと同様にサーバの環境構築をコード化する設定管理ツール。言語はRuby DSL
- **serverspec**…サーバ設定のテストを自動で行うためのフレームワーク。コード化したインフラの継続的インテグレーションが可能になる
- **Docker**…コンテナ技術を用いてサーバ上に独立したOS環境を構築できるツール
- **Serf**…オーケストレーションをコード化するツール。ロードバランサやサーバ管理ツールへの登録などといった作業を自動化できる

これらのツールは、Infrastructure as Codeという考え方が広まり、適用する範囲や方法が明確になるのにともなって今も進化を続けています。Infrastructure as Codeは、インフラ管理のライフサイクルを改善するブレークスルーになるかもしれません。SD

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

あきみち+aico

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 May 2014

［広告索引］

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)**

電子版年間定期購読開始キャンペーン
1年間12,000円(税込み)
(2014年5月17日お申し込み分まで)

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。

PDF電子版発売キャンペーン

2014年5月17日までにGihyo Digital Publishingにて電子版年間定期購読を新規にお申し込みの方は、年間購読料12,000円(税込み)でお求めいただけます。

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

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

Fujisan.co.jpからのお申し込み方法

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

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

Contents

Software Design May 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	retales botijero /gettyimages
Illustration イラスト	フクモトミホ、高野 涼香、中川 悠京
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。

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

技術評論社の本が電子版で読める！

電子版の最新リストは

Gihyo Digital Publishingの

サイトにて確認できます。

http://gihyo.jp/dp

GIHYO Digital Publishing 未来の"普通"を、今。

Google play

amazon.co.jp

※販売書店は今後も増える予定です。

法人などまとめてのご購入については
別途お問い合わせください。

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

Pocket Reference

小さくても、中身充実！

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

岡本隆史、武田健太郎、相良幸範　著
四六判／272ページ
定価（本体2,480円+税）
ISBN978-4-7741-5184-7

石田つばさ　著
四六判／448ページ
定価（本体2,180円+税）
ISBN978-4-7741-4836-6

高江賢　著　山田祥寛　監修
四六判／536ページ
定価（本体2,580円+税）
ISBN978-4-7741-4592-1

古籏一浩　著
四六判／592ページ
定価（本体2,380円+税）
ISBN978-4-7741-4819-9

田口貴久ほか　著　日本仮想化技術株式会社　監修
四六判／352ページ
定価（本体2,980円+税）
ISBN978-4-7741-5600-2

沓名亮典、平山智恵　著
四六判／576ページ
定価（本体2,280円+税）
ISBN978-4-7741-3816-9

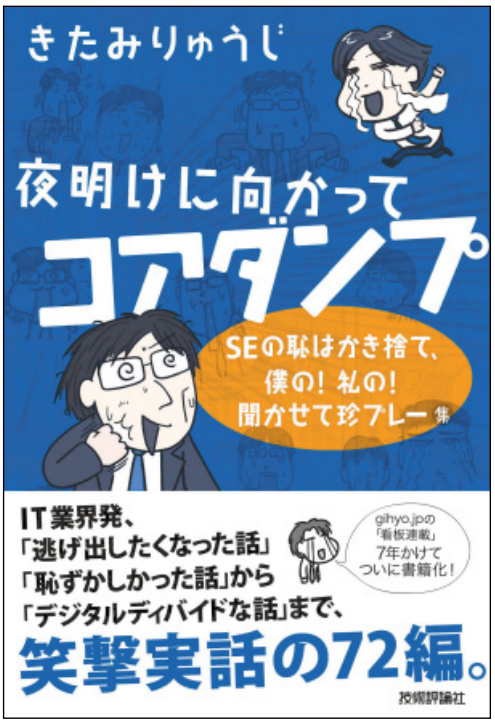
若杉司　著
四六判／400ページ
定価（本体2,980円+税）
ISBN978-4-7741-6332-1

片渕彼富　著　山田祥寛　監修
四六判／512ページ
定価（本体2,780円+税）
ISBN978-4-7741-6127-3

しげむらこうじ　著
四六判／512ページ
定価（本体3,200円+税）
ISBN978-4-7741-6335-2

山森丈範　著
四六判／304ページ
定価（本体2,380円+税）
ISBN978-4-7741-4396-5

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



夜明けに向かって コアダンプ

きたみりゅうじ 著

偉人・変人が語り継がれるIT業界。でも，多くの人はごく平凡に、ごく普通の人々と、日々の仕事に追われています。じゃあそこには何のドラマもないのか？…これがそうでもない。
ささやかだからこそ、「あるある」と共感する体験談が埋まっています。「普通の人々による、様々な好プレーや珍プレー」を、4コマまんがを交えて紹介しよう。そんなWeb連載が一冊の本になりました。

ごくありふれた、だからこそ身近に感じる投稿の数々。是非お楽しみいただければ幸いです。

四六判／176ページ　定価（本体1,380円＋税）
ISBN 978-4-7741-6392-5

図解でよくわかる 改訂4版 ネットワークの重要用語解説

きたみりゅうじ 著

ネットワーク用語集の超定番書の改訂4版です。
本改訂では、すべてのイラストの文字をすべて見直し、読みやすさがさらにアップしました。本書は初級ネットワーク技術者を対象に、キーワードとなる用語を集め、目で見て理解できるように全てをイラストで解説しています。ネットワークの用語集となることはもちろん、重要なネットワーク技術の用語と仕組みを解説しているので、基礎学習書としても活用できます。

A5判／304ページ　定価（本体1,980円＋税）
ISBN 978-4-7741-6324-6

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

Software Design plus

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

プロになるための JavaScript入門
河村 嘉之、川尻 剛 著
定価 2,980円＋税　ISBN 978-4-7741-5438-1

Webサービスのつくり方
和田 裕介 著
定価 2,180円＋税　ISBN 978-4-7741-5407-7

日本一の地図システムの作り方
㈱マピオン、山岸 靖典、谷内 栄樹、本城 博昭、長谷川 行雄、中村 和也、松浦 慎平、佐藤 亜矢子 著
定価 2,580円＋税　ISBN 978-4-7741-5325-4

Androidアプリケーション開発教科書
三苫 健太 著
定価 3,200円＋税　ISBN 978-4-7741-5189-2

養成読本編集部 編
B5判・216ページ
定価 1,980円（本体）＋税
ISBN 978-4-7741-6422-9

養成読本編集部 編
B5判・196ページ
定価 1,980円（本体）＋税
ISBN 978-4-7741-6425-0

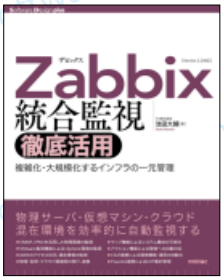
養成読本編集部 編
B5判・184ページ
定価 1,980円（本体）＋税
ISBN 978-4-7741-6424-3

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

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

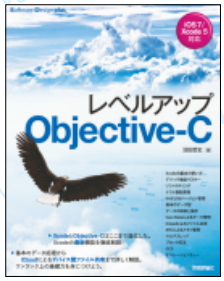
久保田 光則、アシアル㈱ 著
A5判・384ページ
定価 2,880円（本体）＋税
ISBN 978-4-7741-6211-9

乾 正知 著
B5変形判・352ページ
定価 3,200円（本体）＋税
ISBN 978-4-7741-6304-8



TIS㈱ 池田 大輔 著
B5変形判・384ページ
定価 3,500円（本体）＋税
ISBN 978-4-7741-6288-1

大谷 純、阿部 慎一朗、大須賀 稔、北野 太郎、鈴木 教嗣、平賀 一昭 著
㈱リクルートテクノロジーズ、㈱ロンウイット 監修
B5変形判・352ページ
定価 3,600円（本体）＋税
ISBN 978-4-7741-6163-1



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

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

養成読本編集部 編
B5判・224ページ
定価 1,980円（本体）＋税
ISBN 978-4-7741-6377-2

和田 裕介、石田 絢一（uzulla）、すがわら まさのり、斎藤 祐一郎 著
B5判・144ページ
定価 1,880円（本体）＋税
ISBN 978-4-7741-6367-3

菊田 剛 著
B5判・288ページ
定価 2,480円（本体）＋税
ISBN 978-4-7741-6128-0

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

紙面版
A4判・16頁
オールカラー

電脳会議

DENNOUKAIGI

一切無料

新規購読会員受付中!

『電脳会議』は情報の宝庫、世の中の動きに遅れるな!

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

今が旬の情報満載!

新規送付のお申し込みは…

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

電脳会議事務局　検索

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

一切無料!

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

毎号、厳選ブックガイドも付いてくる!!

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

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

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

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

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

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。

DIGITAL GADGET

Volume 185

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

妄想のガジェット、理想のガジェット

妄想と理想の紙一重

妄想の物語が詰まった書籍を多数出版しているクラフト・エヴィング商會の書籍の中に、「シガレット・ムーヴィー」という品物が載っています。それは煙草の1本1本に映画の情景が刷り込まれていて、その煙草を吸うと目の前に映画の情景が浮かび上がるという妄想の品物です。

煙草にはそれぞれ映画のタイトルが小さな文字で書かれており、予告編も何もなく、その煙草に火をつけてみないと、どんな映像が浮かび上がってくるかもわからないのです。また煙草の健康への影響も、その映像次第、という妄想全開の製品です。

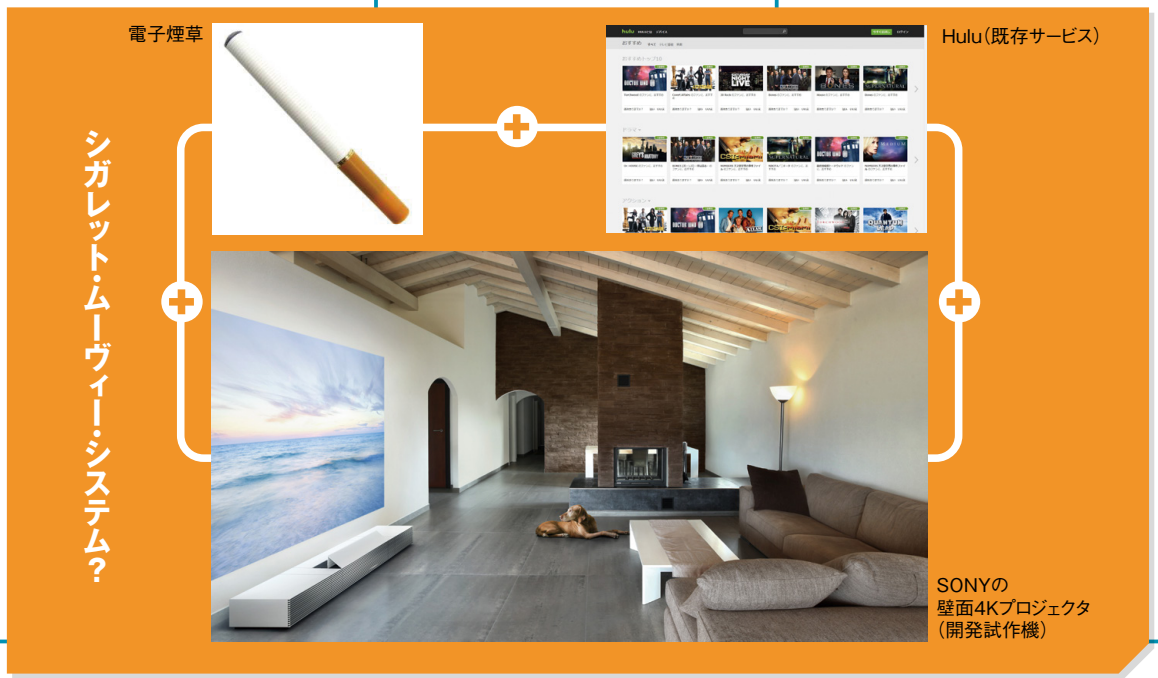
「そんなの無理だろ〜」と笑い飛ばすのは簡単です。けれども、もし何らかの技術によって実現したらどう感じるでしょうか？　100%完全に実現とは言えずとも、少しでも実現する方法はないでしょうか?

たとえば、RFIDタグなどを搭載した電子煙草と、プロジェクタを置いていることを意識しなくて済むような超単焦点の壁面プロジェクタ、ネット経由のオンデマンドの映画配信サービスの組み合わせで、「シガレット・ムーヴィー」の妄想を現実世界で実現できなくはないでしょうか?

単なる妄想を具現化し、アイデアやデザインの宝庫として2005年頃から続いている「Yanko Design」(http://www.yankodesign.com/)というサイトには、数百種類のコンセプトデザインが紹介されています。すぐに商品化できそうなものから、現在の技術では実現できそうもない、妄想全開のデザインもあります。海外のデザインアワードで賞を取ったものもあり、商品化に向けて進んでいるプロジェクトもあります。

妄想の限界と理想の可能性

妄想にもいろいろな種類のものがあります。新製品リリース直前の噂やリー



※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。

>> 妄想のガジェット、理想のガジェット

ク情報を見ながら、もうすぐ発表されるであろう新製品に思いを巡らせるのも楽しい「妄想」です。人間が想像できるものであれば何でも作れそうに思えますが、妄想と理想の間には大きな壁がありそうです。それらを列挙してみましょう。

- 製造限界。その大きさや小ささでは作れない
- 実現しようとなると高額すぎて、コスト的に見合わない
- 物理的、物理現象として不可能
- コンピューティングパワーや処理スピードがまだ足りない
- 音声認識、画像認識や動態認識の技術の不正確さのため必要不十分
- 目的の強度や透明度など、必要な素材がない
- ニッチな領域すぎて商用展開が難しいと考えてしまうもの
- 身近にある不便が不便と感じていないため、その先に進まない
- エネルギー効率やコスト面であまりにも効率が悪い
- 無理な形状。職人芸や金型で製作するのが無理な形状と素材
- 手動やアナログで操作するのが当たり前と思っていた事柄
- 1つ1つが受注生産になってしまうため、実現が難しいもの
- 思いもしなかったような組み合わせであるため、実現していなかったもの
- 商業的に意味をなさないため、実現していなかったもの
- 人手で処理したほうが安価なため、デジタル化が遅れている事柄

なんだか読めば読むほどゲンナリしてくるかもしれませんが、それを逆手に取って、いろいろなものを考えてみましょう。

たとえば、iPhoneを聴診器に変えるスマートフォン用ケース「Steth IO」。部品を組み合わせて自由な機能を持たせることができるスマートフォン「Project Ara」など、たぶん妄想から始まったであろうコンセプトも、最近の技術で実現してしまっているのです。

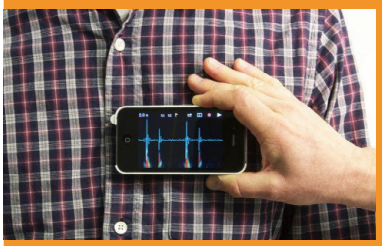
従来は、現実の延長線上に未来がありましたが、これからはコンピュータの中やSF映画の中では実現できている妄想を、どうやって現実世界に持ち出してくれば良いのかを考えていくようになるのかもしれません。

限界点を打ち破る手法

未来に向けた先進的なモノやサービスを考えるのは容易なことではありません。現在のユーザは想定できても、未来のユーザはまだいないからです。そのような未来に向けての企画が求められる場合、“エクストリームユーザ”と呼ばれる極端な特性を持ったユーザ像を想定して考えます。一般的なユーザではなく、相対的な数は少なくとも先進的で極端なユーザを想定するのです。

たとえば、先進的なスマートフォンアプリを作ろうと考えたときは、片時もスマホを手放さないようなヘビーユーザに向けて考えたり、または、いっさいデジタルデバイスを使わないようなユーザに向けて考えたりするのです。

また優秀なエンジニア／プログラマであればあるほど、常識や技術の限界を意識しないで考えることが難しくなります。現在の技術であればこれぐらいまではできる、と限界点を自ら設定してしまうのです。「効率良く処理できるのは、これぐらいのデータ量とこれぐらいの情報」「現在のネットのスピードであれば、これぐらいのことができる」など自分で実装することを想像し、企画の実現のバランスを瞬時に考え、突拍子もない発想を押さえつけてしまうことがあります。

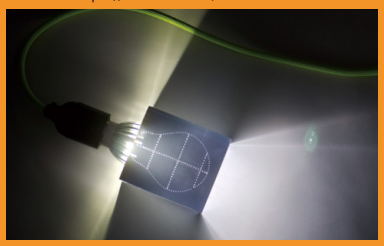


iPhoneを聴診器に変えるスマートフォン用ケース「Steth IO」
http://www.stratoscientific.com/

組み合わせスマートフォン「Project Ara」の部品群
https://dscout.com/ara

Project Araで組み合わせた筐体例

USBで充電できる乾電池（似たアイデアがエコデンという名前で商品化）



3Dプリントされたランプ「Printed Bulbs Light Bulb」

もちろん無理難題な企画を実現するのに苦労したあげく、中途半端なクオリティで実現に至らないことは避けなければいけません。そのようなときに有効なのは、発想を広げるブレインストーミングと、実装の方法を細かく検討するプロダクション会議を分けて、それぞれ別に実施するのです。メンバーは同じでもかまいませんが、目的と発想の観点を変えることで、突飛な発想でも実現する工夫ができたり、技術的要因で、発想を押さえ込まれにくくなります。

さらに、現時点では無理なサービスや企画も、妄想は妄想のままで終わらせずに企画を暖めておけば、近い将来実現できるようになるかもしれません。今から10年前を思い出してみると、今は無理でも5年、10年経てば現実になっている可能性を否定しきれないでしょう。

皆さんが子供の頃、コンピュータに声で尋ねると何でも答えてくれるような漫画やアニメーションがなかったでしょうか？　その当時は未来の出来事でしたが、2014年の今は、音声で検索できたり、音声でスマートフォンを操作したり、知りたいことは何でも検索できたり。

その頃妄想していた未来は、限りなく近づいているように感じられます。SD

クリップとして使えるUSBメモリ

GADGET 1

Know Your Switches

http://www.yankodesign.com/2011/03/02/know-your-switches/

フロアプラン電灯スイッチ

Know Your Switchesはデザイナー Taewon Hwang氏によるコンセプトデザイン。部屋の間取りが複雑な場合、どこの電灯スイッチがどこの電灯に対応するのか、戸惑うことが多くはありませんか？　とくに初めての会議室などですと、「前列」などと文字で書かれていても、実際にはどこが点灯／消灯するのかよくわかりません。Know Your Switchesは、部屋の間取りと同じ図柄のスイッチにすることで、誰もが一目で理解できる見栄えになっています。汎用化が難しく、コストがかかりそうですが、液晶タッチパネル化などで実現してほしいアイデアです。

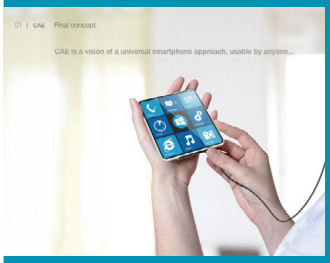
GADGET 2

Blank Calls

http://www.yankodesign.com/2014/02/26/blank-calls/

見ないで操作できるスマートフォン

Leo Marzolf氏デザインによるBlank Callsは、おもに目の不自由な人向けのスマートフォンとして考えられ、画面を見ないで使えるというコンセプトを提示したものです。正方形で、どんな方向で持っていても使え、手触りでアプリを見分けられることを想定しています。形状に独特の切り欠きがあることで、ヘッドホン端子の位置や、持っている方向を素早く把握することができます。文字を大きく表示する機能やコントラストの強い白黒表示にする機能なども考えられています。



GADGET 3

dataSTICKIES

http://www.yankodesign.com/2014/02/21/new-word-of-the-day-datastickies/

付箋紙のようなメモリ

dataSTICKIESはAditi Singh氏、Parag Anand氏による、携帯できる新方式のメモリのコンセプトデザインです。付箋紙のようにメモを書き込んだり、付箋紙を1枚1枚剥がすように、分離して容量を分けて使うことができます。付箋紙と同じく、ノートや手帳に貼り付けることもできます。またUSBポートに差すのではなく、光学式のデータ転送方式を用い、デバイスにペタっと貼るだけでデータを転送してしまうことを考えているもよう。

GADGET 4

Future Control

http://www.yankodesign.com/2014/02/19/future-control/

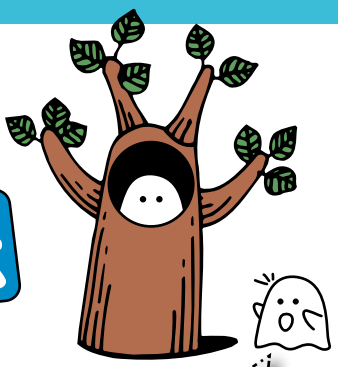
手のひらをタッチパネル化する装置

Dor Tal氏が考えるPredictablesプロジェクトでは、未来の操作方法を提示しています。超小型のピコプロジェクタと認識用のカメラを用いて、手のひらや部屋の壁など、どこでも表示装置とタッチパネルが使えるようにしてしまいます。腕時計のような装置によって、手のひらに投影した映像をスマートフォンのタッチ画面のように操作することで、スマートフォンを直接操作する必要はありません。

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

12

結城浩の再発見の発想法



Hook

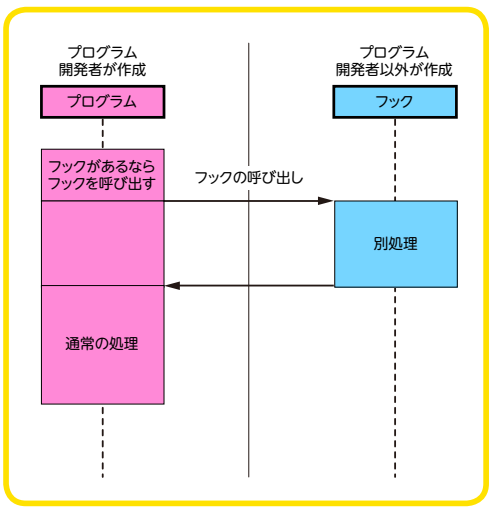
Hook（フック）

フックとは

フック（hook）というのは、プログラムの重要ポイントに「別処理」を紛れ込ませるしくみです。プログラムは、動作中にフックを通過すると、通常の処理から離れてフックに設定された「別処理」に寄り道をします。そして、その「別処理」を終えたらまた通常の処理に戻ります。この様子を**図1**に示します。

フック（鉤）という言葉がなぜこのような意味で用いられているかは知りません。処理の流れが「別処理」に寄り道をする様子が鉤のように曲がっているということなのか、あるいはプログラムの開発者が用意した処理の流れを、別処理を紛れ込ませる人が鉤を使って引き寄せるイメージなのかもしれません。

▼図1　フック



フックの例

フックはプログラミング技術として非常に多く使われています。

簡単な例として、たまたま筆者が先日体験したフックを紹介します。テキストエディタVimを使っていて、「書き込んで終了」というコマンド`:wq`の代わりに`:w:wq`と入力してしまうと`:wq`という変なファイルが作られてしまいます。もともと`:wq`というファイルを作ることは考えにくいので「コロン（`:`）で始まるファイルを作ろうとしたらエラーにする」という処理をしたいとしましょう。でも、Vimのソースコードを書き換えてそのようなちょっとした処理を組み込むことは（不可能ではないですが）たいへん手間がかかることです。

ありがたいことに、VimにはBufWriteCmdというフックがあります（Vimの用語では自動コマンドイベント）。BufWriteCmdは「Vimがバッファ全体をファイルを書き込む前に自動的に呼び出す別処理」にあたり、ユーザが自由に設定することができます。BufWriteCmdをVimの開発者が入れてくれたおかげで、「コロン（`:`）で始まるファイルを作ろうとしたらエラーにする」のようなユーザ固有の処理を入れるこ

とがとても簡単にできます。これはまさにフックの例になります(**図2**)。

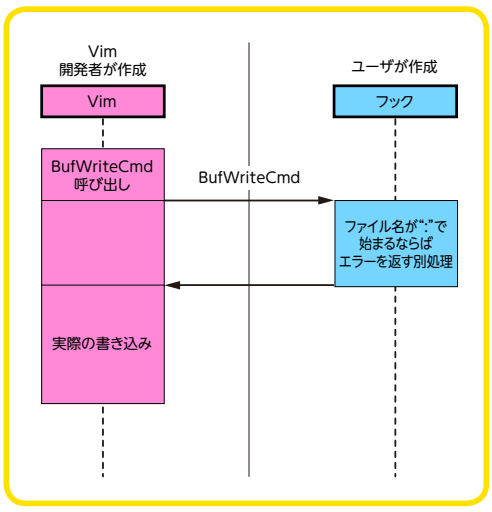
バージョン管理システムGitにもたくさんのフックがあります。コミット時に関連したフックだけでも4個(pre-commit、prepare-commit-msg、commit-msg、post-commit)あります。これはデフォルトのコミットメッセージを準備する前、ユーザがコミットメッセージを編集する前、コミットメッセージを使う前、コミット終了後、という重要ポイントにそれぞれ入れられています。これらのフックを利用することで、Gitのコミット処理の流れをユーザがカスタマイズできるようになります。

フックのメリット

元のプログラムがたとえ書き換え不可能だとしても、フックがあれば、ユーザがプログラムの振る舞いを変更できます。このため、たとえばROM化された組み込みプログラムでもフックは非常によく使われます。適切なフックが用意されているプログラムは機能拡張性に優れていると言えるでしょう。

フックの目的はユーザのカスタマイズだけではありません。開発者がデバッグに用いることもありますし、プログラムの動作テストや、パフォーマンスを測るために用いることもあります。たとえば、プログラムの中で使われているすべての関数の実行前と実行後にフックを入れておけば、プログラムが実行する間にどの関数が何回呼び出されるかを計測することができるでしょう。計測が終わったらフックを外しておけば、製品となるプログラムのパフォーマンスに悪影響は残りません。

▼図2　BufWriteCmd



日常生活におけるフック

さて、私たちの日常生活にもフックに類似するものがあります。

たとえば製品をリリースする場面を考えましょう。リリース手順の中に「チェックリストを見る」という項目を含めておくのはいいことです。失敗事例などを元に、チェックリストを更新しておけばミスを減らせるでしょう。リリース手順はずっと変わらないとしても、「チェックリストを見る」というフックを入れておくことで、細かいカスタマイズができるわけです。

もっと身近な例では「スーパーでの買い物」もあります。スーパーをぐるっと回って買い物するというプロセスの中に「買い物の初めと終わりにメモを確認する」という項目を含めておくのです。「買い物メモを見る」というフックを入れておくことで、「買うべきものをメモに書いておけば買い物に漏れがない」という状況を作ることができるでしょう。

その他にも「毎年必ず行う必要がある作業」を誕生日というイベントのフックとして実装している人は多いかもしれません。

◆ ◆ ◆

あなたの周りを見回して、いつも変化のない一連の作業を探してみましょう。その中に「自分がカスタマイズできる別処理」としてフックを入れることはできるでしょうか。「作業前」と「作業後」にフックを入れたらおもしろいことは起きないでしょうか。

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13回 チェイン・リアクション

湖畔のレストラン

「亮(リャン)よ、中国にはこういうルールがある。兄弟の契りを交わすには、白酒(バイチュウ)を6度飲めと。しかもこいつは超一級品で、最近の38度くらいの軟弱なやつとは違う。古来からの伝統の、50度の奴だ」

工場長のビルはそう言って僕のグラスギリギリまで、白酒を注ぎました。

「カンペー」

グラスを合わせ、白酒をイッキ飲みします。まるで焼けた溶岩が身体の中に入って来るような独特の感覚が僕を襲いました。初めてウィスキーを飲んだときのような、強烈な体験です。

「さあ、6杯目だ。これで明日から量産に入れるぞ」

量産開始の前日。郊外にある湖畔のレストランに僕たちは居ました。蚊取り線香の代わりにヤモリを多数放し飼いにしている店でした。天井のあちこちにヤモリがへばりついています。確かに蚊には刺されませんでした。

ビルは再び白酒をグラスになみなみと注ぎ、しかしそれだけでは満足しないというようにかぶりを振りました。

「こないだ韓国のメーカー衆と飲んだとき、連中はこうしたんだ」

マッチから火を放ち、たちまち白酒はまるでアルコールランプのように……いや、まさにアルコールランプそのものと化して煌々(こうこう)と辺りを照らし始めました。

「これを、飲むッ」

ビルはそう言うと、火のついた白酒をそのままぐいっと丸呑みしました。

「どうだ、リャン、おまえにできるか？」

これが中国の流儀か。

僕は恐る恐る火のついた白酒のグラスをとり、一気に飲もうとしました……が、50度の酒を一気に5杯飲んだ直後です。手元が狂いました。

「キャアアア!!」

写真を撮ろうと構えていた秘書の柿澤彩香が悲鳴を上げます。僕がこぼした白酒で、僕の顔が火だるまになったからです。

慌てておしぼりで拭き取ります。軽く火傷を負いましたが、まあ深刻なほどのものでもありませんでした。円卓では、事の成り行きを工場のスタッフやチェリーさん、そしてUEIの社員が心配そうに見つめていました。

とはいえ、僕は失敗してしまったのです。6杯の白酒を飲むという兄弟の契りの儀式に。

ビルは「いいガッツだったよ。ハッハッハ」と笑いました。冷水でしぼったおしぼりですっかり顔の痛みも引いた僕は、ニヤリと笑ってビルに言いました。

「日本ではな、『郷に入っては郷に従え』という言葉がある。6杯の酒を飲めなかったら、もう

6杯、飲もうじゃないか。今度は火をつけてもしくじらないぞ」
「おまえ、本気か？」
それからさらに6杯に挑戦したところで、僕は記憶をなくしました。

苦闘

深圳（シンセン）での滞在はもう1ヵ月を超えようとしていました。製造が遅れるどころか、始まってすらいなかった、それどころかヨーロッパにコピー品を勝手に売りつけようとしていた、などですっかりこの工場が信用できなくなっていました。

しかし、工場長のビルと親しくすることで、少しでも納期を短縮しようと僕はビルに近づきました。契約書を交わそうにも、英語がわかるのが先日即日でクビになったセールスマネージャーの女性だけで、あとは新入社員のショーンと大手メーカーから引き抜かれて来た工場長のビルだけでした。おかげで先方から渡される契約書の英語はほぼデタラメで、なんどもこちらから修正をかけることになりました。

しかしいつまで経っても量産が始まりません。
「量産が始まるまで、毎日来るぞ」
僕はショーンにそう宣言して本当に毎日工場に行っては、あれがどうなってるんだこれがどうなってるんだと現状の確認を続けました。
「清水は製品ができあがるまで日本に帰るつもりはないらしい」
彼らがようやくそう認識したのは、僕が滞在2週間を数えるころでした。とはいうものの、量産したくてもできない事情が彼らにもありました。部材の調達です。

工場というのは、大量生産には適していますが、生産ラインを作るだけで半日から1日かかってしまうので、できるだけそこまでにはすべての部材をそろえておく必要があります。しかし、圧倒的に不足していたのは、縦横比4：3のタッチパネルでした。当時からすでに16：9が主流になっており、4：3のタッチパネルはほぼデッドストック状態にありました。タッチパネルと液晶は、OGSと呼ばれる技術によって一体化するため、それが来ないことには本体の組み立てには入れません。それでタッチパネルが調達できるまでは、工場は生産ラインを作ることができなかったのです。

僕はタッチパネルの工場や液晶の工場、ボディの工場に何度も足を運び、催促しました。しかし僕たちの雇ったアッセンブリー工場は金払いが悪いという評判で、部材の調達をしようにも、なかなか優先権を渡してくれません。さらにタッチパネルの原料の調達でも手間取っていることが解りました。まるで「ぷよぷよ」の連鎖反応のように、ある工場が滞るとそれに連なる他の工場のスケジュールも遅れていくチェイン・リアクション（連鎖反応）が起きていたのです。

大ベストセラーのAllWinner A10チップも、AllWinner社がすでに製造中止を宣言しており、市場にあるものを買い集めるしかないということもわかっていました。僕はじりじりと時間だけが過ぎて行く日々を深圳で過ごしました。

量産の目処がつき、現地にUEIの社員を呼び寄せ、ソフトウェアの最終確認ができる、という場面までやってくるのに、さらに3週間が必要でした。ビルと湖畔のレストランで白酒を12杯飲んだ日は、まさに部材の調達が完了し、量産に入れるというその直前だったのです。

ソフトウェア開発チームとの溝

日本から呼び寄せたのは、大手メーカーからenchantMOONのプロジェクトに参加したくて転職してきた日高正博と、僕の秘書をしていた柿澤でした。柿澤は僕が不在の間、デバッグを手伝い続け、いつのまにかenchantMOONのテスト項目に関しては誰よりも詳しくなった、と現場から推薦を受けて送り込まれてきました。

僕たちは量産されてくるハードウェアを使って、最終版のソフトウェアを現地でテストするつもりでいました。しかし、彼らが持って来た

「最終版」のソフトウェアは、再現性100％でチュートリアルの途中でハングアップしたり、もともと操作を知っていなければ進めないようになっていたり、あり得ないところで止まったりと、何かの間違いで古いバージョンを持って来たのではないかと思うほどにデグレードしていました。

というのも、僕が深圳に来る前までに確認していたソフトは遥かに高速に動作していて、おかしな挙動もほとんどなかったからです。日本で散々テストを重ねてきた柿澤でさえも「この挙動はちょっとおかしいですね」と言い出すので、てっきり古いバージョンを持って来たのかと思いました。しかし、何度確認しても、それは日本のソフトウェア開発チームが「最終版」として提出してきた、ホンモノのバージョンだったのです。

僕は滞在中ずっと、ソフトウェアが2ヵ月前よりは進化していると信じてひたすらハードウェアの製造を見ていたのですが、僕が見ていない間にソフトウェアはどんどん悪い方向に仕上がっているようでした。なによりチュートリアルの途中で確実に落ちるものを最終版として提出されたこと自体が僕をひどく失望させました。

これまで、ずっと定期的な報告は受けていました。深圳では通信回線が非常に細く、とくに国外との通信は中国政府が設置したグレート・ファイアー・ウォールによって厳しく制限されていました。そのためテレビ会議に参加するためにわざわざ通信環境がマシな香港まで出向き、開発の進捗を聞いていたのですが、その場ではいい報告しか受けていませんでした。僕はよもやこの状態で開発チームが「良し」と考えているとは夢にも思わなかったのです。それどころか、「いま開発チームは非常にいい雰囲気ですよ。協力的でお互いの立場を尊重し合う空気が流れています」という報告すら、現場のプログラマから得ていたのです。

思えばこの時点でイヤな予感を感じてはいました。どうして製造が2ヵ月も遅れているハードウェアを前に、開発チームが穏やかな雰囲気でいられるのでしょうか。一見、穏やかに見える現場は、実は水面下ではドロドロした闇を抱えているものです。その結果が表出したものが、このいみじくも「最終版」と名付けられたMOON Phaseでした。それはこのチームが考える品質やチーム体制の限界を示していました。

苦渋の決断

結局、量産は無事開始されたものの、僕はこのままのソフトウェアを最終版としては到底受け入れられない、という結論を出さざるを得ませんでした。そこで量産の目処はついたものの、発売日を約2週間後の「7月7日」と決めました。苦し紛れの苦肉の策でしたが、1週間でこの最終版をなんとかギリギリ「製品」の体裁まで持って行けるのではないか、という読みからでした。

急ぎ東京に戻り、チームを再編し、「製品版」を整備する必要性を訴えました。

「チュートリアルの途中で確実に落ちる。なぜこれを君たちが“最終版”と呼んだのか。僕にはまったく、理解も想像もできない。確実に言えるのは、君たちの基準が完全に狂ってるということだけだ。チュートリアルの途中で確実に落ちるものは製品とは到底呼べない。最初のバージョンからはチュートリアルは削除する」

「せっかく作ったのに削除するんですか？」

「関係ない。ちゃんと動かないものを売るわけにはいかない。チュートリアルはアップデート版で入れていく。まずは最低限の部分がきちんと動くものを作る必要がある。また、メインプログラマをベテランの布留川英一に変える。コードの主導権を布留川くんに完全に移管するように」

布留川英一は僕とは10年来のコンビを組んできた本当の大ベテランでした。ほかの案件で引っ張りだこだったため、enchantMOONのプロジェクトへかかわってくるのは遅れたのですが、もはやこの土壇場では布留川に頼るしかありませんでした。

「とはいえ、今のコードを完全に捨ててゼロから書いたらとても2週間じゃ間に合わない」

布留川は言い、極力今のコードを活かしつつもちゃんとバグを取るために、コードの道先案内人としてこのコードを書いた人間は必要だと主張します。僕はその主張を認め、言いました。「とにかく、最低限の機能がまずきちんとワークするものを作る。そこから時間をかけて、きちんとひとつひとつ最適化していく。製品の発売はゴールではない。我々にとって、それは始まりなのだ」

磯玲子と上瀧英郎は、ソフトウェアの開発とデバッグをギリギリまで引き延ばすため、ROM焼きを中国ではなく成田で行う手はずを急ぎ整えました。新入社員の片境泰聡は、英語が得意でタフ、という理由だけで僕が帰国した翌日から中国の工場に張り付き、プレッシャーを与え続けます。

そうしてなんとか出荷に間に合わせたROMでしたが、僕は不満でした。しかしユーザとなる皆様にした約束、ディストリビュータを買って出てくださった方々とした約束を果たすためには、これは苦渋の決断でした。アスキーストアでは、発売日が1週間延びるたびに苦情の電話に対応してくださっていたそうです。これ以上出荷を遅らせるわけにはいきませんでした。

少しでも良いものを、少しでも多くの方へ、という思いを込めて、アップデートは毎週行いました。バージョン2.3で劇的に描画速度が改善されたのは、布留川のチューニングによる描画エンジンが初めて実装されたからです。これはバグを回避するために直前にした施策が、逆に最初期バージョンの動作速度を低下させていたためでした。

週ごとのアップデートは開発チームにとって巨大な負担となりました。デバッグ期間を想定すると、開発期間は2日か3日しかとることができません。それでも懸命なアップデートを続けました。一度削除したチュートリアルはすぐに復活し、第2便出荷には間に合いました。それでもすべての予約者の方々に行き渡るころには、もう中秋の名月を迎えてしまいました。そのころにはアップデートは月に1回程度となり、僕はひとまず開発の現場を離れてenchantMOONを活用する方法について知見を深めるため、実際に大学で授業を行うことにしました。

文科系の女子大生にプログラミングを教える

成蹊大学の経済学部でプログラミングの授業が始まりました。坂井直樹先生が担当している授業で、そのうち1コマで僕がプログラミングを教えることになっていたのです。

これはとても得難い経験となりました。経済学部の授業だったのですが、その過半数は女子でした。そして彼女たちが、まったく見たことも聞いたこともないプログラミングというものに触れると、常に歓声を上げたのです。

「すごい！　こんなことができるんだ！」

その姿を見た僕は衝撃を受けました。僕自身、大学をはじめ、さまざまな場所でプログラミングを教えた経験があります。しかしこれほどまでストレートにポジティブな反応が返ってくるような授業を、今までただの一度もしたことがなかったのです。

そして同時に僕は確信しました。——プログラミングは、誰にとっても楽しいのだ——と。

経済学部で、プログラミングとは一生縁のなさそうな女子大生たちが、嬌声を上げながらHTMLやJavaScriptのコーディングを楽しむ光景はとても奇妙で、しかし僕に大きな勇気を与えてくれるものでした。

彼女たちにenchantMOONを渡すと、大喜びで使い始めました。僕は次回、出張による休講が決まっていたので、その間にenchantMOONでなにか作品を作ってごらん、と言うと、水を得た魚のように彼女たちは目を輝かせました。

そしてこのことが、enchantMOONが決定的に前進する大きなきっかけを僕たちに与えてくれることになったのです。SD

コレクターが独断で選ぶ!

偏愛キーボード図鑑

株式会社 創夢
濱野 聖人 HAMANO Kiyoto
khiker.mail@gmail.com
Twitter：@khiker

第13回

小型メカニカルキーボード

BLACK PAWN & Majestouch MINILA

写真1　BLACK PAWN

はじめに

今回はHappy Hacking Keyboard(以下、HHKB)のような形状の小型メカニカルキーボードを紹介します。小型メカニカルキーボードは、近年になっていくつか製品が発売されています。海外で販売されている製品が多いですが、日本でもいくつか販売されています。筆者が所持しているうち、日本でも比較的簡単に手に入るBLACK PAWN(**写真1**)、Majestouch MINILA、Majestouch MINILA Airを紹介します。

小型メカニカルキーボード

小型メカニカルキーボードは、おおむね次のような特徴を持ちます。

- テンキーレスキーボードよりも小さいHHKBのような形状
- Cherryのメカニカルスイッチを採用
- DIPスイッチでの挙動変更

HHKBのような形状で、ファンクションキーのないキーボードが多いです。その代替として Fn というキーが搭載されており、 Fn を押しながら 1 を押すと F1 を押したのと同じ動作をします。もちろん、KBT Race 75%[注1](**写真2**)のようにファンクションキーも搭載し、テンキーレスキーボードの隙間を詰めたような形状の小型キーボードも存在します。

ほとんどの製品がCherryのメカニカルスイッチを採用しており、たいていは黒軸、赤軸、青軸、茶軸とスイッチの異なる4バージョンが販売されています。黒軸と赤軸の特徴は、線形に重さが増す軸で黒軸のほうが重いです。青軸と茶軸の特徴は、キーを押したと認識する直前で荷重が強く、青軸にはクリック音があります。これらの中から好みのタッチを自分で選べます。Matiasのmini Quiet Pro[注2]のようなCherry以外のメカニカルスイッチを採用している小型のメカニカルキーボードも、種類はかなり少ないですが存在します。

DIPスイッチを搭載しているキーボードが多く、細かく挙動が変えられます。 Caps Lock と左 Ctrl を入れ替え

写真2　ファンクションキー搭載の小型キーボード
KBT Race 75%

注1）KBT Raceは一時期日本でも販売していました。今でも探せばあるかもしれません。
注2）大きめの量販店で入手可能です。

られるキーボードが多いです。

BLACK PAWN

BLACK PAWNはセンチュリー㈱が販売するメカニカルキーボードで(**写真1**)、日本語配列のUSBキーボードです。Cherryスイッチを採用し、黒軸、赤軸、青軸、茶軸それぞれのバージョンがあります。キーボードのキーひとつひとつにバックライトLEDが搭載されており、Fn＋vで点灯できるのが大きな特徴です。

DIPスイッチが搭載されており、Caps Lockと左Ctrlを入れ替えられます。

筐体はメタルでおよそ600gあり、ずっしりと重く、キーボードが安定します。センチュリーの直販サイトやAmazon.co.jpで販売されており、価格は約12,000円です。

Mejestouch MINILA

Majestouchシリーズで有名なダイヤテック㈱の小型メカニカルキーボードです(**写真3**)。USB接続で、日本語配列と英字配列が存在します。Cherryスイッチを採用しており、黒軸、赤軸、青軸、茶軸それぞれのバージョンがあります。

スペースバーの両隣にFnがあり、親指でFnを押せるようになっています。Fnとホームポジション周辺のキーを組み合わせることで上下左右の矢印キーやBack Spaceを入力できるようになっています。そのため、ホームポジションを崩さずに矢印キーやBack Spaceを入力できます。

DIPスイッチを搭載し、Caps Lockと左Ctrlの入れ替えや、Fnをスペースに変更することが可能です(**写真4、5**)。

写真3　Majestouch MINILA（日本語配列）

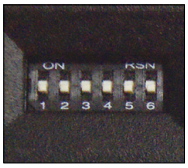


写真4　DIPスイッチ

	JP・68key	US・67key／KR・68key	UK・68key
①	Winキー、Appキーを無効化	Invalidate the Win key and the App key	
②	CapsLockとCtrlの入替	Change CapsLock key and Ctrl key	
③	左Fnキーをスペースに変更	Change the left Fn key into the Spacebar	
④	右Fnキーをスペースに変更	Change the right Fn key into the Spacebar	
⑤	EscとE/Jの入替 Change Esc and [E/J]	Escと[｀~]の入替 Change Esc and [｀~]	Escと[¦¬]の入替 Change Esc and [¦¬]
⑥	右ShiftとDelの入替 Change the right Shift and Del	[←(Backspace)]と[¦\]の入替 Change [←] and [¦\]	

写真5　DIPスイッチの説明

重量は約700gと重くキーボードが安定します。

写真6　Majestouch MINILA Air（英字配列）

価格は約12,000円で、ダイヤテック・オンラインショップやAmazon.co.jpで購入できます。

Mejestouch MINILA Air

MINILAのBluetoothバージョンです(**写真6**)。MINILAと基本的に同じで、日本語配列と英字配列の両方が存在します。Bluetoothの小型メカニカルキーボードとなるとこれだけかもしれません。

Bluetoothキーボードですと、USB経由でバッテリーを充電して使う方式もありますが、これはバッテリーではなく、単三電池2本で動作します。USBポートは存在しないため、USBキーボードとしては使えません。

値段は約14,000円で、MINILA同様、ダイヤテック・オンラインショップやAmazon.co.jpで購入できます。

◆ ◆ ◆

今回は、小型メカニカルキーボードの中でも日本で入手可能な日本語キーボードを中心に紹介しました。英字キーボードであれば、海外ではほかにKBT RaceやKBT PURE、KBC POKER2などが販売されています[注3]。小型のキーボードをお探しであれば、HHKBのProfessionalよりも安く、種類も多いので、探してみると自分にあったものが見つかるやもしれません。SD

注3）PUREやPOKER2は、http://mechanicalkeyboards.com/で購入できます。

はんだづけカフェなう

ステッパーをはじめよう（後編）

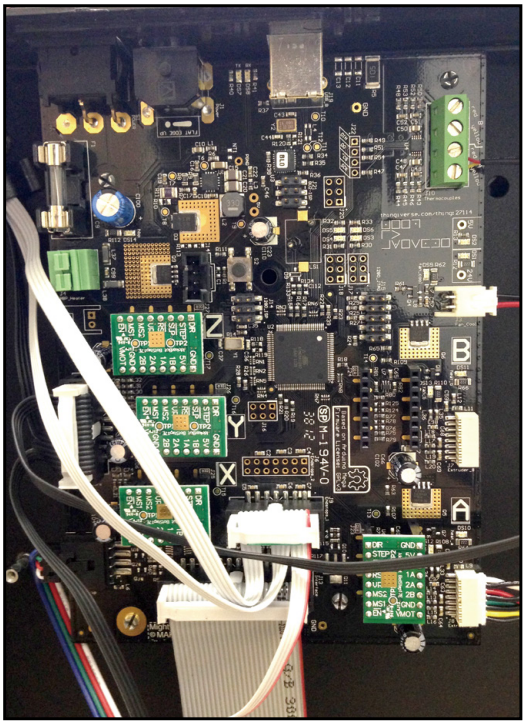
text：坪井 義浩　TSUBOI Yoshihiro　ytsuboi@gmail.com　@ytsuboi
協力：㈱スイッチサイエンス　http://www.switch-science.com/

おさらい

3回続けて紹介してきたステッピングモーター（ステッパー）を動かしてみるという記事ですが、やっと3回目、後編にたどり着きました。

前編ではステッパーのしくみと、トランジスタアレイを使ってGPIO 4本でステッピングモーターを動かすということをやってみました。中編では、I²Cという信号線2本を使う通信方法で接続できる、インテリジェントなPCA9629Aというチップを使ってステッパーを動かしてみました。

▼写真1　Replicator 2の制御基板



今回は、アメリカのAllegro MicroSystemsという会社のA4988というステッパーモータードライバを使ってステッパーを動かしてみたいと思います。

A4988

A4988というチップは、3Dプリンタのステッパーを駆動（ドライブ）するのに多用されています。とりわけ、Pololuというアメリカのロボットやエレクトロニクスのキットを開発・販売している会社の“A4988 Stepper Motor Driver Carrier”という製品がデファクトスタンダードになっており、このボードや互換品が3Dプリンタの制御基板に搭載されているのをよく見かけます。筆者のRepRapでも、このドライバを使っています。

MakerbotのReplicator 2を修理したときにも、このボードと互換のステッパーモータードライバーが搭載されているのを見かけました。写真1の緑色の基板がドライバです。Replicator 2では、X軸、Y軸、Z軸と、エクストルーダー（樹脂を送り出す機構）の4つのステッパーが搭載されていますので、ドライバも4つ搭載されています。

前回の最後でも触れたように、A4988はパワー段（トランジスタアレイなど、大きめの電流を扱うための回路）を内蔵しているのが特徴の1つです。またA4988は、マイクロステップという励磁（れいじ）モードを扱うことができます。

マイクロステップ

前回（中編）では励磁モードを紹介しました。

一相励磁や二相励磁といったフルステップに加えて、一-二相励磁（ハーフステップドライブ）にも触れました。これまでの方法では、電磁石それぞれの磁力が同じだという前提でした。二相励磁というのは隣り合う電磁石が引っ張る力（磁力）が均等だから、モーターの軸は2つの電磁石の中間に位置していました。

ここで隣り合う電磁石の磁力を均等ではなく、7:1の割合[注1]にするとどうなるでしょうか。軸は磁力の強さに応じた位置を取ることになります。次に6：2、5：3と変えていくと、二相励磁（ハーフステップ）の4：4までの間にステップを3つ追加できます。これがマイクロステップと呼ばれる励磁モードの原理です（図1）。

実際には、マイクロステップには、1/4、1/8、1/16など、$1/2^n$のステップをよく見かけます。

電磁石で生じる磁力は電流に比例しますので、電磁石に流す電流の大きさを変えれば、磁力も変わります。つまり、マイクロステップを使用できるステッパーモータードライバには、電磁石のON/OFFだけでなく、電磁石に流す電流の大きさをコントロールする機能が付いているということになります。

A4988基板

A4988というチップは5mm角のとても小さいものです。そこで、ここでは先ほど紹介したPololuのA4988基板を使ってステッパーを動かしてみたいと思います。A4988にモーターを動かす命令を出すために必要な信号線は、I²Cと同様に2本です（といっても、I²Cのように数珠つなぎにすることはできません）。

信号の一方は、STEPと言うもので、回転させるために使う信号です。High（オン）にすると1ステップ進むという命令を伝えることができます。High（オン）とLow（オフ）を高速に切り替えれば、それに応じた速度でA4988はステッパーを回そうとします。

信号のもう一方はDIRで、ステッパーの回転方向を指示するものです。High（オン）とLow（オフ）で、ステッパーの回転方向が変わります。

また、ステッパーを動かすにはマイコンよりも大きな電流を流す必要があります。この電源としてACアダプタなどを用意するのですが、A4988基板には、ステッパーを動かすための電源を接続する端子もあります。

ほかに、基板にはMS1～MS3という端子があり、これを使って励磁モードやマイクロステップの細かさを切り替えることができるようになっています。

接続してみる

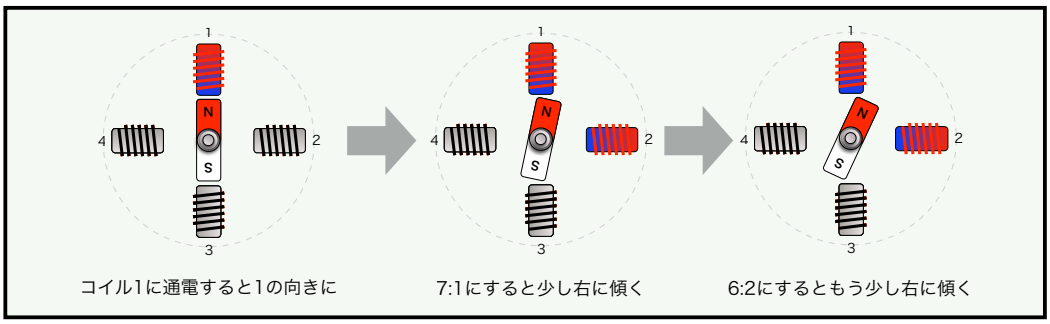
今回もmbedを使って、A4988に信号を送ってみることにしました。

A4988基板はスイッチサイエンスでも販売しています[注2]。今回は、せっかくパワー段のあ

注1） 実際には、7：1の回転角にベクトルが向くように磁力（電流）を計算して流します。

注2） http://ssci.to/582/

▼図1　マイクロステップ

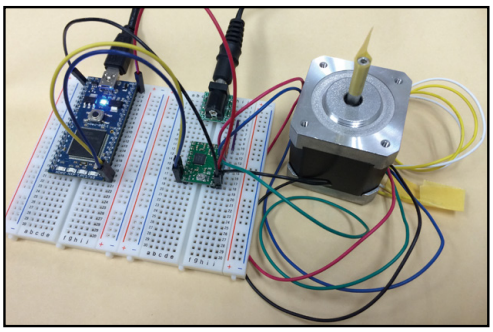


るA4988を使用しているので、筆者の手元にあった比較的大きめのステッパー、SY42STH47-1206Aを使ってみました。Pololuが年末のセールをやっていたときに買っておいたものです[注3]。このステッパーは、ユニポーラーとしても、バイポーラーとしても使うことができます。なおステッパーは、秋月電子通商でも比較的手頃な価格で販売されています。

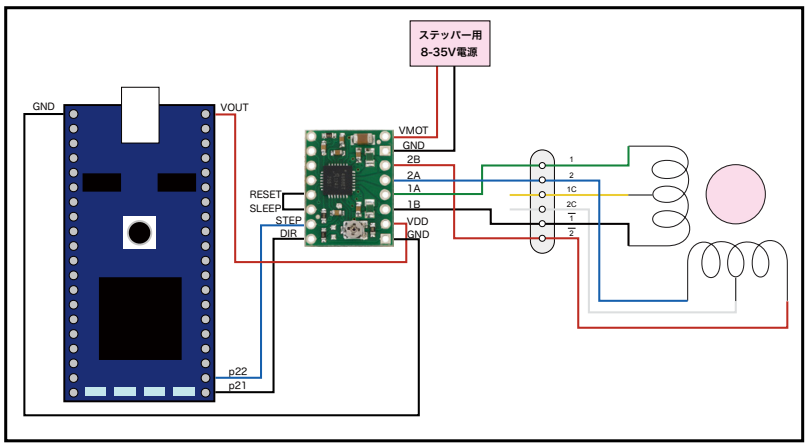
配線図は**図2**のとおりです。ステッパーの線の色は、メーカーなどによって異なります。ほかのステッパーを使うときには、製品の説明を見て、配線図の線の色を適宜読み替えするように注意してください。A4988のステッパーに接続する端子は4ピンですので、バイポーラーとして接続しました。このため、電磁石の真ん中にある線は使っていません。

注3) http://www.pololu.com/product/1200

▼写真2 実験回路



▼図2 配線図



A4988基板には、ステッパーを動かすために使う8～35Vの電源を供給する必要があります。ちょうど筆者の手元には12V 2AのACアダプタ[注4]がありましたので、これと、秋月電子通商の「ブレッドボード用DCジャックDIP化キット[注5]」を使ってブレッドボードの上で回路を組んでみました(**写真2**)。

なお、MS1～3には何も接続していませんので、マイクロステップでは駆動していません。

ステッパーの線をブレッドボードに挿すために、線の端をはんだメッキ(電線の被覆を剥いた部分にはんだを付けて固めること)しました。

本稿では、あまりはんだづけをしなくて済むようにしたいのですが、いっさいはんだづけをせずに実験回路を組んでみるというのは困難です。A4988基板も販売されている状態ではピンヘッダーが付いていませんので、はんだづけをする必要があります。

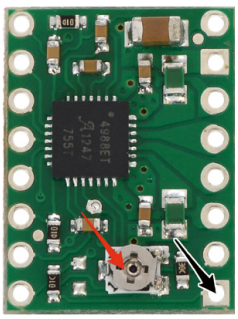
ステッパーの軸には、回転がわかりやすいように、テープを旗のように貼り付けてあります。本当はもう少しおもしろいものを回したほうが良いのでしょうが……。

A4988基板の調整

A4988基板を使うにあたって、まずステッパーに流す電流の上限(Current Limit)を設定

注4) http://akizukidenshi.com/catalog/g/gM-06239/
注5) http://akizukidenshi.com/catalog/g/gK-05148/

▼写真3 テスターを当てる場所



しておきましょう。あまり電流を流し過ぎると、ステッパーもA4988も熱くなり、故障の原因になってしまいます。一方で、流す電流の上限が小さ過ぎると、ステッパーが性能を発揮できなくなり、脱調（だっちょう）しやすくなってしまいます。電流の上限は、計算式「*Current Limit = VREF × 2.5*」で求められます。

このVREF（リファレンス電圧）というのは、**写真3**の部分にテスターを当てることで測定できます。赤い矢印にテスターのプラスを、黒い矢印にテスターのマイナスを当てて電圧を測ります。

購入時の状態では、約0.4Vでしたので、ステッパーのコイルには、最大約1A流れることになります。筆者の手元にあったステッパーの定格電流は相あたり1.2Aと書いてあったので、1Aで良しとしました。定格が0.5Aでしたら0.2Vになるように、先ほどテスターを当てた部品（半固定抵抗といいます）の＋になっている部分をドライバでそっと回して調整します。

プログラム

mbedのプログラムは**リスト1**のとおりです。STEPに接続されているピンをHigh（ON）とLow（OFF）に切り替え、STEPに信号を送っています。200ステップ分の信号を送ったあと、DIRの信号を切り替え、再度200ステップ分の信号を送るというものです。

こうすると、1周200ステップのステッパーを使っていれば、1周回ったあと、反対方向に1周回るということを繰り返すはずです。

ステッパーが回る速さは、wait()内の値によって変わります。wait()内の値を小さくすれば、STEP信号の出る間隔が短くなりますので、ステッパーが速く回転することになります。

前回紹介したPCA9629Aで台形加速を行うときには、動作時間と初速、ステップ数を用意するだけで台形制御を行うことが可能でした。しかし、A4988を使う場合には、前編のトランジスタアレイを直接コントロールするときと同様に、マイコン側で加減速を計算してステップの間隔を求めて制御する必要があります。

まとめ

ステッパーは、デジタル制御で手軽に狙った角度（ステップ）だけ動かせるので、プリンタなどの機器に多用されています。しかし、動作させるためには、マイコンが直接扱うことのできる電流よりも多くの電流を必要とします。このためにトランジスタアレイや今回のようなチップを使って、ステッパーの中にある電磁石に電気を流したり止めたりしてきました。

自分が書いたコードで、何か物理的な動きをさせられるのがマイコンのおもしろいところです。mbedはTCP/IPも扱うことができますので、ネットワーク越しにステッパーをコントロールしたりすることもできるでしょう[注6]。SD

注6）大きな電流を扱う装置は火災の危険も高くなります。そのような実験は安全対策や目の届くところでするようにしましょう。

▼リスト1　mbedのプログラム

```
#include "mbed.h"

DigitalOut A4988STEP(p22);
DigitalOut A4988DIR(p21);
int a=0;

int main() {
    A4988DIR = 0;
    A4988STEP = 0;

    while(1) {
        if (a <  200) {
            a++;
            A4988STEP = 1;
            wait(0.008);
            A4988STEP = 0;
            wait(0.008);
        } else {
            A4988DIR = 1;
            a++;
            A4988STEP = 1;
            wait(0.008);
            A4988STEP = 0;
            wait(0.008);

            if (a>400) {
                a = 0;
                A4988DIR = 0;
            }
        }
    }
}
```

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5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。

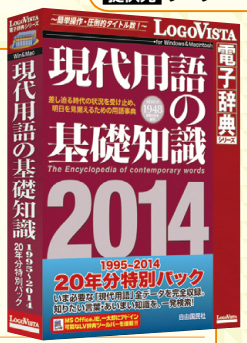
ご記入いただいた個人情報は、プレゼントの抽選および発送以外の目的で使用することはありません。アンケートのご回答については誌面作りのために集計いたしますが、それ以外の目的ではいっさい使用いたしません。ご入力いただいた個人情報は、作業終了後に当社が責任を持って破棄いたします。なお掲載したプレゼントはモニター製品として提供になる場合があり、当選者には簡単なレポートをお願いしております（詳細は当選者に直接ご連絡いたします）。

01 1名 サクス・タブレット7 (L-Sax Tablet 7inch)

Linux マシン専門ブランド「LinuxMania」から初めてのタブレットが登場しました。CPU は Intel Atom Z2460 1.6GHz 1 コア、メモリは 1GB、OS は Android 4.04 を搭載。記憶容量は 16GB です。IEEE 802.11b/g/n 準拠の無線 LAN を使用できるほか、端末本体には 0.3M ピクセルのフロントカメラと 2M ピクセル背面カメラも備わっています。

提供元 ソーソー **URL** http://www.linuxmania.jp/

02 1名 現代用語の基礎知識 1995～2014 20年分特別パック



『現代用語の基礎知識』の 20 年分のデータを収録した特別パック。インクリメンタルサーチに対応しているほか、MS Office や一太郎、IE へ辞典検索機能をアドインできます。Windows 8.1/8/7/Vista/XP（XP は 32bit のみ）と Mac OS X 10.6.8 以上で動作可能。

提供元 ロゴヴィスタ **URL** http://www.logovista.co.jp

03 1名 置くだけスピーカー withバッテリー

本製品の上にスマートフォンを置くだけでスピーカーとして使えます。Wi-Fi などを利用しないので接続設定も必要なし。充電式電池の充電もでき、本体とスマートフォンを USB でつなげば、電池からスマートフォンへ充電できます。 ※本製品にスマートフォンは付属しません。

提供元 サンコー **URL** http://www.thanko.jp

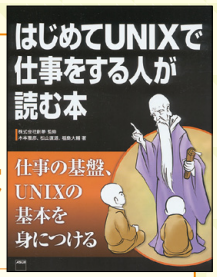
04 1名 iOS アプリテスト自動化入門

長谷川 孝二 著／
A5 判、256 ページ／
ISBN = 978-4-7980-4089-9

コンポーネントを依存関係から独立させテストする方法、ツールを使ってエンドツーエンドのテストを自動化する方法など、iOS 開発者が悩むテスト工程の知識とテクニックを詰め込んだ 1 冊。

提供元 秀和システム **URL** http://www.shuwasystem.co.jp/

05 2名 はじめて UNIX で仕事をする人が読む本



木本 雅彦、松山 直道、稲島 大輔 著、㈱創夢 監修／
B5 変形判、248 ページ／
ISBN = 978-4-04-891392-8

UNIX の教育を受けずに IT 業界に就職した人向けに、仕事で UNIX 系 OS を使うための最低限の基礎知識をまとめました。UNIX の基本操作、プログラミング環境、ネットワーク技術まで幅広く学べます。

提供元 KADOKAWA **URL** http://ascii.asciimw.jp/

06 2名 アドテクノロジープロフェッショナル養成読本

養成読本編集部 編／
B5 判、128 ページ／
ISBN = 978-4-7741-6429-8

いち早くビッグデータを活用してきた広告業界。そのアドテク業界をリードする執筆陣が、オーディエンスデータの理解、データ分析を中心とした組織、広告の配信技術などについて解説します。

提供元 技術評論社 **URL** http://gihyo.jp/

07 2名 データベースの限界性能を引き出す技術

山崎 泰史、武吉 佑祐 著／
A5 判、224 ページ／
ISBN = 978-4-7741-6364-2

ストレージ、CPU、ネットワークなどあらゆる点から「なぜ RDBMS は遅くなるのか」「どうすれば性能を最大限引き出せるのか」を徹底解説。Oracle Exadata などの最新動向も取り上げます。

提供元 技術評論社 **URL** http://gihyo.jp/

新人歓迎 楽しくネットワークを学びませんか？
ITの基礎知識をインストール！

第1特集

ネットワーク技術超入門

ポートとソケットがわかればTCP/IPネットワーク技術がわかる

—— UNIXのしくみから紐解くインターネットのしくみ

日々インターネット上で、ネットワークとコンピュータ技術情報を発信し続けている、あきみち さんと『小悪魔女子大生のサーバエンジニア日記』のaicoさんのコラボレーションによる、新人さんに向けて贈るインターネットのしくみ講座です。

本特集では、おそらくユーザ時代ではなじみがなかったであろうポートとソケットの概念を通して、インターネットのしくみを図解しながら楽しく解説します。ときどきプログラミングの話も出(ひらめ)てきますが、自分が直観でわかるところから、読み進めてください。きっと何か閃くはず！

Writer あきみち(ギークなページhttp://www.geekpage.jp/ Twitter@geekpage)
イラストレーション aico(『小悪魔女子大生のサーバエンジニア日記』 http://co-akuma.directorz.jp/blog/)

ポートとソケットがわかればTCP/IPもネットワーク技術がわかる

――UNIXのしくみから紐解くインターネットのしくみ

何気なくインターネットを利用していると、手元の機器の中に世界中のコンテンツがそのまま入っているかのような錯覚に陥りがちです。たとえば、スマホをちょちょいといじれば、世界の裏側にあるコンテンツだって見られますし、国境を越えてメールのやりとりもできてしまいます。インターネットを使って「簡単につながることができる」わけですが、通信プログラムを書くのも同様です。今回は、そんな便利なインターネットを利用した通信プログラムを理解するために、「ソケットとポート」に着目してTCP/IPを解説します[注1]。

Writer あきみち http://www.geekpage.jp/ Twitter@geekpage
イラストレーション aico(『小悪魔女子大生のサーバエンジニア日記』http://co-akuma.directorz.jp/blog/)

Part 1 UNIXネットワークのしくみ

ソケットとポートからネットワークを眺める

20年前は通信プログラムを書くのであればソケットを使うという選択が一般的でしたが、昨今は非常に便利なツールやライブラリが用意されているので、直接ソケットを使わなくても良いことが多くなりました。そのため、そもそも「ソケット」や「ポート」というものが存在していることを知らずにプログラムを書き続けている技術者に出会うこともあります。そのような現状をふまえ、「ソケット(socket)」と「ポート(port)」という切り口でインターネットを利用した通信を紹介したら面白いのではないかという考え方で本稿を構成しています。なお、本稿はUNIX的な視点を前提に解説します。オペレーティングシステム(以降、OS)などによる差異はあるかと思いますが、「ソケット」と「ポート」という視点で簡単にまとめます。

通信の出入り口となる「ソケット」

ここまで「ソケット」という単語を説明せずに使ってしまっていますが、socketという英単語の日本語訳は「受け口」「コンセント」「差し込み口」などです。身近なものでは、電球を差し込む口や、CPUやチップなどを差し込む部位が「ソケット」と呼ばれています。

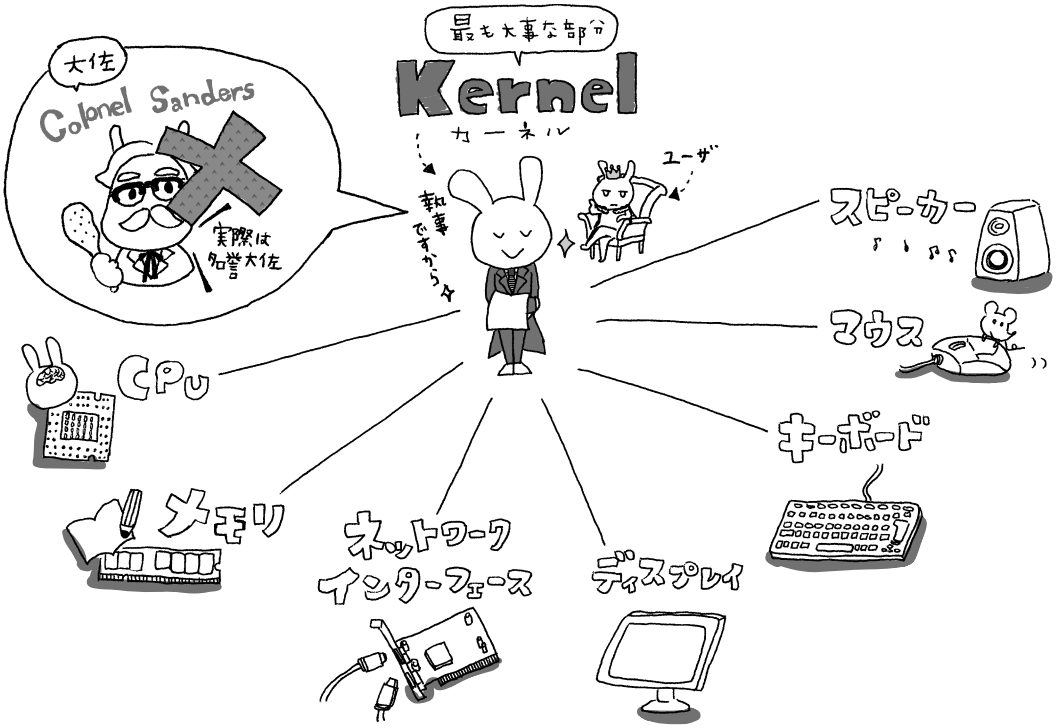
英単語としては、何かをつなぎ込むためのものが「ソケット」ですが、プログラミングにおける「ソケット」は、通信を行うための「口」です。ユーザは、ソケットにデータを書き込んだり、ソケットからデータを読み出したりすることでインターネットを利用した通信を行います。最初にソケットとは何かを紹介しますが、その前にOS側の通信機能の肝となる「カーネル」から説明しましょう。

PCの中の執事「カーネル」

皆さんが使っているPCは、OSと呼ばれる

注1) 本稿は、UNIX系のOSを念頭に執筆しています。

▼ 図1　執事のようなカーネル（カーネル・サンダースではないよ!）



ソフトウェアが稼働しています。OSという単語は、ハードウェアを制御しつつコンピュータそのものを動作させるための基本ソフトウェアである「カーネル」を示す場合と、管理用ソフトウェアなどの各種ソフトウェアを含めたパッケージ全体を示すことがあります。

カーネルは、ユーザのためにハードウェアを制御してくれる執事のようなものです（**図1**）。CPU、メモリ、通信用のネットワークインターフェース、ディスプレイ、キーボード、マウス、スピーカーなど、さまざまなハードウェアの制御などをしています。

たとえば、ユーザがPCを使うとき、情報をファイルに保存したりしますが、物理的なハードディスクそのものが「ファイル」という概念を直接扱っているわけではありません。カーネルがユーザにとってわかりやすいように、「ファイル」という概念を構成してくれているのです。このように、ユーザが使いやすいようにハードウェアの機能などを「抽象化[注2]」するのも、カーネルの重要な役割です。

「カーネル」というと、ケンタッキーフライドチキンの「カーネル・サンダース」を連想される方もいらっしゃるかもしれませんが、そちらは「Colonel Sanders」であり、コンピュータ用語で利用される「Kernel」とはまったく異なる単語です[注3]。

コンピュータ用語の「Kernel」は、その英単語がもともと持っていた意味が語源であると言われています。「Kernel」という英単語は、クルミなどの堅果の中心部や、ものごとなどの中心部分や核心部分など最も大事な部分という意味を

注2） 対象となる物事の特徴を抽出し、その物事をシンプルに表すこと。本稿では、ハードウェアのような複雑なものを、ファイルという概念で表現することをいいます。この言葉は情報科学や計算機科学でよく使います。プログラミングではモデリングとともに重要な概念です。

注3） なお、カーネル・サンダースの「Colonel」は大佐という意味ですが、ハーランド・デーヴィッド・サンダース氏は軍隊で大佐だったわけではなく、ケンタッキー州に貢献した人物に与えられる名誉大佐の称号です。

持ちます。カーネルは、コンピュータを制御するためのまさに中心部分です。

さまざまな作業を同時に実行するしくみとは

最近の一般的なOSのカーネルは、1つのPC上でさまざまな作業を同時に行えるようにするという機能もあります。PCの脳みそであるCPUは、同時に1つのことしかできません[注4]が、カーネルが単一の脳みそで複数のアプリケーションを同時並行に行ってくれるので、ユーザはPC上で複数の作業を同時に行えます。同時に複数の作業を行えることは「マルチタスク」と呼ばれています。

アプリケーションは、「プロセス」という単位で実行されます。一般的に、マルチタスクは、実行されるべき複数のプロセスが存在するときに、各プロセスに対して短いCPUの実行時間を与えつつ、それらを高速に切り替えながら実行します。高速に切り替わりながら複数のプロセスが実行されるので、ユーザから見ると、あたかも複数のプロセスが同時に実行されているかのように見えます。このような機能が存在していなければ、Webページを見ながらメールを読むことはできませんし、PCで音楽を聴きながら文章を書くこともできません。

このように、複数のプロセスが同時に実行されるような環境をカーネルが提供しますが、各プロセスは、互いに干渉しないようにできています。

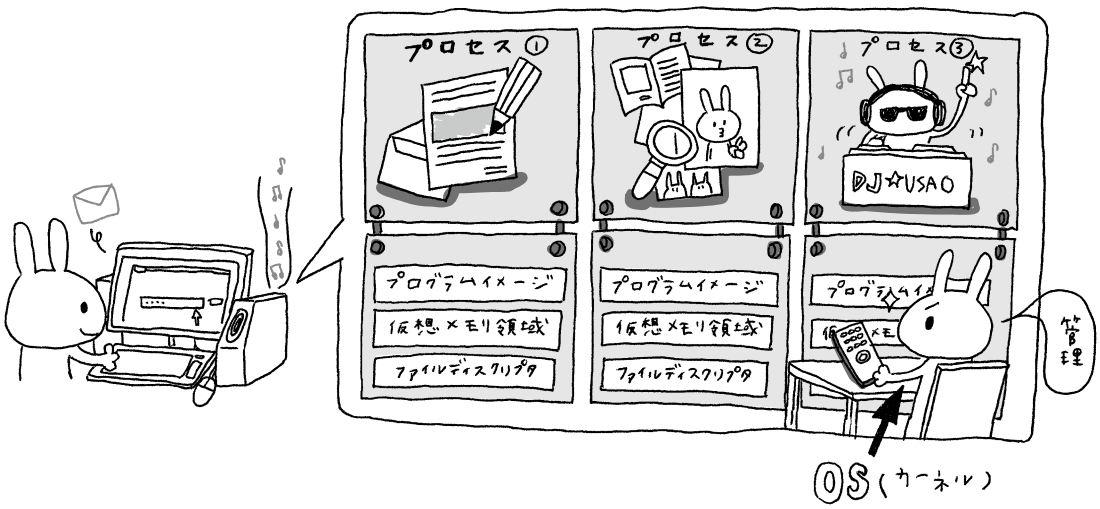
たとえば、各プロセスがコンピュータに内蔵されているメモリを利用するとき、物理的なメモリのどの部分をどうやって書き換えるのかをプロセスが直接指定することはできません(**図2**)。各プロセスに対して、カーネルが提供するのは仮想メモリ空間へのアクセスであって、物理メモリへの直接のアクセスではないためです。

各プロセスは、それぞれが管理する仮想メモリ空間に対してのみアクセスができます。そのため隣のプロセスが使っているメモリを無理矢理書き換えたりするようなことを防ぐしくみになっています。プロセスは、カーネルによって管理された閉鎖空間であり、アプリケーションはその閉鎖空間で動作しているのです。

プロセス間通信

アプリケーションが動作するプロセスは、自

▼図2　プロセスは他プロセスに干渉できない



注4）マルチプロセッサ、マルチコア、SIMD(Single Instruction Multiple Data)の話は、本稿では説明をシンプルでわかりやすくするため割愛します。

力で他のプロセスとのやり取りなどができません。また、直接ハードウェアを制御することもできません。OSに対してハードウェア制御の依頼を出すことしかできません。ユーザが書いたアプリケーションプログラムが、プロセスの外部と何らかのやりとりをするには、カーネルの助けが必要です。カーネルに通信を仲介してもらうわけです。

OSによっては、アプリケーションがカーネルに対する依頼を行うしくみを「システムコール」と呼んでいます。通信を行うためのソケットも、システムコールの1つです。

OS内で稼働するプロセスは、互いに分離されているため、直接やりとりできません。同じコンピュータ内に存在しているプロセス同士が何らかのやりとりをするには、**図3**のように、カーネルにデータの送受信を仲介してもらう必要があります[注5]。

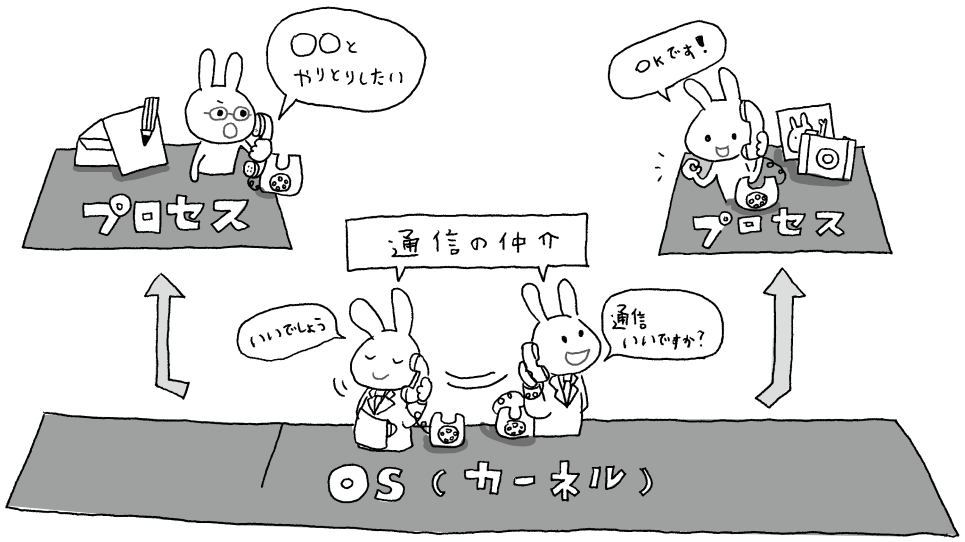
同じコンピュータ内に存在するプロセス同士の通信でさえ、カーネルの仲介が必要です。では、インターネットを介してコンピュータ同士が通信を行う場合ではどうでしょうか。やはりプロセスはカーネルの仲介を経由して通信を行います。インターネットを利用した通信を行えるソケットを用意したうえで、インターネットに接続された相手を指定すれば、通信が使えるというわけです。

図4のように、アプリケーションはソケットを通じてカーネルとやりとりし、カーネルはインターネットに接続されたハードウェアを通じて通信を行うという感じです。インターネットは、各コンピュータから送出されたデータを運ぶ役割を担っています。

このように、ソケットは、閉鎖空間であるプロセスにとって「インターネットの出入り口」であり、アプリケーションをインターネットとつなぐための「コンセント」なのです。

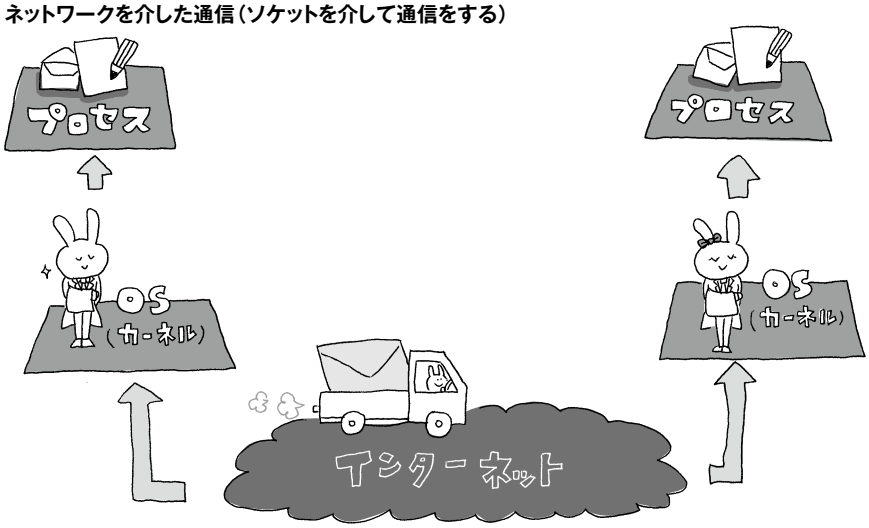
なお、「プロセスにとってインターネットの出入り口となるソケットは必ず1つ」というわけではありません。たとえば、単一のプロセス内に複数のソケットを作成することもできます。具体的には、画像ファイルが複数含まれるWebページを開いたときの一般的なブラウザは、複数のソケットを通じて同時に複数の画像ファイルをダウンロードしています。

▼図3　同一PC内でのプロセス間通信のモデル（カーネルが通信の仲介をする）



注5）プロセス間通信のしくみはソケットだけではありませんが、本節ではソケットだけに着目して解説をしています。

▼図4　ネットワークを介した通信（ソケットを介して通信をする）



普段は隠れている「ソケット」と「ポート」

次は、インターネットを通じてデータのやりとりを行う際に、「どうやって相手を指定するのか？」という話をします。その前に、最近はソケットやポートを直接指定せずに通信を行える環境が整っていることを紹介します。まず、最初に、www.example.comというWebサーバからデータを取得する簡単なプログラムをいくつか見てみましょう。

Rubyでは**リスト1**のように簡単に書けます。Perlでは**リスト2**、allow_url_fopen[注6]が有効になっているPHPでは、fopenを使って**リスト3**のようにHTTPやFTPでの通信を行えます。Objective-Cでは、NSURLConnection[注7]を使うと**リスト4**のように書けます。Javaでは**リスト5**のように書けます。

これらに示すように、いろいろなプログラミング言語でWebサーバからデータを取得して表示するプログラムが簡単に書けます。どれも

▼リスト1　Rubyの例（Net::HTTPを使う）

```
#!/usr/bin/ruby

require 'net/http'

result = Net::HTTP.get('www.example.com', '/')

p result
```

▼リスト2　Perlの例（HTTP::Liteを使う）

```
#!/usr/bin/perl

use HTTP::Lite;

$http = new HTTP::Lite;

$req = $http->request("http://www.example.com/") ||
die $!;

print $http->body();

exit;
```

▼リスト3　PHPの例（fopenを使う）

```
<?php
$fh = fopen("http://www.example.com/", "r");
if (!$fh) {
  exit;
}

while (!feof($fh)) {
  echo fgets($fh, 4096);
}

fclose($fh);
?>
```

注6）http://www.php.net/manual/ja/filesystem.configuration.php
注7）NSURLConnectionはiOSのクラスの1つ。クラスとはオブジェクト指向でいう機能を実現するプログラムの単位。

ソケットとポートをまったく意識する必要がありません。各種プログラミング言語の便利ライブラリに共通しているのが、

①URLを指定する
②データを取得する

という手法です。これらのプログラムは、プログラミング言語や利用するライブラリなどのAPI(Application Programming Interface)によって異なりますが、「インターネットを使う」という部分は同じなので似た構造になっています。このように、裏側で動いているしくみが同じであれば、言語が異なったとしても実現手法が似てくるのです。

「HTTPだから簡単にできる」という側面も

先ほど挙げたサンプルリストのどれもが、ソケットとポートを意識せずに済むのは、それらがHTTPを扱うためのものだからです。Webサーバからデータを取得するためにHTTPを使うということは、次の2点が自明です。

・TCPを利用するということ
・とくに指定しなければ80番ポートを使うこと

上記2点が確定しているので、Webサーバから情報を取得するのであれば、URLを指定すればできます。Web技術で利用されているHTTPは、今やインターネットにおける通信の大半を占めるほど利用されているものですので、それを利用するためのプログラミング環境も整備されていますが、HTTPではない通信が同様に簡単に書けるとは限りません。

HTTP以外の通信プロトコルを利用した通信プログラムを書くときや、細かい処理が必要な場合などには、ソケットを利用したプログラムを書くことが求められることもあります。必要に迫られて勉強する場合もあるとは思いますが、純粋に「知る」という方向での考え方もあります。便利なツールやライブラリが裏で何をしているのかを知ることで、インターネットをよ

▼リスト4　Objective-Cの例(NSURLConnectionクラスを使う)

```
#import <Cocoa/Cocoa.h>

int
main()
{
  id pool = [[NSAutoreleasePool alloc] init];

  NSURL *url = [NSURL URLWithString:@"http://www.example.com/"];
  NSMutableURLRequest *req = [NSMutableURLRequest requestWithURL:url];

  // 結果を格納するオブジェクト
  NSURLResponse *resp;

  // エラーを格納するオブジェクト
  NSError *err;

  // 同期的な呼び出し。sendSynchronousRequestを使っているのが特徴です
  NSData *result = [NSURLConnection sendSynchronousRequest:req
                        returningResponse:&resp
                        error:&err];

  write(1, [result bytes], [result length]);

  [pool release];
}
```

▼リスト5　Javaの例(クラスライブラリのjava.netを使う)

```
import java.io.*;
import java.net.*;

public class SampleCode {
  public static void main(String args[]) {
    try {
      URL url = new URL("http://www.example.com/");

      Object content = url.getContent();

      if (content instanceof InputStream) {
        BufferedReader reader =
          new BufferedReader(new InputStreamReader((InputStream)content));

        String str;
        while((str = reader.readLine()) != null) {
          System.out.println(str);
        }
      }

    } catch (Exception e) {
      System.err.println(e);
    }
  }
}
```

り身近なものと感じられるかもしれません。

ということで、そういったしくみの裏側で動いている「ソケット」と「ポート」に関して、もう少し詳しく見ていきましょう。

通信の「入り口」となるソケット、何と通信するかを指定する「ポート」

まずは通信手順を見てみよう

まずは、実際の例を見てみましょう。ソケットを使ってインターネットでの通信を行うプログラムは、**図5**のように、❶ソケットを用意して、❷通信したい相手のIPアドレスとポート番号を設定し、❸相手とつないで、❹情報をやりとりする、という過程を経て通信します。

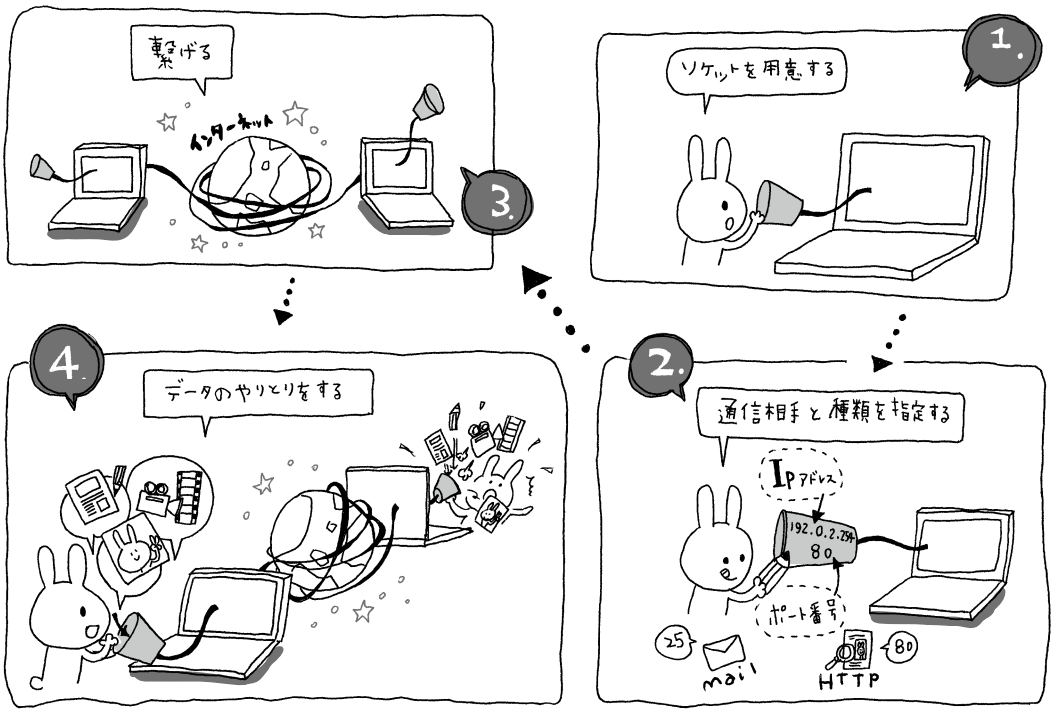
❶は、ソケットを用意するというものです。ユーザから見ると、PCを通じて通信を行うための「口」が、ニュウっと伸びて来ているような感じに見えるかもしれません。ソケットを用意するときに、そのソケットをどのように使いたいのかも指定します。たとえば、インターネットを利用してWebサーバと通信したいのであれば、TCPでの通信を行うためのソケットを用意します。

❷は、通信相手と種類を指定するというものです。多少語弊はありますが、IPアドレスは「通信相手」を指定するために利用されるもので、ポート番号は「通信の種類」を指定するために利用されます。

ポート番号というのは、通信の種類を番号で表したものです。たとえば、HTTPの一般的なTCPポート番号は80番ですし、メール配信の一般的なTCPポート番号は25番です。❷で重要なのが、通信する相手のIPアドレスを指定するというものです。たとえば、www.example.comはIPアドレスではありません。www.example.comは、「名前」であり、それをIPアドレスに変換したものが通信に利用されます(「名前」に関しては、DNSの解説として後述します)。

❸は、❷で設定した相手と実際に「つながる」というものです。この作業が必要なタイプの通信と、そうでないものがあります。TCPでの通

▼図5　ソケットとポートで通信する手順



信では、「つながる」という作業が必要になります。

❹は、つながったあとにデータをやりとりするというものです。

一度つながってしまえば、あとはそのソケットを利用してデータを出し入れするだけです。ユーザから見ると、手元にあるファイルからデータを引き出すのも、ネットワークの向こう側にあるサーバからデータを引き出すのも同じような感覚で利用できます。ソケットを利用してTCPで通信を行うと、こんな感じになります。

ここまでの説明で、「クライアント」だの「TCP」だのという単語を説明なしに使ってしまいましたが、それらをこのあとに徐々に紹介していきます。

IPアドレスとポートを指定する!

先ほどの例では、通信相手をIPアドレスとポート番号で指定しています。たとえば、Webサーバとの通信が行いたいのであれば、WebサーバのIPアドレスを指定したうえで、80番のポート番号を指定します。

IPアドレスは、「どのサーバと接続するのか？」を指定するためのものです。現在のインターネットは、おもにIPバージョン4(以後、IPv4)によって構成されていますが、IPv4でのIPアドレスは32ビットの値で表現されます。

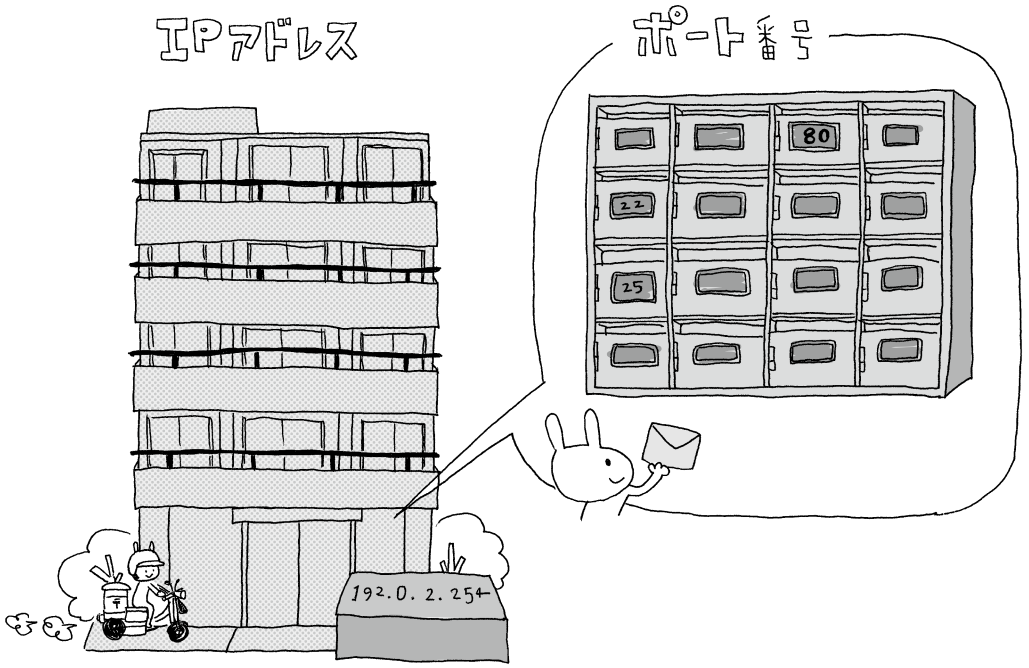
この32ビットのIPアドレスは、ドット付き十進表記(dotted decimal notation)と呼ばれる0～255の数字4組をドットでつないだ記法で表現されます。たとえば、192.0.2.254のような感じです。「.」の間にある数字は、人間が扱いやすいように、32ビットのうちの8ビットずつを4つに分けて十進数で表現したものです。それぞれ8ビットですので、表現できる範囲は0から255になります。

IPv4やv6の話を聞くと(コラム参照)、「WebサーバのIPアドレスだけで、何がいけないの？」という疑問も持たれるかもしれません。

そこで本稿では、IPアドレスとポート番号が意味するところの違いを理解するためのアナロジーとして「マンションと部屋番号」を提案したいと思います(**図6**)。

あるWebサーバに対して1つのIPアドレスが設定されていたとします。その中で、Webサーバとしてのサービスと、メールサーバとしてのサービスが同時に動いていたとします。そのと

▼図6　IPアドレスがマンションで、ポート番号が部屋番号のたとえ



き、IPアドレスだけではWebサーバと通信をしたいのか、それともメールサーバと通信をしたいのかがわかりません。あたかも郵便局員がマンションに到着したけど、部屋番号がわからずにどこに小包を届けて良いのかわからなくなるような感じです。

たとえば、同じサーバでメールとWebとDNSを運用したい場合を考えます。そういった場合、メールが25番、Webが80番、DNSが53番をそれぞれ使ってサービスを提供します。さらに言うと、Webサーバを運用するために80番のポート番号以外を使ってはならないという決まりもないので、80番ではない番号で運用をすることもできます。これにより、複数のWebサーバを1台のコンピュータで稼働させることも可能になります。

このように、ポート番号が存在することによって、1つのマンションに複数家庭が入居できるのと同じように、1つのIPアドレスで複数のサービスを稼働できます。

COLUMN 「IPv4/IPv6」

IPv4でのIPアドレスは32ビット長ですが、インターネット利用者数が非常に多くなったため、世界のIPアドレスが枯渇してしまいました。2011年に発生した「IPv4アドレス中央在庫の枯渇」です。そのため、地域によっては、新たにIPアドレスを取得するのが困難になっています。

このIPv4のIPアドレス枯渇が発生することは、以前から予想されていたため、IPアドレス空間が非常に大きいIPバージョン6(以降IPv6)が作られています。IPv6のIPアドレスは128ビットで表現されているので、IPv4のIPアドレスよりもかなり大きな空間です。これは、IPv4の4倍ではなく、2の96乗倍であり、非常に大きな値です。IPv6のユーザ数は、IPv4と比べると、まだまだ少ないのですが徐々に増えつつあります。

Part 2 インターネットを上から眺めてみる

インターネットのしくみ

TCPやポート番号に関する詳しい説明に入る前に、まずはインターネットそのもののしくみを紹介します。インターネットがどういうしくみであるのかの概要を知っていると、さまざまなしくみが「なぜそうなっているのか？」を理解しやすいからです。

いい加減だけど粘り強い、インターネット

インターネットの特徴である「自律分散システム」に着目して解説します。

物理的な回線に障害が発生したとしても、何らかの方法で障害発生個所を回避するという能力がインターネットには備わっています。備わっているというよりは、むしろ、そういった粘り強いしくみを研究した結果として誕生したのがインターネットです。その粘り強いしくみは、一見いい加減とも思える「自分ができること以外はやらない」、「途中経路上のルータは、データが相手に届くかどうかを気にしない」、「自分が何をどのように転送しているのかも深く考えない」というようなしくみによって構成されています。誰もインターネットの全体像を知りませんし、知らなくてもなぜか相手と通信ができてしまいます。

たとえば、ユーザの手元のPCは、宛先は把握していますが、そこまでどのようにデータが届けられるのかを知りません。一方で、データの転送を行っている途中経路の機器も、具体的にどこをどうデータが届けられるのかを細かく把握せずに動作しています。

こういった「いい加減さ」が、逆に粘り強さを実現しているのがインターネットの面白いところでもあります。インターネットは、各自がそ

のときどきで最善の判断をしようとするので、全体像を把握しつつ全体を制御するようなしくみになっていません。

そのため、「この1個所が潰されたらインターネット全体が完全停止してしまう」といった弱点が構築されにくい構造になっています。誤解を恐れずに書くのであれば、ちょくちょくと小さな障害が発生するのは、インターネットそのものの前提です。

日本からアメリカへの通信を運ぶ物理回線を考える

概念的な話ばかりになってしまうとわかりにくくなるので、まずは具体的に日本にいるユーザがアメリカにあるサーバと通信する場合の物理回線の例を考えてきます。

インターネットは魔法ではないので、何らかの物理回線を通じて通信が行われます。インターネットを構成する物理回線は、単一の事業者がすべてを管理しているわけではなく、各所でさまざまな事業者が独自に用意したものが利用されています。

今回の例の全体像としては、**図7**のようになります。

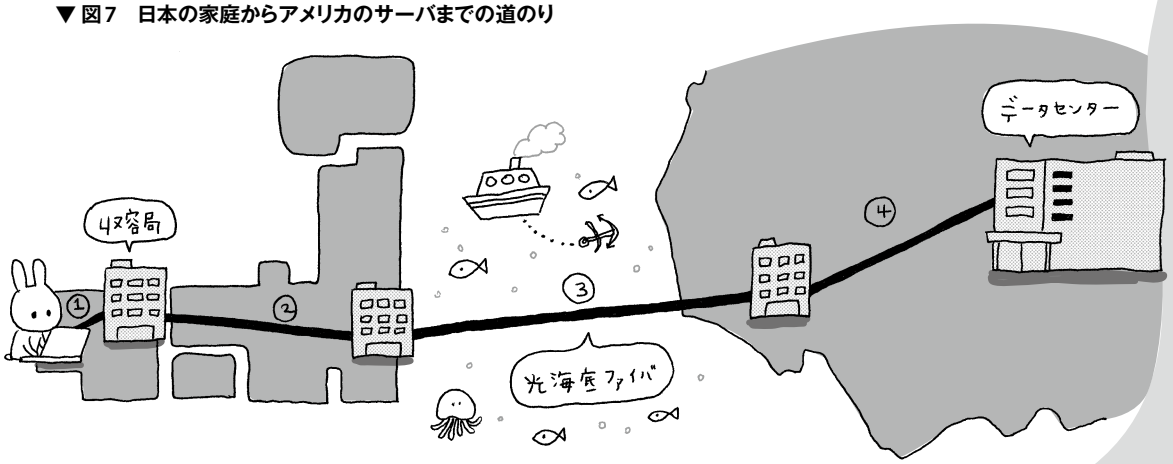
まず最初に、家庭内LANから収容局までの物理ネットワークがあります。各家庭までの回線は「ラストワンマイル」と呼ばれることがありますが、光ファイバや銅線による有線回線や、モバイル通信サービスなどによる無線回線などが考えられます。

各収容局は、他のエリアと光ファイバで接続されています。今回の例のように、日本からアメリカへの通信を考える場合、収容局から光海底ファイバ陸揚局までデータが配送されます。日本からのデータをアメリカまで運ぶには、太平洋を超えた通信が必要になります。太平洋を超える通信は、海底に敷設された光海底ファイバを通じて行われます。アメリカ本土に到達したデータは、アメリカ国内での伝送を行う光ファイバ網を通ってアメリカ国内にあるデータセンターまで運ばれます。

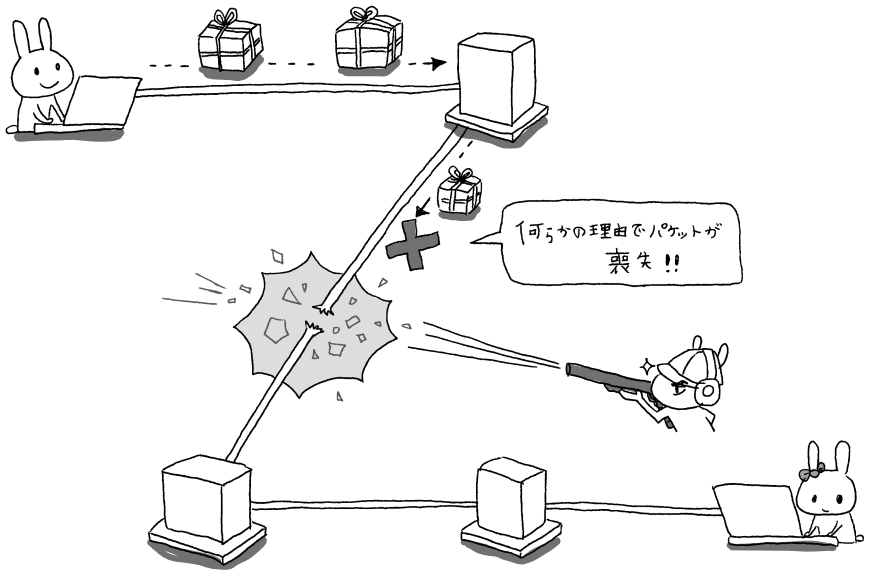
このように、インターネットを利用して海外にある機器と通信が行えるのは、何らかの物理的な方法によって手元のPCから海外まで通信が行える回線がさまざまな事業者によって整備されているためです。インターネットは、さまざまな運営主体が、さまざまな種類の機器を互いにつなげ合って構成された世界的な巨大ネットワークなのです。

しかし、物理的なものは壊れることがあります。インターネットを構成する回線も例外ではありません。たとえば、日本国内でユーザの家庭に伸びる光ファイバケーブルにクマゼミが卵を産見つけて光ファイバが破損することもあります。地中に埋められていた光ファイバをショベルカーが掘り起こして切断されたり、漁船の錨が光海底ケーブルを引っ掛けてしまって壊してしまう

▼図7　日本の家庭からアメリカのサーバまでの道のり



▼図8　なんらかの理由で通信が届かないことも



ようなこともあります。アメリカ国内で庭師が光ファイバをハサミで切ってしまったり、ハンターが空中配線された光ファイバを散弾銃で撃って壊してしまうこともあります[注8]。ケーブルが盗まれることもあれば、ハリケーンなどの自然災害の影響を受けることもあります(**図8**)。

そのような物理回線への障害が発生すると、その瞬間に転送されていたデータが壊れたりします。いついかなるときも、転送中のデータが絶対に壊れない物理回線というものは、恐らく存在しないでしょう。

データを送るための パケット交換技術

データがどのように届くのかを知る前に、まずはデータがどのような単位で相手まで届けられるのかを知る必要があります。

インターネットを流れるすべてのデータは、「パケット」と呼ばれる小さな単位に分割されて、届けられます。各パケットはすべて個別の小包や葉書のように別々に送付されます。たとえば、あるPCからほかのPCにファイルを送信するとき、**図9**のようにファイルが細切れに分割されたパケットとして送信されます。

この「小分けにされる」というのが非常に重要です。小分けにしてデータを送ることができるパケット交換技術が発明されるまでは、たとえば電話などの通信は回線交換技術によって行われていました。その欠点は回線交換技術は通信中に回線を占有することです。

一方、小分けにすると**図10**のように、各小分けデータが回線を専有する時間を短くできるので、あたかも複数のデータが1つの回線を同時に使っているように見えるようになります。複数の通信を1つの回線で共有することで、回線の利用効率が向上します。

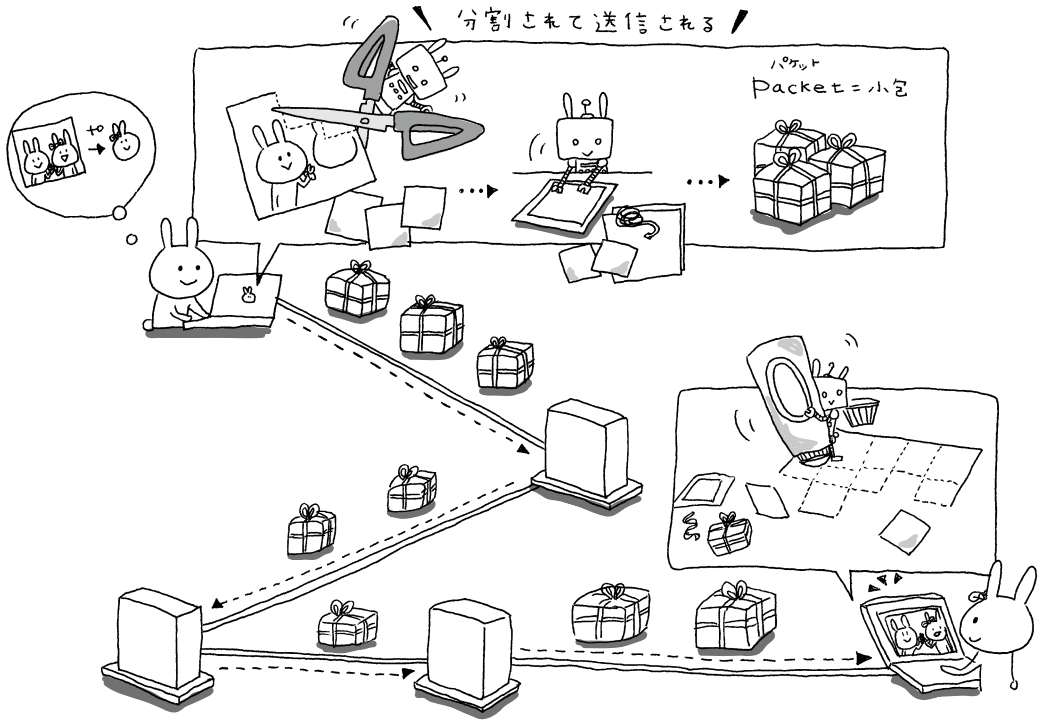
このようなパケットをインターネット上で転送していく機器は「ルータ」と呼ばれます。ルータは、各自の範囲内でできることを行いつつ、互いに連携しながら巨大なネットワークであるインターネットを構成しています。各ルータはインターネットの全体像を把握しているわけではなく、パケットに記載されている宛先を見ながら、次のルータに向けてパケットを転送しているだけです。

パケットの宛先はどこ

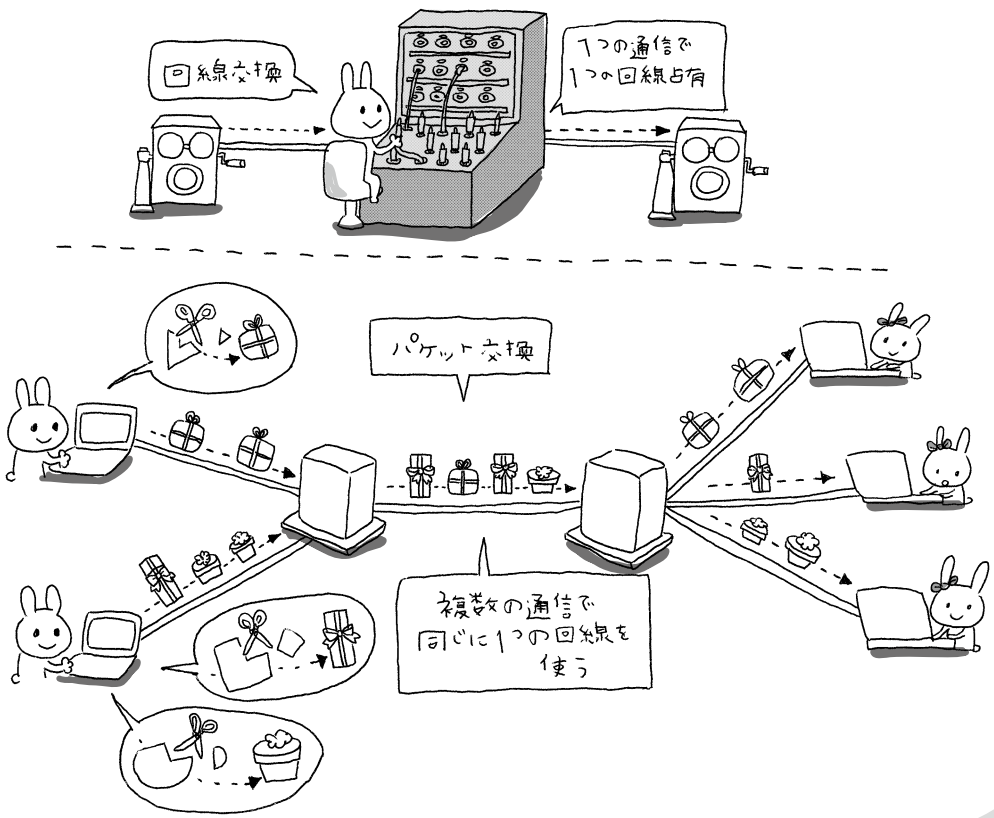
インターネットは、ルータがパケットを転送することで実現されています。ルータは、パケッ

注8) 次のURLは、2010年にGoogleが通信障害に関して語ったプレゼンテーションです。NANOG 49: Worse is better ; https://www.nanog.org/meetings/abstract?id=1595

▼図9　パケットに分割されて送信されるファイル（データを小分けにしている）



▼図10　パケットに区切ることによって回線を共有（少ない回線数で通信可能）



トを可能な限り転送する努力はしますが、パケットの到着を保証しません。そのため、パケットが途中経路で喪失することもあります。パケットが喪失する理由には、ルータ間をつなぐ物理的な回線障害以外にもいろいろあります。たとえば、ルータが物理的に故障することもありますし、ルータに対して管理者が間違った設定をしてしまうこともあります。ある特定のルータに対して過度なトラフィックが集中して、ルータがパケットを処理しきれずにパケットが喪失してしまうこともあります(**図11**)。

何らかの理由によってパケットが喪失する障害が発生したときに、「こっちは通信ができないらしい」と周辺のルータが把握し、代替経路を各自が自律的に計算します(**図12**)。新しい経路が発見されれば、その経路を通じてパケットが送信されます。このように、インターネットを構成するルータが各パケットの詳細を把握せずに宛先だけを見てパケットを処理するのも、インターネットの重要な特徴です。各パケットがどのような順番で配送されるべきかなどを途中経路上のルータが把握しなくても良いので、ルータは必要最低限の情報だけを把握するだけで動作可能です。

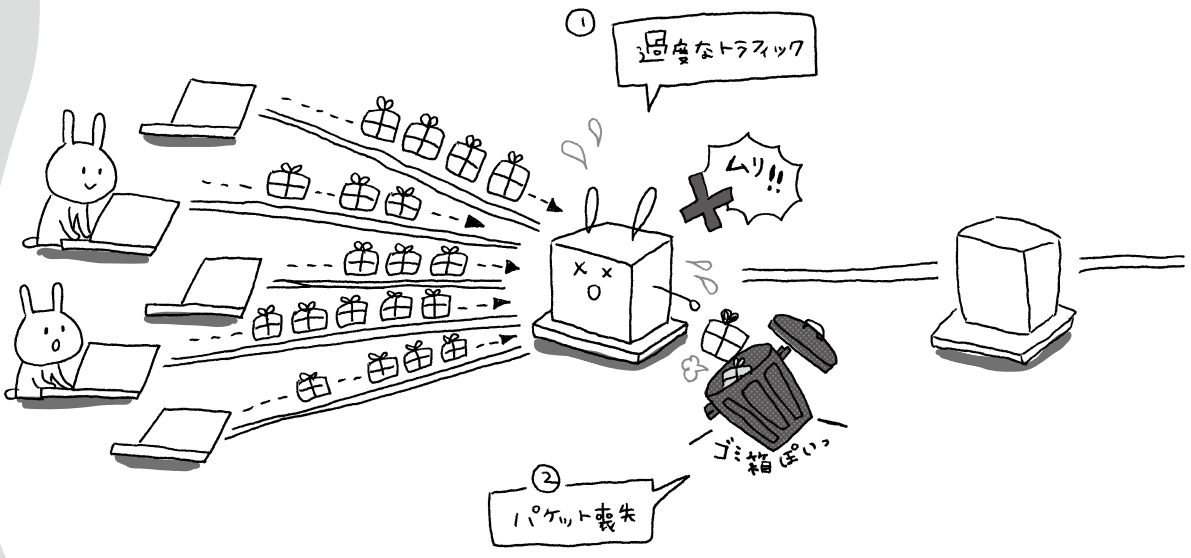
到達性を保証するTCP

パケットの到達性が保証されないネットワークで通信を成り立たせるための機能を実現しているのがTCP(Transmission Control Protocol)です。TCPは、実際に通信を行っている末端機器同士が利用するしくみです。通信経路そのものを実現しているルータは、最低限のことだけを把握すれば良く、どのパケットがどのように喪失したのかに関する処理は、通信を行う末端同士が把握するという形です。

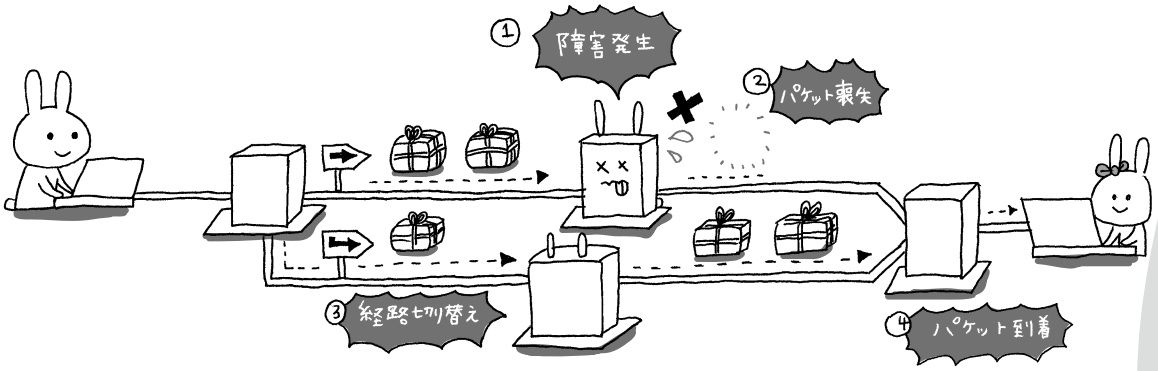
途中経路上のルータがパケット喪失を把握しなければならない設計であれば、ルータに要求される性能が非常に高いものになるだけではなく、そのネットワークに接続可能な機器の数も著しく制限されていたはずです。パケットの配送を行うルータは、可能な限り単純な作業だけを行い、末端同士が複雑なことをするような役割分担にしたからこそ、インターネットは世界規模の巨大ネットワークへと成長できたのだと思います。

TCPは、パケット喪失を検知するだけではなく、**図13**のように、届いていないパケットを再度送信してもらうことを促します。途中経

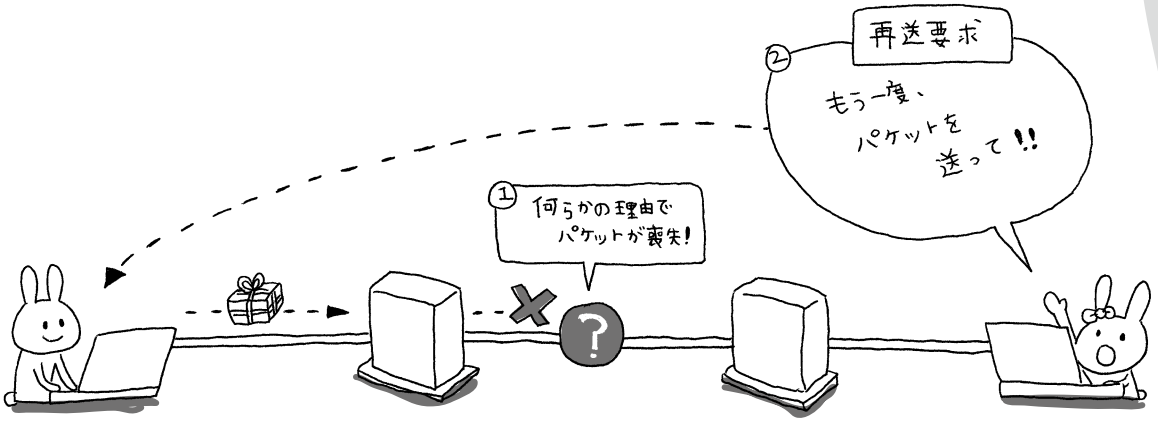
▼図11　過度なトラフィックによるパケットの喪失



▼図12　経路切り替えによるデータの喪失



▼図13　再送要求（パケットが消失したときは、再送信する）

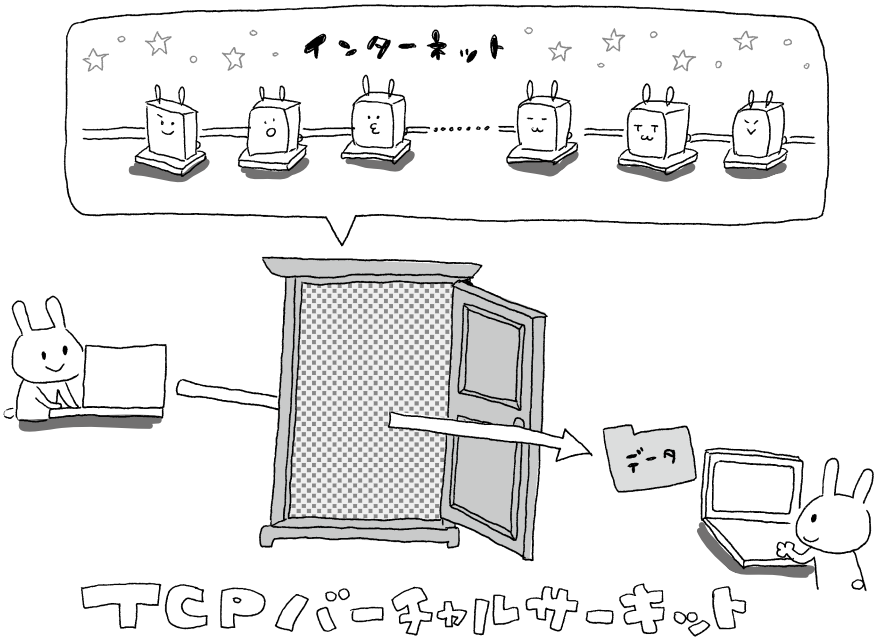


路上でパケットが喪失したとしても、送信元から再度送信され、再送信されたパケットが宛先に無事到着すれば、送信元から宛先まで必要なデータが届けられます。

このように、相手にデータが届いたかどうかを確認することで、TCPは仮想的な接続が存在しているような機能を実現しています。これは、「バーチャルサーキット」と呼ばれています。このバーチャルサーキットは、離れた相手とあたかも直接つながっているかのような錯覚を起こさせるような機能を持っています。バーチャルサーキットによって、ユーザはソケットが「ドラえもんのどこでもドア」のように見えます。「どこでもドア」に対してデータを入れたり出したりするだけで通信が行えるためです（**図14**）。

このように、TCPが裏で頑張ることによって、TCPを利用するアプリケーションはあまり深く考えずに離れた相手とデータのやりとりを行えるわけです。アプリケーションを作る人が、途中ネットワークでのパケット喪失を意識せずにプログラムを書けるというのは、非常に大事です。仮にTCPが存在しなければ、インターネットを利用するすべてのユーザが、ネットワークそのものの挙動を細かく知りつつ通信を行わなければならないという状況が存在したかもしれないと思うと、TCPが存在しなければインターネットは普及しなかった可能性すらある気がしています。

▼図14　バーチャルサーキットのモデル図



TCPによる接続の確立

次は、TCPによる通信開始がどのように行われるのかを紹介します。何もせずに、常にTCPのバーチャルサーキットが存在し続けているわけではありません。TCPでの通信を行うには、まず最初にTCP接続を確立する必要があります。TCPでは、通信を開始したい側が「このポート番号で接続させてください」という接続要求パケットを送信します。接続要求を受け取った側は、その接続を受け入れるのであれば、「いいですよー」という内容の応答を返します。「いいですよー」という応答を受け取った通信開始側は、「ありがとうございます。宜しくお願いします。」という内容を送信して、TCP接続が確立します。このやりとりでは、3回メッセージがやりとりされるので、3 way handshake(3方向の握手)と呼ばれています(**図15**)。

3 way handshakeで最初に送信されるTCP接続要求パケットは「SYNパケット」と呼ばれます。このSYNというのは、「synchronize(同期する)」という意味です。SYNパケットによって同期されるのは、パケットが運ぶデータの位置を示すためのシーケンス番号です。シーケンス番号は、どのパケットが喪失したのかや、配送中にパケットの並び替えが発生したことを検知するために利用されます。接続を開始するというよりも、シーケンス番号を同期することが名称に反映されている点が非常に興味深いと言えます。

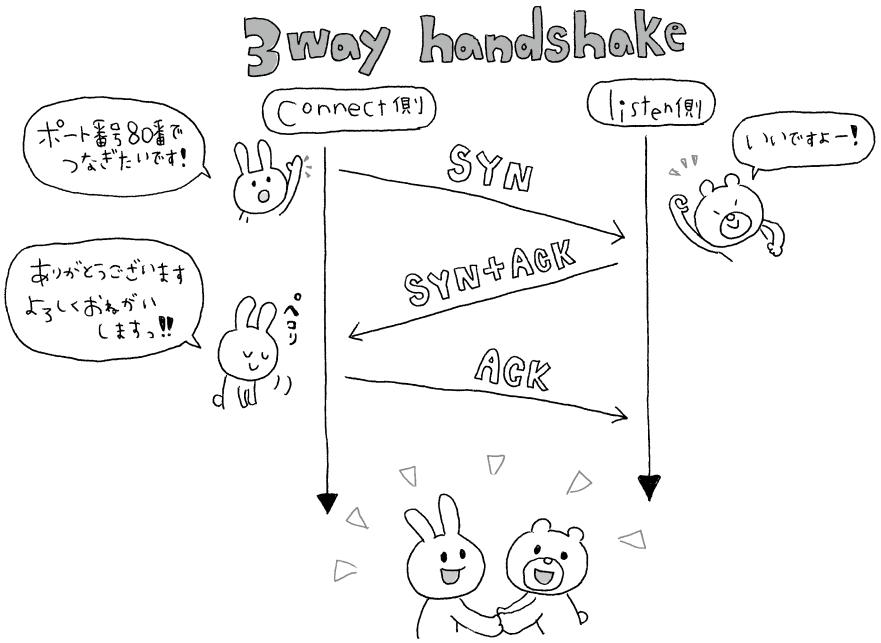
SYNパケットを受け取った側が、接続を許可するときに送信するパケットは「SYN + ACK」と呼ばれています。ACKというのは、受け取り通知や承認という意味を持つ英単語「Acknowledgement」の頭文字です。「SYN + ACK」というのは、SYNに対するACKという意味です。

ACKは、3 way handshakeの最後に送信されますが、その後データをやり取りする際に「データがちゃんと届きましたよー」という通知を接続相手に伝えるためにも利用されます。

セッションの識別と「ポート番号」

これまで、「ポート番号」としてWebサーバ

▼図15　3 way handshake（3方向の握手）

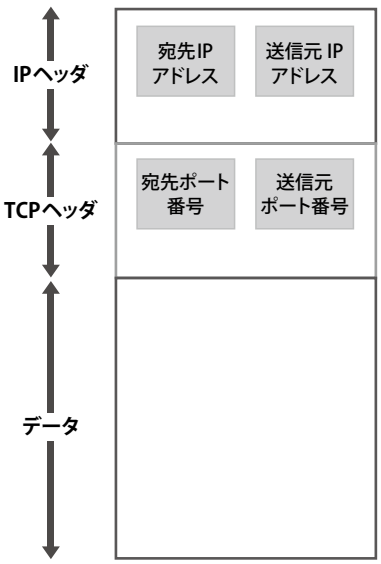


が利用する80番などを紹介してきましたが、それはポート番号の一部でしかありません。TCPによって個々のバーチャルサーキットは、それぞれ独立しており、それぞれの通信内容が混じってしまうと困ります。TCPでは、各バーチャルサーキットは「セッション（session）」と呼ばれています。各TCPパケットが、どのセッションに含まれたものであるかは、パケットのIPヘッダとTCPヘッダに記載された情報によって識別されます。

識別に利用される情報は、IPヘッダに記載されたプロトコル番号（TCPなので6番）と送信元IPアドレスと宛先IPアドレス、TCPヘッダに記載された送信元ポート番号と宛先ポート番号の5つです。TCPセッションの識別に使われるこれらの5つは、5タプル（5 Tuples）と呼ばれることがあります（**図16**）。

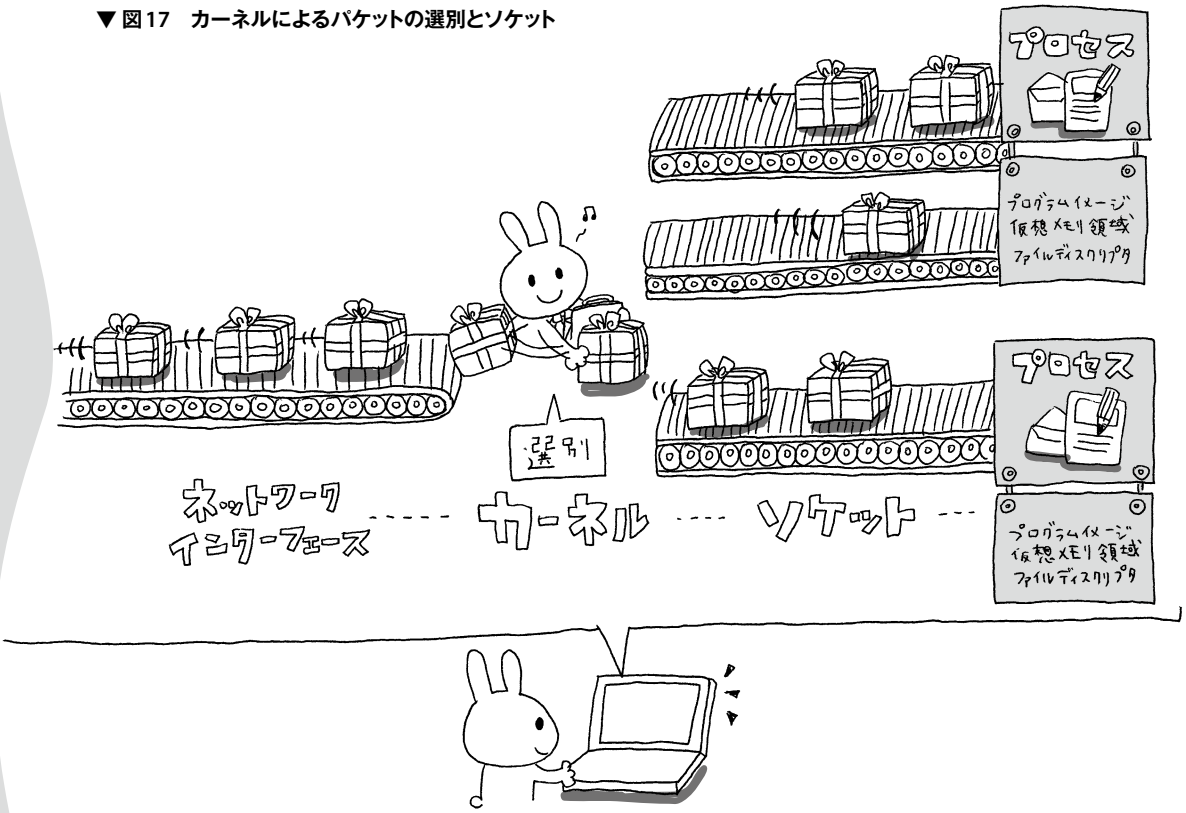
カーネルは、受け取ったパケットのIPヘッダとTCPヘッダに含まれる情報から、各パケットをどのように処理すべきかを判断します。**図17**のように、カーネルがネットワークインターフェースから受け取ったパケットをひとつひとつ確認して、適切なソケットへデータが渡されるようにします。

▼図16　TCPパケットのモデル図

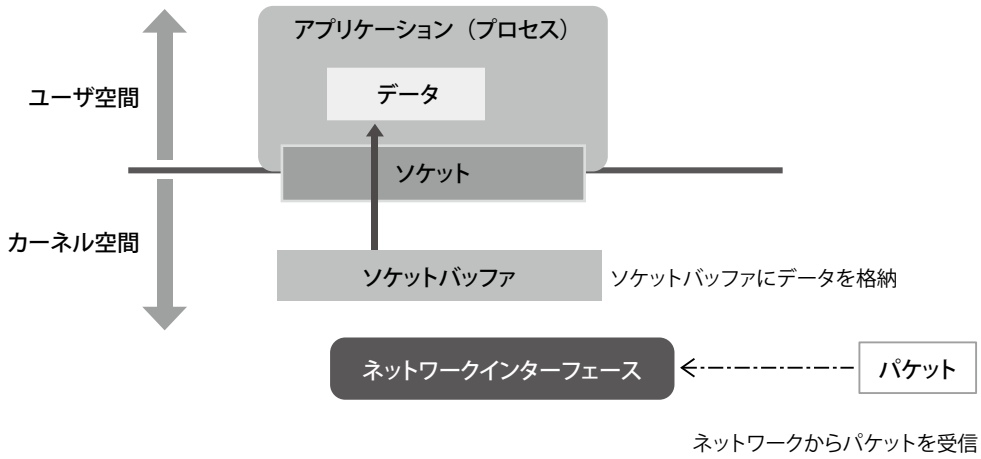


TCPでは、パケットそのものがソケットを経由してアプリケーションに渡されるわけではありません。ユーザが欲しいのは、送信側のアプリケーションで「どこでもドア」であるバー

▼図17　カーネルによるパケットの選別とソケット



▼図18　パケットに含まれるデータがプロセスに渡されるまで

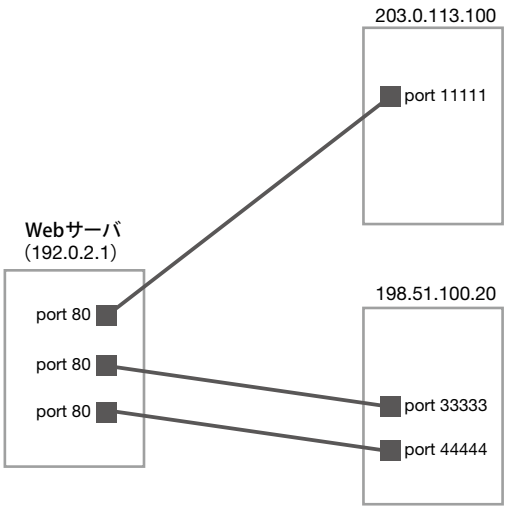


チャルサーキットに送り込んだデータそのものであり、通信経路上でやりとりされるパケットそのものではありません。

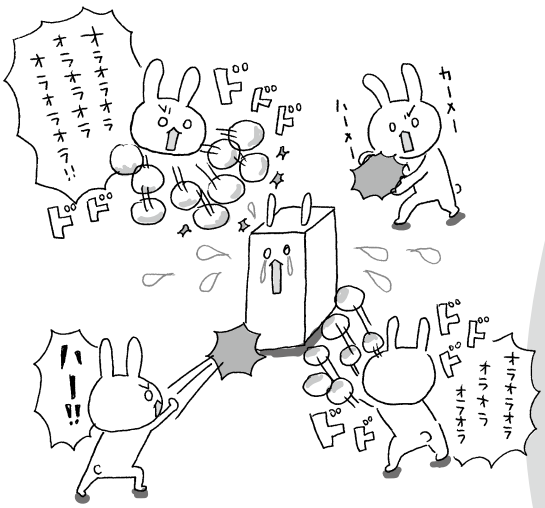
カーネルは、ネットワークインターフェースで受信したパケットを「ソケットバッファ」と呼ばれる一時保管用の領域に格納したうえで、必要に応じてソケットを経由してデータをアプリケーションへと渡します（**図18**）。

さて、次は、もう少しTCPのポート番号を掘り下げて考えてみましょう。TCPは、バーチャ

▼図19　TCP接続の一意性



▼図20　ネットワーク輻輳状態とは



ルサーキットを確立しますが、同じIPアドレス同士で2本以上のバーチャルサーキットを確立するには、どうするのでしょうか？

ポート番号に「送信元ポート」と「宛先ポート」の2種類があるのは、そういった状況にも対応できるようにするためです。

図19は、192.0.2.1というIPアドレスで運用されているWebサーバに対して、2つクライアントからTCP接続が張られています。203.0.113.100というIPアドレスのクライアントから1本と、198.51.100.20というIPアドレスのクライアントから2本です。2つのクライアントは、Webサーバに対してTCPポート80番を宛先とするTCP SYNパケットを送信して、TCP接続を確立します。

図19で着目すべきは、198.51.100.20というIPアドレスを持つクライアントから2本のTCP接続が確立されている点です。この2つのTCP接続で異なるのは、クライアント側の送信元TCPポート番号です。このように、5タプルのうち1つでも違えば、異なるTCPセッションとして認識できるので、同じ機器同士で複数のTCPセッションを通じて通信を行うことができます。

実際、皆さんが利用されているWebブラウザは、ユーザが気がつかないうちに同じサーバに対して複数のHTTPセッション経由で同時にデータを取得しています。たとえば、1つのWebページに複数の画像が含まれるときなどに、複数のTCPセッションがデータを運んでいます。

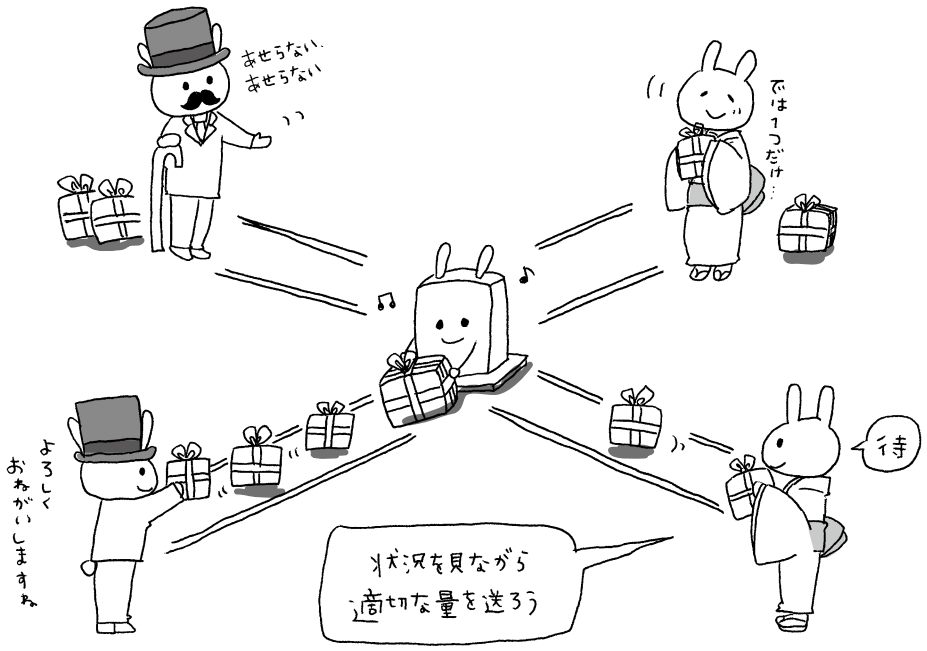
TCPによる輻輳制御機構

TCPの機能は、パケットの喪失に対応するだけではありません。TCPが提供する非常に重要な機能として、「利用可能なネットワーク帯域に合わせた通信を行う」という輻輳制御機能もあります。TCPの輻輳制御機構は、TCPの重要機能であると同時に、インターネットにとっても非常に重要な要素です。

図20のように、インターネットに接続した機器が、途中ネットワークでの輻輳をまったく考慮せずに各自で好き勝手に「オラオラオラー！」とパケットを送信してしまうと、途中経路が輻輳だらけになってしまい、まったく通信ができない状況が定常的に続いてしまう可能性があります。

TCPは、インターネットに接続された末端機器同士が各自の判断で送信量を増やしたり減らしたりしているわけですが、その結果として、**図21**のように末端機器同士があたかも通信帯

▼図21　通信帯域の譲り合い



域を「ゆずりあっている」ようにも見えます。

実際は、最適送信量でパケットを送ることで効率良くデータを送信できる、ということを目的としているので礼儀正しくすることを目的としているわけではないのですが、「オラオラオラー！」と何も考えずにフルパワーでパケットを送信しまくるのと比べると礼儀正しいようにも見えます。

TCPの輻輳制御機構にはさまざまな種類がありますが、基本的なものとしてはたとえば、パケット喪失が発生するまではパケット送信量を倍々にしていき、パケット喪失を検知するとパケット送信量を1に戻すという手法があります[注9]。

このような、徐々にパケット送信量を増やしていく方式は「スロースタートアルゴリズム」と呼ばれます。

スロースタートアルゴリズムでは、TCP接続確立直後に同時送信可能なパケット数が1となるので、TCP接続確立直後は利用可能なネットワーク帯域が狭いという特徴があります。この特徴は、最近さまざまなところで話題のHTTP 2.0が提案される背景にもなっています。

ソケットを利用したTCPプログラミング例

TCP接続が確立されるとき、「ポート○○につなぎたいです」と言う側と、それに対して「いいですよー」と言う側がいます。「つなぎたい」とSYNパケットを送信する側は「クライアント」、それを受け付ける側は「サーバ」と呼ばれます。

TCPの接続が確立したあとは、TCPそのものの機能としてはサーバとクライアントに差異はありませんが、TCP接続確立段階で動作が違うので、その部分はプログラムの書き方も違います。ソケットを利用したプログラミングを行うときの、TCPサーバとクライアントの違

注9）ある閾値を超えると倍々ではなく1ずつ増えます。また正確には「パケット数」ではなく「セグメント数」です。本稿では詳細を割愛しますが、興味がある方はTCPについて調べてみてください。面白いです。

COLUMN 「HTTP 2.0」

本稿執筆時点ではHTTP 2.0はまだ検討途中であり、正式な仕様が決定しているわけではありません[注10]。しかし、長年利用され続けているHTTP 1.1との後方互換性を維持しつつも、大規模な機能拡張が行われようとしています。インターネットで行われる通信で、最も多く利用されているのがHTTPであるとも言われており、HTTPの新バージョン登場はさまざまな変化を起こす可能性があります。

従来HTTPとHTTP 2.0を比べると**図22**のようになります。具体的には、それまで基本的に1つのTCPセッションが1つのHTTPセッションとなり、1つのHTTPリクエストに対してHTTPレスポンスが返されるとTCPセッションも終了するという利用が大半でした[注11]。

それまで、非常に細かい単位で毎回TCPセッションを張り直していたため、各TCPセッション開始直後のTCP確立までの時間や、TCPセッション開始後のスロースタートが、TCPセッションごとに毎回行われていました。

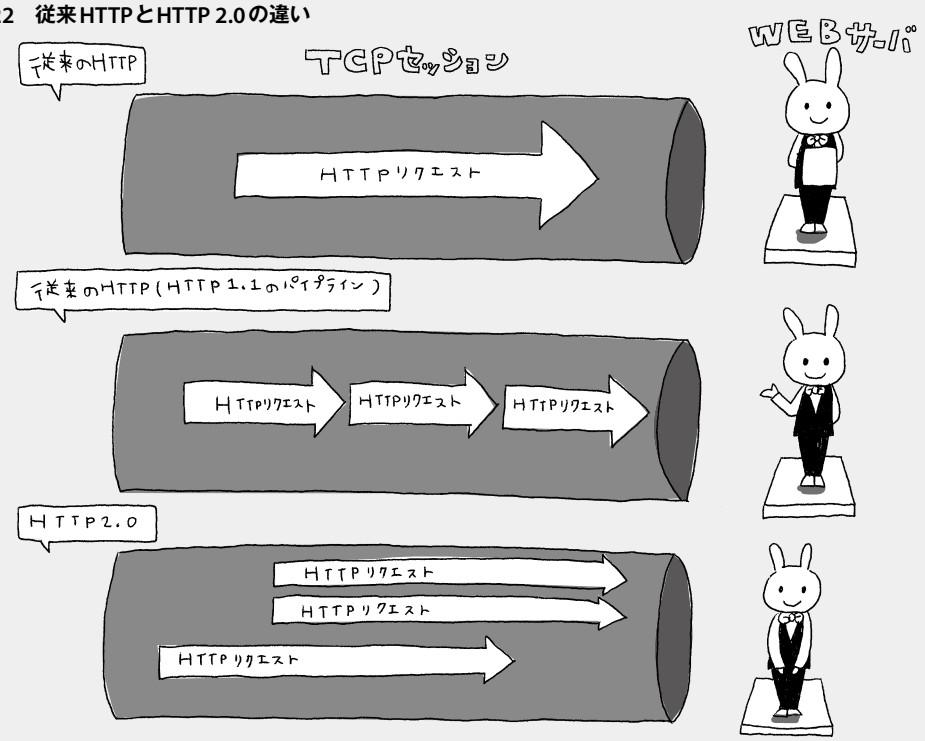
たとえば、あるWebページ内に画像ファイルが50個含まれていれば、元となるWebページを取得する1回のTCPセッションの後に、画像ファイルを取得するために50回のTCPセッションが開始されます。数キロバイトぐらいの小さい画像ファイルが多い場合、接続してはすぐに切断するような細かいTCPセッションが頻発します。

HTTP 2.0は、1つのTCPセッションで複数のHTTPセッションをやりとりできるようにすることで、それまでTCPセッションのたびに行われていた処理が行われずに、HTTPセッションを高速化できるというものです。しかし、TCPセッション数を減らすことによって通信性能が低下する可能性もあるので注意が必要です。

TCPセッションによって利用可能な通信帯域の理論的な限界は、環境によってある程度推測できます(RFC 5348参照[注12])。たとえば、遠距離で通信を行い、RTT[注13]が大きいような環境ではTCPセッションの本数を増やしたほうが通信性能が上昇する傾向があります。そのため、TCPセッション数を何本にするのかや、どういったときにどれぐらいのHTTPセッションを1つのTCPセッションに内包させるのかなどによって、各実装で通信性能などに差が出てくるかもしれません。

なお、HTTP 2.0の仕様は本稿執筆時点では、まだドラフトであり、決定しているわけではありません。

▼図22　従来HTTPとHTTP 2.0の違い



注10) Hypertext Transfer Protocol version 2 (internet draft)(http://tools.ietf.org/html/draft-ietf-httpbis-http2)
注11) HTTP 1.1のパイプラインのしくみを利用すれば1つのTCPセッションで複数のHTTPリクエストを処理できます。
注12) RFC 5348: TCP Friendly Rate Control (TFRC): Protocol Specification(http://tools.ietf.org/html/rfc5348)
注13) RTT(Round Trip Time)ある区間で行われている通信における送受信の経過時間。

いを図23に示します。

TCPによる通信プログラムを書く場合、第1引数をAF_INET(IPv6の場合はAF_INET6)、第2引数をSOCK_STREAMにしたsocketシステムコールでソケットを作成します。この部分は、サーバとクライアントで同じです。

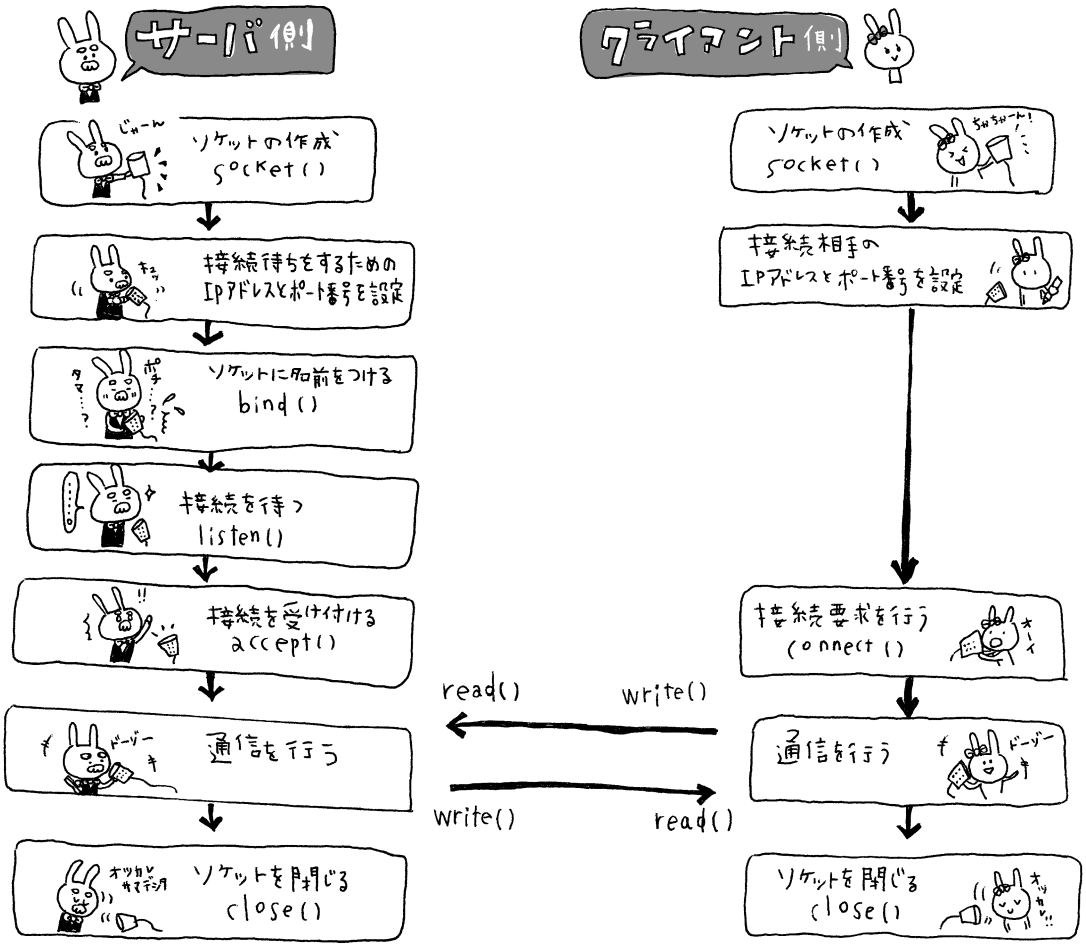
では、まずはクライアント側を見てみましょう。socketシステムコールを使って作成したソケットを利用して、connectシステムコールで通信を行いたい相手とTCP接続を確立します。connectシステムコールの第2引数に接続相手情報を渡すことでTCP接続の相手を指定できます。TCP接続の確立に成功すると、connectシステムコールは成功しますが、そのソケットに対して読み書きを行うことでサーバとデータのやりとりが可能になります。

最後に、使い終わったソケットをcloseシステムコールで閉じて、ひととおり終了です。

次に、サーバ側を見ていきましょう。socketシステムコールを使って作成したソケットに対して、次節で説明するbindシステムコールを使って待ち受け用のIPアドレスとTCPポート番号を設定します。そのあと、listenシステムコールによってTCP SYNを受け付けるようになります。

listen実行後に、待ち受けが行われているポート番号でTCP接続が確立されるようになりますが、確立し終わったTCP接続から新たなソケットを生成するのがacceptシステムコールです。サーバアプリケーションは、acceptシステムコールが生成したソケットに対して読み書きを行う

▼図23　TCPのサーバとクライアント



ことで、クライアントとの通信を行います。

サーバ側では、closeシステムコールをacceptで生成されたソケットすべてに対して個別に行う必要があります。また、listenを行ったソケットも使い終わったらcloseします。

では、実際に、サーバとクライアントのプログラムの例を見てみましょう。

まず、**リスト6**はクライアント側の例です。自分自身を示すIPアドレスである127.0.0.1(localhost)の11111番ポートに対してTCP接続を確立したあとに、サーバからのデータを待ちます。サーバからのデータを受け取ると、標準出力へと受信データを表示して終了します[注14]。

▼リスト6　クライアント側の通信プログラム例

```
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

int
main()
{
 struct sockaddr_in server;
 int sock;
 char buf[32];
 int n;

 /* ソケットの作成 */
 sock = socket(AF_INET, SOCK_STREAM, 0);

 /* 接続先指定用構造体の準備 */
 server.sin_family = AF_INET;
 server.sin_port = htons(11111);
 /* 127.0.0.1はlocalhost */
 inet_pton(AF_INET, "127.0.0.1", &server.sin_addr.
s_addr);

 /* サーバに接続 */
 connect(sock, (struct sockaddr *)&server,
sizeof(server));

 /* サーバからデータを受信 */
 memset(buf, 0, sizeof(buf));
 n = read(sock, buf, sizeof(buf));

 printf("%d, %s\n", n, buf);

 /* socketの終了 */
 close(sock);

 return 0;
}
```

リスト7は、サーバ側のサンプルプログラムです。サーバ側は、ポート11111番で2本の

▼リスト7　サーバ側の通信プログラム例

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

int
main()
{
 int sock0;
 struct sockaddr_in addr;
 struct sockaddr_in client;
 socklen_t len;
 int sock1, sock2;

 /* ソケットの作成 */
 sock0 = socket(AF_INET, SOCK_STREAM, 0);

 /* ソケットの設定 */
 addr.sin_family = AF_INET;
 addr.sin_port = htons(11111);
 addr.sin_addr.s_addr = INADDR_ANY;

 bind(sock0, (struct sockaddr *)&addr,
sizeof(addr));

 /* TCPクライアントからの接続要求を待てる状態にする */
 listen(sock0, 5);


 /* TCPクライアントからの接続要求を受け付ける(1回目) */
 len = sizeof(client);
 sock1 = accept(sock0, (struct sockaddr *)&client, &len);

 /* 6文字送信('H', 'E', 'L', 'L', 'O', '\0') */
 write(sock1, "HELLO", 6);

 /* TCPセッション1の終了 */
 close(sock1);


 /* TCPクライアントからの接続要求を受け付ける(2回目) */
 len = sizeof(client);
 sock2 = accept(sock0, (struct sockaddr *)&client, &len);

 /* 5文字送信('H', 'O', 'G', 'E', '\0') */
 write(sock2, "HOGE", 5);

 /* TCPセッション2の終了 */
 close(sock2);


 /* listen するsocketの終了 */
 close(sock0);

 return 0;
}
```

注14) 本稿のCサンプルはMac OS Xで書いています。Linuxの場合はsocklen_t部分が異なるので適宜修正してお試しください。エラー処理は割愛しています。

TCP接続を受け付けてから終了します。1本目のTCP接続に対しては"HELLO"と書き込み、2本目には"HOGE"と書き込みます。

本稿では、TCPサーバ側サンプルで、あえて2本のTCP接続を受け付けるように書いてみました。これは、bindしたうえでlistenしている待ち受け用のソケットと、acceptによって新たに生成された通信を実際に行うためのソケットが違うものであることを理解していただくためです。このように、サーバ側は待ち受けしているポート番号に対してTCP接続をしてくるクライアントとのTCP接続を次々と確立できるのです。

TCPの送信元ポート番号を設定する

bindの部分をもう少し掘り下げてみましょう。bindは「名前が付いていないソケットに名前を付ける」と解説されますが、TCPにおける「名前」とは送信元もしくは接続を受け付けるIPアドレスと送信元ポート番号です。listenの前にbindを行ったうえでacceptを行えば、bindによって設定されたTCPポート番号でTCP接続要求を待つようになります。

connectの前にbindを行えば、acceptを行うサーバ側が認識するクライアント側のTCPポート番号は、connect前に行ったbindによって設定されたTCPポート番号になります。

なお、ソケットに対してTCPのポート番号が設定されるのは、bindを行ったときだけではないのでご注意ください。connectとlistenは、それらのシステムコールを利用する前にbindが行われていなければ、自動的にカーネルが送信元ポート番号を設定する機能を持っています。

たとえば、TCP SYNを送信するクライアント側にも送信元ポート番号はありますが、一般的なプログラミング手法では、bindを行わずにconnectが行われます。これにより、bindによる「名前付け」が行われずにTCP SYNが送信されるわけですが、TCP SYNを送信するには、何らかの送信元ポート番号が必要です。bindを行わずにconnectができるのは、connectを利用する前にbindが行われていなければ自動的にカーネルが設定を行ってくれているためです。

あと、多少マニアックになってしまいますが、TCP SYNを受け付けるサーバ側においても、bindを行わずにlistenを行うことで、カーネルが自動的に設定したポート番号でTCP SYNを待ち受けることができます。そういう書き方をしたときには、listen時点でどのようなポート番号になるのかを事前に予測するのが難しいため、listen後にgetsocknameというシステムコールを利用して、設定されたポート番号を調べます。

UDP(User Datagram Protocol)

TCPはインターネットの初期から存在していました。そもそも、当初はデータの到着が保証されるTCPだけですべての通信を実現しようとしていました[注15]。

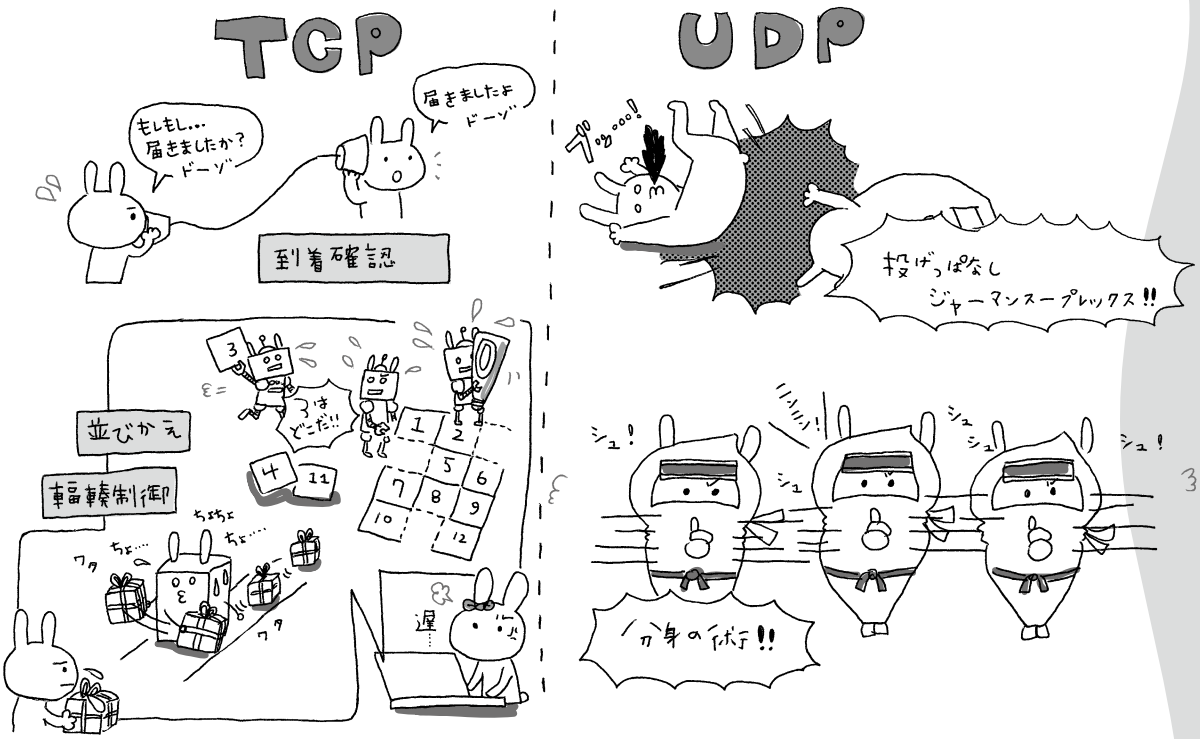
しかし、それでは不都合がありました。音声通信などでは、すべてのデータが正しく到達することよりも、多少のロスが存在したとしても短時間でパケットが到達することなどが必要であったため、UDP(User Datagram Protocol)があとから考案されたとあります。TCPのIPプロトコル番号[注16]が6、UDPが17であることからも、UDPがあとから考案されたことがわかります。このように、ソケットを利用した通信はTCPに限定されるものではありません。

インターネットにおける通信の大半がTCPによるものですが、音声や動画の通信であったり、リアルタイム性が要求されたり、TCPほ

注15) 今のインターネットの前身であるDARPA Internetのしくみを解説した論文が1988年に書かれていますが、その論文の中にもTCPが語られています。(http://portal.acm.org/citation.cfm?id=205458)D.D.Clark. The Design Philosophy of the DARPA Internet Protocols, Proceedings of ACM SIGCOMM, Pages 106-114, September 1988.

注16) IPヘッダに書き込まれた識別番号(http://www.iana.org/assignments/protocol-numbers/protocol-numbers.xhtml)

▼図24　投げっぱなしジャーマンスープレックスのようなUDP



どの機能が不必要であったり、マルチキャストやブロードキャストによって同時に多数の相手と通信したいときなどに使われるプロトコルとして「UDP」があります。のちほど紹介するDNSに対する問い合わせにもUDPが使われています。

投げっぱなしジャーマンスープレックス!

UDPはデータが宛先に届いたかどうかを関知しないため、データの到着を保障しない点がTCPと異なります。TCPでは相手にデータが届いたかどうかなどを含めて丁寧にデータ確認する一方で、UDPパケットの送信側は「投げっぱなしジャーマンスープレックス！[注17]」という感じで、「投げた後は知りません」というスタンスです。TCPとUDPを比べると**図24**のようになります。

このため、UDPを使った通信を前提とするプログラムを書く場合には、パケットがネットワークの途中で消えてしまうことも想定する必要があります。「投げっぱなしジャーマンスープレックス！」なスタンスには利点もあります。TCPでは、相手に対してデータが到着したかどうかを確認したり、並び替えに対処するためにパケットが到着していてもユーザに渡せなかったり、輻輳制御のために送信量が制限されたりします。しかも、これらの処理はOS内部で行われるため、ユーザアプリケーションは何が起きているのか感知しにくい構造になっています。

このように、TCPでの処理はリアルタイム性を損なうことがありますが、UDPにはそれらが存在しないため、UDPはTCPと比べてリ

注17）プロレスの投げ技。試合相手を背後から抱えたまま、自分の体をブリッジ状に後方に反り、相手の後頭部から背中にかけてマットに接地させてフォールをとるのが基本型ですが、技の後半で抱えずに後方にほおり投げます。相手は受け身が取れずにダメージが大きくなるので危険な技とされます。

アルタイム性があります。UDPには複雑なしくみが存在していないので、何か特別な処理が必要である場合には、各アプリケーション実装者がそれぞれプログラムを自作する必要があります。たとえば、UDPを利用しつつ輻輳制御が必要となるような場合には、輻輳制御機構を自作する必要があります。めんどうではありますが、アプリケーションごとに各自が柔軟にしくみを実装できるという利点があります。

分身の術!

IPv4では、ユニキャスト、ブロードキャスト、マルチキャストの3種類の通信方法が存在しています[注18]。TCPは、1対1の通信だけを想定しており、IPv4ではユニキャストでのみ通信ができます。それに対してUDPは、1つのデータパケットを送ればネットワークで必要に応じて増やして送ってくるブロードキャストやマルチキャストが利用できます。

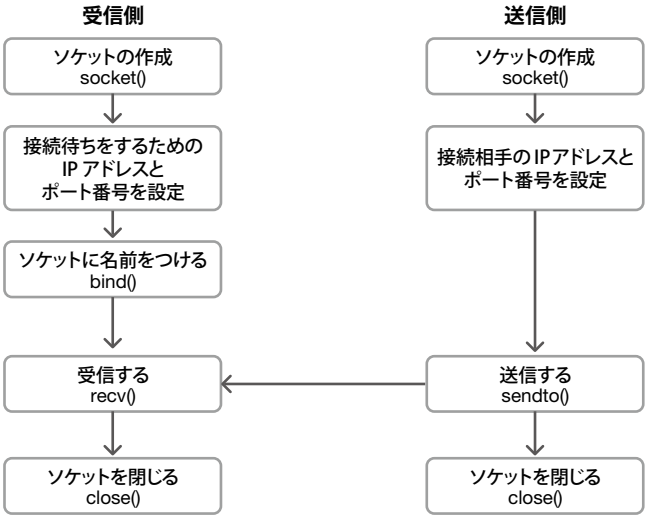
UDPを使うことで、途中経路上のルータがパケットに対して勝手に「分身の術！」といった感じで必要に応じて増えてくれます[注19]。ブロードキャストやマルチキャストを利用することにより、送信側は受信者数に関係なく必要最低限のパケットだけ送っていれば、あとはネットワークが適切に処理をしてくれるため送信側のアプリケーションの負荷を大きく軽減できます。

ブロードキャストやマルチキャストは、不特定多数に送信できるものですが、相手を直接指定せずに「必要な人が受けとって！」ということもできます。TCPを使うには通信相手を明示する必要がありますが、UDPを使うことによって、誰が受け取るかは知らないけど必要に応じて受け取ってほしい通信を実現できます[注20]。

UDPプログラミング例

次は、実際にUDPパケットの送信や受信のプログラム例を見ていきましょう。TCPでは、相手を指定したうえで接続をしてから実際の通信を行うという手順でしたが、UDPでは、**図25**のように相手を指定していきなり送信します。

▼図25　UDPを利用した送信と受信



注18) IPv6では、ユニキャスト、マルチキャスト、エニーキャストの3種類です。IPv6では、ユニキャストとエニーキャストでTCPが利用できます。ただし、昔はIPv6エニーキャストでTCPを利用できないしくみでした。

注19) ここではUDPの特徴というふうに紹介してしまっていますが、実際はブロードキャストやマルチキャストといったIPが持っている特徴をTCPが使えないだけという話であったりもします。

注20) 実際はUDPの特徴ではなくIPの特徴ですが、ここでは割愛します。

まずは、UDPパケットを送信するプログラムの例です(**リスト8**)。TCP用のソケットを作るときには、socketシステムコールの第1引数にAF_INETで第2引数にSOCK_STREAMを指定していましたが、UDPでは第1引数にAF_INETで第2引数がSOCK_DGRAMになっています。AF_INETとSOCK_DGRAMの組み合わせは、IPv4のUDPになります。

IPアドレスは自分自身を示すlocalhost(127.0.0.1)、宛先ポート番号は11111番に送信しています。送信しているデータは「'T', 'E', 'S', 'T', '\0'」という5文字です。

TCPのように、相手に接続する(connectする)という段階が存在せず、いきなりsendtoを行っているのが大きな特徴です[注21]。

このサンプルでは、UDPの送信元ポート番号はsendtoの時点で設定されています。sendtoを行う前にbindを行えば、UDP送信元ポート番号を明示的に指定できます。

次は、受信プログラムを見てみましょう。**リスト9**はUDP用のソケットを作成したうえで、それに対して、bindシステムコールでポート番号11111番という名前を付けています。bindで受信用のポート番号を設定したあとに、recv

▼リスト8　UDPパケット送信プログラムの例

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

int
main()
{
 int sock;
 struct sockaddr_storage ss;
 struct sockaddr_in *to = (struct sockaddr_in *)&ss;
 int n;

 sock = socket(AF_INET, SOCK_DGRAM, 0);

 to->sin_family = AF_INET;
 to->sin_port = htons(11111);
 n = inet_pton(AF_INET, "127.0.0.1", &(to->sin_addr));
 if (n < 1) {
  perror("inet_pton");
  return 1;
 }

 n = sendto(sock, "TEST", 5, 0, (struct sockaddr *)&to, sizeof(struct sockaddr_in));
 if (n < 1) {
  perror("sendto");
  return 1;
 }

 close(sock);

 return 0;
}
```

▼リスト9　UDPパケット受信プログラムの例

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

int
main()
{
 int sock;
 struct sockaddr_storage ss;
 struct sockaddr_in *bindaddr = (struct sockaddr_in *)&ss;

 struct sockaddr_storage senderinfo;
 socklen_t addrlen;
 char buf[2048];
 int n;

 sock = socket(AF_INET, SOCK_DGRAM, 0);
 if (sock < 0) {
  perror("socket");
  return 1;
 }

 bindaddr->sin_family = AF_INET;
 bindaddr->sin_port = htons(11111);
 bindaddr->sin_addr.s_addr = INADDR_ANY;

 if (bind(sock, (struct sockaddr *)bindaddr, sizeof(struct sockaddr_in)) != 0) {
  perror("bind");
  return 1;
 }

 addrlen = sizeof(senderinfo);
 n = recvfrom(sock, buf, sizeof(buf) - 1, 0,
       (struct sockaddr *)&senderinfo, &addrlen);

 write(fileno(stdout), buf, n);

 close(sock);

 return 0;
}
```

注21) UDPソケットに対してconnectを使うこともできますが、本稿では割愛します。

fromシステムコールでUDPパケットの到着を待っています。パケットを受信すると、writeシステムコールを使用して標準出力に受信した内容をそのまま表示しています。

このようにUDPプログラミング例をご覧いただくと、本特集の前半で紹介したソケットプログラミングが、実際は「TCPソケットプログラミング」であることがわかると思います。

TCPやUDP以外にも、ソケットにはさまざまな種類があります。本稿では、割愛しますが、IPパケットを直接作成できるようなソケットや、ホスト内のプロセス間で通信するためのソケットなど、実際にはさまざまな種類のソケットがあるので、興味がある方はぜひいろいろ調べてみてください。

UDPでの返信の例

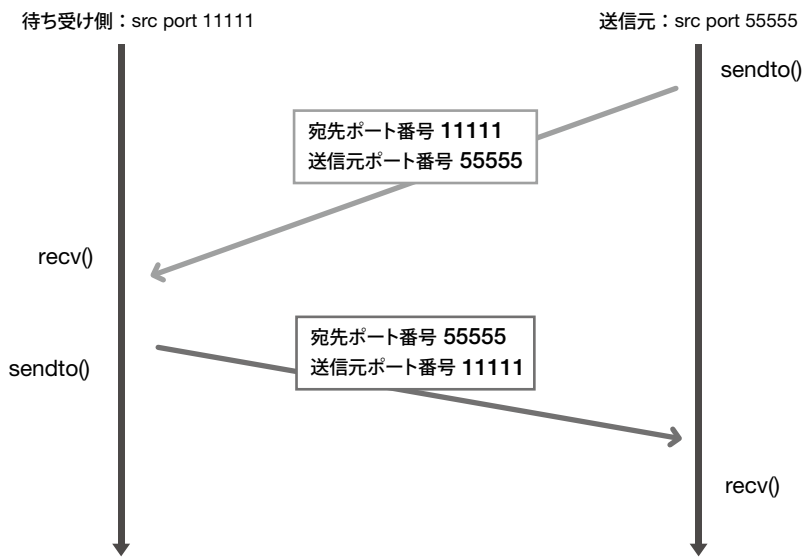
TCPでは、データを双方向にやりとりすることができます。たとえば、Webで利用されるHTTPのように、TCP接続確立後に、受け取ったデータに応じて返答するようなこともできます。UDPソケットからパケットを受け取ったソケットを使って送信してきた相手にパケットを送ることで、UDPでも「何かを受け取ったら、その相手に返答する」ということができます。

図26は、IPアドレス203.0.113.8の送信側が、送信元ポート番号55555のUDPパケットを、IPアドレス192.0.2.9のUDPポート11111番宛に送信しています。IPアドレス192.0.2.9側は、UDPパケットをrecvfromシステムコールで受信します。recvfromシステムコールの第5引数は、UDPパケットの送信元情報が含まれているので、そこに記載された情報を元にUDPパケットを送信しています。

このように、UDPパケットに記載された送信元のIPアドレスとポート番号を、そのまま宛先情報として利用して返答するという実装方法がさまざまなところで行われています。たとえば、UDPを利用してDNSへの問い合わせを行うときも、このような方式でDNSからの応答が送信されています[注22]。

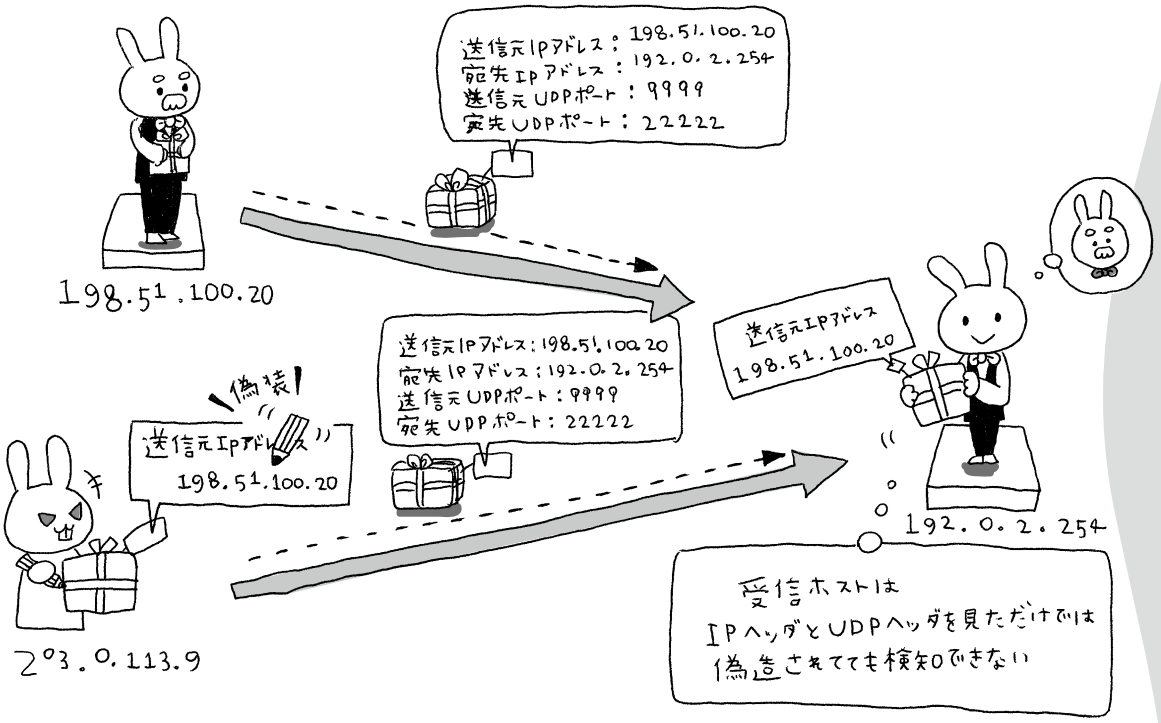
この方式の問題点は、外部のユーザが偽造したものを受け取ったとしても気が付けない場合があるというところです。

▼図26　UDPでの返信の例



注22) DNSへの問い合わせがTCPで行われることもあります。

▼図27　UDPパケットの偽造



TCPでは、最初に3 way handshakeがあり、シーケンス番号が同期していないとパケットが受け付けられないのですが、UDPそのものにはそういったしくみがないので、TCPと比べて偽造パケットを忍び込ませるハードルが低くなっています(**図27**)。

このような攻撃を防ぐためには、アプリケーションを実装する人が、偽造されたパケットを無視できるようなしくみを独自に実装する必要があります[注23]。

注23) RFC 2827(BCP 38)とRFC 3704(BCP 84)に関しては、本稿では割愛させてください。

Part 3　おさえておきたいDNSのしくみ

IPアドレスと「名前」

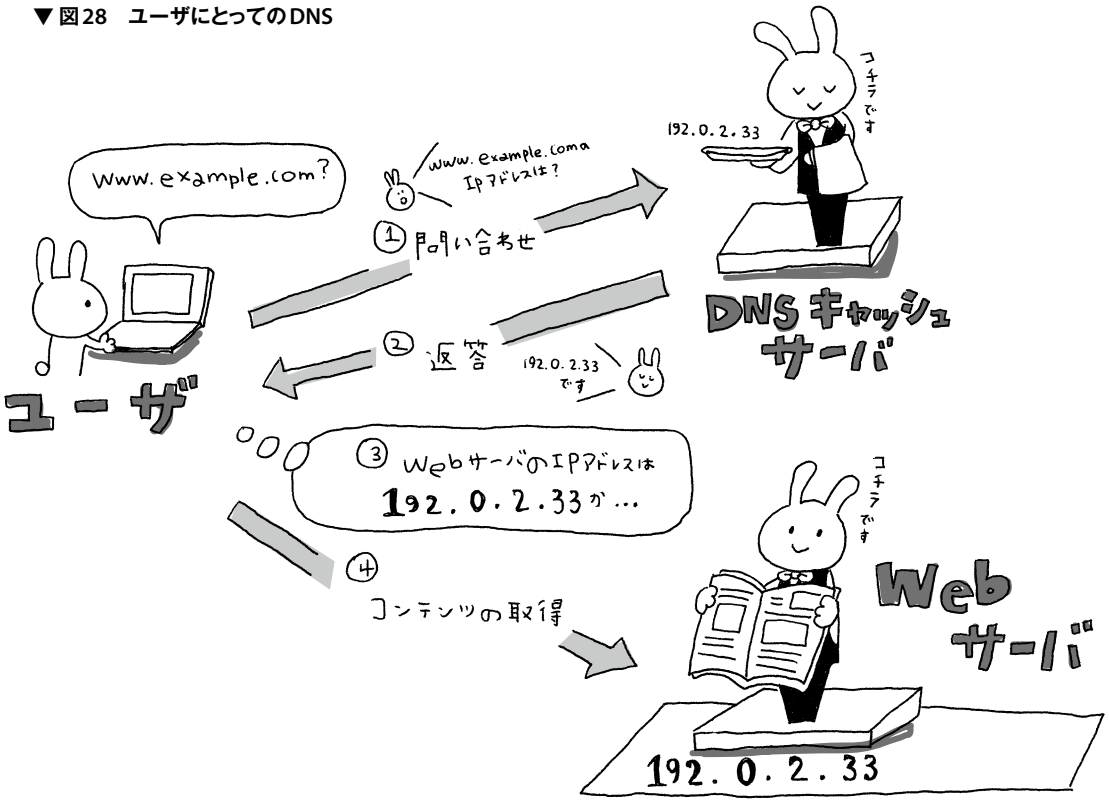
これまで、IPアドレスを直接指定するという前提でソケットプログラミングを紹介してきました。インターネットにおける識別子は、IPアドレスであり、通信はIPアドレスを基に行われます。しかし、数値の羅列であるIPアドレスは人間にとってわかりにくいので、www.example.comなどの「名前」を使ってユーザが通信相手を指定するのが一般的です。とはいえ、名前を直接IPパケットのヘッダに書き込むことはできないので、名前をIPアドレスへと変換するしくみが必要になってきます。

次は、こういった「名前」とは何かと、「名前」をIPアドレスへと変換するしくみであるDNS(Domain Name System)を紹介します。

DNSのしくみ

現在のインターネットにおける一般ユーザに

▼図28　ユーザにとってのDNS



とってのDNSは、DNS全体のしくみではなく、最寄りのDNSキャッシュサーバを示すことが多いです。たとえば、PCやスマホで「DNSの設定」といえば、その機器が利用するDNSキャッシュサーバのIPアドレスです。

一般的なユーザがWebを見るとき**図28**のように、DNSキャッシュサーバに名前解決を依頼し、返ってきたIPアドレスを使ってWebサーバとの通信を行います。

ユーザにとっては、DNSキャッシュサーバは「いろいろ知っているすごいサーバ」のように見えますが、実際はそうではありません。名前に対応するIPアドレスの情報を持っているDNSサーバは、「権威DNSサーバ」と呼ばれていますが、DNSキャッシュサーバはユーザに代わって権威DNSサーバへの問い合わせを行っているだけなのです。ユーザのために、裏でいろいろと頑張っているわけです。

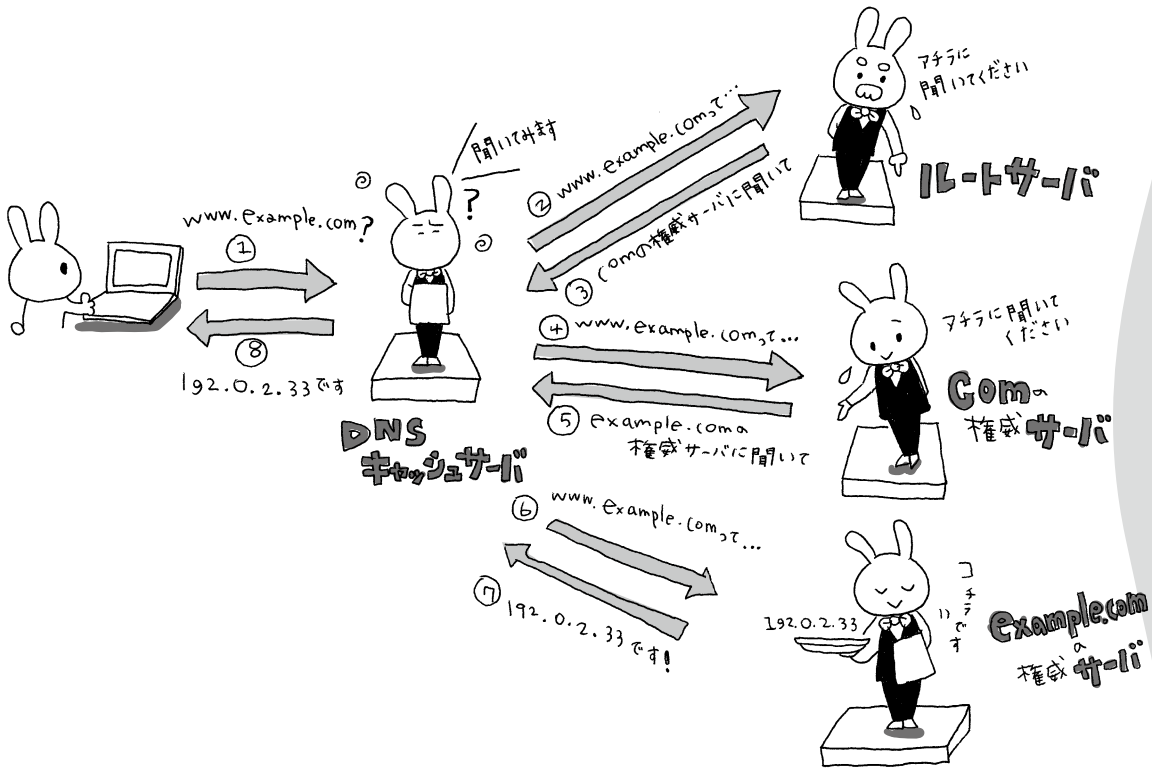
DNSの問い合わせのしくみ

次は、DNSキャッシュサーバがどうやって名前解決を行っているのかを見ていきます。

ユーザが解決したかった名前が「www.example.com」であったとしましょう。まず、DNSキャッシュサーバという名前からもわかるように、DNSキャッシュサーバは以前に行われた解決結果を可能な範囲で保存しています。キャッシュが存在していれば、ユーザに対してそれをそのまま返します。DNSキャッシュサーバがキャッシュを何も持っていない場合は、**図29**のようになります。

①まず、ユーザの手元からDNSキャッシュサーバに対する問い合わせが行われます。このとき、DNSキャッシュサーバはwww.example.comに関連するいっさいのキャッシュを保持していないとします。

▼図29　DNSのキャッシュサーバのしくみ



②DNSキャッシュサーバは、まず、ルートサーバと呼ばれるDNSサーバに「www.example.comを教えて！」と問い合わせます。ルートサーバは、com、net、jpなどのトップレベルドメインの権威DNSサーバに関する情報を持っています。

③www.example.comの名前解決問い合わせを受け取ったルートサーバは、「.comの権威DNSサーバに聞いて！」と応えます。

④ルートサーバから.comの権威DNSサーバのIPアドレスを教えたもらったDNSキャッシュサーバは、あらためて.comの権威DNSサーバに「www.example.comを教えて！」と問い合わせます。

⑤すると.comの権威DNSサーバは、「example.comの権威DNSサーバに聞いて！」と応えます。

⑥そこでDNSキャッシュサーバは、今度はexample.comの権威DNSサーバに「www.example.comを教えて！」と問い合わせます。

⑦するとexample.comの権威DNSサーバは、www.example.comに対応するIPアドレスを返してくれます。

⑧最終的にwww.example.comのIPアドレスを得たDNSキャッシュサーバは、DNSクライアントにその結果を通知します。

DNSでは、このように繰り返しさまざまなサーバに質問を投げかけながら、最終的に情報を知っている権威DNSサーバを探します。DNSのこの検索方法を「再帰検索」といいます。DNSキャッシュサーバにキャッシュがない場合には、再帰検索によって裏でさまざまな問い合わせが発生するので、応答に時間がかかります。逆に、キャッシュが保持された状態では名前解決が早くなります。

Webを見るとき、最初の1回だけが妙に時間がかかって次からは早くなることがありますが、DNSを利用した名前解決にかかる時間も、その一因です。再帰検索の特徴として「各権威

DNSサーバが自分が把握すべき範囲を知っている」という点が挙げられます。すべての情報を誰か1人が知っているのではなく、各自が分担して自分の責任範囲を定義し、知っている範囲内で次を教えるしくみです。

このように、特定個所に負荷が集中することを避け、分散管理ができることを目指したしくみであるからこそ、世界規模のネットワークになり得たとも言えます。

IPv4とIPv6とDNS

IPv4とIPv6には直接的な互換性がありません。しかし、状況によってIPv4が使われたり、IPv6が使われたりという感じになるので、一般ユーザにはあたかも互換性があるような感覚に陥ります。そのように感じるのは、インターネットの名前空間に対してIPv4とIPv6の両方を関連付けられるようになっているためです。

ユーザがIPv4を利用するのか、それともIPv6を利用するのかを大きく左右するのがDNSであるというわけです。たとえば、www.example.comという名前に対応するIPアドレスを調べたときに、IPv6アドレスがついていればユーザはIPv6を利用した接続を試みたうえで、失敗したらIPv4を使うといった挙動を示す場合があります。

これは、DNSに対してどのようなIPアドレスが設定されているのかで、ユーザが選択する通信手法が変わることを意味します。しかし、DNS側がユーザの選択をすべて制御しているわけでもありません。DNSに対して、IPv4とIPv6の両方を1つの問い合わせメッセージで問い合わせることができない仕様になっているので、ユーザ側はDNSに対してIPv4用とIPv6用の問い合わせを個別に行います。

www.example.comに対する問い合わせであれば、「www.example.comのIPv4アドレスを教えてください」と「www.example.comのIPv6アドレスを教えてください」というふうに、別々の問い合わせをするわけです。

このように、IPv4とIPv6の問い合わせを両方するのかどうかを判断するのは、あくまでユーザ側ですし、返ってきた応答に含まれるIPアドレスのうちどれを通信に利用するのかを判断するのもユーザ側です。そして、ユーザが使うプログラムを書く人がどのようにその部分を実装するのかによっても、そのあたりの事情は変わってきます。

getaddrinfo

最近の便利なライブラリを利用していると、アプリケーションが名前解決を行っていることを忘れがちです。しかし、ソケットを利用したプログラミングでは、名前解決部分は非常に重要な要素です。

ソケットを利用したプログラミングでの名前解決は、getaddrinfoを利用します。昔はgethostbynameというAPIが利用されていましたが、gethostbynameはIPv4しか扱えないため、現在はIPv4とIPv6の両方が利用可能であるgetaddrinfoを利用することが推奨されています。では、getaddrinfoを利用したサンプルプログラムをご覧ください(**リスト10**)。

getaddrinfoに対して渡す引数は、名前解決を行う文字列だけではなく、そのあとにソケットに対して設定したいポート番号なども含まれています。**リスト10**では、getaddrinfoが成功したら、その結果に含まれるパラメータを使ってソケットを作成し、そのソケットに対してconnectを行っています。connectに成功すれば、そのソケットを利用しますが、失敗すればソケットを閉じてgetaddrinfoが返した次の結果を試してみます。

getaddrinfoのAPIがソケットの種類やポート番号などの情報を引数として渡すようにできており、getaddrinfoが返す結果にもそれらが含まれるので、その結果をそのまま利用してsocket、connect、bindなどのシステムコールを使えるようになっています。

実は、このサンプルのような書き方をしていると、getaddrinfoが返す順番によってユーザ

がどのようなIPアドレスで通信を行うのかが変わってきます[注24]。そのため、getaddrinfoが結果を返す順番というのは非常に重要な要素となるわけです。DNSによって得られる名前解決結果が複数存在するときに、getaddrinfoがどのような順番で結果を返すのかに関しての標準がRFC 6724[注25]に記述されているので、興味がある方はぜひご覧ください。

▼リスト10　getaddrinfoを利用したサンプルプログラム

```
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>

int
main()
{
 char *hostname = "www.example.com";
 char *service = "http";
 struct addrinfo hints, *res0, *res;
 int err;
 int sock;

 memset(&hints, 0, sizeof(hints));
 hints.ai_socktype = SOCK_STREAM;
 hints.ai_family = PF_UNSPEC; /* IPv4とIPv6両方を取得 */
 if ((err = getaddrinfo(hostname, service, &hints,
&res0)) != 0) {
  printf("error %d\n", err);
  return 1;
 }

 for (res=res0; res!=NULL; res=res->ai_next) {
  sock = socket(res->ai_family, res->ai_socktype,
res->ai_protocol);
  if (sock < 0) {
   continue;
  }

  if (connect(sock, res->ai_addr, res->ai_addrlen)
!= 0) {
   close(sock);
   continue;
  }

  break;
 }

 if (res == NULL) {
  /* 有効な接続ができなかった */
  printf("failed\n");
  return 1;
 }

 freeaddrinfo(res0);

 /* ... */
 /* ここ以降にsockを使った通信を行うプログラムを書いて
ください */
 /* ... */

 return 0;
}
```

Part 4 自分でネットワークを確認してみよう！

ネットワークを体感する

プログラムを書かなくても、ネットワークを体感できます。ネットワークを体感するためにお勧めなのが、次の3つのコマンドです。これらは主要なOSに最初からインストールされています。

・ping/ping6
・traceroute/tracert/traceroute6
・dig

これらのツールは「少し試してみる」ためだけのものではありません。ネットワークを使ううえで非常に有用で、重要なツールでもあります。たとえば、ネットワークが何かおかしいと思ったときなどに、これらのツールを使って現状把握や問題の切り分けを行うことができます。覚えておいて損のないツールですので、ぜひ試し

注24）ユーザがシステム内でgetaddrinfoが返す優先順位を設定できます（man gai.conf参照）。なお、余談ではありますが、getaddrinfoとgethostbynameはカーネル内部に実装しなくても実現可能であるため、システムコールではなく、C言語用の基本ライブラリ（libc）の一部として提供されています（man 3 getaddrinfo参照）。たとえば、ファイルを扱うためのfopenやfclose、標準出力に文字列を表示するprintfなどもlibcの一部ですが、getaddrinfoとgethostbynameも同様の扱いです。

注25）RFC 6724（Default Address Selection for Internet Protocol Version 6 (IPv6), http://tools.ietf.org/html/rfc6724）

てみてください。

ping / ping6

まず、最初に紹介するのが最も原始的であり、一般的なネットワークコマンドであるpingです(ping6は、UNIX系OSでのIPv6用pingです)。pingは、指定した宛先までパケットが届いているのかどうかを推測するために使えるツールです。やっていることは非常に単純で、パケットを相手に送り付けて、相手はパケットを送り返すというものです。pingの名前も由来は「ping pong」から来ています。

百聞は一見にしかず、ですので、「なぜ動くか」の先にどうやったら使えるかを説明します。まず、Windowsであれば最初にコマンドプロンプトを起動してください。Mac OS Xなら「ターミナル」を実行してください。LinuxやFreeBSDなどのUNIX系OSを利用しているときは、何らかの方法でコマンドラインを出してください[注26]。

次に、「ping ホスト名」とプロンプトが表示されたコマンドラインで打ってください。

「ホスト名」の部分は適当に思いつくホスト名か、もしくはIPアドレスを使ってください。筆者の手元の環境では、192.168.0.1がルータですので**図30**では「ping 192.168.0.1」とします。

図30は、pingが成功している例を示しています。では、この成功例は何を示しているのでしょうか？　まず、前半の行を見ると56バイトのデータパケットを送ってpingを行っているのがわかります。その後の行を見ると、pingに対する6回の応答が約4msで返ってきているということもわかります。TTLというのはIPパケットのTime To Liveです。このTTLとは、パケットがルータによって転送されてもよい回数がIPヘッダに記述されたものです。

次に、失敗している例を見たいと思います。今度は、存在しないホストに対してpingを行います。筆者の手元の環境では、192.168.0.200というホストは存在しません。今度はそこに向けてpingを行います(**図31**)。

図31を見ると「Request timeout」が4回続いています。これは、「pingの応答を待ったけど帰ってこなかった」ということを示しています。pingは、ICMP(Internet Control Message Protocol)というプロトコルを利用しています。ICMPは、エラーメッセージや制御メッセージを転送するIP上のプロトコルです。ICMPのIPプロトコルIDは「1」です。これは、TCPのプロトコル番号である6よりも小さい数字です。このことからも、ICMPが非常に根本的なプロトコルであることがわかります。

ICMPにはさまざまな機能がありますが、pingはそのうちのEchoメッセージとEcho Replyメッセージを利用しています。Echoメッセージとは、「Echoを返してくれ」というメッセージで、Echo Replyメッセージは「Echoに対する応答」です。pingは、Echoメッセージを送信し、Echo Reply

▼図30　Mac OS Xでのpingコマンド成功例

```
% ping 192.168.0.1
PING 192.168.0.1 (192.168.0.1): 56 data bytes
64 bytes from 192.168.0.1: icmp_seq=0 ttl=255 time=3.895 ms
64 bytes from 192.168.0.1: icmp_seq=1 ttl=255 time=3.986 ms
64 bytes from 192.168.0.1: icmp_seq=2 ttl=255 time=3.890 ms
64 bytes from 192.168.0.1: icmp_seq=3 ttl=255 time=4.669 ms
64 bytes from 192.168.0.1: icmp_seq=4 ttl=255 time=3.951 ms
64 bytes from 192.168.0.1: icmp_seq=5 ttl=255 time=3.973 ms
^C
--- 192.168.0.1 ping statistics ---
6 packets transmitted, 6 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 3.890/4.061/4.669/0.274 ms
```

注26)というよりUNIX系OSをご利用の方々なら、ここら辺は説明する必要はないだろうと推測されます(汗。

▼図31　Mac OS Xでのpingコマンド失敗例

```
% ping 192.168.0.200
PING 192.168.0.200 (192.168.0.200): 56 data bytes
Request timeout for icmp_seq 0
Request timeout for icmp_seq 1
Request timeout for icmp_seq 2
Request timeout for icmp_seq 3
^C
--- 192.168.0.200 ping statistics ---
5 packets transmitted, 0 packets received, 100.0% packet loss===
```

メッセージが返ってくるのを待ちます。pingコマンドは、Echo Replyメッセージを確認すると、Echo Replyを受け取るのにかかった時間を計測し表示しています。

traceroute / tracert / traceroute6

次は、ネットワークの向こうにいるホストまでの経路を知ることができるtracerouteコマンドです(UNIX系OSではtraceroute、Windowsではtracert、UNIX系でのIPv6用がtraceroute 6)。tracerouteは、指定した宛先までの途中経路を表示してくれます。名前も「trace」と「route」と、経路を探索するという意味を持っています。

では、実際にtracerouteコマンドを試してみましょう。たとえば、www.example.comまでtracerouteした場合には**図32**のような出力になります。

tracerouteの結果では、指定した宛先までの途中ルータがわかります。また、それぞれまでのRTT(Round Trip Time)も表示されます。IPアドレスに対応する名前がDNSの逆引きで取得できない場合には、IPアドレスのまま表示されます。この例では、目的のホストまで6ホップあることがわかります。

tracerouteの原理

では、なぜtracerouteは動作するのかという説明をしたいと思います。インターネットには、特定のパケットが永遠にネットワーク内を徘徊しないように、各パケットに安全装置があります。安全装置は、IPヘッダ内にTTL(Time To Live)というフィールドを作ることによって実現しています。このTTLフィールドは、ルータによりパケットが転送されるたびに値が1つ引かれます。IPパケットの転送が繰り返されると、TTLの値は転送ごとに減っていきます。最終的にIPパケットが宛先まで届けば良いのですが、宛先に届く前にTTLが0になってしまうとIPパケットは消滅します。しかし、単に消滅してしまうと何が起きたのかがわからない場合があるので、ルータはTTLが0のIPパケットを破棄するときにはIPパケットを送った送信元に対してICMP Time Exceededという種類のICMPパケットを送信します。

tracerouteは、このICMP Time Exceededを利用しています。意図的にTTLの値を小さくして、ICMP Time Exceedが発生する環境を作成しているのです。

では、実際のtracerouteの動作を見ていきましょう。tracerouteは、まず最初にTTL=1でIPパケットを送信します。すると、パケットが一度転送された状態でTTLが0となり、tracerouteを実行した機器の隣のルータからICMP Time Exceedが返ってきます。次に、

▼図32　traceroute実行例

```
% traceroute www.example.com
traceroute to www.example.com (93.184.216.118), 64 hops max, 52 byte packets
 1  192.168.0.1 (192.168.0.1) 1.435 ms  0.744 ms  0.952 ms
 2  mito06.ap.XXXX.ne.jp (203.0.118.1) 12.963 ms  10.456 ms  10.793 ms
 3  09ig2-0.net.XXXX.ne.jp (192.0.2.1) 10.828ms  11.121 ms  11.434 ms
 4  nrt1.asianetcom.net (202.172.1.181) 14.400 ms  14.238 ms  15.390 ms
 5  sj1.asianetcom.net (202.147.51.127) 120.861 ms  121.717 ms 119.551 ms
 6  www.example.com (93.184.216.118) 131.956 ms  132.308 ms  129.135 ms
```

TTL=2でIPパケットを送信します。今度は、隣の隣にいるルータがICMP Time Exceedを返してきます。

このように順次TTLを上げていき、徐々に届く範囲を広げていきます。最終的にIPパケットが目的の宛先に到着するまで送信するTTLは上がっていきます。

最後の1ホップ

このようにTTLの値を利用して1ホップずつ把握していけるtracerouteですが、このままでは最終的な目的地に到達したときに困ります。本来の目的地についたということは、TTLとして十分な値が設定されたということだからです。ICMP Time Exceedが送信されるのはTTLが0となった場合であるため、パケットが目的地に到達した場合にはICMP Time Exceedは送信されません。

そのため、tracerouteは最後の1ホップだけはICMP Time Exceedを利用しません。最終的な目的地にIPパケットが到着したことを知る手段で一般的なものは2つあります。1つは、tracertによって送信されるIPパケットをICMP Echoパケットにすることです。それにより、ICMP Echoパケットを受け取った宛先はICMP Echo Replyを返してくれます。

2つめの方法は、tracerouteによって送信されるIPパケットをUDPにすることです。UDPの宛先ポート番号は、宛先でサービスが存在しないものを利用します。そうすることにより、宛先にUDPパケットが届いたときに、宛先ホストはICMP Port Unreachを送り返してくれます。ICMP Port Unreachは、「そのポートは開いていないよ」と教えてくれるICMPメッセージです。

このような方法でtracerouteは途中経路を計測しています。tracerouteはインターネットのしくみを巧みに利用したアプリケーションであり、ネットワークのトラブルシューティングにはなくてはならないものです。

dig / nslookup

digやnslookupは、DNSに対しての問い合わせをします。本稿ではdigを説明します。筆者がdigコマンドをよく使うのは、何らかの障害が発生しているときです。「あれ？　何が起きているのだろう？　そもそも、この名前と通信を行おうとしたときに、どことつながるのだろう？」といったことをdigコマンドで調べられます。そのほか、そのドメイン名で運用されているWebサーバなど、どのようなホスティングサービスを利用しているのかを推測したいときや、DNSの設定を確認するときなどにも利用できます。

digは、問い合わせを行うDNSメッセージの詳細を指定できます。digの出力結果も、返ってきたDNSメッセージの詳細を知ることができる表示になっています。

まずは、www.example.comのIPv4アドレスを調べる例です(**図33**)。IPv6アドレスを示すAAAA(「クアッドA」と読みます)レコードを問い合わせた場合には、**図34**のようになります。

図33、**図34**のサンプルは、digコマンドを実行しているシステムに対して設定してあるキャッシュDNSサーバへの問い合わせ結果です。問い合わせを行うDNSサーバを指定するには、「dig @サーバのIPアドレス www.example.com」のように使います。

ほかに良く使うのが「+trace」オプションです。ルートサーバから目的とする名前までの権威DNSサーバを順次調べたいときに「dig +trace www.example.com」というふうにできます。そのほかにも、digを使っていろいろな問い合わせができます。さらに興味があるかたは、man digをご覧ください。

ソースコードを読んでみよう

pingとtracerouteは、さまざまなオープンソースバージョンが存在していますし、大学の授業で宿題として自作することがあるので、検

索エンジンで容易に発見できます。

たとえば、「ping source code」や「traceroute source code」などのキーワードで検索してみて下さい。そのうえで、さらに理解を深めるために、それらを自作してみるのも楽しいと思います。digは、ISCのBINDに含まれています[注27]。

おわりに

本稿は、「ソケットとポート」に着目しつつ、インターネットを解説することに挑戦してみました。実際にソケットを使ってプログラミングを行うには、ここで解説している情報だけでは足りないため、さらに色々と調べながらコードを書くことになると思いますが、本稿がその一助となれば幸いです。SD

COLUMN 「乱用禁止のpingとtraceroute」

pingやtracerouteの実験をするときに注意すべき重要なことがあります。覚えたら使いたくなるのが人情ですが、自分で管理していないホストへむやみに大量のpingやtracerouteをしないでください。pingやtracerouteで、パケットを大量に送られるということを「攻撃をされた」と受け取る人もいます。トラブルに巻き込まれないためにも、他人のホストに対して、むやみやたらに何かをするのは控えましょう。

もう一点注意が必要なのが、セキュリティ上の理由でpingやtracerouteなどができなくなっている環境もあるということです。たとえば、会社のネットワークなどでは、セキュリティ上の理由でICMPパケットの通過を許可していない場合があります。そのようなネットワークでは、ホストに不具合がなくてもpingに対する応答は返ってきません。また、セキュリティソフトウェアやパーソナルファイアウォールにより、ICMPパケットがホスト側でフィルタリングされている場合も考えられます。

▼図33　dig実行例

```
% dig www.example.com

; <<>> DiG 9.9.5 <<>> www.example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR,
id: 22000
;; flags: qr rd ra; QUERY: 1, ANSWER: 1,
AUTHORITY: 2, ADDITIONAL: 4

;; QUESTION SECTION:
;www.example.com.   IN  A

;; ANSWER SECTION:
www.example.com. 40084 IN A 93.184.216.119

;; AUTHORITY SECTION:
example.com. 5547 IN  NS  b.iana-servers.net.
example.com. 5547 IN  NS  a.iana-servers.net.

;; ADDITIONAL SECTION:
a.iana-servers.net. 1788 IN A    199.43.132.53
a.iana-servers.net. 1788 IN AAAA 2001:500:8c::53
b.iana-servers.net. 1095 IN A    199.43.133.53
b.iana-servers.net. 1095 IN AAAA 2001:500:8d::53

;; Query time: 22 msec
;; SERVER: 192.168.0.1#53(192.168.0.1)
;; WHEN: Wed Mar 12 13:07:47 JST 2014
;; MSG SIZE  rcvd: 185
```

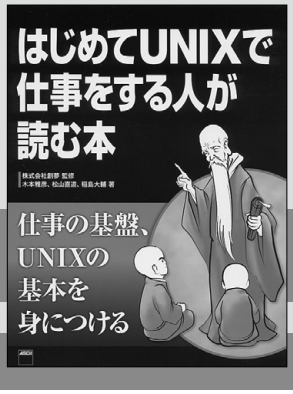
▼図34　dig実行例（IPv6アドレスの場合）

```
% dig aaaa www.example.com

; <<>> DiG 9.9.5 <<>> aaaa www.example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 49742
;; flags: qr rd ra; QUERY: 1, ANSWER: 1,
AUTHORITY: 2, ADDITIONAL: 4

;; QUESTION SECTION:
;www.example.com.   IN  AAAA

;; ANSWER SECTION:
www.example.com. 36521  INAAAA 2606:2800:220:6d:
26bf:1447:1097:aa7

;; AUTHORITY SECTION:
example.com.  1003 IN NS  b.iana-servers.net.
example.com.  1003 IN NS  a.iana-servers.net.

;; ADDITIONAL SECTION:
a.iana-servers.net. 377 IN  A    199.43.132.53
a.iana-servers.net. 377 IN  AAAA 2001:500:8c::53
b.iana-servers.net. 377 IN  A    199.43.133.53
b.iana-servers.net. 377 IN  AAAA 2001:500:8d::53

;; Query time: 19 msec
;; SERVER: 192.168.0.1#53(192.168.0.1)
;; WHEN: Wed Mar 12 13:10:55 JST 2014
;; MSG SIZE  rcvd: 197
```

注27）https://www.isc.org/downloads/bind/

BOOK no.1 はじめてUNIXで仕事をする人が読む本

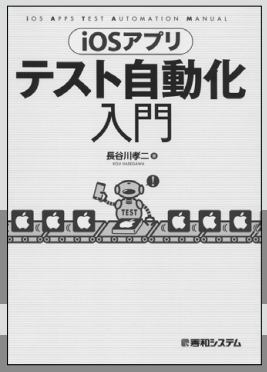
木本 雅彦、 松山 直道、 稲島 大輔 【著】 ／㈱創夢 【監修】
B5変形判、 248ページ／価格＝1,800円＋税／発行＝KADOKAWA
ISBN＝978-4-04-891392-8



㈱創夢といえば、泣く子も黙るUNIX虎の穴といった腕利きぞろいの企業として有名。そのノウハウを学べるならば、ぜひ読まねばなるまいと正座をして精読。同社の新人研修用のテキストを下書として執筆された本書は、まさにエッセンシャルにUNIXの技術をまとめあげられていて非常に心地良い作り込み。ログイン方法の説明から始まり、vi、Emacsの使い方、そしてシェルスクリプトの書き方と進み、セキュリティはsshから綿密に。開発はCやPerlなど古いところから、新しいネタとしてGitまで幅広く取り上げている。最後にUNIXのネットワーク管理とセキュリティ。このあたりも盤石。新年度に後輩に勧める本としてふさわしい。

BOOK no.2 iOSアプリテスト自動化入門

長谷川 孝二 【著】
A5判、 256ページ／価格＝2,200円＋税／発行＝秀和システム
ISBN＝978-4-7980-4089-9



本書は、iOSアプリのユニットテスト、UIレベルのテストなどを自動化する手法を紹介する。Jenkinsを使った継続的インテグレーションも取り上げている。自動化テクニックの細かな解説というよりも、自動化を行う際の目的、検証手段、ROI（投資利益率）などの考え方を示し、それに使えるツールやフレームワークを紹介するという構成だ。ツールの使い方も基本的な手順にとどめ、そのぶんより多くのツールやテスト手法を紹介している。個々のツールについて詳細を知りたい人のために、参考資料が示されているのが助かる。「テスト自動化に取り組みたいが、どこから手をつければいいかわからない」そんな人には道しるべとなる1冊だろう。

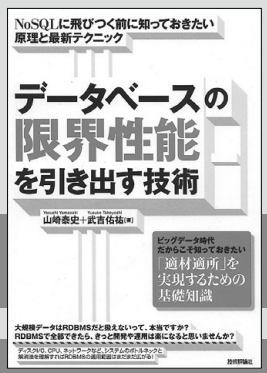
BOOK no.3 高橋信頼のOSS進化論 （日経 BP Next ICT 選書）

高橋 信頼 【著】
103ページ （推定） ／価格＝500円／発行＝日経BP社
※本書は電子書籍です。 Amazon.co.jp （Kindle版） での販売になります。

OSSの専門記者として有名なITProの記者、高橋信頼氏が2013年末に逝去された。その高橋氏が2001～2013年に同メディアで発表した記事の中から13本を選び、電子書籍としてまとめられた。過去記事をこうして読んでみると、高橋氏の視野の広さを感じさせられる。たとえば、Linuxなどの有名なOSSだけでなく、地味ながらも多くのユーザを獲得している業務アプリのOSSを紹介している。また、技術的な話題だけでなく、ライセンスにかかわる諸問題、コミュニティ運営、震災におけるITやOSSの役割といった話題も取り上げている。OSSにかかわっている人なら、一読することをお勧めする。OSSに対して広い視点が持てるようになると思う。

BOOK no.4 データベースの限界性能を引き出す技術 NoSQLに飛びつく前に知っておきたい原理と最新テクニック

山崎 泰史、 武吉 佑祐 【著】
A5判、 224ページ／価格＝2,680円＋税／発行＝技術評論社
ISBN＝978-4-7741-6364-2



話題のNoSQLとひとくくりにされることが多いMongoDB、Redis。これらはスケールアウトメリットがあるデータストアとして、迅速なサービス展開が必須なWeb業界でもてはやされている。しかし、どんなものにでもNoSQLを導入すればOKなのかと言われるとそうではない。今まで使ってきた手になじんだ道具＝RDB（Oracle）があるじゃないですか！――というのが本書の核心。まずはRDBのパフォーマンス向上のための基礎理論を解説しながら、後半ではチューニングとボトルネックの探し方を紹介。実際にSQLを書くというよりも、マネージャ層向けの理論的な裏付けのある話題が多いので、技術動向を知りたい人にもお勧め。

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

Writer くつなりょうすけ　㈱ネットワーク応用通信研究所／Twitter：@ryosuke927

新人エンジニアのなかには、UNIX系OSのコマンドラインに慣れていない人もいるのではないでしょうか？　今回はそんな皆さんのためのコマンド特集です。単発のコマンドだけではなく、よく使うオプションとともに、あるいは、便利なコマンドの組み合わせとして紹介しますので、まずは実際にコマンドラインで入力してみてください。実際に試すことで、オプションの使い方、コマンドの組み合わせ方など、勘どころがつかめますよ。

CONTENTS

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

第1章 コマンドとの付き合い方 ――書き方からパイプ／リダイレクトまで

実務で使える実践的なコマンドを具体的に見ていく前に、まずはLinuxにおけるCLI(Command Line Interface)の使い方を再確認します。コマンドの書き方だけでなく、コマンドの調べ方、パイプによるコマンドの連結、リダイレクトによる入出力の切り替えといった各コマンドの活用の幅を広げるためのテクニックについても解説します。

はじめに

LinuxなどのUNIX互換OSの普及や、Mac OS XをWebクリエイターの人たちが使っていることから、以前のWindows一色の時代に比べてCLIに触れるケースは増えたと思います。何よりこの記事を読み始めているあなたは、CLIのこと、本当は気になってるんですよね？いいんですよ、無理しなくても。

コマンドラインは全然古くなく、今でもサーバとしてUNIX系OSを使うときにはCLIが必須で、GUIがあることが稀です。CLIとGUIを比較した場合、GUIのメリットは、そこに表示されているプルダウンメニューやセレクトボックスにオプションがあるので、直感的に使えることでしょうか。逆に、CLIのメリットはマウス操作を必要とせず、キーボードで入力した命令をそのまま実行してくれること。あとは、コンピュータを使いこなしているようでカッコ良く見られそうなことかな。

CLIとGUI、それぞれに一長一短はありますが、今回はLinuxのコマンドライン初心者に向けて、コマンドのすばらしさ／便利さをお見せし、習得していただいてカッコよくなろう、という特集です。なお、ここではコマンドを実行するシェルにbashを使うことを前提にしています。

コマンドを実行する際の注意事項

コマンドを実際に実行する前に、注意することがありますので、ココロのノートに書いておきましょう。

複数のユーザが同時にシステムを利用できる「マルチユーザ環境」であるLinuxでは大きく分けて3種類のユーザがいます。管理者、システムユーザ、一般ユーザです。管理者はLinuxが稼動しているそのシステムで何でもできるユーザです。すべてのコマンドの実行、デバイスへのアクセスができ、どこのファイルでも閲覧／編集／削除ができます。一方、一般ユーザはそのユーザが許可されたアクションしかできません。システムユーザは管理者でも一般ユーザでもなく、WebサーバやSMTPサーバなどのサーバソフトウェアを動作させるために利用されるユーザです。

管理者は自らが実行したコマンドで、システムそのものを破壊することができることを覚えておいてください。これから実行するコマンドは絶対に一般ユーザで実行することにしましょう。自分が一般ユーザであるかどうかを判断するにはシェルプロンプトとコマンドから判断できます。

Linuxディストリビューションの標準シェルbashで、ログイン後の管理者と一般ユーザのそれぞれの表示を見てみましょう。次の例は、

一般ユーザのユーザ名はryosukeで、管理者のユーザ名はroot、ホストaxlにログインした場合の表示例です。

```
ryosuke@axl:~$    ←一般ユーザ
root@axl:~#       ←管理者
```

この表示は「***ユーザ名@ホスト名：ディレクトリ名 プロンプト***」を意味します。一般ユーザではプロンプトが「**$**」に、管理者では「**#**」になっているので、プロンプトを見れば、今利用しているユーザが管理者か一般ユーザかを判断できます(「**~**」はホームディレクトリを示します)。

必要に迫られて管理者権限を利用したい場合は、sudoという実行ユーザを切り替えてコマンドを実行するツールを利用しましょう。Ubuntuなど、そもそもrootユーザを使わせないポリシーのディストリビューションもありますが、これらも一般ユーザで**sudo**を実行することを勧めています。

コマンド入力方法

CLI環境であればユーザ名とパスワードを入力してログインしたあと、プロンプトが表示されれば、あなたの戦場は用意されたようなものです。

GUI環境であればCLIと同じようにログイン後に「GNOME端末」のような端末エミュレータを起動すれば、そこがあなたの戦場です。GNOME 3やUnityなら、検索窓から「端末」と検索すればショートカットが出てきます(**図1**、**図2**)。端末エミュレータが見つからない場合は、Alt＋Ctrlと F1～F6 キーのどれか1つを押してみるとCLI環境になるので、そこを利用しましょう。Linuxは6つの仮想コンソールが標準で用意されており、画面とキーボードをほかの仮想コンソールと独立して利用できます。GUIに戻る場合はAltと F7 キーを押すと戻れます。

コマンドの実行は、実行したいコマンドと、そのコマンドに与えるオプション、もしくは引数からなります。たとえば、ディレクトリ「/var/log」以下にあるすべてのファイル／ディレクトリ一覧を表示する`ls -al /var/log/`を例として挙げると、「ls」がコマンド、「-al」がオプション、「/var/log/」が引数になります(**リスト1**)。

コマンドによっては、オプションの中に内部コマンドがある場合があります。次に示すのは

▼図1　GNOME 3で「端末」と検索

▼図2　Unityで「端末」と検索

▼リスト1　コマンド記述例

```
ls -al /var/log/
```

引数
オプション
コマンド

opensslの例です。

```
openssl x509 -in cert.pem -noout -text
```

コマンド　内部コマンド　オプション　引数　オプション　オプション

このコマンドラインはSSL(Secure Socket Layer)を実装した暗号ツールであるopensslの内部コマンド**x509**を利用し、読み込むファイルを指定するオプション-inに証明書ファイルの「cert.pem」を付け、さらにファイルに出力せずに標準出力に結果を出力する-noutオプション、テキストファイル形式で出力する-textオプションを付けて、SSL証明書ファイルの内容を標準出力に出します。

このような内部コマンドや複数のオプションとそれに対応する引数を渡せることもあります。

ここで例に出したlsとopensslコマンドは実行後に結果を表示するコマンドですが、UNIXコマンドのほとんどは、実行に成功した場合はたいてい寡黙であることを覚えておいてください。たとえば、空のファイルを作成したり、ファイルのタイムスタンプを現在時刻に変更したりする**touch**コマンドで、空ファイルtest.txtを作成すると何も表示されません。次のように、実行後すぐにプロンプトが返ってきます。

```
$ touch test.txt
$
```

何も反応がないと成功しているのかどうかわからないものですが、失敗した場合はエラーを出力するのもUNIXコマンドの特徴です。次はファイルを作成できなかった場合のエラーメッセージです。英語のメッセージですが、日本語環境が整っていれば「touch: `test.txt'にtouchできません:許可がありません」のように表示されます。

```
$ cd /var/log
$ touch test.txt
touch: cannot touch 'test.txt': Permission
denied
```

エラーメッセージは失敗した理由が書いてある重要情報です。ちゃんと読んで間違っているところを修正して目的を達成しましょう。そうでないと、連載「ひみつのLinux通信」(181ページ)のように先輩や顧客に絞られることになります。

オプション

オプションはそのコマンドの動作を変えるための指令です。lsコマンドはカレントディレクトリ(ユーザが今いるディレクトリ)にあるファイルとディレクトリ名を一覧表示するコマンドですが、-lオプションで名前以外にサイズとファイル属性を表示させる、-aでファイル名の先頭に「.」が付いた隠しファイル/ディレクトリも表示対象にする、-tでタイムスタンプ順に並べ替える、--colorでファイルの種類によって画面の文字に色を付けるなど、動作を切り替えられます。

一般的に、オプションにはショートオプションとロングオプションがあります。lsコマンドでは-aがショート、--allがロングオプションですが、どちらを利用しても動作は同じです。ショートオプションは-aと-lをあわせて-alと記述できますが、ロングオプションは個別で入力する必要があります。

コマンドによって用意されているオプションは異なります。詳細はマニュアルを参照するのが正攻法です。

引数

引数は、そのコマンドの操作対象などを指定します。お約束のlsコマンドでは引数を指定しないとカレントディレクトリ(ユーザが今いる

ディレクトリ）の内容を一覧表示しますが、「**ls /bin/ls**」のように引数として表示したいPATHを指定すればそのディレクトリやファイルの情報を表示することができます。

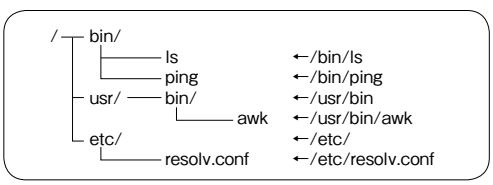
別のコマンドの例では、ICMPパケットを送信してそのレスポンスによって対象ホストが稼動状態にあるかの判断材料にする**ping**コマンドであれば、引数としてその対象ホストのIPアドレスかホスト名を指定します。「**ping www.gihyo.jp**」と実行する場合、「www.gihyo.jp」が引数です。

PATH

PATHには大きく2つの意味があり、1つは「ファイルシステム中の位置」で、ディレクトリやファイルがそのシステムではどこに配置されているかを示します。コマンドのlsであれば/bin/ls、pingであれば/bin/pingになります。ディレクトリ構成の一部とファイル／ディレクトリのPATHの対応を描いてみたものが**図3**です。この「**/**（ルートディレクトリ）」からファイル」ディレクトリまでの経路をPATHと言います。

もう1つは環境変数PATHで、シェルはこの変数に指定されたディレクトリに配置されたコマンドを優先的に探します。本来、lsコマンドを実行するには、/bin/lsとプロンプトに入力する必要があります。しかし、lsと入力するだけで実行できるのは、PATHに/binが設定されているので、/bin以下を探索してlsコマンドにたどり着けるからです。コマンド実行までのシェルのコマンド探索の旅を簡単にまとめると次のようになります。

▼図3　ディレクトリ構成とPATHの対応



（1）シェルはユーザからコマンド「ls」を実行することを命令される
（2）シェルはlsがどこにあるかわからないので、環境変数PATHの中を1つ1つ探す
（3）コマンドを見つけたらそれを実行するが、見つけられなかったら「command not found（コマンドが見つかりません）」というエラーメッセージを表示する

環境変数PATHの中身は**echo**コマンドで確認できます。次の例では「**:**」区切りで複数のPATHが設定されているのがわかります。

```
$ echo $PATH
/usr/local/bin:/usr/bin:/bin:/usr/local/games:/usr/games
```

この環境変数PATHに追加でPATHを設定する場合は、利用しているシェルの設定ファイルに追加したいPATHの一文を追記します。bashを利用している場合、.bash_profileに記述すれば、再ログイン後から有効になり、.bashrcに記述すれば新たにシェルを起動したあとからそのPATHが有効になります。PATHの追加は次のように記述します。下の例では、環境変数HOMEで参照できるユーザのホームディレクトリ以下のadt/sdk/platform-toolsというAndroid開発環境ディレクトリと、Google Chromeブラウザがインストールされている/opt/google/chromeディレクトリをPATHに追加しています。

```
PATH=$PATH:$HOME/adt/sdk/platform-tools:/opt/google/chrome/
```

実行権

UNIX環境ではファイルやディレクトリへの権限が設定されます。設定できる権限の種類は3つで、読み込み／書き込み／実行です。コマ

ンドの場合は、読み込み／実行がユーザに許されていないとユーザはコマンドを実行できません。

ファイルへの権限はファイル属性を表示するlsコマンドで参照できます。**図4**のように、対象ファイルを指定して-lオプションを付けて実行した際に表示される「-rwxr-xr-x」が権限です。

この表示は「読み込み／書き込み／実行」を「所有ユーザ／所有グループ／その他」でアクセスできるかを示しています。

```
-   rwx   rwx   rwx
❶   ❷     ❸     ❹
```

❶はファイルの種類を表す文字が入り、「-」はファイル、「d」はディレクトリを示します。その後は「rwx」と同じ文字が並びます。❷はファイルの所有者、❸は所属グループ、❹はその他ユーザです。❷～❹のそれぞれに、rは読み込み(Read)、wは書き込み(Write)、xは実行(eXecute)としてその文字があれば、ファイルにその権限が付いていることを示します。この記号は別の表現として「r--」であれば「100」、「r-x」であれば「101」のようにビットパターンに変換して、それを2進数から10進数に読み替えることができます。その記号と10進数値、権限の意味を整理したのが**表1**です。

▼図4　ls -lの実行例

```
$ ls -l /bin/ls
-rwxr-xr-x 1 root root 109992  7月 21  2013 /bin/ls
```

▼表1　記号と10進数表現と意味

記号	10進数値	権限の意味
r--	4	読み込み可
-w-	2	書き込み可
--x	1	実行可
rw-	6	読み書き可
r-x	5	読み実行可
rwx	7	読み書き実行可
---	0	読み書き実行不可

ファイルの権限は**chmod**コマンドで変更します。chmodに設定したい権限と変更したいファイル／ディレクトリを指定します。この権限の書き方が上記ビットパターンに関連するので、ここで覚えてしまいましょう。

よく使われる権限設定を**表2**にピックアップしてみました。記号部分、先頭3文字のユーザ、グループ、その他それぞれに設定される文字をビットに見立てて数値化し、3桁目、2桁目、1桁目にしました。

ここでモードという指定方法もあるので、ついでに見てみましょう。「***対象フィールド*[+-]*ビット***」のように指定します。対象フィールドはユーザは「u」、グループは「g」、その他は「o」、すべては「a」で指定し、ビットにrwxのいずれかを「+」で追加、「-」で削除します。サンプルのように複数フィールド、複数ビットをまとめて、「,」区切りで複数指定することができます。「,」区切りの場合は左から順番に設定されます。

次の2行は、ファイルtest.txtを「rw-r--r--」に設定するchmodコマンドの実行例です。

```
$ chmod 644 test.txt
$ chmod u+rw-x,og+r-xw test.txt
```

表2のモードは、あらかじめファイルに設定されているモードを無視して、新たに権限を設定しています。すでに「rw-r--r--」の権限設定がされている場合、簡単にどのユーザでも実行できるようにするには、**chmod +x test.txt**とすれば「rwxr-xr-x」になります。対象フィールドの「a」は省略でき、何も指定しない場合は「a」

▼表2　よく使われる権限設定

記号	数値	モード
rwx rwx rwx	777	a+rwx
rw- r-- r--	644	u+rw-x,og+r-xw
rw- r-- ---	640	u+rw-x,g+r-rw,o-rwx
rwx r-x r-x	755	a+rwx,ug-w
rw- --- ---	400	u+rw-x,ou-rwx
r-- r-- ---	440	ug+r-wx,o-rwx

が暗黙に指定されます。

システムにあらかじめ用意されているコマンドのほとんどは、全ユーザに対し、読み込み／実行権限が付いています。実行してみて、管理者権限が必要と警告される場合に初めてsudoコマンドを利用して管理者権限で実行しましょう。

コマンドの探し方

やりたいことを実行する場合、どのようにコマンドを探しましょうか。インストールされているコマンドのマニュアルはデータベースに保持されています。これを検索する**whatis**や**apropos**というコマンドを使ってみましょう。

whatisはコマンド名からマニュアルの要約を検索表示します。例として「ユーザ管理に使えそうなコマンドはインストールされているかな？」と「user」を含むコマンド名の要約一覧を表示してみましょう(**図5**)。

-wオプションはワイルドカード(文字列の長さが0以上の文字と数字を表す記号)「*」を使うためのオプションです。「***user***」が「**"**」で囲まれているのはシェルでの不用意なシェル展開を防ぐためです……が、ここではとりあえず囲っておいてください。すると、日本語マニュアルのあるコマンドについては日本語で、日本語がなければ英語で検索結果が表示されます。

aproposは引数に渡された文字列をコマンドの名前と要約から検索表示します。例として「ユーザ」という文字列を検索してみましょう(**図6**)。こちらもwhatisと同じように日本語のマニュアルがインストールされていれば、日本語で検索できます。

ほかの手段としては、書籍のコマンドリファレンスなどを手元に置いておくのもいいでしょう。書店に行くとコマンド本は多数用意されています。大きさや重さ、値段、参照しやすさ、サンプルの実用性がありそうかなどを見て1冊手元に置いておきましょう。

ちょっと慣れてくると、「何かを作るコマンドの場合は、mkから始まるコマンド名が多い」

▼図5　whatisコマンドの実行結果

```
$ whatis -w "*user*"
cuserid (3)          - ユーザ名を取得する
endusershell (3)     - 許可されたユーザシェルを得る
ftpusers (5)         - FTP デーモン経由でのログインを許さないユーザのリスト
fuser (1)            - ファイルやソケットを使用しているプロセスを特定する
getusershell (3)     - 許可されたユーザシェルを得る
```

▼図6　aproposコマンドの実行結果

```
$ apropos ユーザ
access (2)           - ファイルに対する実ユーザでのアクセス権をチェックする
chage (1)            - ユーザパスワードの有効期限情報を変更する。
chfn (1)             - ユーザの氏名や情報を変更する。
crontab (1)          - 各ユーザのためのcrontabファイルを管理する (V3)
```

▼図7　シェルの補完機能を利用してコマンドを調べる

```
# ch      ←ここまで入力してTABキーを2回押しましょう。以下が表示されます。
chacl           chardetect3     checkbashisms   chkdupexe       chrt
chage           chat            chem            chmod           chsh
chardet         chattr          chfn            chown           chvt
chardet3        chcon           chgpasswd       chpasswd
chardetect      chdist          chgrp           chroot
```

「何かを変更するコマンドの場合は、chから始まるコマンドが多い」などを予測して「ch」まで入力してTabキーを押してシェルの補完機能からコマンドリストを出して調べるという手段もあります(**図7**)。この中からwhatisコマンドで検索すると目的のコマンドが見つかるかもしれません。

最後は、Googleなどの検索エンジンに任せるという手段です。マニュアルに載ってない裏技をブログに書いている人もいるので、世界の広さを実感できて本当にお勧めです。

コマンドの使い方を調べる

システムにコマンドがインストールされているならば、マニュアルファイルも一緒にインストールされているはずです。ここは**man**コマンドを利用してコマンドの使い方を見てみましょう。manコマンドに調べたいコマンドの名前を引数として付けて、**man ls**のように実行してみましょう。

manコマンドを実行すると画面にコマンドのマニュアルページが表示されます。終了するときにはqキーを押します。ページ内はスペースキー、Enterキー、jキー、↓キーなどで下方向に、kキー、↑キーで上方向に移動します。

manファイルには**表3**の内容が記載されています。

この中の「DESCRIPTION(説明)」「EXAMPLE(サンプル)」と「NOTES(備考)」「SEE ALSO(関連項目)」あたりを見て概要をつかみ、「OPTIONS(オプション)」で細かく何ができるかを把握するのが真のコマンダーです。

日本語マニュアルページは、有志が英語マニュアルを翻訳してパッケージングしているものがほとんどです。日本語でマニュアルを参照できることに感謝すべきですが、タイミングによってはコマンドの最新バージョンに追随しきれていない場合があります。最新の情報を参照したい場合は、manコマンドの前に「LANG=C」を付けて**LANG=C man ls**のように実行することでオリジナルの英語マニュアルを参照することができます。

パイプと標準入力／標準出力／標準エラー出力

コマンドの実行結果をほかで利用するときは、出力をコピーしてエディタを起動してペーストすればいいの？　そんなことはありません。UNIXコマンドの出力を別コマンドの入力にしたり、ファイルに出力したりできます。

▼表3　manファイルの内容

見出し(日本語)	見出し(英語)	記載内容
名前	NAME	コマンドの名前と簡単な説明
書式	SYNOPSIS	コマンドとオプションの記述方法
説明	DESCRIPTION	コマンドの詳細説明
オプション	OPTION	コマンドのオプション説明
終了ステータス	EXIT STATUS	コマンド終了時に返す終了ステータス
戻り値	RETURN VALUE	コマンド終了時に返す値
環境変数	ENVIRONMENT	コマンド実行時に利用できる環境変数
ファイル	FILES	コマンド関連ファイル
備考	NOTES	注意書き
サンプル	EXAMPLE	コマンド実行例
関連項目	SEE ALSO	関連したコマンドや参考資料
バグ	BUGS	コマンドですでに知られているバグ
著者	AUTHOR	コマンドの開発者

UNIXコマンドは実行されるとプロセスとなり、3つの入出力チャンネルを持ちます。標準入力というプロセスへの入力となるチャンネル、標準出力というプロセスの結果を出力するチャンネル、標準エラー出力というプロセスのエラーメッセージを出力するチャンネルです。

とくに指定しない限り、標準入力はキーボード、標準出力／標準エラー出力はディスプレイが利用されます。実際にコマンドを使ってみましょう。まずは「|(パイプ)」を使ってコマンドの出力を別のコマンドに渡してみます(**図8**)。

①はechoコマンドに引数を渡して標準出力、つまりディプレイに文字列を出力したところです。②はechoの出力と次のコマンドへの入力を「|(パイプ)」で連結します。echoの出力がrevコマンドの標準入力として利用され、標準出力はディスプレイに出力します。revは入力文字列を逆さまに並べ替えた文字列を出力します。コマンドはいくつでも連結でき、③では3つのコマンドの出力と入力を「|」2個で連結しています。この場合はrevコマンドの出力を、文字を削除／変更するtrコマンドにつなげ、スペースを削除しています(**[:blank:]**を指定することでスペースを削除します)。パイプの入出力の概念図を**図9**に示します。上が特別に入出力を指定しない場合、下が、「|」を使って入力と出力を指定する場合です。

出力を別のコマンドの標準入力にする場合は「|」ですが、ファイルを入出力に使う場合は「**>**」と「**>>**」を使うとシェルの機能であるリダイレクトを利用できます(**図10**)。

④はpsコマンドにaxオプションを付けて実行した結果をファイル/tmp/ps.txtに出力しています。実行後、ファイルの中身を見るページャのlessコマンドで/tmp/ps.txtを開くと、psコマンドを実行したのと同じ内容がそこにあるはずです。「**>**」はその後に指定されたファイルへ出力する指示です。すでにそのファイルが存在する場合は空にしてから書き込むので、重要なファイルに対して実行する場合は気をつけましょう。

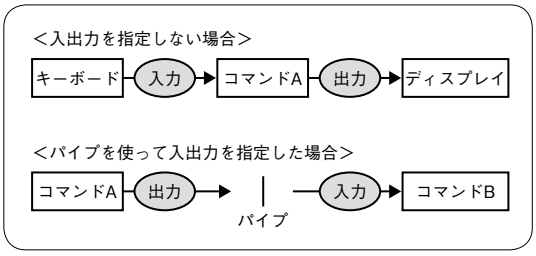
⑤はシステムの稼働時間／負荷状況などを表示するuptimeコマンドを実行し、それを/tmp/uptime.txtに「追記」します。「**>>**」は直後に指定したファイルがすでに存在する場合は、ファイルの末尾に出力内容を追記出力します。ファイルが存在しない場合は、新規作成して出力します。

⑥は「**>**」だけではなく、「**<**」を利用しています。これはdos.txtを入力としてtrコマンドに渡し

▼図8　パイプの使用例

```
$ /bin/echo "Hello Linux"  ←①
Hello Linux
$ /bin/echo "Hello Linux" | rev  ←②
xuniL olleH
$ /bin/echo "Hello Linux" | rev | tr -d [:blank:]  ←③
xuniLolleH
```

▼図9　パイプの入出力の概念図



▼図10　リダイレクトの使用例

```
$ ps ax > /tmp/ps.txt  ←④
$ uptime >> /tmp/uptime.txt  ←⑤
$ tr -d ¥¥r < dos.txt > unix.txt  ←⑥
$ find /tmp/ > /dev/null  ←⑦
$ find /tmp/ 2> /dev/null  ←⑧
```

ます。trの出力は「>」の先にあるunix.txtです。文字の変換／削除を行うtrコマンドのオプション「**-d ¥¥r**」は「復帰コード」(CR)を削除することを示しています。Windowsで保存したテキストファイルの改行コード「¥r¥n」(CR + LF)からUNIXのテキストファイル改行コード「¥n」(LF)を残すために「¥r」(CR)を削除しています。これでLinuxでも、Windowsで作成されたテキストファイルを見やすく表示できるはずです(文字コードの違いはまた別ですが)。

⑦と⑧はfindコマンドで/tmp/以下のファイル／ディレクトリを出力しています。違うのはリダイレクトの前に数字が付いているかどうかです。⑦は「**>**」ですが、じつは「**1>**」の「1」が省略されているのです。

標準入力、標準出力、標準エラー出力はそれぞれ0、1、2という番号が振られます。先ほどのコマンドの⑦⑧の「1」と「2」はそれぞれ標準出力と標準エラー出力を示す番号です。

⑦の場合、標準出力「1」を/dev/nullにリダイレクトしています。/dev/nullは入力されたものは何でも吸い込むブラックホールで、この場合は、標準出力を/dev/nullに渡すことで標準エラー出力の内容のみを画面に出力します。この例では、一般ユーザでfindコマンドを実行しているので、権限がないディレクトリを表示しようとした際のエラーが出力されるでしょう。

⑧では、標準エラー出力「2」を/dev/nullにリダイレクトしています。同じように考えると、標準出力「1」は画面に出力されるので、エラーメッセージを抜いたファイルリストが出力されます。

標準出力と標準エラー出力はデフォルトでは画面に出力しますが、⑦⑧の方法ならそれぞれをファイルに出力できます。両方を同じファイルに出力するには「**2>&1**」という「標準エラー出力を標準出力の複製にする」という指令を利用すると一緒に標準出力経由で書き出すことができます(⑨)。

```
find /tmp > /tmp/find.txt 2>&1  ←⑨
```

1>/tmp/find.txtと**2>&1**のリダイレクトがある場合、左から評価されます。まず標準出力がディスプレイから/tmp/find.txtに変更され、次に標準エラー出力が標準出力に設定されているファイル出力になるので/tmp/find.txtに両方出力されます(**図11-上**)。

これを「**find /tmp 2>&1 > /tmp/find.txt**」のように順番を変えると、まず**2>&1**が評価されて標準エラー出力は標準出力であるディスプレイ出力をコピーして利用することになります。そのあとで標準出力はファイルに向けられますが、標準エラー出力はディスプレイ出力のままですので、標準出力のみがファイルに出力されてしまいます(**図11-下**)。

リダイレクトの順番は、失敗が多いほど慣れて体が覚えます。何度も間違えて体系的に慣れ、理解しましょう。SD

▼図11　リダイレクトの順番が違うと結果が異なる

command	2>&1	>file	
①ディスプレイ ②ディスプレイ	①ディスプレイ ②ディスプレイ (①のコピー)	①file ②ディスプレイ	(´・ω・`)

command	>file	2>&1	
①ディスプレイ ②ディスプレイ	①file ②ディスプレイ	①file ②file (①のコピー)	(・∀・)

※①は標準出力、②は標準エラー出力を表す

第2章 面倒なことをコマンドでまとめてやろう ——ファイル一括処理にマウスは不要!

UNIXコマンドは1つでも十分強力なこともありますが、基本的には1つ1つのコマンド機能はシンプルな設計です。複数のコマンドを連結して目的の結果を得ることもコマンド環境の魅力のひとつです。ここでは用途からコマンドの便利な使い方を見てみましょう。

1 rmコマンドで複数ファイルを指定して削除する

```
$ rm *~
```

あるディレクトリ以下にある特定のファイルを一括で削除する方法です。GUI環境であればエクプローラのようなアプリケーションで目的のディレクトリを表示して、マウスを左クリックしながら対象ファイルのアイコンを有効範囲に入るようにドラッグして囲み、Delete キーを押せば消えると思います。UNIXコマンドでは**rm**を使います。

rmコマンドは引数に渡されたファイルを削除します。ファイル名にはスペース区切りで複数のファイル名を指定できます。よく「*」のような記号を使って複数ファイルをまとめて削除している場面を見ますが、これはシェルの特殊パターン文字を使うことで任意のファイルをまとめて削除対象にしています。冒頭のコマンド例では、Emacsというエディタを利用した際にバックアップファイルとして作成される「ファイル名~」をすべて削除します。

複数ファイルを候補にする特殊パターン文字は、bashのマニュアルであれば「パターンマッチング」で参照することができますが、ここでほかのパータンマッチングも少しだけ見てみましょう。

```
例❶ $ rm ./*
例❷ $ rm test[0-9].{sh,txt}
```

上記例①での「**./***」はカレントディレクトリ(./)のすべてのファイルを「*」で指し示しています。これを実行すると、今いるディレクトリに存在するファイルは次の瞬間消えます(気をつけて!)。

例②の「**test[0-9].{sh,txt}**」は“test0からtest9までの名前で、かつ「.txt」か「.sh」の拡張子がついたファイル”として展開されます。それらがすべてrmの対象になるということです。

rmコマンドはオプションを渡さないとファイルのみを削除し、ディレクトリは削除できません。ディレクトリも含めて削除する場合は-rオプションを指定するとディレクトリ以下も含めて削除してくれます。-rはrecursiveのオプションで再帰的にディレクトリの中身を削除します。次はディレクトリを削除する例です。

```
$ rm -r testdir
```

たとえば、Subversionの管理ディレクトリ「.svn」を削除したい場合、次のように実行します。

```
$ rm -r .svn
```

これでは直下のディレクトリにある.svnのみが削除対象になります。さらに階下のディレクトリに.svnがあり、こちらも削除したい場合は

どうしましょう。大丈夫、シェルのパターン展開を利用すれば`rm -r */.svn`、さらに下なら`rm -r */*/.svn`で消せます。でも……もっと下があると面倒くさいですね。

2 findコマンドの検索結果を使って一括削除

```
$ find . -name .svn -exec rm -rf {} \;
```

findコマンドは指定したPATH以下のファイルとディレクトリに対し、検索条件を用意してそれにマッチする対象をリスト表示したり、リストを別コマンドへの引数として渡すことができます。

冒頭のコマンド例では、ユーザが今いるディレクトリ(カレントディレクトリ、ワーキングディレクトリとも呼びます)を示す「.」をPATHとして指定し、検索条件に-nameオプションで名前を「.svn」と指定しています。-execオプションからはfindコマンドでは「アクション」と呼ぶ動作を指定する部分で、例ではrmコマンドに「{}」を指定することで、検索条件でマッチしたファイルが1つずつ入れられてrmが実行されます。

-exec以降を指定しないで、末尾に「~」がついたファイルを階下まるっと表示する例を見てみましょう。

```
$ find . -name "*~"
./proj1/config/database.yml~
./proj1/apps/app1.rb~
```

該当するファイル一覧が表示されました。-execのようなアクションが指定されていない場合は-printのアクションが暗黙に指定されて検索結果が出力されます。

これにアクションとしてlsコマンドを実行してみましょう。-execオプションの後は検索結果を渡すコマンドを渡し、「{}」をコマンドへの引数として「;」の直前までを渡します。「;」はシェルにコマンド区切り文字として展開されてしまうので「\;」のように「バックスラッシュ」でエスケープします。

lsコマンドで“素振り”している状態ですが、これでrmコマンドに置き換えれば目的のファイルを一括削除できますね。

```
$ find . -name “*~” -exec ls {} \;
./proj1/config/database.yml~
./proj1/apps/app1.rb~
```

この記述では1つ1つ検索結果ファイルを削除することになります。「{}」の後に「+」をつけると、処理する前に検索結果をコマンドに対する引数の末尾に追加するようになります。

```
$ find . -name "*~" -exec ls {} +
./proj1/apps/app1.rb~  ./proj1/config/database.yml~
```

3 findとxargsとの合わせ技で削除する

```
$ find . -name “*~” | xargs rm
```

findコマンドの-execオプションでrmコマンドを指定してファイルを削除するのもいいのですが、**xargs**コマンドに引数として渡して実行するという手段もあります。

xargsコマンドは標準入力に渡された文字列を、余分な空白や改行を取り除いて“引数”として使いやすく整形します。xargsに実行するコマンドを指定しないと、echoコマンドを実行するので1行表示されます。実行してみましょう。

```
$ ls
test1   test11  test13  test15  test17
test19  test20  test4   test6   test8
test10  test12  test14  test16  test18
test2   test3   test5   test7   test9
$ ls | xargs
test1 test10 test11 test12 test13 test14
test15 test16 test17 test18 test19 test2
test20 test3 test4 test5 test6 test7
test8 test9    ←実際には1行
```

この合わせ技の利点の1つはシステム負荷を軽くできることです。findコマンドの-execオプションでrmコマンドを実行するとファイル1つ1つにrmコマンドを実行するため、ファイル数が大量になるとシステム負荷が高くなります。一方、冒頭のようにxargsコマンドを利用するとfindコマンドの結果、カレントディレクトリにあるファイル名末尾に「~」が付いているファイル一覧を1行にしてrmコマンドに渡すため、実際にrmコマンドを実行するのは基本1回になります。

別のケースでfindコマンドを使うほどでもない場合、上記の例で言えば「**rm *~**」でもいいのですが、「*~」でシェル展開されるファイルが多すぎる場合に期待どおりファイルをすべて削除できないことがあるので注意が必要です。

Linuxはコマンドライン引数の最大文字数が設定されているために、引数を無限に渡せるわけではありません。**getconf**コマンドでARG_MAXを指定する(**getconf ARG_MAX**)とそれが得られますが、この最大文字数を超える引数をrmコマンドに渡すと「Argment list too long」のエラーメッセージが出力されて実行されません。xargsはこの部分を考慮し、ARG_MAXを超える引数が渡された場合はそれを複数に分割してコマンド実行します。

findコマンドの-execやrmコマンド単体で利用するか、xargsコマンドと連結するかは削除する対象ファイル数を予想して判断してもいいでしょう。

4 「.txt」を持ったファイルをすべて「.csv」に変更する

```
$ for f in *.txt; do
mv $f ${f/.txt/.csv}
done
```

ディレクトリ下ファイルの拡張子「.txt」を持ったファイルをすべて「.csv」に変更する方法を考えてみましょう。

```
例❶ $ mv test1.txt test1.csv
例❷ $ mv test1.{txt,csv}
例❸ $ rename 's/\.txt$/\.csv/' *.txt
例❹ $ rename txt csv *.txt
例❺ $ for f in *.txt; do
     mv $f $(basename $f .txt).csv
     done
```

ファイル名の変更といえば、普通は**mv**コマンドを例①のように変更前ファイル名と変更後ファイル名を指定して行うと思います。実は1つのファイル名を変更するのならば例②のようなやり方もあります。

例②はシェルのブレース展開[注1]を利用した指定方法です。「**test1.{txt,csv}**」とすると「**test1.txt test1.csv**」と展開されます。それをmvコマンドの引数として渡すので例①と同じですね。

例③と④の方法は、**rename**コマンドを利用して複数のファイルを対象に名前の変更をします。Debian系ディストリビューションでは、renameはプログラミング言語Perlで書かれたコマンドで、例③のように第2引数に渡した変更前ファイル名に、第1引数で渡すPerlの正規表現(コラム参照)での変更ルールを適用して変更後ファイル名になります。RHEL(Red

注1) ブレース({})で囲まれた複数の文字列をブレース前後の文字列(省略可)へ補完して複数の文字列にする処理。

Hat Enterprise Linux)系ディストリビューションではutil-linuxパッケージに含まれるバイナリを利用するので、Perl版とは使い勝手が異なります。例④のように、引数として「変換前文字列　変換後文字列　対象ファイル」を指定します。使っているマシンにrenameが入っているなら、これを使うとファイル名の一括変換は便利です。

そして冒頭のコマンドと例⑤が、カレントディレクトリにある「.txt」のファイルをまとめて.csvに変換する汎用的な実行方法です。シェルの**for**文を利用し、「***.txt**」の結果を1つずつ変数fに設定し、mvコマンドを実行します。forは候補がなくなるまで同じコマンドを繰り返します。

冒頭のコマンドでは、mvコマンドの1つ目の引数「**$f**」はそのまま変更前ファイル名が入ります。2つ目の引数の「**${f/.txt/.csv}**」はシェルのパラメータ展開を利用して変更後ファイル名を表現しています。

シェル変数をfとすると、値を参照する際は$fと記述して利用します。そのまま変数の値を使う場合はこれでいいのですが、変数名もあわせて「{}」で囲んで、その中で呪文を追加するともう少し凝った利用ができます。この場合「${変数名/パターン/文字列}」のルールで、変数名の値が持つパターンを文字列で置き換えます。変数fに「test.txt」が入っていると考えた場合、最初に「.txt」のパターンを用意し、文字列として「.csv」を指定しているので「test.csv」と展開され、最終的に「**mv test.txt test.csv**」になります。

例⑤では、変更後のファイル名を生成するのに**basename**コマンドを利用しています。basenameは引数に渡された文字列からディレクトリ名を取り除くコマンドです。たとえば「**basename /var/log/syslog.2.gz**」を実行すると「syslog.2.gz」が出力されます。また、末尾に文字列を追加指定するとそれも除去します。「**basename /var/log/syslog.2.gz .gz**」を実行すると「syslog.2」が出力されるわけです。これを踏まえて例⑤を見ると、「**$(basename $f .txt)**」は変数fがtest.txtだとすると、これからディレクトリ名と「.txt」を除去するので「test」が実行結果として出てきます。これに「.cvs」をつけているので「test.csv」となります。

なお、forなどのシェルの制御構造の詳細は今回は紙幅の都合上残念ながら扱えませんので、詳しくはマニュアルや書籍を参考にしてください。

COLUMN

正規表現とは

正規表現とは文字列をパターンで表現する記述方法で、Linuxコマンドではgrepやsedなどのコマンドで文字列の検索や置換に利用します。例としてgrepコマンドでテキストファイル「dp.txt」の中にある「daft punk」の文字を検索するケースを想定します。タイプミスして「duft pank」と書かれているかもしれませんね。これを考慮するとgrepは「**grep 'd[au]ft p[au]nk' dp.txt**」のように実行します。

「daft punk」を検索結果として絞り込むためにパターンに当てはまる部分についてメタ文字を使って表現します。上記の例では「**[**」と「**]**」で出てくる可能性のある文字「**a**」と「**u**」を括ることで、想定したタイプミスも検索結果として出せます。このメタ文字の代表的なものを表Aに並べてみます。

簡単に説明しましたが正規表現は奥が深いので、興味を持ったらぜひ専門書を参照することをオススメします。

▼表A　代表的なメタ文字

メタ文字	意味
*	0回以上の直前文字繰り返しに一致
+	1回以上の直前文字繰り返しに一致
.	任意の1文字に一致
?	0か1回の直前文字繰り返しに一致
[ABC]	[ ]内に記述されたいずれか1文字に一致
^	行頭
$	行末
[0-9], \d	数字

5 画像ファイルの一括サイズ変更

```
$ for f in *.jpg; do
convert -resize 33% $f ${f/.jpg/-33per.jpg}
done
```

この冒頭コマンドは、シェルのfor文を利用し、ImageMagickパッケージに含まれる画像処理コマンドのconvertを使って元画像から33%にリサイズし、ファイル拡張子前に「-33per」の文字を追加したファイル名に出力します。

convertはGIMPのような画像処理はできませんが、画像ファイルの変換には頼もしいツールです。出力ファイル名の拡張子にあわせてフォーマットを変換する機能は重宝します。「.pdf」を指定すれば画像をPDFに埋め込んで出力してくれます。オプションでは-rotateで回転、-cropで始点と幅、高さを引数で渡せば範囲を切り抜くこともできます。

次は文字を画像に書き出してみましょう。convertで使えるフォントは「convert -list font」で得られます。

下の例①では「label:」に指定した文字の画像が作られます。-backgroundは背景色、-fillは文字色、-fontはフォントを指定するオプションです。例②の「label:」直後にある@はファイル参照の目印です。ここでは標準入力から文字を得るので「@-」を指定しています。ここではパイプ前半の実行結果が標準入力になるので、それが画像になります。

```
例❶ $ convert -background white -fill red -size 800x600 -font "IPA-UIゴシック-Regular" ↵
     label:"ソフトウェアデザイン" sd.png
例❷ $ /sbin/ifconfig eth0 | convert -font "IPA-UIゴシック-Regular" label:@- eth0.png
```

COLUMN

シェル変数の文字列展開

シェル変数に入れた文字列を一部加工して再利用する文字列展開は便利ですね。見出し4の「**${f/.txt/.csv}**」や5の「**${f/.jpg/-33per.jpg}**」のような記述法ですが、ほかにも記述方法があるのでちょっと触れておきます。

シェルの変数は「$」をつけて参照します。変数名を「f」とした場合「$f」でも「${f}」でも参照できます。「{}」をつけるのは、ほかに「file」のような変数名があった場合に、変数fが変数file巻き込まれずに展開できるようにします。また、「{}」で変数を囲み、そこに条件をつけることで格納されている値を操作できます。代表的な操作を表Bにまとめておきましょう。

▼表B　文字列展開の代表操作例

書き方	処理内容	例
${変数:先頭文字数:出力長}	先頭文字列数の文字から出力長の文字数だけ出力	val="abcdefghijklmn"; echo ${val:3:5} ↵ #=> defgh
${変数#文字列}	先頭から「文字列」にマッチした部分を取り除いて出力	val="abcdef"; echo ${val#abc} #=> def
${変数%文字列}	末尾から「文字列」にマッチした部分を取り除いて出力	val="abcdef"; echo ${val%def} #=> abc
${変数/パターン/置換文字}	変数内「パターン」にマッチする文字を「置換文字」に置き換える	val="abcdef"; echo ${val/def/abc} ↵ #=> abcabc

6 各ディレクトリの利用容量を一覧表示する

```
$ sudo find /home -mindepth 1 -maxdepth 1 -type d -exec du -hs {} \;
```

「ああ、サーバの/homeが溢れてる……誰が溜め込んでいるんだろう」というときに、ディレクトリ利用量を表示するduコマンドでユーザ一人ひとりのホームディレクトリを調べるのは手間がかかります。

冒頭のコマンドでは、/home以下の各ディレクトリを-execアクションでduコマンドを呼び、人間が読みやすい形式で(-h)、合計容量のみ表示(-s)するオプションをつけて渡します。findコマンドの-type dはディレクトリを指定するオプションで、-type fならファイルのみが検索対象になります。-mindepthと-maxdepthはディレクトリ階層最低深度と最高深度を示し、それぞれ「1つ下以上」「1つ下まで」を示すことで、/home直下の各ディレクトリの合計容量、各ユーザのホームディレクトリ以下に作られているディレクトリの容量まで表示しないようにします。

この実行結果は次のように表示されます。

```
$ sudo find /home -mindepth 1 -maxdepth 1 -type d -exec du -hs {} \;
248G    /home/ryosuke
4.8M    /home/rino
7.6M    /home/lfs
```

長いコマンドラインを例にしましたが、同じ結果は「**du -hs /home/***」でも得られます。

7 文字列を一括変換

```
$ sed -e "s/\(charset=\)EUC-JP/\1UTF-8/i" -i *.html
```

カレントディレクトリに「.html」の拡張子がついたHTMLファイルが複数あり、今までEUC-JPで編集していたのをUTF-8にすべて文字コード変更したと仮定しましょう。このとき、HTMLファイルなので文字コードの変換を行ったら、HTMLヘッダにある次のような行のcharsetもあわせて変更したいところです。

```
<meta http-equiv="Content-Type"
content="text/html; charset=EUC-JP" />
```

これを実現するのが冒頭のコマンドです。**sed**コマンドはストリームエディタというツールです。入力として入ってくるデータを、与えたフィルタで処理して出力します。冒頭のコマンドは「-e」オプションでスクリプトを指定します。「**s/\(charset=\)EUC-JP/\1UTF-8/i**」がスクリプトの部分で、「s/*正規表現*/*置き換え文字列*/」という形式で記述し、正規表現にマッチする文字列を「置き換え文字列」で置き換えます。正規表現部分の「**\(charset=\)**」は部分正規表現として囲んだ部分を、置き換え文字列内で特殊エスケープとして「**\1**」だけで扱えます(部分正規表現が複数あれば9つまで「\番号」のように使えます)。

これで「charset=EUC-JP」の「EUC-JP」部分を「UTF-8」に置き換えられます。また、最後の「i」は大文字小文字を区別しないオプションなので「EUC-JP」も「euc-jp」も対象になります。本文中に「EUC-JP」単体での記述がある場合にも変換対象になってしまうので、なるべく候補を絞れるように「charset=」も検知対象にしています。

最後に-iオプションを利用します。これは入力ファイルと出力ファイルを同じファイル名で利用するオプションです。

8 テキストファイルの指定した行数を削除・編集する

```
$ for f in *.html; do
ed $f <<EOF
3,5d
w
q
EOF
done
```

ファイルの特定の行を編集する場合、対象文字列のパターンがある程度絞れるならsedを利用するのが楽かもしれませんが、目的を達成するのに行数指定でも事足りるなら冒頭コマンドのやり方を覚えておくと役に立つでしょう。

いきなりfor文から始まっていますが、カレントディレクトリにある「.html」の拡張子があるファイルに対し、すべて3〜5行目を削除して保存するコマンドです。実際のファイル編集はedという行指向エディタを利用しています。

行指向エディタとは1行ずつ編集する非常にコンパクトなエディタで、ターミナルで実行した後はexコマンド使ってファイルを編集します。viのコマンドモードを使ったことがある方は、そのコマンドモードを利用すると思えばわかりやすいかもしれません。ちなみにviは端末画面すべてを使えるエディタです。行指向に比べてヴィジュアル効果が高いですよね。だからVIsual editorの略で「vi」だそうです。

この例では、edの引数にfor文で変数fに入っているファイル名を渡します。「<<EOF」はヒヤドキュメントと呼ばれる構文で、指定した文字列が現れるまでの文字列をコマンドの入力にするという機能があります。ここでは「3,5d〜q」までの3行がedの入力になります。「3,5d」は3行目から5行目までを削除(deleteのd)する、「w」は書き込み(writeのw)、そして「q」は終了(quitのq)という意味のコマンドです。

対象ファイルの末尾に文字列を追加する場合、edへの入力は次のようにします。

```
ed $f <<EOF
  3,5d
  $
  a
  this is test
  .
  w
  q
EOF
```

aで追記(appendのa)を指定すると挿入モードに移行します。その後に追加するテキスト「this is test」を入力し、コマンドモードに戻るために「.」を入力、「w」で書き込んで「q」で終了します。するとそのファイルの末尾に「this is test」が書き込まれています。

大量の同じフォーマットのテキストファイルが操作対象で、同じ個所のみを編集する、という目的の場合だけに有効なように見えますが、いざその場面が出てきたときに「このやり方ならいける！」と思い出してほしいものです。

9 コマンドを連結してログファイルの解析をする

```
$ sudo zgrep "02/Mar" /var/log/apache2/access.log.3.gz | ➥
awk '{ print $1 }' | sort | uniq -c | sort -r | head -n 5
```

Apache httpdサーバのアクセスログから特定の日付(例として3月2日)のアクセス元を抽出し

て、その結果からどれぐらいの回数アクセスがあるかを調べてみましょう。httpdサーバのログはデフォルト設定では次のように出力されます。

```
49.YYY.157.XXX - - [02/Mar/2014:02:15:57 +0900] "GET / HTTP/1.1" 200 4373 "-" "Mozilla/5.0 (iPhone; CPU iPhone OS 7_0_6 like Mac OS X) AppleWebKit/537.51.1 (KHTML, like Gecko) Version/7.0 Mobile/11B651 Safari/9537.53"
```

この中から、特定の日付ということで「02/Mar」を対象にログから検索すれば良さそうなのがわかります。3月2日のログの所在を調べると/var/log/apache2/access.log.3.gzというファイルにすでにローテート[注2]されていました。gzip圧縮されているためgrepでは検索できないので、**zgrep**という圧縮ファイルを検索できるコマンド利用しています。

zgrepの出力は「|(パイプ)」によって**awk**コマンドの入力に利用されます。awkはプログラミング言語ですが、sedと似ており、流れてくる文字列を処理するのに便利なのでこのような使い方が多用されます。「**'{ print $1 }'**」はawkスクリプトです。この意味は、入力された各行の「空白区切り[注3]」で1つ目の項目を出力するということになります。

第一項目はIPアドレスなので、それがそのまま「|」を経由して次の「**sort**」、「**uniq -c**」に続きます。**sort**コマンドは入力内容を並び替えて出力します。**uniq**コマンドは重複する行をまとます。-cオプションは出現回数をカウントします。カウントした出現回数の降順でソートし、「**head -n 5**」で上位5つのIPアドレスを抽出します。

ここまで説明した冒頭コマンドの実行結果が次になります(アクセス回数上位5位までのIPアドレスを表示)。

```
$ sudo zgrep "02/Mar" /var/log/apache2/access.log.3.gz | awk '{ print $1 }' | sort | uniq -c | sort -r | head -n 5
366 27.YYY.203.XXX
344 66.YYY.79.XX
315 66.YYY.79.XX
289 66.YYY.79.XX
279 27.YYY.195.XXX
```

同じ要領で、SSHログファイルから不正アクセスに利用されるユーザ名を調べてみましょう。SSHのログイン失敗ログはDebian GNU/Linux系では/var/log/auth.log、RHEL系では/var/log/secureに記録されます。フォーマットはデフォルトでは次のようになります。

```
Mar 18 03:05:55 oakley sshd[29830]: Invalid user recovery from 41.228.X.XX
```

メッセージ「Invalid user」をログから抽出し、先頭から空白区切りで8番目の文字列のログイン試行ユーザ名をカウントします。筆者のサーバで調べてみたところ、次のものがSSHログイン失敗ユーザ名ベスト10です。

```
$ sudo grep "sshd.*Invalid user" /var/log/auth.log  | awk '{ print $8 }' | sort | uniq -c | sort -r | head -n 10
 34 admin
 20 postgres
 16 guest
 15 webmaster
 15 user
 15 oracle
 15 ftpuser
 12 nagios
 11 www
 11 webadmin
```

本当のところ、このようなログイン試行はアタックなので、適切にIPアドレスを取り出してiptablesなどでブロックするのが望ましいところです。SD

注2) ローテートは、ログでストレージを埋め尽くさないように定期的に退避・削除・新しくログファイルを作成するしくみです。退避する際にgzip圧縮されることがあります。

注3) 「-F」オプションで文字列を指定すれば、デフォルトでは空白を区切りと見なすところを指定した文字列で区切り処理できます。たとえば「,」を指定すれば、CSVファイルをいじれるようになります。

第3章 サーバ管理者になったら頼りになるコマンド
——作業の効率化と自動化を見据えて

サーバ管理者になると、一般ユーザとはまた違った観点でコマンドを使う必要があります。複数の情報を一度に把握する、複数の作業をまとめて行う、といったことができる必要があります。また、作業効率化や自動化のために、GUIでできることでもコマンドラインで実施できると便利な場合があります。

1 ユーザのログイン状態を表示する

```
$ w
```

稼働中のサーバでは、複数ユーザがログインできるため、ほかのユーザがシステムの設定作業などを行っていることもあります。システムにログイン中のユーザが、何をしているかを把握するのも思いやり、配慮です。同じ設定ファイルを編集しようとしてエラーメッセージが出ると「何か悪いことしたかな……」と心配しちゃいますものね。そんなときは、冒頭の`w`コマンドでログイン中のユーザを見てみましょう(**図1**)。

図1ではユーザryosukeが3人、ユーザrinoが1人ログインしているのがわかります。横1列にUSER(ユーザ名)、TTY(端末名)、FROM(接続元)、LOGIN(ログイン時間)、IDLE(何もしていない時間)、JCPU(経過時間)、PCPU(プロセス経過時間)、WHAT(何をしているか)を示しています。

表示内容について、1行目はユーザryosukeが仮想端末7番目(TTYのtty7)からGDM(Gnome Display Manager)を11時36分から使っていること、次はユーザryosukeが仮想端末2番目(TTYのtty2)からログインしてシェルを利用中、14時6分ログインして17秒触っていないことを示しています。3行目のユーザryosukeが利用しているのはGNOME端末で、その端末が疑似端末のpts/0を利用しています。

この疑似端末は直接接続されていない端末への入出力を擬似的に扱います。仮想端末はキーボードとディスプレイが直接つながっていますが、端末エミュレータやSSHなど直接つながっていない場合に擬似端末を使います。

最後の行はユーザrinoがFROMに表示されている接続元192.168.1.5から11時31分にログインしていること、そして/etc/resolv.confをviで開いている様子がわかりますね。

▼図1　ログイン中のユーザを表示する

```
$ w
 14:06:49 up  2:32,  3 users,  load average: 0.27, 0.40, 0.27
USER     TTY      FROM             LOGIN@   IDLE   JCPU   PCPU  WHAT
ryosuke  tty7     :0               11:36    2:32m  1:03   0.06s gdm-session-worker [pam/gdm3]
ryosuke  tty2                      14:06   17.00s  0.08s  0.06s -bash
ryosuke  pts/0    :0               11:36    1.00s  0.03s  4.88s /usr/lib/gnome-terminal/gnome-
terminal-server
rino     pts/2    192.168.1.5 11:31    5.00s  0.02s  0.00s vi /etc/resolv.conf
```

2 システム全体のプロセス状態を一覧表示する

```
$ ps aux
```

システムで稼働しているプロセスを見るには、一覧表示するならpsコマンド、リアルタイムで負荷などを表示するなら後述のtopコマンドを利用します。

psコマンドはオプションを指定しないで実行すると、その端末で実行しているプロセスのみを表示します。次の例はGNOME端末上で実行したので、シェルのbashと実行したpsコマンドが検知されています。ヘッダに表示されているのはPID(プロセスID)、TTY(端末番号)、TIME(実行時間)、CMD(プロセス名)です。

```
$ ps
  PID TTY          TIME CMD
 5819 pts/1    00:00:00 bash
 5833 pts/1    00:00:00 ps
```

しかし、現在の端末上のプロセスがわかってうれしいときなど、ほとんどありません。実際はシステム全体の状況を知りたいときが多いので、その場合は冒頭のコマンドを実行します。オプションにすべてのプロセスを表示する「a」、実効ユーザ名を表示する「u」、端末を持たないプロセスも表示する「x」を指定します。

図2がその実行結果です。左からUSER(ユーザ名)、PID(プロセスID)、%CPU(CPU利用率)、%MEM(メモリ利用率)、VSZ(プロセスが起動した際に確保される仮想メモリ)、RSS(物理メモリ利用量)、TTY(端末)、STAT(プロセス状態)、START(起動時刻)、TIME(CPU利用時間)、COMMAND(プロセス名)を示します。STATはD(スリープ)、R(実行可能状態)、S(スリープ)、T(停止)、Z(ゾンビ化)などがあります。

画面に収まりきらない場合は`ps aux | less`など、「|(パイプ)」と`less`コマンドなどのページャに渡してじっくり見るのがいいでしょう。

プロセスには親子関係があり、たとえば、httpdであれば最初に1つのプロセスが起動し、そのプロセスが複数の子プロセスを生成します。その子プロセスたちが、別ホストで稼働するWebブラウザからのリクエストに答えます。このようなプロセスの親子関係はpsコマンドにfオプションを付けることで見られます(**図3**)。また、`pstree`コマンドがインストールされていれば、簡略化されたプロセスツリーを見

▼図2 システム全体のプロセスを表示する

```
$ ps aux
USER   PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root     1  0.0  0.0  13104   876   ?        Ss 11:34   0:00 init [2]
root     2  0.0  0.0      0     0   ?        S  11:34   0:00 [kthreadd]
root     3  0.0  0.0      0     0   ?        S  11:34   0:00 [ksoftirqd/0]
root     5  0.0  0.0      0     0   ?        S< 11:34   0:00 [kworker/0:0H]
```

▼図3 プロセスの親子関係を表示する

```
$ ps axf
  PID    TTY   STAT   TIME COMMAND
【略】
 3736      ?     Ss   0:00 /usr/sbin/apache2 -k start
 3739      ?      S   0:00  \_ /usr/sbin/apache2 -k start
 3740      ?      S   0:00  \_ /usr/sbin/apache2 -k start
 3741      ?      S   0:00  \_ /usr/sbin/apache2 -k start
【略】
```

られます(**図4**)。

プロセスが暴走した場合などは、そのプロセスに対して対応しなければいけないので、プロセスリストからの特定が必要です。あとで出てくるkillコマンドではPID(プロセスID)を指定するので、これを得るために**grep**コマンドと連携しましょう。**図5**の例はapache2プロセスに条件を絞って表示するpsコマンドの例です。

これよりは**pgrep**コマンドでも可能です。pgrepコマンドはプロセス名を指定すればそのPIDを出力してくれます。プロセス名を指定しただけではPIDのみを表示しますが、-lオプションを付ければプロセス名も表示するので、目的のプロセスにマッチしているか確認ができます(**図6-①**)。また、-dで出力項目間に挟む文字として空白を指定すれば、1行にPIDを出力でき(**図6-②**)、ほかのコマンドの引数に渡すのも楽になりますね。

▼図4　プロセスツリーを表示する

```
$ pstree
(略)
  |-tmux-+-bash-+-lv
  |      |      `-pstree
  |      `-bash
(略)
```

▼図5　プロセス名からプロセスIDを調べる(psとgrep)

```
$ ps ax | grep apache2
17614  pts/0     S+   0:00 grep apache2
18790      ?     S    0:00 /usr/sbin/ ➥
apache2 -k start
18791      ?     Sl   0:02 /usr/sbin/ ➥
apache2 -k start
18792      ?     Sl   0:02 /usr/sbin/ ➥
apache2 -k start
32308      ?     Ss   0:31 /usr/sbin/ ➥
apache2 -k start
```

▼図6　プロセス名からプロセスIDを調べる(pgrep)

```
$ pgrep apache -l   ←①
18790 apache2
18791 apache2
18792 apache2
32308 apache2
$ pgrep apache -d ' '   ←②
18790 18791 18792 32308
```

もう1つのプロセス状態把握のツールtopを実行してみましょう。**図7**はオプションなしで実行した画面です。標準の更新間隔は3秒ですが、これが長い場合は-dオプションに更新間隔を秒単位で指定するといいでしょう。

PIDから始まる黒い帯から上部分をサマリーエリア、下部分をタスクエリアと呼びます。サマリーエリアの1行目にはプログラム名(ここではtop)、現在時刻、システム起動時間、ログインユーザ数、1/5/15分間隔でのロードアベレージが表示されます。2行目は状態に合わせたプロセスの統計数、3行目はCPUの状態、4行目はメモリの利用状況、5行目はスワップメモリの利用状況が表示されます。topを終了する場合は[q]を押しましょう。

普段の利用で「なんだか動作が重いなぁ」と思ったらtopを見てみましょう。重いプロセスがタスクエリアの上のほうに居座っているかもしれません。サマリーエリアの4行目メモリ部分でfreeで示されている空きメモリがなく、5行目スワップメモリのfreeの空き領域も少ない場合は、何かプロセスを抹殺したほうがいいかもしれません。

デフォルトのタスクエリアの見方は、左からPID(プロセスID)、USER(実効ユーザ)、PR(タスクの実行優先順位)、NI(タスクのnice値)、VIRT(仮想メモリサイズ)、RES(利用物理メモリ)、SHR(利用共用メモリ)、S(プロセス状態)、%CPU(CPU利用時間)、%MEM(物理メ

▼図7　リアルタイムにプロセス状態を表示する

```
top - 16:51:08 up  5:17,  3 users,  load average: 0.49, 0.38, 0.29
Tasks: 218 total,   1 running, 217 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.4 us,  0.2 sy,  0.0 ni, 99.2 id,  0.2 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem:   7897008 total,  5394844 used,  2502164 free,   205944 buffers
KiB Swap:  7812496 total,        0 used,  7812496 free.  3158732 cached Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
 9201 ryosuke   20   0 1935504 668204  48272 S   2.7  8.5  13:52.35 iceweasel
 8922 ryosuke   20   0 1880316 351764  34308 S   2.0  4.5   2:13.55 gnome-shell
 4504 root      20   0  171708  25964   9644 S   0.7  0.3   2:33.42 Xorg
 8950 ryosuke   20   0  297456   5968   4604 S   0.3  0.1   0:00.61 mission-co+
15462 ryosuke   20   0   23164   1708   1168 R   0.3  0.0   0:00.01 top
15468 ryosuke   20   0  644740  16740  12824 S   0.3  0.2   0:00.11 gnome-scre+
23908 ryosuke   20   0  813108  70552  24396 S   0.3  0.9   2:30.96 plugin-con+
    1 root      20   0   13104    876    728 S   0.0  0.0   0:00.53 init
    2 root      20   0       0      0      0 S   0.0  0.0   0:00.00 kthreadd
    3 root      20   0       0      0      0 S   0.0  0.0   0:00.78 ksoftirqd/0
    5 root       0 -20       0      0      0 S   0.0  0.0   0:00.00 kworker/0:+
    7 root      20   0       0      0      0 S   0.0  0.0   0:04.41 rcu_sched
    8 root      20   0       0      0      0 S   0.0  0.0   0:00.00 rcu_bh
    9 root      rt   0       0      0      0 S   0.0  0.0   0:00.06 migration/0
   10 root      rt   0       0      0      0 S   0.0  0.0   0:00.04 watchdog/0
   11 root      rt   0       0      0      0 S   0.0  0.0   0:00.04 watchdog/1
   12 root      rt   0       0      0      0 S   0.0  0.0   0:00.00 migration/1
```

モリ利用率)、TIME(実行時間)、COMMAND(プロセス名)となります。CPU利用時間やメモリ利用率、稼働時間を見てシステム負荷の判断材料にしましょう。

3 複数プロセスをまとめて終了させる

```
$ killall apache2
```

Linuxでプロセスが暴走しているなど、諸事情ですぐに特定のプロセスを止めたいときは、**kill**コマンドを使います。直訳で「殺す」という意味で物騒ですが、実はkill(1)のマニュアルには「プロセスを殺す」とは書かれておらず、「プロセスにシグナルを送る」コマンドと説明されています。

シグナルとはプロセスに対して再起動や強制終了などを依頼するメッセージです。プロセスはシグナルを受け取るとその依頼に沿って動作します。

シグナル一覧はkillコマンドに-lオプションを指定すると表示されます(**図8**)。プロセスに送るシグナルを指定するには、**kill -9 12345**のように、1、2、3のような数字か、SIGHUP、SIGINT、SIGQUITの文字列のどちらかを「-」の後ろに付け、さらに対象のプロセスIDを付けて実行します。これらの意味は**man 7 signal**で確認できます。シグナルは重要な概念ですが、ここではプロセスを終了させることについてだけ扱います。

シグナルを指定せずにkillコマンドを実行すると、プロセスを終了させる15)SIGTERMが送られます。プロセスがこのシグナルを受け取ると、決められた終了処理を実施してから終了します。もしプロセスがこの終了処理を実行できない状態で止まっている場合は、何も反応がなくプロセスが稼働しているように見えるでしょう。このような状況で使うのが9)SIGKILLです。このシグナルを受け取ったプロセスは強制終了します。終了処理がされるわけではないのでファイルを書き込んでいる最中だった場合は、ファイルが破損する可能性もあります。

まとめてプロセスを処理したい場合は、冒頭に挙げた**killall**コマンドが便利です。**killall apache2**のようにプロセス名を指定し、該当する名前のプロセスにシグナルを送ります。

▼図8 シグナル一覧を表示する

```
$ kill -l
1) SIGHUP       2) SIGINT       3) SIGQUIT
4) SIGILL       5) SIGTRAP
6) SIGABRT      7) SIGBUS       8) SIGFPE
9) SIGKILL     10) SIGUSR1
(略)
```

4 コマンドラインでIPv4の設定を行う

```
# ifconfig eth0 addr 192.168.1.111 netmask 255.255.255.0
```

Linuxが自宅や職場のLANに接続するには、ネットワーク設定が必要です。そのネットワークでユニークなIPアドレスが設定され、ほかのネットワークへアクセスするためのゲートウェイの設定などは確認できないと困りますし、そして必要に応じて設定を変更できる必要があります。

定番のIPアドレス確認／設定コマンドは**ifconfig**です。/sbin/以下に格納されているので、一般ユーザで環境変数PATHが通っていない場合には「/sbin/ifconfig」と実行しましょう。**図9**では引数にeth0を渡しています。この

出力から「HWaddr」でNICのMACアドレス、「inet addr」／「Bcast」／「Mask」でIPv4アドレス／ブロードキャスト／ネットマスク、「inet6 addr」でIPv6アドレスが設定されているのがわかります。

行頭に「UP」のある行から下はインターフェースの状況を示します。UPはインターフェースが稼働状態になっているかどうかです。未稼働状態ではここに「UP」が表示されず、「DOWN」も書かれず、ifconfigコマンドに-aオプションを付けないとインターフェースすら見えません。BROADCASTとMULTICASTはそれぞれブロードキャスト、マルチキャストをサポートしていることを指します。RUNNINGはこのインターフェースを使ってパケットが転送されていることを示します。

RXは受信したパケット、TXは送信したパケット、collisionsは衝突パケット数、RX bytesは受信したバイト数、TX bytesは送信したバイト数です。

RHEL系では/etc/sysconfig/network-scripts/ifcfg-インターフェース名、Debian系では/etc/network/interfacesにインターフェースの設定を記述して、**service**コマンドでネットワーク設定をリロードすれば、その設定が有効になるでしょう。しかし、そうもいかない状況が……ある場合もあるので、ここではifconfigを使ってIPv4を設定する方法を見ておきましょう（管理者権限が必要です）。それが冒頭のコマンドです。

第1引数はeth0というインターフェースを指定します。**man interface**の「書式」にもあるように、第1引数はインターフェースに決まっているので、順番に気をつけましょう。それ以降は、addr引数のあとにIPv4アドレス、net mask引数のあとにネットマスク値を指定します。実行後、**ifconfig eth0**を実行してアドレスが変わっていることを確認しましょう。実はこれだけでは不十分な場合があり、ifconfigで異なるネットワークアドレスのIPv4アドレスを変更したあとは**route**コマンドでのゲートウェイ設定が必要になります。

routeコマンドはオプションも引数も与えないで実行すると、現在のルーティングテーブル一覧を表示します（**図10**）。「Destination」は宛先ネットワークホスト、「Gateway」がゲートウェイの指定です。宛先が「0.0.0.0」の行がデフォルトゲートウェイの状態です。

図11はネットワーク状態を表示する**netstat**コマンドで、複数設定されたルーティ

▼図9　IPアドレスなどを確認する

```
$ sbin/ifconfig eth0
eth0      Link encap:Ethernet  HWaddr d0:27:88:XX:XX:XX
          inet addr:192.168.1.98  Bcast:192.168.2.255  Mask:255.255.255.0
          inet6 addr: fe80::d227:88ff:XXXX:XXXX/64 Scope:Link
          inet6 addr: 2001:380:XXXX:XXXX:d227:88ff:XXXX:XXXX/64 Scope:Global
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:376744 errors:0 dropped:130 overruns:0 frame:0
          TX packets:273601 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:387490821 (369.5 MiB)  TX bytes:77572048 (73.9 MiB)
```

▼図10　ルーティングテーブルを表示する

```
$ route
Kernel IP routing table
Destination     Gateway         Genmask         Flags   MSS Window  irtt Iface
0.0.0.0         192.168.2.1     0.0.0.0         UG        0 0          0 eth0
192.168.2.0     0.0.0.0         255.255.255.0   U         0 0          0 eth0
```

ングテーブル一覧を表示します。表示される項目はrouteも**netstat -rn**も同じです。ここで「192.168.1.0」と「192.168.5.0」ネットワークは192.168.0.1のゲートウェイを利用するように設定されていることがわかります。

ゲートウェイの追加は**図12**のようにコマンド実行します。**add**コマンドで-netと対象ネットワーク、netmask引数に対象ネットマスク値、経路であるゲートウェイアドレスをgw引数と合わせて指定します。削除するときはaddをdelに変えて実行すれば消えます。ゲートウェイの追加は管理者権限が必要です。

実行したらrouteコマンドでルーティングテーブルを表示してみましょう(**図13**)。期待どおりにゲートウェイが追加されたでしょうか。「172.16.111.0」から始まる行の設定が追加されたことが確認できますね。

ifconfigとrouteを使ってネットワーク設定しましたが、両方をこなせる**ip**コマンドというものもあります。興味が湧いたらマニュアルを見て使ってみてください。

世の中は便利なもので、DHCPが稼働している環境であれば、**dhclient eth0**のように**dhclient**コマンドを実行するとIPアドレスなどが自動設定されると思います。

▼図11　ネットワーク状態を表示する

```
$ netstat -rn
Kernel IP routing table
Destination     Gateway         Genmask         Flags   MSS Window  irtt Iface
0.0.0.0         219.XXX.XXX.XXX 0.0.0.0         UG        0 0          0 eth0
172.16.1.0      0.0.0.0         255.255.255.0   U         0 0          0 eth2
192.168.0.1     0.0.0.0         255.255.255.255 UH        0 0          0 tun0
192.168.1.0     192.168.0.1     255.255.255.0   UG        0 0          0 tun0
192.168.2.0     0.0.0.0         255.255.255.0   U         0 0          0 eth1
192.168.2.0     0.0.0.0         255.255.255.0   U         0 0          0 br0
192.168.5.0     192.168.0.1     255.255.255.0   UG        0 0          0 tun0
219.XXX.XXX.XXX 0.0.0.0         255.255.255.248 U         0 0          0 eth0
```

▼図12　ゲートウェイを追加する

```
# route add -net 172.16.111.0 netmask 255.255.255.0 gw 192.168.2.100
```

▼図13　ルーティングテーブルを確認

```
$ route
Kernel IP routing table
Destination     Gateway         Genmask         Flags   MSS Window  irtt Iface
0.0.0.0         192.168.2.1     0.0.0.0         UG        0 0          0 eth0
172.16.111.0    192.168.2.100   255.255.255.0   UG        0 0          0 eth0
192.168.2.0     0.0.0.0         255.255.255.0   U         0 0          0 eth0
```

5 デバイスを利用しているユーザ／プロセスを調べる

$ lsof /media/

USBメモリや外付けHDDをマウントして使っていたが、メンテナンスのために**umount**コマンドで取り外そうとしたら、**図14**のようなエラーが表示されました。そんな場合はメッ

セージに書いてあるとおり、**lsof**か**fuser**で原因がわかります。

冒頭で挙げたlsofコマンドを使ってみましょう。lsofの引数にアンマウントしようとした/mediaを与えてみます(**図15**)。結果、プロセスbashがPID29764としてユーザryosukeに実行されていることがわかります。FDが「cwd」になっているのでユーザが/mediaにとどまっているのでしょう。

この場合、ユーザryosukeに違うディレクトリへ移動してもらうか、プロセスIDがわかっているので**kill -9 29764**でプロセスをkillすればumountを実行することができます。

lsofはストレージ以外にもポートでのプロセス特定もできます。**図16**はTCPの80番を利用しているプロセスの情報を表示しています。見知らぬポートが開いているな、と感じたらすかさずこれで利用しているプロセスを調べましょう。この出力例では、apache2プロセスをユーザwww-dataが実行しているのがわかります。

▼図14　umount時のエラーメッセージ

```
$ sudo umount /media
umount: /media: device is busy.
        (In some cases useful info about processes that use
         the device is found by lsof(8) or fuser(1))
```

▼図15　ストレージを利用しているユーザ／プロセスを把握する

```
$ lsof /media/
COMMAND   PID    USER   FD   TYPE DEVICE SIZE/OFF NODE NAME
bash    29764 ryosuke  cwd    DIR   8,17     4096    1 /media
```

▼図16　ポートを利用しているユーザ／プロセスを把握する

```
# lsof -i:80
COMMAND     PID     USER   FD   TYPE   DEVICE SIZE/OFF NODE NAME
apache2   14370     root    4u  IPv6 18715417      0t0  TCP *:http (LISTEN)
apache2   14373 www-data    4u  IPv6 18715417      0t0  TCP *:http (LISTEN)
apache2   14374 www-data    4u  IPv6 18715417      0t0  TCP *:http (LISTEN)
apache2   14375 www-data    4u  IPv6 18715417      0t0  TCP *:http (LISTEN)
```

6 Webサーバに接続してサービスの稼働確認を行う

$ telnet www.gihyo.jp 80

「接続先のサーバにはpingが通るけど、Webサーバは生きているか？　SMTPサーバは生きているか？」というときに、GUIがなければ、サービスの稼働確認はどうしたらいいでしょうか。

```
例❶ $ w3m http://www.gihyo.jp
例❷ $ wget http://www.gihyo.jp
例❸ $ curl http://www.gihyo.jp
例❹ $ telnet www.gihyo.jp 80
```

コマンドラインで使えるブラウザがあります。例①の**w3m**ブラウザをインストールしていれば、Webサイトへのアクセスを確認できます(**図17**)。

ほかに例②の**wget**や例③の**curl**は引数に渡したURLのページを取得、出力するのでこちらもWebサーバの稼働確認に使えます。

例④の**telnet**コマンドは対話的にWebサーバにアクセスします。これはもともとtelnetサーバに接続し、TELNETプロトコルを利用して

リモートメンテナンスするツールです。telnet通信は暗号化されておらず、認証のためのパスワードが平文でやりとりされます。そのため盗聴を危惧して現在はSSHを利用してリモートログインするのが一般的になりました。

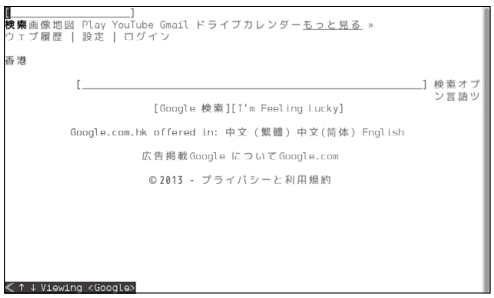
telnetはホスト名とポート番号を指定すれば、TELNETプロトコルを使わない通信ができます。ここではそれを利用してWebサーバとして技術評論社のサイトに接続します(**図18**)。telnetの引数に、対象ホスト名かIPアドレスと、ポート番号を指定します。Webサーバから返事が返ってきたら、HTTPの**GET**コマンドと、取得するPATH、HTTPバージョンを入力します(❶)。その後、空行を入力するために、何も文字を入れずに Enter を押します。数行レスポンスがあったあと、そっけなくCloseされてしまいました。

❷を見るとステータスコード302が返ってきています。技術評論社のトップページを取得しようとしましたが、別の場所を参照するように言われています。❸のLocationがその参照先ですね。

HTTPは基本的に1つの接続で1つのファイルを取得します。302のコードとLocationが返ってきてしまったら、もう一度アクセスしてURLを指定しなおさなければいけません。Webブラウザはページにリンクされたimgタグやlinkタグなどを参照して画像やCSSファイルを都度接続／取得します(Webサーバの設定によっては、ページにリンクされているリソースすべてを取得するまで接続を切らないということもできます)。

ブラウザはページ内の別ファイルの取得にも、「別の場所を参照しろ」ということにも、ちゃんと対応してページを表示してくれてエラいですね。では、その参照先にアクセスしてみましょう(**図19**)。無事に技術評論社のトップページが得られましたね。

▼図17　w3mでのWebサイト表示



▼図18　telnetでWebサーバに接続する(初回)

```
$ telnet www.gihyo.jp 80
Trying 49.212.34.191...
Connected to www.gihyo.jp.
Escape character is '^]'.
GET / HTTP/1.0   ←❶
                 ←何も入力せずにENTER
HTTP/1.1 302 Moved Temporarily   ←❷
(略)
Location: /?ard=1395297528   ←❸

Connection closed by foreign host.
※下線部分はユーザが入力する部分を表す
```

▼図19　telnetでWebサーバに接続する(2回目)

```
$ telnet www.gihyo.jp 80
Trying 49.212.34.191...
Connected to www.gihyo.jp.
Escape character is '^]'.
GET /?ard=1395297528 HTTP/1.0

HTTP/1.1 200 OK
(略)
<!DOCTYPE html>
(略)
<title>トップページ | gihyo.jp ... 技術評論社</title>
<meta name="description" content="" />
<meta name="keywords" content="技術評論社,gihyo.jp," />
※下線部分はユーザが入力する部分を表す
```

7 curlを使ったサーバとのデータ取得／送信

```
$ curl -k -o data.txt -u ryosuke scp://example.com:22/tmp/ ➥
data.txt
$ curl -k --upload-file data.txt -u ryosuke scp://example. ➥
com:22/tmp/
```

コマンドラインでの強力なツールの1つにcurlがあります。「Client for URL」の略で、ネットワークでつながるリソースURLに対するクライアントツールです。さまざまなプロトコルをサポートし、豊富な機能を持っています。

先ほどもcurlコマンドは登場していますが、HTTPプロトコルを利用してファイルのダウンロードに利用していました。引数にURLを指定すれば、標準出力に得られたファイルを出力してくれます。htmlファイルを取得する場合は、-oで保存先ファイル名を指定するか、「|」でテキストブラウザに入力(**curl http://gihyo.jp/ | w3m**を実行してvを入力)すれば、HTML表示を確認できます。

認証が必要なページにアクセスする場合は、**図20**のように実行すれば、アクセスできます。例①がSSLクライアント認証、例②がBASIC認証、例③がDIGEST認証になります。鍵ファイルにパスワードがかかっている際は、「***証明書ファイル:パスワード***」のように引数を指定できます。-kは自己署名証明書を使っているサーバに対してアクセスする際に、「認識済み」を伝えるオプションです。

コマンドラインからHTTP POSTメソッドも実行できます。**図21**はWiki実装の1つであるHikiの検索ボタンを使って「WORD」を検索した例です。出力に検索結果のHTMLが出てきます。

また、curlでメールを送ることができます。**図22**では、--upload-fileオプションで指定するテキストファイルにメッセージを込めて送信してくれます。ファイルに「From:」と「To:」、「Subject:」の文字列があれば、相手のメールクライアントでもそれが表示されます。

SSH経由でファイル転送する**scp**も利用できます。ファイルを取得する場合は、冒頭のコマンド(1行目)のように-oオプションでファイルの保存先を指定します。

scpでアップロードする場合は、冒頭のコマンド(2行目)のように--upload-fileオプションで転送ファイルを指定します。

応用技として、**図23**はcurlを使ってチケッ

▼図20　認証が必要なページにアクセスする

```
例❶ $ curl -k --cert 証明書ファイル --key 鍵ファイル --basic -u ユーザ名:パスワード URL
例❷ $ curl --basic -u ユーザ名:パスワード URL
例❸ $ curl --digest -u ユーザ名:パスワード URL
```

▼図21　HTTP POSTメソッドを実行する

```
$ curl --data "key=WORD&comment=search&c=search" http://example.com/hiki.cgi
```

▼図22　メールを送る

```
$ curl --mail-from ryosuke@example.com --mail-rcpt ryosuke@example.jp --upload-file msg.txt ➥
smtp://example.com:25
```

ト管理システムTracに新しいチケットを追加する様子です。BASIC認証だけがかかっているサイトですので、--basicと-uでユーザ名とパスワードを指定します。❶はTracにアクセスしてCookieを--cookie-jarで取得します。最近のTracはチケット登録にトークンを必要としているため、❸の登録コマンドで利用するために、❷のコマンドでCookieからトークンを取り出します。❸で--dataと--data-urlencodeを使ってPOSTデータを作り、サーバに送ります。URLエンコードが必要なパラメータのみを--data-urlencodeオプションの引数に渡します。

これで変数の値を入れ変えて実行するスクリプト作れば、いちいちブラウザでちょこちょこしなくても楽にチケット登録ができますね！

curlはHTTPヘッダを出力したり、複数NICを持つホストでどのインターフェースからアクセスするかを指定できたり、FTPやtelnetをしたりとできることが豊富です。自動化に使えるコマンドですのでマニュアル読んで習得しましょう。

▼図23　Tracにチケットを追加する

```
$ curl --basic -u ryosuke:passw0rd \  ←❶
--cookie-jar trac.cookie --output trac.html \
http://example.com/trac/sd/newticket
$ COOKIE=`tail -n1 trac.cookie | awk '{ print $7 }'`  ←❷
$ curl --basic -u ryosuke:passw0d \  ←❸
--cookie-jar trac.cookie --cookie trac.cookie \
--data "__FORM_TOKEN=$COOKIE" \
--data-urlencode "field_summary=テスト" \
--data-urlencode "field_description=テストチケット" \
--data
"field_type=defect&field_priority=major&field_milestone=&field_component=component1&field_
version=&field_keywords=&field_cc=&field_owner=&submit=SUBMIT" \
http://example.com/trac/sd/newticket
※行末に\を入力することで、複数行に渡ってコマンドラインを記述できる
```

8 httpsサーバに接続する

```
$ openssl s_client -host example.com -port 443
```

SSLを利用しないWebサーバへの接続はtelnetでできるのを確認しました。ここではSSLを利用して暗号化通信をしているサーバへの接続を試してみましょう。

telnetではSSL通信を手作業で処理するのは面倒ですので、openssl パッケージがインストールされているのなら、そちらを使いましょう。まずは、**openssl**コマンドの内部コマンド**s_client**を利用します(**図24**)。-hostで接続先ホスト名、-portで接続先ポート番号を指定します。ほかのコマンドは1文字オプションには「-」が1つで、2文字以上のオプションには2つの場合が多いのですが、openssl コマンドは、長い名前のオプションでも「-」が1つですので気をつけてください。

openssl s_client実行後、サーバ証明書のダウンロードや検証がされていることを確認できますね。証明書が返ってくるところまでくれば、あとはtelnetでのHTTPのやりとりと同じです。コンテンツを確認してください。

▼図24　httpsサーバに接続する

```
$ openssl s_client -host example.com -port 443
CONNECTED(00000003)
depth=0 C = COM, O = EXAMPLE, CN = example.com
verify error:num=18:self signed certificate
verify return:1
depth=0 C = COM, O = EXAMPLE, CN = example.com
verify return:1
---
Certificate chain
 0 s:/C=COM/O=EXAMPLE/CN=example.com
   i:/C=COM/O=EXAMPLE/CN=example.com
---
Server certificate
-----BEGIN CERTIFICATE-----
MIIDozCCAougAwIBAgIJAOvqkzGPQ0FNMA0GCSqGSIb3DQEBBQUAMD8xCzAJBgNV
BAYTAkpQMRUwEwYDVQQKEwxERUVSIE4gSE9SU0UxGTAXBgNVBAMTEGRlZXItbi1o
b3JzZS5uZXQwHhcNMTMwNzI2MDc0MDMzWhcNMjMwNzI0MDc0MDMzWjA/MqswCQD
(略)
---
GET /index.php HTTP/1.0
Host: example.com

HTTP/1.1 200 OK
Date: Sat, 22 Mar 2014 13:55:46 GMT
Server: Apache/2.2.22 (Debian)
X-Powered-By: PHP/5.4.4-14+deb7u7
(略)
※下線部分はユーザが入力する部分を表す
```

9 ファイル名に自動的に日付を入れる

```
# tar cfvJ /var/backups/etc-`date +%Y-%m-%d`.tar.xz /etc
```

dateコマンドは日時を設定／表示するコマンドで、オプションも引数も渡さずに実行すると、今の日時を表示します。この日時表示は、次のように柔軟にカスタマイズできるので、有効活用したいコマンドの1つです。

```
$ date +%Y-%m-%d
2014-03-18
```

dateコマンドが重宝するのはバックアップファイル名の指定などです。ファイルをバックアップしたり、圧縮ファイルを作成するときに「backup.tar.gz」や「backup.zip」など、ファイルのタイムスタンプをたよりにバックアップを作成するのは、非常に危険です。「backup-2014-05-18.tar.gz」のようにファイル名に日付や時間が入るのが理想的ですが、これをわざわざカレンダーや時計を見て圧縮ファイル名を指定するのではなく、dateコマンドから生成してもらうのがいいでしょう。

圧縮ファイルを作成する場合は、**図25**のようにコマンドを実行します。ここでは**tar**コマンドで/etc/ディレクトリを1つのファイルにまとめ、XZ圧縮して/var/backups/etc-2014-05-18.tar.xzとして書き込まれます。

dateコマンドは今の日付のみを扱うわけではなく、-dオプションで指定した文字列に対応した日付を出力することもできます。この指定文字列が比較的柔軟にできています。次の例では「3 days ago」を-dオプションに指定すること

で、今日から3日前の日付を整形して出力します。

```
$ date -d "3 days ago" +%Y-%m-%d
2014-03-15
```

バックアップで日付ごとに分けているディレクトリがあるとして、1日前と2日前との差分を取りたい場合に**図26**のように実行すると楽に取得できます。

これは差分を出力するdiffコマンドに、-u(diffのunified形式を利用して出力する)、-N(新しいファイルも比較対象とする)、-r(サブディレクトリも再帰的に探索する)のオプションを付けて、指定した1日前と2日前の2つのディレクトリの差分を出力します。

-dの指定文字列には「2014/05/18」や「1 month ago」などの文字も使えますが、実際に希望した文字列が返ってくるかは、「素振り」をしっかりしてからやりましょう。

▼図25　バックアップを取る際にファイル名に日付を入れる

```
# tar cfvJ /var/backups/etc-`date +%Y-%m-%d`.tar.xz /etc
```

▼図26　1日前と2日前のバックアップファイルを比較する

```
$ diff -uNr /var/backups/etc-`date -d yesterday +%Y-%m-%d` /var/backups/etc-`date -d ➥
"2days ago" +%Y-%m-%d`
```

10 リモートホストにバックアップファイルを作成する

```
# rsync -auvz -e ssh /var/backups/ ryosuke@backupsrv:~/ ➥
mybackups/
```

定期的バックアップはとても大切なことです。ここではtarによるアーカイブと**rsync**で別ディレクトリ／別ホストへのバックアップを利用します。

tarコマンドは複数のファイルを1つのアーカイブファイルにまとめるコマンドです。オプションによっては、圧縮ライブラリを利用してアーカイブを圧縮してファイルを作成します。**図27**では、設定ファイルが詰まった/etc/ディレクトリを/var/backups/etc-*年*-*月*-*日*.tar.xzのファイルに圧縮して保存します。tarのオプション「cfvJ」は前から「アーカイブを作成(Create)」「ファイル名を指定(File)」「詳細出力(Verbose)」「XZ圧縮する(なぜ"J"なんですかね……)」の意味があります。オプションのあとに、出力ファイル名と対象ファイル(この場合は/etcディレクトリ)が指定されます。

tarで圧縮しただけでは/var/backups以下にアーカイブがたまる一方です。これをある程度経ったら、古いファイルを削除するようにしましょう。**図28**では、**find**コマンドを使って/var/backups/以下を対象に、最終更新日が3日以前のファイルを抽出し、**-exec rm**で削除を実施しています。findコマンドに**-type f**オプションを付けることでファイルのみが対象になり、さらに、-ctimeで最終更新日3日以前のファイルが対象になります。

tarでのバックアップもいいですが、ローカルへのファイルコピーでは、そのPCやHDDがクラッシュした際が心配です。ここはリモートのホストに転送するようにしましょう。SSH接続できるバックアップファイルコピー先ホス

トがあるとして、図29のように実行します。

この例では、/etcをアーカイブすることは変わりませんが、出力ファイルを示す引数に「-」が指定されています。コマンドオプションで出力ファイルを指定できる場合、それに「-」を指定すると、標準出力に出力されることがUNIXコマンドにはよくあります。ここではそれを利用しています。

また、続く「|」の先のsshでは、バックアップ先ホストへのログインオプションのあとに、catコマンドを「"」で囲んで付けています。sshはログインオプションのあとにコマンドを記述すれば、それをログイン先で実行してログアウトして戻ってきます。今回はコレを利用して、tarコマンドの出力を、catコマンドの標準入力として読み込み、/tmp/backups以下にアーカイブを出力します。catコマンドはとくに指定しなければ、標準入力からの入力を標準出力に書き出します。

アーカイブファイルでの保存であれば、これでもいいでしょう。しかし、アーカイブではなくそのままのファイル構造をバックアップしておきたい場合は、rsyncコマンドを利用するのがいいでしょう。rsyncコマンドはリモートやローカルにファイルをコピーできるツールです。

冒頭に挙げたコマンドでは、ローカルの/var/backupsディレクトリ以下を対象に、rsyncコマンドを実行しています。rsyncコマンドには-auvz -e sshのオプションを付けています。「ryosuke@backupsrv:~/mybackups/」はバックアップ先の指定で、ホスト名がbackupsrvの/home/ryosuke/mybackups/以下にコピーすることを示しています。

rsyncのオプションは、「a」はアーカイブモード(Archive)と呼ばれるオプションで、指定したディレクトリ以下をシンボリックリンク、ファイルの権限、タイムスタンプ、オーナー／グループ権限、デバイス／特殊ファイルも含めてバックアップします。「u」はバックアップ先より新しい(Update)ファイルを転送、「v」は詳細な出力(Verbose)、「z」は圧縮することを示します。

「u」オプションを利用すると、新規／更新されたファイルのみをバックアップ先に転送できるため、毎回大容量転送になりません。そこがcpやtarよりも便利なところです。

rsyncはネットワークを経由せずにローカルストレージへのコピーもできます。次のように実行すれば、/etc以下で追加／更新されたファイルだけがバックアップされます。

```
$ rsync -auvz /etc /var/backups/etc
```

cronなどで実行させる際は「v」オプションをはずさないと、メールが飛びます。テストを行ったあとは「v」オプションを外し、標準エラーのみがメールで飛ぶように調整しましょう。SD

▼図27　バックアップファイルを作成する

```
# tar cfvJ /var/backups/etc-$(date +%Y-%m-%d).tar.xz  /etc
```

▼図28　最終更新日が3日以前のファイルを削除する

```
# find /var/backups/ -type f -ctime +3 -exec rm -f {} \;
```

▼図29　tarでリモートホストにバックアップファイルを作成する

```
$ tar cvfJ - /etc | ssh -l ryosuke 192.169.1.4 "cat > /var/backups/backup-$(date
+%Y-%m-%d).tar.xz"
```

第4章 共同作業で役立つコマンド ——複数OS間でのファイル共有、文字コード対応

世の中には、さまざまな人種、国と言語、文化、習慣があるように、コンピュータを使って仕事する際には、自分が使うOSとは違うOSを使っている人とのコミュニケーションが必要です。ここでは、WindowsやMac OS Xを使っている人との共同作業を前提に、ファイルのやりとりができるようになりましょう。

Windows共有へのアクセス

Windowsを利用することが多い環境では、Windowsのファイル共有用のサーバが用意されていると思います。この場合は、**smbclient**コマンドを利用するとファイル一覧の取得やファイルのダウンロード／アップロードが手軽にできます。

まずは、Windowsファイル共有にアクセスできるかを確認しましょう。-Lオプションに接続先ファイル共有ホストのIPアドレスかホスト名を指定します。**図1**ではIPアドレス192.168.1.4で稼動しているLinux上のSambaサーバに接続します。

Windowsがファイル共有サーバの場合は、接続表示部分が「Domain=[ホスト名] OS=[Windows 7 Home Premium 7601 Service Pack 1] Server=[Windows 7 Home Premium 6.1]」のようになるだけで、その後の使い勝手は変わりません。

smbclientは引数にファイル共有サービス名だけを指定(コマンドラインでは**smbclient //*サーバ名*/*共有名***/)すれば、パスワード認証後に対話的にファイル共有ディレクトリを歩きまわり、ファイルの取得、手元から共有ディレクトリへのファイルアップロードを行えます。

スクリプトで自動実行したい場合などには、対話的な利用は不便ですが、smbclientは1行でのファイルのアップロード／ダウンロード／

▼図1　ファイル共有ホストにアクセスする(対話形式)

```
$ smbclient -L 192.168.1.4
Enter ryosuke's password:
Domain=[ENDEAVER] OS=[Unix] Server=[Samba 3.6.6]

      Sharename       Type      Comment
      ---------       ----      -------
      print$          Disk      Printer Drivers
      IPC$            IPC       IPC Service (endeaver server)
      mfc             Printer   mfc
      ryosuke         Disk      Home Directories
Domain=[ENDEAVER] OS=[Unix] Server=[Samba 3.6.6]

      Server               Comment
      ---------            -------
      ENDEAVER             endeaver server

      Workgroup            Master
      ---------            -------
      ENDEAVER             ENDEAVER
```

閲覧もサポートしています。

すでに共有ファイルサーバ内の目的のファイルへのPATHがわかっているならば、コマンドラインで**図2**のように実行することで、ダウンロード／アップロードができます。-cオプションで、実行するコマンドとファイルを「'（シングルクォート）」で括って指定することで、1行で実行できます。たとえば、ファイルの取得であれば「**get ファイル名**」、ファイルのアップロードであれば「**put ファイル名**」をコマンドとして指定します。複数ファイルを扱う場合は、それぞれ**mget**、**mput**に変えて使ってください。

テキストファイルの中身を見る程度であれば、**図3**のように**more**コマンドを使えば、手元にダウンロードする手間が減り、ディスクにもやさしいですよね。終了する際は、q を入力するとファイル内容表示から抜けます。

ファイル共有に認証が必要であれば、**図4**の形式のテキストファイルを作成して-Aオプションで指定すると、パスワードを読み込んでくれます。便利な機能ですが、パスワードの書いてあるテキストファイルは他人に見えないように適切にパーミッションを設定しましょう。

特定のディレクトリ以下をまとめて取得する場合は、**図5**のように実行します。

Web共有へのアクセス

Windowsでも UNIX環境でも、手軽に利用できるWeb共有のWebDAV[注1]もよく利用される共有方法の1つです。ここへのアクセス方法を見てみましょう。1つは**cadaver**コマンドの利用です。認証情報は~/.netrcに記述します。

cadaverには引数としてWebDAVのURLを指定します。アクセスができると**図6**のように表示されます。

注1）Web-based Distributed Authoring and Versioning

▼図2　共有ホストのファイルをダウンロード／アップロードする（1行で実行）

```
$ smbclient //ENDEAVER/share -c 'get \photos\2014-04-01\IMG-0001.jpg'
$ smbclient //ENDEAVER/share -c 'put \photos\2014-04-01\index.html'
```

▼図3　共有ホストのファイルを参照する（1行で実行）

```
$ smbclient //ENDEAVER/share -c 'more \photos\README'
```

▼図4　ファイルからパスワードを読み込んで認証する

```
$ cat ~/.smbshare
username = ryosuke
password = passw0rd
domain = endeaver
$ chmod 600 ~/.smbshare
$ smbclient //ENDEAVER/share -c 'get \photos\2014-04-01\IMG-0002.jpg' -A ~/.smbshare
```

▼図5　ディレクトリ以下のファイルをまとめてダウンロードする

```
$ smbclient //ENDEAVER/share -c 'prompt; recurse; mget \photos/2014-04\matome\'
```

▼図6　WebDAVサーバにアクセスする

```
$ cadaver http://example.com/dav/
dav:/dav/> ls
Listing collection `/dav/': succeeded.
Coll:   無題のフォルダー                    0   2月 15  2013
Coll:   ProjectDNH                          0   9月 22  2012
Coll:   photos                              0  12月 16  2013
        README                             52  11月 27  2009
        index.html                       8047   2月  1  2010
```

curlでアップロード

もう1つ、curlを利用してファイルのアップロードもやってみましょう(図7)。curlでは-upload-fileオプションでファイルのアップロードができます。また、IDとパスワードを含めたURLを設定ファイルとして用意すれば、パスワードの入力なしでファイルのアップロードが可能です。設定ファイルは任意のファイル名で構いません。ただ、ほかのユーザに見られないようにchmod 600 .davshareと権限設定しておきましょう。ファイルの中身はURLにIDとパスワードを「:」を挟んだものです。

ファイル名文字化けに立ち向かう

日本語ファイル名がZIP圧縮ファイルに格納されると文字コードの取り扱いがイマイチのため、ディストリビューションに含まれるunzipコマンドではファイル名の文字化けが発生します(図8)。Ubuntu Japanese Teamが提供しているunzipパッケージなら-Oオプションで文字コード指定できますが、ほかのディストリビューションではunarコマンドを利用するのが便利です(図9)。

Debian GNU/Linux、Ubuntu、Fedoraにはunarコマンドが含まれていますが、CentOS 6

▼図7　curlでファイルをアップロードする

```
$ curl -upload-file IMG-0003.jpg http://dav.example.com/photos/
$ cat ~/.davshare
http://ryosuke:passw0rd@dav.example.com/photos/
$ curl -upload IMG-0004.jpg `cat ~/.davshare`
```

▼図8　Windowsで作った日本語ファイル名のZIPを展開(unzip)

```
$ unzip /tmp/ログ20091021.zip
Archive:  /tmp/ログ20091021.zip
  inflating: 20091021/readme.txt
  inflating: 20091021/?[/aptitude.log
  inflating: 20091021/?[/clientlog(TCP).log
  inflating: 20091021/?[/clientlog.txt
$ find .
.
./???020091021
./???020091021/?G???[
./???020091021/?G???[/vpnclient.ini
./???020091021/?G???[/clientlog.txt
./???020091021/?G???[/aptitude.log
```

↑文字化けしている

▼図9　Windowsで作った日本語ファイル名のZIPを展開(unar)

```
$ unar /tmp/ログ20091021.zip
/tmp/ログ20091021.zip: Zip
  ログ20091021/readme.txt  (1188 B)... OK.
  ログ20091021/エラー/aptitude.log  (2958 B)... OK.
  ログ20091021/エラー/clientlog(TCP).log  (15352 B)... OK.
$ find .
.
./ログ20091021
./ログ20091021/正常
./ログ20091021/正常/clientlog.txt
./ログ20091021/正常/aptitude.log
```

↑文字化けしていない

にはまだ入っていないようです。ソースからビルドするか、どこかからSRPMをダウンロードし、ビルド、インストールするしかありません。

convmvコマンドというファイル名の文字コードを変換するperlスクリプトもあります。これは-fオプションで元文字コード、-tオプションで変換先文字コードを指定します(**図10**)。メッセージにあるように、-fと-tオプションだけでは実際の変換は行われません。--notestオプションを付けることで、実際のファイル名変更が行われます。

convmvコマンドは変更対象ファイルを指定する必要があります。しかし、ファイル名が文字化けしている段階で、それをコマンドラインから指定することは難しいので、**図10**のようにワイルドカードを利用する必要が生じます。この例では、もし同じディレクトリに「.zip」を拡張子に持つファイルがあると、単純に***.zip**とは指定できないため、厄介です。実行する場所を選ぶことになってしまうのが無念ですね。

すでに展開してしまったファイルや、もらったUSBメモリに直接入っていたファイルの日本語ファイル名の変換が難しい場合には、GUIが設定されている環境ならば、nautilusを起動してファイルマネージャとして利用し、ファイル名変更するのがいいでしょう。こういうのを適材適所と言います。

半角カナ／全角カナの変換

最近の日本語変換エンジンは半角カナを出してくれるので、あまり意味がないかもしれませんが、**nkf**で半角ファイル名も作ってみましょう(**図11**)。-Z4オプションを付けるとカタカナを半角にしてくれます。入力に漢字がある場合、それは全角のままでキープされます。半角カナから全角カナに戻す場合は-Wオプションを利用します(**図12**)。

まとめ

これからLinuxを利用する人を対象にコマンドの使い方をまとめましたが、この特集でコマンド使いになることを誓ってくれた方が1人でもいれば幸いです。ここで扱ったコマンドはディストリビューションに含まれるごく一部ですし、コマンドオプションもその中のごく一部です。もし興味がわいたら、各マニュアルや開発元Webサイトや Webの情報などを検索して、奥の深さを楽しんでください。SD

▼図10　ファイル名の文字コードを変換する

```
$ convmv -f sjis -t utf8 *.zip
Your Perl version has fleas #37757 #49830
Starting a dry run without changes...
mv "./???{??@?C????.zip"        "./日本語ファイル名.zip"
No changes to your files done. Use --notest to finally rename the files.
```

▼図11　ファイル名の全角カナを半角カナにする

```
$ cp ninjya.txt $(echo "ニンジャスレイヤー" | nkf -Z4).txt
$ ls *.txt
ninjya.txt      ﾆﾝｼﾞｬｽﾚｲﾔｰ.txt
```

▼図12　ファイル名の半角カナを全角カナにする

```
$ ls ﾆﾝｼﾞｬｽﾚｲﾔｰ.txt | nkf -W
ニンジャスレイヤー.txt
```

短期集中連載

アプリケーション開発者がインフラ担当に!

Rettyのサービス拡大を支えた“たたき上げ”DevOps

第1回 ＞ Vagrantを使って既存サーバの見通しを改善する！

実名ユーザたちによるお勧めからレストランを探せるグルメ系Webサービス「Retty」。急成長するサービスの裏側では、融通のきかない古いシステムから大規模システムへの移行という難題が立ちはだかっていました。それを乗り越えたのはインフラ経験なしのアプリケーションエンジニア。スマートなだけではすまされない、現場でのInfrastructure as Code実践を紹介してもらいます。

Writer 梅田 昌太(うめだ しょうた)　Retty㈱　チャーハン担当　Twitter @ebisusurf

まあチャーハンでも食べながら

初めまして。Retty株式会社でチャーハン[注1]兼インフラを担当している梅田です。私どもは「食を通じて世界中の人々をHappyに」の理念のもと、日本最大級の実名性グルメサイト「Retty[注2](レッティ)」を運営しています。Rettyは2011年にサービスが開始して3年が経ち、おかげさまで現在ではユーザ数200万人、投稿数80万まで成長してきました。

この短期連載ではRettyの開発を通して、スタートアップサービスが急速な成長過程に入ったときにどのようなことが起こり、それらにどう対処すべきか、その一例をインフラ担当の筆者の経験から紹介したいと思います。

Rettyというサービスの開発的歴史

Rettyはプログラミング未経験者も含めた(スゴイ！)、ほんの数名での開発から始まったWebサービスです。このようなスタートアップサービスが次第に多くのユーザ様に使ってもらうこととなり、会社の成長を経て中規模、そして大規模な開発への移行が必要となってきています(2014年には1000万人以上のユーザ獲得を目指してます)。現場ではその時々で、どのようにWebアプリケーションとインフラ、そして開発プロセスをデザインしていくのかを考えてきました。

詳しい話に入る前に、まずは筆者の経歴、普段の業務、そして業務へのスタンスを知っておいてもらったほうが、本稿の視点が明確になると思います。

アプリケーション開発出身のインフラ担当

前職でソーシャルゲームの開発支援(おもにミドルウェア周りの選定や保守、チューニング)、前々職ではSNSプラットフォームで自社サービスの開発を行っていました。なのでインフラ担当となったのは今回が初めてです(しかもインフラ担当者は筆者1人)。

“アプリケーション開発者がインフラ担当になる”というキャリアパスは想像していなかったことなので、我ながらとても面白く、日々勉強、勉強です。

Rettyインフラ担当としてAWSとのかかわり方

RettyではインフラにAWS(Amazon Web

注1) Rettyでは社員それぞれに担当メニューが付きます(料理に限らずバーやスイーツといったカテゴリもあります)。チャーハンを選んだ理由はもちろん筆者が好きだからですが、
・多くの人に愛されるメニュー
・ラーメンのように論争になりにくい平和なメニュー
という点が挙げられます。

注2) http://retty.me

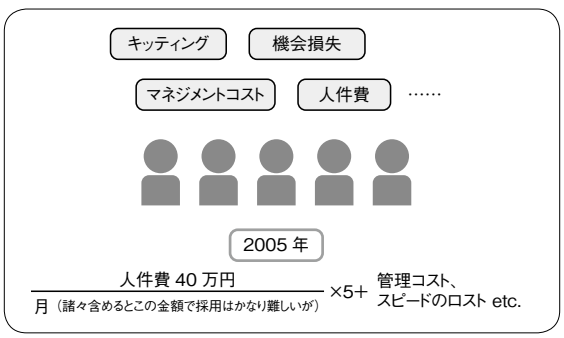
Services)を利用しています。現在筆者の仕事もほとんどがAWSとのお付き合いになります。アプリケーション開発者出身なので、ハードウェアのキッティング[注3]やケーブリング、データセンター(以下、DC)選定のスキルなどはほとんどありません。それでもなぜ1人でインフラ担当ができるのか。

それはひとえにAWSが提供しているサービスの豊富さ、API、CLIツールがそろっていることで、プログラマブルにインフラを扱うことができるからです。若干言い尽くされた感もありますが、クラウドサービスを使ったインフラデザインのすごいところはコストの低さや導入のスピードではなく(もちろんそれもありますが)、インフラデザインを含めたオペレーションそのものが変わったことだと思います。

これまでのフローでサーバを増やそうと思ったら、「サーバの購入、DCでのキッティング、環境の構築」などなど想像しやすいところから、実際は「稟議書いてFAXで注文して」とか「電源は足りるのか?」とか「ラックに入らなくなってきたがDC内にもう空きがない」などといった実務的なことや周辺環境に至るまで、本当に多くのことを考慮する必要がありました。また、技術レイヤーの壁も大きな問題です。アプリケーション開発者よりかなり下のレイヤー(物理層、ネットワーク層)がカバーできているエンジニアにしか手が出せなかったことが多かったのです。

注3) ここではおおざっぱに機器の選定や組み立て、配置などの意味です。

▼図1 2005年くらいのオンプレミスインフラの管理コストイメージ



こういうワークフローそのものを、
「プログラマブルに」
「すべてオフィスや自宅からAPI経由で」
「必要なときに必要な分だけ素早く」
「インフラの状態をコード化してGitHubのようなVCS(Version Control System)で管理」
できるようになったことが非常に大きな転換だったのではないでしょうか。最終的にはこれが開発コストの削減や開発速度の向上といったことにつながると考えています。

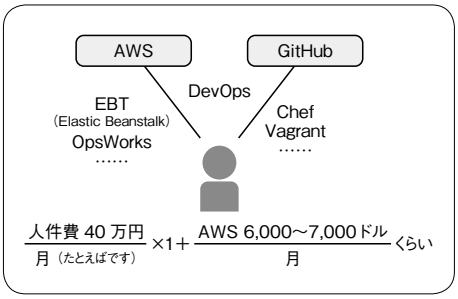
AWSは高い?

実際問題、AWSの請求はそれなりによい金額です。しかしながら「これまで5人いないとできなかったことが1人でできるようになった」となれば十分コストメリットがあります。リザーブドインスタンスの購入などで揺れ幅はありますが、Rettyではサーバコストがおおむね6,000~7,000ドル/月前後で推移しています(ユーザ数が増えているので緩やかに上がってきてはいますが)。これを高いと取るかどうかが非常に重要です。

筆者が考えるに、旧来のオンプレミス構成で現状のRettyのスピード感に対応しようとすると3~5人は必要になると思っています。いや、経験的にスピード感で言うと、人数を増やしてもあまり上がらないことがほとんどで、意思疎通コストや管理コストばかりが上がる印象です。

仮に5人と仮定して2005年くらいのオンプレミスインフラをイメージすると、**図1**のような

▼図2 現在の筆者のインフラ管理コストイメージ



感じでしょうか。これを筆者1人で行うと**図2**のようになります。

急遽スケールさせたくなったときはAMI（Amazon Machine Image）から即座にインスタンスを立ち上げてELB（Elastic Load Balancer）の下に入れることが、AWSのAPIを使えば簡単に行えます（ELBそのもののスケールはちょっとテクニックが必要です）。Multi-AZ[注4]で配備することで可用性も容易に確保できます。スケールアップもインスタンスタイプの変更を行うだけですぐにできます。ロール分けやコンテキストの維持は、GitHubのコードに落としておきます（Infrastructure as Codeですね！）。

なので「開発の問題は増員でなく、お金で解決してしまえ」と言えるかどうかはかなり大事です。従来のように、ベンダーやDCから見積書をもらってから「もう少し安くならないか」「サービスに対して適切か」といった検討をする、という行為は、スタートアップ企業にはやりにくい状況です。体感値として適切な見積もりはほぼ無理ですし、そもそもそんなことを考えてる時間がもったいない。クラウドのお金より人件費のほうが高くついてしまいます。それよりも、使ってから「実際必要になった金額」に対して予算が適切かを検討できるクラウドを使ったインフラデザインのほうが、Rettyのようなサービスにはマッチしてると考えています（逆に、「適切に見積もってキッチリ予定どおりの金額で収める」という利用にはクラウドはあまり向いていないとも言えます）。

ということで、小規模のスタートアップではエンジニアが使える予算感はかなり大事ですし、必然的に1人当たりの予算割当は大きくなってきます。インフラ担当者としては、サービスをさばくのがつらくなってきたら、インフラエンジニアを1人増やすよりも、「とりあえずAWSのサービスを使って賄ってしまおう（お金で解決）。その後チューニングで適切にスケールさせよう」という流れをよしとするコンセンサスを会社側としっかり取っておくべきだと思います。これははっきり言って技術の話ではなく会社との信頼関係の話ですが、現実問題として避けては通れない部分です。

注4）EC2のインスタンスを物理的に分けられた環境に配備できる機能。ロードバランサ配下に均等に置くことで、耐障害性を高めることができます。RDSの場合、異なるAvailability Zoneにスタンバイを配備してくれるだけでなく、障害時には自動フェイルオーバーも行ってくれます。

Rettyインフラ担当としてのゴール

最終的には、DevOpsと呼ばれるインフラ担当をやりつつサービスの開発にどんどんコミットしていくことを目指しています。筆者自身サービス開発は大好きなので、設定ばかりやらずに少しでもはみ出していきたいと思っています。

サービスに途中参加したエンジニアとして大事にすべきこと

先程も書きましたが、Rettyは非エンジニアが作り出したサービスで、徐々にエンジニアがアサインしてきたチームです。こういった特性上、開発やインフラに対するアンチパターンやら技術的負債やら、そんな格好いい呼び方もできないような、はっきり言ってしまえば「汚れた構成」がいろいろと見受けられます。

たとえば、

- dynamicな領域であるアプリケーションのソースコード中に、staticな領域の構成情報（例：ドメイン名、サーバ環境情報）がハードコードされていて、GitHub上のソースコードと実稼働中のアプリケーションサーバ内のソースコードとに乖離がある。単純に言うとGitHubで管理されているソースコードをそのままデプロイすると動かない
- サブシステムが結合していてオペレーションや構成の変更に対応できない
- メールサーバ、バッチサーバ、デプロイサーバ、監視サーバ、集計サーバが複数のサーバ上（アプリケーションサーバ含む）で同居しているため

「デプロイフローを変更したい」「メールサービスをアウトソースしたい」「パッケージをアップデートしたい」といった業務効率化のための作業が行いにくい

といった問題があります。ですが……。

今あるしくみは誰かが試行錯誤した結果

今あるしくみというのは、サービスを成長させるために誰かが試行錯誤しながら何とか作り上げてきたものです。まがりなりにも動いているものを安易に否定することは絶対に良くないと思っています。筆者自身がこれまで、「あるべき論」を大きな声で叫び過ぎたことで周囲を不快にさせてしまったこともあり、非常に反省してます。

とくにインフラレイヤーは構成の意図を読み取ることが難しいので(多くの場合意図はなくて「何となく」だったりしますがw)、まずは「できることから1つずつ」を心がけてます。

Rettyのリファクタリング

ここからやっと具体的な話になってきます。最も重要なことは「変化に対応できる」ようにすることです。Yahoo砲やWBS砲[注5]でアクセススパイクが起こるといった、指数関数的にユーザ数が伸びるチャンスはいつ来るかわかりません。チャンスを逃さないためには素早く変更できるようにする必要があります。

今回はRettyのアプリケーションサーバのリファクタリングを中心に書きたいと思います。次回以降はサブドメインサービス、サブシステム、アプリケーションのリファクタリングなどについて触れたいと思っています。

注5) 言わずもがなかもしれませんが、Yahoo! Japanで人気トピックになったり、テレビ東京の人気番組「World Business Satellite」で取り上げられたりすることですね。

目的:64bit Amazon Linuxへの移行

入社直後、最初に取りかかったのが64bit Amazon Linuxへの移行です。RettyではEC2にAmazon Linuxを利用してます。が、これが32bitのAMIで起動されていたため64bit Amazon Linuxへの移行を行いました。

なぜ移行の必要があったのか?

- サードパーティのサービスが使いにくい(New Relicなど)
- 当時新発売のc3.large[注6]が使いたかった(お買い得&32bit非対応)

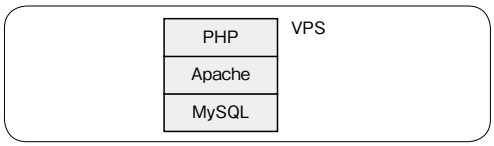
とくに後者の理由が大きいです。世の中が64bitに移行してるのは間違いないので、早々に対応しておかないと使いたいサービスが使えなかったりして開発の足を引っ張ります(ちなみに執筆時点ではc3.largeを利用していますが、リファクタリング直後は特定のAvailability Zoneが売り切れで、利用お預けを食らった期間があるくらい当時人気のインスタンスでした)。

アプリケーションサーバをリファクタリング

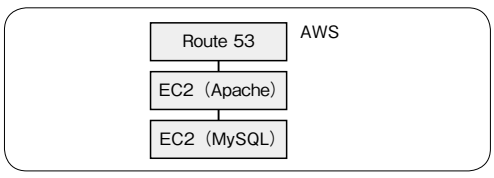
イメージを入れ替えるということは、「素の64bit Amazon Linux」に今本番環境で動いてい

注6) http://aws.typepad.com/aws_japan/2013/11/a-generation-of-ec2-instances-for-compute-intensive-workloads.html

▼図3 Rettyのアプリケーションサーバ構成(最初期)



▼図4 Rettyのアプリケーションサーバ構成(AWS移行期)



るサーバと同じ環境をセットアップする必要があります。わざわざ構成を掘り起こしてセットアップしなおすのであれば、多少手間をかけてでも構成を粗結合化することにしました。

ここでRettyのアプリケーションサーバの歴史について紹介したいと思います。

最初期はVPS（仮想専用サーバ）で稼働してました（**図3**）。スタートアップだベンチャーだといってもそんなもんだと思います。これを**図4**のように、前任者がAWSへ移行してくれました。

そして**図5**が、現状の大まかな構成になります。「AWSで提供されているものをできるだけ使う」というのがコンセプトです。Route 53（DNS）→ CloudFront（CDN）→ Elastic Load Balancer（ロードバランサ）→EC2（アプリケーションサーバ）→RDS（MySQL）という役割を担っています。そのほかS3、ElastiCache、SQS、Elastic Beanstalkといったサブシステムや、EC2上にログ集積用のためのMongoDBを立てていたりもしますが、それらについては次回以降の記事で解説する予定です。

主観ではありますが、このへんのハンドリングを主体的に行えるのはスタートアップならではの面白いところだと思います。インフラとオペレーションの最適化をセットで考えられます。

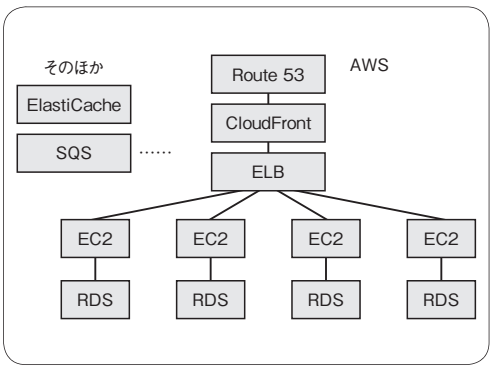
さて、前述の「汚れた構成」から脱却するため、アプリケーションサーバのリファクタリング時に、バッチと監視の機能を引きはがして別サーバとしました。**図4**のEC2（アプリケーションサーバ部分）での構成を抜き出したものが**図6**です。この状態から、PHPとApacheのみが動いているシンプルなアプリケーションサーバにします（Nagios + NRPE[注7]での監視は行います）。おおまかに図解すると**図7**のようになります。

変更前のアプリケーションサーバ（**図6**）ではPHPとApache以外に、各サーバにMonit[注8]（プロセス監視、ディスクなどの一部リソース監視）、Nagios（一部エラーログ監視）、cron（バッチ処理）が入っていて依存関係の高い構成になってました。

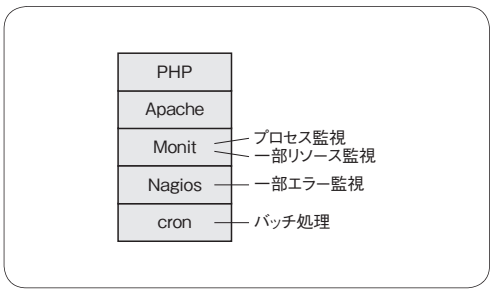
変更後のアプリケーションサーバ（**図7**）には、PHP & ApacheとNagiosのクライアントのみとし、次のような役割を切り離しました。

- ディスクなどサーバーリソースの監視はCloudWatch（AWSのサービス）
- モニタリングにNew Relic[注9]（一部のサーバ、

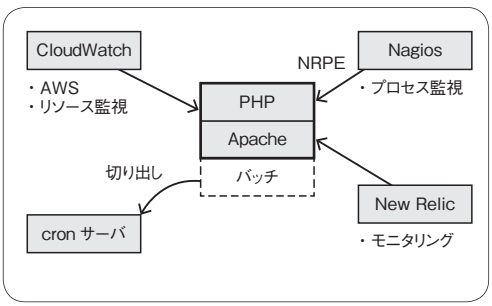
▼図5　Rettyのアプリケーションサーバ構成（現状）



▼図6　変更前のアプリケーションサーバ（図4の詳細）



▼図7　変更後のアプリケーションサーバ



注7）Nagios Remote Plugin Executor：http://exchange.nagios.org/directory/Addons/Monitoring-Agents/NRPE--2D-Nagios-Remote-Plugin-Executor/details
注8）http://mmonit.com/monit/
注9）http://newrelic.com/

サードパーティサービス）
・プロセスの監視はNagios（NRPEでのネットワーク監視）
・cron処理は別サーバに切り出し

しかしこの分離作業がたいへんでした。なにしろアプリケーションサーバの構成がわかるのは「今動いているものだけ」なので、トライ&エラーでインストールされてるパッケージを探したり、コンフィグレーションを読みとる必要がありました。この作業負担をなるべく下げるため、次に解説するようにVagrantを使って試行錯誤を繰り返しました。

Vagrantを使ったサーバのリバースエンジニアリング

Vagrant[注10]は今さらお伝えすることもないくらい人気の、VM（Virtual Machine）の起動周りを管理してくれる便利ツールです。Vagrantfileというファイルに Rubyで VMの起動時の挙動を記述しておいて、**vagrant up**とコマンド入力することで基本的にはどの環境でも同じ環境を構築できます。こうしてローカル上で気軽にテスト環境を作り、その環境が壊れてしまったら、**vagrant destroy**で削除して作り直せるという、トライ&エラー作業にもってこいのツールです[注11]。

Vagrantはサーバの構成を読み解くのに使うためのツールではないのですが、現状のサーバが「そもそもどういう構成になってるのかがわからない」ので、インストールしたり、設定を変えたりをひたすら繰り返します。その「途中の状態」や「経過」を記録するためにVagrantを通して行いました。Vagrantを使わなくてもできるのですが、“ラクに”行うために使っていました。

Vagrantはプラグインも充実していて「vagrant-aws」というプラグインを使うことでEC2へのインターフェースにもなります。ローカルVirtualBox上での環境構築と、EC2上での環境構築を同じインターフェースで行えるのでとても便利です。**図8**のようにVagrantを通して、VirtualBox上でひたすら作っては壊してを繰り返して、現状稼働しているRettyのアプリケーションサーバ状態に近づけます。

ここで重要なのは、**あれこれイジったVMのイメージそのものを管理しない**ということです（即時起動させることが重要なもの、たとえば本番アプリケーションサーバのAMIなどは別）。サーバのセットアップをしていると、変更を行った「前と後」について比較したくなることが必ずあります。パッケージを入れる前と後、バージョンを上げる前と後などなどです。しかし、これを行うために“VMのイメージに日付をつけて管理するようなことはしない”ということです。VMのイメージを管理するのではなく、プロビジョニングファイルをGitでバージョン管理することで「前」の状態と「後」の状態を保持するようにします。

そしてある程度環境が近づいたところで、EC2上に環境を用意して再度テストします。このときに前述したvagrant-awsが登場します。プラグインインストールは次のように実行します。

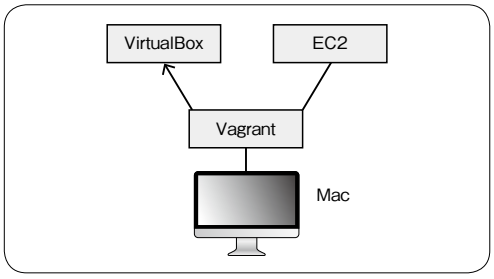
```
# vagrant plugin install vagrant-aws
```

インストールが完了したら、Vagrantfileを用意します。**リスト1**にサンプルを記します。

Vagrantfileの準備ができたら、次のコマンドを実行します。

注10）http://www.vagrantup.com/
注11）Vagrantの基本的な使い方は、『Vagrant入門ガイド』（新原雅司 著、技術評論社 刊、Kindle版）がコンパクトにまとまっていて良いと思います。

▼図8　Vagrantを使ったサーバ構成の解析環境



```
% vagrant up --provider=aws
```

これでVirtualBox上で作った環境と同じ環境のEC2が起動します。**リスト1**のVagrantfileにはポイントが3つあります。

①環境変数を用意すること

VagrantfileはGitHubのようなVCS上で管理することを前提にしているので、VagrantfileにAWSのシークレットキーやシークレットアクセスキーを書いておくのはよろしくありません。環境変数を用意しましょう。aws-cli（AWSコマンドラインインターフェース）を使う際にも必要になるので、用意しておいたほうが良いです。

https://console.aws.amazon.com/iam/home?#security_credential

からキーを取得します。筆者はzshを使ってるので、.zshrcに、

```
export AWS_ACCESS_KEY_ID=YourAccessKey
export AWS_SECRET_ACCESS_KEY=YourSecretAccessKey
```

こんな感じで設定しておいて、Vagrantfileからは次のように環境変数で読み込みましょう（**リスト1**では❶の個所）。

▼リスト1　サンプルVagrantfile

```
# -*- mode: ruby -*-
# vi: set ft=ruby :
VAGRANTFILE_API_VERSION = "2"

AWS_DUMMY_BOX = "dummy"
AWS_DUMMY_BOX_URL = "https://github.com/mitchellh/vagrant-aws/raw/master/dummy.box"

AWS_KEY_PAIR_NAME = "YoureKey"
OVER_RIDE_SSH_USER_NAME = "ec2-user"
OVER_RIDE_SSH_PRIVATE_KEY = "~/.ssh/YoureKey"

Vagrant.configure(VAGRANTFILE_API_VERSION) do |config|
    config.vm.define :webserver do |webserver|
        # ec2用のダミーbox
        webserver.vm.box = AWS_DUMMY_BOX
        webserver.vm.box_url = AWS_DUMMY_BOX_URL
        # provisioning
        config.vm.provision :shell, :path => "retty_confs/provision.sh"  ―❷
        webserver.vm.provider :aws do |aws, override|
            # (VM 固有設定)
            aws.tags = {
              'Name' => 'Gihyo',
              'Description' => 'GihyoTest'
            }
            aws.instance_type = "t1.micro"
            aws.ami = "ami-0d13700c"
            aws.security_groups = ["YoureSecurityGroup"]

            # (AWS共通設定)
            aws.access_key_id = ENV['AWS_ACCESS_KEY_ID']
            aws.secret_access_key = ENV['AWS_SECRET_ACCESS_KEY']  ―❶
            aws.region = "ap-northeast-1" # "Tokyo"
            aws.availability_zone = "ap-northeast-1a"
            aws.keypair_name = AWS_KEY_PAIR_NAME
            override.ssh.username = OVER_RIDE_SSH_USER_NAME
            override.ssh.private_key_path = OVER_RIDE_SSH_PRIVATE_KEY
        end
    end
end
```

```
ENV['AWS_ACCESS_KEY_ID']
ENV['AWS_SECRET_ACCESS_KEY']
```

②プロビジョニングはshell

Vagrantの話になると必ずといってよいほど話題に上がるのが「プロビジョニングを何のツールで行うのか？」です。これはVagrantの登場とプロビジョニングフレームワークの盛り上がりが時期として近かったのが背景にあるのだと思いますが、入社当初はshellを選択しました。個人的にChef[注12]やAnsible[注13]を利用しているのですが、RettyではPHPを中心に開発していることと、プロビジョニングフレームワークの導入が行われていなかったためです。

shellでもコンテキストの維持はできます。Vagrantの主な役割はsandbox環境の構築なので幂統制(べきとうせい)はあえて無視しました(現在はロールが増えてきたので、プロビジョニングフレームワークの導入を進めています)。VagrantはVMの起動管理とプロビジョニングをセットにして動かせるところがとても魅力的なので、プロビジョニングの設定を使わない手はないと思います。

③confファイルはcopy

プロビジョニングフレームワークを使わないので、confファイルの動的な生成はしません。用意しておいたconfファイルをシェルスクリプトで上書きするようにしました(**リスト1**の❷の個所)。

Vagrantはホストとゲストとでフォルダを共有してくれるしくみがあるので(デフォルトはホストVagrantfile配下とゲストの/vagrant以下が共有)、シェルスクリプトprovision.sh内部で、

```
# httpd.confを元々用意していたファイルでcopy
cp /vagrant/retty_confs/httpd.conf /etc/ア
httpd/conf/httpd.conf
```

といった形で用意したファイルで上書きします。**図9**のように、confファイルをVagrantfileと同居させておくと楽です。

最近はDocker[注14]のようなLXCベースのコンテナがとても流行っているみたいなので、いずれチャレンジしたいなと思っています。

第1回のまとめ

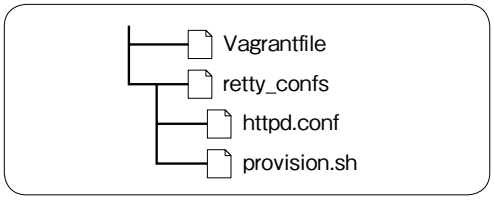
「ローカルのVMでプロビジョニング＆テスト」→「EC2で同じプロビジョニング＆テスト」→「不備があったらまたローカルでプロビジョニングしてテスト」→「EC2で同じプロビジョニングしてテスト」……。こういった作業もVagrantのおかげでサーバの起動管理が非常に楽です。

上記の作業をひたすら繰り返して、ようやくアプリケーションサーバの状態を掘り起こすことができました。前述したとおり、掘り起こしたサーバの状態はVagrantfileとshellで管理します。くれぐれも日付の入ったVMのイメージファイルを管理するようなフローにはしないように(現実的には最新版のVMイメージくらいは存在しても良いと思います、何かしらの用途で「すぐに今の状態を使いたい」ってことはあると思います)。

そして2013年の12月に、無事64bit Amazon Linuxへの入れ替えが完了しました。実際のところはEBSの付け替え作業や、CloudWatchでのモニタリング＆アラートの設定といった作業も必要になるのですが、このあたりは次回以降の監視周りで触れたいと思います。SD

注12) http://www.getchef.com/
注13) http://www.ansible.com/

▼図9　Vagrantfileとconfファイルの置き方例



注14) https://www.docker.io/

短期集中連載 最終回

Mac as a desktop Service -MaaS- !?

#3 さらに踏み込む、Mac OS Xと仮想デスクトップ

複数のOS環境を必要とするMac使いのエンジニアにとって、仮想デスクトップ環境をMacに構築することはもはやあたりまえのことのようです。筆者もその一人ですが、日常的に使っているとOS間を行き来するオペレーションに煩わしさを感じるようになりました。この短期連載では、筆者がこの煩わしさから解放されるために行った、普通とはちょっと違ったアプローチをご提案します。

後藤 大地（ごとう だいち）（有）オングス 代表取締役

作ってみよう仮想環境＋Mac OS Xデスクトップハイブリッド環境

本連載第1回、第2回では、NFS、ssh(1)、X Window Systemの基本的な機能の紹介と簡単な使い方を紹介してきました。短期集中連載最終回となる今回は、これら機能を組み合わせてハイブリッドなデスクトップ環境を作成する方法を紹介します。組み方は用途に応じて人それぞれだと思いますので、随所々々の設定を抜き出して活用いただければと思います。

[モデルA] モバイル（デスクトップMac、仮想環境UNIX系OS）

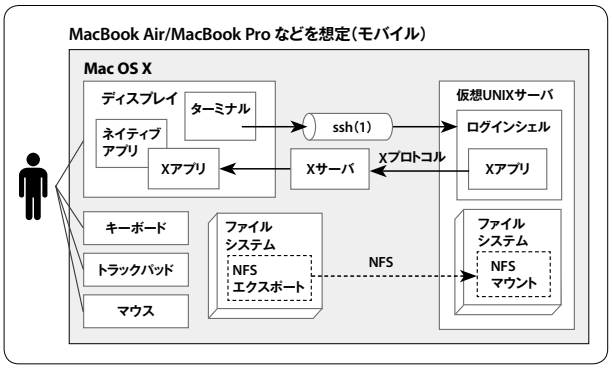
MacBook AirやMacBook Proなど、おもにモバイルで使っているマシンの場合を考えます。この場合、UNIX系オペレーティングシステム（以下、OS）は仮想環境上で動作させ、ホストとゲストの間をNFS、ssh(1)、Xで接続します。ゲスト側のIPを固定しておきたいので、ここでは仮想化ソフトウェアが提供しているNATを経由して接続するものとします（**図1**）。

説明に使用するモバイル環境は次のとおりです。

ホストマシン：MacBook Pro Retina 13-inch Late 2013
ホストOS：Mac OS X Mavericks
仮想化ソフトウェア：VMware Fusion 6
ゲストOS：Ubuntu 12.04 LTS（64bit）
ゲストOS：FreeBSD 10.0-RELEASE amd64（64bit）
ゲスト〜ホスト間ネットワーク：Macを共有（NAT）

CUIの作業はMac OS Xのターミナルから、ゲストで動作しているUNIX系OSにログインして行います。ホストとゲストの間は可能な限りさまざまな部分を共有することで、別のOSで動作していることを意識できないようにします。

▼図1　MacBook AirやMacBook Proのようなモバイル環境でのモデル



▼図2　Mac OS Xでのホームディレクトリ、ユーザ名、ユーザID、グループ名、グループIDを確認

```
% echo $HOME
/Users/daichi
% id
uid=1002(daichi) gid=20(staff) groups=20(staff),1011(daichi),1002(smbusers),12(everyone),61(local
accounts),79(_appserverusr),80(admin),81(_appserveradm),98(_lpadmin),399(com.apple.access_ssh),
33(_appstore),100(_lpoperator),204(_developer),398(com.apple.access_screensharing)
%
```

▼表1　作成するアカウント情報

項目	値
ホームディレクトリ	/Users/daichi
ユーザ名	daichi
ユーザID	1002
グループ名	staff
グループID	20

システムの組み方：ユーザアカウント

NFSでファイルシステムを共有したいので、**ユーザのユーザIDとグループIDをホストとゲストでそろえます**。設定のしやすさを考えると、Mac OS Xで使っているユーザIDとグループIDの設定を、UNIX系OS（例ではUbuntu）に反映させるほうがよいでしょう。ホームディレクトリのパスも同じにしておきます。ホームディレクトリやユーザIDなどは**図2**のように確認します。作成するアカウント情報を整理すると**表1**になります。

LinuxやFreeBSDでは管理者権限で**vipw**(8)コマンドを実行するなどして、ユーザID情報を書き換えます。グループIDは/etc/groupファイルを編集することで実施します。**表1**をもとに、ここでは**リスト1**のようにユーザID、グループID、ホームディレクトリなどを設定します。

グループIDとグループ名のマッチングは/etc/groupファイルに記載します。ここは**リスト2**のように書いておきます。

▼リスト1　vipw（8）で編集したユーザ情報の例

```
daichi:暗号化されたパスワード:1002:20::0:0:
Daichi GOTO:/Users/daichi:/usr/local/bin/zsh
```

▼リスト2　/etc/groupファイルの記述例

```
staff:*:20:
```

システムの組み方：ネットワーク

ホストとゲストの間のネットワークを設定します。ここではVMware Fusionがデフォルトで割り振るIPアドレスを使っています。使用しているソフトウェアに合わせて適宜読み替えてください。IPアドレスはMac OS Xで**ifconfig**(8)コマンドを実行することで確認できます（**図3**）。VMware Fusion 6であればvmnet8ネットワークインターフェースに設定されているアドレスが該当します。

ゲストで指定するデフォルトゲートウェイのIPアドレスは、いったんDHCPで接続した状態で**図4**のように**netstat**(1)コマンドを実行することで確認できます。表示されたGatewayの部分に記載されています。

このIPアドレスは常に固定です。MacBookが接続するWi-Fiルータを変更しても、このローカルアドレスは固定したまま変わりません。基

▼図3　仮想化ソフトウェアが使っているNAT向けのIPアドレスをifconfig（8）コマンドで確認

```
% ifconfig
 …省略…
vmnet1: flags=8863<UP,BROADCAST,SMART,RUNNING,SIMPLEX,MULTICAST> mtu 1500
        ether 00:50:56:c0:00:01
        inet 192.168.218.1 netmask 0xffffff00 broadcast 192.168.218.255
vmnet8: flags=8863<UP,BROADCAST,SMART,RUNNING,SIMPLEX,MULTICAST> mtu 1500
        ether 00:50:56:c0:00:08
        inet 192.168.185.1 netmask 0xffffff00 broadcast 192.168.185.255
 …省略…
%
```

▼図4 netstat(1)コマンドでデフォルトゲートウェイを確認(Ubuntuの場合)

```
$ netstat -nr
Kernel IP routing table
Destination     Gateway         Genmask         Flags   MSS Window  irtt Iface
0.0.0.0         192.168.185.2   0.0.0.0         UG        0 0          0 eth0
169.254.0.0     0.0.0.0         255.255.0.0     U         0 0          0 eth0
192.168.185.0   0.0.0.0         255.255.255.0   U         0 0          0 eth0
$
```

▼表2 ネットワーク設定

項目	内容
ホスト側のホスト名	host
ゲスト側のゲスト名	virt
ホスト側のIPアドレス	192.168.185.1
ゲスト側のIPアドレス	192.168.185.50
ゲスト側で指定するデフォルトゲートウェイのIPアドレス	192.168.185.2

▼図5 Mac OS Xのホスト名は[設定]→[共有]→[コンピュータ名]で設定

本的に外部のホストからアクセスすることはできません。ゲストでDHCPを使っても構いませんが、スタティックな設定として設定ファイルに書いておきましょう(**表2**)。

Mac OS Xのホスト名は[設定]→[共有]→[コンピュータ名]で設定できます(**図5**)。名前はなるべく統一しておいたほうが問題が少なくて済みますので、ホスト名も~/.ssh/configの設定もOSごとの設定もそろえておきます。

システムの組み方：ホームディレクトリ共有

モデルAはMac OS Xがベースになっていますので、**Mac OS Xのユーザのホームディレクトリをまるごと仮想環境のユーザのホームディレクトリになるようにマウント**して使います。

まず、Mac OS X側でユーザのホームディレクトリをNFSでマウントできるように、**リスト3**のような内容の/etc/exportsファイルをMac OS X(ホスト)側に作成します。編集作業は**sudo**(8)コマンドを経由して実施してください。マウント提供する対象をもっと絞り込みたいなら、**リスト4**のように書いておいてもかまいません。**リスト3**では192.168.185.0/24のネットワークに所属するホストに対してマウントが許可されますが、**リスト4**では192.168.185.50のホストのみがマウントできます。

/etc/exportsファイルを作成すると**nfsd**(8)など必要なデーモンが自動的に起動して処理が開始されます。/etc/exportsファイルの記述を間違えると正しく処理できません。ファイルが適切に記述されているかどうかはnfsd(8)コマンドで確認できます。**図6**のように何も報告されずにコマンドが終了すれば適切に記述されています。なお、Mac OS X Mavericksのnfsd(8)コマンドはサブコマンドとしてcheckexports以外にもenable、disable、start、stop、restart、update、status、verboseを指定できます(**図7**)。

次に仮想環境(ゲスト)側では、/etc/fstabに**リスト5**のようなエントリを追加して、仮想環境のユーザのホームディレクトリとしてMac OS Xのユーザのホームディレクトリをそのまま使うようにします。NFSマウントエントリの書き方はOSごとに異なりますが、基本部分は同じです。指定できるオプションなどに違いがあります。まずは**リスト5**のようになにもオプションを指定しないで使ってみましょう。

システムの組み方：ssh(1)による双方向ログイン

ホストとゲストの間で**ssh**(1)による双方向

▼リスト3　Mac OS Xの/etc/exportsファイルの例 その1

```
/Users/daichi -mapall=1002 -network 192.168.185.0 -mask 255.255.255.0
```

▼リスト4　Mac OS Xの/etc/exportsファイルの例 その2

```
/Users/daichi -mapall=1002 -network 192.168.185.50
```

▼リスト5　Ubuntu 12.04とFreeBSD 10.0の/etc/fstabに追加するエントリ

```
192.168.185.1:/Users/daichi /Users/daichi nfs rw 0  0
```

▼図6　/etc/exportsファイルの記述が適切であるかチェック(適切であればなにも報告されない)

```
/Users/daichi% nfsd checkexports
/Users/daichi%
```

▼図7　nfsd(8)の動作状況をチェック

```
/Users/daichi% nfsd status
nfsd service is enabled
nfsd is running (pid 75, 8 threads)
/Users/daichi%
```

のやり取りができるようにします。**公開鍵認証によるログインのみを許可し、相互にログインできる**ように~/.ssh/configファイルを編集します(**リスト6**)。NFSでホームディレクトリを共有していますので、~/.ssh/configの設定もホストとゲストで共有されます。

連載第2回目で紹介したように、この環境ではssh(1)のX11フォーワーディングの機能のほうが負担が少なかったので、この機能を有効にするために「ForwardX11」と「ForwardX11 Trusted」を有効にしています。NFSでホームディレクトリごとマウントしているので、秘密鍵はホストとゲストで共有することになります。Mac OS Xの認証エージェントによるパスフレーズの入力機能を活用したいので、「Forward Agent」も有効にしておきます。

ここではゲストへのログインは必ず"ホストからゲスト"にログインすることで開始されるようにします(仮想化ソフトウェアの提供する画面からログインして作業するといった使い方はしません)。このため、ホストからゲストにログインするときは「ForwardAgent」のみを有効にしています。これで相互にシームレスにログインできるようになります。仮想化ソフトウェアが提供する画面からもログインして、そこからホストにログインするような場合にはさらに適宜設定を追加します。

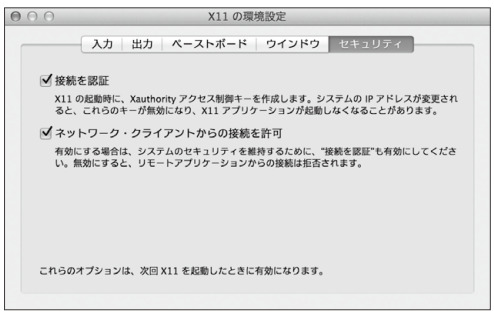
▼リスト6　~/.ssh/configに追加する設定

```
# 仮想環境ではNIC仮想デバイスやNAT変換にかかる負担が
# 大きく、SSHのX11フォワーディング機能を使ったほうが
# 負担が軽い
Host virt
        Hostname                192.168.185.50
        IdentityFile            ~/.ssh/id_rsa
        ForwardAgent            yes
        ForwardX11              yes
        ForwardX11Trusted       yes

Host host
        Hostname                192.168.185.1
        ForwardAgent            yes
```

システムの組み方：X Window System

連載第2回目を参考にXQuartzをインストールしておいてください。デフォルトインストールの状態だとネットワーククライアントからの接続が許可されないので、XQuartzの設定ダイアログから[セキュリティ]→[ネットワーク・クライアントからの接続を許可]にチェックを入れて有効化しておきます(**図8**)。

▼図8　XQuartzインストール後「ネットワーク・クライアントからの接続を許可」にチェックを入れて再ログイン



さらに踏み込む、Mac OS Xと仮想デスクトップ

システムの組み方：スタートアップスクリプト

モデルAの場合、ssh(1)のX11フォワーディング機能を使うので、使うにあたっての下準備としては**ゲストにログインした段階でホスト側でXQuartzが動作しておいてほしい**、といったところになります。この場合、もっとも実行が優先されるパスとして~/bin/などを追加して、そこに**リスト7**のようなssh(1)のラッパスクリプトを作成して置いておきます。

このスクリプトでホストからゲストに対して`ssh virt`でログインしようとした場合にXQuartzが起動していなければ、起動するようになります。`open -a XQuartz`で起動されるプログラムの名前は「Xquartz」になっていますので、このあたりの大文字・小文字には注意してください。

システムの組み方：融合させるためのシェルスクリプト

ここでゲスト側にホスト側で使用するコマンドと同じ名前のコマンドを用意するといったように、細かに融合を進めるためのスクリプトを作成していきます。まず、**ゲストで実行したコマンドが実はホストで実行される**というしくみを実現するために**リスト8**のように「mac」というスクリプトファイルを作成します。たとえばゲストで`mac ps`のように実行すると、ホストで`ps`(1)を実行した結果が表示されます。

次に、Mac OS Xの`open`(1)コマンドに相当するスクリプトを作成します。これでゲストで`open .`のように実行すると、ホストでFinderが起動するようになります。本質的には`mac open " ${@}"`のように実行するだけでよいのですが、オプションや引数のパスなども考慮して処理しておくと**リスト9**のようなスクリプトになります。

システムクリップボードにテキストをコピーしたり、そこからテキストを取り出すラッパースクリプトを「copy」および「paste」として用意します。Xサーバを経由してホストとゲスト間でのコピー&ペーストが可能です。copyとpasteコマンドを使うことでこの処理をコマンドベースで実施できます。実際にシステムクリップボードを操作するコマンドとしてはMac OS Xでは`pbcopy`(1)および`pbpaste`(1)を使います。UNIX系サーバでは`xsel`(1)などを使います(適宜インストールしてください)。**リスト10、11**のようなスクリプトを作成します。

普段頻用するコマンドは人それぞれだと思いますので、macスクリプトを使って自分がよく使うMac OS Xでのコマンドをゲスト側からシームレスに使えるように、シェルスクリプトを作成していきます。このあたりはユーザが好みでいろいろ作成する部分です。

以上でモデルAの構築は完了です。

▼リスト7　~/bin/sshに設置するsshのラッパスクリプト

```
#!/bin/sh

case $(hostname -s):"$1" in
host:virt)
    pgrep Xquartz > /dev/null || open -a XQuartz
;;
esac
exec /usr/bin/ssh "$@"
```

▼リスト8　macスクリプト：ホスト側でコマンドを実行するスクリプト

```
#!/bin/sh

[ "0" = $# ] && exit
exec ssh -t virt PATH=$PATH LANG=$LANG "${@}" 2> /dev/null
```

[モデルB]据え置き2台構成（デスクトップMac、サーバUNIX系OS）

次に、据え置きタイプのMacとUNIX系サーバを連動させる方法を説明します(**図9**)。古くなったMacBook AirやMacBook Proを据え置きとして使う場合にも使えます。

システムの組み方：ユーザアカウント

ユーザアカウントについてはモデルAと同じように設定します。

システムの組み方：ネットワーク

Mac OS XとUNIX系OSサーバ

▼リスト9　openスクリプト：ゲストでopenコマンドを実行するとホストで実行されるスクリプト

```
#!/bin/sh

l=""
while getopts etfFWRnghb:a: o
do
    case $o in
    e|t|f|F|W|R|n|g|h)
        l="$l -$o"
        ;;
    b|a)
        l="$l -$o $(echo $OPTARG | sed 's/ /\\ /g')"
        ;;
    *)
        exit 1
        ;;
    esac
done
shift $((OPTIND - 1))
while [ $# -gt 0 ]
do
    case "$1" in
    --args)
        l="$l --args"
        shift
        break
        ;;
    *)
        l="$l $(realpath "$1" | sed 's/ /\\ /g')"
        shift
        ;;
    esac
done
while [ $# -gt 0 ]
do
    l="$l $(echo "$1" | sed 's/ /\\ /g')"
    shift
done
exec mac open "$l"
```

▼リスト10　copyスクリプト：システムクリップボードへコピー

```
#!/bin/sh

case $(hostname -s) in
host)
    pbcopy
    ;;
virt)
    xsel --input --primary --secondary
--clipboard
    ;;
esac
```

▼リスト11　pasteスクリプト：システムクリップボードからペースト

```
#!/bin/sh

case $(hostname -s) in
host)
    pbpaste
    ;;
virt)
    xsel --output --clipboard
    ;;
esac
```

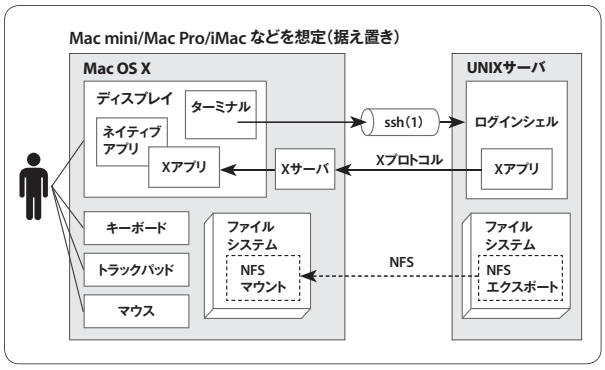
▼表3　Mac OS XとUNIX系OSサーバのネットワークアドレス

項目	内容
Mac OS Xのホスト名	mac
UNIXサーバのホスト名	unix
Mac OS XのIPアドレス	192.168.1.10
UNIXサーバのIPアドレス	192.168.1.20

▼表4　NFSマウントの対象

UNIX系サーバのディレクトリ	Mac OS Xでのマウント先
/Users/daichi/Desktop	/Users/daichi/z/Desktop
/Users/daichi/Documents	/Users/daichi/z/Documents
/Users/daichi/Downloads	/Users/daichi/z/Downloads
/Users/daichi/Music	/Users/daichi/z/Music
/Users/daichi/Pictures	/Users/daichi/z/Pictures

▼図9　Mac ProやiMac、Mac miniのような据え置き環境とPCサーバを組み合わせるモデル



のネットワークアドレスはここでは**表3**のように設定しておきます。このアドレスはLANの設定に合わせて読み替えてください。

ネットワークの設定はMac OS Xであれば「システム環境設定」から、UNIX系OSの場合はそれぞれのやり方に合わせて設定してください。UNIX系の場合は基本的に**ifconfig**(8)コマンドでネットワークアドレスを設定できます。

システムの組み方：データディレクトリ共有

どちらを母体にするかによりますが、ここではUNIX系サーバのほうがストレージのサイズが大きいと仮定して、**データは基本的にUNIX系サーバで保持する**ものとします。作業はUNIX系サーバにログインして行うことになります。UNIX系サーバとMac OS Xとのデータ共有は、UNIX系サーバのユーザのデータ領域

を必要に応じてMac OS X側からNFSでマウントするものとします(**表4**)。

UNIX系サーバ側の/etc/exportsに**リスト12**のような設定を追加します。/etc/exportsの書き方には実装ごとに制限や書き方がありますので、環境に合わせて設定してください。たとえばFreeBSDの場合、同じファイルシステム上のディレクトリは1行に書く必要あります。ZFSのデータセットになっている場合にはそれぞれ個別に記述できます。**リスト12**の例だと、/Users/daichi/MusicはZFSのデータセットなので別の行に書いてあります。

Mac OS X側では、NFSでこれら領域をマウントするためのスクリプトを用意しておきます。たとえば**リスト13**のようなスクリプトを「nfsctl」といった名前で作成して、パスの通ったディレクトリに実行権限を付与した状態で配置しておきます。

Mac OS X側で`nfsctl attach`でマウント、`nfsctl dettach`でアンマウントです。UNIX系サーバ側でMac OS Xの領域をマウントして使いたいなら、同じ要領でマウントするスクリプトを用意します。Mac OS X側でNFSマウントする場合、オプションに`-o nfc`を設定するのを忘れないようにします。また、文字列をUFC(Unicode Normalization Form C:ユニコード正規化形式C)に変換しておかないと濁点などを含むファイルやディレクトリなどが適切に扱えなくなります。

システムの組み方:ssh(1)による双方向ログイン

相互に`ssh`(1)でログインできるように、~/.configを設定します。連載第2回目で紹介したように、物理サーバ間の場合はXクライアント・サーバ通信のほうが負担が少なかったので、ssh(1) X11フォワーディングの設定は入れない設定にします(**リスト14**、**15**)。

システムの組み方:スタートアップスクリプト

Mac OS XからUNIX系サーバにログインする段階で、UNIX系サーバで動作するXクライアントがMac OS Xで動作するXサーバにログインできるように、`xauth`(1)を自動的に実行するようにします。しくみは連載第2回目で紹介したとおりです。もっとも実行が優先されるパスとして~/bin/などを追加して、そこにssh(1)のラッパスクリプトを作成して置いておきます。紙幅の都合でコードを掲載できませんでしたが、ご興味のある方は本誌サポートサイト[注1]からダウンロードできるようにしておくのでご覧ください。

このスクリプトでMac OS XからUNIX系サーバに対して`ssh unix`でログインしようとすると、XQuartzが起動していなければXQuartzを起動し、さらにUNIX系サーバのXクライアントがXサーバにアクセスできるようにxauth(1)コマンドがUNIX系サーバで実行されます。

システムの組み方:融合させるためのシェルスクリプト

基本的にモデルAで作成したシェルスクリプトのホスト名やIPアドレスを、この環境向けに編集することで使用できます。

以上で、モデルBの環境構築作業は完了になります。

[発展系モデルC]モバイルにも据え置きにも使うモデル

持ち運んでも使うし据え置きとしても使うと

注1) http://gihyo.jp/magazine/SD/archive/2014/201405/support(本記事で取り上げたコードすべてがダウンロードできます)

▼リスト12 /etc/exportsのサンプル(FreeBSDの場合)

```
/Users/daichi/Desktop /Users/daichi/Documents /Users/daichi/Downloads /Users/daichi/Pictures ➥
-mapall=1002 -network 192.168.1.0 -mask 255.255.255.0
/Users/daichi/Music -mapall=1002 -network 192.168.1.0 -mask 255.255.255.0
```

▼リスト13　nfscltスクリプト

```
#!/bin/sh

case "$1" in
attach)
    # 文字列をUFC(Unicode Normalization Form C：ユニコード正規化形式C)に
    # 変換して送信するオプション -o nfc を指定しておかないと、濁点を含む
    # ファイルなどが適切に扱えなくなる。
    sudo mount_nfs -o nfc 192.168.1.101:/Users/daichi/Desktop   /Users/daichi/z/Desktop
    sudo mount_nfs -o nfc 192.168.1.101:/Users/daichi/Documents /Users/daichi/z/Documents
    sudo mount_nfs -o nfc 192.168.1.101:/Users/daichi/Downloads /Users/daichi/z/Downloads
    sudo mount_nfs -o nfc 192.168.1.101:/Users/daichi/Music     /Users/daichi/z/Music
    sudo mount_nfs -o nfc 192.168.1.101:/Users/daichi/Pictures  /Users/daichi/z/Pictures
    ;;
dettach)
    sudo umount -f /Users/daichi/z/Desktop
    sudo umount -f /Users/daichi/z/Documents
    sudo umount -f /Users/daichi/z/Downloads
    sudo umount -f /Users/daichi/z/Music
    sudo umount -f /Users/daichi/z/Pictures
    ;;
esac
```

▼リスト14　Mac OS X側の~/.ssh/configファイル

```
Host unix
    Hostname            192.168.1.20
    IdentityFile        ~/.ssh/id_rsa
    ForwardAgent        yes
```

▼リスト15　UNIX系サーバ側の~/.ssh/configファイル

```
Host mac
    Hostname            192.168.1.10
    IdentityFile        ~/.ssh/id_rsa
    ForwardAgent        yes
```

いった場合、モデルAとモデルBの設定を両方とも施します。Mac OS Xのホスト名を統一する必要があるので、たとえば「mac」という名前にするとしましょう。そのうえでシェルスクリプトにホストを加味した分岐処理を加えてあげることになります。

システムの組み方：融合させるためのシェルスクリプト

スクリプトはOSごとに個別に持っていると管理が面倒になるので、どのホストで実行してもよいように組み上げます。例として、**リスト7**のsshスクリプトや**リスト8**のmacスクリプトをどちらの環境にあっても動作するように書き換えましたので、本誌サポートサイトからダウンロードしてみてください。

ここから先は用途に応じてスクリプトを書いていく感じになります。不便だと感じたら、すぐにスクリプトを作成して違和感を吸収していきます。使い込み、作り込むことでもっと体になじむ環境ができあがります。

手になじむ道具を作りあげる

これまで3回に渡ってNFS、ssh(1)、X Window Systemについて取り上げ、それぞれをどのように使えばよいか、最終的にどういった組み合わせ方があるかを説明しました。今回紹介した内容は基本的なことばかりです。もっと複雑なネットワーク構成であったり、いったん踏み台サーバを経由してssh(1)する必要がある環境であるとか、さまざまな要求に対しても基本的に今回紹介した技術の組み合わせで対応できます。

毎回煩わしい操作をしているとか、人間がする必要のない操作を繰り返していると感じたときは、シェルスクリプトを書いて自動化してしまうのがよいでしょう。こうしたスクリプトの作成と環境の整備を繰り返していくことで、自分のノートPCが自分の手になじんだ扱いやすい道具になっていきます :) SD

「Web標準技術」で行う Webアプリのパフォーマンス改善

短期集中連載

第1回 ファイル読み込みで高速化を図る

Writer 川田 寛(かわだ ひろし)
技術者コミュニティ「html5jエンタープライズ部」部長
NTTコムウェア株式会社 技術SE部
URL furoshiki.hatenadiary.jp Twitter @kawada_hiroshi

HTML5のW3C勧告化が間近に迫っています。リッチなコンテンツが作れることはもとより、実用的なパフォーマンスが得られるのかが気になるところです。今回から3回にわたって、Web標準技術におけるパフォーマンス改善手法について解説します。ブラウザの対応状況や業務での実用性などもふまえて検証していきます。

ブラウザはネイティブアプリに勝てないのか?

いつからでしょう。Webアプリはよくネイティブアプリと比較されます。以前では、ブラウザはHTMLドキュメントを文章として表示するための単なる「ドキュメントビューア」だったわけですが、Web2.0やjQueryなどがもてはやされ、すっかり目的も用途も変わってしまいました。今では、アプリケーションのクライアントを支える「プラットフォーム」として、さまざまな業界を巻き込み幅広い領域で活用されるようになりました。

そして最近の議論の中心は、やはりHTML5です。エンジニアの目から見ると、さかのぼること2009年、「Google I/O」で高い注目を集めました。ベンダに縛られない、オープンなクライアントプラットフォーム技術であるHTML5に、誰もが多くの期待を寄せたのではないでしょうか。

ところが2012年、Facebookのマーク・ザッカーバーグ氏の「HTML5に失望!」「HTML5はまだ早い!」とメディアを騒がせる事件が有名となり、「HTML5って本当に大丈夫なの?」と、ポテンシャルを不安視する声も聞こえ始めました。Webとネイティブの比較は、このころから議論され始めたように思います。

世の中はいったい何を不安視しているのでしょう? なぜ、Web標準技術ではダメなのでしょう? Web標準技術とネイティブ、2つを比較したとき、議論として盛り上がりをみせるのはやはりこのテーマです。

「パフォーマンス」

「新しいデバイスに追従できない!」「ビジネス色が強過ぎる仕様はなかなか標準化が進まない!」という話もありますが、それは時間が解決する問題です。ただ、Web標準技術はいくらあがいても、パフォーマンスではネイティブを超えることはできません。

とはいえ、企業向けのシステムなどでは、ネイティブほどのパフォーマンスは望めないもののWebアプリがデスクトップアプリの代わりとしてずっと使われてきました。ブラウザでデスクトップアプリのような動作をさせようと、ポップアップを駆使してUIから戻る/更新ボタンを消し去ったり、JavaScriptで右クリックを禁止させたりといった泥臭い対策が施されることも少なくなかったようです。

最近では、「そういう用途を目的としたプラットフォームを作ってしまおう」というアイデアが生まれていたりします。MEAP(Mobile Enterprise Application Platform)という、Web標準技術でマルチデバイスなアプリケーション

を開発する環境も、徐々にデスクトップアプリの世界に進出する勢いです。JavaでもJavaFXというブラウザのレンダリングエンジンを活用する技術を作っていますし、ベンダ製品だとデスクトップアプリの置き換えを狙ったようなJavaScriptライブラリが広がりを見せています。

今後は、そのオープン性の高さからWeb標準技術を使わざるを得ない、そんな状況も増えていくのでしょう。ネイティブに勝てないのはわかっていても、取り巻く環境や業界の波が、HTML5でやらざるを得ない状況に追い込んでいるように思えます。

Web標準でできるパフォーマンスへのアプローチ

「このWebアプリ、すごく重い！」という場合に、よくあるのは次の2つ議論です。

・JavaScriptのパフォーマンスを上げる
・DOMアクセスを最適化する

前者は、JavaScriptのアルゴリズム、JavaScriptVMの特性をふまえたうえで、高いパフォーマンスが得られるよう記述を工夫する方法。後者は、DOMアクセスを最適化するために、利用するAPIの種類や使い方を工夫しようというもの。どちらもプログラムの書き方に関することです。プログラマはパフォーマンスの高いプログラムを書くことに喜びを感じるので、こういう観点で議論するのが大好きです。

しかし、大規模なアプリケーション開発になると、パフォーマンス対策をプログラマだけに依存させてしまうのは、あまり良い方法とは言えません。もちろん、プログラマがそういうことを意識するのは重要ですが、ある一定のラインを超えるとそれはハッカーの仕事です。

マネージャやITアーキテクトなら、プロジェクトに参加する全プログラマがハッカーであることを前提に開発を進められないので、特定の人間に依存しないアプローチを試みたいと考えるでしょう。それもプロジェクトの早期から、せめて設計の段階から、パフォーマンスが明らかに悪くなるとわかっているところぐらいは、何か手を打てないかと考えるはずです。

実はこうした問題に対して、Web標準側でもさまざまなアプローチ方法が議論されています。W3Cでは、「Web Performance Working Group」と呼ばれる団体が、パフォーマンスの改善をめざした仕様を策定しています。Web標準技術固有のパフォーマンス問題に対して、対策するための仕様を検討し続けてきました。

このワーキンググループは、Microsoft、Google、Mozillaなどのブラウザベンダが集まり議論していますが、そこになぜか、何のWebプラットフォームも持たないFacebookが参加していたりします。2012年には「まだダメだ！」と言い放った彼らですが、1年後の2013年にはちゃっかり「アプリのHTML5化を進めよう！」という構想を再び立てていたりします。ダメだと思った点を、改善しようと取り組んでいるのです。

最近はモバイルの登場で、安定した通信環境を前提にできなくなりましたし、CPUやメモリリソースも意識しなくてはいけない。バッテリー稼働ですので、電気消費だって気にしなくてはいけない。そんな状況であるにもかかわらず、Webアプリは今まで以上のことをやろうとするわけですから、パフォーマンス改善のために求められる機能も増えつつあります。そして、Web標準における多くのパフォーマンス系の機能は、ほかの機能とは比較にならない速度で標準化が進められ、ドラフト公開からわずか3年足らずでW3C勧告に到達しています。

本連載では、目的別に分類して、Web標準でできるパフォーマンス改善方法をまとめます。もし、あなたが開発しているWebアプリに、パフォーマンスを悪化させる特徴的な機能が盛り込まれるのであれば、本記事で解決のためのヒントを探してみるのも良いかもしれません。

ファイルを先に読み込ませる

Webページの表示処理は、とにかく無駄が多いです。ハイパーリンクやフォームのボタンをクリックしてから、サーバへHTMLドキュメントのリクエストを行い、HTMLドキュメントから参照されている画像ファイルやJavaScript/CSSファイルを読み込み、逐次画面に表示させていく、という過程を踏みます。ユーザはコンテンツの内容がある程度見えるようになるまで、待つ必要があります。

最近は、JavaScriptフレームワークが流行っていますが、モノによってはメガバイトレベルのものがあります。大量の画像ファイルを読み込ませるだけでなく、フォントファイルに音声ファイルや動画ファイルと、とにかく大容量のデータを読み込ませるWebアプリは少なくありません。いくらコンテンツがすばらしくても、表示されるまでの間、ユーザは待つことを強いられます。これが悪化すると、コンシューマ向けサービスの場合は機会損失につながりますし、ビジネスアプリだと作業効率の低下につながります。目をつむれない問題です。

この読み込みから表示までのプロセスを改善すべく、余っているネットワークリソースに目を向けました。ネットワークは、Webページの読み込みをしている間リソースを限界まで食いつぶします。しかし、読み込みが終わると、ほとんど何もしていないスカスカの状態になります(**図1**)。ダイアルアップ接続の時代はこの方法が歓迎されたのですが、今は常時接続が当たり前の時代。

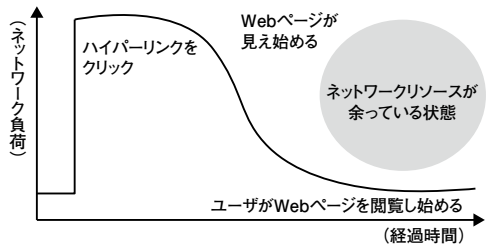
そこでワーキンググループは、余っているネットワークリソースを有効活用するためのアイデアとして「prerender」「prefetch」「dns-prefetch」という仕様の標準化を始めました。Webページを表示している間に、裏でこれから使う可能性のあるファイルなどのリソースを先読みできます。

先読みは、その目的に応じて**リスト1**のタグをhead要素内に記述するだけと、非常にシンプルです。

prerenderとprefetchは、ファイルを先読みすることでパフォーマンスを改善する技術です(**図2**)。dns-prefetchは、モバイル環境で問題になりがちなDNSの名前解決を事前実行し、パフォーマンスを改善する技術です。ここでは、prerenderとprefetchについてより詳しく触れてみましょう。

prerenderは指定したWebページの表示に必要なファイルを丸ごと先読みさせる技術です。URIにHTMLドキュメントを指定すると、HTMLドキュメント内で参照されているCSS/JavaScriptも一緒に先読みしてキャッシュしてくれます。さすがにJavaScriptの初期実行ま

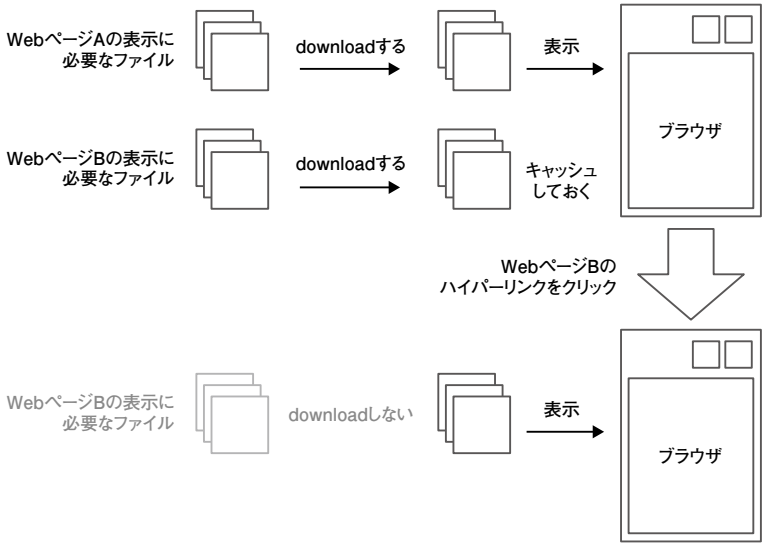
▼図1　Webページ読み込み時のネットワークリソースの偏り(ユーザのWeb閲覧時に注目)



▼リスト1　prerender、prefetch、dns-prefetchの記述例

```
<!-- 次のページの先読み -->
<link rel="prerender" href="./?page=2" />

<!-- 近いうちに必要になるリソースの先読み -->
<link rel="prefetch" href="https://ajax.googleapis.com/ajax/libs/angularjs/1.2.6/angular.min.js" />

<!-- 近いうちに必要になるDNSの名前解決 -->
<link rel="dns-prefetch" href="http://example.com/" />
```

▼図2　ファイル先読みのイメージ



ではしてくれませんが、「pre(事前に)+render(レンダリングする)」という名前のとおり、Webページを裏で表示して待機しているような振る舞いをします。

対して、prefetchはファイル単体での先読みです。prerenderと違って、HTMLドキュメント内で参照されているCSS/JavaScriptなどのファイルを読み込むような動きはしません。そもそも、指定するURIはHTMLドキュメントよりも画像ファイルやJavaScriptファイルなどの単体のリソースの先読みが求められるケースに向いています。

prerenderは1つのURIしか指定できないのに対し、prefetchは複数指定できるという特性を持ちます。たとえば、Internet Explorer 11(以下、IE11)だと、prerenderは1つまでしかページを先読みできませんが、prefetchは10個のファイルまで指定できます。

実際にどういう活用方法があるでしょうか?

まずは、prerender。これはニュースサイトのようないわゆる「Webサイト」と呼ばれるタイプのサービスに向いています。ニュースサイトだと、1ページ目を表示している間に、裏で2ページ目を読み込んでおくような活用方法が挙げられます。インターネット上のWebサイトはSEO(Search Engine Optimization:検索エンジン最適化)が求められるため、JavaScriptも装飾程度にしか活用されないことが多いでしょう。このようなタイプのWebページは、HTMLドキュメント内で表示に必要なファイルは一通り定義されていることが多いため、高い効果が期待できます。JavaScriptの初期実行は先読みの段階で行われないため、ソーシャルボタンや広告が無駄に取得されるのを避けることもできます。

一方で、prefetch。こちらはWeb版Gmailのような「Webアプリ」と呼ばれるタイプのサービスに向いています。JavaScript内から、データやファイルをガッツリ読み込むタイプのものです。prerenderはHTMLドキュメント内で参照するファイルぐらいしか読み込んでくれませんが、prefetchはprerenderでは拾えないような、たとえばJavaScript内から参照されるようなファイルを、直接指定して先読みできます。

最近は、SPA(Single-page Application:単一ページアプリケーション)と呼ばれる、単一のWebページ上で複数の画面を扱えるようにするWebアプリのアーキテクチャが広がりを

見せています。ただ、SPAはファイルサイズが大きいため、初期読み込みに時間がかかるという問題を抱えています。読み込みのパフォーマンスを高めるために、ファイルそのものを圧縮したり結合したりとさまざまなアプローチで改善を試みていますが、それでも大規模ならメガ超えが避けられないことがあります。しかも、SPAは高度なオフライン処理を行うのに向いており、ネットワークが貧弱な環境でも多くの機能を与えることができたりします。ネットワークの貧弱さを補うための対策が、読み込むファイルを肥大化させ、貧弱なネットワークに耐えられないアプリになるという、ジレンマに悩まされるのです。

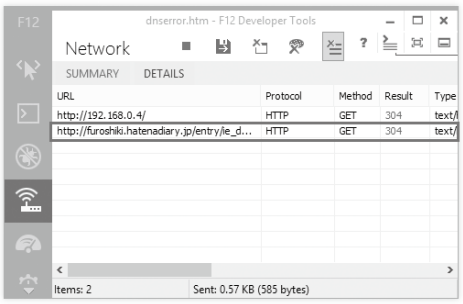
このような特性を持つアプリでは、ログインページを表示している間に、裏でprefetchを使ってSPAを構成するファイルをある程度読み込んでおく、というアプローチが行えるでしょう。「SPAって初期読み込みが遅い！」「この前、タブレットで読み込もうとしたら、1分待たされたんですけど！」という意見をよく耳にするのですが、その多くはこうしたWeb標準の機能で改善できたりします。

なお、動作チェックについては、ブラウザ付属の開発者ツールを使います。筆者のブログのページをURIに指定したものを参考にしてみましょう。prefetchのようなシンプルなものは、開発者ツールの「ネットワーク」タブの中で確認できます(**図3、4**)。自身のページに加えて、prefetchで指定されたHTMLドキュメントが読み込まれています。

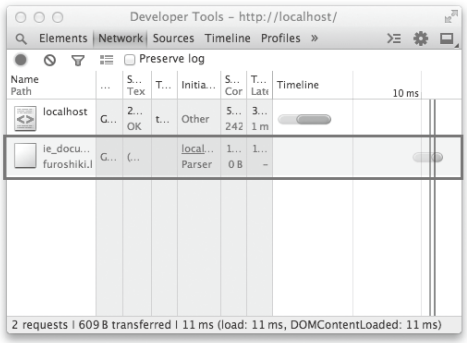
対してprerenderは、IE11では「ネットワーク」、Chromeはやや特殊で「タスクマネージャ」で確認が行えます(**図5、6**)。Chromeは隠蔽されているためわかりにくいですが、IE11の例では、HTMLドキュメント以外にも大量のファイルの読み込みを行っているのが確認できます。HTMLドキュメント内に記述した、scriptタグやimgタグで指定した外部ファイルが取得されているのです。

prefetchは最新の主要ブラウザでは対応済み

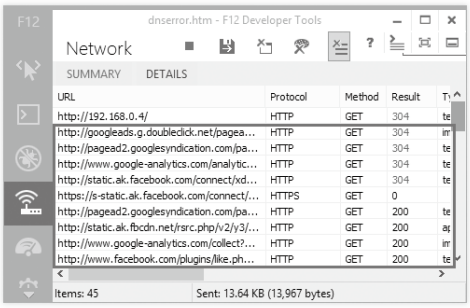
▼図3　IE11でのprefetchの確認方法



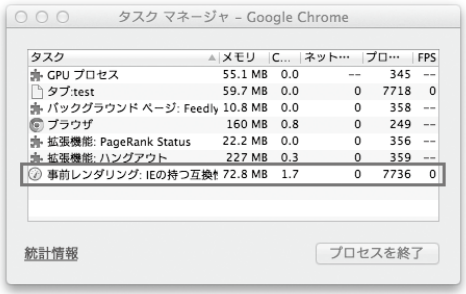
▼図4　Chromeでのprefetchの確認方法



▼図5　IE11でのprerenderの確認方法



▼図6　Chromeでのprerenderの確認方法



ですが、prerenderは2013年末までChromeの独自機能という扱いだったため、現在、最新のFirefoxでもまだ未対応です。基本は無視され動作しないという、ベストエフォートの扱いでしょう。未対応ブラウザでも動かしたいというのであれば、IE8からサポートされているWebStorageを利用して、擬似的にこの機能を再現させるアプローチもあります。

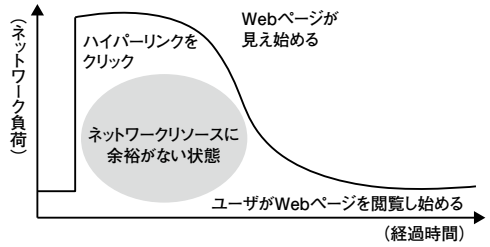
ただ、この方法はお世辞にもきれいな方法とは言えず、また、Web標準側が想定している使い方かと言えば正直微妙なところです。このため、本記事では解説を割愛させていただきます。

ファイルを後で読み込ませる

ファイルの先読みはパフォーマンス改善で非常に効果的な手段ですが、読み込めるファイルの数には限界がありますし、ネットワークリソースにも限界があります。なんだかんだ言っても、Webページの読み込み時は、ネットワークリソースを限界まで使いつぶして、必要なファイルをすべて読み込むという前提のもと、作り方を考えなくてはいけないでしょう。

Webページのファイルの読み込みは、リソースの種類にもよりますが、並列で読み込むことが多いです。ファイルサイズが大きい場合、並列に読み込んでいるファイルすべてが、ネットワークリソースを奪い合うように共有するため、個々のファイルはゆっくりと読み込まれます。同時に何本の並列化が行われるかは、HTTPのバージョンやブラウザの種類／設定に依存するのですが、いずれにせよ、Webページ読み込み時にネットワークリソースに負荷が一気に集中するという問題に変わりはありません(**図7**)。

▼図7　Webページ読み込み時のネットワークリソースの偏り(ハイパーリンククリック直後に注目)



何をもってパフォーマンスが高いとするか、その定義しだいでも変わってきますが、体感速度を改善させたいのであれば、初期表示時に見えているエリア内だけでも早く表示してしまいたいところです。背景画像などは優先度を下げても良いでしょう。縦長のサイトだと、スクロールしないと見えない画像ファイルを多く含むことになりますが、それをわざわざWebページの読み込み開始直後に取得する必要はないはずです。動画／音声ファイルやCSS/JavaScriptファイルなど、あとで読み込んでも大丈夫なものもあるでしょう。

こうした「あとに読み込ませたい」というニーズに対しては、Web標準の「Resource Priorities」が有益な手段です。HTMLドキュメント内から参照されるファイルに対して、読み込むタイミングの優先付けを行うものです。Webページの読み込み直後、ネットワークリソースに余裕がない状況下で、最善のパフォーマンスが得られます。

これも利用方法は、非常にシンプルです。HTMLとCSSの双方から指定可能です。

・タグで指定する場合

```
<img src="./img/hoge.png" lazyload />
<img src="./img/fuga.png" postpone />
```

・CSSで指定する場合

```
img.hoge { resource-priorities : lazy-
load; }
img.fuga { resource-priorities : post
pone; }
```

サンプルでは「lazyload」と「postpone」の2種類の設定方法を紹介しています。lazyloadはプライオリティが低いという意味です。ほかのファイル読み込みに対して優先的にネットワークリソースを与えるという動作をします。

対して、postponeは必要なタイミングでの読み込みを意味します。img要素に対してpostponeを指定すると、スクロールして対象のimg要素が画面内に入り込むまで、読み込みを開始しなくなります。

今のところ、実用的なのはlazyloadだけです。postponeは現在、IEの最新版である11でもまだサポートされていません。lazyloadもpostponeも、Web標準としては2013年4月にドラフトが公開された比較的新しい仕様であるため、残念ですが、IE10以下を含む少し古いブラウザだとこの機能を持ちません。どうしても必要なら、JavaScriptライブラリを活用することになります。

画像ファイルでは、「Lazy Load Plugin for jQuery」[注1]が有名です。今回紹介したlazyload/postponeの動作を、完璧ではないにせよ模倣してくれます。

また、JavaScriptは古くからdeferという属性が実装されており、単純な遅延読み込みであればこれでも代替できます。

```
<script src="xxx.js" defer></script>
```

とはいえ、最近のJavaScriptファイルは、jQueryのプラグイン然り、ファイル間に依存性を持つものが多い状況です。遅延読み込みも大事ですが、それ以上に、AMD(Asynchronous Module Definition)と呼ばれる依存性解決を行える機能が求められます。W3Cのメーリングリストでも、2013年12月にこの話が持ち上がりましたが、今のところ反応がなく、まだまだという印象です。実用性で見ると、Web標準からはややズレますが、しばらくはCommonJSというサーバサイドJavaScript標準のアイデアを継承したJavaScriptライブラリ「RequireJS」[注2]を利用するのが良いでしょう。

CSSの遅延読み込みは、FOUC(Flash of Unstyled Content)と呼ばれる、ページ読み込み時に一瞬だけ表示が乱れる現象を発生させるため、あまり推奨されるものではありません。とはいえ、どうしても必要な場合は、クロスブラウザ対策で有名なModernizr[注3]のloadメソッドや、これと同じ機能を単独で実装するyepnope.js[注4]が、CSSファイルの遅延読み込みにも対応しており、代替手段として利用できます。

ファイルを圧縮する

ファイルを読み込むタイミングの制御だけでなく、ファイルをそのもの圧縮してしまおうというアイデアもあります。

圧縮というと、JavaScriptやCSSファイル内のスペースや改行を削除する処理を想像する方もいますが、ここで取り扱うのはもう1つ下のレイヤの機能です。ファイルの種類を問わず、ファイルをgzip圧縮させた状態でクライアント側に送信することで、ネットワークリソースを効率化させよう、というアイデアです(**図8**)。画像ファイルやPDF、Microsoft Office 2007以降で採用されているファイル形式(OpenXML/ECMA-376)やODF(OpenDocument Format)のような、圧縮を含むフォーマットに対してはまったく効果が期待できません。テキストファイルのようなバイナリレベルで未圧縮のものに適しています。

この機能を利用するには、ファイルが圧縮されていることをブラウザに伝えるため、HTTPレスポンスヘッダに、次のプロパティを設定します。

注1) http://www.appelsiini.net/projects/lazyload
注2) http://requirejs.org/
注3) http://modernizr.com/
注4) http://yepnopejs.com/

▼図8　ファイルの圧縮転送のイメージ

```
Content-Encoding: gzip
```

実際には、HTTPヘッダにプロパティを付与するような設定を直に作りこむのは、ものすごくハックしている感じがして、サーバ管理者だと抵抗感を感じること間違いなしです。現場で運用する場合は、たとえばApacheでは、1.xはmod_gzip、2.xはmod_deflateを用いてファイル圧縮やHTTPヘッダの制御を自動化させるのが一般的です。deflateという見慣れないキーワードが出てきましたが、これは圧縮フォーマットの1つです。実はここで指定する圧縮方式はgzipじゃなくても良いわけですが、ブラウザ側の対応状況を鑑みると、gzipのほうが安全です。名前こそmod_deflateですが、実際にはgzipを使うというやや泥臭い感じがする設定を作ることになります。

筆者の手元の環境では、mod_gzipの場合、Apacheのhttpd.confファイルに**リスト2**のような設定が必要になりました。これは、古くからの作法に則り、拡張子が「css」「js」のものに対して圧縮設定をしています。最近のWeb標準、たとえば、WebGLのようなものになると、これら以外のテキストファイルも含まれたりするので、そこは柔軟に対応する必要があるでしょう。

この標準はわりと歴史が古く、W3CではなくRFC 2616として標準化されています。十分に枯れていて、かなり実用的な技術です。ベンダ系のWeb開発製品だと、パッケージング／ビルド／デプロイ処理の中に、デフォルトでこの圧縮設定が組み込まれているものもあります。

ただ、IE6で最新のサービスパックが適用されていないものには、不具合があるという話を聞きます。筆者は手元の環境で再現ができていないため断言はできないのですが、実質的にはIE7からの活用が望ましいと思われます。この記事が世に出るころには、Windows XP版IE6もEOLを迎えていますので、この問題はあまり意識しなくて良いと、信じたいところですね。

また、この機能に未対応のブラウザでは、プロキシが悪さをするという問題もあります。圧縮がらみのネタは語りだすとキリがないので、解説はこのあたりで止めておこうかと思います。

まだ終わりません

さて、今回はファイルの読み込みにフォーカスして、パフォーマンスを改善するWeb標準について紹介しましたが、いかがでしたでしょうか。パフォーマンス自体は、最近議論が始まったわけでもなく、RFCにもその痕跡が多く残されています。gzipなどがまさにそれですね。古い技術が役立たずになるのかと言えばそうでもなく、いつになってもそのエッセンスは活用され続けるものです。

次回は、こういう古い標準の考え方を取り込み、パフォーマンスの改善に取り組んだ技術であるネットワークプロトコルやキャッシュの改善について紹介します。また、今回は「速く動かす」を目的とした改善方法を紹介しましたが、「速く動かす」=「パフォーマンスが高い」というわけでもありません。あえて「遅く動かす」ことを視野に入れる必要もあるのです。そういう話もしようかと思います。

それでは次回、またお会いしましょう！　ありがとうございました。SD

▼リスト2　Apacheにおける圧縮転送の設定例

```
<IfModule mod_gzip.c>
  mod_gzip_item_include file ¥.js$
  mod_gzip_item_include mime ^application/x-javascript$
  mod_gzip_item_include file ¥.css$
  mod_gzip_item_include mime ^text/css$
</IfModule>
```

思考をカタチにするエディタの使い方

るびきち流 Emacs超入門

Writer | るびきち twitter@rubikitch | http://d.hatena.ne.jp/rubikitch/ http://www.rubyist.net/~rubikitch/

新連載 第1回 なぜEmacsをお勧めするのか?

Unixとともに育ち、多くのプログラマの道具として愛されてきたEmacs。本連載では、その使い方だけでなく、その背景にある思想からもEmacsの解説を行います。まずはエディタとしての性格、一番大きなメリットである文字入力の一元化、そして必須のEmacs Lispとの関係など、最初に知っておくとよい情報をガイダンスします。

ようこそ、Emacsの世界へ

はじめまして、Emacsを溺愛しているるびきちと申します。『Emacsテクニックバイブル』と『Emacs Lispテクニックバイブル』を著しました。本誌でも昨年のEmacs特集のときには記事を書かせていただいています。Emacs愛が高じて毎週土曜日に無制限メール相談付きの有料メルマガも発行しています。そして今回はこの連載を任されました。ありがとうございます。

色あせぬ・尽きぬ魅力

筆者は、1996年はじめからEmacs一筋でたくさんのEmacs Lispプログラムに触れ、自分でも作ってきました。今年で19年目になりますがいまだにEmacs愛は冷めることを知りません。

今回は第1回目ということで、難しいことは抜きにしてEmacsとは何なのか、そしてEmacsの魅力について語っていこうと思います。

潤沢リソース時代のエディタの役目とは

「Emacsとは何なのか?」と聞かれればテキストエディタというのが一般的な答えです。Emacs vs Vi(m)はUnixテキストエディタ界の「きのこ・たけのこ戦争」といったところで、両者は人気を二分している状況でした。最近はSublime TextやAtomなどの新しい勢力も現れてきているうえ、Vimにも後れをとっている状況です。今ではEmacsをわざわざ使う人はますます減ってきています。その流れでも、あえてEmacsを選ぶ意味があります。

元来テキストエディタはとても軽いアプリケーションで、シェルから瞬時に立ち上がり、テキストを変更したら即終了という使い方をするものです。Vi(Vimではない)はまさにそんな使い方をすることが念頭にありました。

対してEmacsはヘビー級のアプリケーションで、1つ立ち上げたらログイン中はずっとそのEmacsセッションを使い回す文化です。いわゆる「Emacsひきこもり」ですね。Emacsはあまりにも大きいプログラムだったので、リソースが厳しかった大昔は、Emacsを複数個立ち上げると怒られたものでした。今はリソースが潤沢なのでEmacsを複数個立ち上げたくらいではびくともしません。Vimなど、ほかのテキストエディタも高機能化・充実したプラグインによりひきこもり生活が送れるようになっています。Emacsは30年以上昔からひきこもり文化であったことから、いかに特別な存在であったかがうかがい知れます。今はヘビー級ひきこもりアプリケーションといえばWebブラウザにその地位を譲ってしまっていますが、Emacsは先駆者といえます。

Emacs Lisp

Emacsはテキストエディタの皮をかぶったLispマシンであり、1つの世界をつくっています。Emacs環境にはさまざまなアプリケーションが存在します。これらはすべてEmacs Lispで書かれています。

シェル：eshell
ファイラ：dired、nav、speedbar
ページャ：view-mode
メーラ：gnus、mew
Twitterクライアント：twittering-mode
Webブラウザ：emacs-w3m
RSSリーダー：newsticker
Skypeクライアント：skype
カレンダー：calendar
ワープロ：org
スケジュール・TODO管理：org
プレゼンテーションツール：org
バージョン管理：vc、magit、psvn、pcvs
メディアプレイヤー：emms
テキスト翻訳：text-translator
ゲーム：gomoku、tetris、doctor
manインターフェース：man
grepインターフェース：grep
……その他もろもろ

Emacsの魅力は以下の3点に集約されます。それでは1つ1つ見ていきましょう。

・文字入力が一元化される
・外部プログラムとの連携が得意
・拡張言語がLispであること

文字入力の一元化

Emacsを使うことによって受ける最大の恩恵が「文字入力の一元化」です。テキストエディタとしてのEmacsは成熟していて、強力なカーソル移動や入力機能が使えます。もちろん好きな機能を自分で追加できます。それらの機能はプログラミング時だけでなく、メールや文書作成、TwitterやIMのメッセージ入力でさえも同じように使えるのです。

想像してみてください。あなたは今プログラミング用には普通のエディタを使っています。メール、Twitter、Skypeはそれぞれ別のソフトを使っています。ブラウザのフォーム入力はブラウザからそのまま入力しています。ウィンドウが画面中に散らばっており、タスクを切り替えるときにアプリケーションのウィンドウを切り替えるのが面倒だと感じています。エディタで使えるはずの強力なコマンドが他のアプリケーションでは使えずにもどかしい思いをしています。ブラウザで見ているサイトの内容をURL付きで引用するメールを面倒だと思いながら書いています……。

あなたがEmacsを使うようになったら、その状況は一変します。普段使っている文字入力方法、コマンドが「文字入力をするすべての場面で」使えるようになるのです。つまり、文字入力するときは一貫して同じ方法が使えます。アプリケーションごとに操作方法を切り替えるストレスから解放され、入力する内容に集中できます。ブラウザのフォーム入力時にEmacsを呼び出せます。サイトの内容をURL付きで引用するメールなど、Emacsを使えば楽勝で書けてしまいます。シェルコマンドの実行結果をそのまま書き込むことができます。ふとアイデアを思い付いたら、一瞬でメモ入力コマンドを起動し、メモを書いたら自動的にメモ起動前のタスクに戻ってくれます。

アプリケーション間でデータのやりとりをするクリップボードなど、Emacsのキルリングと比べたらオモチャ同然です。なぜたった1つのテキストしか記憶できないのでしょうか？キルリングは大昔から何個もテキストを記憶できていたというのに。

プログラマはコマンドラインシェルも好んで

使います。文字入力の一元化というのは、シェルにも波及します。M-x shell[注1]はいつも使っているシェルをEmacs上で動かしているので、Emacsの持つ強力な機能がそのまま使えます。しかし、シェルの持つ本来の機能が使えない欠点もあります。それに対してeshellは完全にEmacs Lispで書かれたシェルであり、しっかり作り込まれています。通常のシェルとは異なりeshellはフルEmacs Lispなのでコードを書けば完全に自由にアレンジできます。しかもWindowsでも問題なく使えるので複数のOSを使う人ならばeshellは手軽でお勧めです。真新しいWindows PCが筆者に与えられた場合、真っ先にEmacsをインストールしてeshellを立ち上げるところからスタートです。

外部プログラムとの連携

Emacs自体でもEmacs Lispでいろいろなものが作れますが、それだけだと力不足なこともあります。そこで外部プログラムとやりとりすることになります。シェルコマンドの実行結果を表示・挿入することはもちろん、シェルなどの対話的プログラムを動かせます。また、プログラムをEmacsから使いやすくするコマンドがたくさんあります。「Emacsはエディタではなくて環境だ」とか「EmacsはOSだ」と言われている最大の要因はこのプロセスを扱う機能のおかげです。

各コマンドの説明

M-!は、シェルコマンドの実行結果を表示します。このコマンドを使えばシェルコマンドの実行結果を含む文書を楽に作れます。

M-x shell、M-x telnet、M-x rlogin、M-x run-rubyなどは対話的プログラムを実行する例です。Emacsのバッファ上で通常の端末と同じように実行できます。端末での実行とは違い、過去の出力を遡れるし、コピーもできるメリットがあります。

M-x compileはコンパイルコマンドを実行させて、エラー行にジャンプできます。さまざまなプログラムのエラーメッセージ表示形式に対応していて自動判別してくれます。コンパイルエラーが起きたら次にやることは該当行にジャンプすることなので、このインターフェースはとても素晴しいです。

M-x grepはその応用例でgrep -n形式の出力に対応しています。grepの出力結果から該当行にジャンプできます。Emacs Lispでgrepを書くこともできますが、速度はgrepプログラムの圧勝です。そこで検索処理をgrepプログラムに丸投げして、grepを呼び出す部分と検索結果からジャンプする部分を、Emacs Lispで書く方法を採用しました。これにより全部Emacs Lispで書くことと比べて行数を大幅に削減でき、grepの速度も活かせます。

この方法の嬉しい副産物として、grep以外のコマンドでも行番号にジャンプすることができます。M-x grepではgrepの代わりにソースコード検索に特化したackやagを呼び出してもよいのです。出力がgrep -n形式でありさえすれば任意のプログラムが使えます。

同じことはEmacs内でmanpageを開くM-x manでも言えます。これは内部でmanを呼ぶのですが、manの代わりにmanと同じようにふるまう別のプログラムに設定できます。

manpageだけだとわかりにくいので、具体例も表示してほしいですよね。それならば、ワンライナー検索サイトcommandline-fuの検索結果も出力するスクリプトを作成し、manの代わりに呼び出すように設定すればM-x manで具体例つきのmanpageが見られるようになります。外部プログラム側を拡張することで、M-x manに「憑依」する形でcommandline-fuのEmacsインターフェースも同時に手に入ります。一石二鳥ですね。

M-x find-diredはfindプログラムの引数を

注1）「M-x」は Alt を押しながら X を押すEmacs流キー表記。M-xはコマンド名を指定して実行します。Emacsのコマンドを明示的に表記するときにも「M-x コマンド名」が使われます。

入力することで、その結果をdiredで表示します。diredなので表示されたファイルを開くことはもちろん、その他のdiredのコマンドが使えます。

このように、Emacsは外部プログラムとつなぐことを得意としています。外部プログラムを呼び出して、その結果を処理することで、あたかもEmacs組み込みの機能であるかのように動作してくれます。

Lisp・関数プログラミング養成エディタ

Emacsについて語る場合、Emacs Lispについて触れないわけにはいきません。高機能でひきこもりができるエディタは今やほかにもありますが、なぜEmacsがいいのかというと、それはEmacs Lispの存在です。プログラマにとってのエディタは野球選手でいうバットやグローブのようなもので、まめに手入れをする必要があります。エディタの手入れとは設定やカスタマイズですが、EmacsではLispを使って行います。

昔から「プログラマならばLispを学べ」と言われています。実際に使うか使わないかはともかく、Lispを学べばよりよいプログラマに成長していきます。現在のプログラミング言語の多くはLispの影響を受けているので、Lispを学ぶことで、常用している言語に新たな視点を与えてくれます。

Lispは最初の関数型言語として知られていて、進化を続けて今も使われています。Lisp自体は古代の言語ですが、当初から高階関数、ガーベッジ・コレクション、クロージャーなどといった先進的な機能がありました。Emacsを使うということはEmacs Lispに日々触れるということであり、Emacsを心から愛するようになったら、それこそLisp漬けの日々です。Emacsを好きになれば、だんだんLispも好きになっていき、それがさらにEmacs愛を深めることになります。エディタのカスタマイズという身近なテーマを通してLispを学べるのです。

筆者は2月号で関数プログラミング特集の記事も書きましたが、今まさに関数プログラミングが注目されています。Emacs Lispはその性質から命令型プログラミングが主流になっていますが、腐ってもLispです。Emacs Lispから関数プログラミングを学ぶことができるんです。

LispハッカーにとってみればEmacs Lispはオモチャでしかありませんが、Emacs Lispは着実に進化しています。Emacs24でレキシカルクロージャーが正式サポートされました。CPANやAPTを連想させるパッケージシステムが登場し、現代的で強力なライブラリも簡単に導入できるようになりました。Emacs Lispだってスタイリッシュに記述できる時代です!

Lispの考え方、関数プログラミングが好きな人やこれから学びたい人ならば迷わずEmacsです。大量の括弧が嫌いな関数プログラミング好きも実はEmacs向きです。括弧アレルギーなどすぐに克服してしまう方法があるからです。

おわりに

今回はEmacsの世界に初めて足を踏み入れた人向けに、Emacsとはどんなものなのかについて書きました。今後も本連載では入門者にもわかりやすいようにEmacsの魅力を余すことなくお伝えしていきます。次回をお楽しみに!

もっとEmacsについて学びたい意識の高いあなたのために、筆者は毎週土曜日にEmacs専門メルマガを発行しています[注2]。無制限メール相談権付きであなたを徹底サポート致します。個別メールでよりよい提案を行ったり、Emacsに関するトラブルを解決いたします。月頭に登録すれば無料で1ヶ月間サポートが受けられます。次の月からは月々512円かかりますので、不満であれば解除していただいてかまいません。メルマガの登録お待ちしております。SD

注2) http://www.mag2.com/m/0001373131.html

シェルスクリプトではじめるAWS入門

――ゼロから始めるAWS API

第2回　AWS APIの利用方法と環境の構築

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

JAWS DAYS 2014に参加してきました

2014年3月15日の土曜日、新宿のイベントホールでAWSのユーザコミュニティによる交流イベント「JAWS DAYS 2014」が開催され、全国から1,000人以上の参加者が集い、大小6つのブースで並行してテクニカルセッションやディスカッションが行われました。筆者もほぼ終日参加していましたが、各ブースの発表者の声が会場内に響きわたり、参加者の皆さんが好きなタイミングで気になるセッションを立ち聞きしたり、空いているスペースで意見交換をしたりと自由な雰囲気で参加していることが大変印象的でした。

今年の会場では「Immutable Infrastructure」というキーワードが特に注目を集めていました。サーバなどのITインフラを管理運営するうえで、利用しているITリソースが「状態を持っていること」が物事を複雑にしているのであれば、「状態を持たない、不変の（immutable）」のインフラストラクチャリソースを作ればよいのではないか、そのための環境が昨今は整い始めている、というのがこの議論の起点となっています（と筆者は理解しています）。今後、AWSをはじめとするクラウドプラットフォームサービスを利用するうえで「プログラマブルであること（Infrastructure As Code）」「状態を持たせないこと（Immutable Infrastructure）」「疎結合であること」の3点を意識することの重要性が高まっていくことは避けられないのではないかと感じました。そしてその基盤技術としてのAWS APIをはじめとするWebAPIの利用知識も必須となっていくと確信したのでした。ということで、今月号の本編に入ります。

AWS APIの利用方法

AWS APIを利用するには、おもに次の3つの方法があります。

1.AWS Software Development Kits（AWS SDK）の利用
2.AWSコマンドラインインターフェース（AWS CLI）の利用
3.APIに直接アクセス

AWS Software Development Kits（AWS SDK）

AWSのサービスをプログラミング言語から扱うためのSDK（Software Development Kits）がAWS本体やサードパーティから提供されています。AWS SDKでは、AWS APIのリソースや機能をプログラミング言語ごとに抽象化してメソッドやプロパティとして提供しており、ユーザは好みの言語からAWS APIを間接的に操作できます。

2014年3月現在、AWS公式サイトで紹介さ

れているSDKには次の9種類があります。

・AWS SDK for Android
・AWS SDK for iOS
・AWS SDK for Java
・AWS SDK for JavaScript(in the Browser)
・AWS SDK for JavaScript(in Node.js)
・AWS SDK for .NET
・AWS SDK for PHP
・AWS SDK for Python(boto)
・AWS SDK for Ruby

AWSコマンドラインインターフェース(AWS CLI)

コマンドラインからAWSの各サービスを操作するための公式ツールとして、AWSコマンドラインインターフェース(AWS CLI)が、Windows向け、Mac/Linux向けに公開されています[注1]。AWS CLIには、サービス別にリソースや機能を操作するためのサブコマンドが定義されており、コマンドライン上からAWS APIを間接的に操作できます。作業手順に必要な一連のCLIコマンドをシェルスクリプトで記述し、作業の自動化も比較的容易に実現できます。

APIに直接アクセス

3つ目の方法として、AWS SDKやAWS CLIに頼らずに、生のHTTPリクエストを自力で作成し、コマンドやブラウザで直接APIに送信する方法があります。AWS SDKやAWS CLIに比べると格段に情報が少なく、AWS公式ドキュメント以外にはサポートも得られにくいというイバラの道ですが、AWS APIの動作を知る最も効果的な方法だと考えています[注2]。

本連載では、AWS APIの理解を深めることを目的に「APIに直接アクセス」する方法を中心に解説し、補助的にAWS CLIを利用していきます。また、APIへのリクエストとそのレスポンスを防護するために、APIへのアクセスにはHTTPS通信を利用することにします。

クライアントマシン環境の準備と確認

まず、APIに直接アクセスしてリクエストを行ううえで必要な、クライアントマシン環境を準備します。ここでは、Mac OS X Mavericks上でのクライアントマシン環境の構築について解説していきます。

インターネットコネクティビティ

最初に、当然なのですがAWS APIに接続するためにはインターネットにつながっていなければなりません。ブラウザでAWSの公式ページ(http://aws.amazon.com/jp/)にアクセスできていれば問題ありません。

デジタル署名ツール

AWS APIでは、リクエスト時にリクエストメッセージにデジタル署名を添付します。本連載では、主要ディストリビューションに標準で含まれているopensslコマンドでデジタル署名

注意

AWSに限りませんが、APIに直接アクセスする場合は要求メッセージのサイズや単位時間あたりの送信量が過大にならないように注意してください。たとえば、Amazon Elastic Compute Cloud(Amazon EC2)においては、AWSアカウントごとに「クエリAPIリクエスト率」による制限があることが明記されています[注3]。

注1) URL http://aws.amazon.com/jp/cli/
注2) これ以外にも、統合開発環境(IDE)向けのツールキットやサードパーティによるコマンドラインツールなども提供されています(https://aws.amazon.com/jp/tools/)。
注3) URL http://docs.aws.amazon.com/ja_jp/AWSEC2/latest/UserGuide/using-query-api.htm

を行います。

Mac OS X Mervericksに標準で組み込まれているopensslコマンド(/usr/bin/openssl)は0.9系と古いため、本稿では最新の1.0系バージョンのものをインストールして利用します。MacPortsもしくはHomebrewで何らかのパッケージをすでにインストールしている場合は、依存関係に応じてすでに最新のopensslコマンドが入っていることがあります。ここでは、MacPortsを利用した場合の、opensslコマンドのインストール先とインストールコマンドを例示します。

・MacPortsの場合
インストール先:

```
/opt/local/bin/openssl
```

・インストールコマンド:

```
$ sudo /opt/local/bin/port install openssl
```

HTTPSクライアント

AWS APIにリクエストを送信するためにHTTPSに対応したクライアントソフト(HTTPSクライアントとして利用可能なコマンドもしくはWebブラウザ)が必要です。

コマンド

Mac OS Xなどのいわゆる UNIX系OSには、HTTPS通信が可能なコマンドがいくつか存在します。本連載では、前述のデジタル署名で利用するopensslコマンドを、おもに手動でのHTTPS通信を説明するときのHTTPSクライアントとして利用します。必要に応じてほかのHTTPS対応のコマンドにも言及する予定です。

Webブラウザ

前回の「AWS APIの全体像」で説明したとおり、AWSの一部のサービスではREST形式のAPIが提供されています。このREST APIをWebブラウザから直接利用するためには、そのWebブラウザがGET/POST/PUT/DELETEの4種類のHTTPメソッドを適切に扱える必要があります。しかし、Firefox/Safari/Chromeなどの主要Webブラウザは、2014年3月現在PUTメソッドとDELETEメソッドには対応していません。WebブラウザでAWS APIにアクセスしたい場合は、Google Chromeとその拡張であるREST Consoleの利用を検討してください[注4]。

・Chrome:

```
https://www.google.com/intl/ja/chrome/browser/
```

・REST Console:

```
https://chrome.google.com/webstore/detail/rest-console/cokgbflfommojglbmbpenpphppikmonn
```

AWS CLI

AWS CLIはPythonのパッケージとして提供されているため、pipコマンド経由でインストールします。

・インストールコマンド

```
$ sudo easy_install pip
$ sudo pip install awscli
```

/usr/local/bin/awsとしてコマンドがインストールされていますので、バージョンを確認してみましょう。

・バージョン確認コマンド

```
$ aws --version
```

注4) AWSマネジメントコンソール利用時にも、基本的にGoogle Chromeを利用したほうが良いようです。筆者はSafariを使っていて、AutoScalingで作成したリソースが(いくらリロードしても)CloudWatchから見えず悩んだ経験があるのですが、Google Chrome上でリロードすれば見えるという助言に助けられました。

・バージョン確認結果

```
aws-cli/1.3.1 Python/2.7.6 Darwin/13.1.0
```

その他のコマンド

一般的なUNIXコマンドや上記コマンドの他に、エンコードを行う際に必要となるコマンド(base64、odなど)があります。これらは必要なタイミングで個別に紹介する予定です。

WebAPI検証環境の構築

AWS APIは、リクエストを処理するうえで何らかの問題があった場合、HTTPステータスコードとXML形式のレスポンスデータでエラーを返します。リクエストのボディ部分(クエリデータなど)が不正な場合は、AWS APIが返すレスポンスデータ内のエラーメッセージを参考にして対応すればよいのですが、リクエストヘッダが不正な場合は、APIにたどり着く手前のWebサーバが返す"400 Bad Request"などのHTTPステータスコードしか手掛かりがないため、エラー原因の切り分けが難しくなってしまいます。

本番のAWS APIを直接叩く場合は、事前に手元の環境でリクエストが正常な("200 OK"が返ってくる)形式であるか確認することをお勧めします。HTTPリクエストとして正常であることがあらかじめ確認できていれば、あとはAWS APIの仕様との格闘に集中できるわけです。

本稿ではこの確認環境として、Mac OS X Mavericksに標準でインストールされているApacheを利用して、ローカルマシン上にHTTPS対応のWebサーバを構築し、PHPによる簡易WebAPIを実装します。具体的には次の作業を行います。

・SSL証明書の作成
・Apacheのセットアップ(HTTPS、PHP)
・簡易WebAPIの作成(PHPスクリプト)

SSL証明書の作成

まず、ローカルマシン上のWebサーバをHTTPS対応にするために、SSL証明書を作成します。今回は、ローカルマシン上のユーザしかWebAPI検証環境にアクセスしない前提のため、自己証明の証明書を作成します。

1.秘密鍵の作成

秘密鍵の作成にはopensslコマンドを使います。

```
% cd /private/etc/apache2/
% sudo openssl genrsa -des3 -out server.key 2048
```

次にパスフレーズの入力が求められますので任意の文字列を入力します(あとで使うので忘れないでください)。

```
Enter pass phrase for server.key:
Verifying - Enter pass phrase for server.key:
```

秘密鍵ファイル(ここではserver.key)が生成されます。

2.パスフレーズなしの秘密鍵の作成

SSLが有効となっているApacheは、秘密鍵のパスフレーズを入力しないとプロセスの起動ができません。外部に公開されているWebサーバではセキュリティを考慮するうえで当然の動作なのですが、今回は、停止および起動が頻繁に発生することが多いクライアントマシンを利用すること、ローカル環境でのWebAPI検証環境としてのみ利用することの2点を前提に、パスフレーズありの秘密鍵はバックアップしておき、別途パスフレーズなしの秘密鍵を作成して利用することとします。

次のように秘密鍵ファイルをバックアップし、パスフレーズのない秘密鍵ファイル(ここでは

server.key)を作成します。

```
% sudo mv server.key server.key.org
% sudo openssl rsa -in server.key.org ↵
-out server.key
Enter pass phrase for server.key.org: ↵
(秘密鍵のパスフレーズを入力する)
```

3.CSR(Certificate Signing Request)の作成

次に、サーバ証明書を発行するうえで必要な署名要求(CSR)を作成します。

```
% sudo openssl req -new -key server.key ↵
-out server.csr
入力例
Country Name (2 letter code) [AU]:      ↵
JP
State or Province Name (full name) ↵
[Some-State]:           Tokyo
Locality Name (eg, city) []:            ↵
Shinjuku-ku
Organization Name (eg, company) ↵
[Internet Widgits Pty Ltd]: Sample
Organizational Unit Name (eg, section) ↵
[]:                 System
Common Name (e.g. server FQDN or YOUR ↵
name) []:           example.jp
Email Address []: 入力不要
A challenge password []:    入力不要
An optional company name [] 入力不要
```

これでCSRファイル(ここにはserver.csr)が生成されます。

4.SSLサーバ証明書の作成

CSRファイルと秘密鍵を利用して、SSLサーバ証明書を作成します。1年間有効の証明書を作成する場合、次のコマンドでSSLサーバ証明書ファイル(ここではserver.crt)が生成されます。

```
% sudo openssl x509 -req -days 365 -in ↵
server.csr -signkey server.key -out ↵
server.crt
```

Apacheのセットアップ

1. 設定ファイルの修正

次にApacheの設定をします。ポイントは3点です。

・Apacheの基本設定(ServerName)
・HTTPSの設定
・PHP(CGI)の設定

/private/etc/apache2/httpd.conf ファイル(**リスト1**)の編集をします。設定ファイルを修正したら、必ずバリデーションチェックをします。

```
% apachectl configtest
Syntax OK
```

結果が"Syntax OK"であれば、Apacheの設定は完了です。

▼リスト1　httpd.confファイル

```
#118行目付近(コメントをはずす)
LoadModule php5_module libexec/apache2/libphp5.so
#163行目付近(コメントをはずし、ホスト名の修正をする)
ServerName localhost:80
#181行目付近(Option行に"ExecCGI"を追加。Allow行を追加)
Options FollowSymLinks ExecCGI
Allow from localhost
#389行目付近(行を追加)
AddType application/x-httpd-php .php
#400行目付近(コメントをはずす)
AddHandler cgi-script .cgi
#492行目付近(コメントをはずす)
Include /private/etc/apache2/extra/httpd-ssl.conf
```

2. Apacheの起動

次のコマンドでApacheを起動します。

```
% sudo apachectl start
```

https://localhost/index.html にアクセスして、"It works!"と表示されれば、Apacheの起動およびHTTPS対応は成功しています。

3. PHPの動作確認

Apacheが起動したら、PHPスクリプトの動作確認のためのコンテンツを作成します。

```
% sudo /bin/sh -c "echo '<?php
phpinfo();' > /Library/WebServer/
Documents/phpinfo.php"
```

https://localhost/phpinfo.phpにアクセスして、PHPのバージョンやモジュール情報の一覧が表示されればPHP対応は成功しています。

簡易WebAPIの作成

ApacheのDocument Root下に簡易WebAPIのスクリプトを設置します(**リスト2**)。

簡易WebAPIの動作確認

スクリプトの設置が完了したら、実際に簡易WebAPIにリクエストを投げてみましょう。HTTPS経由でアクセスするため、opensslコマンドでlocalhostのTCP443番ポートにアクセスします。

```
% openssl s_client -connect localhost:443
```

▼リスト2　/Library/WebServer/Documents/request.php

```
<?php
echo "<!DOCTYPE html>";
echo "<html lang=\"ja\"><head><meta charset=\"utf-8\">
<title>request checker</title></head><body>";
// method
echo "<h1>Method: " . $_SERVER['REQUEST_METHOD'] . "</h1>";
// HTTP header
echo "<h2>Header</h2>";
$headers = getallheaders();
while (list ($key, $value) = each ($headers)) { echo "$key:
$value<br>"; }
echo "<hr>";
// HTTP body
echo "<h2>body (POST/PUT/DELETE)</h2>";
$body = file_get_contents("php://input");
var_dump($body);
echo "<h2>body (GET)</h2>";
$query = $_SERVER["QUERY_STRING"];
var_dump($query);
echo "</body></html>";
```

コマンドを実行すると、数十行のメッセージが流れて、最後に

```
……(省略)……
Timeout   : 300 (sec)
Verify return code: 18 (self signed
certificate)
```

と表示されてコマンドが待機状態になります。ここにリクエスト文を入力すると、Webサーバにリクエストが送信されますが、入力ミスやタイムアウトを避けるためにクリップボード経由で貼り付けるのが良いでしょう。

まず、リクエストが正常な場合の動作の確認をします。ここでは**リスト3**のGETリクエストを送信します。

リクエストの最後に空行を送信すると、実際にWebサーバにリクエストが送信され、すぐに次のようにサーバからのレスポンスが表示されます(**図1**)。

次に、リクエストが不正な場合の動作確認と原因の確認をしてみましょう。たとえば、HTTP/1.1ではHost行を忘れると、"400 Bad Request"が返ってきます(**図2**)。

ご覧のとおり「あなたのブラウザはサーバが理解できないリクエストを送りました」という情報しかないため、これだけではエラーの原因を特定するのは難しいと思われます。

そのときはApacheのエラーログ(OSX標準では/var/log/apache2/error_log)を見てみましょう(**図3**)。

ログの記述内容から、エラーの原因がホスト名の指定漏れによるものだとわかるわけです。

次に、GET以外のメソッド(POST/PUT/DELETE)

のサンプルリクエストを例示します(**リスト4〜6**)。ぜひ実際に簡易WebAPI環境で実行してみてください。この環境は、AWS APIに限らず各種WebAPIを検証するときにも便利に使えると思いますので活用ください。

次回は、実際にAWSを触るために必要なAWSアカウントの作成、AWSマネジメントコンソールでの初期設定について解説します。SD

▼図1　実行結果

```
HTTP/1.1 200 OK
  ……(省略)……
<!DOCTYPE html><html lang="ja"><head><meta
charset="utf-8"><title>request checker</
title></head><body><h1>Method: GET</
h1><h2>Header</h2>Host:
localhost<br><hr/><h2>body (POST/PUT/
DELETE)</h2>string(0) ""
<h2>body (GET)</h2>string(47) "AWSAccessK
eyId=AKIAIOSFODNN7EXAMPLE&Action=Test"
</body></html>
```

▼図2　エラーレスポンス(例)

```
HTTP/1.1 400 Bad Request
  ……(省略)……
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML
2.0//EN">
<html><head>
<title>400 Bad Request</title>
</head><body>
<h1>Bad Request</h1>
<p>Your browser sent a request that this
server could not understand.<br />
</p>
</body></html>
```

▼図3　エラーログ(例)

```
[Wed Mar 12 16:34:59 2014] [error] [client
127.0.0.1] client sent HTTP/1.1 request
without hostname (see RFC2616 section
14.23): /request.php
```

▼リスト3　サンプルリクエスト(GETメソッド)

```
GET /request.php?AWSAccessKeyId=AKIAIOSFODNN7EXAMPLE&Action=Test HTTP/1.1
Host: localhost
 空行
```

▼リスト4　サンプルリクエスト(POSTメソッド)

```
POST /request.php HTTP/1.1
Host: localhost
Content-Type: application/x-www-form-urlencoded
Content-Length: 47
 空行
AWSAccessKeyId=AKIAIOSFODNN7EXAMPLE&Action=Test
```

▼リスト5　サンプルリクエスト(PUTメソッド)

```
PUT /request.php HTTP/1.1
Host: localhost
Content-Type: application/x-www-form-urlencoded
Content-Length: 47
 空行
AWSAccessKeyId=AKIAIOSFODNN7EXAMPLE&Action=Test
```

▼リスト6　サンプルリクエスト(DELETEメソッド)

```
DELETE /request.php HTTP/1.1
Host: localhost
Content-Type: application/x-www-form-urlencoded
Content-Length: 47
 空行
AWSAccessKeyId=AKIAIOSFODNN7EXAMPLE&Action=Test
```

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）

2014年2月号

第1特集
λ式からはじめませんか?
関数型プログラミング再入門

第2特集
目利きによるトレンド予測
2014年IT業界はどうなるのか?

一般記事
・会社組織を活性化するスパイス「コンパ」

定価（本体1,219円+税）

2014年1月号

第1特集
シェルがわかればシステムがわかる
あなたの好きなシェルは何ですか?

第2特集
未来を作るITインフラ
10ギガビットを実現するケーブリング技術

特別付録 & 一般記事
・法輪寺鎮守社電電宮 情報安全護符シール Ver.2
・ソーシャルゲームのDevOpsを支える技術（後編）

特別定価（本体1,314円+税）

2013年12月号

第1特集
SDN/OpenFlowの流れを総まとめ!
SDN/OpenFlowで幸せになれますか?

第2特集
下手でも好印象で効果絶大
エンジニアの伝わる図解術

一般記事
・LinuxとFreeBSDのファイルシステムの良い・悪いところをご存じですか?
・「Mirama」ほか

定価（本体1,219円+税）

2013年11月号

第1特集
思考をコード化する道具
我が友 Emacs

第2特集
コンピュータの加速装置
サーバサイドフラッシュ Fusion-io全方位解説

一般記事
・小規模プロジェクト現場から学ぶJenkins活用（5）
・ソーシャルゲームのDevOpsを支える技術（前編）ほか

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。

分散データベース「未来工房」最終回

第11回 Riakはなぜデータをなくさないのか(3)

風流の初やおくの田植うた

Writer 上西 康太(うえにし こうた) Basho ジャパン株式会社 Twitter@kuenishi

ここまで、2回にわたって、Riakではなぜデータがなくならないかということを、分散データベースにまつわる整合性と永続化の問題と一緒に解説した。Riakがそのような問題にどう対処しているか、これまで解説してきたヒント付きハンドオフ、Siblings、ベクタークロック、に加えて、本稿ではアクティブアンチエントロピーについて解説する。

注)本稿は、筆者の意向により常体で表記している。

レプリカの不一致または不足が起きるケース

Riakでは、レプリカ(複製)を3個作成する。しかし、レイテンシを重視するため、各ノードへ転送した書き込みリクエストのAckはすべてを待たない場合がある。たとえばN=3、W=2であれば、3つ目のレプリカの書き込みが失敗してしまった場合、このままではデータの複製の数が2のままシステムが稼働し続けてしまう。

ほかにも、レプリカの数が既定よりも少なくなるケースはいくつもある。データの書き込み時にはネットワークの瞬断、タイムアウトなどが考えられる。また、レプリカを3つ作成した後に、ディスク故障やノード故障により、レプリカ数が既定より少なくなる場合もある。

また、Quorumによるレプリケーションで、1つだけ書き込みに成功し、残りの2つの書き込みに失敗した場合、クライアントには失敗が返るが実際には新しいデータが1つだけ残っている場合がある。

このように、ノードやネットワークが不安定になると、すべてのレプリカを同じ値に保つことができなくなる。これまで2回にわたって説明してきたように、いつでも書き込める高い可用性を実現するために、これは必要な設計である。しかしながら、それだけではレプリカはハードウェアの故障が起きるたびに減っていき、最終的にはなくなってしまうだろう。それを防ぐ手段として、リードリペアと、アクティブアンチエントロピー(以降、AAEと呼ぶ)というレプリカの修復機構がRiakには用意されている。

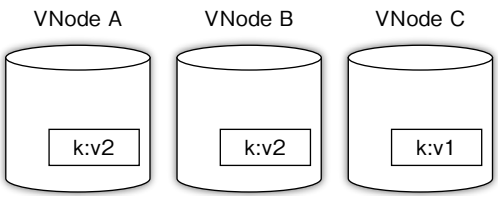
リードリペアによるレプリカの復元

レプリカの不整合や不足を補償する手段の1つとして、Riakにはリードリペアというしくみが組み込まれている。これは、データを読み出した際に、異なるレプリカが見つかると、最新の値で書き戻すというしくみだ。

たとえば、あるキーに対して、**図1**のように、レプリカCだけが古い値を持っていたものとする。おそらくは、v2を書き込んだときにレプリカCを持ったノードが落ちていて、そのままv2をCに書き込めないままタイムアウトしたのだろう。

このように古いデータが残ってしまうような

▼図1 レプリカが不整合な状態
vnode A、Bは最新の値v2を保持しているが、vnode Cは古い値v1しか持っていない。



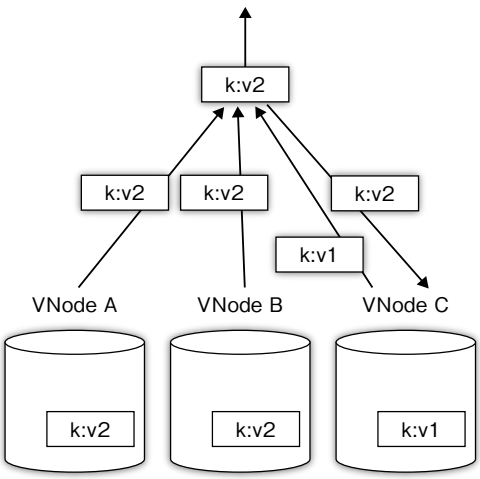
不整合状態を防ぐ方法として、他には、Cがクラスタに戻って書き込みリクエストに返答するまで、レプリカの書き込みをリトライし続けるという設計案もある。しかし、Riakはそういった設計にはしていない。たとえばCが復帰するのは1年後の場合もあったり、そのまま故障して復帰できない場合もある。復帰するかどうかもわからないレプリカに対してリトライをし続けるのは現実的ではないから、このような設計にはなっていない。

Riakでは、その代わりとして、このデータが読まれた際にv2をCに書き戻すことによって**図2**のように、Cも最新の値で上書きしてしまう。これによって、最終的に**図3**のようにレプリカを整合した状態に戻すことができる。

このとき、どちらのレプリカが新しいかを判定することは、ただのキーと値のペアだけでは不可能だ。しかし、Riakでは前回解説したように、すべてのキーにはベクタークロックが割り当てられ、どちらが新しいかというデータの因果関係がわかるようになっている。前回の解説で述べたように、この「新しい」という言葉は、単にタイムスタンプの値が大きいという意味ではないことに注意していただきたい。ここではv1、v2と便宜的に表記したが、実体はベクタークロックである。

もちろんベクタークロックなので、v1とv2は因果関係を定義できない組み合わせになっている場合もある。この場合は、Sibling同士をマージして、v1＜v3かつv2＜v3となる新しい値v3を作成し、それを用いて上書きを行う。

▼図2 リードリペアの動作
GETのリクエストに対してvnode A、Bは最新の値v2を返すが、vnode Cは古い値v1を返す。このとき、クライアントに対してはv2を返すが、非同期にv2をvnode Cに書き戻す。

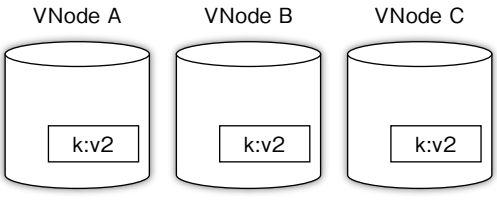


リードリペアの上書きを確かめてみよう

それでは、ネットワークが分断したような状況でも、実際にリードリペアによってデータが正しく修復され、マージされていることを確かめてみよう。前回と同様、3台でのRiakのクラスタのセットアップをしているものとする。手順は次のとおりである。

```
$ git clone git://github.com/basho/riak
$ cd riak
$ git checkout riak-2.0.0pre15
$ make devrel
$ dev/dev1/bin/riak start
$ dev/dev2/bin/riak start
$ dev/dev3/bin/riak start
$ dev/dev2/bin/riak-admin cluster \
  join dev1@127.0.0.1
$ dev/dev3/bin/riak-admin cluster \
  join dev1@127.0.0.1
$ dev/dev1/bin/riak-admin cluster plan
$ dev/dev1/bin/riak-admin cluster commit
$ dev/dev1/bin/riak-admin transfers
$ curl -i -X PUT \
  http://localhost:10018/buckets/b/➥
props \
  -H 'content-type:application/json' \
  -d '{"props":{"allow_mult":true}}'
```

▼図3 レプリカが整合した状態
vnode A、B、Cが最新の値v2を保持している。



ネットワーク分断が起きて、ノード1だけの世界と、ノード2、3だけの世界に分かれてしまったとしよう。ノード1とほかのノードが一切通信できないことを、起動時間をずらして表現する。まずは**riak stop**によってノード2、3を停止させた状態で、**図4**のようにノード1だけを起動してデータを入れよう。

次に、ノード1を停止し、ノード2、3を起動する(**図5**)。

これによって、ノード2、3に'r'というデータが記録された。最後に、ノード1を起動することで、擬似的に実現していたネットワーク分断を元に戻して、実際にどのようなベクタークロックになるかを確認しよう。おそらく、ノード1が起動してから、実際にこのキーの値が変わるのには何度かリクエストを繰り返す必要があるだろう。

はたして、**図6**では、どのようになっただろうか。末尾の4文字に注目すると、**図4**では、**...wxfFgA=**、**図5**では**...hm+LAA=**でそれぞれ終わっていたベクタークロックが**図6**では**...FQWAA==**で終わる異なる値になっており、また長さも長くなったことがおわかりいただけるだろうか。また、実際に手元で試したとき**図6**では、2つのSiblingもできているだろう。このように、リードリペアによって、ネットワーク分断によって異なる値が保存されてしまったケースでも、一致していないレプリカを一致させることができる。

このあと、ノード1を停止してからノード2にGETをしても、ノード2、3を停止してノード1にGETをしても同じ値がとれるようになっている。ぜひ確認していただきたい。

▼図4　ノード1にデータを記録する

```
# dev1/bin/riak start
$ curl -X PUT http://localhost:10018/buckets/b/keys/k -d 'l'
$ curl -i http://localhost:10018/buckets/b/keys/k
HTTP/1.1 200 OK
X-Riak-Vclock: a85hYGBgzGDKBVIcR4M2cgcrf56WwZTImMfK0Oo16wxfFgA=
```

▼図5　ノード2、3にデータを記録する

```
# dev1/bin/riak stop
# dev2/bin/riak start
# dev3/bin/riak start
$ curl -X PUT http://localhost:10028/buckets/b/keys/k -d 'r'
$ curl -i http://localhost:10028/buckets/b/keys/k
HTTP/1.1 200 OK
X-Riak-Vclock: a85hYGBgzGDKBVIcMRuucwUrf56TwZTImMfK8Nhr1hm+LAA=
```

▼図6　ネットワーク分断が修復された後のリードリペアを確認する

```
# dev1/bin/riak start
# curl -i http://localhost:10018/buckets/b/keys/k
…… (中略) ……
# curl -i http://localhost:10018/buckets/b/keys/k
HTTP/1.1 300 Multiple Choices
X-Riak-Vclock: a85hYGBgymDKBVIcMRuucwUrf56TwZTImMfK8Nhr1hk+qNTRoI3cQKlpUKlWkFQWAA==
```

リードリペアを用いた故障時のオペレーション

では、実際に複数のvnodeを保持しているRiakノードのディスクが故障したケースを考えてみよう。ディスク交換後に、新しいディスクと共にクリーンになって戻ってきたマシンのIPアドレスとRiakのノード名は同じと仮定する[注1]。単に起動しただけではRiakはもとのクラスタには復帰しないので、クラスタに対するjoin/plan/commitを次のように行う必要がある。

```
# riak-admin cluster join riak@➥
<クラスタ内の別ノードのアドレス>
# riak-admin cluster plan
# riak-admin cluster commit
```

これで、Riakはクラスタに参加したことになる。**riak-admin transfers**の様子を見ても、故障前に持っていたデータは復旧していないことがわかるだろう。これを復旧させるためにリードリペアを使うことができる。たとえば、故障していたノードにあるvnodeをBとして、Riakのリング上で前後にあるvnodeをそれぞれA、Cとして説明する。このとき、リング上でA、B、Cと連続しているので、Bが持つべきデータはすべてA、C上にコピーがなければならない。そこでA、Cが持っているすべてのキーに対してGETを実行することによってリードリペアを強制的に発動する。このときBはデータを持ってないために、常にデータがないと返すが、実際には他のレプリカからデータがあることがわかり、リードリペアによって、Bに対する書き戻しが行われる。

これが完了すると、リードリペアによってBが持つべき全てのデータが復元されたことになる。しかし、故障や瞬断が起きる度にこういった操作を手動で実行したり、データの完全性を守るために外部から定期的にこのような操作を行うことはRiakの運用性を大きく下げることになる。実際に、過去には上記のようなリードリペアのオペレーションを行っていた。

Riakのあるバージョンからは、前後のvnode A、Cからデータの該当範囲をまるごとコピーしてくるという機能が追加され随分簡単になった。しかし、やはり手動での操作には運用上の難しさがある場合も多く、Riakの1.3からは、アクティブアンチエントロピー(AAE)という仕組みでこれを自動的に行うようにした。

もちろんこのようなリペアのオペレーションは、いまでもAAEを使わない場合の代替策として残っており、折にふれて役立っている。最新のRiakでは、動作しているノードに次のようにattachして実行できるので、ぜひ試してもらいたい。

```
# riak attach
> {ok,Ring} = riak_core_ring_manager➥
:get_my_ring().
> MyPart = riak_core_ring:my_➥
indices(Ring).
> [riak_kv_vnode:repair(Part) || Part ➥
<- MyPart].
```

アクティブアンチエントロピー(AAE)

リードリペアを故障時または定期的にマニュアルで行うことは、やはり運用のコストを上げることになる。そこでRiak 1.3からは、アクティブアンチエントロピー(AAE)という、レプリカの不整合を自動的に修復していく機構がRiakに組み込まれた。これによって、Riakは完全な結果整合性(Eventual Consistency)を備えたといってよいだろう。つまりアトミックブロードキャストを必要とするようなレプリカ作成のメカニズムを用いずに可用性を優先しつつ、最終的にレプリカが整合した状態を保証するた

注1) 異なるIPアドレスでも、**force-replace**というコマンドを使うと同様のことができるが、ここでは単純のため解説しない。

めのしくみである。

AAEのハッシュツリー

しかしながら、障害回復時のオペレーションのように、定期的にすべてのキーをスキャンしてGETしていくのではシステムに対する負荷が非常に大きい。そこで、AAEではvnodeごとのレプリカの不整合を検出するためにハッシュツリーを用いている。具体的には、レプリカに含まれるデータが更新されるごとにハッシュツリーを更新し、AAE動作時にはハッシュツリーをチェックすることで不整合が起きているキーを効率的に検出し、差分のあったデータだけリードリペアを動作させる。

AAEのしくみは非常に簡単で、①最初にvnodeごとにツリーをビルド、②データの更新があればAAEツリーを更新定期的にチェック、③ツリーをロックして同時に1つだけ交換、不整合が検出されたらリードリペアを実施、という流れになる。ほかにもDynamo論文に習ってハッシュツリーでのAAEを実装しているシステムがあるが、Riakではこのハッシュツリーはディスク上に永続化されているため、メモリを不必要に圧迫することがない、再起動をしてもツリーの再ビルドが発生しないなどの利点がある。

AAEの動作状態の確認

AAEの動作状況は`riak-admin aae-status`というコマンドで確認できる。ここでは、そのノードが保持しているvnodeごとに、

- 最後にハッシュツリーが最後に前後のvnodeと交換・比較されたのがいつか(Exchanges)
- 最後にハッシュツリーが構築されたのがいつか(Entropy Trees)
- 最新の修復で修復されたキーの数(Keys Repaired)

を見ることができる。

まとめ

ここまでの3回で、Riakがなぜデータをなくさないかというテーマについて解説してきた。

第1回では、可用性を重視しつつネットワークの分断耐性を持った設計にするためには、書き込み競合の扱いが非常に難しいことを解説した。暗黙の上書きを防ぐためには、アトミックブロードキャストプロトコルによるCASのような操作を提供するか、RiakのようにSiblingsで両方のデータを保持しておくかしかない。しかし、前者ではネットワーク分断時にRiakほどの可用性を得にくい場合があり、Riakでは運用を簡単にするためにもハンドオフというしくみを用意していることも解説した。

第2回では、Siblingsによって書き込み競合時も双方のデータを保持した上で、ベクタークロックを導入して異なるデータの間に、タイムスタンプとは異なる形で因果関係を導入した。これによって、暗黙の上書きが起きないデータの上書きを実現すること、ネットワーク分断と削除の組み合わせでも削除したはずのデータが復帰しないことも示した。

本稿では、第3回としてネットワーク以外の故障、具体的にはディスクの故障などによってレプリカが一時的に不整合となったり、レプリカの数が故障のたびに縮退していくことを防ぐ方法として、リードリペアとアクティブアンチエントロピー(AAE)の解説をした。これによって、アトミックブロードキャストがないシステムであっても最終的にレプリカが一致するという結果整合性を実現できる。

これで、Riakを使っている限り、データが絶対になくならないということを理解いただけただろうか。SD

Dで小飼弾

第5回

『断絶への航海』（James P. Hogan／ハヤカワ文庫SF）

「常識とは18歳までに君が積み重ねた偏見である」と言ったのはアインシュタイン――というのもまた我々が積み重ねた偏見の1つだという疑惑もあるのですが[注1]――それはさておき、もしそれが真実だとしたら、その偏見から自由になるためには、その偏見が存在しない環境で育つしかないことになります。そんなことは可能でしょうか？　そう、SFならね。

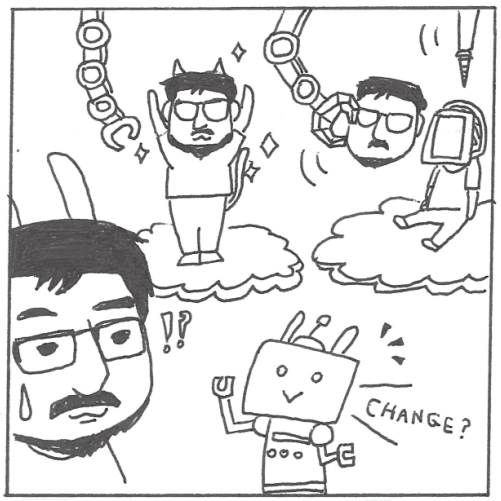
作品名としては、『断絶への航海』（James P. Hogan）ということになります。第三次世界大戦で3つに分かれて戦い、その結果疲弊しきった地球人たちは、一路アルファ・ケンタウリを目指します。大戦前に送り込まれたロボット移民船によって植民星を、ライバルたちより一足先に我が者にするために。しかしその結果、我が者となったのは誰か……。

ロボット移民船に乗船していたのは、凍結受精卵と子守りロボット。常識に染まった社会（人）は文字通りの意味で重たすぎて積載できなかったわけです。こうして地球の「偏見」と断絶された社会ができ上がったわけですが、それが「常識」と出会う時、一体何が起るのか……。

今は亡き作者James P. Hoganは、これまた今は亡きDECの元エンジニア。それもあってか彼の作品の構成は文芸作品というよりはサイト構築を彷彿とさせます。とくにそれが強いのは、サンドボックスの使い方。『未来の二つの顔』ではスペース・コロニーが、そして本作ではアルファ・ケンタウリの惑星ケイロンがそれにあたります。「いくら考えてもわからないなら、実際に作ってみればいいじゃん。でも現状が壊れないように隔離環境で」というのは、昔も今もエンジニアの常套手段。ライト兄弟が偉いのは、最初に有人動力飛行機を飛ばしたことよりも、風洞を作ったことにあると私は思っています。

AWSにAzureに（Open|Cloud）Stack……ましてやITの世界では、少なくとも「あちら側」に関しては仮想環境をいくつも作って、そのうち「一番いい」のをそのまま本番環境とするのも当たり前。本作の仮想環境は、現実世界にデプロイ可能でしょうか？　とりあえずはご一読を。気に入らなければポイできるのも仮想環境の良さなのですし。SD

注1）http://en.wikiquote.org/wiki/Albert_Einstein



題字・イラスト／aico

より速く、より莫大に、より高みへ！

サーバマシンの測り方

―― ベンチマークを極める実践テクニック ――

第6回［最終回］　続・HTTPベンチマークからネットワーク

前回はabを用いたベンチマークの方法を紹介しました。最終回の今回は、abよりも強力なHTTPのベンチマークツールを紹介し、ファイアウォールやロードバランサがあるネットワーク全体に対してベンチマークし、ボトルネックがどこになるかを見ていきます。

Writer ㈱IDCフロンティア　ソリューションアーキテクト　藤城拓哉（ふじしろたくや）／Twitter@tafujish

ネットワークボトルネックを見つけるには

ある程度のWebアクセスを見込んだネットワーク構成を考えます。ファイアウォールがあり、Webからのアクセスをロードバランサが捌くという、よくあるシステムです。このような構成のとき、ファイアウォールやロードバランサがボトルネックになることもあります。構成要素が増えるにつれ、ボトルネックの調査はたいへんになっていきます。そこで今回は、HTTPベンチマークを使用し、ネットワーク上のボトルネックを調べるプロセスを紹介します。

図1のネットワーク構成を例に挙げ、それぞれのマシンについて説明します。

ファイアウォールはVyattaCore 6.6R1[注1]で構築しました。ロードバランサはCentOS 6.5でDSR[注2]構成のLVS[注3]を構築しました。Webサーバは第5回で取り上げたものと同じCentOS 6.5上にNginxにて構築しロードバランサのバランシング先（リアルサーバ）に4台をラウンドロビンで設定しました。すべてのマシンが4コアのCPUを搭載し、1Gbpsのネットワークで接続しています。ファイアウォール以外の各マシンには、第5回で取り上げたチューニング設定を入れています[注4]。

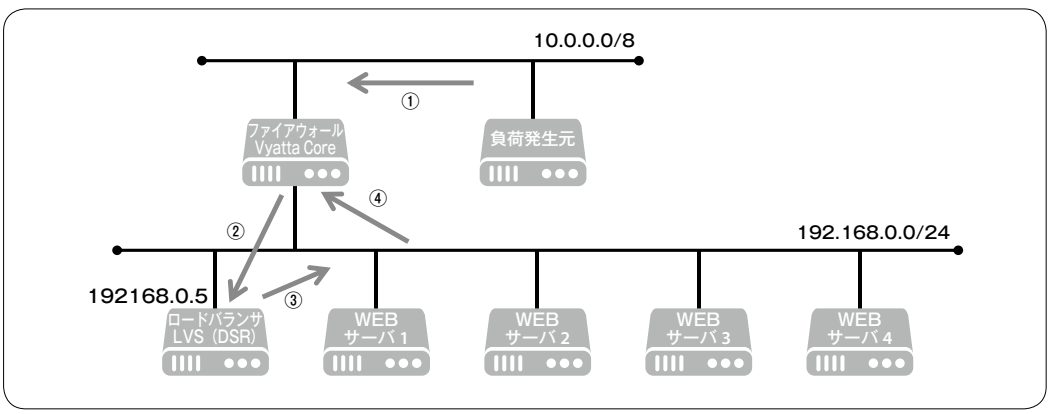
注1）URL http://www.vyatta.org/

注2）DSR（Direct Server Return）構成：ロードバランサがサーバにリクエストを投げ、受けたサーバがダイレクトにクライアントにレスポンスするしくみ。

注3）LVS（Linux Virtual Server）

注4）Nginxで「Too many open files」エラーが出ていたので、configに「worker_rlimit_nofile 50000;」を追記しています。

▼図1　ネットワーク構成



通信の流れとしては、次のとおりです。

①負荷発生元からabなどを実行し、ファイアウォールへ向かう(宛先IPアドレスはロードバランサでここでは192.168.0.5)
②ファイアウォールを経由したあと、ロードバランサへ向かう
③ロードバランサからいずれかのWebサーバへ転送される
④DSR構成のためWebサーバからゲートウェイであるファイアウォールに直接返す(返りはロードバランサを経由しない)

ネットワーク帯域がボトルネック

連載第5回でも触れましたが、ネットワークの負荷試験をするときはネットワーク帯域が一番ボトルネックになりやすくなります。負荷をかける先のコンテンツとして、22KBのテキストを設置しました。たとえば次のように負荷をかけます。

```
$ ab -c 1000 -n 100000 ↵
http://192.168.0.5/test22k.html
```

ab実行後、まずはTransfer rateの値を確認しましょう。値が125,000KB/sec(1Gbps)付近であれば1Gbpsの帯域がボトルネックになっているとわかります。今回の構成では、ファイアウォールのところで1Gbpsに達しています。ロードバランサがDSR構成のためロードバランサやWebサーバの帯域がボトルネックにはなりませんでした。負荷試験中に各マシンでトラフィックの値を確認するのも良いでしょう。

abがボトルネック

こちらも連載の第5回で触れたとおり、負荷をかける元がボトルネックとなり、十分な負荷をかけきれない場合もよくあります。では実際にその様子を見てみましょう。帯域がボトルネックにならないように2Byteのテキストを設置し、次のように負荷をかけました。

```
$ ab -c 1000 -n 10000000 ↵
http://192.168.0.5/test2b.html
```

負荷試験の最中は各マシンにてtopコマンドでCPU使用率を確認し、このときの様子が**図2**です。負荷発生元の1つのCPUコアでアイドルが0%すなわち使用率100%になっています。そして、ほかのマシンのCPU使用率はまだまだ余裕がある状態です。abは1つのCPUコアしか利用できないので、ここがボトルネックです。そして、これ以上負荷がかからない状態だとわかります。スループットとしては約57Mbpsでした。

強力なベンチマークツールを使う

abのようなHTTPのベンチマークツールは、ほかにもたくさんあります。abはCPUが1コ

▼図2　ab実行時のCPU使用率

負荷発生元（ab）

```
Cpu0  :  0.0%us,  0.0%sy,  0.0%ni,100.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu1  :  0.0%us,  0.0%sy,  0.0%ni,100.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu2  :  7.9%us, 65.3%sy,  0.0%ni,  0.0%id,  0.0%wa,  0.0%hi, 26.7%si,  0.0%st
Cpu3  :  0.0%us,  0.0%sy,  0.0%ni,100.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
```

ファイアウォール

```
Cpu0  :  0.0%us,  0.0%sy,  0.0%ni,100.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu1  :  0.0%us,  0.0%sy,  0.0%ni,100.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu2  :  0.0%us,  1.0%sy,  0.0%ni, 99.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu3  :  0.0%us,  0.0%sy,  0.0%ni,100.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
```

ロードバランサ

```
Cpu0  :  0.0%us,  0.0%sy,  0.0%ni,100.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu1  :  0.0%us,  0.0%sy,  0.0%ni, 98.8%id,  0.0%wa,  0.0%hi,  1.2%si,  0.0%st
Cpu2  :  0.0%us,  0.0%sy,  0.0%ni, 98.9%id,  0.0%wa,  0.0%hi,  1.1%si,  0.0%st
Cpu3  :  0.0%us,  0.0%sy,  0.0%ni, 98.9%id,  0.0%wa,  0.0%hi,  1.1%si,  0.0%st
```

WEB サーバ 1

```
Cpu0  :  8.2%us, 19.6%sy,  0.0%ni, 32.0%id,  0.0%wa,  2.1%hi, 38.1%si,  0.0%st
Cpu1  :  0.0%us,  1.0%sy,  0.0%ni, 92.8%id,  0.0%wa,  0.0%hi,  6.2%si,  0.0%st
Cpu2  :  0.0%us,  1.0%sy,  0.0%ni, 90.9%id,  0.0%wa,  0.0%hi,  8.1%si,  0.0%st
Cpu3  :  0.0%us,  0.0%sy,  0.0%ni, 94.8%id,  0.0%wa,  0.0%hi,  5.2%si,  0.0%st
```

アしか利用できないためパワー不足となりましたが、マルチスレッドに対応したツールもあります。筆者の知る限りこれが定番と言えるまでのツールはありませんが、よく名前を聞くツールとしてweighttp[注5]とwrk[注6]があります。今回のネットワーク構成に対して、先述の負荷試験と同じく**-c 1000**としたとき、wrkのほうがより負荷をかけられたのでここではwrkを紹介します。

wrkの使い方

wrkは**図3**のようにインストールします。ソースをGitHubから取得しビルドします。基本的にabと同じ操作感で使えます。**-c**は同じくconcurrency(同時接続数)を指定しURLを書きます。abと異なる部分として、abでは**-n**でリクエスト総数を指定していましたが、wrkでは**-d**にて負荷をかけ続ける時間(秒)を指定しこの間実行され続けます。また、マルチスレッドに対応しているので**-t**にて指定します。実行するマシンのCPUのコア数と同じ値にするので、今回のマシンが4コアのため**-t 4**としました。

実行結果もabと同様、秒間のリクエスト処理数(Requests/sec)や秒間の転送量(Transfer/sec)が得られるのでこの値を結果とします。

abでは31827.26req/sec(6.98MB/sec)だったのが、wrkでは143664.51req/sec(32.19MB/sec)と4倍以上の結果となりました。

補足：concurrencyを大きく

abではconcurrency(-c)の値は20,000までしか指定できませんでしたが、wrkやweighttpではこれ以上の値を指定可能です。ただし、CentOS 6系のデフォルトでは、ソースポートの範囲が32768～61000(28232ポート)までとなっておりこの範囲を超えてconcurrencyの値を設定できません。この範囲を最大限大きくするには、**/etc/sysctl.conf**に

```
net.ipv4.ip_local_port_range = 1024 65535
```

と記述し、**sysctl -p**で反映させることでconcurrencyの値を60,000以上まで扱えるようになります。

結局どこがボトルネックになったか

図3のwrkを実行したときの各マシンのCPU使用率が**図4**です。ファイアウォールの2つのコアの使用率が大きく、ネットワーク処理の負荷によりこのファイアウォールの限界近くまできていることがわかります。一方で、wrk側の1つのコアでもCPU使用率が大きいです。si(ソフトウェア割り込み)の値が1つのコアだけ大きいことから、ネットワーク処理により1つのコアの使用率が大きくなっているとわかります。

注5) URL https://github.com/lighttpd/weighttp
注6) URL https://github.com/wg/wrk

▼図3　wrkのインストールと実行方法

```
$ sudo yum -y install gcc openssl-devel
$ git clone https://github.com/wg/wrk.git
$ cd wrk/
$ make
$ ./wrk -t 4 -c 1000 -d 300  http://192.168.0.5/test2b.html
Running 5m test @ http://192.168.0.5/test2b.html
  4 threads and 1000 connections
  Thread Stats   Avg      Stdev     Max   +/- Stdev
    Latency     6.73ms   32.15ms   2.68s    97.23%
    Req/Sec    36.19k     7.25k   66.65k    68.99%
  43099022 requests in 5.00m, 9.43GB read
  Socket errors: connect 0, read 0, write 0, timeout 165
Requests/sec: 143664.51
Transfer/sec:     32.19MB
```

ファイアウォールとwrk側のどちらがボトルネックか確認するためには、wrkの実行結果でエラーが出ていないか確認します。**図3**の結果のとおり、timeoutエラーが出ており、これはネットワーク上で処理しきれなくなったかWebサーバが応答できなかったということになります。Webサーバ側のCPU使用率に余裕があり、WebサーバのmessagesやnginxにエラーログがなければWebサーバ側ではなくファイアウォールがボトルネックになったとわかります。

おわりに

今回はwrkを用いてネットワーク全体に対して負荷試験をしました。測定したい対象とボトルネックのポイントをきちんと見極めてベンチマークしないと本当に必要な値は取得できないことがわかっていただけましたでしょうか。

本誌2013年7月号のベンチマーク入門と本連載の内容をご覧いただき、サーバや各パーツ自体の性能測定からシステム全体のネットワーク性能測定まで一通りできるよう参考になっていれば幸いです。最後にこれまで紹介したツールを、おすすめ度というか筆者が実際に業務で活用している度合に応じて星取り表でまとめました(**表1**)。半年間ありがとうございました。SD

▼図4　wrk実行時のCPU使用率

負荷発生元（wrk）

```
Cpu0  :  8.2%us, 28.6%sy,  0.0%ni,  2.0%id,  0.0%wa,  2.0%hi, 59.2%si,  0.0%st
Cpu1  :  9.2%us, 28.6%sy,  0.0%ni, 44.9%id,  0.0%wa,  0.0%hi, 17.3%si,  0.0%st
Cpu2  :  8.5%us, 26.4%sy,  0.0%ni, 46.2%id,  0.0%wa,  0.0%hi, 18.9%si,  0.0%st
Cpu3  :  6.8%us, 23.3%sy,  0.0%ni, 57.3%id,  0.0%wa,  0.0%hi, 12.6%si,  0.0%st
```

ファイアウォール

```
Cpu0  :  0.0%us,  0.0%sy,  0.0%ni,  5.9%id,  0.0%wa,  0.0%hi, 94.1%si,  0.0%st
Cpu1  :  0.0%us,  0.0%sy,  0.0%ni,  9.8%id,  0.0%wa,  0.0%hi, 90.2%si,  0.0%st
Cpu2  :  0.0%us,  1.0%sy,  0.0%ni, 99.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu3  :  0.0%us,  0.0%sy,  0.0%ni,100.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
```

ロードバランサ

```
Cpu0  :  0.0%us,  0.0%sy,  0.0%ni, 96.7%id,  0.0%wa,  0.0%hi,  3.3%si,  0.0%st
Cpu1  :  0.0%us,  0.0%sy,  0.0%ni,100.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu2  :  0.0%us,  0.0%sy,  0.0%ni, 97.4%id,  0.0%wa,  0.0%hi,  2.6%si,  0.0%st
Cpu3  :  0.0%us,  1.1%sy,  0.0%ni, 97.8%id,  0.0%wa,  0.0%hi,  1.1%si,  0.0%st
```

WEB サーバ 1

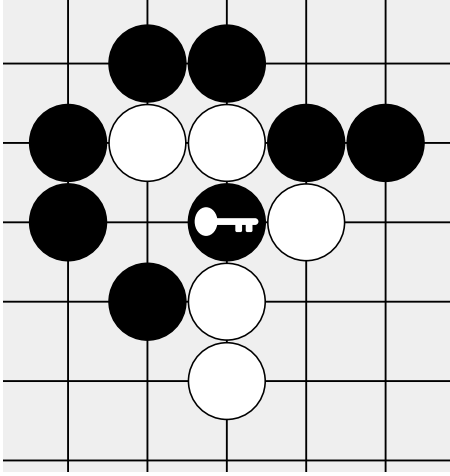
```
Cpu0  : 13.0%us, 29.0%sy,  0.0%ni, 21.0%id,  0.0%wa,  5.0%hi, 32.0%si,  0.0%st
Cpu1  : 10.0%us, 18.2%sy,  0.0%ni, 56.4%id,  0.0%wa,  0.0%hi, 15.5%si,  0.0%st
Cpu2  :  8.6%us, 16.2%sy,  0.0%ni, 61.9%id,  0.0%wa,  0.0%hi, 13.3%si,  0.0%st
Cpu3  :  7.7%us, 15.4%sy,  0.0%ni, 65.4%id,  0.0%wa,  0.0%hi, 11.5%si,  0.0%st
```

▼表1　本連載で紹介したツールのまとめ

ツール名	お勧めレベル	振り返り
UnixBench	★★★	おもにCPUの性能を測定する。さまざまな種類のテストを繰り返し実行するので手軽に良い測定結果が得られる
SysBench	★★★	CPUやメモリなどさまざまな対象の性能を測定する。中でもOLTP(DB)のベンチマークでは、MySQLやPostgreSQLなどいろいろなDBの測定が手軽にできて便利
fio	★★★	DiskI/Oの性能を測定する。さまざまなオプションが用意されており多用なI/Oパターンを測定できる。jobfileを利用し自動化すれば測定も楽である
ioping	★★	DiskI/Oの測定だがレイテンシに特化している。ping間隔でレイテンシを測定できるので簡単便利
mysqlslap	★	DB(MySQL)の性能を測定する。MySQLをインストールしたらすぐに測定を開始できる。本気でベンチマークをしようとすると少し不便
tpcc-mysql	★★	DB(MySQL)の性能を測定する。ディスクI/Oまで負荷をかける、大規模なDBへ長時間負荷をかけるなど本気でDBベンチマークするならこちらがお勧め
netperf	★	ネットワークの帯域を測定する。ネットワークベンチマークを、帯域の測定するだけならばこれで十分。オプションも豊富なので帯域をフルに測定可能
ab	N/A	Webサーバの性能を測定する。定番ツールだが非力。ただし、たいていのWebアプリに対しては十分負荷をかけることができる
wrk	★	Webサーバの性能を測定する。abで足りなければこちらをどうぞ。ただし、今回の記事で紹介したネットワーク規模への負荷が限界。もっと大規模に負荷をかける場合はJmeter(https://jmeter.apache.org)や商用ツールを使用すること

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

【第十一回】スノーデン事件が意味するもの

今回はいつもの実践的な話題ではなく、情報セキュリティを広い視点から見つめ直すために、スノーデン事件とその背景を追っていきます。この事件を追うことで、いつもとは違う角度から情報セキュリティとは何かを考えてみたいと思います。

スノーデン事件とは

アメリカやヨーロッパとは違い、日本ではスノーデン事件をきちんと報道しているところをほとんど見かけないので、まずはスノーデン事件について説明します。

エドワード・J・スノーデン(Edward Joseph Snowden)は1983年アメリカ生まれの男性です。この一大スキャンダルは、2013年6月にスノーデンが英ガーディアン紙と米ワシントンポスト紙に、米国諜報部局の活動についてリークしたことから始まります。このリークにより、諜報部局がインターネット監視のために行っているプログラムPRISM、MUSCULAR、XKeyscore、Temporaと、米国内とEU国内の通話記録を大規模に収集していることが公表されました[注1]。

日本では、彼のことを元CIA職員と紹介しますが、正しくは、ブーズ・アレン・ハミルトンというコンサルティング会社の社員として、NSAに派遣されていたときに、外部公開することを前提にこれらの情報を密かに収集していました[注2]。日本ではNSAという組織に馴染みがないためか、この問題と直接は関係ないそれ以前のキャリアである元CIAという肩書きを使うため、この事件の本質がぼやけてしまっている気がします。

NSA

NSA(National Security Agency)は、日本では国家安全保障局という名前が使われています。インテリジェンス(諜報)のとくにSIGINT(Signal Intelligence)を中心に担当している専門部局です。SIGINTとは国家安全保障上の観点から通信(シグナル)を取り込み、諜報活動を行うことを指します。

電信、電波という技術を持って以降、これらの通信技術は連絡をするのに重要な役割を果たしてきました。その通信によって重要な情報を送るわけですが、その通信を盗聴して相手が行動に出る前に何を企てているかがわかれば、これほど有利なことはありません。もちろん重要な通信内容は暗号化されています。この暗号を解読することが重要かつ決定的な意味を持ちます。

暗号解読の重要性が認知されたミッドウェイ海戦

1941年12月7日(現地日時)に、大日本帝国海軍(以下、日本海軍)はハワイオアフ島の真珠湾にある米国海軍太平洋艦隊基地を攻撃します。この真珠湾攻撃はまったくの不意打ちでした。米海軍のSIGINT部隊であるOP-20-G(Office of Chief Of Naval Operations, 20th Division of the Office of Naval Communications, GSection / Communications Security)

注1) http://www.theguardian.com/world/the-nsa-files
注2) http://bigstory.ap.org/article/ap-impact-snowdens-life-surrounded-spycraft

が当時、日本海軍の使っていた「海軍暗号D号」、米国側でJN-25と呼んでいた暗号を解読できなかったためです[注3]。

そのJN-25は1942年5月までにほぼ解読されます。1942年6月4日(現地日時)に、ミッドウェイ島周辺で行われた日米の大規模な海戦「ミッドウェイ海戦」では、すでに日本側の通信が解読されて、その動きが米軍に事前に把握されていました。当初、米軍は日本海軍が"AF"へ攻撃を加えるということはわかったのですが、そのAFがどこかがわかりませんでした。そこで米軍は、「ミッドウェイ島では水が足りない」という偽の情報を流します。日本軍は「AFは水が足りない」という暗号文を流し、AFがミッドウェイ島だということがわかってしまいます(**写真1**)。

ミッドウェイ海戦で日本海軍は大打撃を受け、これが太平洋戦争のターニングポイントになります。このような歴史的とも言える状況を作り出したミッドウェイ海戦は米国戦史においてSIGINTによる劇的な成果をあげた歴史的場面でもありました。この勝利から「暗号解読技術というのは、戦況を大きく左右する重要な軍事技術だ」という認識になり、その後、長い間厚いベールに隠されます。

筆者は、ハワイの真珠湾にあるアリゾナメモリアルを訪れたことがあります。そのメモリアルには、ミッドウェイ海域に偵察に向かったパイロットが日本の艦隊を見つけ、基地に打電するという伝説的な物語が紹介されています。その物語では、あたかもパトロール中に偶然発見したかのような書きぶりです。もちろんそれは違い、日本海軍の行動を暗号解読により把握し、海域を集中的に探査したものです。とはいえ、大海原で探すのはたいへん労力を要したのは言うまでもありません。しかし、事前に察知したにもかかわらず、そのようなことがいっさい書かれていないのは、米国の暗号解読技術が戦略技術として長い間、極秘事項にされていたからです。

秘密裏に設立されたNSA

第二次世界大戦後、いくつかの経緯を経て1952年にSIGINT専門の諜報組織NSAが作られます。しかし、第二次世界大戦で戦況を大きく左右させた暗号技術は、軍事機密として最高レベルのものであり、米国政府はNSAという組織の存在すらも公式には認めず、NSAは"No Such Agency(そんな組織はない)"と言われるぐらい徹底した極秘ぶりでした。1983年にニューヨークタイムス紙にNSAについての報道がなされるまで米国政府は正式にその存在を認めませんでした[注4]。

ところがいったん認めてしまうとあっさりしたもので、それから13年後の1995年、ワシントンD.C.にいる友人と、郊外をドライブしていると高

◆**写真1　"AF IS SHORT OF WATER"**

筆者がNSAの国家暗号博物館(National Cryptologic Museum)に訪れたときに撮影した写真。パネル中央の写真下のプレートに有名な"AF IS SHORT OF WATER"という言葉が刻まれている。

注3) What Every Cryptologist Should Know about Pearl Harbor **http://www.nsa.gov/public_info/_files/cryptologic_quarterly/pearlharbor.pdf**
Pearl Harbor Review - JN-25 **http://www.nsa.gov/about/cryptologic_heritage/center_crypt_history/pearl_harbor_review/jn25.shtml**

注4) **http://www.nytimes.com/1983/03/27/magazine/the-silent-power-of-the-nsa.html**

速道路の出口にデカデカと「NSA職員のみ利用可能」と看板が出ていてビックリしました（極秘組織じゃなかったのか……）。

さらに余談ですが、それから6年後の2001年USENIX SecurityのエクスカーションでNSA敷地内にある国家暗号博物館を訪れて、過去にNSAが利用していた歴代のスーパーコンピュータや暗号解読の歴史を見ることができました。ついでに売店でNSAのロゴが入ったポロシャツシャツ、帽子、ステッカー、マウスパッドも購入しました。

NSAは矛と盾を持つ

NSAのWebサイト[注5]にいくと“NSA/CSS”という表記がなされていることに気づくと思います。CSSはCentral Security Serviceの略で、NSAは通信を盗聴し暗号解読するエキスパートですが、その能力をもって、今度は同時に通信を守ることもそのミッションとして割り当てられています。

NSAは軍事だけではなく、インテリジェンス・コミュニティ、つまり数ある米国の諜報部局の中の1つとして、あらゆる分野のSIGINTを担当しています[注6]。

DNIとNSA

日本ではあまり知られていないですが、現在、米国のインテリジェンス・コミュニティを理解するうえで、重要な役割を果たすDNI（Director of National Intelligence：国家情報長官）を紹介しましょう。

2001年9月11日に米国で発生した同時多発テロ事件、いわゆる911テロのあと、「米国インテリジェンス・コミュニティがうまく機能していなかったために、この米国史上最悪のテロを事前に防げなかった」という批判が噴出しました。テロ計画の情報は、断片的にはいろいろな諜報部局でつかんでいたのは事実のようです。しかし、おのおのの諜報部局はセクショナリズムが横行し、情報を共有することはなく、その結果として取り返しのつかない大きなテロへつながった。そのような分析がなされているようです。

それまでインテリジェンス・コミュニティの取りまとめの役割を負っていたのは、DCI（Director of Central Intelligence：中央情報長官）で、CIA長官が兼任していました。CIA長官という立場とほかの組織をまとめるという立場で利益相反してしまい、結果としてうまく調整ができていないと議会による911検証レポート（The 9/11 Commission Report）で指摘されてしまうこととなります。その結果として、2004年よりDNIが正式に発足しました（その前身は2002年から）。

ちなみに、16組織あるこのインテリジェンス・コミュニティ（**図1**）の総予算は2005年時点で440億ドル（当時のレートで約4兆8,000億円）だったことを、当時の副長官だったメアリー・マーガレット・グラハム（Mary Margaret Graham）が講演で語っています[注7]。

のちにODNI（Office of DNI）が中心となり、イン

◆**図1　16組織から成るインテリジェンス・コミュニティ**

出　典：http://en.wikipedia.org/wiki/United_States_Intelligence_Community
911テロ以降、米国のインテリジェンス・コミュニティは体制が再編され、現在はODNI傘下の組織として再編されている。

注5）http://www.nsa.gov/
注6）1995年の日米自動車交渉といった政府間協議の場面でも暗躍しています。
注7）Defense Dept. pays $1B to outside analysts　http://usatoday30.usatoday.com/news/washington/2007-08-29-dia_N.htm

テリジェンス・コミュニティが、今日使っている情報システム共通基盤が構築されていきます。情報システム基盤が共通化されることで、どこからでも情報を入れたり出したりでき、情報が有機的につながり、911テロのような諜報作戦が後手後手に回るようなことがないような体制を整えられます。ただし、これはあくまでも理想論であり、本当にできるかどうかは別問題です。

この共通化システムという背景が、米国のインテリジェンス・コミュニティにおけるスノーデン事件のインパクトをより大きくします。なぜならば、スノーデンがリークしたシステムは、1つの諜報部局に止まらず、ODNI傘下にある米国すべての諜報部局に影響を与えるからです。

共通化システムが筆者にもたらした意外な展開

共通化システムでは、ODNIの各諜報部局が持っていたいろいろなデータベース（たとえば、人物プロファイルなど）をマージしています、というか、たぶんマージしたのだと思います。だいたいにおいて、似ているけど少しずつ違うデータベースをかき集めたら、その内容が怪しくなるものです。なぜ、そんなことが言えるのか？　これは筆者の体験談によるものです。

コンピュータ・セキュリティや暗号システムの研究や開発などいろいろなことをしている筆者ですが、その中で、海外の方々と名刺交換したり、あるいは米国政府組織主催のワークショップに参加したりしています[注8]。

米国において暗号技術は伝統的に国家安全保障の扱いですので、米国政府主催の暗号関連カンファレンスなどに登録すれば自動的にNSAのデータベースにプロファイリングされます。FBI、シークレットサービス、国防総省傘下の研究所の人とコンピュータ・セキュリティのカンファレンスなどで名刺交換するのもそうです。それらの人物プロファイリングが各組織にあったときは、おのおののデータは整合性が取れていたことでしょう。

ODNI傘下に再編され、911以降にマージされたプロファイリング・データベースに、どこからか筆者に関係する重要なキーワードが入りました。それは「オープンソース」です。諜報での「オープンソース」とは開示されている情報から、諜報活動を行うことを指します。そのことをOSINT（Open Source Intelligence）と言います。

なぜ、そんなことを筆者が知ったかというと、直接ODNIからオープンソース・カンファレンスの案内メールが筆者に送られてきたからです。そのときは、ODNIの組織体制も知らず、単純に「米国政府関連のセキュリティ組織がオープンソースを本格的に使うのか」くらいにしか考えていませんでした。米国政府が主催で無料だし、おもしろそうなので、自由ソフトウェアがどのように使われているか一度調べてみたかったので、さっそく申し込み、ワシントンD.C.まで飛びました（**写真2**）。

実際にカンファレンス会場について、最初のキーノートスピーチを聞いて初めて、筆者は大きく勘違いしていることに気がつきました。もちろんODNIが筆者のことを大きく間違えているのが、この信じられない出来事の発端なのですが、事実は小説より奇なり、周りの人のネームタグにある所属はCIA、NSA、FBI、DoDなどが8割で、あとの2割は大学

◆**写真2　DNI Open Source Conferenceの看板**

2007年7月16～17日に米ワシントンD.C.で開催されたDNI Open Source Conferenceに参加した。参加費は無料だが、参加人数が限られており、案内メールが送られて3日後には定員に達してしまった。

注8）過去に、それらのレポートを『Software Design』に書いたこともあるので、知っている読者もいらっしゃるかもしれません。

の人間でした。

そのときに、壇上で講演するメアリー・マーガレット・グラハムを直接見ることができたのですが、CIAで27年間勤めあげ、アメリカの諜報のNo.2だとは思えない品のいいやさしそうなおばさんでした。グラハムは政府の仕事から引退し、現在はハーバード大学のInstitute of Politicsにフェローという肩書きを持って活躍しているようです[注9]。

たいへん横道に外れてしまいました。さて、繰り返しになりますが、911テロ以降は、米国のインテリジェンス・コミュニティは、昔のようにCIAやNSAといった諜報部局がおのおの独立して情報を管理しているわけではなく、ODNI傘下で共通化しているという時代になっています。

そして、現在、DNIのポジションには米空軍出身のジェームズ・クラッパー(James R. Clapper)が就いています。スノーデン事件から明るみに出た同盟国首脳の電話盗聴など数々のスキャンダルの事態の収拾を図っています。

PRISM

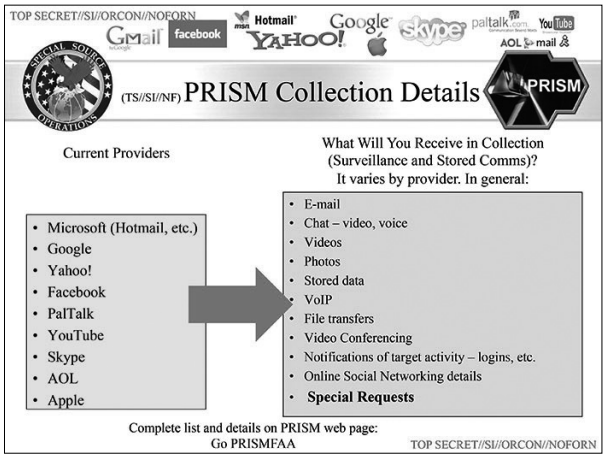
PRISMとは、NSAが2007年からスタートさせた極秘の巨大データをマイニングする監視システムです。これはSIGINT Activity Designator(SIGAD)と呼ばれる情報収集に分類される活動です。この活動の政府コードはSIGAD US-984XNとなっています。つまり、SIGINTの活動範囲なのでNSAの作業分担であることがよくわかります。

PRISMはインターネット上のデジタル情報を収集する活動です。**図2**～**4**はスノーデンがリークした内部資料の一部です。**図2**を見ると、大手のポータルサイトやインターネットサービスから、電子メール、チャット、VoIP、ビデオなど、我々がコミュニケーションで使っているほぼすべてのデータを監視しているのがわかります。

インターネットに使われている回線を盗聴し、データを入手する方法を採っているようで、これらのデータを保持している会社から直接入手する、あるいはサーバに侵入してデータを盗む、といった手法ではないようです。ですからターゲットとされた会社もユーザも誰もわからない間に監視が行われているという状態になっていたはずです。

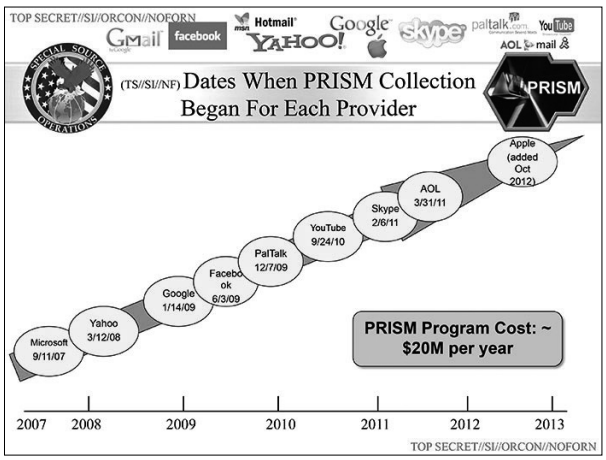
PRISM計画の進捗(**図3**)を見ると、2007年にMicrosoftからスタートし、2012年のApple社まで

◆**図2　PRISMが収集している情報**



出典：http://en.wikipedia.org/wiki/PRISM_(surveillance_program)

◆**図3　PRISM計画の進捗**



出典：http://en.wikipedia.org/wiki/PRISM_(surveillance_program)

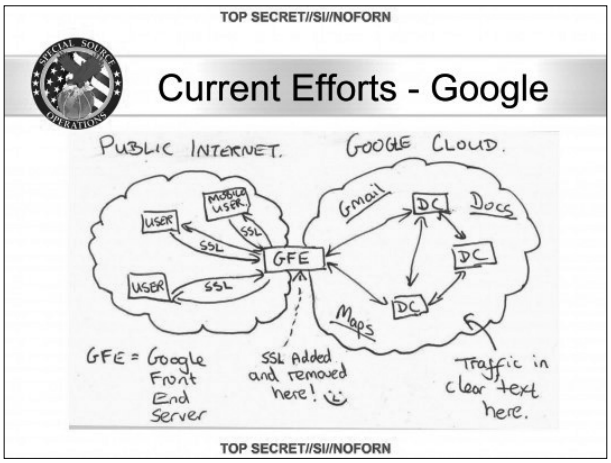
注9) Mary Margaret Graham 2008 Fall Resident Fellow http://www.iop.harvard.edu/mary-margaret-graham

毎年20百万ドル（約20億円）を投入し、徐々に作り上げていったことがわかります。また、これらで入手した監視情報はNSAで最高機密として扱われ、必要に応じてFBIやCIAからアクセスできるかたちになっています。

GoogleのGmailにアクセスするときは、SSLで保護されています。そして、この連載でもSSLが正しく設定されていれば、その通信は確実に保護されると説明してきました。では、なぜ暗号で保護した通信が盗聴できるのでしょうか。Googleのクラウド内の盗聴を表している**図4**を見ると一目瞭然です。

ユーザからのSSLはフロントエンドのサーバに接続され、インターネットの回線上は確実に保護されています。クラウド内部の矢印は、データベースへのアクセスや処理分散をしている部分、はたまたプロキシをかけているようなデータの流れを意味しています。いったん内部に入ってしまえば、データセンター内では暗号化していないのが普通です。なぜならば、その領域は外部からは接続できない内部ネットワークだからです。このような構成はごくごく一般的なものです。**図4**を見る限り、Googleクラウドが入っているデータセンター内のネットワークトラフィックを盗聴していると判断できます。しかし、最大の謎が残ります。なぜデータセンターのトラフィックをやすやすと盗めるのだと。このあたりが我々には計りしれない米国の諜報部局のなせる技なのでしょう。

◆**図4　Googleのクラウド内の盗聴**



出典：http://en.wikipedia.org/wiki/PRISM_(surveillance_program)

陰謀論ではない現実

莫大な予算を持つNSAという組織が存在し、インターネット上の情報を盗聴、監視していることは、はるか以前から言われていました。

たとえば、Emacsではspook.elというプログラムが用意されています[注10]。Emacsで“M-x spook”と実行するとバッファに（文章としては意味をなさない）NSAの盗聴システムが反応するようなキーワードが入ります（**リスト1**）。そのプログラムのリポジトリを見ると、最初のバージョンは1988年に登録されています。つまり、NSAが電子メールを盗聴／監視しているのは、1988年から織り込み済みというわけです。ただし、これまでそれを証明するチャンスはありませんでした。

もしスノーデンがリークしなかったら、あるいは内部から持ち出したこれらの資料の存在を英ガーディアンや米ワシントンポストといった有力新聞社の手によって公開されなかったら、監視システムがあると噂されていても、永遠に陰謀論として一蹴されていたでしょう。

なんであれ第三者から情報を保護するのが情報セキュリティの重要な役割です。これらの盗聴からデータを守る技術を開発することが我々セキュリティ技術者には課せられているのではないでしょうか。SD

◆**リスト1　spook.elで表示されるキーワードの例**

```
nitrate insurgency NORAD event security $400 million in gold bullion red noise Exon Shell
bullion AIMSX Janet Reno assassinate NSA Craig Livingstone passwd Geraldton
```

注10）http://cvs.savannah.gnu.org/viewvc/emacs/emacs/lisp/play/spook.el

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19回 bhyveにおける仮想NICの実装

Writer 浅田 拓也（ASADA Takuya） Twitter @syuu1228

はじめに

これまでに、ゲスト上で発生したI/Oアクセスのハンドリング方法、virtio-netのしくみなど、仮想NICの実現方法について解説してきました。今回の記事では、/usr/sbin/bhyveが、仮想NICのインタフェースであるvirt-netに届いたパケットをどのように送受信しているのかを解説していきます。

bhyve における仮想 NIC の実装

bhyveでは、ユーザプロセスである/usr/sbin/bhyve にて仮想I/Oデバイスを提供しています。また、仮想I/Oデバイスの1つであるNICは、TAPを利用して機能を提供しています。仮想NICであるTAPを物理NICとブリッジすることにより、物理NICが接続されているLANへ参加させることができます（**図1**）。

どのような経路を経て物理NICへとパケットが送出されていくのか、ゲストOSがパケットを送信しようとした場合を例として見てみましょう。

▶①NICへのI/O通知

ゲストOSはvirtio-netドライバを用いて、共有メモリ上のリングバッファにパケットを書き込み、I/Oポートアクセスによってハイパーバイザにパケット送出を通知します。I/OポートアクセスによってVMExitが発生し、CPUの制御がホストOSのvmm.koのコードに戻ります[注1]。vmm.koはこのVMExitを受けてioctlをreturnし、ユーザランドプロセスである/usr/sbin/bhyveへ制御を移します。

▶②共有メモリからパケット取り出し

ioctlのreturnを受け取った/usr/sbin/bhyveは、仮想NICの共有メモリ上のリングバッファからパケットを取り出します[注2]。

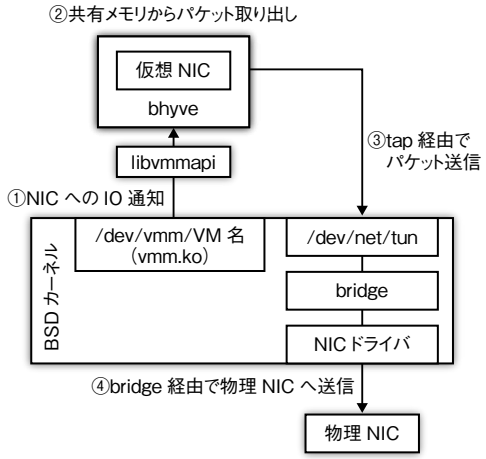
▶③tap経由でパケット送信

②で取り出したパケットをwrite()システムコールで/dev/net/tunへ書き込みます。

注1） I/Oアクセスの仮想化とVMExitについては連載第3回を参照してください。

注2） 仮想NICにおけるvirtio-netのデータ構造とインターフェースの詳細に関しては、連載第11回・第12回を参照してください。

▼図1 パケット送信手順



▶④ bridge経由で物理NICへ送信

TAPはbridgeを経由して物理NICへパケットを送出します。

受信処理ではこの逆の流れをたどり、物理NICからtapを経由して/usr/sbin/bhyveへ届いたパケットがvirtio-netのインタフェースを通じてゲストOSへ渡されます。

TAPとは何か

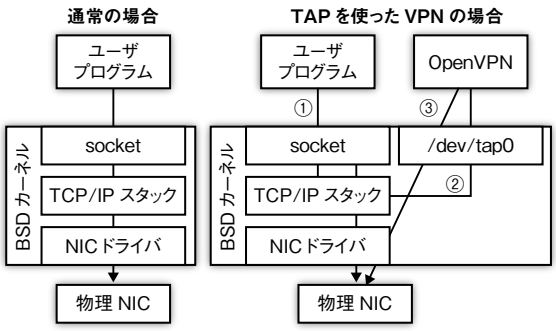
bhyveで利用されているTAPについてもう少し詳しくみていきましょう。TAPはFreeBSDカーネルに実装された仮想Ethernetデバイスで、ハイパーバイザ／エミュレータ以外ではVPNの実装によく使われています[注3]。

物理NIC用のドライバは物理NICとの間でパケットの送受信処理を行いますが、TAPは/dev/net/tunを通じてユーザプロセスとの間でパケットの送受信処理を行います。このユーザプロセスがSocket APIを通じて、TCPv4でVPNプロトコルを用いて対向ノードとパケットのやりとりを行えば、TAPは対向ノードにレイヤ2で直接接続されたEthernetデバイスに見えます。

これがOpenVPNなどのVPNソフトウェアがTAPを用いて実現している機能です(**図2**)。

では、ここでTAPがどのようなインターフェースをユーザプロセスに提供しているのか見ていきましょう。TAPに届いたパケットをUDPでトンネリングするサンプルプログラムの例を**リスト1**に示します。

▼図2　通常のNICドライバを使ったネットワークとTAPを使ったVPNの比較



注3）正確にはTUN/TAPとして知られており、TAPがEthernetレイヤでパケットを送受信するインターフェースを提供するのに対しTUNデバイスはIPレイヤでパケットを送受信するインターフェースを提供します。また、TUN/TAPはFreeBSDのほかにもLinux、Windows、OS Xなど主要なOSで実装されており、共通のAPIで操作することができます。

▼リスト1　TAPサンプルプログラム(Ruby)

```
require "socket"
TUNSETIFF = 0x400454ca
IFF_TAP = 0x0002
PEER = "192.168.0.100"
PORT = 9876
# TUNTAPをオープン
tap = open("/dev/net/tun", "r+")
# TUNTAPのモードをTAPに、インターフェース名を"tapN"に設定
tap.ioctl(TUNSETIFF, ["tap%d", IFF_TAP].pack("a16S"))
# UDPソケットをオープン
sock = UDPSocket.open
# ポート9876をLISTEN
sock.bind("0.0.0.0", 9876)
while true
    # ソケットかTAPにパケットが届くまで待つ
    ret = IO::select([sock, tap])
    ret[0].each do |d|
        if d == tap # TAPにパケットが届いた場合
            # TAPからパケットを読み込んでソケットに書き込み
            sock.send(tap.read(1500), 0, Socket.pack_sockaddr_in(PORT, PEER))
        else # ソケットにパケットが届いた場合
            # ソケットからパケットを読み込んでTAPに書き込み
            tap.write(sock.recv(65535))
        end
    end
end
```

ユーザプロセスがTAPとやりとりを行うには、/dev/net/tunデバイスファイルを用います。

パケットの送受信は通常のファイルIOと同様にread()、write()を用いることができますが、送受信処理を始める前にTUNSETIFF ioctlを用いてTAPの初期化を行う必要があります。

ここでは、TUNTAPのモード(TUNを使うかTAPを使うか)とifconfigに表示されるインターフェース名の指定を行います。

ここでTAPに届いたパケットをUDPソケットへ、UDPソケットに届いたパケットをTAPへ流すことにより、TAPを出入りするパケットをUDPで他ノードへトンネリングできます(**図2**の右側相当の処理)。

bhyveにおける仮想NICとTAP

VPNソフトウェアではTAPを通じて届いたパケットをユーザプロセスからVPNプロトコルでカプセル化して別ノード送っています。

ハイパーバイザでTAPを用いる理由はこれとは異なり、ホストOSのネットワークスタックに仮想NICを認識させ物理ネットワークに接続し、パケットを送受信するのが目的です。

▼リスト2　/usr/sbin/bhyveの仮想NICパケット受信処理

```
/* TAPからデータが届いたときに呼ばれる */
static void
pci_vtnet_tap_rx(struct pci_vtnet_softc *sc)
{
  struct vqueue_info *vq;
  struct virtio_net_rxhdr *vrx;
  uint8_t *buf;
  int len;
  struct iovec iov;
      ……(中略)……
  vq = &sc->vsc_queues[VTNET_RXQ];
  vq_startchains(vq);
      ……(中略)……
  do {
      ……(中略)……
    /* 受信キュー上の空きキューを取得 */
    assert(vq_getchain(vq, &iov, 1, NULL) == 1);
      ……(中略)……
    vrx = iov.iov_base;
    buf = (uint8_t *)(vrx + 1); /* 空きキューのアドレス */
    /* TAPから空きキューへパケットをコピー */
    len = read(sc->vsc_tapfd, buf,
         iov.iov_len - sizeof(struct virtio_net_rxhdr));
    /* TAPにデータがなければreturn */
    if (len < 0 && errno == EWOULDBLOCK) {
      ……(中略)……
      vq_endchains(vq, 0);
      return;
    }
      ……(中略)……
    memset(vrx, 0, sizeof(struct virtio_net_rxhdr));
    vrx->vrh_bufs = 1; /* キューに接続されているバッファ数 */
      ……(中略)……
    vq_relchain(vq, len + sizeof(struct virtio_net_rxhdr));
  } while (vq_has_descs(vq)); /* 空きキューがある間繰り返し */
      ……(中略)……
  vq_endchains(vq, 1);
}
```

このため、VPNソフトウェアではソケットとTAPの間でパケットをリダイレクトしていたのに対して、ハイパーバイザでは仮想NICとTAPの間でパケットをリダイレクトすることになります。

それでは、このリダイレクトの部分についてbhyveのコードを実際に確認してみましょう（**リスト2**）。

この関数はsc->vsc_tapfdをkqueue()/kevent()でポーリングしているスレッドによってTAPへのパケット着信時コールバックされます。コードの中では、virtio-netの受信キュー上の空きエリアを探して、TAPからキューが示すバッファにデータをコピーしています。これによって、TAPへパケットが届いたときは仮想NICへ送られ、仮想NICからパケットが届いたときはゲストOSに送られます。その結果、bhyveの仮想NICはホストOSにとってLANケーブルでtap0へ接続されているような状態になります。

TAPを用いたネットワークの構成方法

前述の状態になった仮想NICでは、IPアドレスが適切に設定されていればホストOSとゲストOS間の通信が問題なく行えるようになります。しかしながら、このままではホストとの間でしか通信ができず、インターネットやLAN上の他ノードに接続する方法がありません。この点においては、2台のPCをLANケーブルで物理的に直接つないでいる環境と同じです。

これを解決するには、ホストOS側に標準的に搭載されているネットワーク機能を利用します。1つの方法は、すでに紹介したブリッジを使う方法で、TAPと物理NICをデータリンクレイヤで接続し、物理NICの接続されているネットワークにTAPを参加させることます。しかしながら、WiFiでは仕様によりブリッジが動作しないという制限があったり、LANから1つの物理PCに対して複数のIP付与が許可されていない環境で使う場合など、ブリッジ以外の方法でゲストのネットワークを運用したい場合があります。

この場合は、NATを使ってホストOSでアドレス変換を行ったうえでIPレイヤでルーティングを行います[注4]。bhyveではこれらの設定を自動的に行うしくみをとくに提供しておらず、TAPにbhyveを接続する機能だけを備えているので、自分でコンフィギュレーションを行う必要があります。

リスト3、**4**に/etc/rc.confの設定例を示します。なお、OpenVPNなどを用いたVPN接続に対してブリッジやNATを行う場合も、ほぼ同じ設定が必要になります。

まとめ

今回は仮想マシンのネットワークデバイスについて解説しました。次回は、仮想マシンのストレージデバイスについて解説します。SD

注4）NATを使わずにルーティングだけを行うこともできますが、その場合はLAN上のノードからゲストネットワークへの経路が設定されていなければなりません。一般的にはそのような運用は考えにくいので、NATを使うことがほとんどのケースで適切だと思われます。

▼リスト3　/etc/rc.conf設定例1（ブリッジの場合）

```
cloned_interfaces="bridge0 tap0"
autobridge_interfaces="bridge0"
autobridge_bridge0="em0 tap*"
ifconfig_bridge0="up"
```

▼リスト4　/etc/rc.conf設定例2（NATの場合）

```
firewall_enable="YES"
firewall_type="OPEN"
natd_enable="YES"
natd_interface="em0"
gateway_enable="YES"
cloned_interfaces="tap0"
ifconfig_tap0="inet 192.168.100.1/24 up"
dnsmasq_enable="YES"
```

新連載

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1回 技術と技術の間にあるもの

Red Hat Enterprise Linuxは、企業向けの商用Linuxディストリビューションの国内市場において圧倒的なシェアを持つ一方で、意外とその利用方法がきちんと理解されていない面があります。連載第1回では、Red Hatとサポート契約を結ぶメリットとサポートの利用方法を理解しましょう。

レッドハット株式会社　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）

スペックアップしませんか?

連載のタイトルを決めるにあたって正直なところちょっと悩みました。Red HatやFedora関連の旬のニュースを取り扱いつつ、主に初心者に向けた内容をとの執筆依頼でしたが、"Hot Topics"はすでにDebian山根さんが使ってますし、浅田さんや青田さんのように"ハイパーバイザ"とか"kernel"といった深い内容でもありませんし。

連載プランを検討している最中にRPM[注1]パッケージについては必ず説明しようと思ったところで、ふと、RPMパッケージを作成する際に用いる定義ファイルであるSPEC（Specification、「仕様」）ファイルに行き当たり、初心者が「スペックアップ」するきっかけになるような連載を心がける、ということと掛けて「.SPECs（ドット・スペックス）」としました。

Red Hat Enterprise Linux（以降、RHELと省略、RHELはレルと発音する）の技術面での基礎的な内容については、同僚である中井悦司氏が6月号で説明する予定です。そこで連載第1回となる本稿では商用Linuxに初めて触る、あるいはしばらく触っていなかったエンジニアの方を対象として、「エンジニアが扱わないとならないことも多いけれど、純粋に技術的とは言えない話題」についてご紹介します。

純粋に技術的とは言えない話題

上述の導入部に対して読者諸氏は「何を書くんだろう？」と思われたかもしれません。というのも技術誌である本誌で、プログラミング言語でも、最新のLinuxカーネルのパッチでも、仮想化ハイパーバイザでも、そしてエディタの話題でもないというと、何を？と思われるのはもっともなことです。

でも、実際にRed Hatに問い合わせいただいている質問[注2]の多くは、契約に関することや、サポートの内容に関すること、あるいはRed HatのWebサイトに掲載されていることだったりするのも純然たる事実です。

また、学生時代にLinuxに触れたことがある、あるいは入社して商用Linux（ここではもちろんRHELのことですが）に初めて触ったという

注1） RPM：もともとは"Red Hat Package Manager"の略だが、現在は"RPM Package Manager"となっている。
注2） Red Hatへの一般的な問い合わせ窓口（http://jp.redhat.com/contact/sales.html）。

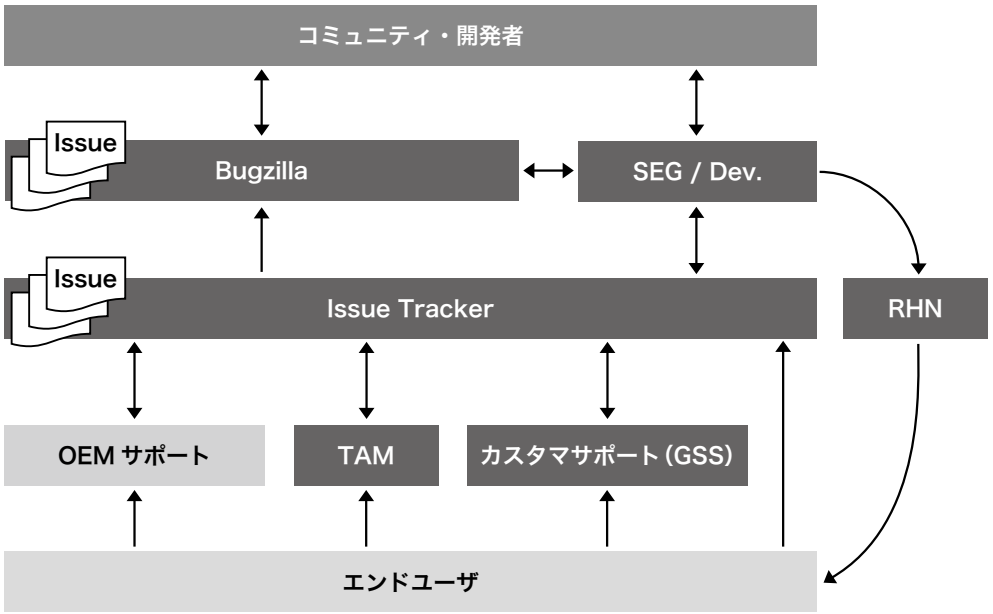
エンジニアは、Linuxのコマンドや設定ファイルに通じていても、いざ商用サービスの構築に際してRed Hatとサポート契約を結ぶという段になると知らないことの方が多いというのは、技術を深掘りしてきた人にほど当てはまることのように思えます。

Red Hatとの契約＝サブスクリプション

Red HatのLinuxの歴史は1993年に遡ることができます。Red Hat LinuxはSlackwareやDebianと並ぶ最古参のLinuxディストリビューション[注3]の1つで、雑誌の付録CD-ROMやftpサイト[注4]などを通じて無償で配布され、技術サポートが必要になったらRed Hatとサポート契約を結ぶというのが、そもそもの始まりでした。

1990年代後半になると、商用のアプリケーション、たとえば独・SAP社や米・Oracle社が提供する製品がRed Hat Linuxでの動作を保証するようになりました。しかし当時のRed Hat Linuxはマイナーバージョンが更新（例：7.0から7.1）されるとカーネルやライブラリの互換性が失われ、企業のシステムで用いるには安定性に欠けるという弱点が露呈しました。

▼図1　Red Hatのサポートフロー



TAM：Technical Account Manager、顧客専属のサポートで通常のサポート契約とは別の契約が必要。
SEG：Software Engineering Group、RHN：Red Hat Network、GSS：Global Support Services。
「直接サポート」はGSSに問い合わせ、OEMサポートは（おもに）サーバベンダーのサポート窓口に問い合わせることになる。

注3） ディストリビューション：配布物。"Linux"は狭義には"Linux"という名前のカーネルを指し、シェルやライブラリと組み合わせて初めて実用的な"OS"となる。この「組み合わせて配布されるもの」を"Linuxディストリビューション"と呼ぶ。組み合わされるソフトウェアの出自がGNUプロジェクトに由来するものが多いため、"GNU/Linux"（グニュー（ニュー）・スラッシュ・リナックス）と呼ぶべきだという意見も根強い。

注4） 古いRed Hat Linuxなどの入手（ftp://archive.download.redhat.com/）。

そこで2002年にRed Hat Linux 7.2をベースに、Red Hat Enterprise Linux 2.1(リリース当初は、Red Hat Linux Advanced Server 2.1)という、企業向けに長期のサポートを提供する専用のディストリビューションを提供開始しました。

このサポート契約は期限(例:1年間、3年間)やサポートレベル(Standard、Premium)を定めて提供されるもので、サブスクリプション(Subscription、「購読権」)と呼びます。よく「LinuxはGPL(GNU General Public License)なんだから無償でしょ?」と言われるのですが、それはソースコードやバイナリコードについて言えること[注5]で、サポート契約はサポートエンジニアの工数やノウハウを提供するものなので有償となっています。

サブスクリプションの手続き

上述したようにRed HatはRHELやRed Hat JBoss Middlewareなどさまざまな製品のサポートを提供しており、サポートの窓口は大別して2種類あります。1つはRed Hatの直接サポートで、もう1つはOEMパートナー各社が提供するサポートです(**図1**)。これらは最終的にRed Hatがサポートするという点では同じなのですが、サポートの一次問い合わせ先が異なり、おおむねサーバハードウェアと一緒に購入した場合はOEMサポート[注6]、別に購入した場合は直接サポートです。前者ではハードウェアとソフトウェアのサポートをワンストップで提供してもらえます。後者の直接サポートについては商流が複数あります。大別するとRed Hatから直接購入する方法[注7]と、ディストリビュータと呼ばれる販社から購入する方法があります。いずれの場合も購入申込書やWebサイトのフォームに記入して発注することになります。

サブスクリプション契約が成立すると「納品書」と「サブスクリプション証書」ならびに、もし発注していれば「インストールメディアキット」であるDVD[注8]が送付されます。また、同時にユーザのRed Hatアカウントにサポートを利用する権利が紐付けされます。これはダイレクトエンタイトルメントと呼ばれ、発注時にRed Hatアカウントを持っていればそれを指定します。持っていなければ新規にアカウントを作成するためのURLがメールで送信されるようになっています。

サポートを受ける前提条件

サポートを受けるにはいくつか条件があります。詳細はEA(Enterprise Agreement)という契約文書[注9]に記載されているので、契約内容やサポートレベルの確認も含め、必ず読むようにしてください。

サポートの可否を判断する際に特に重要な項目として、ハードウェア動作認定リスト(HCL[注10])が挙げられます。HCLはハードウェアベンダーがRed Hatの「ハードウェアプログラム」に参加し[注11]、認定を取得したいバージョンのRHEL上でテストプログラムを実行し結

注5) RHELをはじめとするLinuxディストリビューションにはGPLだけでなくBSDライセンスやApacheライセンスなど、オープンソースソフトウェア(OSS)で用いられるさまざまなライセンスが設定されたソフトウェアが含まれ、それらを集めた「集合著作物」のライセンスとしてGPLを設定している。

注6) OEMサポートベンダーの一覧(http://jp.redhat.com/partners/)。

注7) 購入手順について(http://jp.redhat.com/footer/japan-buy.html)。

注8) メディアキットはRHELのすべてのマイナーバージョンについて用意されているわけではない。Red Hat NetworkからDVDのイメージファイル(.isoという拡張子のファイル)をダウンロードして、DVD-Rドライブで「焼く」か、仮想化環境でイメージファイルをそのまま使う方が一般的。

注9) 「RED HATエンタープライズ契約」という名称で入手可能(http://jp.redhat.com/footer/japan-buy.html)

注10) HCL:Hardware Certification List(https://hardware.redhat.com)

注11) 2014年3月現在、ハードウェアプログラムの参加費用は$5,000/年。

果をRed Hatに送付すると、RHELのバージョンやCPUアーキテクチャ(i686、x86_64など)と共に掲載されるようになっています。Red Hatは、動作認定されたマイナーバージョン以降[注12]のRHELでハードウェアに依存する問題が発生した場合、このHCLの掲載の有無でサポートするか否かを決定します。ただし、ハードウェアに依存しない問題、たとえばネームサーバであるbindにセキュリティの脆弱性が発見された、といった場合にはHCLがサポートの可否に影響することはないので安心してください。

Red Hatは何をしてくれるの?

ここまで「サポート」という単語を何の定義もなく用いてきましたが、Red Hatが提供する「サポート」にはさまざまな要素が含まれています。ソフトウェアの不具合の修正はもちろんですが、電話やwebサイトによる問い合わせや、サブスクリプション契約の金額の範囲内での訴訟費用補償[注13]などが挙げられます。契約によって提供される項目やレベルは前述したEA契約に定義されているので、Red Hatに支払う金額分以上に上手に利用してください。

ソフトウェアの不具合の修正に関して、少し詳しく説明しましょう。Red Hatが提供するソフトウェアのうち、どのパッケージや機能がサポートの対象となるのかはWebサイトに定義されています[注14]。かいつまむと、Red Hatが提供するRPMパッケージのうち"Optional"や"Supplementary"というチャネル[注15]に含まれず、第三者によって改変されていないものや、Release Notes / Technical Notes[注16]で"Technical Preview"に指定されていない機能がサポートの対象となります。

Sales Kick Off in Macau

最後にRed Hat周辺の話題を。3月11日から14日までマカオでSales Kick Offという社内イベントがありました。日本法人であるレッドハット株式会社はAPAC(Asia Pacific)地域に属しており、APACでは毎年3月に営業系の社員を集めて営業戦略や新製品の情報を共有するイベントを、APACのいずれかの都市(昨年はバンコク、一昨年は北京)で開いています。今回はやはりRHEL 7やOpenStack、Red Hat Storageなどがプラットフォーム側の話題の中心でした。これらの新製品の情報については、今後の連載の中で紹介していこうと考えていますので、ご期待ください。

一方で(おそらく)PM2.5による大気汚染は一昨年の北京より深刻度を増しているように見受けられ、せっかくの海外出張であるにもかかわらず、ホテルのチェックインからチェックアウトまで、筆者はまったくホテルの外に足を運びませんでした。ちょっと残念でしたね……。SD

注12) たとえばRHEL 6.4で動作認定を取得していれば、6.5、6.6……での動作が保証される。ただしメジャーバージョンについては適用されないため、RHEL 7.xで再度取得する必要がある。

注13) Open Source Assuranceと呼ばれる。Red Hatが提供するソフトウェアがソフトウェア特許に抵触した場合に、当該部分のソースコードの差し替えや、エンドユーザが訴訟の対象となった場合の訴訟費用の補償などを含む(http://www.redhat.com/rhel/details/assurance/)。

注14) サポートの対象となるソフトウェアについて(https://access.redhat.com/site/support/offerings/production/soc)。

注15) RPMパッケージをまとめたものをチャネルと呼ぶ。いわゆる"OS"と呼ばれているものは「親チャネル」でkernelやglibcなどが含まれる。一方でRed Hatが Add-onとして提供している高可用性クラスタなどは「子チャネル」と呼ばれ、ベースとなる親チャネルに紐付く。"Optional"や"Debug Info"なども子チャネルとなる。

注16) Release Notes/Technical Notesは次のURLを参照(http://docs.redhat.com/)。インストーラに含まれるのはString Freezeという開発フェーズのもので、オンライン版は必要があれば更新される。

Be familiar with FreeBSD.

チャーリー・ルートからの手紙

第7回 ◇ BINDの廃止とUnbound/LDNSの導入

◉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。

ベースシステムからBINDを廃止?

FreeBSD 10.0-RELEASEではいくつかソフトウェアの入れ替えも行われました。もっともユーザに影響を与える可能性が高い変更は、ベースシステムからBINDが取り除かれたことでしょう（**図1**）。これまでFreeBSDのベースシステムにはBINDがマージされていました。chroot(8)環境下で動作するように設定されたBINDで、rc.conf(5)に設定を追加したあとはBINDの設定ファイルを書けば使える状態になっていました。

ネームサーバの実装系の中でデファクトスタンダードに近いポジションにあるソフトウェアがBINDなわけですが、BINDのリリースエンジニアリングがFreeBSDプロジェクトと合わないためベースシステムから抜くことになりました。BINDはセキュリティ脆弱性が発見されることが多いソフトウェアでもあります。ベースシステムにマージされたソフトウェアのアップデートはpkg(8)やports(7)経由でインストールしたアプリケーションほどは簡単にアップデートすることができません。ports(7)で提供するほうが適切と判断したわけです[注1]。

▼**図1 FreeBSD 10.0-RELEASEからはベースシステムからBINDが抜かれている**

```
% freebsd-version
10.0-RELEASE
% which named
named not found
% which dig
dig not found
%:
```

▼**図2 補完目的でUnboundとLDNSが導入されている**

```
% which unbound
/usr/sbin/unbound
% which drill
/usr/bin/drill
% which host
/usr/bin/host
%
```

ローカルキャッシュリゾルバとツール UnboundとLDNS

BINDを置き換える目的ではなく補う目的で、FreeBSD 10.0-RELEASEにはUnboudとLDNSが追加されています（**図2**）。Unboundはローカルキャッシュリゾルバとして使用する目的で、LDNSはBINDが提供していたユーティリティを提供する目的で導入されています。なお、BINDが提供していたユーティリティと同じものを継続して使いたい場合には、bind-toolsをインストールして使います（**図3**）。

正引きや逆引きを実施するのに使われてきた

▼**図3 BINDユーティリティと同じコマンドが使いたいならbind-tools**

```
% pkg search -o bind-tools
dns/bind-tools
%
```

注1 ports(7)からインストールするBINDはベースシステムにマージされていたときのBINDと、rc.conf(5)での設定方法や設定ファイルの配置される場所が同じとは限りません（ちょくちょく変わっています）。本番機に導入する前に実験機で試すようにしてください。

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。

dig(1)ですが、このコマンドはLDNSの**drill**(1)コマンドに置き換わりました。drill(1)コマンドはすべてのDNSクエリを発行できるように開発が進められているコマンドで、dig(1)よりも多機能です。使い方や表示されるフォーマットなどはdig(1)と同じです。今後は基本的にdrill(1)を使うものだと考えておいてください。

使ってみようUnbound

さっそくUnboundを使ってみましょう。効果の大きい機能なので基本的に有効にしておいてよいと思います。まず/etc/rc.confに次の設定を追加します。

```
local_unbound_enable="YES"
```

そして、**service**(8)コマンドでUnboundを次のように起動します。

```
$ service local_unbound start
```

drill(1)コマンドを使って正引きを実施してみます。Unboundを起動した直後に1回実行すると**図4**の結果が得られます。クエリにかかった時間は690ミリ秒です。この状態でもう1回正引きを実行してみます。すると**図5**のようにクエリにかかった時間が0ミリ秒と報告されます。

DNSクエリの処理がローカルでキャッシュ処理されたため高速になりました。このように、有効化

▼**図4　www.freebsd.orgを正引き：1回目**

```
% drill www.freebsd.org
;; ->>HEADER<<- opcode: QUERY, rcode: NOERROR, id: 36771
;; flags: qr rd ra ; QUERY: 1, ANSWER: 2, AUTHORITY: 0, ADDITIONAL: 0
;; QUESTION SECTION:
;; www.freebsd.org.     IN      A

;; ANSWER SECTION:
www.freebsd.org.        170     IN      CNAME   wfe0.ysv.freebsd.org.
wfe0.ysv.freebsd.org.   3415    IN      A       8.8.178.110

;; AUTHORITY SECTION:

;; ADDITIONAL SECTION:

;; Query time: 690 msec
;; SERVER: 127.0.0.1
;; WHEN: Sat Mar 15 17:25:30 2014
;; MSG SIZE  rcvd: 72
%
```

▼**図5　www.freebsd.orgを正引き：2回目**

```
% drill www.freebsd.org
;; ->>HEADER<<- opcode: QUERY, rcode: NOERROR, id: 3014
;; flags: qr rd ra ; QUERY: 1, ANSWER: 2, AUTHORITY: 0, ADDITIONAL: 0
;; QUESTION SECTION:
;; www.freebsd.org.     IN      A

;; ANSWER SECTION:
www.freebsd.org.        108     IN      CNAME   wfe0.ysv.freebsd.org.
wfe0.ysv.freebsd.org.   3353    IN      A       8.8.178.110

;; AUTHORITY SECTION:

;; ADDITIONAL SECTION:

;; Query time: 0 msec
;; SERVER: 127.0.0.1
;; WHEN: Sat Mar 15 17:26:33 2014
;; MSG SIZE  rcvd: 72
%
```

するだけで効果が期待できる機能です。

Unboundの設定は/var/unbound/

Unboundを起動すると、システムの状態から/var/unbound/以下に**図6**の設定ファイルが自動的に生成されます(**リスト1、2**)。forward.confにDNSサーバを指定して、unbound.confにローカルで使用する名前解決の設定などを追加できるようになっています。

▼図6　自動生成されるUnboundの設定ファイル

```
% tree /var/unbound
/var/unbound
├── forward.conf
├── root.key
└── unbound.conf

0 directories, 3 files
%
```

hosts(5)の代わりにUnboundを使ってみよう!

ちょっとした数のホストの名前解決であれば、/etc/hostsに名前を書いておくという使われ方をしている方もいらっしゃるかと思います。Unboundが導入されたので、FreeBSD 10.0-RELEASE以降はこの設定をUnboundで肩代わりするといったことができます。

たとえば**リスト3**のようにunbound.confを書き換

▼リスト1　/var/unbound/forward.confの例

```
# Generated by local-unbound-setup
forward-zone:
        name: .
        forward-addr: 192.168.185.2
```

▼リスト2　/var/unbound/unbound.confの例

```
# Generated by local-unbound-setup
server:
        username: unbound
        directory: /var/unbound
        chroot: /var/unbound
        pidfile: /var/run/local_unbound.pid
        auto-trust-anchor-file: /var/unbound/root.key

include: /var/unbound/forward.conf
```

▼リスト3　編集した/var/unbound/unbound.confファイル

```
server:
        username: unbound
        directory: /var/unbound
        chroot: /var/unbound
        pidfile: /var/run/local_unbound.pid
        auto-trust-anchor-file: /var/unbound/root.key

        # Unbound で名前解決を提供する場合の設定例
        interface: 127.0.0.1
        interface: 192.168.1.10
        access-control: 192.168.1.0/24 allow

        local-zone: "ongs.co.jp." transparent

        local-data: "localhost.         IN A 127.0.0.1"
        local-data: "gps1.              IN A 192.168.1.11"
        local-data: "gps1.co.jp         IN A 192.168.1.11"
        local-data: "gps2.              IN A 192.168.1.12"
        local-data: "gps2.ongs.co.jp    IN A 192.168.1.12"

        local-data-ptr: "127.0.0.1      localhost."
        local-data-ptr: "192.168.1.11   gps1.ongs.co.jp."
        local-data-ptr: "192.168.1.12   gps2.ongs.co.jp."

include: /var/unbound/forward.conf
```

えます。この設定で表のような名前解決(正引き、逆引き)を設定したことになります。

`service local_unbound restart`のようにコマンドを実行してUnboundを再起動したら、**図7、8**のようにdrill(1)を使って正引きを逆引きを実施してみましょう。

BINDに代わる機能の導入はFreeBSD 11を目指す

Unboundはローカルキャッシュリゾルバの目的で導入されています。BINDが提供してきたコンテンツサーバとしての目的では導入されていません。コンテンツサーバを運用する場合にはpkg(8)やports(7)でBINDやNSDといったソフトウェアをインストールする必要があります。

FreeBSDのベースシステムにDNSのコンテンツサーバをマージするかどうか、どのソフトウェアをマージするかは現在議論が進められている段階にあります。FreeBSD開発者会議での会議やメーリングリストでの会議などを経て選定が進められ、FreeBSD 11を目処に導入される見通しです。SD

▼表1　設定した内容

ホスト名	IPv4アドレス	備考
localhost	127.0.0.1	
gps1.ongs.co.jp	192.168.1.11	
gps2.ongs.co.jp	192.168.1.12	
gps1	192.168.1.11	正引きのみ
gps2	192.168.1.12	正引きのみ

▼図7　正引きの確認

```
% drill gps1
;; ->>HEADER<<- opcode: QUERY, rcode: NOERROR, id: 27920
;; flags: qr aa rd ra ; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0
;; QUESTION SECTION:
;; gps1.        IN      A

;; ANSWER SECTION:
gps1.   3600    IN      A       192.168.1.11

;; AUTHORITY SECTION:

;; ADDITIONAL SECTION:

;; Query time: 0 msec
;; SERVER: 127.0.0.1
;; WHEN: Sat Mar 15 17:37:29 2014
;; MSG SIZE  rcvd: 38
%
```

▼図8　逆引きの確認

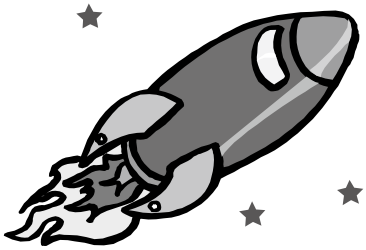
```
% drill -x 192.168.1.11
;; ->>HEADER<<- opcode: QUERY, rcode: NOERROR, id: 9431
;; flags: qr aa rd ra ; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0
;; QUESTION SECTION:
;; 11.1.168.192.in-addr.arpa.   IN      PTR

;; ANSWER SECTION:
11.1.168.192.in-addr.arpa.      3600    IN      PTR     gps1.ongs.co.jp.

;; AUTHORITY SECTION:

;; ADDITIONAL SECTION:

;; Query time: 0 msec
;; SERVER: 127.0.0.1
;; WHEN: Sat Mar 15 17:40:35 2014
;; MSG SIZE  rcvd: 72
%
```

14

Debian Developer　やまねひでき　henrich@debian.org

「ピン留め」でパッケージのバージョンを細かく管理する

Debian Hot Topics



DebianにもLTS？

セキュリティチームが「Bits from the Security Team」というアナウンスで、DebianでもLTS（Long Term Support、長期間サポート版）の提供を検討していることを伝えました。これは現在の安定版Debian 7「Wheezy」についてではなく、2014年5月上旬にサポートが終了する予定のDebian 6「Squeeze」に対してとなっています。Squeezeに対して実験的にLTSサポートを提供し、その結果を見てWheezyやJessieでもLTSを提供するかどうかを決めよう、ということのようです。また、次の方針が出されています。

- LTSの対象アーキテクチャはi386とamd64のみ
- LTSではマイナーな問題の修正は行わない
- 情報の分断を避けるため、LTS専用メーリングリストは設置しない
- 今のところサポート期間の明示はなし

Squeeze-LTSの提供が行われる場合は、Squeezeのサポート提供終了後の2014年5月以降となる予定です。Debian公式ブログ「Bits from Debian」[注1]では「LTSは空から降ってくるものではないので、興味がある人みんなの協力が必要です。セキュリティチームが行うのは、実施作業そのものではなく取りまとめです」とあります。実現に協力いただける方は、お早めにteam@security.debian.orgまでご連絡ください。

注1）URL http://bits.debian.org/

詳解apt line ～「ピン留め」編

前回、案内したbackportsは便利ですが、たまに「backportsにはないけれど、testing/unstableにはある」パッケージを使いたい、という場合もあるでしょう（メンテナががんばってbackportsに含めておいてくれると安心便利……ですが、ベストエフォートでの提供ですのでしかたがないですね）。そのような場合でも、aptを適切に設定することでtesting/unstableなどのパッケージを導入できます。作業は次の手準で行います。

①apt lineにtesting/unstableの設定を追加
②/etc/apt/preferencesの設定で「ピン留め」を実施
③「apt-cache policy」でピン留めの設定を確認
④明示的に別バージョンからパッケージを入れるテストをして、どのパッケージが導入されるかを確認したあと、パッケージをインストール

まず、①についてですが、/etc/apt/sources.

▼リスト1　/etc/apt/sources.listに複数のバージョンを記述

```
wheezy（stable）を使いつつ、jessie（testing）を追加
deb http://ftp.jp.debian.org/debian/ wheezy main
deb http://ftp.jp.debian.org/debian/ jessie main
```

listに複数のバージョンを記述してapt-get updateします(**リスト1**)。

重要なのは、②の「ピン留め」(apt pinning)という設定作業です。apt lineには複数のバージョン(squeeze/wheezy/jessie/sid/experimental)を記述できますが、そのままapt-get upgradeをすると、すべてのパッケージが可能な限り一番新しいバージョン[注2]へと一律に更新されます。**リスト1**の場合は多くのパッケージがjessieに更新されてしまい、wheezyと混ぜて記述する意味がなくなります。「全体として安定した状態を保ちつつ、一部のソフトウェアだけは更新」という目的を満たすには、ピン留めが不可欠です。

ピン留め設定は/etc/apt/preferences(または/etc/apt/preferences.d以下のファイル)に次のような形式で記述します(ちなみにデフォルト状態では/etc/apt/preferencesファイルは存在していませんので、追加してください)。

```
Package: *
Pin: release a=testing
Pin-Priority: 100
```

Pinで指定した「release a=xxx」はリリースについてのピン留めで、Archive名(stable/testing/unstable)を指定します。コードネームで指定する場合は「release n=jessie」のようにしてください。Pin-Priorityの数値は大きいほど優先度が高くなります。ここではtestingに対してデフォルト値「500」より低い値である「100」を割り当てることで、明示的にtestingでのインストールを要求しない限りstableのパッケージのインストールを優先するように設定しています。数値の概要は**表1**を参照してください。

この設定を行った結果、

- 未インストールのパッケージをインストールする→stableがインストールされる
- インストール後にピン留め設定を実施、apt-get upgradeを行う→自動的にtestingのバージョンに更新される

という動作になります。

特定のパッケージについて設定をしたいときは、Package行にパッケージ名を明記します。複数の場合はパッケージ名の間に空白を挟みます。

```
Package: iceweasel iceweasel-l10n-ja
Pin: release a=experimental
Pin-Priority: 990
```

特定のバージョンに固定させたい場合、パッケージ名とバージョンを指定することで実現できます。たとえば、アップデートは基本的に実施したい(squeezeを利用していてwheezyにアップグレードしたい)けど、PHPは5.3で固定したい場合は、次のように設定します。

```
Package: php5*
Pin: version 5.3*
Pin-Priority: 1001
```

*を使ってどの名前／バージョンにでもマッチするように指定

注2) 「可能な限り」というのは一部のパッケージが「dist-upgrade」を実行しないとアップデートされない場合があるからです。パッケージによっては、アップグレードのためにパッケージの削除が必要だったり、新規のパッケージをインストールしないといけないものがあったりします。そのような場合、「upgrade」は安全側に倒してアップデートを行いません。**「dist-upgrade」はアップデートを優先して関連パッケージの削除や追加を行います**。「dist-upgrade」の乱用を行って「なんでパッケージが削除されるんだー！」などと叫んでいる人がいますが、そうならないように**「まずはupgrade、必要に応じて変更点を確認してdist-upgrade」**というのを肝に命じておきましょう。

▼表1　Pin-Priorityの概要

Pin-Priorityの数値	意味
1	指定すればインストールできるが、updateの対象にならない(experimentalがこの設定)
100	インストール済みパッケージについては、ピン留めされたバージョンまでアップデートする
500	現在インストールされていないパッケージの優先度(デフォルト)
990	ピン留めされたバージョンでインストールを実施する
1001	ダウングレードになるとしても、そのパッケージを指定されたバージョンでインストールする

PHP 5についてはバージョンを5.3.*で優先度「1001」とすることで、「何が何でもバージョン5.3.*を利用する」という意味になります。

次に③の作業です。「apt-cache policy」コマンドを実行すると、現在のピン留め設定が出力されます。**図1**ではPHP 5関連のパッケージがピン留めされていることや、testingのピン留めが優先度100で設定されていることが確認できます。

図1の「release o=Debian,a=testing,n=jessie,l=Debian,c=main」などの意味ですが、参照先サーバにあるReleaseファイル(ftp.jp.debian.org/debian/dists/wheezy/Releaseなど)を覗くと、先頭に**リスト2**のような記載があります。

Releaseファイルの記載と/etc/apt/preferencesでの記述は**表2**のように対応しています(おおよそ頭文字でわかるかと思います)。さらなる詳細設定については「man apt_preferences」を参照してください。

ピン留め設定を行ったら、④の作業です。意図どおりにインストールされるか、apt-getでインストールのテストをしてみましょう(テストだけをして、実際にインストールを行わないのは「-s」オプションを指定)。**図2**では、筆者

▼リスト2　Releaseファイルの内容(先頭のみ抜粋)

```
Origin: Debian
Label: Debian
Suite: stable
Version: 7.4
Codename: wheezy
Date: Sat, 08 Feb 2014 10:36:03 UTC
Architectures: amd64 armel armhf i386 ia64
kfreebsd-amd64 kfreebsd-i386 mips mipsel
powerpc s390 s390x sparc
Components: main contrib non-free
Description: Debian 7.4 Released 08
February 2014
```

▼表2　Releaseファイルの記載と/etc/apt/preferencesの対応

Releaseファイル	/etc/apt/preferences
Origin: Debian	o=Debian
Suite：stable	a=stable[注3]
Codename:wheezy	n=wheezy
Label: Debian	l=Debian
Components: main	c=main

注3) 「Suiteでなぜaなんだ?」という疑問はもっともです。stable/testing/unstableは、場面によってArchiveと呼ばれたり、Suiteと呼ばれたり、ディストリビューションと呼ばれたり、バージョンと呼ばれたりとブレがあり、ドキュメントでも一致していません。

▼図1　apt-cache policyコマンドでピン留め設定を確認

```
$ apt-cache policy
パッケージファイル:
100 /var/lib/dpkg/status
    release a=now
100 http://ftp.jp.debian.org/debian/ testing/main amd64 Packages
    release o=Debian,a=testing,n=jessie,l=Debian,c=main
    origin ftp.jp.debian.org
500 http://ftp.jp.debian.org/debian/ wheezy-updates/main amd64 Packages
    release o=Debian,a=stable-updates,n=wheezy-updates,l=Debian,c=main
    origin ftp.jp.debian.org
500 http://security.debian.org/ wheezy/updates/main amd64 Packages
    release v=7.0,o=Debian,a=stable,n=wheezy,l=Debian-Security,c=main
    origin security.debian.org
500 http://ftp.jp.debian.org/debian/ wheezy/main amd64 Packages
    release v=7.4,o=Debian,a=stable,n=wheezy,l=Debian,c=main
    origin ftp.jp.debian.org
Pinされたパッケージ:
(中略)
     php5-xsl -> 5.3.3-7+squeeze17
     php5-mongo -> (見つかりません)
     php5-mysqlnd -> (見つかりません)
     php5-remctl -> (見つかりません)
     php5 -> 5.3.3-7+squeeze17
(略)
```

がパッケージをメンテナンスしているIRCクライアント「loqui」のパッケージを例にします。

普通にapt-get installすると、Debian:7.4/stableからバージョン0.5.3-3が取得されます。ここで-tオプションを付けて取得先のターゲットとしてtestingを明示してみましょう(**図3**)。同じパッケージの取得ですが、出力が変わり、Debian:testingからバージョン0.5.5-2がインストールされることがわかります。

依存関係には注意

このように便利なピン留めですが、testing/unstableからパッケージを借りてくるため、依存関係によっては意図しない大量のパッケージの変更が必要になることがあります。

図4ではJDパッケージをtestingから取得しようとしたところ、依存関係から221個のパッケージがアップグレード、新規インストールが160個、そして38個のパッケージが削除されるという事態になっており、このまま進めると1/6ぐらいがtestingという状態になってしまいます。中途半端で、ちょっとstableとは言いづらいですね。パッケージによって依存関係の状況は異なりますので、必要に応じてテストしてPinを使うかどうかを判断してください。

「ピン留め」の活用でstable生活を豊かに

「常にtesting/unstableを使うのはためらわれるけど、stable(とbackports)のパッケージだけだと不満が……」という方も、ここで紹介したピン留め(apt pinning)を使えば解決できる場合があります。ちょっと動作が難解かもしれませんが、トライしてみる価値はありますので、一度お試しあれ。SD

▼図2　インストールのテスト

```
$ sudo apt-get -s install loqui   ←-sでシミュレーション
パッケージリストを読み込んでいます...完了
依存関係ツリーを作成しています
状態情報を読み取っています...完了
以下のパッケージが新たにインストールされます:
loqui
アップグレード: 0個、新規インストール: 1個、削除: 0個、保留: 7個。
Inst loqui (0.5.3-3 Debian:7.4/stable [amd64])
Conf loqui (0.5.3-3 Debian:7.4/stable [amd64])
```

▼図3　testingを明示的に指定してインストール

```
$ sudo apt-get -s install loqui -t testing   ←-tでターゲットを明示
パッケージリストを読み込んでいます...完了
依存関係ツリーを作成しています
状態情報を読み取っています...完了
以下のパッケージが新たにインストールされます:
loqui
アップグレード: 0個、新規インストール: 1個、削除: 0個、保留: 1234個。
Inst loqui (0.5.5-2 Debian:testing [amd64])
Conf loqui (0.5.5-2 Debian:testing [amd64])   ←testingから先ほどと異なるバージョンを取得できていることに注目
```

▼図4　大量のパッケージ変更が必要になる場合

```
$ sudo apt-get install jd -t testing
パッケージリストを読み込んでいます...完了
依存関係ツリーを作成しています
状態情報を読み取っています...完了
(中略)
アップグレード: 221個、新規インストール: 160個、削除: 38個、保留: 981個。
233 MBのアーカイブを取得する必要があります。
この操作後に追加で246MBのディスク容量が消費されます。
続行しますか[Y/n]?
```

1つのパッケージ更新だが、libcなどへの依存から大量のパッケージがアップグレードされている

グローバル企業のキャリアパス

鶴野 龍一郎
TSURUNO Ryouichiro
レッドハット㈱ シンガポール支社
Platform BU Manager APAC

シンガポールより こんにちは

こんにちは、レッドハットの鶴野です。筆者は、2011年まで東京オフィスでプリセールスの仕事をしていましたが、今はシンガポールオフィス(**写真1**)にてプロダクトマネージメントの仕事をしています。担当としては日本を含むアジア圏でのRed Hat Enterprise Linuxやアドオン製品のプロダクトマネージメントをしています。仕事の内容としてはプリセールスのころよりも表舞台というよりは裏方に近い立場になります。毎日のように客先に訪問していたプリセールス時代よりも夜中に本社と電話会議したり、という地味な内容です。会社は同じでも国、仕事を変えられるというのが当社の面白いところだと思います。

国境のないオフィスと柔軟なキャリアパス

Red Hatでは社内の異動というのがある程度自由にできるようになっています。しょっちゅう知らない人がデスクに座っていて「1ヵ月だけ日本で仕事することにした」、とか「ちょうど休みでロンドンに行くから、休暇のあとはロンドンオフィスでしばらく仕事する」というような話を聞きます。実は、今日も今まで一緒に仕事をしていたほかの部署の女性が旦那様の仕事の都合でロンドンに引っ越し、そのままRed Hatのポジションもロンドンに持っていくと聞きました。

一時的に違う場所で仕事ということであれば、仕事の内容や上司の了解などをクリアしておけば手続きは簡単ですが、完全に移籍となるとある程度の手順を踏むことになります。そこで社内のイントラネット上にはジョブポスティングがあり、どの国のどんなポジションに空きがあるかわかるようになっています。

「お、これは」というポジションがあれば応募書類を出して人事や上司と話を進めていく形になります。筆者の場合も仕事の引き継ぎなどいろいろとあり、異動まで1年弱かかりましたが、思ったよりもスムーズでした。同じ会社内で築いてきた経験や人脈を無駄にすることなく異動できる、というのはキャリアパスとしても良かったと思います。

シンガポールのお国事情

さてここからシンガポールと周辺東南アジアのお話を少ししたいと思います。シンガポールというと南国、ドリアン、マーライオン、Marina Bay Sandsあたりが有名かと思いますがだいたいそんな感じです。マレー半島が指だとすると、シンガポールは「あ、そろそろ爪切るかな」というときの爪の先のような小さな国なので見どころというのはそんなに多くありません。逆に言うと数日もあればだいたい普通の観光は完了できる便利な国とも言えます。気候は暖かく過ごしやすいところですが雨季にはゲリラ豪雨が日に一度ドカッと来てサッと止みます。夏の気候が好きな筆者としては気に入っています。食事は外食産業が発達していて値段も安価なため太るのが困りものですが……。

人口の約1/3が外国人というシンガポールは国際色豊かで個人的には楽しいところです。最近は貧富の差も広がり、生粋のシンガポール人にとっては仕事が外国人に奪われてしまうという現実もありますが、人口の少ないシンガポールにとっては移民受け入れも国策の1つであり、シンガポールをユニークな国にしているといえると思います。

また、シンガポールは外資系企業を誘致するために税制が優遇されており、アジアのヘッドクオーター(HQ)をシンガポールに置くケースが少なくありません。Red Hat以外にも大手のIT企業のアジアHQをシンガポールに置いている企業がたくさんあります。筆者の日本人の友人にも日本のポジションをアジアに拡大してシンガポールに移住してきた人がいます。もちろん一足飛びに本国の本社で仕事、というのも良いと思いますがやはり今熱いアジアでの仕事が個人的には面白いと思います。

シンガポールビジネスのチャンスとは

それ以外にシンガポールの利点としてはアジア圏のどこにでも行きやすいという点が挙げられます。日本までは飛行機で6〜7時間といったところですがアジア圏に出張の多い身としては短時間でいろいろな国に行けるのは大変便利です。日本以外で出張が多いのが——ご近所様のマレーシア、インドネシア、タイなど——ですが、どこも1〜2時間で行けます。

インドネシアやマレーシアの成長率は非常に高く、注目されていますが出張に行くと、さらにその勢いを肌で感じます。イベントなどで講演をする機会もあるのですが、そこでお会いする技術者の皆さんのレベルも高くなってきています。現在アジア内でトップレベルの成長率というのも首肯(しゅこう)できます。彼らはオープンソースソフトウェアに対する造詣も深く、政府レベルでの推進が進んでおり、場合によっては日本よりも先を行っているケースもあるようです。

写真1　向かって右のビルにシンガポール支社のオフィスがある

お客様の案件もUNIXからLinuxへの移行案件など、日本でもよくあるものなのが面白いところです。プロダクトマネージメントの仕事ということもあり、アジアの他国の技術者や顧客から「今こんな案件を扱ってるんだけど事例ない？」と聞かれることがあります。

多くの場合日本でも似たような案件があり日本での経験が役に立つのですが、筆者の技術力では全面的にサポートして差し上げることができず、いつも「日本の経験のある技術者がいればなあ……」と思うことしきりです。

エンジニアが活きる海外の職場

そういうことがあるたびに日本の優秀な技術者の皆さんはすでに世界で活躍できるレベルにあること、そのための土壌はもうできており、あとは皆さんが思い切って一歩を踏み出せば良いだけ、ということを実感します。

日本の優秀なエンジニアの皆さんがさらに海外でも活躍してくれることを願ってやみません。SD

第49回 Ubuntu Monthly Report

最新のAPUに最新のUbuntuを

Ubuntu Japanese Team
あわしろいくや　AWASHIRO Ikuya
Mail ikuya@fruitsbasket.info

今回は1月に発売されたAMD APU A10 7700KをUbuntu 14.04（の開発版）にインストールし、活用する様子を報告します。

APU

APUはAMDの造語で、“Accelerated Processing Units”の略です。要するにCPUとGPUを統合したものであり、先日発売されたSony PlayStation 4にも搭載されていたりもします[注1]。現行のAMDのマイクロアーキテクチャはおもに3つあり、うち1つはAMD FXというCPU向けのPiledriverで、もう1つはその後継のSteamrollerです[注2]。今回紹介する「AMD APU A10 7700K（以下A10-7700K）はこれを採用しています。残るJaguarがPlayStation 4で採用されており、要するに今回取り上げるA10-7700Kとは別のマイクロアーキテクチャです。

Steamrollerを採用したAPUのコードネームがKaveriで、Jaguarを採用したAPUのコードネームがKabiniと、何を考えてこう名付けたのか首を傾げてしまうほど似通っています。

3月中旬現在Kaveriは2種類発売されていて、A10-7700Kとその上位のA10-7850Kです。後者はA10-7700Kを購入したころには品薄で入手困難でしたが、本誌が店頭に並ぶころには品切れ感は解消されていることでしょう。

Kaveriの特徴はいくつかあって、まずは28nmプロセスルールで製造されていることです。AMDは数年前に製造部門を分社化してGLOBALFOUNDRIESという企業が生まれましたが、Kaveriはここで製造を行っています。一方KabiniやPlayStation 4で採用しているAPUはTSMCで製造しています。前の世代は32nmプロセスルールだったので進歩はしていますが、Intelは22nmプロセスルールであり、王者はさすがの貫禄だなと思うわけです。

おもしろい機能として、Configurable TDPというのがあります。A10-7700KはTDP 95Wですが、BIOS（UEFI）がこのConfigurable TDPに対応していると、65Wと45Wに変更できます。消費電力が減るのは当然のこととして、それに伴ってどのぐらい性能が落ちるのかをテストしてみました（詳細後述）。

前の世代と比較して一番大きな特徴はAPU内部のCPUとGPUの統合がより進み、HSA（Heterogeneous System Architecture）に対応したことです。HSAが何かというのはここでは解説しませんが[注3]、LibreOffice 4.2はこのHSAに対応したアプリケーションのうちの1つです。HSAを使用するためにはGPUドライバーのサポートが必須で、もちろん最新のバージョンでなくてはいけません。となると、LibreOfficeもGPUドライバーも最新版が使えるUbuntu 14.04をインストールするのは必然、というわけです。

注1）日本では未発売のMicrosoft Xbox Oneも同様です。

注2）将来的にSteamrollerを採用したAMD FXシリーズが発売されるかもしれませんが、とりあえず置いときます。

注3）CPUの処理をGPUに渡したり、またその逆がよりスムーズにできるようになったという理解でいいのではないかと思います。

構成

表1のような構成にしました。新規に購入したのはAPUとマザーボードだけであり、メモリは在庫品[注4]、SSDとケースは使い回しです。

今回マザーボードはASRockのFM2A88X-ITX+にしました。前述のとおりConfigurable TDPはBIOSサポートが必須ですが、アップデートによってA10-7700Kにも対応したことがわかったからです[注5]。購入前に注意すべきは、BIOSがKaveriに対応したものになっていないと起動すらしないということです。PCパーツショップで事前にアップデートしてもらうか、手持ちのSocketFM2 APUでアップデートする必要がありますが、後者は割に非現実的です[注6]。ちなみに筆者はPCワンズ[注7]というPCパーツショップに行き、BIOSアップデート済みのマザーボードを購入しました。このようなサービスを行っているPCパーツショップで購入するのがお勧めです。

購入した時点のBIOSのバージョンは2.10でしたが、このバージョンだとA10-7700KのConfigurable TDPには対応していません[注8]。ですので2.20以降にアップデートする必要がありますが、今はとても簡単に行えます。LANケーブルが接続されていればDHCPでIPアドレスを取得してBIOSのアップデートを確認し、自動的に取得する機能があるので、今回はそれを使用しました。

W3U1600F-4Gは型番からも推測できるとおりDDR3-1600のはずなのですが、1333で認識されてしまいました。BIOSからこの修正も簡単にできます[注9]。

mini-PCIeスロットには無線LANとBluetoothがセットになったモジュールが接続されており、Ubuntu 14.04ではAR9462で認識していました。IEEE 802.11aに対応していて巨大なファイルをやりとりしないのであれば有線LANは不要なくらいですが、アンテナの取り付けは困難でしたのでお気をつけください。このモジュールを取り外せば、mSATAのSSDも接続できます。

表1　テストしたマシン構成

APU	AMD A10-7700K
マザーボード	ASRock FM2A88X-ITX+
メモリ	CFD販売 W3U1600F-4G
SSD	Samsung SSD 830 MZ-7PC256N/IT※
ケース	IN-WIN IW-BP671B/300 相当品

※ もともとはクレバリーという自作PCパーツショップの専売品だったようで、その時期に購入したのですが、残念ながら倒産してしまい、現在はIN-WINブランドで販売されています。

Ubuntu 14.04のインストール

Ubuntu 14.04のリリース予定日は4月17日、すなわち本誌の発売予定日前日です。予定どおりにリリースされても、日本語Remixはまだである可能性が極めて高いタイミングであり、また本誌来月号で特集が予定されているのでアップグレードはまだちょっと待ったほうがいいかもしれませんが、新規インストールであればとくに問題ないでしょう。というか多少の不具合があったとしてもAPUの機能をフルに引き出せるほうがいいでしょう。

インストールイメージはUbuntu 12.04でダウンロードし、[ブータブルUSBの作成]で転送します。この際[シャットダウン時に、すべての変更を破棄する]を選択するといいでしょう[注10]。USB 3.0のUSBメモリを使用すれば、転送もインストール自体もあっという間に終わります。

インストールを完了し(**図1**)、各種アップデートの後、[システム設定]-[ソフトウェアとアップデート]-[追加のドライバー]タブから[注11]プロプライエタリなドライバーをインストールしてください(**図2**)。再起動するとこのドライバーが有効になります。

注4) 一昨年の年末、メモリなど各種PCパーツが値上がりすることを見越し、現在の1/3近い価格で購入しています。

注5) ほかにも対応しているマザーボードはあるはずです。

注6) 筆者ですら所有していません。

注7) http://1-s.jp/

注8) A10-7850Kは対応しています。

注9) 今回はとくに表記がない限り1333でベンチマークを計測しています。

注10) 以前いろいろトラブルがあり、イメージの転送が終わらないということがありました。それが理由で[データ保存領域を確保し、行われた変更を保存する]を使用しないようになりましたが、現在はこのトラブルがなくなっているかもしれません。

注11) もちろん画面右上のインジケーターに告知アイコンが表示されている場合は、そちらからでもいいです。

LibreOffice 4.2のHSA

Calcを起動して、[ツール]-[オプション]-[LibreOffice Calc]-[数式]の[詳細な計算の設定]を[ユーザー定義]にし、[詳細]をクリックして[一部の数式の演算にOpenCLを有効にする]をクリックしてください。この方法は本連載第47回でも紹介したのですが、まずは同じものを再掲します(**図3**)。これはA8-3820というAPUを使用して撮影したものです。世代的にはA10-7700Kの3世代前です。[周波数]が[2500]、[演算ユニット]が[4]、[メモリー(MB)]が[7459]というところから、APUのCPU部分に関する表示であることがわかります。

一方GPU部分の表示はありません。**図4**はNVIDIA GeForce GT 640での状態であり、どう見てもGPUの表示です。これがKaveriだとどうなるのかというと、**図5**と**図6**のようになります。**図3**はAPUのGPU部分であり、**図6**はAPUのCPU部分です。当然**図5**の設定のほうが速いはずです。まさにこれこそがHSAの目に見える特徴ということです。それと、CPU部分の演算ユニットが4、GPU部分のそれが6、というのも興味深いです。AMDはこれらをまとめてCompute Coreと呼んでおり、パッケージには[10 Compute Cores(4 CPU + 6 GPU)]と書かれています。

残念ながら筆者はCalcをそれほど活用しているわけでもなく、またベンチマークと行ったものも存在しないので、どのぐらい速くなっているか定量的に計る手段がないのは残念です。

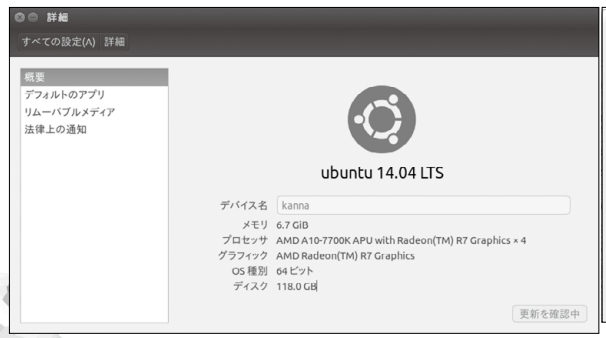
CPU部分の速度とConfigurable TDP

前述のとおり筆者にとっては久しぶりに購入したAPUですので、CPU部分がどのぐらい速くなっているのか興味があります。また、Configurable TDPによってTDPを下げるとどのぐらいの影響があるのかも計ってみたくなりました。ベンチマークの方法もいろいろありますけど、今回はLibreOfficeのビルド時間を計測することにしました。具体的には次のよ

図1　ソフトウェアとアップデート画面

[追加のドライバー]タブで[AMDグラフィックアクセラレーター用のビデオドライバーをfglrxから使用します(プロプライエタリ)]にチェックを入れて、[変更の適用]をクリックしてください

図2　[システム設定]-[詳細]の概要



プロセッサとグラフィックで内容が重複しているのが興味深いです

図3　本連載第47回で紹介した。A8-3820での結果

うなコマンドを実行します。

```
$ sudo apt-get build-dep libreoffice
$ apt-get source libreoffice
$ cd（ソースコードがあるフォルダ）
$ time dpkg-buildpackage -r -uc -b -j6
```

-j6はスレッド数×1.5から決定しています。2スレッドであれば-j3、8スレッドであれば-j12です。ビルドしたLibreOfficeのバージョンは検証時期の関係で4.2.1です。1秒未満は四捨五入しています。結果は**表2**をご覧ください。

Pentium G3220はOSが違う[注12]のであくまで参考値ですが、それにしても6,000円ほどで売られているCPUに負けているというのはいささかショックでした。A8-3820と比較しても30分ほどしか速くなっていないのもかなりのショックだったのですが。ただ、TDPを落としても誤差ほどの違いしか出なかったのはなかなか興味深いです。今回のベンチマークはGPU部分をまったく使っていないので、こちらに回される電力をCPU部分に振り分けた結果、CPU部分の速度が落ちなかったのではないかと推測できます。CPUとGPUどちらも使うベンチマークだった場合は速度が落ちるはずですし、確かにいくつかのベンチマークを見た限りでもそのような結果になっていました。

メモリの速度を上げてみると誤差とは言えないほどの差が出ており、Kaveriを使用する場合は可能な限り高速なメモリを使用するのがいいということがわかりました。

表2　LibreOffice 4.2.1のビルドにかかった時間

APU/CPU	かかった時間
A8-3820	217分33秒
A10-7700K（TDP 95W）	186分22秒
A10-7700K TDP 65W	185分41秒
A10-7700K TDP 45W	186分14秒
A10-7700K TDP 45W DDR3-1600	179分31秒
Pentium G3220（参考値）	172分27秒

どういう人が買うべきなのか

今回はGPU部分の検証は行いませんでしたが、巷のベンチマークを見る限りではかなり強力ですので、UbuntuもサポートしているSteam[注13]のゲームをするにはなかなかよさそうですが、それ以外は筆者のようなAMDにロマンを感じる人でないと厳しいかな、というのが正直な感想です。でもそれでいいのではないでしょうか。大事です、ロマン。SD

注12） Ubuntu 12.04で計測したほか、HDDも別など計測環境が異なるため。

注13） http://store.steampowered.com/?l=japanese

図4　NVIDIA GeForce GT 640での結果

図5　A10-7700KのGPU部分

図6　A10-7700KのCPU部分

第26回

Linux 3.14のコードネームとlockdep機能

Text：青田 直大　AOTA Naohiro

Linux 3.14-rc7がリリースされ、3.14のリリースも間もなくではないかと思われます。Linusも-rc7のリリースメール[注1]で、“but with some luck thatwon‘t happen and this is the last rc.”と書いているとおり、3月23日には3.14がリリースされているかもしれません。今月はLinux 3.14のコードネームについてと、userlandでも使えるようになったlockdepという機能について解説します。

3.14のコードネーム

Linux kernelのトップレベルのMakefileを見ると、“NAME”というものが設定されているのに気がつきます。これはいくつかのバージョンごとに付けられる愛称のようなもので、たとえばLinux 3.11では“Windows for Workgroups 3.11”とかけて“Linux for Workgroups”と名付けられていました。このコードネームを3.14ということで、円周率にちなんだ名前にしてくれとのリクエストがLinusのもとにもたくさん届けられていたようです。しかし「円周率なんて20ケタぐらい昔から覚えてるでしょ。そしたら3.14なんて全然円周率に近くないよね。」と言って退けてしまいました。その代わりに3.14には“Shuffling ZombieJuror”というコードネームが付けられました。

lockdep

ロックはときにやっかいな問題をひき起こします。単純な例であれば**リスト1**のコードのように、AとBというつ2つのロックがあって片方のスレッド(locktest-f)がAのロックのあとにBのロックを取ろうとする一方で、もう1つのスレッド(locktest-g)がBのあとにAを取ろうとすれば、互いにどちらのロックも取ることができないデッドロックが引き起こされます。これぐらいのコードであれば、カーネルがハングしてしまってもどこに原因があるのかすぐにわかりますが、コードが複雑になってくると、なぜカーネルがハングしたのかを特定することは難しくなってしまいます。

こうした問題への対抗策として、Linux kernelにはlockdepというしくみが導入されています。これはロックの依存関係を追跡し、デッドロックを検出した場合にどのようなロックが行われていたかなどの情報をダンプしてくれる機能です。ダンプを見ることでロック周りの問題の解決が容易になります。

注1）https://lkml.org/lkml/2014/3/16/166

たとえば、**リスト1**のプログラムを実行すれば**図1**のような出力を得ることになります。

❶循環依存しているロックの取り方をしていると表示されています。

❷では“locktest-g”というスレッドが“lockA”を取ろうとしているが、❸すでに“lockB”というロックをこのスレッドが取っていることを示しています。

❹この“locktest-g”におけるロックの取り方と整合しない、ほかのロックの取り方が示されています。ロックの依存関係が逆順に表示されていて、“locktest-f”のスレッドで行っている“lockA”を取って、“lockB”を取る操作と対応しています。このようにロックを取る順番が一致していないときに、その情報を出力してくれるわけです。

❺また、「CPU0がlockBを取ったあとに、CPU1がlockA、lockBを取得し、CPU0がlockAを取得しようとするとデッドロックが起きるよ」とロックがおかしくなる状況の例示までしてくれます。

❻最後に、この時点で取得されているロックの一覧と、スタックトレースとが表示されています。

lockdepのしくみ

lockdepでは個々のロックにそれぞれ1つのIDを与えています。このIDを用いて新しくロックを取るときに、すでに同じIDのロックを取っていないか、循環依存が含まれていないかなどをチェックします。さらに、ロックのIDを使ってロックのチェーンのハッシュ値を計算します。チェーンの検証後ハッシュ値を覚えておくことで、同じチェーンの検証はスキップし、検証の負荷を抑えるようにしています(**図2**)。

userland lockdep

さて、このようにロックの問題を見つけ出し、

▼リスト1　デッドロックを起こすコード

```
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/init.h>
#include <linux/kthread.h>
#include <linux/spinlock.h>

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Naohiro Aota");
MODULE_DESCRIPTION("Lock test");

static struct task_struct *t1, *t2;
DEFINE_SPINLOCK(lockA);
DEFINE_SPINLOCK(lockB);

static int f(void *data)
{
  while (!kthread_should_stop()) {
    spin_lock(&lockA);
    spin_lock(&lockB);
    pr_info("locktest: f n");
    spin_unlock(&lockB);
    spin_unlock(&lockA);
    schedule_timeout_interruptible(HZ);
  }
  return 0;
}

static int g(void *data)
{
  while (!kthread_should_stop()) {
    spin_lock(&lockB);
    spin_lock(&lockA);
    pr_info("locktest: g n");
    spin_unlock(&lockA);
    spin_unlock(&lockB);
    schedule_timeout_interruptible(2*HZ);
  }
  return 0;
}


static int __init lock_init(void)
{
  t1 = kthread_run(f, NULL, "locktest-f");
  t2 = kthread_run(g, NULL, "locktest-g");
  return 0;
}

static void __exit lock_cleanup(void)
{
  kthread_stop(t1);
  kthread_stop(t2);
}

module_init(lock_init);
module_exit(lock_cleanup);
```

▼図1　lockdepの出力

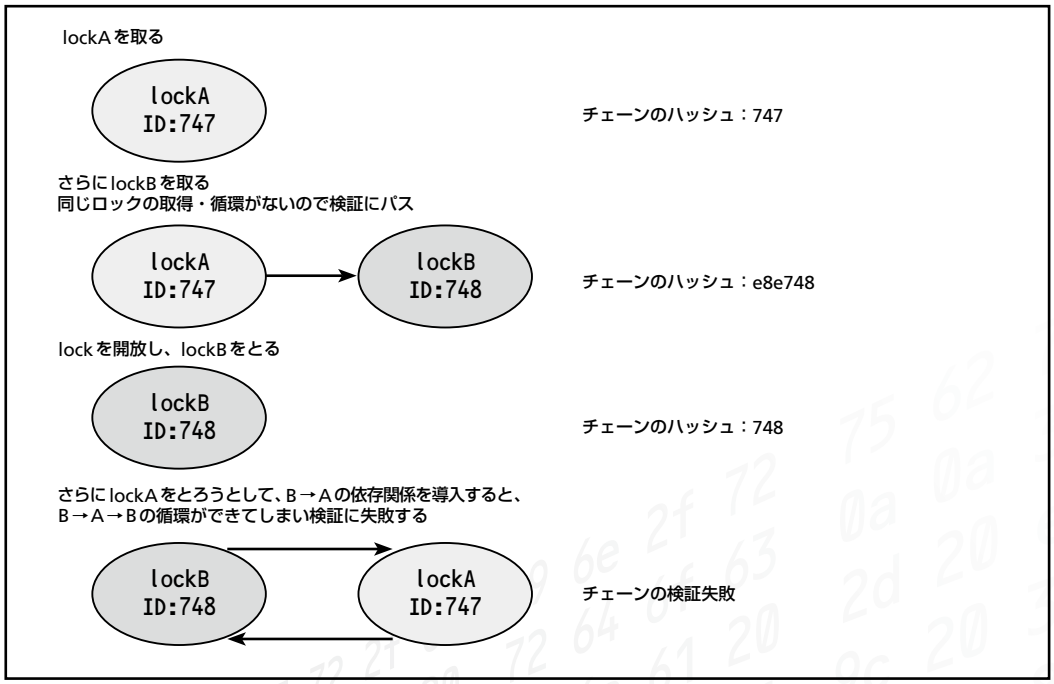
```
[ 6087.543426] ============================================
[ 6087.543428] [ INFO: possible circular locking dependency detected ]    ❶
[ 6087.543432] 3.14.0-rc7+ #338 Tainted: G        W  O
[ 6087.543434] --------------------------------------------------------
[ 6087.543436] locktest-g/19086 is trying to acquire lock:                ❷
[ 6087.543439]  (lockA){+.+...}, at: [<ffffffffa02c1028>] g+0x28/0x60 [locktest]
[ 6087.543450]
but task is already holding lock:                                         ❸
[ 6087.543452]  (lockB){+.+...}, at: [<ffffffffa02c101c>] g+0x1c/0x60 [locktest]
[ 6087.543460]
which lock already depends on the new lock.

[ 6087.543463]
the existing dependency chain (in reverse order) is:                      ❹
[ 6087.543465]
-> #1 (lockB){+.+...}:
[ 6087.543471]        [<ffffffff810e7073>] lock_acquire+0x93/0x130
[ 6087.543477]        [<ffffffff81790ac0>] _raw_spin_lock+0x40/0x80
[ 6087.543482]        [<ffffffffa02c1088>] f+0x28/0x57 [locktest]
[ 6087.543486]        [<ffffffff810b79bf>] kthread+0xff/0x120
[ 6087.543492]        [<ffffffff817998ec>] ret_from_fork+0x7c/0xb0
[ 6087.543497]
-> #0 (lockA){+.+...}:
[ 6087.543502]        [<ffffffff810e6553>] __lock_acquire+0x1773/0x1ae0
[ 6087.543506]        [<ffffffff810e7073>] lock_acquire+0x93/0x130
[ 6087.543509]        [<ffffffff81790ac0>] _raw_spin_lock+0x40/0x80
[ 6087.543513]        [<ffffffffa02c1028>] g+0x28/0x60 [locktest]
[ 6087.543517]        [<ffffffff810b79bf>] kthread+0xff/0x120
[ 6087.543521]        [<ffffffff817998ec>] ret_from_fork+0x7c/0xb0
[ 6087.543526]
other info that might help us debug this:

[ 6087.543528]  Possible unsafe locking scenario:                         ❺

[ 6087.543531]        CPU0                    CPU1
[ 6087.543532]        ----                    ----
[ 6087.543534]   lock(lockB);
[ 6087.543538]                                lock(lockA);
[ 6087.543541]                                lock(lockB);
[ 6087.543544]   lock(lockA);
[ 6087.543548]
 *** DEADLOCK ***

[ 6087.543551] 1 lock held by locktest-g/19086:                           ❻
[ 6087.543553]  #0:  (lockB){+.+...}, at: [<ffffffffa02c101c>] g+0x1c/0x60 [locktest]
[ 6087.543561]
stack backtrace:
[ 6087.543565] CPU: 4 PID: 19086 Comm: locktest-g Tainted: G        W  O 3.14.0-rc7+ #338
[ 6087.543568] Hardware name: System manufacturer System Product Name/P8H67-M PRO, BIOS 3806
08/20/2012
[ 6087.543570]  ffffffff8254e7b0 ffff8804afcdfc98 ffffffff817885e1 ffffffff8254e7b0
[ 6087.543578]  ffff8804afcdfcd8 ffffffff81783992 ffff8804afcdfd30 ffff88057a468860
[ 6087.543583]  0000000000000000 ffff88057a468828 ffff88057a468000 ffff88057a468860
[ 6087.543589] Call Trace:
[ 6087.543596]  [<ffffffff817885e1>] dump_stack+0x4e/0x7a
[ 6087.543602]  [<ffffffff81783992>] print_circular_bug+0x201/0x210
[ 6087.543606]  [<ffffffff810e6553>] __lock_acquire+0x1773/0x1ae0
[ 6087.543611]  [<ffffffff81791c20>] ? retint_restore_args+0xe/0xe
[ 6087.543616]  [<ffffffff810e7073>] lock_acquire+0x93/0x130
[ 6087.543620]  [<ffffffffa02c1028>] ? g+0x28/0x60 [locktest]
```

（次ページに続く）

（前ページの続き）

```
[ 6087.543627] [<ffffffffa02c1000>] ? 0xffffffffa02c0fff
[ 6087.543631] [<ffffffff81790ac0>] _raw_spin_lock+0x40/0x80
[ 6087.543636] [<ffffffffa02c1028>] ? g+0x28/0x60 [locktest]
[ 6087.543640] [<ffffffffa02c1028>] g+0x28/0x60 [locktest]
[ 6087.543645] [<ffffffff810b79bf>] kthread+0xff/0x120
[ 6087.543651] [<ffffffff810b78c0>] ? kthread_create_on_node+0x250/0x250
[ 6087.543655] [<ffffffff817998ec>] ret_from_fork+0x7c/0xb0
[ 6087.543659] [<ffffffff810b78c0>] ? kthread_create_on_node+0x250/0x250
```

▼図2　lockdepの動作



解決するのに有用な情報を提供してくれるlockdepを、kernelの中ではなくuserlandで使えるように移植したものが、Linux 3.14のtools/lib/lockdepにコミットされています。たとえば**リスト2**のようなデッドロックを起こすコードを書いて、-D__USE_LIBLOCKDEPを付けて、liblockdep.aとリンクしてビルドして実行します(**図3**)。するとkernelのときと同じような出力をuserlandのプログラムでも得ることができることが確認できます。

この方法ではliblockdepに静的リンクを行っていますが、動的リンクを用いた方法も使うことができます(**図4**)。静的リンクのほうは"-D__USE_LIBLOCKDEP"によってmutex.h(**リスト3**)がpthreadのlock関数をコンパイル時に置き換えていて、動的リンクのほうでは"LD_PRELOAD"を用いて"pthread_mutex_lock"を実行時に上書きしています。

まとめ

今回は、userlandでも使えるようになったlockdep機能について紹介しました。Linux kernelでいくつものロック関係の問題を解決してきたというlockdepを、ぜひuserlandのプログラムでも試してみてください。SD

▼リスト2　デッドロックを起こすコード

```
#include <liblockdep/mutex.h>
#include <stdio.h>
#include <unistd.h>

pthread_t t1, t2;
pthread_mutex_t lockA, lockB;

static void *f(void *data)
{
  for(;;) {
    pthread_mutex_lock(&lockA);
    pthread_mutex_lock(&lockB);
    printf("locktest: f n");
    pthread_mutex_unlock(&lockB);
    pthread_mutex_unlock(&lockA);
  }
  return NULL;
}

static void *g(void *data)
{
  for(;;) {
    pthread_mutex_lock(&lockB);
    pthread_mutex_lock(&lockA);
    printf("locktest: g n");
    pthread_mutex_unlock(&lockA);
    pthread_mutex_unlock(&lockB);
  }
  return NULL;
}

int main(void)
{
  pthread_mutex_init(&lockA, NULL);
  pthread_mutex_init(&lockB, NULL);

  pthread_create(&t1, NULL, &f, NULL);
  pthread_create(&t2, NULL, &g, NULL);
  pthread_join(t1, NULL);
  pthread_join(t2, NULL);
  return 0;
}
```

▼図3　コンパイルと実行

```
$ cd ~/linux-2.6/tools/lib/lockdep
$ make
  CC FPIC            common.o
  CC FPIC            lockdep.o
  CC FPIC            preload.o
  CC FPIC            rbtree.o
  BUILD STATIC LIB   liblockdep.a
  BUILD SHARED LIB   liblockdep.so
$ cd ~/locktest
$ gcc -o userlockdep -ggdb -pthread -lpthread userlocktest.c -I../linux-2.6/tools/lib/
lockdep/include -Wall -D__USE_LIBLOCKDEP ../linux-2.6/tools/lib/lockdep/liblockdep.a
$ ./userlockdep
locktest: f
locktest: f
…
locktest: f
locktest: f

=====================================================
[25/1846]
[ INFO: possible circular locking dependency detected ]
liblockdep 0.0.1
-----------------------------------------------------
userlockdep/3700 is trying to acquire lock:
 (&lockA){......}, at: ./userlockdep() [0x400ce7]

but task is already holding lock:
 (&lockB){......}, at: ./userlockdep() [0x400cdd]

which lock already depends on the new lock.


the existing dependency chain (in reverse order) is:

-> #1 (&lockB){......}:
./userlockdep[0x4011f8]
./userlockdep[0x402d53]
./userlockdep[0x402ff0]
./userlockdep[0x40360e]
```

（次ページに続く）

（前ページの続き）

```
./userlockdep[0x4040b8]
./userlockdep[0x404b7f]
./userlockdep[0x400c43]
./userlockdep[0x400ca7]
/lib64/libpthread.so.0(+0x8227)[0x7fe6985fc227]
/lib64/libc.so.6(clone+0x6d)[0x7fe698331c8d]

-> #0 (&lockA){......}:
./userlockdep[0x4011f8]
./userlockdep[0x402667]
./userlockdep[0x402c39]
./userlockdep[0x402ff0]
./userlockdep[0x40360e]
./userlockdep[0x4040b8]
./userlockdep[0x404b7f]
./userlockdep[0x400c43]
./userlockdep[0x400ce7]
/lib64/libpthread.so.0(+0x8227)[0x7fe6985fc227]
/lib64/libc.so.6(clone+0x6d)[0x7fe698331c8d]

other info that might help us debug this:

 Possible unsafe locking scenario:

       CPU0                    CPU1
       ----                    ----
  lock(&lockB);
                               lock(&lockA);
                               lock(&lockB);
  lock(&lockA);

 *** DEADLOCK ***

1 lock held by userlockdep/3700:
 #0:  (&lockB){......}, at: ./userlockdep() [0x400cdd]

stack backtrace:
./userlockdep[0x400efa]
./userlockdep[0x402721]
./userlockdep[0x402c39]
./userlockdep[0x402ff0]
./userlockdep[0x40360e]
./userlockdep[0x4040b8]
./userlockdep[0x404b7f]
./userlockdep[0x400c43]
./userlockdep[0x400ce7]
/lib64/libpthread.so.0(+0x8227)[0x7fe6985fc227]
/lib64/libc.so.6(clone+0x6d)[0x7fe698331c8d]
```

▼図4　動的リンクを用いる方法

```
$ LD_PRELOAD=../linux-2.6/tools/lib/lockdep/liblockdep.so ./userlockdep
```

▼リスト3　mutex.h

```
#ifdef __USE_LIBLOCKDEP

#define pthread_mutex_t         liblockdep_pthread_mutex_t
#define pthread_mutex_init      liblockdep_pthread_mutex_init
#define pthread_mutex_lock      liblockdep_pthread_mutex_lock
#define pthread_mutex_unlock    liblockdep_pthread_mutex_unlock
#define pthread_mutex_trylock   liblockdep_pthread_mutex_trylock
#define pthread_mutex_destroy   liblockdep_pthread_mutex_destroy

#endif
```

May 2014

NO.31

Monthly News from jus

Japan UNIX Society

日本UNIXユーザ会　http://www.jus.or.jp/
法林 浩之　HOURIN Hiroyuki　hourin@suplex.gr.jp

ハッカーへの第一歩、OSSとの正しい付き合い方

今回は、3月に行ったjus研究会東京大会の模様をお伝えします。

東京大会

■オープンソースの利用とハッカー文化
【講師】よしおかひろたか（楽天）
【司会】法林 浩之（日本UNIXユーザ会）
【日時】2014年3月1日（土）15:15～16:00
【会場】明星大学 日野キャンパス 28号館 201教室

■OSSの発展にはライセンスの理解が不可欠

楽天の技術理事を務めているよしおかさんを講師にお迎えし、オープンソースやハッカー文化についての解説と、それらを楽天の社内に広める活動について紹介していただきました。参加者は46人でした。

はじめにオープンソースの概説がありました。1970年代からパブリックドメインやプロプライエタリソフトウェアといった区別があり、1980年代にはフリーソフトウェアが登場しました。オープンソースの概念が定義されたのは、1990年代の後半にNetscape Navigatorがソースコードを公開したころです。現在では、オープンソースソフトウェア（OSS）の利用はごく普通のことになりましたが、ここまで普及した背景には、「従来にない新しいソフトウェアを利用できること」「自分で機能拡張や修正ができること」「そのような改良の積み重ねとしてソフトウェアの品質が高いこと」などが挙げられます。

次に、ソフトウェアライセンスについて多くの時間を割いて解説されました。フリーソフトウェアには2種類のライセンスがあります。1つはcopyleftで、GNUなどが該当します。copyleftは再配布などのときに同じライセンスを表示することが義務づけられています。もうひとつはpermissiveで、MIT、Apache、BSDなどのライセンスはこちらに分類されます。permissiveでは再配布時にコードの公開をしなくても良いなど、copyleftとの違いがあります。

ライセンスの利用状況としてはGPL2が約3割で首位、Apache、GPL3、MIT、BSDなどがこれに続いています。2008年のデータでは7割ぐらいがGPLだったのですが、近年はGPLが減少傾向にあり、代わってpermissiveの類が増加しています。ライセンスの設定は、そのソフトウェアをどう育てていきたいかという意志の表れであるとともに、ソフトウェアを使うだけでコミュニティへの還元を行わないフリーライダーを排除する意図もあります。

しかし、2012年に行われた調査によるとGitHubで公開されているソースコードの77%はライセンス条項が付いておらず、これでは良くないということで、今ではGitHubで公開する際にライセンスをMIT、Apache、GPLから選ぶようになりました。

■エンジニアのかかわり方がOSSの性格や勢いを決める

さらに次の話題として、OSSコミュニティの話がありました。OSSコミュニティの典型的構造は、そのソフトウェアの作者が最上位にいて、その下に数名から数十名のコミッタやメンテナ、さらにその下に多数のコントリビュータ、その下に無数のユーザ

という階層構造です。具体例として、Linuxはコントリビュータが1万人、コミットが460万回、ソースコードが1,600万行であり、Rubyはコントリビュータが90人、コミットが3万回、ソースコードが100万行という規模です。

これらのデータでおもしろいのは、同種のソフトウェアでもコミッタの人数とコミット回数に大きな違いがあることです。クラウド基盤ソフトウェアのOpenStack、CloudStack、Eucalyptusを比較すると、Eucalyptusは少数のコントリビュータがたくさんコミットし、OpenStackは多数のコントリビュータが少しずつコミットしています。これらのデータからプロジェクトの性格やコミュニティの勢いみたいなものも感じ取ることができます。

もうひとつ、OSSを語るうえで欠かせないのがバザールモデルと呼ばれる広域分散協調による開発モデルです。その根底となる考え方であるハッカー倫理の話もありました。ハッカー倫理については、1980年代のハッカーたちについて記された『ハッカーズ』[注1]という書籍に詳しく書かれています。要約すると、ハッカー共通の価値観は「コンピュータは生活を向上させるものである」「ラフなコンセンサスと動くコードが重要」「何かをするときに周囲に許可を求めない」「間違ったことをしたときは謝罪する」といったものです。

■楽天におけるOSSの利用／開発／教育

OSSやハッカー文化の話はここで一段落し、ここからは楽天でのOSS利用についての話になりました。楽天でのOSSの利用動機は、ユーザとしてすでに社内の至るところで使っていたというのもありますが、今後もベンダによる囲い込みから逃れたいという意図があり、積極的にOSSへの参加を行っています。OSSへの関与の成熟度として、発見→使う→コミュニティに参加→コミュニティを作る、という段階があります。現在の楽天は「使う」と「参加」の中間ぐらいの段階にあるだろうとのことです。楽天でもOSSの開発を行っており、分散KVSの「ROMA」やファイルシステム「LeoFS」などの開発事例があります。

社内でOSSを利用および開発するときは、OSSライセンスを遵守するよう注意する必要があります。何も考えずに使うと著作権違反や特許侵害などの恐れがあるため、とくにアプリケーションや電子書籍リーダーなどクライアントに配布するソフトウェアのコードを書くときは注意が必要です。また、MongoDBなどAGPLで公開されているソフトウェアは改変したコードも公開しなければならないので、サーバサイドで利用するソフトウェアについてもライセンスは要確認です。楽天ではライセンスに関するトレーニングやマニュアルを作って教育しているとのことで、併せて特許や著作権などの教育も必要だそうです。

最後にまとめとして、「こういう場を通してエンジニア共通の価値観を共有していきたい」「OSSコミュニティはイノベーションのエンジンになっている」「みなさんもハッカーになり、より良い社会を作っていってほしい。そして自分は楽天でその手伝いをしていきたい」というメッセージで講演を締めくくりました。

◆ ◆ ◆

今回の研究会もオープンソースカンファレンスの中で開催しました。同じ時間帯に人気のセミナーが多く、集客に苦戦するのではないかと心配しました。しかし、実際には予想以上に多くの方に参加していただき、これからOSSの世界に入る若い人への教材となるような講演を提供できてよかったと思います。

今回の講演はよしおかさんの手により資料が公開されています。また動画も公開していますので、時間がありましたらご覧ください。SD

- 資料：http://www.slideshare.net/hyoshiok/using-oss-and-hacker-culture-at-an-internet-company-osctokyo-20140301rev
- 動画：http://www.youtube.com/watch?v=BJD9eb-dLV4

注1） Steven Levy 著、古橋芳恵、松田信子 訳、工学社、1990年

Hack For Japan
エンジニアだからこそできる復興への一歩

● Hack For Japan スタッフ
佐々木 陽　SASAKI Akira
Twitter @gclue_akira
佐伯 幸治　SAEKI Koji
Twitter @widesilverz

第29回 街をハックする「Hack For Town in Aizu」開催!

“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」。今回は街をハックするという新しい試みのHack For Townというイベントについて紹介します。

先端テクノロジを街で活かすためのハッカソン

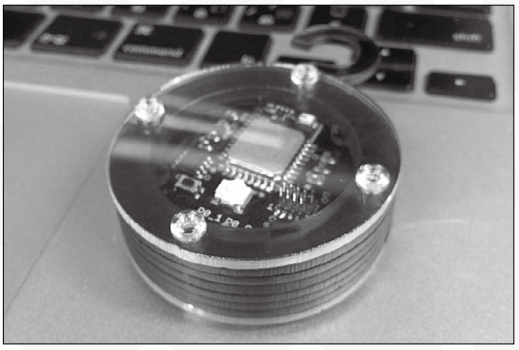
2月15日、16日にHack For Japan主催による「Hack For Town in Aizu」が開催されました。Hack For Townとは、最先端テクノロジを用いて街中に設置された最先端ハードウェアをハックするという主旨のイベントで、将来的には最先端テクノロジを用いて新しい街を創造することを目指しています。

その第1回目の場所が福島県会津若松市となり、Hack For Town in Aizuとして開催されました。会津若松市には、会津若松駅からクルマで5分ほどのところに神明通りという商店街があります。今回はこの神明通りの各所にBeaconデバイスを設置[注1]し、商店街という環境ならではのロケーションを活かしたHackathonに、参加者たちがチャレンジしました(**写真1、写真2**)。

またBeaconデバイス以外にも、スマートフォンやタブレットで操縦可能な4翼ヘリコプターのガジェットAR.Drone 2.0、パーソナル向けで小型・軽量タイプの3Dプリンタ、バーチャルな空間をよりリアルに楽しめるヘッドマウントディスプレイのOculus Rift、名刺サイズの格安PCボードであるRaspberry Pi、人感センサ・照度センサ・音センサ・温度センサ・湿度センサ・気圧センサを搭載した環境センサといったハードウェアを用意しました(**写真3**)。それらに加え、市内企業が保有しているfabスペース、APIとしては会津若松市に関するオープンデータなどが案内され、さまざまなハードウェアやデータを組み合わせた開発ができるような環境が用意されていました。

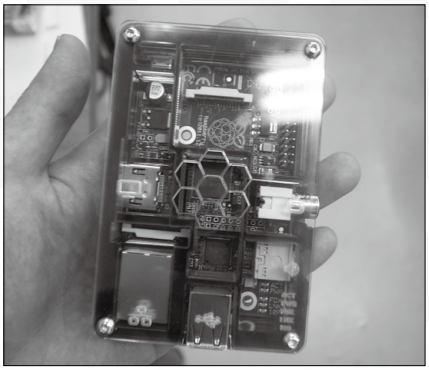
◆**写真1　神明通りとポールに設置されたBeacon**

◆**写真2　今回使用したDIYのBeacon。ポールに引っ掛けるためのフックが付いている**



注1　https://mapsengine.google.com/map/edit?mid=z-JddsFmwUAw.k9yiocwcHk0k

◆写真3　環境センサ（写真左上）、Raspberry Pi（写真右上）、AR.Drone 2.0（写真右下）といったデバイスも用意



こういったHack For Town in Aizuの事前の情報はwikiを使ってまとめられ[注2]、参加者はこちらで概要をつかみ、どんなことができるかを前もって検討しておくことができるようになっていました（**図1**）。

大雪の影響から開始を遅らせてのスタート

Hackathon当日。この日は雪害となる大雪で交通機関に影響が出ており、Hackathonの運営にも大きな支障をきたしました。そこで1日目の午前中に予定していた開始時間を遅らせてスタート。午前中は参加者の来場状況を見ながらHackathonを進めていくことを決定し、予定されていた参加者からのアイデアピッチを午後からとして、Hack For Japanスタッフであり、今回のイベントの提案者でもある佐々木陽からBLE（Bluetooth Low Energy）やiBeacon[注3]についてのレクチャーがなされました。

次に、Hack For Japanスタッフの及川卓也からHack For Townの説明[注4]があり、合わせてオープンデータを用意していることも伝えられました。具体的に案内されたオープンデータは、「会津若松市内のバス停データ」、「会津のバス運行情報（時刻表）」、「会津若松市内の消防水利位置情報」、「会津若松市の公共施設マップデータ」、「毎月大字別人口」、「月別1歳毎年齢別人口」、「開発プロバイダ向けクラウドサービス」といったものです。オープンデータについては「現状、必ずしも必要なデータがそろっているわけではありません。もし『こんなデータが公開されていたらこんなことができるのに』というアイデアがあれば、データが公開されているという前提で開発をしてください。日本全国すべての自治体がデータをオープンにすることに対して積極的に取り組んでいるわけではありません。なかには、データが公開されていたらどんなことができるかというイメージを持てないために、オープンデー

◆図1　Hack For Townのwikiにあるbeaconのページ。さまざまな情報を集約

注2　http://www.hack4town.org/wiki/index.php?Hack%20for%20Town
注3　iBeaconは、Apple Inc.の登録商標です。コラム参照。
注4　http://www.slideshare.net/skkj2014/hack-fortowninaizu

タに取り組んでいないというケースも考えられます。Hackathonでは『住民の方に使ってもらえる便利なサービスを作るためにこういったデータの公開が必要』といったアプローチも大歓迎です。データがないからといってあきらめないでチャレンジしてください」とその重要性にも触れていました。

Column iBeaconについて

iBeaconとは、AppleがiOS 7から搭載したBluetooth Low EnergyベースのMicro Locationのしくみです。iBeaconはCoreLocation APIを参照しますのでロケーション用のしくみになっています。

Bluetooth Low EnergyはBluetooth 4.0からLow Energyという新しい規格が取り入れられたもので、それまでのBluetoothにおいて懸念されていた電力消費の大きかった点が見直され、たとえばスマートフォンの電力消費改善にもつながっています。最近ではBluetooth Low Energyに対応したハードウェアが出てきており、今後のトレンドになるとも言われていますが、そのしくみの1つがiBeaconとなります。

iBeaconにはUUIDを設定します。iOS上でiBeaconのUUIDがマッチした場合、minor ID（2バイト $0000～$ffff=0-65535）、major ID（2バイト $0000～$ffff=0-65535）、rssi（受信信号 dBm）、proximity（Beaconとの距離／Immediate、Near、Far、Unknownで定義）がアプリで取得できます。

イベントとしてはiBeaconを見失ったときに呼び出されるdidExitRegion、iBeaconを発見したときに呼び出されるdidEnterRegion、常時まわりにあるiBeaconをスキャンするdidRangeBeaconsがあり、アプリがバックグラウンドにあろうと、フォアグランドにあろうとイベントを取れます。

このしくみを使って何ができるかというと、たとえば、お店に入った時点でポケットに入れておいたiPhoneがiBeaconを発見して、何かしらのイベントが発生するといった使い方ができます。サーバに接続するロジックを入れておくとログも取れますので、オフィスにiBeaconを設置しておいて、iBeaconを認識したら出勤・見失ったら退勤といったような出勤表を作ることもできます。アメリカのアップルストアではiBeaconとアップルストアのアプリを使って、アップルストアに入ると場所ごとにiPhoneの画面にさまざまな情報が案内されるサービスも提供されています。

最後にフリービット社の渡邉知男氏から環境センサについての説明[注5]がなされました。この環境センサは佐賀県唐津市と同社による「高齢者向け安心見守り・健康相談システム」の実証実験に使用されているものでもあるそうです。

午後から本格的にHackathonに入りました。参加者各々が考えているアイデアを披露し、興味を持ったアイデアに参加するといった形でチーム分けを行った後、開発が始まりました（この日のアイデアピッチについてはYouTube[注6]にてご覧いただけます）。2日目の午前中も前日から引き続いてのHackathonを行い、夕方に成果発表となりました。

数々の街ハックのアイデアが生まれる

今回のHackathonでは以下のような成果が発表されました[注7]。

●サックマンのパックマン

街中に設置したBeaconをエサに見立てたAR版パックマン。ゲームとしても、スタンプラリーとしても活用できることを目指したもの。

●Anko-Kivy

KivyというPythonで記述するNUI（Natural User Interface）でフレームワークを作成。BLE、Beaconをマルチプラットフォームで使えるようにする。

●BoarDrone

利用者の持っているAndroidでBeaconの信号を受信すると、その場所にDroneが飛んできて案内をするというもの。

●iMenu

最寄りのBeaconを検知してユーザがいる店舗の情報を取得し、メニューを表示するシステム。

●障害者のための障害物検知アプリ

目の不自由な方に対して、たとえば滑りやすい道や工事中で通れない道、段差のある道があった際に音声で通知してくれるアプリ。

注5　http://www.slideshare.net/tomowatanabe/hack4-town
注6　http://www.youtube.com/playlist?list=PLjaU4dlb6AW5Arc9NyqBODNTCBs0upWkQ
注7　成果発表や表彰式は動画で見られます。http://www.youtube.com/playlist?list=PLjaU4dlb6AW5Arc9NyqBODNTCBs0upWkQ

●消えたプリンセスを探せ！

神明通りの各所に設置されたBeaconから情報を得て、神明通りに逃げ込んだキャラクターを探すアドベンチャーゲーム。

●一緒に歩こう

Beaconの設置してある場所に行くと女性の音声が流れる。設定された速度で歩かないと声が遠ざかる。二次元キャラとリアルに歩く感覚を追求したアプリ。

●会津クライシス

Googleストリートビューの神明通りを取り込んだDroneを使ったシューティングゲーム。向かってくるゾンビを打ち倒す。

●トロッコ列車

神明通りをトロッコが走る3DゲームをUnityを使って目指した。

●動物レストラン

親子連れをターゲットに商店街に行くきっかけを作るためのアプリ。商店街に設置したBeaconを果物のエサと見立てて、その果物を動物に見立てたiPhoneが食べるというもの。

●ぴかり

Beaconのハード制御によって通信でLEDを光らせる。特定のiPhoneが近づいたときにBeaconが光るほかに、音声を流してしゃべっているように見せることもできる。

発表では室井照平 会津若松市長、会津大学 次期理事長兼学長 岡嶐一氏といった方に審査委員として参加いただき、各発表に対して評価がなされ、市長賞「BoarDrone」、Hack For Japan賞「障害者のための障害物検知アプリ」、会津ゲームLAB賞「消えたプリンセスを探せ！」という結果となりました。

ITの実証実験ができる場づくりを

今回、街をハックするというはじめての試みで開催されたHack For Town in Aizuでしたが、数多くの方々からのご協力の下、運営がなされました。市役所の方々が運営のサポートに入ったこともお伝えしておきます。開催場所も市役所の会議室や会津若松市生涯学習総合センターを提供いただきました（**写真4**）。

◆写真4　会津若松市役所に貼られた案内

会津若松市では、たとえば年齢別人口や町・大字別人口といった統計データ、消火栓位置情報や公共施設マップの公開などを実施することで、オープンデータの普及・啓発活動を積極的に行っており（注8）、今回のようなITイベントについても理解・協力が得やすい環境が整っていると実感しました。

「日本には実際のフィールドでセンシング、IT機器、クラウド連携を試せる場所がない。アメリカのサウスバイサウスウエストのようなイベントが日本であってもいいんじゃないか。『会津若松に来れば2日間だけは街中を自由にハックできます』といったITの実証実験ができるような、そういったしくみをつくるためにHack For Townを開催することにしました。地方も東京も同じですが、商店街に人が来ませんとか、お年寄りが増えていますとか、そういった根本の問題を解決するきっかけにもHack For Townはなるんじゃないか」

スタッフ佐々木のそんな想いからはじまったHack For Town。実行委員会も立ち上げる予定ですので、またイベントが開催されることと思います。次回のHack For Townで皆様にお会いできるのを楽しみにしています。SD

注8　会津若松市 オープンデータの取り組み。http://www.city.aizuwakamatsu.fukushima.jp/docs/2009122400048/

温故知新
IT むかしばなし

第33回

ハードディスクと接続インターフェース

杉田 正　SUGITA Tadashi　Twitter:@sugipooh

はじめに

今回はハードディスクと接続インターフェースのお話です。

IBM PC

1981年IBM PCが発売されました。IBM PCはインターフェースをオープンにしたため、さまざまな周辺機器がサードパーティから発売されました。その中のコンパック社が開発したのが、拡張スロットに接続インターフェースとシーゲート社製ハードディスクを一体化した5MBハードディスク装置[注1](以後HDD)でした。

汎用機やワークステーションで使われていた13インチや8インチのHDDは非常に高価でありながら、運搬中や使用中に簡単に"壊れる"ことでも有名でした。

コンパック社のHDDは、インターフェースボード上にドライブを搭載しIBM PCに内蔵できる構造にして、取り扱いを簡単にした製品でした。

フロッピーディスクと比べて外部記憶へのアクセススピードを画期的に高速化した、このコンパック社の5インチHDDは大ヒット製品となりました。その後、本家IBM PC/XTでもインターフェースバスを高速化し、10MB HDDを搭載していました。

Macintosh

1986年SCSIインターフェースが搭載されたMacintosh Plusから漢字対応OS[注2]も発売され、日本でMacintoshの普及が始まります。

Macintosh Plus購入時、友人に「買え！」と言われて手に入れたApple社製HD-20SCが、筆者が初めて購入したHDDです。容量20MBで定価28万円と高価な買い物で、当時の月給の2倍もしました。

このHDDに出会う10年前、初めて買ったApple製ディスク装置に使うフロッピーディスクは、1Dタイプ140KB用で1枚2,400円。2Dタイプとして使うためにノッチを切り込んで使っていました[注3]。カセットテープユーザだった筆者には、カッチャンカッチャンと動くのが気持ち良い製品でした。しかしHDDに出会ってしまうとカッチャンカッチャンは、"なんと遅いんだ"に変わってしまいました。

GUIを持つMac OSはサイズが大きくフロッピーからの起動は遅く、Macintoshユーザにとって SCSI HDDは必需品となります。MacintoshではSCSI HDDを複数台接続すると起動しない経験を持つ方々は"非常に多く"おられると思います。Macintosh内蔵SCSIコントローラNCR53C90は最大10MB/sec(以後MB/s)と高速でしたが、非常に不安定でもありました。

不安定な原因はケーブルのインピーダンスにあり、当時の技術では小型コネクタでSCSI規格に合致するインピーダンス100Ωのケーブルが作れません

注1) それまでの外部記憶装置は8インチや5インチサイズの柔らかいフロッピーディスクが多く、それに対して固い円盤を使うため「ハード」ディスクと呼ばれています。

注2) 漢字Talkという名前でした。

注3) フロッピーディスクは、片面(1)と両面(2)があり、容量(単密度(S)／倍密度(D)／倍密度倍トラック(DD)、高密度倍トラック(HD))と合わせて呼ばれていた。1Dは片面単密度のこと。Apple用フロッピーディスクではライトプロテクト部分(ノッチ)に切り込みを入れ、両面使うのが常識で、そのための器具も売られていました。

でした。また、ターミネータも安定に整合できない製品が多く販売されていました。外部HDDやMO、スキャナやカラープリンタなど接続台数が増えるとさらに不安定になり、安定に動かす“ノウハウ”がMacintoshユーザには大事なことでした。

ケーブルはハイインピーダンス化され、ターミネータはSCSI規格が20MB/s、40MB/sと上がるにつれて安定化電源内蔵や、端子ごとにインピーダンスを合わせるICを搭載するアクティブ型に発展し、SCSI利用は安定化していきます。

Cバスと SCSI

NEC PC-9801には、オリジナルな拡張インターフェース“Cバス”を持っていました。SASIインターフェースボード（NEC型番から27ボードと呼ばれた）や、SCSI HDDを接続する“55ボード”がありました。

当時のSCSI最大転送速度規格は10MB/sでありながら純正55ボードの実効転送速度は0.5MB/sと低速だったため、互換ボード高性能化競争が始まり、FIFO（First In, First Out）と呼ばれるキャッシュ機構を搭載しSCSIインターフェースボードとセットされた安価で高性能なHDDがアイシーエム社、アイテック社、緑電子社などから発売されました。

HDDは3.5インチタイプが一般的になり、SCSIインターフェースを内蔵して20MB、40MB、80MBと容量が上がっていきます。

1991年ごろ200MBハードディスクを30万円で、筆者はたいへん安く買ったと喜んだことがあります。

ADAPTEC

インテル社486DX 33MHz DOS/V互換機は、グラフィック性能も高く、海外互換機でも漢字が扱えたためPC-9801に変わって普及していきます。DOS/V互換機でSCSI HDDやMOを使う場合に標準的に使われたSCSIインターフェースがアダプテック社AHA-2940です。DOS/V互換機拡張インターフェースISAバスに対応したAHA-2940はヒット製品で、このボードが動作しない互換機やマザーボードは不良品扱いされる時代でした。

1994年ごろには4GBコナー社製HDDが約10万円で購入できるようになりました。

サーバでは、通電時に交換できるホットプラグなコネクタを持ち、80MB/s、160MB/sに高速化され、電源も供給するSCA-2コネクタを持つSCSI HDDが使われるようになります。

Windows 95の普及により、Macintoshと同様にDOS/V機ユーザも使う外部ストレージ容量が大きくなり、ADAPTEC社製SCSIボードは、DOS/V機でのSCSIインターフェースの主役になります。SCSI規格高速化により20MB/s、40MB/s、80MB/s、160MB/sまで製品化されていきます。

IDE

IBM PC互換機やPC-9801においては内蔵HDD専用インターフェースが進化していきます。SCSIよりもインターフェースを簡便化し、接続ドライブ数を2台までとして低コスト化したIDEインターフェースをコンパックとウエスタンデジタルが開発します。

接続コネクタは逆挿しもできるし、1列ずれて差し込むことができてしまうため、パソコンを組み立てて“動かない！”というドキドキを何度も経験しました。それでも壊れない丈夫なインターフェースでした。後にATAインターフェースとして規格化されます。

ATA

IDEが標準規格化されたのがATAインターフェースです。33MB/sが限界と言われていましたがケーブルの80本化などで133MB/sまで高速化されました。SCSIのような起動時の接続機器確認がないので起動時間が短く、ドライブコストも安価であったため、サーバなどエンタープライズ系を除いてパソコンに内蔵されるHDDインターフェースはATAが主流になります。

DOS/V互換機では当初120GBを超えるHDDを想定せず、160GB HDDが使えない時期もありましたが、2000年ごろにはATA 300GB HDDが10万円以下で購入できるようになりました。SD

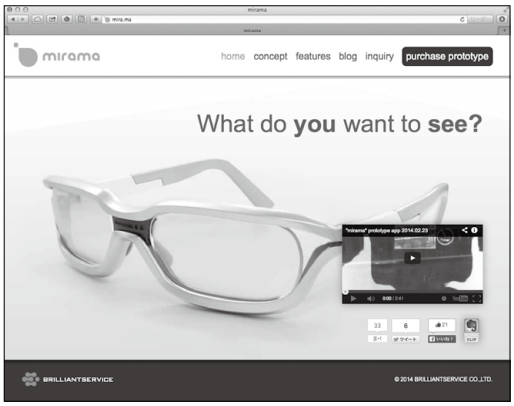
Report

Mirama Prototype IIで未来を見る!

取材・文　後藤 大地

Mirama Prototype II登場!

Mirama（a.k.a Viking）については以前、Software Designの2013年12月号に詳細を掲載しました。いわゆるスマートグラスのひとつで、㈱ブリリアントサービスによって開発が進められているデバイスでありソフトウェアでもあります。2014年3月13日に最新の「Mirama Prototype II」の販売が開始されたのですが、翌日から東京で開催されたAsiaBSDCon 2014で、開発者のJohannes Lundberg（ヨハネス・ルンベリ）氏と議論する機会を得ましたのでその様子をお伝えします。もちろん実際に試作機を使わせてもらったわけですが、なかなかの近未来感でした :-)



▲図1　Mirama（http://mira.ma）

よりスタイリッシュに

MiramaはFreeBSDベースのオペレーティングシステムで開発が進められているスマートグラスです。アプリの開発にはLLVM ClangおよびObjective-Cが使われます。眼鏡型デバイスのグラス部分が透過タイプの液晶になっており、人間の視界に情報がオーバーラップされて見えるというしくみになっています。

▲図2　Mirama Prototype IIをセットアップするJohannes Lundberg氏

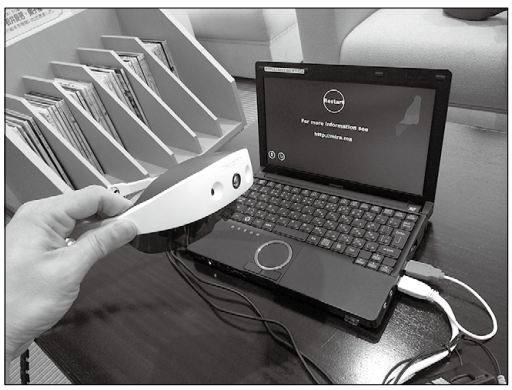
Prototype IIでは顔認識機能、顔認識からメールの作成と送信、画像ファイルの閲覧、ジェスチャーによるアプリの操作などを体験してきました。スペック上ではDSPプロセッサが眼鏡デバイス側に搭載されるようになったあたりが大きな変更点です。Prototype Iのときと同じですが操作に難しいことはなく、早く製品化されないか期待して止まないデバイスです。

▲図3　眼鏡デバイス部分

Prototype IIIは眼鏡デバイス、分離構造へ

Johannes Lundberg氏から今後のロードマップについても教えてもらいました。次期Prototype IIIは2015年第2四半期あたりを目指していること。このプロトタイプでは眼鏡デバイスと処理装置を分離して、フィールド検証などもできるようにするということです。Prototype IIIはおもに産業界向けということになるようです。

演算処理装置はARMボードのような小型のPCを採用し、ペンダントのようにしてポケットに入れて持ち運ぶといったスタイルを考えているということでした。眼鏡デバイス部分もダウンサイジングを進めて、現在よりも実機に近いものにしていくということです。今からMirama Prototype IIIの登場が楽しみですね。



▲図4　演算処理装置と眼鏡デバイス

CONTACT　㈱ブリリアントサービス
URL http://www.brilliantservice.co.jp/

Hardware

ユビキタスエンターテインメント、新製品「enchantMOON S-II」と関連プロジェクトを発表

㈱ユビキタスエンターテインメントは3月14日、同社発のタブレットデバイス「enchantMOON」の新バージョン「enchantMOON S-II（エンチャントムーン エス・ツー）」（以下、S-II）の内覧会を行い、その開発経緯や旧バージョンからの変更点などを披露した。

S-IIでの主な強化点は、「ペンの書き心地向上（レイテンシ最大81ms）」（同社計測では旧バージョンが112ms、Galaxy Note3は118ms）「カメラの高画質化（暗所に強くなった）」「Webブラウザの高速化」「YouTubeでの動画再生対応」「ページ遷移の高速化」「ハイパーリンクの高速化」「新しいページめくり機能」「領域形状の自動補正」といったところ。おもに旧バージョンで不満とされてきた点を改善している。

評価用機材を試用してみたところ、起動時間が旧＝約53秒、新＝約32秒。同じデータの10ページを手動でめくるのにかかった時間が旧＝約46秒、新＝約8秒。サムネイル画面への切り替え速度が旧＝約11秒、新＝約3秒など、かなりの速度向上がみられた（いずれも本体内に同じデータがある状態で実測。計測の数値はデータによって大きく変動するため参考値）。

これらの速度向上は、ソフトウェアの改善のみで行われている。OSであるMOONPhaseを0から書き直す徹底的なリファクタリングにより実現。S-IIのハードウェアは旧バージョンとまったく同じものが採用された。そのため旧バージョンユーザに対しても、同社はOSの無償アップデートを提供し、この速度向上の恩恵を受けられるとしている。

発売は4月を予定。価格は16GB版が49,800円、32GB版が59,800円（いずれも税込み）。

さらに今後のプロジェクトとして、enchantMOONで作成したハイパーテキストをPCやMac、iOSやAndroidでも動作可能にする「Project Gemini」や、enchantMOON専用クラウドサービスの「Project Skylab」を紹介。本体についても2014年第4四半期には「S-IVB」へのバージョンアップを、2015年にはコードネーム「MkII」という新製品を計画していることが同社CEO 清水亮氏の口から告げられた。

CONTACT ㈱ユビキタスエンターテインメント
URL http://www.uei.co.jp/

Service

ハイパーボックス、サーバ／ネットワーク機器監視のSaaSサービスを提供開始

㈱ハイパーボックスは3月10日、サーバ監視サービス「NetKids iMark SaaS」を、専用サーバサービス「blue Box」とクラウドサービス「HyperCloud」の新しいオプションとして提供を開始した。

同サービスは、㈱アイ・エス・ティが提供するサーバ監視ツール「NetKids iMark」をパッケージソフトではなくSaaS（Software as a Service）で提供するというもの。「NetKids iMark」はサーバやネットワーク機器などの稼動状況をさまざまなプロトコルでリモート監視するとともに、対象機器の異常発生時にはメールなどでリアルタイムに通知が行える。これまではCD-ROMでの提供だったため、納品やインストール、アップグレードに時間や費用がかかるという課題があった。そこでSaaSで提供することにより、納品後すぐの監視運用開始と、低料金での提供を実現した。同サービスの特徴は次のとおり。

- 監視対象のサーバの設定が不要なエージェントレスでも50種類以上の監視項目をサポート。エージェントを利用した場合は、100種類以上の項目を監視することが可能。また、各種OS（Windows、Linux、Solaris）に対応したエージェントは無料提供であるため、必要に応じていくつでも利用可能
- 監視設定はWindows Server上のGUIで行える。使い慣れたWindowsアプリケーションと同様の感覚で操作、設定ができる
- エラー発生時には、電子メール、メッセージ表示などさまざまな方法で通知できる。1つのアラートに対し複数の通知を行ったり、閾値を超えた場合や、正常に戻った場合に、通知を行うなどの設定もできる

▼ライセンス料金（税込）

サービス	初期費用	月額	年額
50項目版	0円	19,440円	194,400円
100項目版	0円	21,600円	216,000円
250項目版	0円	23,760円	237,600円
500項目版	0円	31,320円	313,200円
1000項目版	0円	35,640円	356,400円

※項目とは、1つの監視チェックを表す。

CONTACT ㈱ハイパーボックス
URL http://www.hyperbox.co.jp/

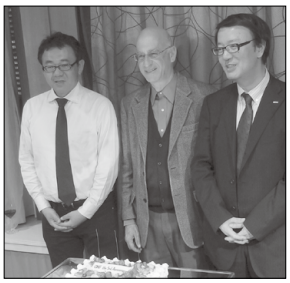
Report

Open Network Foundation、3周年パーティを開催

3月13日、ONF（Open Network Foundation）の設立3周年を祝う懇親会がザ・ランドマークスクエア・トーキョー（東京都港区）にて開催された。国内のSDN（Software Defined Networking）普及を推進する業界関係者が集い、情報交換と交流を行った。

当日は、ONFのエクゼクティブ・ディレクターのDan Pitt氏も参加し、海外におけるSDNの普及／振興状況について話し合われた。6月に開催されるネットワーク技術の総合展示会「Interop Tokyo 2014」でSDNをどのように紹介していくかなど話題は尽きなかった。

Dan Pitt氏へのサプライズで3周年を記念したケーキが振る舞われ、懇親会は盛り上がりを見せた。



▶ ONFエクゼクティブ・ディレクター Dan Pitt氏（写真中央）

CONTACT Open Network Foundation
URL https://www.opennetworking.org/ja/

Hardware

ぷらっとホーム、「OpenBlocks」シリーズのIoTルータエディション「OpenBlocks A7/IoTR」を発表

ぷらっとホーム㈱は3月6日、M2M（Machine to Machine）やIoT（Internet of Things）通信に対応し、ルータ機能を内蔵したマイクロサーバの新モデル「OpenBlocks A7/IoTR」を発表した。

同製品は、汎用Linuxサーバ「OpenBlocks A7」がベースとなっており、拡張性に優れたインターフェースを持つ。TCP/IPはもちろん、IEEE1888、REST、SOAPなどの高度な広域インターネットプロトコルに対応し、データの加工や処理／判断のための柔軟で高度なプログラミングが可能。また、Oracle Java SE Embeddedが搭載されており、既存のソフトウェア資源を高い移植性を活かして動作させることができる。

加えて、ルータ機能の強化により、多拠点／多地点へのIoT展開が安定かつ容易に実現可能となった。今後のIoTサーバに必要なローカルエッジにおける情報処理の実現とともに、従来はルータ＋サーバの2機種構成であったシステムを1機種で実現する。

価格は数量や構成などにより異なるため、オープン価格となっている。

CONTACT ぷらっとホーム㈱
URL http://www.plathome.co.jp/

Hardware

ソーソー、LinuxMania初のタブレット「サクス・タブレット7」を発売

㈱ソーソーは、2月5日より「LinuxMania」初のタブレット製品となる「サクス・タブレット7（L-Sax Tablet 7inch）」を発売している。

「LinuxMania」とは、同社が展開するLinuxマシン専門ブランド。これまではデスクトップPCを中心に製品を販売してきたが、このたび初めてタブレット製品を発表した。OSにはAndroid 4.04を搭載している。

ハードウェアのおもなスペックは、CPUがIntel Atom Z2460 1.6GHz 1コア、メモリは1GB。記憶容量は16GB。Bluetoothおよび、IEEE 802.11b/g/n準拠の無線LANを使用できるほか、端末本体には0.3Mピクセルのフロントカメラと2Mピクセル背面カメラが備わっている。価格は19,440円（税込）で、同社Webサイトから注文できる。

▲サクス・タブレット7（L-Sax Tablet 7inch）

CONTACT ㈱ソーソー
URL http://www.linuxmania.jp/

ひみつのLinux通信

作)くつなりょうすけ
@ryosuke927

第5回 モヒカン先輩

四十肩でペンタブ以外持てない体に進化したくつな先生のマンガが読めるのは本誌だけ！

to be Continued

エラーメッセージはあなたを救いますよ。それが英語でも大事だから絶対読みましょう。某勉強会で「insult（侮辱）」用オプションをつけてビルドし、設定ファイルを編集することで「sudoに罵られる」というネタがありまして、やってみたらこれが結構面白いのでオススメです。sudo実行時のパスワードを間違えることで「Where did you learn to type？（どこでタイピング習ったんだよ？）」とか「Are you on drugs？（キメてんの？）」などと罵られるとちょっと快感になるかもしれませんよ。

Letters from Readers

技術評論社はExcel方眼紙なんぞ使いません！

一時期、インターネット上でExcel方眼紙の是非について、議論が交わされていましたね。幸い弊社の社内文書には、Excel方眼紙はありません。「さすがコンピュータ技術書の出版社。Excelの正しい使い方をわかっていらっしゃる!」そう思うのは早いですよ。なぜなら事務的な申請用紙はいまだに紙なのです！　承認はもちろんハンコです!

2014年3月号について、たくさんのお便りありがとうございました！

第1特集　RDBとNoSQL どちらを選びますか?

新しいデータベースとしてNoSQLが注目されています。しかし、流行しているからという理由だけでNoSQLを導入していいのでしょうか?　本特集では従来からあるRDBと新進気鋭のNoSQL、両者の特徴を整理し、「どんな場面で導入するべきか」について解説しました。

リレーショナルデータベースの設計ネタを今後もお願いします。

東京都／池野さん

後輩に教えるための良い教科書になりました。

神奈川県／ふかさん

それぞれのデータベースの特徴がわかりやすく取り上げられており、たいへん参考になりました。ところで、22ページの「リレーショナルモデルの限界」に取り上げられていたグラフ、ツリー、履歴に適したデータベースモデル（アプリケーション）にはどのようなものがあるのか、機会があれば、取り上げてくださるとうれしいです。また、それらとRDBと連携して使用する方法についても知りたいです。

大阪府／出玉のタマさん

正規化はもう少し実践的なサンプルのほうがいいかもしれません。どうしても情報処理試験を意識して読み飛ばす傾向にあると思うので。

兵庫県／yoneさん

普段はRDBを使っているので、NoSQLのことを知ることができてよかった。

静岡県／もものがさん

NoSQLが台頭してきたからといって、RDBがまったく不要になるということはありえません。それぞれに得意不得意があり、適材適所で使い分ける必要がある。なんとなくはわかっていた結論かもしれませんが、技術的な裏付けとともに説明されると、あらためて納得できますね。

第2特集 プロキシサーバの教科書

Webシステムではすでにあたり前のものとなっているプロキシサーバ。その基本的な役割をおさらいするとともに、クラウドサービスで提供されるプロキシサーバなど最近の話題についても取り上げました。

ちょうどリバースプロキシ導入予定だったので、興味のある内容だった。

東京都／まさかり持った桃太郎さん

リバースプロキシはWebシステム構築時に頻繁に使っており、今の開発でも使っているので、現場の若い子にも読ませたい記事でした。

北海道／後藤さん

プロキシは古い書籍にしか情報がないため、この特集があって良かったです。

大阪府／Yutakaさん

ひとくちにプロキシサーバといっても、時代の変化とともにいろいろな役割が出てきました。そのあたりが詳しくまとめられていて参考になる内容でした。

埼玉県／犬棟梁さん

設計、構築部分をもっと詳細に取り上げてほしかった。

東京都／blackbirdさん

第1特集のデータベースと同じく、第2特集のプロキシサーバも昔からあるものですが、こちらもみなさんの関心が高かったようです。Webシステムには必須のものだから、現場で触れる機会も多いのでしょう。

短期集中連載　さらに踏み込む、Mac OS Xと仮想デスクトップ

Macの仮想デスクトップについての新連載です。複数のOS環境を今まで以上

に快適に利用するためのノウハウを紹介しました。

Macにそんなに興味がなかったが、おもしろかった。

岐阜県／トッチさん

Macだけは専用ハードウェアを購入しないと使えないので、次回購入対象として考えている。仮想環境にしてさまざまなOSを試そうと自分でも考えていた。今後の連載にも期待します。

岩手県／バイク王さん

次にOSをバージョンアップするときはMac＋VMware＋Windowsと決めております。

千葉県／Tayuさん

Macを導入しようと考えている読者の方々がみんな、仮想デスクトップで複数OSを使いたいと考えているのが、印象的でした。「複数OS使うならMac」となるのは、「OS XはWindowsで動かせないから」という理由もあるのでしょうが、基本的な機能も使いやすいからですかねえ。

連載

「レッドハッド恵比寿通信」がいつもユニークな文章でおもしろいです。毎回、筆者が変わるにもかかわらず、ほとんどの号で専門知識が少なくても読めるような配慮と小気味よい言い回しが好きです。これからも毎月読むのでがんばってください。

北海道／ひみなとさん

レッドハットといえば、今月号からは、Red Hat Enterprise Linuxの技術解説を行う連載記事「.SPECS」も始まりました。こちらもご注目ください。

表紙

派手過ぎず落ち着いた風合いがたいへん良いです。

愛知県／CoSMoSさん

ジャポニカ学習帳の路線かな?

大阪府／オブジェクト脳192さん

今月号から表紙のデザインが新しくなりました。2012年度の表紙は花、2013年度は鳥でしたが、2014年度は犬です。ジャポニカ学習帳の路線じゃありませんよ!

エンジニアの能率を高める一品

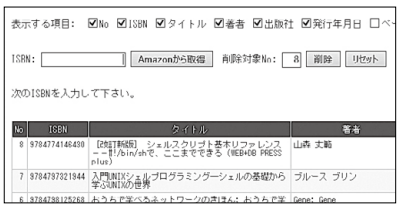
仕事の能率を高める道具は、ソフトウェアやスマートフォンのようなデジタルなものだけではありません。
このコーナーではエンジニアの能率アップ心をくすぐるアナログな文具、雑貨、小物を紹介していきます。

バーコードリーダ［USB］

9,180円（税込）／快読ショップYomupara　http://www.yomupara.com/

▲図1 ISBNがテキストデータ（改行付き）で入力される

今回紹介するのは蔵書目録を作るのに便利なバーコードリーダ。使い方ですが、まず本製品をPCとUSBでつなぎます。次に、PC上でテキストエディタを開きます。その状態で書籍背面にあるISBNのバーコードを本製品で読み取ります。すると、**図1**のように読み取ったISBNコードがエディタに入力されます。世の中には、Amazon社が公開する書誌情報にアクセスし、ISBNから書名、著者名、出版社名などの情報を取得してくれるサービス／ツールがいくつかあります。それらと併用すれば、バーコードで読み取るだけで蔵書の一覧が作れます。たとえば、**図2**は本製品の販売元Yomuparaが提供する「書誌コレクター」サービス。バーコードで読み取ると、逐次、書名などの情報を補って一覧表示してくれます。一覧はCSVにも変換できて、便利です。



▲図2　http://www.yomupara.com/isbn_corrector.php

3月号のプレゼント当選者は、次の皆さまです

①REALFORCE108UG-HiPro ……東京都　梅原潤様
②JetFlash 380 ……神奈川県　八巻淳様

★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
June 2014

2014年6月号
定価（本体1,220円+税）
176ページ

5月17日発売

［第1特集］新人Linux使い養成講座

設定ファイルの読み方・書き方でわかるLinuxのしくみ

上達のヒントは設定ファイルを大事に扱うこと

UNIXを使い始めて、戸惑うのは設定ファイルの多さ、そしてその保存場所です。本特集では、Linuxをベースに、ディレクトリとファイル編集の方法を最初に学びます。さらにユーザ管理、システム設定、ネットワーク設定、各種アプリケーション設定などの要点を押さえていきます。スジ良くLinuxのシステムを深く理解するための基礎体力をつける基礎講座です。システム管理者予備軍だけでなく、クラウドでDevOpsな皆さんにもぜひ!

［第2特集］今日から始めるUbuntuライフ

Ubuntu 14.04 LTS 徹底解説

エントリーユーザからベテランまでUbuntu漬け!!

［特別企画］「Google Glass」GDK先取り解説

※特集・記事内容は、予告なく変更される場合があります。あらかじめご容赦ください。

休載のお知らせ

「Androidエンジニアからの招待状」（第48回）は都合によりお休みいたします。

SD Staff Room

●『王様達のヴァイキング』というハッカー漫画をご存じだろうか。これは、はじめて「ハッカー」を漫画でかなりリアルに表現できた記念すべき物語なのだ。奥付を見ると、協力者の皆さんはどこかで見かけた方ばかり。本誌も出させてくれないだろうか。なーんてことを思ったりして。そのくらい大好き。（本）

●健康のため、週に数回、終業後に市ヶ谷からお茶の水まで歩いている。途中、靖国神社の前を通るが、ちょっと前は警官が大勢いた。不審火事件があったらしい。今は春の宴で出店が出てお花見真っ最中。その中にも例年より多くの警察官を見かける。それを見て怖いと思うか頼もしいと思うか。とにかくお疲れ様です。（幕）

●編集後記を通勤電車の中で書こうとNexus 7を取り出したそのとき! 乗車してきたビジネスマンが取り出したのは「HP200LX」!! 「この人、できる」「ふん、しょせん貴様はマシンの性能に頼りすぎなのだよ」……ガーン（いや、目すら合ってないですけどね）。そんな妄想日和のうららかな春の朝でした。（キ）

●「増税前に、駆け込みで物を買うなんてみっともない」そう思っていたら、3月末に自宅のトイレットペーパーがきれました。しかも、タイミング悪く、腹痛にも見舞われました。このままでは、家で手洗いもままなりません。消費税とは関係なく、べつの意味で駆け込み買いせざるをえませんでした。（よし）

●新たにクレジットカードをつくってみたが、マイページへのログインパスワードの設定で超難儀。英数全半角および記号を使うよう指示されつつエラーを繰り返し設定すること十数回……やっと設定できたパスワードは24ケタ。まったく覚えられません……まだ一度も使ってないけど早く解約しよ。（なか爺）

●いままで我が家にはパソコンがなかったのですが、増税を機にとうとう購入することにしました。私自身まったくこだわりがないので、メーカーすら旦那さんにすべてお任せ。晴れてご購入となったが、回線工事に時間がかかり、まだ回線がつながっていないのでただのオブジェ。この号が出るころにはネットも見れるようになってるはず……。（ま）

ご案内
編集部へのニュースリリース、各種案内などがございましたら、下記宛までお送りくださいますようお願いいたします。

Software Design 編集部
ニュース担当係

［FAX］
03-3513-6173
※FAX番号は変更される可能性もありますので、ご確認のうえご利用ください。

［E-mail］
sd@gihyo.co.jp

Software Design
2014年5月号

発行日
2014年5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

●編集アシスタント
松本涼子

●広告
中島亮太
北川香織

●発行所
㈱技術評論社
編集部
TEL：03-3513-6170
販売促進部
TEL：03-3513-6150
広告企画部
TEL：03-3513-6165

●印刷
図書印刷㈱

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

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