Special Feature 01
Vimのスキル向上
Special Feature 02
最新ルーティング技術
2013 October
10
Software Design
[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
2013年10月18日発行
毎月1回18日発行
通巻342号
（発刊276号）
1985年7月1日
第三種郵便物認可
ISSN 0916-6297
定価
1,280円

プログラマ・インフラエンジニア御用達

Vim至上主義

Vim is the Best Editor.

Special Feature 01

Vimを使いこなしていますか?

Special Feature 02

ネットワーク技術力のパワーアップ

ルータの教科書

ルーティングの基礎からBGPまで

Article

Key Value Storeをゼロから創る

okuyamaでわかるkvsの本質とは

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。

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58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。

Tizen

オープンソースのモバイルOS「Tizen」

iOSとAndroidがシェアの大半を握っているスマートフォンOS市場ですが、開発者やユーザの選択肢を拡げる“第3勢力”として、オープンソースのモバイル端末向けOSに対する期待が高まってきています。その中の1つに、インテルやサムスン電子らの主導によって開発が進められている「Tizen」があります。

Tizenはスマートフォンやタブレット、スマートテレビなどのデバイスを対象としたOSで、Linux Foundationの「Tizen Project」および非営利団体「Tizen Association」によって仕様の策定や開発が行われています。前者はおもに技術面を担当するグループで、インテルとサムスン電子が中心となって管理しています。一方、後者は産業面を担当するグループで、ビジネス的な視点からの要求仕様の検討やサービスモデルの選定、市場開拓などを行っています。

TizenはもともとモトローラやNECらが中心となって開発されたLinuxベースのモバイルOS「LiMo」を引き継ぐ形で誕生しました。Tizenの発表当時、インテルとLinux Foundationは同じくLinuxベースのモバイルOSである「MeeGo」を推進していましたが、両者はTizenプロジェクトへの合流にともなってMeeGoの開発を収束させ、Tizenに移行していく方針を発表しました（MeeGoについては本誌2010年10月号の本連載でも取り上げました）。

Tizenのアプリケーション開発環境

Tizenは、大きく分けると次の4つのサブシステムから構成されます。

- **Webフレームワーク**……W3C/HTML5準拠のWeb技術によってアプリケーションが開発できる
- **ネイティブフレームワーク**……C/C++を用いてネイティブアプリケーションが開発できる
- **コアモジュール群**……上記のフレームワークに対して各種コア機能を提供する。オープンソースライブラリや追加APIセットなども含まれる
- **カーネル**……Linuxカーネルおよびデバイスドライバ

この構成からもわかるように、Tizenではアプリケーション開発のために2つの異なる方法が提供されます。1つはHTML5/CSS3/JavaScriptといったWeb標準技術によって開発が行えるWebフレームワークです。Webフレームワークでは、アニメーションやローカルストレージ、ビデオ、オーディオなどのAPIもサポートされているため、リッチアプリケーションの開発にも対応します。また、通話機能やBluetooth、NFCといったデバイス固有の機能に対しても、JavaScriptで利用するためのAPIが用意されます。

もう1つはC/C++ベースのネイティブフレームワークです。これを利用すれば、高度なグラフィック機能やメディア機能、メッセージング、ソーシャルサービスなど、デバイスやプラットフォームの性能を最大限に活用したアプリケーションの作成が可能です。

アプリケーション開発環境としては「Tizen SDK」と呼ばれる開発ツールが提供されています。Tizen SDKには、Eclipseベースの IDEやエミュレータ、各種関連ツール、サンプルコードなどが含まれており、Windowsのほかに Mac OSや Ubuntu上で利用することが可能です（執筆時点）。開発したアプリケーションは、Tizen Storeを通じて配布・販売することができます。

モバイル端末以外にも広がるTizenの世界

Tizenと同様に“第3勢力”として注目されているモバイルOSには、本誌2013年2月号で取り上げた「Firefox OS」があります。両者ともネイティブに近いレイヤでWeb技術によるアプリ開発をサポートしている点は共通していますが、Tizenの場合にはパフォーマンスが要求される用途向けにネイティブフレームワークという選択肢も用意されており、ハイエンドな端末の要件も満たせるという強みを持っています。

さらに、Tizen Projectではテレビやカメラなどといったさまざまな機器への搭載を視野に入れた計画が進行中です。その先駆けと言えるのが2013年5月に公開された「Tizen IVI」で、これはTizenの技術をベースとした車載情報機器（IVI）向けのプラットフォームです。

Tizenプロジェクトではインテル／サムスン電子を中心に精力的に開発が進められており、2013年後半にはいくつかの端末が発売される予定となっています。SD

Tizen
http://www.tizen.org/ja

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。

Software Design Oct. 2013

Contents

1

第1特集 Special Feature 01

Vimを使いこなしていますか?

Vim至上主義

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。

ソフトウェア・デザイン［目次］

Contents

Software Design Oct. 2013

2

第2特集 Special Feature 02

ネットワーク技術力のパワーアップ

ルータの教科書 ルーティングの基礎からBGPまで

一般記事 Article

巻頭Editorial PR Editorial PR

アラカルト A La Carte

広告索引 AD INDEX

	広告主名	ホームページ	掲載ページ
ア	アールワークス	http://www.astec-x.com/	裏表紙
	エーティーワークス	http://web.atworks.co.jp	P.4-P.5
	NTTコミュニケーションズ	http://www.ntt.com/	第1目次対向
サ	サイバーエージェント	http://www.cyberagent.co.jp/	第2目次対向
	シーズ	http://www.seeds.ne.jp/	P.6
	システムワークス	http://www.systemworks.co.jp/	P.26
ナ	日本コンピューティングシステム	http://www.jcsn.co.jp/	裏表紙の裏
ハ	ハイパーボックス	http://www.domain-keeper.net/	表紙の裏-P.3

広告掲載企業への詳しい資料請求は、本誌Webサイトからご応募いただけます。 http://sd.gihyo.jp/

Software Design plus

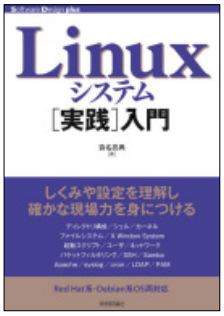
最新刊！

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

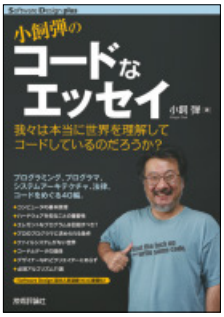
データサイエンティスト養成読本編集部 編
B5判・152ページ
定価 1,980円(本体)+税
ISBN 978-4-7741-5896-9

Software Design編集部 編
B5判・176ページ
定価 1,880円(本体)+税
ISBN 978-4-7741-5888-4

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。



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



小飼 弾 著
A5判・200ページ
定価 2,080円(本体)+税
ISBN 978-4-7741-5664-4

青木 直史 著
A5判・288ページ
定価 2,980円(本体)+税
ISBN 978-4-7741-5522-7

Japanese Raspberry Pi Users Group 著
B5変形判・256ページ
定価 2,380円(本体)+税
ISBN 978-4-7741-5855-6

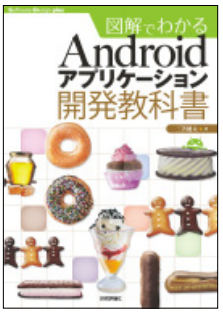
菅野 裕、今田 忠博、近藤 正裕、杉本 琢磨 著
B5変形判・336ページ
定価 3,200円(本体)+税
ISBN 978-4-7741-5567-8

河村 嘉之、川尻 剛 著
B5変形判・480ページ
定価 2,980円(本体)+税
ISBN 978-4-7741-5438-1

㈱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
B5変形判・256ページ
定価 2,580円(本体)+税
ISBN 978-4-7741-5325-4

データベースエンジニア養成読本編集部 編
B5判・136ページ
定価 1,980円(本体)+税
ISBN 978-4-7741-5806-8

WINGSプロジェクト 著
B5判・352ページ
定価 2,580円(本体)+税
ISBN 978-4-7741-5611-8



三苫 健太 著
B5判・400ページ
定価 3,200円(本体)+税
ISBN 978-4-7741-5189-2

Software Design編集部 編
B5判・200ページ
定価 1,980円(本体)+税
ISBN 978-4-7741-5038-3

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

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

プロのためのLinuxシステム・10年効く技術
中井 悦司 著
定価 3,400円+税　ISBN 978-4-7741-5143-4

サーバ/インフラエンジニア養成読本 管理・監視編
Software Design編集部 編
定価 1,980円+税　ISBN 978-4-7741-5037-6

業務に役立つPerl
木本 裕紀 著
定価 2,780円+税　ISBN 978-4-7741-5025-3

Apache[実践]運用/管理
鶴長 鎮一 著
定価 2,980円+税　ISBN 978-4-7741-5036-9

プロになるための データベース技術入門
木村 明治 著
定価 3,180円+税　ISBN 978-4-7741-5026-0

データベース技術[実践]入門
松信 嘉範 著
定価 2,580円+税　ISBN 978-4-7741-5020-8

2週間でできる! スクリプト言語の作り方
千葉 滋 著
定価 2,580円+税　ISBN 978-4-7741-4974-5

PCのウイルスを根こそぎ削除する方法
本城 信輔 著
定価 1,980円+税　ISBN 978-4-7741-4867-0

Androidエンジニア養成読本
Software Design編集部 編
定価 1,880円+税　ISBN 978-4-7741-4859-5

Vyatta入門
近藤 邦昭、松本 直人、浅間 正和、大久保 修一(日本Vyattaユーザー会) 著
定価 3,200円+税　ISBN 978-4-7741-4711-6

プロのためのLinuxシステム・ネットワーク管理技術
中井 悦司 著
定価 2,880円+税　ISBN 978-4-7741-4675-1

サーバ/インフラエンジニア養成読本
Software Design編集部 編
定価 1,880円+税　ISBN 978-4-7741-4600-3

Linuxエンジニア養成読本
Software Design編集部 編
定価 1,880円+税　ISBN 978-4-7741-4601-0

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

ソフトウェア・デザイン［目次］

Contents

Software Design Oct. 2013

3

Column

Development

OS/Network

Inside View

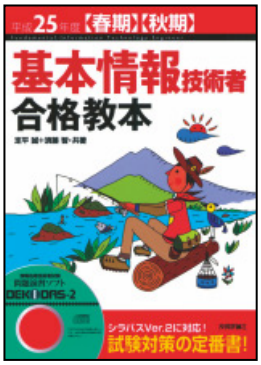
Logo Design ロゴデザイン ＞ デザイン集合ゼブラ+坂井 哲也
Cover Design 表紙デザイン ＞ Re:D
Cover Photo 表紙写真 ＞ Oktay Ortakcioglu/Getty Images
Illustration イラスト ＞ フクモトミホ、高野 涼香、中川 悠京
Page Design 本文デザイン ＞ 岩井 栄子、近藤 しのぶ、SeaGrape、安達 恵美子
［トップスタジオデザイン室］轟木 亜紀子、阿保 裕美、佐藤 みどり
［BUCH+］伊勢 歩、横山 慎昌、 森井 一三、Re:D、［マップス］石田 昌治

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。



定平誠、須藤智　著
A5 判／ 544 ページ
定価 1,764 円（1,680 円+税）
ISBN 978-4-7741-5386-5

栢木厚　著
A5 判／ 456 ページ
定価 1,869 円（1,780 円+税）
ISBN 978-4-7741-5398-8

きたみりゅうじ　著
A5 判／ 656 ページ
定価 2,079 円（1,980 円+税）
ISBN 978-4-7741-5437-4

山本三雄　著
B5 判／ 544 ページ
定価 1,554 円（1,480 円+税）
ISBN 978-4-7741-5720-7

あなたを合格へと導く一冊があります！

大滝みや子、岡嶋裕史　著
A5 判／ 728 ページ
定価 3,129 円（2,980 円+税）
ISBN 978-4-7741-5351-3

加藤昭、芦屋広太、矢野龍王　著
B5 判／ 472 ページ
定価 1,764 円（1,680 円+税）
ISBN 978-4-7741-5721-4

大滝みや子　著
B6 判／ 376 ページ
定価 1,554 円（1,480 円+税）
ISBN 978-4-7741-4807-6

内田保男　他　著
A5 判／ 504 ページ
定価 3,360 円（3,200 円+税）
ISBN 978-4-7741-4944-8

岡嶋裕史　著
A5 判／ 656 ページ
定価 3,129 円（2,980 円+税）
ISBN 978-4-7741-5355-1

エディフィストラーニング株式会社　著
B5 判／ 392 ページ
定価 3,129 円（2,980 円+税）
ISBN 978-4-7741-5722-1

岡嶋裕史　著
A5 判／ 624 ページ
定価 3,129 円（2,980 円+税）
ISBN 978-4-7741-5558-6

エディフィストラーニング株式会社　著
B5 判／ 376 ページ
定価 3,129 円（2,980 円+税）
ISBN 978-4-7741-5573-9

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

低レイヤーはおもしろさで満ちあふれている

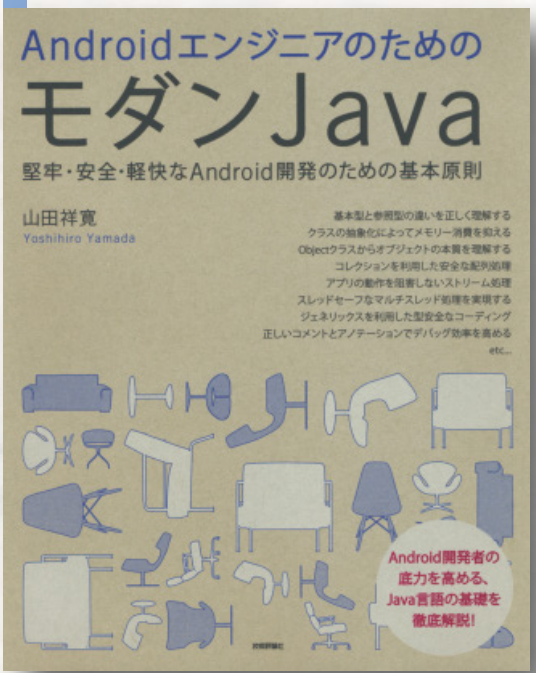
たのしいバイナリの歩き方

「シューティングゲームをチートから守るには？」
「リバースエンジニアリングされないためには？」
「脆弱性を見つけ、権限を奪取するには？」

普通のプログラミングだけでは意識しない低レイヤーの世界は、コンピュータを自在に操れる楽しさでいっぱい。アセンブラの読み方から最新の応用事例まで、技術と考え方が実例を通じてわかります。

愛甲健二　著／A5 判／320 ページ
定価 2,919 円（2,780 円+税）
ISBN 978-4-7741-5918-8

堅牢・安全・軽快なAndroid開発のための基本原則



Android エンジニアのための
モダン Java

本書は、複雑かつ高度な Android アプリの開発に必要となる、Java 言語の基礎を理解することに主眼を置いています。Android SDK については触れず、Android に関連するモダン Java（Java 5 以降）のファンダメンタルな言語仕様を中心に解説を行います。また、解説中では、Android エンジニアが知っておくべき最適化のテクニックを適宜紹介しています。JavaScript や C++、Objective-C での開発経験のある Android エンジニアが、Java 言語に関する知識の底上げを目指す際、特に有効な一冊です。

山田祥寛　著／B5 変形判／448 ページ
定価 3,360 円（3,200 円+税）
ISBN 978-4-7741-5878-5

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

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。

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。

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。

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

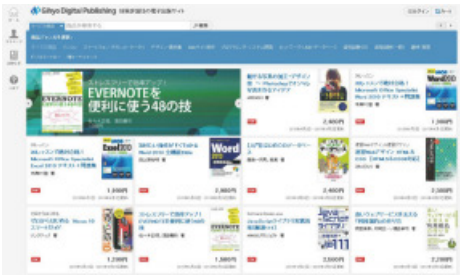
技術評論社の本が電子版で読める！

電子版の最新リストは

Gihyo Digital Publishingの

サイトにて確認できます。

http://gihyo.jp/dp



GIHYO Digital Publishing 未来の"普通"を、今。

Google play

楽天 Rakuten KC　kobo

※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

Software Design 総集編 1990→2000

絶賛発売中!!

11年分のバックナンバーがこの1冊に！

Software Design 1990 年創刊号～2000 年 12 月号の特集、一般記事、連載、特別企画の記事 PDF を収録。
総ページ数 2 万 1,000 ページ超。

- 1冊1ファイル形式でiPadなどでも使いやすい
- PCより全号一括検索が可能
- 書き下ろし特集「プログラミング技術革新の歴史」「インターネット発展の 20 年」を収録

B5 判　108 ページ
2,499 円（本体 2,380 円＋税）
ISBN 978-4-7741-5714-6

技術評論社　全国の書店、またはオンライン書店でお買い求めください。
〒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 総集編 2001→2012

絶賛発売中!!

12年分のバックナンバーがこの1冊に！

Software Design 2001年1月号～2012年12月号の特集、一般記事、連載、特別企画の記事PDFを収録。
総ページ数2万8,000ページ超。

- 1冊1ファイル形式でiPadなどでも使いやすい
- PCより全号一括検索が可能
- 書き下ろし特集「〈OSS全盛期を生き抜くために〉技術の進化をたどりながらLinuxを完全理解」を収録

B5判　100ページ
2,079円（本体1,980円+税）
ISBN 978-4-7741-5593-7

技術評論社　全国の書店、またはオンライン書店でお買い求めください。
〒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

ネットワークエンジニア虎の穴

『自宅ラックのススメ』

文／ tomocha（http://tomocha.net/diary/）

第6回 SNMPと監視

イラスト：高野涼香

ネットワーク機器の監視

前回、言葉に出てきたSNMPや監視について簡単にお話させていただきます。

SNMPとは、Simple Network Management Protocolの略で、言葉のとおり、ネットワーク機器の情報の取得や制御などができるプロトコルになります。最近では、ネットワーク機器に限らず、サーバや各種OSなどの情報の取得が可能です。

SNMPによって具体的には何ができるかというと、ネットワーク機器であれば、CPU使用率、メモリの使用率、トラフィックの使用率などの値が取得できます。そこで各種監視ツールを使用すれば、それらの値からグラフ化し、視覚的に確認ができるようになります。仮に異常が発生した場合、過去に蓄積されたデータを基に作られたグラフから、その発生原因を究明することもできるでしょう。

2種類のMIB

どういったデータが取得可能かは、機器によって若干異なりますが、MIB（Management Information Base）により規定されています。MIBにも2種類あって、標準MIBという共通の定義と、拡張MIB（Enterprise MIBやPrivate MIBとも呼ばれる）という各社ベンダーが独自に実装しているMIBの2つがあります。

具体的に紹介してみましょう。筆者はCacti[注1]というツールを愛用しており、これで各ネットワーク機器のネットワーク負荷状態や、CPUの使用率、さらに温度などを取得しています。温度などは各社ベンダーが実装している拡張部分になります。それを調べたい場合は、ベンダーが公開しているMIB情報をダウンロードして、必要な情報をそれから取れば良いということになります。

さて、定番はCiscoかもしれませんが、それでは当

注1） Cacti（http://www.cacti.net/）

▼写真1　NEC IXシリーズ

たり前すぎておもしろくないので、NEC IXシリーズ（写真1）から取得したルータのグラフを紹介しましょう（図1～図3）。

ログ監視（Syslog）

CPUおよびTrafficは標準MIBに入っていますが、温度については、ベンダー独自実装である拡張MIBに入っています。MIBに関する情報をきちんと設定すると、いろいろな情報が取得できます。SNMPで情報がとれなくなったときは機器が落ちたか、ネットワークが落ちたか、などのトリガーとなりますので、それらアラートをモニターにあげることにより、死活監視もできますし、閾値を超えたらアラートをあげるといったこともできます。

そのほか監視といえば、ログによる監視もありますね。こちらは、Syslogサーバというものを用意して、各種ネットワーク機器からのログを受け取ります。Syslogサーバを導入する理由を説明します。ネットワーク機器には、メモリやディスクが非常に少なく、ログをたくさん保持することができません。とくにルータやFireWallとなれば、機器の異常やステータスの変化だけではなく、接続・切断、ルールに基づいて処理された結果なども出力されます。ログをネットワーク機器で保存するとなると、それなりのストレージが必要になってしまいますが、その分高価になってしまいます。そのためにSyslogサーバにログを転送する方法が採られています。

ログの保存は非常に重要です。ほとんどのネットワーク機器の場合、電源を落としたり再起動したら、ログは消えてしまうことが少なくありません。ログを取得しておくと、ネットワークに異常が発生したときに気づく手がかりとなりますので、なるべくログは取得しておきましょう。そしてSyslogやSNMPはUDPで転送されますので[注2]、なるべくセキュアで隔離されたネットワークを用意しましょう。とりあえずは、専用のVLANを用意してもいいですね。

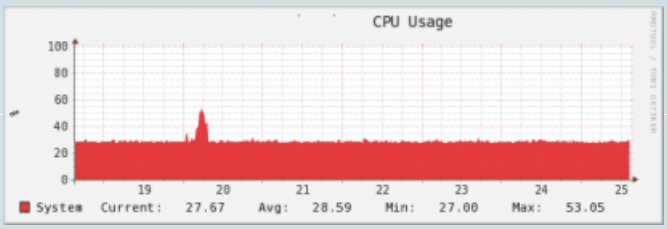
NTPにも注意

ログや情報を取得する場合、機器によっては時間がずれているといったことになると整合性がとれなくなりますので、NTP（Network Time Protocol）を使用して、機器の時刻は正確に合わせておきましょう。

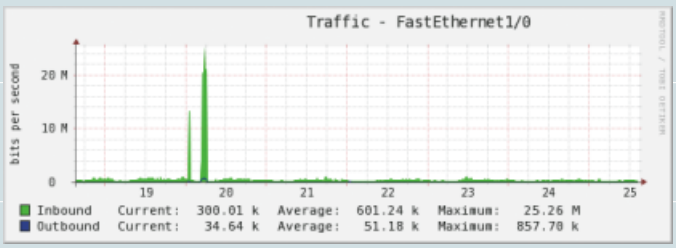
SyslogもSNMPのどちらも、ログを受け取ったり、監視するためにはサーバが必要になりますので、サーバの知識も必要になってきますね。本稿では、具体的な構築方法については詳しくは述べませんが、こういう技術もあるということで、今後のネットワーク構築時に参考にしてみてください。SD

注2）TCPもありますが機器によっては実装されていないなど、多くの場合UDPが用いられます。

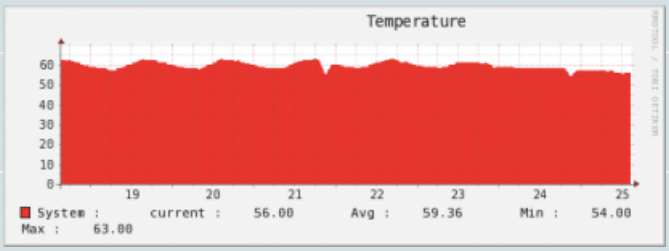
▼図1　CPU使用率



▼図2　ネットワークトラフィック



▼図3　温度



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

DIGITAL GADGET

安藤 幸央 —— Yukio Ando ——
EXA CORPORATION
[Twitter] >> @yukio_andoh
[Web Site] >> http://www.andoh.org/

Volume 178 >>>>>

コンピュータグラフィックスの祭典 SIGGRAPH 2013[前編]
～ディズニーランドの街アナハイムで出会ったテクノロジ、デジタルガジェット編

ハリウッド映画のメイキングから3Dプリンタまで

2013年7月21日から25日の5日間、コンピュータグラフィックスとインタラクティブ技術に関する世界最大の学会・展示会である第40回SIGGRAPH 2013が開催されました。今年は、昨年よりも若干少ない77ヵ国から17,162人の登録参加者であった一方、昨年より2割ほど多い180社の展示が行われました。

40周年となる今年のSIGGRAPHのテーマは、最近のSIGGRAPHの流れである、アートとテクノロジの融合を端的に示す、“LEFT BRAIN AND RIGHT BRAIN”でした。アートや創造性を司る右脳と、論理的思考や数値計算の左脳とが連携しあい、より素晴らしい研究や作品を生みだそうという想いが込められています。

最近のSIGGRAPHの傾向は、他の分野の応用で、研究や映像作りに広がりが出てきているとともに、CG関連の基礎研究がデジタルカメラやゲーム業界などさまざまな分野で応用されていることです。クラウドコンピューティングや、Webの活用も広がってきました。

たとえば映画撮影の際、実際の撮影前に場所を選定するロケハン（ロケーション・ハンティング）と呼ばれる現場の下見が行われます。そのロケハンも最近では世界各地をパノラマ写真で確認できるGoogle Street Viewが活用されることが多くなったそうです。また、ディズニーランドのアトラクションに生かせそうな研究の数々を進めるディズニーリサーチや、Microsoft Researchの研究がゲーム体験に生かされるような事例もあります。

ほかにも、物体の奥行き情報を計測できるXbox Kinectの登場によって、旧来Kinectのようなインターフェースを開発すること自体が研究目的だったと

« SIGGRAPH会場となったアナハイム・コンベンション・センター。ディズニーランドの花火が見えるほど近くにある

今年のテーマとなったArtとScienceを象徴したロゴ »

« 会場で人気だった全身の3Dスキャン体験コーナー

映画『モンスターズ・ユニバーシティ』で注目のピクサーのブース »

※本記事で紹介しているものは国内未発表・未発売のものを含んでおります。

ころが、急にKinectを活用する方向のさまざまな研究に転化してきていることが挙げられます。

さらに、安価に大量のコンピュータパワーが入手できるようになったことで、実写映像をもとに解析したり、大量のデータ素材をもとに解析するなど、課題解決のためのアプローチの方法も変わってきています。

SIGGRAPHの中で数多くの素晴らしい研究や展示をみかけ、感心しきりですが、良いものに共通した法則のようなものがあります。それは、どれも該当のテーマに関して膨大な過去事例、調査、リファレンスを調べていることです。研究者の世界では「巨人の肩の上に立つ」とよく言われますが、研究や開発の深みや、それらにかかわる人々の層の厚み、そしてSIGGRAPH 40年の歴史が感じらる世界です。

先進的なアイデアとヒントの固まり。CG論文の数々

SIGGRAPHの本分は学会であり、毎年多くの先進的な論文が発表されます。論文については次のサイトを参照ください。公式のPDFでは各論文の1ページ目が無料で公開されており、概要と主要画像が把握できます。

SIGGRAPH 2013発表論文一覧
(Ke-Sen Huang氏らによる非公式なまとめ)
http://kesen.realtimerendering.com/sig2013.html

SIGGRAPH公式サイト「Technical Papers First Pages (44MB PDF)」
http://s2013.siggraph.org/sites/default/files/firstpages-lores.pdf

SIGGRAPHで採択される論文は純粋なコンピュータグラフィックス研究に加えて、最近では画像処理や動画処理など実写系のもの、近年公開された映画の特殊効果で使った最新技術の紹介、インタラクティブ技術に関するものなど多岐にわたります。昨年から論文のジャンル、カテゴリが増え、より広域の研究成果が取り上げられるようになりました。今年はとくに、音／音像、ファブリケーション（素材）、ディスプレイ技術、（3Dプリンタ技術そのものではなく）3Dプリンタ周辺のツールやアルゴリズムなどの技術が注目されていました。

Sketch2Scene : Sketch-Based Co-Retrieval and Co-Placement of 3D Models

ラフにスケッチした室内家具の配置メモから、適切な3Dイメージを生成する方法。室内のオブジェクト配置をスケッチベースで行う。スケッチした画像から正確で意味のある配置を行うしくみ。
http://cg.cs.tsinghua.edu.cn/sketch2scene/

Exposing Photo Manipulation with Inconsistent Shadows

写真の影を解析することで、合成写真かどうかを調べる技術。発表ではアポロの月写真が合成ではないことを紹介していた。
http://graphics.berkeley.edu/papers/Kee-EPM-2013-09/

Interpreting Concept Sketches

プロダクトデザインのラフスケッチから形状の可動部の操作や移動、動きをも三次元表現する手法。2枚のスケッチから可動部や移動可能な部品を解析し、中間構造を自動生成してくれる。
http://vecg.cs.ucl.ac.uk/Projects/SmartGeometry/concept_sketch/conceptSketch_sigg13.html

Make It Stand : Balancing Shapes for 3D Fabrication

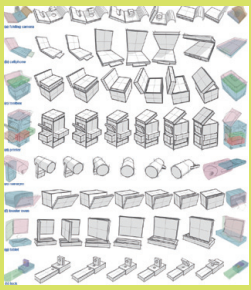
不安定な形状でも地面に立つように、オブジェクトのバランスを考慮して成形する手法。3Dプリンタで制作するフィギュア用の技術。適切な変形と、中身をくりぬくことでバランスを保つようにする。
http://igl.ethz.ch/projects/make-it-stand/

Parsing Sewing Patterns Into 3D Garments

ある人の洋服の型紙パターンから、違う服の型紙をサイズがあうように自動生成する方法。採寸したデータが一式あれば、あらゆるフィットする服の型紙ができる職人技なしくみ。
http://vis.berkeley.edu/papers/clopat/

Sketch2Scene（上がスケッチ。右が生成されたもの）

Exposing Photo Manipulation with Inconsistent Shadows



Interpreting Concept Sketches

Make It Stand

DIGITAL GADGET

Dense Scene Reconstruction with Points of Interest

Kinectセンサーで大量に撮影して3Dキャプチャの精度をあげ、より細かなディテールを再現する方法。

http://www.stanford.edu/~qianyizh/projects/scene.html

Worst-Case Structural Analysis

3Dプリントの落下テストで壊れたものの構造を解析し、壊れやすい部分、構造をあらかじめ指摘する手法。

http://www.cs.nyu.edu/~qnzhou/projects/StructAys/

Edge-Aware Point Set Resampling

3Dスキャナで取得した点データの集合から、エッジ（角）の部分を補正し高精細にする方法。

http://web.siat.ac.cn/~huihuang/EAR/EAR_page.html

来年のSIGGRAPH 2014（http://s2014.siggraph.org/）は2014年8月10～14日にカナダのバンクーバーで開催となります。またその前の2013年11月19～22日には、SIGGRAPH ASIA 2013が香港で開催されます。

これからもSIGGRAPHから登場するテクノロジが続々と新しいデジタル機器や応用例を生み出し、世の中の役に立ってほしいものです。SD

Parsing Sewing Patterns Into 3D Garments

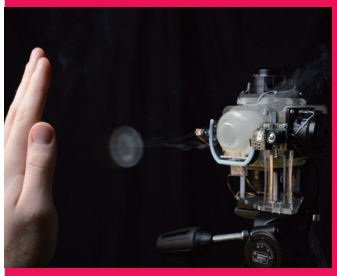
gadget 1

AIREAL

http://www.disneyresearch.com/project/aireal/

空気で感じるディスプレイ付加装置

AIREAL（Tactile Gaming Experiences in Free Air）はディズニーの研究所とイリノイ大学の共同研究。手の位置検出と、空気の振動を打ち出す機器の組み合わせで、ディスプレイ画面に触覚を加える装置。ディスプレイ画面に対するジェスチャインターフェースは、物に触った感覚がありませんが、AIREALを応用すると手のひらに何かを触ったかのような抵抗感を得ることができ、リアルな現象が体験できます。



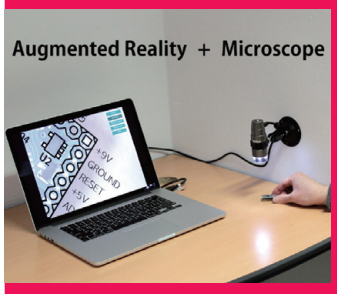
gadget 2

MicroTips

http://s2013.siggraph.org/attendees/emerging-technologies/events/microtips-augmenting-information-microscopic-inspection

部品ボードに拡張現実

MicroTips（Augmenting Information for Microscopic Inspection）は電子部品のボードを顕微鏡カメラで撮影すると、ボード上の微細な部品やコネクタ端子の情報などを写真に重ねて見ることのできる、AR（拡張現実）技術の応用例です。番号しか書かれていない端子の回路図と対比させた詳細情報や、多層基板の回路の内容など、目では見えないさまざまな情報を得ることができます。



gadget 3

IllumiRoom

http://research.microsoft.com/en-us/projects/illumiroom/

部屋中がディスプレイ！テレビとプロジェクションマッピングの組み合わせ

IllumiRoomはマイクロソフト研究所らによる、ゲームやテレビ体験の没入感を増すための技術。最近テレビの大型化が進みましたが、それでも大きさに限界があります。IllumiRoomでは、大型のテレビ画面だけでなく、周囲の壁面に投影したプロジェクタの映像を組み合わせます。たとえばゲーム画面やスポーツ中継などで、テレビ画面に集中しつつも、そこに表示しきれなかった広範囲の映像をプロジェクタで壁面投影し、臨場感、没入感を増す映像空間を作り出します。

gadget 4

Cloud Pink

http://everyware.kr/home/cloud-pink-exhibitions/

デジタルな空

Cloud Pinkは指先で雲を感じることのできる、デジタルで模倣した空と雲です。柔らかな素材で作られた、手を伸ばせば届く高さの天井スクリーンに、空と雲が投影されています。巨大な雲の映像は没入感があり、しばし会場に居ることを忘れさせてしまうような展示でした。リアルタイム映像はアート系プログラミング環境Processingとシェーダー言語GLSLを駆使して作られているそう。

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

結城浩の再発見の発想法 5

Interface

Interface—インターフェース

インターフェース（interface）とは、2つのものの間にある**境界面**を意味します（**図1**）。

ユーザインターフェース

たとえばユーザがアプリケーション（以下、アプリ）を使っているとき、アプリがユーザに何を見せるか。ユーザがアプリにどんなアクションを許すか。これがユーザとアプリの間にあるインターフェース、すなわちユーザインターフェースです（**図2**）。

大昔、まだコンピュータがパーソナルではなかった時代にはユーザインターフェースは大きな問題になりませんでした。コンピュータのパワーをできるだけ活用するために、人間のほうがコンピュータに歩み寄っていたからです。

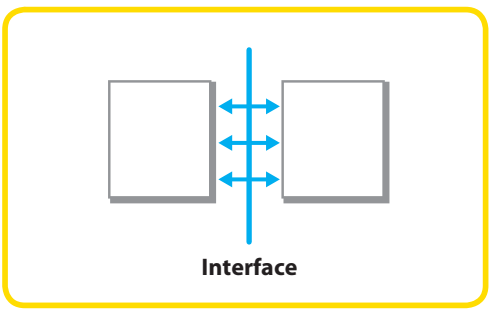
しかし、最近のコンピュータはすっかりパーソナルなものになり、取り扱う情報も大量になりました。そうなると、情報をユーザにどう提示するか、ユーザのアクションをどのようにコンピュータに伝えるかが重要になります。

Webサイトがリニューアルすると、使いやすくなったかどうかという議論が必ず起きます。それは、これまで慣れてきたユーザインターフェースが**変化**するからです。ユーザとWebサイトの間にあるユーザインターフェースが変化すれば、それに合わせてユーザも行動を変化させなければなりません。その変化を好ましいと思うユーザは「使いやすくなった」といい、好ましくないと思うユーザは「使いにくくなった」というでしょう。

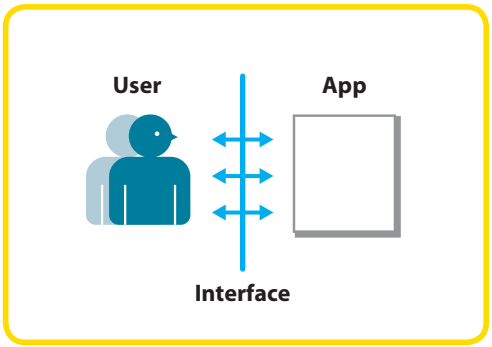
API

アプリと人間の間のインターフェースに対し、アプリとほかのプログラムの間のインターフェースもあります。これが**API**（Application Programming Interface）です（**図3**）。

▼図1　インターフェース



▼図2　ユーザインターフェース



アプリの機能はAPIで規定されますから、アプリ開発者にとってAPIは重要です。たとえば、Twitterアプリ開発者はTwitter APIを研究します。Twitterアプリが持つ「ツイートの投稿」や「タイムラインの取得」といった機能は、Twitter社が提供するAPIを通じて実現するからです。

アプリ開発者はAPIの変化にも敏感です。APIの変化に追従できないアプリは動作しなくなるからです。実際、2013年6月にTwitter APIが1.1へ切り換わったとき、それに追従できないアプリは動作しなくなってしまいました。

インターフェースの意義

インターフェースの意義の1つは、**インターフェースを変化させない限り内部を変更できる**点にあります。たとえば、ユーザインターフェースを変化させない限り、アプリの内部はいくらでも変更できます。バグを直すためであれ、機能アップに備えて構造を整理するためであれ、ユーザインターフェースが変化しなければ、ユーザから文句がくる心配はありません。

APIも同様です。Twitterアプリを企業や個人が開発できるのは、Twitter社が公開APIを定めているからです。Twitterの内部をTwitter社がいくら変化させたとしても、APIが変化しなければアプリ開発者の負担にはなりません。インターフェースの意義は境界面の相手に負担をかけないことでもあるのです。

システムへ

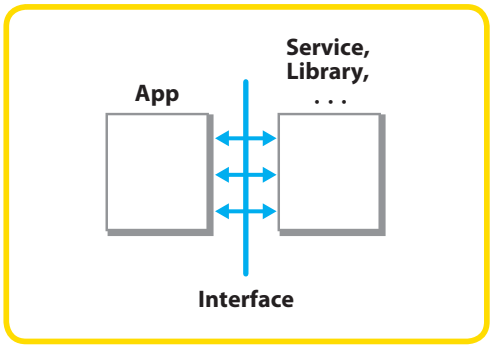
インターフェースの相手を**パートナーチェンジ**しやすくするのもインターフェースの意義の1つです。大きなプログラムを開発するとき、その一部を同じインターフェースを持つ別の部品（モジュール）と交換する場合はよくありますね。現代のソフトウェアは巨大なシステムです。それを崩壊させないためには複数モジュールに分けることが必須です。多くのモジュールを組み合わせるとモジュール間のインターフェースも多くなります（**図4**）。インターフェースを変えることなく個々のモジュールを交換できるなら、堅牢なシステムになるでしょう。ですから、インターフェースが鍵となるのです。

私たちが住んでいる社会も巨大なシステムです。それを崩壊させないようにするためには複数モジュールに分けることが必須です。もしかしたら、国が分かれ、会社や組織が分かれていることは、世界を崩壊させないために役立っているのかもしれません。国と国の間のインターフェース、会社や組織の間のインターフェースに相当するものは何でしょうね。

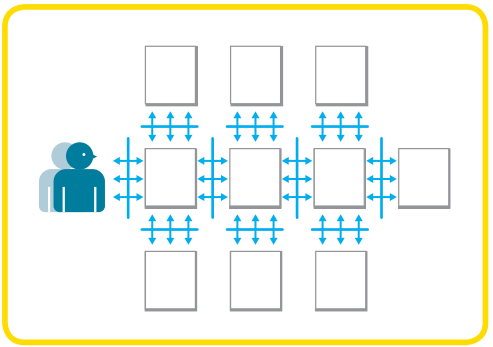
◆ ◆ ◆

あなたの周りを見回して、インターフェースが変化して困ったことはありますか。ぜひ考えてみてください。SD

▼図3　API



▼図4　システム



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6回 グッバイ・キーボード

その日、僕たちは眠ることも忘れ、ひたすら画面をクリックしていました。

深夜2時を過ぎているというのに、まるで真昼のようににぎやかなオフィスは、ある予感と、期待と、そして不安のないまぜになった感情を抱えながら、来るかもしれないその瞬間が来ることを待ち続けたのです。

「あ、出ました」

第一声を発したのは、山崎祥でした。

「え、出た？」

僕は慌てて自分のノートPCの前に座り直します。コマンド＋Rキーでリロード。みんなが画面の周りに集まってきます。

「こ、これは……」

固唾を呑んで僕たちが見つめていたのは、9.7インチのタブレット端末を持って自信満々に微笑むスティーブ・ジョブズの姿でした。

iOSは、静電容量タッチパネルのために専用設計された初めてのOSです。それがそのまま大きくなって販売されることで、これが果たして市場に受け入れられるか否かは、Appleという会社のみならず、コンピュータ業界全体の関心事でした。

僕たちがこの製品の登場を心待ちにしていた理由は、いくつかあります。それはいつの日かコンピューティングの基本が、キーボードのようなメカニカルなスイッチを打鍵する世界から、iPhoneのようにタッチスクリーンだけを基本とした世界へと移行していくであろうこと。その野心的な目標の最初のマスプロ製品が、2010年1月27日(米国時間)に発表された、このiPadだったからです。

僕たちが取り組んでいた、マルチタッチスクリーン上のユーザインターフェースへの研究が、本当に意義のあるものなのか、それを世の中に問いかけるための最初の試金石としてiPadの動向に注目していたのです。

極秘プロジェクト

「ちょっと待ってください、いまなんと仰ったのですか？」

その日、とある大手企業に呼び出された僕は、まったく不意を突かれました。最初は先方の担当者の言っている言葉の意味がわからず戸惑ったほどです。向き合ったY氏は事も無げに繰り返しました。

「ですから、私たちの提供する次世代のハードウェア、その骨格となるOSに相当する部分を、一緒に考えてほしいのです。我々はソフトの素人ですから」

当時から、僕たちUEIはさまざまなメーカーの技術戦略コンサルティングという仕事も手がけていましたが、そんな依頼は前代未聞でした。

まったく新しいソフトウェアプラットフォームをゼロから設計する。

なんと魅力的な、なんと素晴らしい仕事なのでしょう。そんな依頼をいただけた幸運を神に感謝するような気持ちでした。

かくしてプロジェクトK(仮称)がスタートしました。数十名のプロパー社員が各所から集められ、若手からベテランまでずらりとそろいます。対するは、僕、布留川英一、近藤誠を含むUEIの人間が若干名。士気は高く、誰もがやりがいのある仕事に燃えていました。

このプロジェクトにおける我々UEIの役割は、次世代プラットフォームの具体像を描き出すこと。そのためには、アプリケーションのあり方やコンピュータそのものの再定義まで多岐に渡る研究を行いました。

それまで、アプリケーションとは、単に独立したそれぞれのプログラムを呼び出すしくみに過ぎませんでした。しかしこれでは、ごく少数のアプリさえあればたいていの機能は満たせるはずです。ソフトウェアプラットフォームとして見た場合、アプリが増えれば増えるほどユーザ体験が向上していくのが望ましいのですが、現実にはアプリをあまりにもたくさんインストールするとユーザ体験は低下していきます。これでは新しいタイプのソフトウェアはなかなか産まれないことになります。

新しいソフトウェアプラットフォームは、新しいアプリケーションの概念によって実現されるべき、というのが僕の持論でした。

たとえばデータひとつとっても、通常はそれぞれのアプリケーションが別々のデータを管理するというのが一般的です。たとえば、乗り換え案内のコンテンツは乗り換え案内のコンテンツとして乗り換え案内アプリの中に内包されています。しかしまったく逆に、コンテンツとそれに対する処理を与えるアノテーションをシールという形で付加し、コンテンツとアプリケーションを分離するというアイデアを我々は提案しました。

シールはいわばブックマークレットのようなもので、コンテンツに対して貼り付けるとさまざまな機能をアドホックに行うことになります。たとえば、グルメ情報コンテンツが表示されているページに「お気に入り」シールを貼ると、そのお店に近づいた際にGPSが検知してアラートが鳴る、というような使い方です。

iPad発表の際にUEIですぐに制作した実物大モックアップ

システム手帳のようにカスタマイズでき、すべてのコンテンツは書き込んだり切り取ったりして自分のものとして活用することができます。こうすれば、コンテンツが増えれば増えるほど、アプリが増えれば増えるほど、ユーザができることは増えていくことになります。これが我々の考える新しいソフトウェアプラットフォームでした。

プロジェクトは急ピッチで進み、連日朝から深夜まで議論は続きます。シールがアプリになる、というアイデアは実はこの時点で生まれました。これはUEI独自のアイデアだったので我々が特許化し、実験アプリも多数開発したのです。

窮地

プロジェクトKは順調でした。

しかし、何が起きるかわからないものです。突然、本当に、まったく突然、このプロジェクトは中止を命ぜられてしまいます。プロジェクトを推進していた部長は社内で失脚し、左遷されました。さまざまな原因があったと言われますが、文字どおり全身全霊でプロジェクトに取り組んでいたメンバーはまるで魂の抜け殻のようになってしまいました。

「こんなことになってすみません」

プロパーのなかでも人一倍熱心だったKさんは言いました。

「いや、こんなこともあるでしょうよ」

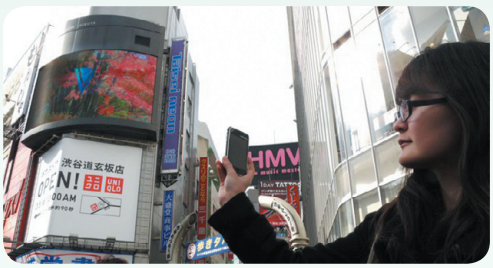
僕は慰めの言葉を口にしました。しかし内心、僕自身もこのプロジェクトに専念するため、膨大な時間を使っていました。その結果、肝心の経営のほうが疎かになっていたのです。しかもプロジェクトは中途放棄されたので、約束されていた報酬も支払ってもらうことができず、億単位の売り上げが吹き飛びました。

まさに絶体絶命。

さらにタイミングの悪いことに、辻が自らの教師への道を諦め、来年からUEIでソフトウェアの道を進むと決めたのは、この直前のことでした。さあどうしよう。本当に困りました。

即戦力

廃止されたプロジェクトルームから会社の現場に戻ると、状況は見るからに悲惨の一言でした。少し見ないうちに完全にデスマーチ化していたプロジェクト5本をいったんは全部僕が引き取り、期日までになんとか終わらせるという悲壮な戦いが始まりました。



ARゲームCRIMSONFOXを開発中の辻

CRIMSONFOXの実施場面。全参加者の位置がリアルタイムで把握できる

しかしどうしても、5本ものプロジェクトを同時に管理しようとすると手がたりません。そこで修士論文があらかた片付いた辻に助けを求めました。

「本当にヒドい話ですよ。あそこまで言っておいて雇うアテがなかったなんて」

今でも辻は当時を振り返ると怒ったような口調でこう言います。

辻が現場に入ると、開発は急に進み始めました。彼女は要領良く進行管理の補佐を行い、海外向けの翻訳作業をあっと言う間にこなしました。そのうち僕はゲームの台詞まわしを頼んだり、ストーリーを頼んだりするうちに彼女の学習能力が人並外れて高いことを悟りました。

経済産業省との絡みでやることになっていた渋谷を舞台にした代替現実感(AR)宝探しゲーム、「CRIMSONFOX」も、辻がほぼ1人でまとめあげます。ARゲームの場合は店舗との交渉がメインですが、これは非常にハードな仕事でした。

このゲームはニュースで大きくとりあげられ、話題になりました。これをベースとしたAR技術を応用したアプリ開発の依頼がひっきりなしに来るようになり、3年後の今でも、その技術を活かしたアプリが提供されています。

また、1年後にこれに興味を持った伏見遼平がUEIの門を叩きに来たわけです。なんとも縁を感じる話です。

Zeptopad Planner Note

そして2010年の春。辻がインターン生活を終え、ついにUEIにやってきました。正社員としての辻の初仕事は、発売されたばかりのiPadで動くZeptopadの最新版「Planner Note」の企画と設計。新入社員にやらせるにはいささか荷の重い仕事にも思えましたが、インターン時代に数々の活躍をしたこともあり、思い切って挑戦してみないか、と持ちかけたのです。

「やってみます」

辻はいつもこの調子でした。彼女と組んだの

は、ベテラン、布留川英一。彼なら新人をうまくフォローしながら複雑なプログラムを書いてくれるはずと思いました。

アプリの性質を正しく理解していた辻が書き上げた仕様書は、ほとんど差し戻し修正もなくプログラミングに入れるという見事なものでした。そうして出来上がった「Zeptopad Planner Note」はUEIのスマッシュヒットとなります。iPadという、より大きく相応しいハードウェアを得て、ようやくZeptopadは本来の企画目的「ホワイトボードを持ち運ぶ」ということを実現できるようになったのです。

そして、それまでどちらかというと裏方の存在だったUEIという会社が、それまでとは違ったジャンルの人々に知られるようになったきっかけでもありました。あるとき、IT系のカンファレンスに講師として呼ばれて、司会を勤めていらした有名大学の先生からこんな紹介を受けたのをよく覚えています。

「みなさんこちらのUEIという会社、ご存じないでしょうが、なんとあのZeptopad Planner Noteの開発元なのです」

入社早々、社名よりも有名なソフトを設計した辻に寄せられる期待とプレッシャーは相当なものだったと思います。しかし彼女は常に周囲の期待に、期待以上の成果で応えました。

レジスタンス

辻によるZeptopadがようやく形になったころ、僕は再びあの大手企業に呼び出されていました。ただし、正面玄関ではなく、直接オフィスのあるビルまで来てほしいという変わった連絡でした。

プロジェクトKは解散され、中心となったメンバーの多くは失意のもと転職したり、それぞれバラバラな部署に転属させられたりしていました。僕を呼び出したのは、そうして別部署に転属させられたKさんでした。

「自分は今、休憩時間で離席していることになってるんですよ」

辻が入社して最初に開発したZeptopad Planner Note

Kさんはおもむろにそう言いました。

え、どういうこと？と混乱していると、ガチャ、とドアが開き、Y氏が顔を覗かせました。

「こっちもバッチリ。この部屋も空調が不調で“使用禁止”にしておいた。ほかの連中もおっつけ来るはずです」

そしてぞろぞろと見覚えのある人々がその“使用禁止”の会議室に入ってきました。在りし日、プロジェクトルームでともに頭を悩ませた懐かしい面々でした。

「これはどういうことですか？」

「どうって、プロジェクト“K”ですよ」

Y氏はニヤリと笑って答えました。

「あれはもうなくなったんじゃ……」

「ええ、残念ながら。けれども、本当になくすべきだと思いますか？　あれは素晴らしいアイデアの宝庫でした。それを本当に棄ててしまうのが、いいと思いますか？」

「そりゃ僕だってできればやりたいですが」

「やりましょうよ。なんとか僕たちも力になります。会社として協力するのは難しいけど、アイデアは出します。協力できるところは協力します。アプリケーションとコンテンツの分離。あれは素晴らしい試みです。やってください。今日はそれをお伝えしたくて集まったんです」

僕は彼らの厚い友情に、目頭に熱いものがこみ上げて来ました。

いつか、必ずやり遂げる。僕たちは堅い誓いを交わしたのです。アテはぜんぜん、なかったのですが。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6回

ポインティングスティック付属

ThinkPadトラックポイント・キーボード & OKI Mini Keyboard

写真1　ThinkPad Bluetoothワイヤレス・トラックポイント・キーボード

はじめに

今回はポインティングスティックの付いたマウス機能も扱えるキーボードを紹介します。

ポインティングスティックとは、キーボードのGとHの間にあるポッチのことで、ノートPCのThinkPadのキーボードに付いているトラックポイントが有名です(**写真2**)。ポインティングスティックが付いているとキーボードのホームポジションから手を動かさずにマウス操作ができます。

写真2　トラックポイント

写真3　トラックポイントのキャップ

数あるポインティングスティック付キーボードの中からThinkPad Bluetoothワイヤレス・トラックポイント・キーボードとOKI Mini Keyboard Pro-Rを紹介します。

ThinkPad Bluetoothワイヤレス・トラックポイント・キーボード

Lenovo社が販売しているキーボードです。ThinkPadのキーボードをそのままBluetooth対応にしたような製品です(**写真1**)。英字配列バージョンと日本語配列バージョンが存在します。

特徴

特徴は次のとおりです。

- アイソレーションキーボード
- Bluetoothによるワイヤレスキーボード
- トラックポイント

最近のThinkPadのキーボードと同じアイソレーションキーボードで、キー同士が分離しており、フラットなキートップが特徴です。打鍵感はThinkPadと同様に静かで心地良いです。

Bluetoothによるワイヤレスキーボードです。バッテリは内蔵で、充電は付属しているUSBケーブルで行います。PCとのUSB接続は充電用途のみで、USBキーボードとしての利用はできないようです。

ThinkPad伝統のトラックポイントが付属しています。これによりマウスの機能がすべてまかなえます。マウススクロールも中クリックボタンを押しながら、トラックポイントを動かすことで実現できます。トラックポイントの交換用のキャップは付属していないようです。交換用のキャップは市販されており、ソフト・ドーム、クラシック・ドーム、ソフト・リムの種類があります[注1](**写真3**)。キャップの種類によって操作性が変わりますので、自身で試し

注1) キャップは1,000円ほどで購入できます。

てみて合っているものを選択すると良いです。

キーボード自体の重量は460gと軽く、大きさも小さく薄いので、持ち運びにも適していると言えます。なお、ThinkPadのキーボードは、USBタイプであればいくつも種類があります。本キーボードのUSBバージョンや、ThinkPad USB トラックポイントキーボードなどです。

入手方法

Lenovoの直販やAmazonから購入できます。値段は日本語配列、英字配列ともに14,000円ほどです。英字配列であれば、執筆当時、アメリカのLenovoでは$70ほどで販売していました。英字配列を複数まとめて購入するのであれば、輸入代行業者を通じてアメリカのLenovoから購入することも選択肢に含めると良いでしょう。

OKI Mini Keyboard Pro-R

ぷらっとホーム㈱が販売しているキーボードです。スティックポインタ機能が付属しているUSBキーボードです(**写真4**)。英字配列バージョンと日本語配列バージョンが存在します。

特徴

特徴は次のとおりです。

- スティックポインタ機能
- スクロール用ホイールの付属
- Caps Lock と Ctrl の入れ替え機能

ポインティングスティックであるスティックポインタがついており、マウスカーソルの移動をまかなえます。スティックポインタはThinkPad のトラックポイントと比べると多少操作が重く感じますが、使っているうちに慣れる程度の差です。どうしても慣れない場合は、OS側の設定でマウスカーソルの移動速度を速くすると良いです。交換用のキャップも販売されており、これもいくつか種類があります。

写真4　OKI Mini Keyboard Pro-R

キーボード下部にスクロール用のホイールが付いています(**写真5**)。ThinkPadのようにスティックポインタと中クリックの組み合わせでスクロールすることはできませんが、ホイールにより上下のスクロールができます。

写真5　スクロール用ホイール

キーボード裏側のDIPスイッチを切り替えることで、キーボードの機能でCaps Lockと左のCtrlを入れ替えられます(**写真6**)。そのほかの機能としてUSB Hubも内蔵しています。キーボードをメインに使うユーザにとって必要な機能は一通りそろったキーボードであると言えます。

写真6　DIPスイッチ

入手方法

インターフェースがPS/2のOKI Mini Keyboard III-Rが、ぷらっとオンラインとAmazonで購入できます。また、インターフェースがUSBのOKI Mini Keyboard Proは執筆当時、ぷらっとオンラインでのみ取り扱っているようでした。どちらの値段もおよそ10,000円です。なお、スティックポインタの交換用キャップは200円ほどです。

◆ ◆ ◆

ポインティングスティックが付属したキーボードは、今回紹介したもの以外にもいくつか存在します。たとえば、Samsung社のSamsung USB Portable KeyboardやUnicomp社のEnduraProなどです。筆者はこのポインティングスティックが付属したプログラマブルなキーボードが販売されることを切に願っています。SD

秋葉原発!

第36回

はんだづけカフェなう

RepRapを組み立ててみた

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

はじめに

今回は、本誌8月号の特集「3Dプリンタが未来を拓く理由」でも紹介をさせていただいた、フリー（自由）な3DプリンタであるRepRapの世界を紹介したいと思います。同特集には間に合わなかったのですが、筆者の手元には**写真1**のatomのRPパーツ（Reprapped Parts、つまりRepRapで出力された部品）がありましたので、これを組み立てました。

部品の調達

RPパーツはRepRapで出力されるので、すでに持っている人に出力を頼まなければなりません。しかし、それなりのボリュームですので、このパーツ（**写真2**）の出力にはのべ36時間を要するそうです。

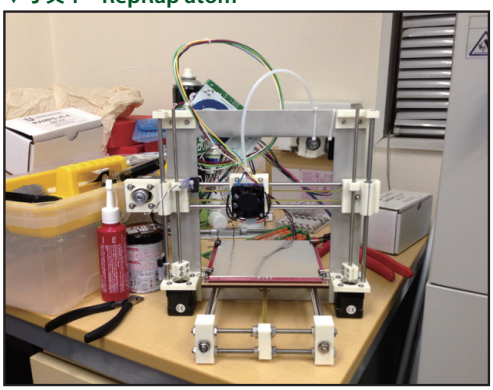
RPパーツ以外のネジといった金属の部品は「ビタミン」と呼ばれています。ビタミンのうち、ネジなどはホームセンターで入手できますが、ベアリング、全ネジ（寸切りボルトとか長ネジとも呼ばれます）、リニアシャフトといった部品は通販などで探して購入する必要があります。また、最近では加工済みのものが売っています[注1]が、筆者は金属板を加工してくれる店にフレーム板の切り出しを依頼しました。必要な長さの全ネジやリニアシャフトの調達は少し面倒で、こういったものを販売してくれる会社を見つけてきて発注するなどしなければなりません。

また、ホットエンドやエクストルーダのスタッドボルトなどの金属部品は、海外のRepRap関連の通販サイトで購入する必要がありました。これらは大規模に生産されていないので、たいていは売り切れ中です。筆者は、見かけたら今すぐいらなくても1～2個余分に買ってストックするようにしています。

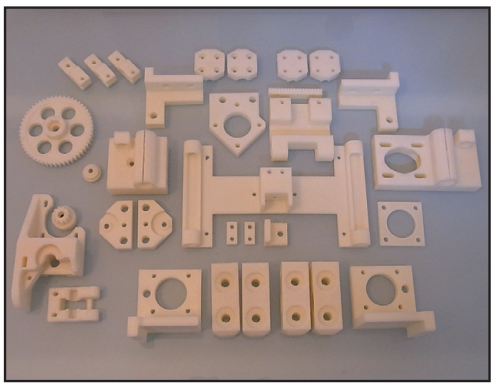
また、RepRapを動かすためのステッピングモータやそれをコントロールする基板なども別に入手する必要があります。これらは比較的容

注1）http://genkei.thebase.in/

▼写真1　RepRap atom



▼写真2　atomのRPパーツ



易ですが、選択肢がいくつかあって迷います。筆者はRAMPS 1.4というArduino MEGAのシールド型のものを選択しました。

必要な部品のリストは、atom RC版のリポジトリ[注2]に記されています。部品だけでは入手方法がピンと来ませんので、まずはお近くのホームセンターや東急ハンズなどのDIY用品を売っているお店でそろう物を消していき、手に入らなさそうな物はRepRap Community Japanのmixiにあるコミュニティ[注3]を参照したり質問したり、あるいはGenie氏という方がRC版の前にbeta版の部品を頒布なさったときの情報ページ[注4]を参照すると良いでしょう。

筆者の個人的な感想ですが、RepRapの組み立てで一番手間がかかるのは部品を集めてくるプロセスです。とくに初めてのRepRap組み立てだと、どこに何を頼んで良いのかわからないので相当に手間がかかると思います。

組み立て

部品が集まれば組み立てはそんなに難しくはありませんでした。とはいえ、留意しなければいけない点がいくつかあり、そういった情報はGenie氏が「atom組み立てのチップス」というブログ記事[注5]を書いてくれています。これと、完成図[注6]を参照しながら組んでいきます（**写真3**）。

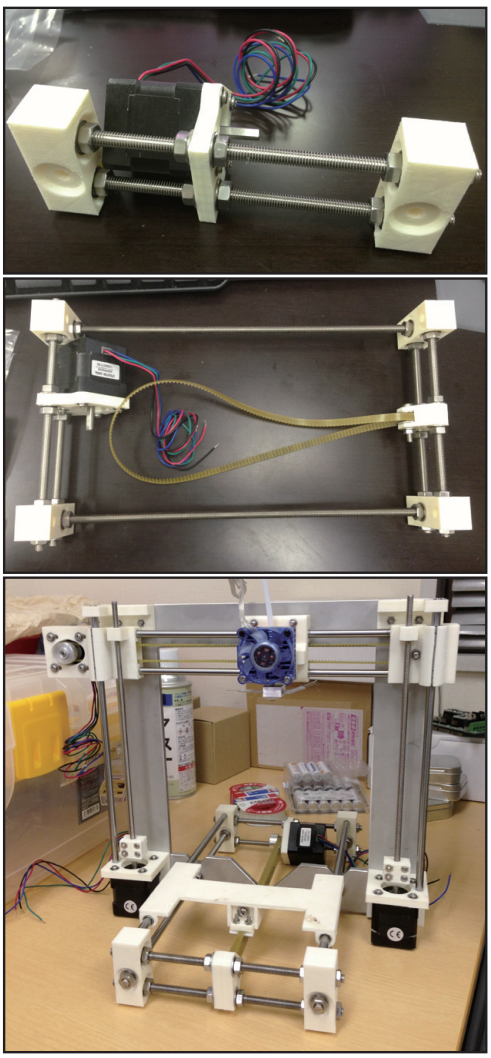
筆者は、テーブルから組んでいきました。プラモデルやキットのように全体の手順のことを考慮した手順書みたいなものがあるわけではないので、あとから必要な部品の取り付けを忘れていることに気づいたりします。ひとつひとつしっかり組んでいくと分解もたいへんですから、最初は部品がそろっていることを確認しつつ組み合わせているくらいの心構えでざっくり組んでみたほうが良いかもしれません。

筆者はしっかり組み上げて調整もしたあとに、エンドストップという3Dプリンタの原点（プリントするエリアの一番最初の場所）を確認するスイッチを取り付け忘れていることに気づきました。その場で再度バラす気力も湧かなかったので、結束バンドで仮留めして当初動かし始めました（**写真4**）。

コントローラ

筆者の場合、先述のようにコントローラとし

▼写真3　組み立ての様子



注2）https://github.com/hironaokato/atom_rc
注3）http://c.mixi.jp/reprap
注4）http://etherpod.org/blog/?page_id=5680
注5）http://etherpod.org/blog/?page_id=5736
注6）https://github.com/hironaokato/atom_rc/blob/master/stl/atom_rc_assem.STL

てRAMPS 1.4を選択しました。はんだづけが苦手な方はUltimachineというサイトの組み立て済みのRAMPS Pre-Assembled Kit Completeが売っていますので、これを手に入れると良いでしょう。RAMPSは先述のとおり、Arduinoのシールドですので、プリンタ側のファームウェアはArduinoのスケッチになります。ファームウェアとして人気なのは、Marlin[注7]というものです。

このMarlinのConfiguration.hを書き換えて、RAMPSを使うようにし、ステッピングモーターが何ステップ動けばRepRapが1mm動くのかといった情報を記入します。このステップ数は計算したり実際に動かして決めるのですが、筆者は先述のmixiのコミュニティに先人が記してくれていた値を参照して入力しました。

出力してみよう

特集でも記しましたが、3Dプリンタで出力するデータはSTLという形式が一般的です。このSTLを3Dプリンタのファームウェアが解釈できるデータであるGコードへの変換(スライス)はパソコンで行わなければなりません。3Dプリンタの操作(ホットエンドの温度や位置を変える操作)や、プリントアウトするデータ(Gコード)をプリンタに流し込むソフトウェアを使うのですが、RepRap界隈ではPrintrun[注8]やRepetier-Host[注9]が有名どころのようです。また、UltimakerのCuraもRepRapで使うことができるようになっています。筆者が使ってみたところ、Repetier-Hostのほうが気に入ったのでこちらを使っています。

まずは、先ほどMarlinに設定した値が正しいのかを確認するためにも、一辺が20mmのキャリブレーション用のキューブを出力してみました。左から結果を並べてみました。一番左の最初に出力してみたものは、ノズルが正しく移動していないことに気づいたので、すぐに出力を中止しました。プリンタのベルトの張りが甘かったらしく、ステッピングモータの動きが正しく伝わらなかったみたいです。

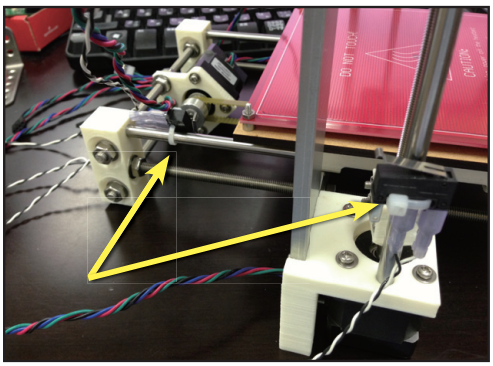
これを直して試行したものが真ん中のものです。XY方向(平面)のできは納得で、測ってみたところ20mmだったのですが、高さが20mmありません。よくよく原因追及をしてみると先ほどMarlinの設定ファイルに書いたZ方向のステップ数が誤りでした。

こうして調整をした結果、どうにか動いたと言える程度の出力ができるようになりました。ちなみに、これを1個出力するのに20分程度が必要です(**写真5**)。

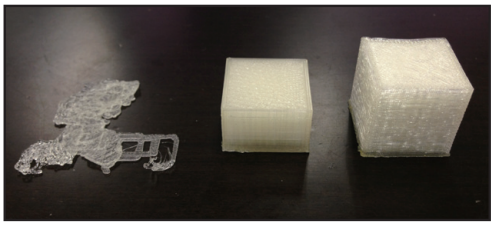
Repetier-HostにはSlic3rとSkeinforgeという2種類のスライサ(STLからGコードを生成するソフト)が添付されており、手軽に使えるようになっています。このスライサにもいろいろな種類があるのですが、前述の2つにに加えてKISSlicerというスライサもよく見かけます。RepRapのRPを出そうとSTLファイルを

注7) https://github.com/ErikZalm/Marlin

▼写真4 結束バンドで仮止めしたエンドストップ



▼写真5 調整用の20mmキューブ



注8) https://github.com/kliment/Printrun
注9) http://www.repetier.com/

Slic3rでスライスしようとしたところ、うまくSTLを読み込んでくれないため、KISSlicerを使ってGコードを出力してみたりしています。

フィラメント

atomの動作確認や調整をする際には、筆者の手元にあったMakerbotのPLAフィラメントを使用して調整していました。しかし手元にあった1.75mm径のフィラメントはこの透明のものだけだったのです。色つきのものも使ってみたくなり、国内で手軽に買えるフィラメントがないものかといくつか買って試してみました。

写真6は、国内で買ったフィラメントを使ってみて、出力に失敗したものです。どうやら、PLAがノズルから出てからなかなか冷えず、まだ柔らかいうちに上に積層しようとノズルが来たときに引っかけてしまうことでインフィル（プリントの中に作られる支え）がグダグダになってしまうようです。中の支えがグダグダなので、一番上の面を正しくプリントできていません。

色つきのフィラメントですので入っている塗料が原因かと思い、同じ店から買った透明のものも使ってみました。しかし、グレーほどひどくはありませんが、やはり思うような結果を得られませんでした。フィラメントが十分に冷えて硬化してないようなので、プリントの速度を落とすのも手かもしれませんが、速度を詰めていないところをさらに遅くするのは気が進みません。印刷したところをすぐに冷やすようなファンをプリンタに追加してみたくはありますが、当面これはお蔵入りにしておきます。今のところ、国内の通販で手軽に買えて品質も良いという評判のものが見当たりませんので、当面は面倒でも評判の良いフィラメントを海外から通販で購入して使いたいと思います。

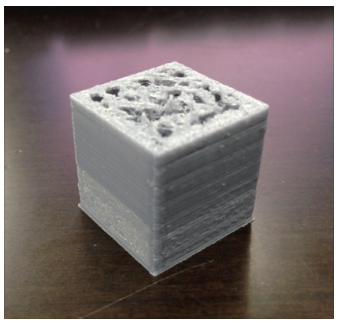
まとめ

RepRapの自作のハードルは、やはりRPのみならず部品集めだと思います。この問題はキットを買うことで解決できるのですが、筆者が組んだほかのRepRapのキットではRPが割れていたり（**写真7**）、部品が足りないといった問題も見うけられました。また、このキットに入っていたベアリングは筆者がatom組み立てのときに使ったものと比べて明らかに質が悪く、動きがスムースではありませんでした。キットを買うにしても、質の良いものを、実績のあるメーカーから購入したほうが良いでしょう。

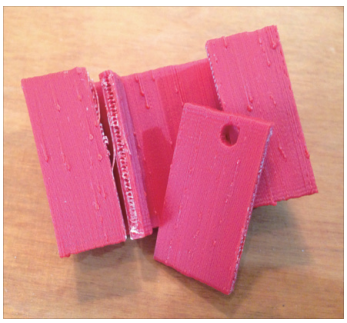
組み立てで困ったことがあれば、先のRepRap Community Japanのmixiコミュニティで質問すれば解決できるかもしれません。もし自分でできるか心配に感じるのであれば、atomのキットとワークショップのセット[注10]に参加するのも1つの手段です。

ここまで苦労して3Dプリンタを使わなくても、と感じる方も多いかもしれませんが、筆者はRepRapを組み立ててみて3Dプリンタの構造への理解はもちろん、いろいろなノウハウを身に付けることができました。Makerbotの Replicator 2にあった初期不良を発見し、サポートに必要な交換部品を送ってもらうことができたのもRepRap組み立てで得た知識があってのことだと思っています。3Dプリンタを使うために必要な知識と技術を身に付けるためにもRepRapの自作をお勧めします。SD

▼写真6　失敗した20mmキューブ



▼写真7　割れていたキットのRP



注10）http://my.ebshop.info/

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 2013年10月17日です。プレゼントの発送まで日数がかかる場合がありますが、ご了承ください。

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

01 Samsung SSD 840 EVO ベーシックキット 250GB

1名

SATA 3.0 対応（SATA 2.0/1.0 互換）の SSD（Solid State Drive）です。サムスン製の Toggle DDR 2.0 の高速 NAND フラッシュと、第 5 世代 MEX コントローラを搭載し、NAND フラッシュの一部の領域を高速バッファとして使用する新機能「ターボライトテクノロジー」により、連続書き込みのパフォーマンスを最大 3 倍近く向上させました。

提供元 日本サムスン（販売特約店 ITG マーケティング） **URL** http://www.itgm.co.jp

02 Super Win Utilities 3

3名

Windows 起動高速化、不要ファイルの全自動クリーニングなどの機能を搭載した PC 高速化ソフトウェア。最新版ではデフラグツール、アプリ起動高速化ツール、SSD 故障監視ツールをさらに強化。Windows 8/7/Vista（32、64 ビット）と XP（32 ビット）に対応。

提供元 インターコム **URL** http://www.intercom.co.jp

03 はてな Tシャツ 2013

5名

㈱はてなが毎年作成しているオリジナル T シャツの 2013 年版。「はてなブログ」「はてなロゴ」「はてなブックマーク」の 3 パターンのデザインのいずれかが当たります。色とデザインの組み合わせは写真と異なる可能性があります。

提供元 はてな **URL** http://www.hatena.ne.jp

04 雲人 Tシャツ

2名

日本仮想化技術㈱の Facebook ページ開設を記念して作られた雲人（くもんちゅ）のロゴ入り T シャツ。M サイズ（水色）と L サイズ（青色）の 2 種類です。アンケートの中で希望サイズをご記入ください。

提供元 日本仮想化技術 **URL** http://virtualtech.jp

05 実践 Vim

2名

Drew Neil 著、新丈 径 訳／
A5 判、400 ページ／
ISBN = 978-4-04-891659-2

Vim のエディタとしてのコア機能をマスターするためのレシピ集。モードの基本的な役割、ファイル管理、カーソル移動や検索のスピードアップなどテーマごとに実践的なテクニックを紹介します。

提供元 アスキー・メディアワークス **URL** http://asciimw.jp

06 はじめての 3Dプリンタ

2名

水野 操、平本 知樹、神田 沙織、野村 毅 著／
B5 判、128 ページ／
ISBN = 978-4-7741-5973-7

パーソナルファブリケーションに興味がある方を対象に、3D プリンタのしくみから 3D データを簡単に作成できる Web サービス、3D 造型物出力サービスの活用方法を解説します。

提供元 技術評論社 **URL** http://gihyo.jp

07 独習 Linux 専科 サーバ構築/運用/管理

2名

中井 悦司 著／
B5 変形判、384 ページ／
ISBN = 978-4-7741-5937-9

自分で試しながら Linux のしくみを根底から学べる 1 冊。サーバ利用を中心にインストール／環境設定、プロセスとジョブ管理、ファイルシステム、サーバ管理、アプリサーバの動作などを取り上げます。

提供元 技術評論社 **URL** http://gihyo.jp

第1特集

Vimを使いこなしていますか?
Vim至上主義

UNIXユーザにとって、最初に使うエディタはviではないでしょうか。多くのLinuxディストリビューションにも標準的に搭載されている便利なエディタです。サーバエンジニアのデフォルトエディタとも言えます。viさえあれば最高という方も多くいらっしゃいます。

viは非常にシンプルです。ある意味使用目的が特化しているため、現代風な使いやすさをもって多くの点で改善されたVimが主流になってきています。多くのユーザにとって、いきなり超一流の使い手になることはいろいろな面からみてハードルが高いかもしれません。そこで本特集ではVimの達人達の使いこなしを紹介します。彼らの経験と実績から生まれたたくさんのヒントを掲載しましたので日頃のソフトウェア開発などに役立ててください。

CONTENTS

第1特集 Vim至上主義

第1章 これからVimを始めたいあなたに

Writer 後藤 大地(ごとう だいち) BSDコンサルティング㈱ Twitter:@daichigoto、@BSDc_tweet

プログラミングを行う開発者はもちろん、HTMLやCSSを扱うWebデザイナ、書籍や雑誌・ニュースなどの執筆を職業にしている物書きの方、企業システムの計画立案からプロジェクト推進まですべてを束ねるマネージャ、企業システムの管理運用を任されているメンテナ、挙げていけばきりがないわけですが、文字を入力する作業が必要なすべての方にとって作業効率を大きく左右する超重要なソフトウェアが「エディタ」です。世界中には数限りないエディタがありますが、その中でもとくに異色で、使うのに訓練を必要とし、その関門を乗り越えた者に信じられないほどの高効率を与えるエディタが「Vim」です。本特集ではこのエディタの魅力と、最小限の努力で最大限の効果を得るためのさまざまなティップスやテクニックを紹介します。

1 Vimが優れている理由:ラインエディタの特性を継承

みんな大好き「攻殻機動隊」。その世界観もさることながら、やはり憧れるのは、映画「GHOST IN THE SHELL/攻殻機動隊」において、ドクター・ウィリスがキーボードを叩き出す瞬間です。ドクター・ウィリスの腕は義手になっており、タイピングの瞬間に1本の指がそれぞれ3本の指に別れて猛烈な速度でタイピングを始めます。1本が3本になるわけですから、合計で30本。入力効率は3倍に違いない(違)。

人間のままではドクター・ウィリスのような速度でタイピングすることは無理ですが、実は限りなくそれを可能にしてくれるソフトウェアがあります。そう、それが「Vim」です。物理的な入力速度には限界がありますが、Vimのコマンドモードを駆使すると、物理的な入力だけでは得られない効率での作業が可能になります。

……ドクター・ウィリスの使っているエディタがVimだった場合にはもはや太刀打ちできませんが、Vimを習得すればVimを使わないドクター・ウィリスには対抗できそうです……

Vimは「vi」と呼ばれるエディタを模倣し、さらに多くの機能を追加していった多機能エディタです。現在使われているエディタの中では「nvi」と呼ばれるエディタが、オリジナルのviに近いエディタです。nviはFreeBSD、NetBSD、OpenBSDのデフォルトのエディタとして使われています。nviはviと互換性が高く、機能を追加していったVimと違って軽量で高速という特徴があります。vi系エディタはLinuxやFreeBSDに最初からインストールされていることがほとんどで、sshでログインしてリモート管理する場合には欠かせないエディタでもあります。すでに使われている方は多いでしょう。Mac OS Xにもデフォルトでインストールされているので、使われている方は少なくないと見られます。

viには現在主流のエディタにはない異色な特徴があります。viは「ex」と呼ばれるエディタを拡張したものです。exはライン指向のエディタで(ときにラインエディタと呼ばれます)、行ごとにテキストを編集します。exは「ed」と呼ばれるラインエディタに影響を受けて開発されたラインエディタです。edは「QED」というラインエディタに影響を受けて開発されています。

つまり、Vimやviはラインエディタの系譜から機能を拡張していったエディタということになります。現在のエディタはラインエディタの系譜ではありませんので、Vim/viとそれ以外のエディタではそもそもの設計思想が異なっています。

Vim/viの動作がほかのエディタと比較して異色的に見えるのにはこうした理由があります。

ラインエディタでの編集

たとえばedと呼ばれるラインエディタでファイルを編集してみましょう。ファイルの一部を書き換えたい場合、**図1**のように操作します。edはラインエディタですので、出力は1行だけです。わかりにくいですが、edを実行したあとはedからの出力が1行、ユーザによる入力が1行という操作が繰り返されています。edはとても寡黙で、必要がなければ何も表示しません。

viはこうしたラインエディタを拡張したものです。edの操作だけみると、edの操作を引き継いだviはエディタとしては欠陥品のように思えてきますが、そうはなりませんでした。テキストの入力とは別にコマンドを入力できるというのは、編集という作業においてとても強力なものだったからです。

viでは1行だけではなくターミナル全体を編集領域として使います。ファイルの中身が見えた状態で編集できます。edやexと比較すると大きな進歩といえます。こういったエディタはスクリーンエディタと呼ばれています。

viは分類上はスクリーンエディタということになりますが、基本的にex/edといったラインエディタを拡張したような設計になっているのでex/ed時代の操作が使えるようになっています。つまり、**図1**のように「3」や「s/0003/000a/」、「w」、「q」をテキストの入力ではなく「コマンドの入力」として扱う機能を持っています。

▼図1　edによるファイル書き換えの例

```
% cat data
0001 8485 494
0002 1124 570
0003 9720 956   ← この行の0003を000aへ変更したい
0004 3780 218
0005 9335 769
% ed data       ← ed(1)でファイルを編集開始
70              → 開いたファイルサイズが表示される
3               ← 3と入力して3行目に移動
0003 9720 956   → 3行目の内容が表示される
s/0003/000a/    ← 0003を000aへ変更せよという命令
w               ← 保存せよという命令
70              → 保存したファイルのサイズが表示される
q               ← edを終了
% cat data
0001 8485 494
0002 1124 570
000a 9720 956   ← 編集されていることがわかる
0004 3780 218
0005 9335 769
%
```

コマンドモード

vi/nviではこれを実現するために「入力モード」と「コマンドモード」という2つのモードを持っています。入力モードのときはテキストをテキストとして入力できますが、コマンドモードになると入力されるテキストはコマンドとして解釈されます。edやexで実行できることはviでもできます。

皆さんは少なくとも1回はsedやgrepを使ったことがあるかと思いますが、sedとgrepはedの機能を個別のコマンドとして切り離したものです。edでの機能が便利だったので、外にその機能だけを持っていったコマンドです。sedは置換(swap)を意味する「s」を付けてsed、grepは正規表現に一致した行を表示するというedの命令「g/re/p」がコマンドになったものです。エディタを操作していてエディタの中でgrepやsedが使えれば良いのにと感じることがあるかと思いますが、edはもともとその機能を提供していますし、edを設計思想の源流においているviにもその機能が最初からエディタ内に含まれています。

Vimのモード

そしてVimです。Vimはviの設計をベースにさまざまな機能を取り込んでいきました。たとえばnviの提供するモードは「入力モード」と「コマンドモード」だけですが、Vimでは10を超えるモードが提供されています。基本となるモードは「挿入モード」「コマンドモード」「コマンドラインモード」「ビジュアルモード」です。

とくに「ビジュアルモード」はVimの操作性を引き上げる重要なモードです。Vimではコマンドモードはノーマルモードと呼ばれていますが、慣れないうちは混乱するのでここでは「コマンドモード」とviでの呼び方を踏襲しておきます。

Vimの機能は、一生訓練を続けても使いこなすことは不可能ではないかと思われるほどたくさんあります。しかしこれは逆に、Vimは使い続けるほどスキルアップにつなげることができることを意味しています。最初は十中八九、間違いなく慣れない操作に苛立ちを感じるでしょう。しかし、スキルアップし続ければいずれはドクター・ウィリスのように高速な操作ができるんだ、と気力を奮い立たせてこの異色なエディタと四つに組んでいきましょう。

Vimの環境構築（.vimrcの書き方）

Vimは無限にも近いカスタマイズが可能なエディタでもあります。Vimの設定ファイルはホームディレクトリ直下の「.vimrc」というファイルです。ここではいくつか基本的な設定を解説しておきます。Vimのデフォルトの動きは現在のスクリーンエディタに慣れたユーザにとってはハードルが高いので、現在のエディタに近い動作をするようにしておいたほうが何かとストレスがなくなります。

viとの互換モードではなく、Vimの機能をフルに発揮できるようにnocompatibleを設定しておきます。

```
set nocompatible
```

Vimはデフォルトでは挿入モードで、バックスペースキーを押しても文字を削除できません。これは現在のスクリーンエディタに慣れたユーザに強いストレスを与えます。backspaceにstartを設定すると、挿入モードでバックスペースキーによる文字の削除ができるようになります。

```
set backspace=start,eol,indent
```

eolは改行も削除できるようにする指定です。行の先頭でバックスペースキーを押すとその行が1つ前に行にくっつくようになります。indentはインデントモードでインデントを削除できるようにする指定です。

Vimはデフォルトでは行の先頭または行の終わりにカーソルが到達しても、そこを超えて前の行や次の行に移動するといった設定になっていません。これも現在のスクリーンエディタに慣れたユーザに強いストレスを与えます。バックスペースキーやスペースキー、矢印キーでの移動が現在のスクリーンエディタと同じになるように「whichwrap=b,s,[,],<,>,~」を設定しておきます。よくわからないうちはこういう設定をするものだと思っておくと良いでしょう。

```
set whichwrap=b,s,[,],<,>,~
```

Vimを活用するにはマウスを使わないのが1つのポイントです。また、マウスとの連動機能はターミナルアプリケーションの実装に依存するため問題になることも多く、無効にしておいたほうが何かと無難です。

```
set mouse=
```

マウスは便利な機能ですが、画面を見ながらのマウスの細かい操作は実は大きなストレスになります。マウスを使わないでキー入力(**表1**)のみで済むなら、それにこしたことはありません。

Vimは強力なシンタックスハイライト機能を持っています(**図2、3**)。この機能を使いたい

▼表1　Vimの代表的なキー入力

記号	対応するキー	対応するモード
b	バックスペースキー	ノーマルモード、ビジュアルモード
s	スペースキー	ノーマルモード、ビジュアルモード
<	← キー	ノーマルモード、ビジュアルモード
>	→ キー	ノーマルモード、ビジュアルモード
[	← キー	挿入モード、置換モード
]	→ キー	挿入モード、置換モード
~	~ キー	ノーマルモード

からVimを使うというプログラマは少なくありません。syntax onでシンタックスハイライトを有効にするとともに、検索時にはハイライトが有効にならないようにしておきます。検索時のハイライトは見難くなることが多いように思います。

```
syntax on
set nohlsearch
```

シンタックスハイライトを有効にした結果、どのような色が表示されるかはターミナルアプリケーションの実装に依存しています。まったく同じ設定ファイルでも、利用するターミナルを変更すると表示される色が変更されることがあります。次にバックグラウンドが黒色のターミナルでの色設定例を掲載しておきます(**図4、5**)。

図6の設定はターミナルの背景が白色の場合の色設定例です(**図7**)。こうした設定はターミナルアプリケーションごとに見やすさや見にくさが変わりますので、お使いのターミナルアプリケーションに合わせて調整してください。

次の設定はVimのステータスラインを設定しています。laststatus=2で常にステータスラインを表示させています。statuslineはステータスラインに何を表示するかの指定です。**図8**では長いパスでファイル名を表示させています。似たような名前のファイルを編集していると、今どこのディレクトリのファイルを操作していたのかわからなくなることがあるからです。このあたりは好みの問題でもあります。

```
set laststatus=2
set statusline=%F%r%h%=
```

▼図2　syntax onを設定する前

```
ssh — ttys000
set nocompatible
set backspace=start,eol,indent
set whichwrap=b,s,[,],<,>,~
set mouse=
~
~
~
~
~
".vimrc" 4L, 87C
```

▼図3　syntax onを設定した後

```
ssh — ttys000
set nocompatible
set backspace=start,eol,indent
set whichwrap=b,s,[,],<,>,~
set mouse=
syntax on
set nohlsearch
~
~
~
".vimrc" 6L, 112C
```

▼図4　バックグラウンドが黒のときの色設定例

```
highlight StatusLine ctermfg=black ctermbg=grey
highlight CursorLine ctermfg=none ctermbg=darkgray cterm=none
highlight MatchParen ctermfg=none ctermbg=darkgray
highlight Comment ctermfg=DarkGreen ctermbg=NONE
highlight Directory ctermfg=DarkGreen ctermbg=NONE
```

▼図5　図4の設定後のターミナル表示

```
ssh — ttys000
set nocompatible
set backspace=start,eol,indent
set whichwrap=b,s,[,],<,>,~
set mouse=
syntax on
set nohlsearch
highlight StatusLine ctermfg=black ctermbg=grey
highlight CursorLine ctermfg=none ctermbg=darkgray cterm=none
highlight MatchParen ctermfg=none ctermbg=darkgray
highlight Comment ctermfg=DarkGreen ctermbg=NONE
highlight Directory ctermfg=DarkGreen ctermbg=NONE
~
~
```

set numberで行番号を行の先頭に表示させることができます(**図9**)。行数が邪魔な場合はset nonumberとします。

```
set number
```

incsearchはインクリメンタル検索を有効にする設定です。この設定をしておくと、コマンドモードから「/文字列」のように検索を実施した場合、入力されるごとにインクリメンタルに検索結果を表示するようになります(**図10、11**)。ignorecaseを指定しておくと大文字を小文字を区別せずに検索できます。

```
set incsearch
set ignorecase
```

Vimは各種モードで補完入力を実施できます。wildmenu wildmode=list:fullはコマンドラインモードでの補完表示の形式をどのようにするかの指定です。この設定が比較的にわかりやすいかな

▼図6　背景が白色の場合の色設定例

```
highlight Normal ctermbg=grey ctermfg=black
highlight StatusLine ctermfg=grey ctermbg=black
highlight CursorLine ctermfg=darkgray ctermbg=none cterm=none
highlight MatchParen ctermfg=none ctermbg=darkgray
```

▼図7　図6の設定後のターミナル表示

ターミナル — ssh — 62×13

```
set nocompatible
set backspace=start,eol,indent
set whichwrap=b,s,[,],<,>,~
set mouse=
syntax on
set nohlsearch
highlight Normal ctermfg=black
highlight StatusLine ctermfg=grey ctermbg=black
highlight CursorLine ctermfg=darkgray ctermbg=none cterm=none
highlight MatchParen ctermfg=none ctermbg=darkgray
~
-- INSERT --
```

▼図8　ステータスラインを表示させたところ

```
# version: $Revision: 34532 $

.include "../../mk/articles.base.mk"
/z/localdocs/SoftwareDesign/201310/vim-01/Makefile
:set nonumber
```

▼図9　行番号表示

ssh — ttys000

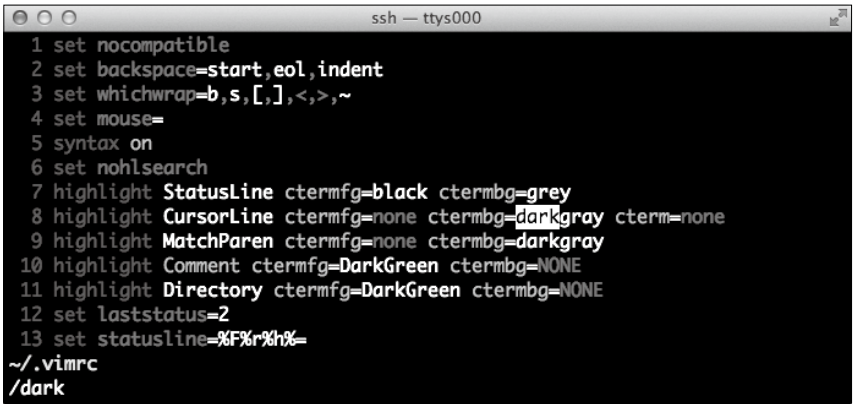
```
  1 set nocompatible
  2 set backspace=start,eol,indent
  3 set whichwrap=b,s,[,],<,>,~
  4 set mouse=
  5 syntax on
  6 set nohlsearch
  7 highlight StatusLine ctermfg=black ctermbg=grey
  8 highlight CursorLine ctermfg=none ctermbg=darkgray cterm=none
  9 highlight MatchParen ctermfg=none ctermbg=darkgray
 10 highlight Comment ctermfg=DarkGreen ctermbg=NONE
 11 highlight Directory ctermfg=DarkGreen ctermbg=NONE
 12 set laststatus=2
 13 set statusline=%F%r%h%=
~/.vimrc
".vimrc" 14L, 425C
```

と思います(もちろんこのあたりは好みの問題です)。補完表示されている例が**図12**です。

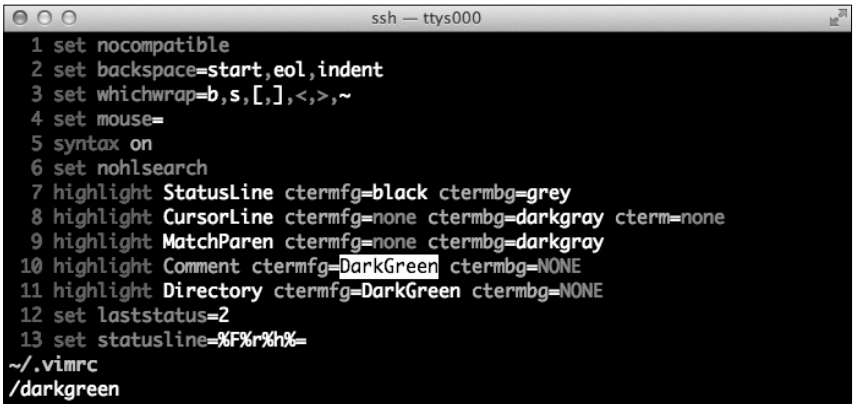
```
set wildmenu wildmode=list:full
```

次の設定はちょっとしたショートカットキーの設定です。まず、タブキーで15文字分右へカーソルを移動、シフトキーを押しながらタブキーを押して15文字分カーソルキーを左へ移動です。

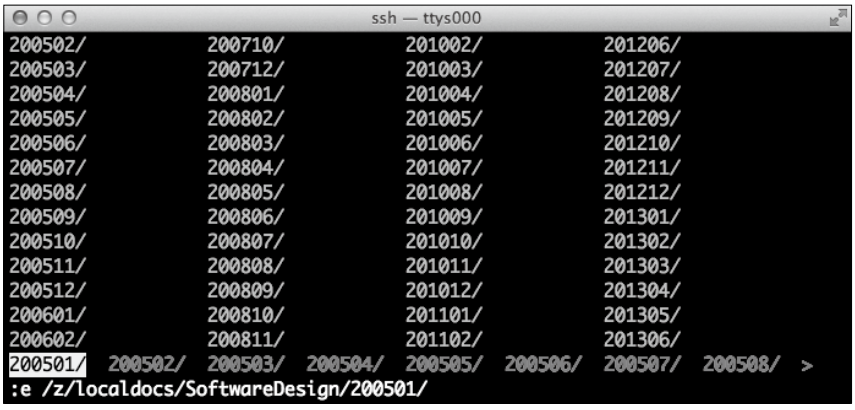
▼図10　/darkと入力した段階ではdarkgrayが一致している



▼図11　/darkgreenと入力すると次のDarkGreenが一致している



▼図12　タブを押してファイル一覧を表示させたところ



HTMLファイルやXMLファイルのように改行せずに長い文字列を入力または編集するようなケースではカーソルを高速移動させる上でこのショートカットが便利です。

```
nmap <silent> <Tab> 15<Right>
vmap <silent> <Tab> <C-o>15<Right>
nmap <silent> <S-Tab> 15<Left>
vmap <silent> <S-Tab> <C-o>15<Left>
```

次のショートカットは複数のファイルを同時に編集してる場合に、次のファイルに移動するためのショートカットです。Ctrl-nで順々に次のファイルへ編集対象を切り替えます。Vimはコマンドラインモードでの操作が履歴に入りますので、似たような編集をする場合には対象ファイルを一気に開いて作業をしたほうが効率が良いことがあります。ファイル間の移動をショートカットキーに割り当てておくと便利です。

```
nmap <silent> <C-n>      :update<CR>:bn<CR>
imap <silent> <C-n> <ESC>:update<CR>:bn<CR>
vmap <silent> <C-n> <ESC>:update<CR>:bn<CR>
cmap <silent> <C-n> <ESC>:update<CR>:bn<CR>
```

これまでの設定をまとめると**図13**のようになります。

最低限のVimの設定ファイルとしてはこのあたりを押さえておけば良いかと思います。

お勧めチートシート

Vimはさまざまな機能を提供していますが、よく使う操作はある程度絞られてきます。いくつかのコマンドや操作は手癖として身についてきますので、何も考えずにタイプできるようになります。**表2〜4**に使われることが多いと思われる操作をまとめておきます。

viでは「:行数,行数s/パターン/置換後文字列/g」のように行数でコマンドの適用範囲を指定できます。Vimでもこの方法が使えますが、ビジュアルモードで選択したものに対して置換を実施したほうが何かと便利です。行数を指定する場合、結局のところその行数を調べるために時間をかけることになり、あまり効果的ではないところがありました。ビジュアルモードを活用することで範囲指定をしたうえでもコマンドの実行がとても直感的でわかりやすく高速なものになります。

またカーソルの移動に関してはカーソルキーを使っての移動を極力しないというのが、高速編集への1つのポイントです。数字を指定しての繰り返し移動や、/や?による検索を駆使して一気に移動します。最短の時間で最大の効果を得る癖を付けておくのがポイントです。SD

▼図13　すべての設定をしたところ

```
set nocompatible
set backspace=start,eol,indent
set whichwrap=b,s,[,],<,>,~
set mouse=
syntax on
set nohlsearch
highlight StatusLine ctermfg=black ctermbg=grey
highlight CursorLine ctermfg=none ctermbg=darkgray cterm=none
highlight MatchParen ctermfg=none ctermbg=darkgray
highlight Comment ctermfg=DarkGreen ctermbg=NONE
highlight Directory ctermfg=DarkGreen ctermbg=NONE
set laststatus=2
set statusline=%F%r%h%=
set number
set incsearch
set ignorecase
set wildmenu wildmode=list:full
nmap <silent> <Tab> 15<Right>
vmap <silent> <Tab> <C-o>15<Right>
nmap <silent> <S-Tab> 15<Left>
vmap <silent> <S-Tab> <C-o>15<Left>
nmap <silent> <C-n>      :update<CR>:bn<CR>
imap <silent> <C-n> <ESC>:update<CR>:bn<CR>
vmap <silent> <C-n> <ESC>:update<CR>:bn<CR>
cmap <silent> <C-n> <ESC>:update<CR>:bn<CR>
```

▼表2　挿入モードでの操作

キー	意味
Esc	コマンドモードへ移行

▼表3　コマンドモードでの操作

コマンド	意味
:	コマンドラインモードへ移行
/文字列	コマンドラインモードへ移行。カーソルの位置からファイルの終わりへ向かって検索
?文字列	コマンドラインモードへ移行。カーソルの位置からファイルの最初へ向かって検索
i	挿入モードへ移行。カーソルの位置から入力開始
a	挿入モードへ移行。カーソルの次の位置から入力開始
I	挿入モードへ移行。行の先頭から入力開始
A	挿入モードへ移行。行の終わりから入力開始
o	挿入モードへ移行。カーソルの次の行の先頭から入力開始
O	挿入モードへ移行。カーソルの前の行の先頭から入力開始
dd	その行を削除
d数字d	その行から指定した行数だけ行を削除
yy	その行をコピー（Vim内部でのコピー）
y数字y	その行から指定した行数だけ行をコピー（Vim内部でのコピー）
p	次の行にコピーした内容をペースト。vまたはCtrl-vで選択された領域がコピーされたものである場合にはカーソルの次の場所からペースト
P	前の行にコピーした内容をペースト。vまたはCtrl-vで選択された領域がコピーされたものである場合にはカーソルの場所からペースト
x	カーソルの位置の文字を削除
X	カーソルの1つ前の文字を削除
D	カーソルから行の終わりまでの文字を削除
v	ビジュアルモードへ移行。文字単位で選択
V	ビジュアルモードへ移行。行単位で選択
zz	カーソルがある行がターミナルの上下で見た場合の中央に来るようにページを移動
Ctrl-v	ビジュアルモードへ移行。矩形で選択
数字コマンド	上記コマンドなどを指定した数字の回数だけ繰り返す。10xなら10文字削除。100→なら100回→キーを押したのと同じ意味

▼表4　:でコマンドラインモードになった場合の操作

文字	意味
番号	指定した行番号へ移動
w	ファイルへ書き込み
q	Vimの終了
wq	ファイルへ書きこんでからVimを終了
wq!	書き込みが許可されていないファイルに書き込みを実施したのちVimを終了
undo	アンドゥ
redo	リドゥ
s/パターン/置換後文字列/	カーソルのある行でパターンに一致した最初の文字列を置換
s/パターン/置換後文字列/g	カーソルのある行でパターンに一致したすべての文字列を置換
%s/パターン/置換後文字列/	s/パターン/置換後文字列/を全行に対して適用
%s/パターン/置換後文字列/g	s/パターン/置換後文字列/gを全行に対して適用
'<,'>s/パターン/置換後文字列/	s/パターン/置換後文字列/をビジュアルモードで選択した行に対して適用。ビジュアルモードにおいて選択した状態で:を押すとこの先頭の5文字は自動的に表示される
'<,'>s/パターン/置換後文字列/g	s/パターン/置換後文字列/gをビジュアルモードで選択した行に対して適用。ビジュアルモードにおいて選択した状態で:を押すとこの先頭の5文字は自動的に表示される
set number	行番号を行頭に表示する
set nonumber	行番号を行頭に表示しない
set wrap	ターミナルの幅を超える文字列を回りこんで表示する
set nowrap	ターミナルの幅を超えた文字列を回りこんで表示しない
D	カーソルから行の終わりまでの文字を削除
v	ビジュアルモードへ移行。文字単位で選択
V	ビジュアルモードへ移行。行単位で選択
zz	カーソルがある行がターミナルの上下で見た場合の中央に来るようにページを移動
Ctrl-v	ビジュアルモードへ移行。矩形で選択
数字コマンド	上記コマンドなどを指定した数字の回数だけ繰り返す。10xなら10文字削除。100→なら100回→キーを押したのと同じ意味

COLUMN 1

Vimで快適執筆環境

Writer 結城 浩(ゆうき ひろし) Twitter：@hyuki

ThinkPadからMacBookへ

この夏は、私の執筆環境がThinkPadからMacBook Airへ移るという大変化が起きました。以下、どんなふうにして執筆環境が移ったかをお話しします。

2013年の6月から7月にかけて、愛用のThinkPadのSSDが2回も不調になりました。保証期間内で無償修理なのはいいのですが、その間も〆切は待ってくれません。しかたがないので、ThinkPadが戻ってくるまでの間、MacBook Airで原稿を書くようになりました。そのときはまさかMacがメインの執筆環境になるとは思っていなかったのですが。

これまで私は何回かMacに執筆環境を移行しようと思っていましたが、その障害となるのはいつも**キーボード**と**テキストエディタ**でした。

キーボードは慣れで解決

まずはキーボードです。MacBook Airのキーボードはキーの縁でどうも指先が引っかかる。思わず「面取り」をしたくなるほどです。しかしThinkPadが故障している間は、すべての文章をMacBook Airのキーボードで書かなければなりません。原稿を何本か書くうちに指先の引っかかりがなくなってきました。**正確なキーの位置を指が覚え始めた**のです。

これまでは、いつもそばにThinkPadがありました。そのため、Macを使って少し不満があると「やっぱりThinkPadがいいや」と戻っていました。今回のように強制的に書き続けていればちゃんと慣れるものだったのですね。

日本語モードはKeyRemap4MacBookで解決

キーボードでは日本語モードの切り換えも問題でした。Windowsでの Alt ＋ 漢字 キーという手の動きが、Macに移っても直らないのです。かな キーと 英数 キーを使えばいいのですが、頭がどうしても切り換わりません。

そこで、**システム環境設定のキーボードショートカットで「前の入力ソースの選択」を ⌘ ＋ 1 に割り当てました。さらに、KeyRemap4MacBookを使って 英数 キーを ⌘ に割り当て**ました。これによって、Windowsのときと似た手の動きで日本語モードの切り換えが可能になりました。

秀丸エディタからVimへ

Macは秀丸エディタがない。執筆環境を移行できない大きな理由がこれでした。私にとって「手になじむエディタ」というのはまさに秀丸エディタのこと。Macにも良いエディタはたくさんあるのですが、帯に短し襷に長し。20数年前、UNIXをメインの環境にしていたときのエディタはviでした。viも当時の私にとって「手になじんだエディタ」でしたが、もっぱらコードを書くためのものであり、viで日本語入力をしたことはありませんでした。viのモード切り換えと、日本語入力のモード切り換えが干渉しあってひどい目に遭うからです。

しかし、**KeyRemap4MacBookを使って Esc を Esc ＋ 英数 に割り当てる**とモード切り換えの干渉が激減しました。**入力モードからノーマルモードに戻ったときに日本語入力が自動的にオフ**

になるからです。それでもノーマルモードで文字を入力してしまうという勘違いがときどき起きました。そこで、**入力モードに入ったときにステータス行の色を変える**という改善を行いました。

キーボードの問題とモードの問題が解決して、私はMacBook AirとVimに自分の執筆環境を移行しようと本気で考え始めました。

Emacs風の入力モードで快適に

Vimを使い始めて驚いたのは**入力モードで自由にカーソルが動ける**という点です。古(いにしえ)のviではそんなことはできませんでしたから。そこでVimの**入力モードのカーソル移動をEmacs風に調整**しました。これで、ノーマルモードではVimの気分で動きまわり、入力モードではEmacsの気分で書く感覚になりました。

ここまで来ると、モードの干渉は気にならなくなり、それどころか**Vimのモードが文章書きにしっくりくる**ようにさえ感じられてきました。つまり、Vimのノーマルモードと入力モードが、文章を眺めているモードと書いているモードにそれぞれ対応するということです。

フォントを変えて快適にMacに

執筆環境を移行してフォントの美しさに感動しました。日本語には**ヒラギノ明朝ProN**を使い、ラテン文字には心が洗われるフォント**Inconsolata**を使っています。

Vimで三点リーダー(…)、星マーク(★)、矢印(→)の表示がおかしい現象がありました。文字の一部が欠けたり、幅が狭く表示されたりするのです。これは:set ambiwidth=doubleという設定で改善しました。

カラーリングを調整して快適に

Vimでうれしいのは**シンタックスカラーリング**の充実です。自分が書くものはLaTeX、Perl、Ruby、Java、HTML、CSSくらいですが、すべて自動判断でカラーリングされました。またLaTeXの太字指定コマンドである\textbf{**テキスト**}を使うと、テキストがちゃんと太字になってくれるのもうれしいですね。

構文ファイルを見よう見まねで作成し、**自分が書くテキストファイルもシンタクスカラーリングする**ようになりました。細かい色合いは、Vimの側で調整するよりも、**ターミナルソフトiTerm2の環境設定で色割り当てを調整**したほうが楽なのでそうしています。

うれしい誤算の夏

振り返ってみますと「キーボードがいやでエディタがないからMacに移行できない」という私の思い込みは「強制的にVimで原稿を書く」という経験で消えました。ThinkPadの故障のおかげでMacBook Airへ執筆環境が完全に移行し、現在もこの原稿はMacBook AirとVimで書いています。さらに、以前から願っていたテキストのUTF-8化も進みました。災い転じて福となすとはこのことです。

MacBook AirでVimを使って文章を書いていると、タイムスリップしたような不思議な感覚に陥ることがあります。20数年前の古巣に戻ってきた感覚とでもいうのでしょうか。

執筆環境を整えるのは楽しい作業です。今回で書いたVimの改善のほとんどは、検索したブログや、Twitter経由で教えていただいたものでした。皆さんに感謝します。SD

● 参考Webページ

- ターミナル上CUIのvimでノーマルモードに戻ったときに日本語入力モードを自動的にオフにする方法(http://hyukimac.tumblr.com/post/55089242744/cui-vim)
- 挿入モードでステータスラインの色を変更する(https://sites.google.com/site/fudist/Home/vim-nihongo-ban/vim-color#color-insertmode)
- Vimで入力モード中に簡単なEmacs風カーソル移動と編集をする設定(http://tmblr.co/ZpJMirpNsS9N)
- Inconsolataフォント(http://levien.com/type/myfonts/inconsolata.html)

第1特集 Vim至上主義

第2章 押さえるべき基本技

Writer 後藤 大地(ごとう だいち) BSDコンサルティング㈱ Twitter：@daichigoto、@BSDc_tweet

Vimでファイルを編集している間、別のターミナルアプリケーションからシステムにログインして、インタラクティブシェルでコマンドを実行することがあります。編集中のファイルに挿入したいデータを作成したり、コマンドを実行して動作を確認したり、Webサーバを再起動したりなど、目的はさまざまですが、Vimとシェルは切っても切れない関係にあるといえます。本章ではこうしたシェルとVimの活用方法について解説するとともに、プラグインの例としてvim-fugitiveを紹介します。

1 :shellでインタラクティブシェルを使う

Vimはシェルを起動する機能を提供しています。コマンドモード(Vimではノーマルモードと呼んでいます)で「:shell」と入力してみてください(**図1、2**)。Vimからシェルに制御が切り替わります。

```
:shell
```

シェルを終了するとVimに戻ってきます。一度Vimを終了してから何かの操作をして、再度Vimでファイルを開いて編集するという手間に比べると、若干操作が速く楽になります。ファイルを開くためにコマンドを入力する必要が省けますし、カーソルの位置も記憶されていて便利です。

実際に何が起こっているのか調べてみましょう。**図3**のようにps(1)コマンドを実行すると、Vimから新たにシェルが起動されていることがわかります。起動されているシェルはユーザがログインシェルに指定しているシェルです[注1]。

Vimでシェルを操作する方法はおもに次の3つの方法があります。

- :shellのようにシステムのシェルを起動する
- シェルの動きを模倣したプラグインを使用する
- システムのシェルと通信するプラグインを使用する

どの方法にも利点と欠点があります。たとえば:shellを使う場合の欠点は、:shellで起動したシェルとVimの間でテキストデータをやりとりする便利な方法がありません。シェルの動きを模倣するプラグインはVimとの相性が良いのですが、普段使っているシェルと違う動きをするため、操作にストレスを感じることがあります。システムのシェルと通信するプラグインを使用する場合にも同様な懸念が残ります。

普段使っているインタラクティブシェルを変更するのはストレスを伴うことですので、ここではインタラクティブシェルとVimの間で簡単にデータを共有するしくみをちょっとした工夫でカバーする方法を紹介します。

こうした問題を解決する場合、システムクリップボードにテキストデータをコピーすることで共有するという方法がありますが、ssh経由でログインしている場合にはそのアプローチが使えません。かわりに、データをファイルに出力してそのファイルを経由してデータのやりとりをすることにします。

まず、シェルで使うコマンドを2つ作ります。bfとbfcatです。bfがバッファファイルにデータを書き込むコマンド、bfcatがバッファファイルからデータを取り出して表示するコマンドだと思ってください。**図4～7**のような感じで作れば良いでしょう。

次に、Vimの設定ファイル~/.vimrcに次の設定

注1) 「-d」はプロセスの親子関係をツリー状に表示するオプション、「-t 5」は制御端末/dev/pts5に関連付けられているプロセスのみを表示せよというオプションです。FreeBSDで動作します。

を追加します。これはコマンドモードまたは挿入モードで Ctrl-B キーを押すと、先ほどのバッファファイルの中身を読み込んで貼り付けるというものです。逆に、ビジュアルモードで選択した状態で Ctrl-B を押すと、先ほどのバッファファイルに選択した内容を書き出すように振る舞います。これでVimとシェルの間で比較的簡単にテキストデータのやりとりができるようになります。

```
imap <C-b> <ESC>:read ~/.vim/bf<CR>i
nmap <C-b> :read ~/.vim/bf<CR>
vmap <C-b> :w!~/.vim/bf<CR>
```

このしくみは簡単ですが、使ってみるとけっこう重宝します。

擬似端末アプリケーション（tmuxやscreen）を使う

:shellの場合の問題点は、編集内容を見ながらのコマンド実行がやりにくい点にあります。ときには同時に見ながら作業をしたいということがあります。こういったケースではtmuxやscreenといった擬似端末アプリケーションを使うのが得策です。とくにssh経由でログインして作業するようなケースでは、回線状況で接続が切れたりすることがありますので、tmuxやscreenなどの擬似端末経由にしておいたほうが何かと安全です。

たとえば**図8**、**9**のように使うことができます。ファイルを編集しながら、擬似端末アプリが提供しているショートカットキーを押して端末を2つに増や

▼図1　Vimでファイルを編集しているところ

```
ssh — ttys000
 1 # Copyright (c) 2009 ONGS Inc.
 2 # All rights reserved.
 3 #
 4 # This software may be used, modified, copied, and distributed, in
 5 # both source and binary form provided that the above copyright and
 6 # these terms are retained. Under no circumstances is the author
 7 # responsible for the proper functioning of this software, nor does
 8 # the author assume any responsibility for damages incurred with its
 9 # use.
10
11 # author: Daichi GOTO (daichi@ongs.co.jp)
12 # first edition: Wed Sep 16 12:01:59 BST 2009
13 # last modified: $Date: 2013-07-21 22:14:43 +0900 (2013/07/21 (日)) $
14 # version: $Revision: 34532 $
15
16 .include "../../mk/articles.base.mk"
~
/z/localdocs/SoftwareDesign/201310/vim-02/Makefile
```

▼図2　:shellでシェルを起動

```
ssh — ttys000
 5 # both source and binary form provided that the above copyright and
 6 # these terms are retained. Under no circumstances is the author
 7 # responsible for the proper functioning of this software, nor does
 8 # the author assume any responsibility for damages incurred with its
 9 # use.
10
11 # author: Daichi GOTO (daichi@ongs.co.jp)
12 # first edition: Wed Sep 16 12:01:59 BST 2009
13 # last modified: $Date: 2013-07-21 22:14:43 +0900 (2013/07/21 (日)) $
14 # version: $Revision: 34532 $
15
16 .include "../../mk/articles.base.mk"
~
/z/localdocs/SoftwareDesign/201310/vim-02/Makefile
:shell
PARANCELL ...docs/SoftwareDesign/201310/vim-02% ls
Makefile        csvs            pdfs            typescript.xml
commands        images          sources
PARANCELL ...docs/SoftwareDesign/201310/vim-02% exit
```

※どのように表示されるかは利用しているOSやターミナルアプリケーションごとに異なる

▼図3　ps(1)コマンドを実行してみる

```
% ps -d -t 5
  PID TT  STAT    TIME COMMAND
48624  5  Is   0:00.37 -zsh (zsh)
48901  5  I    0:00.25 - vim Makefile
48939  5  S    0:00.19 `-- /usr/local/bin/zsh
49012  5  R+   0:00.00   `-- ps -d -t 5
```

▼図4　bfコマンド — エイリアス版

```
alias bf="cat > ~/.vim/bf"
```

▼図5　bfcatコマンド — エイリアス版

```
alias bfcat="cat ~/.vim/bf"
```

▼図6　bfコマンド — シェルスクリプト版

```
#!/bin/sh
cat > ~/.vim/bf
```

▼図7　bfcat — シェルスクリプト版

```
#!/bin/sh
cat ~/.vim/bf
```

し、縦方向に分割表示させています。topコマンドを実行しながら編集という作業をしています。

シェルとVimの間のデータ共有は、先ほど作ったbf、bfcat、Ctrl-Bのやり方で実現できます(図10〜12)。

▼図8　tmux上でVimを実行しているところ

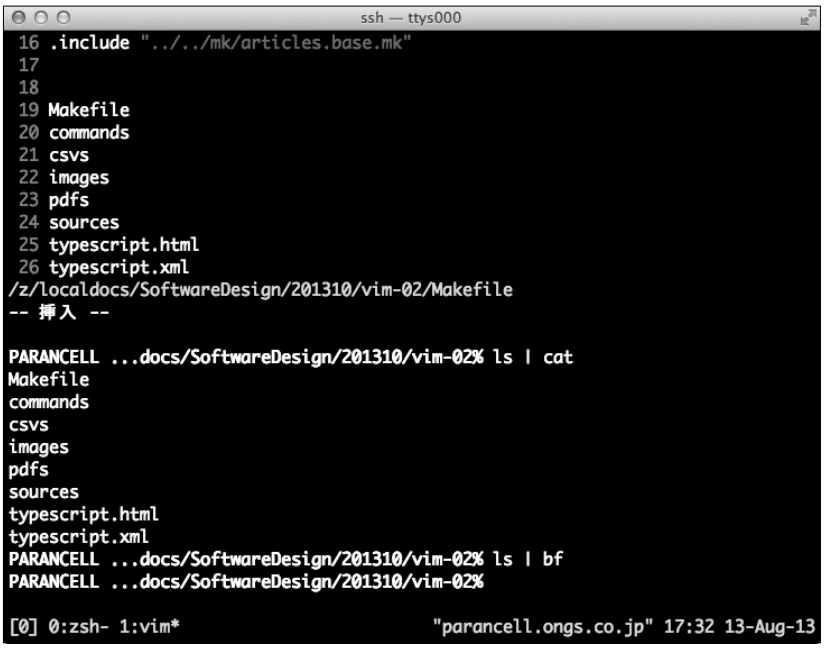
```
ssh — ttys000
  1 # Copyright (c) 2009 ONGS Inc.
  2 # All rights reserved.
  3 #
  4 # This software may be used, modified, copied, and distributed, in
  5 # both source and binary form provided that the above copyright and
  6 # these terms are retained. Under no circumstances is the author
  7 # responsible for the proper functioning of this software, nor does
  8 # the author assume any responsibility for damages incurred with its
  9 # use.
 10
 11 # author: Daichi GOTO (daichi@ongs.co.jp)
 12 # first edition: Wed Sep 16 12:01:59 BST 2009
 13 # last modified: $Date: 2013-07-21 22:14:43 +0900 (2013/07/21 (日)) $
 14 # version: $Revision: 34532 $
 15
 16 .include "../../mk/articles.base.mk"
~
~
~
~
~
~
~
~
/z/localdocs/SoftwareDesign/201310/vim-02/Makefile

[0] 0:zsh- 1:vim*                          "parancell.ongs.co.jp" 17:30 13-Aug-13
```

▼図9　tmuxの機能で擬似端末を2つに増やし、シェルでtopコマンドを実行

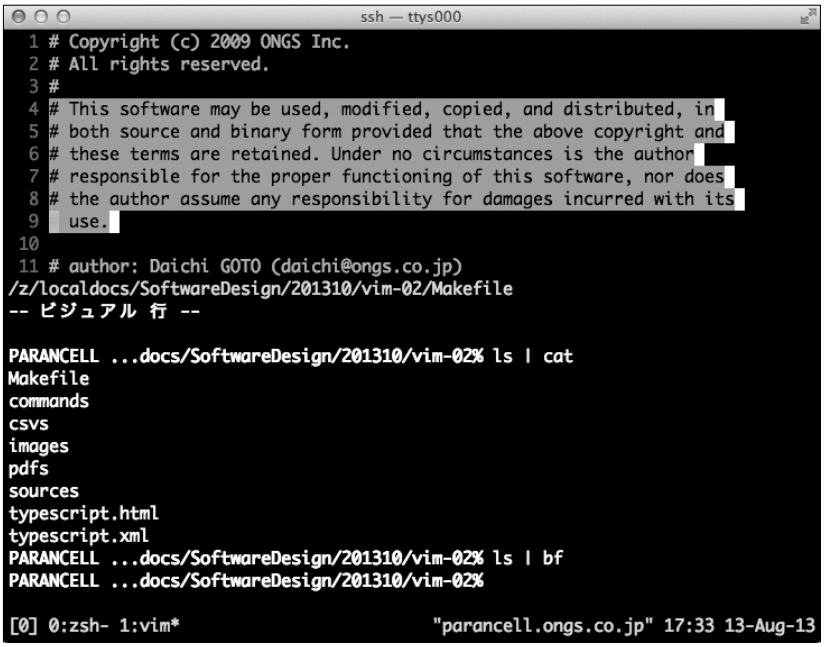
```
ssh — ttys000
  1 # Copyright (c) 2009 ONGS Inc.
  2 # All rights reserved.
  3 #
  4 # This software may be used, modified, copied, and distributed, in
  5 # both source and binary form provided that the above copyright and
  6 # these terms are retained. Under no circumstances is the author
  7 # responsible for the proper functioning of this software, nor does
  8 # the author assume any responsibility for damages incurred with its
  9 # use.
 10
 11 # author: Daichi GOTO (daichi@ongs.co.jp)
 12 # first edition: Wed Sep 16 12:01:59 BST 2009
 13 # last modified: $Date: 2013-07-21 22:14:43 +0900 (2013/07/21 (日)) $

last pid: 51869;  load averages:  0.26,  0.24,  0.17    up 2+02:33:02  17:30:51
79 processes:  1 running, 78 sleeping
CPU:  2.4% user,  0.0% nice,  0.2% system,  0.3% interrupt, 97.2% idle
Mem: 895M Active, 2048M Inact, 11G Wired, 1192K Cache, 1642M Buf, 1368M Free
ARC: 4916M Total, 353M MFU, 1954M MRU, 16K Anon, 171M Header, 2439M Other
Swap: 4096M Total, 4096M Free

  PID USERNAME    THR PRI NICE   SIZE    RES STATE   C   TIME   WCPU COMMAND
 8557 daichi        6  20    0   547M   143M pcmwrv  3 207:48  7.76% vlc
 1087 daichi       44  21    0  1934M  1542M uwait   4 125:46  1.76% firefox
 1046 daichi        2  23    0   284M 96080K kqread  3  29:07  0.29% compiz
51869 daichi        1  20    0 16600K  2568K CPU5    5   0:00  0.20% top
[0] 0:zsh- 1:top*                          "parancell.ongs.co.jp" 17:30 13-Aug-13
```

擬似端末アプリケーションを使う方法は、たくさんのターミナルアプリケーションを同時に起動することが困難な環境(たとえばノートPCのように画面サイズが限られた環境)でとくに有益です。ディスプレイサイズが小さくても、多くのターミナルアプリケーションを起動して利用

▼図10　ls | bfを実行して、Vimでは Ctrl-B で貼り付け



▼図11　Vimでビジュアルモードで範囲選択して Ctrl-B



することはできますが、ターミナルアプリケーション間の移動がネックになって作業効率が低下しがちです。

▼図12　シェルでbfcatでその内容を出力

▼図13　Vimから:!LANG=C dateと入力

▼図14　LANG=C dateの実行結果が表示される

Windowsでは仮想化技術またはVimShellプラグイン

現在はVirtualBoxなど優れた仮想化技術を無償で活用できますので、Windowsを使っている場合には仮想環境にUbuntuやPC-BSDをインストールして使う方法が手軽で便利です。ファイルの共有が必要な場合にはゲストOS側でSambaをセットアップして相互利用できるようにするくらいでことが済みます。

WindowsそのものでVimを使いたいという場合には、VimShellというプラグインを活用するという方法もあります。実に優れたプラグインで、とくにWindowsのようにインタラクティブシェルの機能が制限されている場合に代替のシェルとして活用できます。

:!コマンドと:r!コマンド

コマンドを1回だけ実行できれば良いという場合には、:!を使うという方法もあります。「:!コマンド」とするとそのコマンドが実行されます(**図13、14**)。psでプロセスの親子関係を表示させると、Vimがそのコマンドを実行していることがわかります。

コマンドの実行結果を取り込みたい場合には:r!を使います。「:r!コマンド」でそのコマンドの実行結果がVimのカーソルの位置へ貼り付きます(**図15、16**)。

:!は1回だけコマンドを実行したい場合に便利で、たとえばsvn commitのようにコミット処理を行いたいといった場合に使えます。一度Vimを抜けてしまうと処理の流れが途切れてしまいますが、Vimからコマンドラインモードで実行できるしくみになっていると作業の分断感覚が少なくて済みます。

ターミナルアプリケーションを2つ起動しておいてもう片方でsvn commitを実行するといったこともできますが、キーボードから手を離してからマウスやトラックパッドを操作するというのは、それだけで時間のロスになります。このやり方を身につけておくと、そうしたロスを減らすことができます。

ちょっとした応用編 – vim-fugitiveプラグインを使ってみよう

GitHubの人気もあって、ソフトウェア開発にGitHubを採用しているという開発者は少なくありません。そうなると、プログラミングしなが

▼図15　Vimから:r!LANG=C dateと入力

```
ssh — ttys000
 4 # This software may be used, modified, copied, and distributed, in
 5 # both source and binary form provided that the above copyright and
 6 # these terms are retained. Under no circumstances is the author
 7 # responsible for the proper functioning of this software, nor does
 8 # the author assume any responsibility for damages incurred with its
 9 # use.
10
11 # author: Daichi GOTO (daichi@ongs.co.jp)
12 # first edition: Wed Sep 16 12:01:59 BST 2009
13 # last modified: $Date: 2013-07-21 22:14:43 +0900 (2013/07/21 (日)) $
14 # version: $Revision: 34532 $
15
16 .include "../../mk/articles.base.mk"
17
18
/z/localdocs/SoftwareDesign/201310/vim-02/Makefile
:r!LANG=C date
```

▼図16　実行結果が挿入される

```
ssh — ttys000
 5 # both source and binary form provided that the above copyright and
 6 # these terms are retained. Under no circumstances is the author
 7 # responsible for the proper functioning of this software, nor does
 8 # the author assume any responsibility for damages incurred with its
 9 # use.
10
11 # author: Daichi GOTO (daichi@ongs.co.jp)
12 # first edition: Wed Sep 16 12:01:59 BST 2009
13 # last modified: $Date: 2013-07-21 22:14:43 +0900 (2013/07/21 (日)) $
14 # version: $Revision: 34532 $
15
16 .include "../../mk/articles.base.mk"
17
18
19 Tue Aug 13 17:43:05 JST 2013
/z/localdocs/SoftwareDesign/201310/vim-02/Makefile
```

らgitコマンドを使う必要があります。:!ですべて補っても良いのですが、Vimにはこうした場合に使える便利なプラグインがありますので使ってみましょう。ここではプラグインとして「vim-fugitive」を取り上げます。gitと連携する場合の代表的なプラグインの1つです。

インストールは**図17**のように実施します。なお、プラグインの管理に便利なので、最初にvundleという別のプラグインをインストールしています。ここではこういうものがあるくらいに思っておいていただければ大丈夫です。

~/.vimrcには次の設定を追加します。これでvim-fugitiveが使えるようになります。

```
filetype off
set rtp+=~/.vim/bundle/vundle/
call vundle#rc()
Bundle 'tpope/vim-fugitive'
filetype plugin indent on
```

vim-fugitiveをインストールすると**表1**のようなユーザコマンドが利用できるようになります。gitコマンドに対応したような作りになっています(**図18、19**)。

bf、bfcatに対応するショートカットキーを設定したときのように、よく利用する処理にはショートカットキーを設定したり、ちょっとした工夫をすることでもっと便利に使えるようになります。設定ファイルの書き方はVimの配布物を展開して、中に入っている拡張子が.vimのファイルを読んでみるとなんとなくわかるようになります[注2]。SD

注2) FreeBSDであれば/usr/local/share/vim/vim73/以下にまとまっているので、このあたりを読んでいくと便利です。

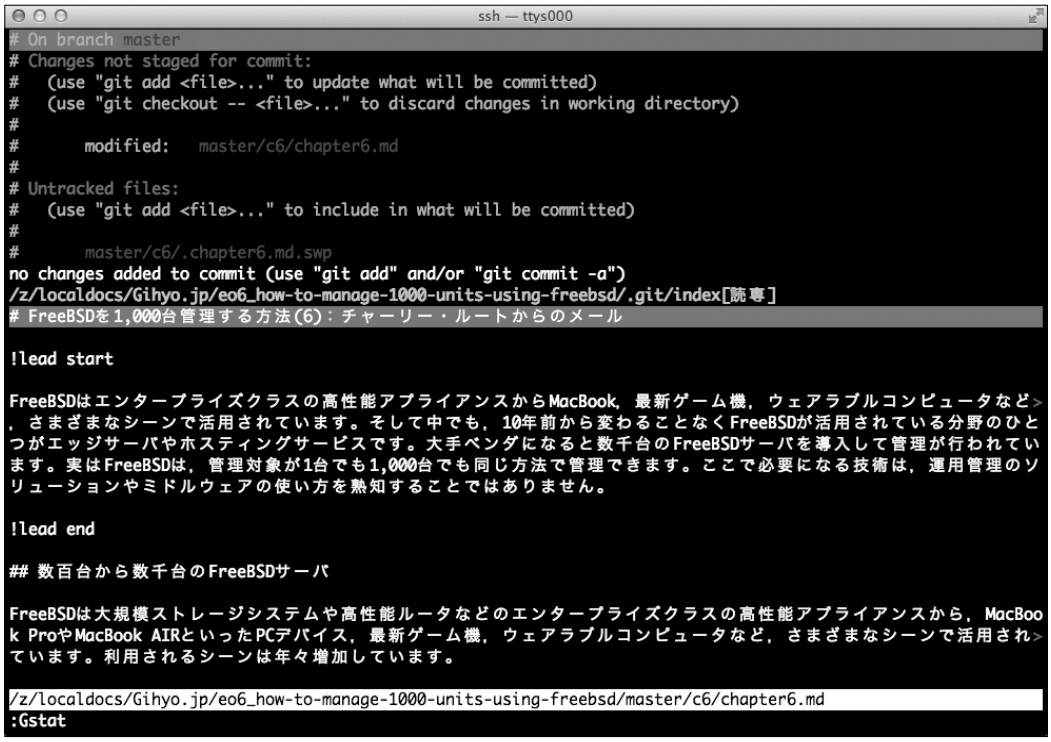
▼表1 vim-fugitiveで使えるコマンド一覧

コマンド	意味
:Git	gitに相当。git(1)コマンドを任意に実行できる
:Gedit	リポジトリ内データの閲覧
:Gsplit	リポジトリ内データの閲覧。左右スプリットスタイル
:Gvsplit	リポジトリ内データの閲覧。上下スプリットスタイル
:Gtabedit	リポジトリ内データの閲覧。タブスタイル
:Gdiff	差分表示
:Gstatus	git statusに相当
:Gcommit	git commitに相当
:Gblame	git blameに相当
:Gmove	git mvに相当
:Gremove	git rmに相当
:Ggrep	git grepに相当
:Glog	履歴情報を一覧表示
:Gread	git checkout -- filenameに相当
:Gwrite	git addに相当

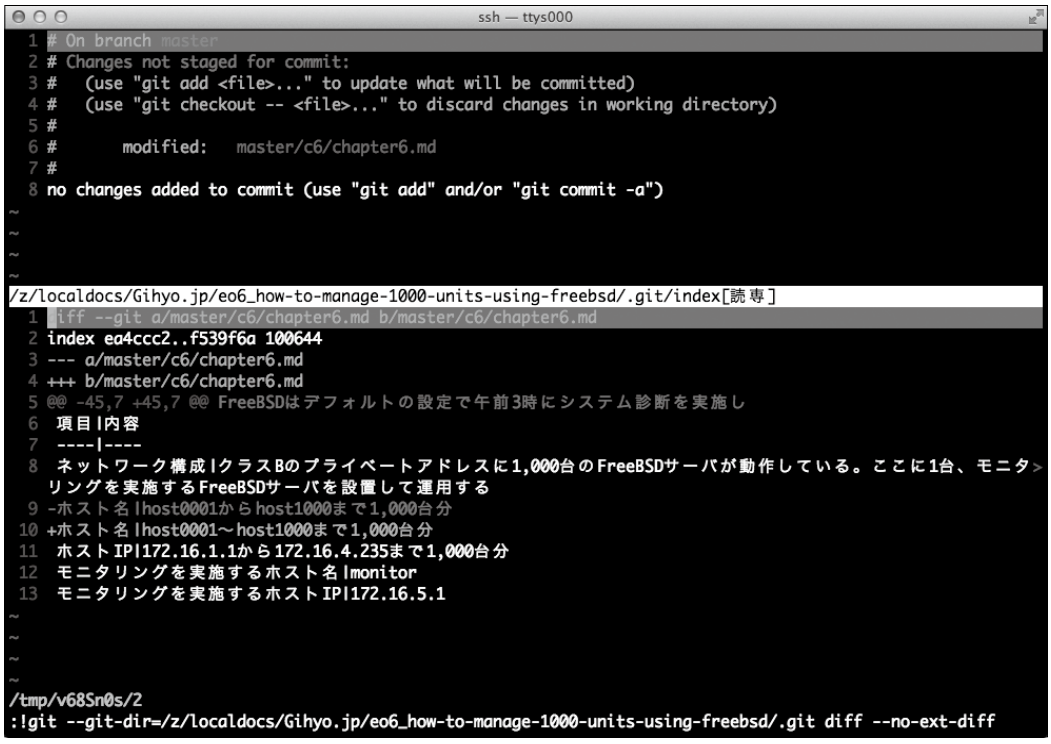
▼図17 vim-fugitiveのインストール

```
% git clone https://github.com/gmarik/vundle.git ~/.vim/bundle/vundle
Cloning into '/z/daichi/.vim/vundle.git'...
remote: Counting objects: 2461, done.
remote: Compressing objects: 100% (1588/1588), done.
remote: Total 2461 (delta 834), reused 2389 (delta 779)
Receiving objects: 100% (2461/2461), 297.06 KiB | 145.00 KiB/s, done.
Resolving deltas: 100% (834/834), done.
Checking connectivity... done
% cd ~/.vim/bundle/
% git clone git://github.com/tpope/vim-fugitive.git
Cloning into 'vim-fugitive'...
remote: Counting objects: 1554, done.
remote: Compressing objects: 100% (764/764), done.
remote: Total 1554 (delta 518), reused 1416 (delta 397)
Receiving objects: 100% (1554/1554), 215.08 KiB | 180.00 KiB/s, done.
Resolving deltas: 100% (518/518), done.
Checking connectivity... done
%
```

▼図18　Gstatを実行したところ



▼図19　Gstate実行中にDキーを押して差分を表示させたところ



COLUMN 2

Vi〆をお勧めする理由

Writer 中井 悦司(なかい えつじ)Twitter@enakai00

viとの出会いはrootユーザから

筆者と「vi」の出会いは、1990年代に遡ります。そのころは、大学で物理学の勉強をしていたのですが、複雑な数式まじりの文書は、UNIX上のLaTeX(ラテック)で作成するのが慣例でした。たとえば、**リスト1**のプレインテキストを「LaTeXコンパイラ」にかけると**図1**の数式ができあがります。

そういえば、筆者のLaTeXを使った「作品」の1つがインターネットで公開されているようです[注1]。複雑な数式がきれいに描かれていることがわかるでしょう(論文の内容は気にしないでくださいね(笑))。

このように、論文作成用の「文房具」感覚でUNIXワークステーションを使っていたわけですが、使い始めた当初は、viエディタの存在を知らずに、もっぱらEmacsを利用していました。viエディタをはじめて見たのは、ワークステーションの管理者がシステムの設定変更作業を行う場面に出会った時です。その人は、「root作業は、viだよね〜」と謎の言葉をつぶやきながら、テキストモードのシステムコンソールに向かって設定ファイルの編集を行っていました。

その時、天啓がひらめいたのです。「よくわからないが、これはマスターしなければならない！」 さっそく書店に向かい、購入したのがオライリーの『vi入門』です[注2]。当時、この本の「訳者まえがき」には、次のような言葉がありました。

> しかし、人間もエディタも、見かけや第一印象で判断してはいけない。最初は愛想のないやつだと思っても、つきあってみるとシャイなだけで本当は実にいい人だったりするケースがあるだろう。viがまさにそれだ。

最近、Gitに関する書籍[注3]の前書きで、似たような表現を見た記憶がありますが、流行の言葉で言うと「ツンデレ」というやつです。極限的にシンプルなインターフェースとコマンドの組み合わせによる強力な編集機能、そして、外部

▼リスト1　質量とエネルギーの等価性を示す公式

```
E = mc^2 ¥¥
m = ¥frac{m_0}{¥sqrt{1-¥frac{v^2}{c^2}}}
```

▼図1　「リスト1」から生成される数式

$$E = mc^2$$

$$m = \frac{m_0}{\sqrt{1 - \frac{v^2}{c^2}}}$$

注1) 『古典特異系における第二次拘束条件の取扱いについて』素粒子論研究 89-4(1994)http://ci.nii.ac.jp/naid/110006412355

注2) 『入門vi 第6版』リンダ ラム、アーノルド ロビンス(著)、福崎俊博(翻訳)オライリー・ジャパン、2002年

注3) 『Gitポケットリファレンス』岡本隆史、武田健太郎、相良幸範(著)技術評論社、2012年

コマンドとの連携など、またたく間に手放せないツールになりました。キーマッピングを利用して、編集中のLaTeXのテキストファイルをviエディタからコンパイルできるように作りこんだことを覚えています。

今から考えると、これが、「シンプルなツールの組み合わせで複雑な事を実現する」というUNIX/Linuxの思想を体感した初めての経験だったのかもしれません。

素顔こそがviの魅力

先ほどの「root作業は、viだよね～」という発言にはどんな意味があったのでしょうか？20年も前の出来事で、もはや正確なことはわかりませんが、必要な初期設定がされておらず、rootユーザではEmacsが起動できなかったというように記憶しています。

数年前に、Linuxシステム管理の研修コースを担当したことがあるのですが、その際に、受講生の1人から「どうしてEmacsを使わないのですか？」と素朴な質問を受けました。その時、脳裏に浮かんだのが先の経験で、「もしかしたら、Emacsが使えないサーバがあるかもしれないよね。お客様先のシステムだったら、勝手にEmacsをインストールするわけにはいかないから、確実に入っているviを使うのがいいんだよ」と、とっさに返答しました。

「どこでも確実に使える」というポータビリティは、確かに、UNIX/Linuxシステム管理者にviの利用をお勧めする大きな理由の1つです。その意味では、まったくカスタマイズしていない、「素の状態」であっても十分に強力な編集機能を利用できることもviエディタの価値と言えるでしょう。あらゆる環境、あらゆる状況のシステムを的確に取り扱うことがシステム管理者の腕の見せ所ですから、まずは、「素の状態」のviエディタを理解して使いこなすことが大切です。複雑な設定を駆使して独自のvi環境を作りこむことも、viの楽しみ方の1つですが、まずは、viの「素顔の魅力」を知ってあげてください。

「Vim」いうな?!

そういえば、最近、一部エンジニアの間では、「vi」ではなく、あえて「Vim」と呼ぶことが流行っているようです。viは、もともとUNIX環境で利用されていたツールですが、Bram Moolenaarが「Amigaコンピュータ」で使えるviクローンとして開発したのが「Vim（Vi IMproved）」です。その後、Vimはさまざまなプラットフォームに移植されて、現在、Linuxで動いているviの大部分はこのVimです。

そういういう意味では、「vi」ではなく「Vim」と呼ぶのは間違いではないのですが、UNIX時代からviを使い始めた身からすると、少しばかりの違和感を感じることもあります。企業システムにかかわる仕事をしていると、UNIXを扱うこともまだまだあります。viにはない、Vimの独自機能を普段から使っていると、UNIXでviエディタを起動した際に、思うように操作できなくて歯がゆくなることがあります。

Vimを愛するみなさんも、一度、「Visual Mode」などのVim固有機能を使わずに、古き良きviの編集機能を再確認してください。極限までシンプルでしかも強力――そんな、UNIX/Linux文化の源泉をあらためて感じられるかもしれません。

ちなみに、8月末に筆者の新著『「独習Linux専科」サーバ構築/運用/管理――あなたに伝えたい技と知恵と鉄則』が技術評論社から出版されました。Linux初心者を対象に、Linuxの魅力を理解して、Linuxと深く・長く付き合うために必要な「技・知恵・鉄則」を筆者の経験に基づいて書き下ろしました（viエディタの使い方ももちろん解説しています！）。Linuxの基礎を体系的に勉強しなおしたい方にもお勧めです。SD

第3章 Vimプラグインの導入

Writer daisuzu daisuzu@gmail.com http://daisuzu.hatenablog.com/

Vimは単体でも豊富な機能を持つエディタですが、プラグインを導入することでさらに機能を拡張することができます。本誌2012年7月号の第1特集の第3章“Vimプラグイン108選”でもVimプラグインを紹介しましたが、今回はプラグインの導入方法および以前紹介したプラグインの中でもとくに拡張性の高いプラグインについて紹介します。

1 プラグインの導入（neobundle.vim）

プラグインを導入するにあたり、最も基本的な方法はホームディレクトリに“.vim”[注1]を作成し、プラグインの種別に合わせてサブフォルダ（**図1**）に<プラグイン名.vim>をコピーする方法です。http://www.vim.org/からダウンロード可能なプラグインは、おおむね**図1**の形式でアーカイブされているため、“~/.vim”にダウンロードしてきたプラグインのアーカイブを展開するだけでインストールは完了です。

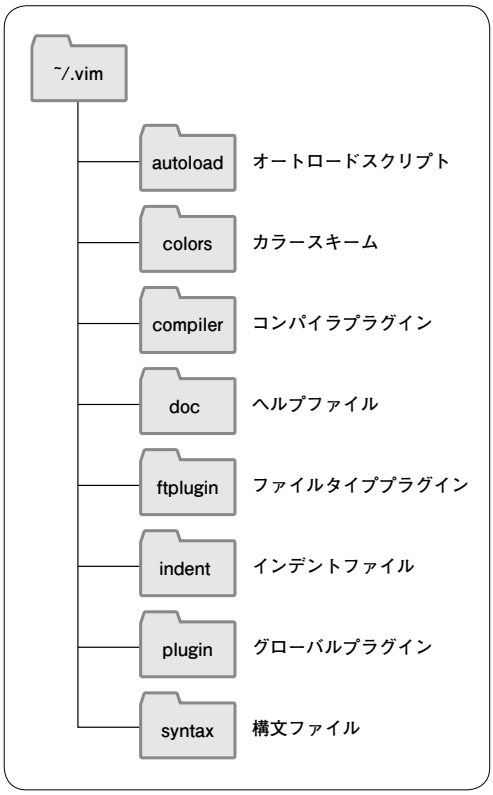
しかし、上記の方法でインストールしていくと、プラグインの数が増えるにつれて管理が煩雑になってしまいます。たとえば不要になったプラグインを削除しようとした際、どのファイルを削除すれば良いのかわからなくなってしまいます。また、新たにインストールするプラグインの中にインストール済みのプラグインと同名のファイルが含まれていた場合、古いプラグインは上書きされてしまうかもしれません。

これらの問題を解決するにはプラグインごとに専用のフォルダを作成するという方法があります。Vimにはプラグインをどこから読み込むのかを設定する機能があり、この機能を使用したプラグインとして2008年頃にpathogen.vim[注2]がリリースされました。ただし、pathogen.vimにはプラグインの自動インストールやアップデートといった機能はなく、ユーザ

注1） Windowsの場合は“vimfiles”。

注2） https://github.com/tpope/vim-pathogen

▼図1 プラグインのインストール先



が個別にインストールやアップデートを行う必要がありました。

その後、2010年頃に自動インストールやアップデートに対応したVundle[注3]がリリースされました。現在のように、多くのVimプラグインが作成されるようになったのはこのころからだったのではないかと思います。

そして現在、最も機能が豊富なプラグイン管理プラグインはneobundle.vim[注4]です。これからVimプラグインをインストールする方は、このneobundle.vimを使用することをお勧めします。pathogen.vimやVundleでサポートされている機能はすべてneobundle.vimでも使用でき、また筆者がneobundle.vimを使用する決め手となった、プラグインの遅延読み込み機能を使用できます。

筆者はVimプラグインを".vim"ディレクトリにインストールする方法からpathogen.vim、Vundleと使ってきましたが(一時期GetLatestVimScriptsも試してみましたが、すぐに使わなくなってしまいました[注5])、次第にインストールするプラグインの数が多くなり、Vimの起動がかなり遅くなってしまったのです[注6]。そのため、起動時間を短縮しようと、vim-ipi[注7]や自作の関数などを駆使して特定のイベントが発生した際にプラグインを読み込む設定をしていましたが、neobundle.vimの作者であるShougo氏がプラグインの遅延読み込み機能を実装してくれたことにより、よりスマートに遅延読み込み機能を使用できるようになりました。

さて、前置きが長くなってしまいましたが、neobundle.vimの機能と設定方法を紹介していきたいと思います。その前に、neobundle.vimのインストール方法ですが、コンソールから**図2**のコマンドを実行します。

続いてVim上でneobundle.vimを使用するため、vimrcに**リスト1**の設定を追加します。これでneobundle.vimを使用する準備が整いました。

それではさっそくプラグインをインストールしてみましょう。neobundle.vimには**表1**のコマンドが用意されています。

Vimの実行中にプラグインをインストールする際にはNeoBundleDirectInstallコマンドが便利です。または、vimrcにNeoBundleコマンドでインストールするプラグインの設定を追加し、Vim起動後にNeoBundleInstallコマンドを実行することでインストールできます。また、新規にプラグインを作成する際やGitHubやBitbucketなどで管理されていないプラグインを使用する場合などはNeoBundleLocalコマンドで対象となるプラグインが配置されているディレクトリを指定するとプラグインを読み込むことができます。また、プラグインの遅延読み込みを行う場合は**リスト2**のように、NeoBundleコマン

注3) https://github.com/gmarik/vundle
注4) https://github.com/Shougo/neobundle.vim

▼図2　neobundleのインストール

```
$ cd ~/.vim
# インストール先のディレクトリを作成
$ mkdir bundle
$ cd bundle
$ git clone git://github.com/Shougo/neobundle.vim
```

▼リスト1　neobundleの設定

```
" vim起動時のみruntimepathにneobundle.vimを追加
if has('vim_starting')
    set runtimepath+=~/.vim/bundle/neobundle.vim/
endif

" neobundle.vimの初期化
call neobundle#rc(expand('~/.vim/bundle'))

" neobundle.vimを更新するための設定
NeoBundleFetch 'Shougo/neobundle.vim'

" 読み込むプラグインを記載
" NeoBundle 'プラグイン名'

" 読み込んだプラグインも含め、ファイルタイプの検出、
" ファイルタイプ別プラグイン/インデントを有効化する
filetype plugin indent on
```

注5) 詳細は:help glvsを参照。
注6) プラグイン数が約80個で起動時間は10秒以上。
注7) https://github.com/jceb/vim-ipi

▼リスト2　遅延読み込みの設定

```
" NeoBundleLazyコマンドを使用し、
" :Vinariseコマンド実行時に読み込む
NeoBundleLazy 'Shougo/vinarise', {
          ¥ 'autoload': {
          ¥     'commands': 'Vinarise',
          ¥ }}

" NeoBundleコマンドを使用し、
" :Vinariseコマンド実行時に読み込む
NeoBundle 'Shougo/vinarise', {
          ¥ 'lazy': 1,
          ¥ 'autoload': {
          ¥     'commands': 'Vinarise',
          ¥ }}
```

ドに“lazy”オプションを設定するかNeoBundle Lazyコマンドを使用します。

NeoBundleコマンドはほかにもさまざまなオプションがありますが、誌面でそのすべてを紹介するのは困難なため、この先で紹介するプラグインのインストール例(**リスト3**)を参考にし、詳細についてはプラグインのドキュメントを参照してください。

▼表1　neobundle.vimのコマンド

コマンド名	概要	使用例
NeoBundle	neobundle.vimで管理するプラグインを設定する	:NeoBundle 'Shougo/vimproc.vim'
NeoBundleCheck	未インストールのプラグインがあった場合はインストールする	:NeoBundleCheck
NeoBundleLazy	neobundle.vimで管理するプラグインを設定し、“lazy”オプションを有効にする	:NeoBundleLazy 'Shougo/vimproc.vim'
NeoBundleFetch	指定したリポジトリのダウンロードのみを行う	:NeoBundleFetch 'Shougo/vimproc.vim'
NeoBundleLocal	プラグインの管理は行わないが、読み込みは行うフォルダを設定する	:NeoBundleLocal ~/.vim/local
NeoBundleDepends	プラグインの依存関係を設定する。主にプラグインの作成時に利用する	:NeoBundleDepends 'Shougo/vimproc.vim'
NeoBundleDirectInstall	プラグインを直接インストールする。インストールしたプラグインの情報は“direct_bundles.vim”に記録される	:NeoBundleDirectInstall 'Shougo/vimproc.vim'
NeoBundleDirectEdit	NeoBundleDirectInstallでインストールしたプラグインの設定を編集する	:NeoBundleDirectEdit
NeoBundleSource	“lazy”オプションが設定されているプラグインを読み込む。プラグイン名が指定されなかった場合は“lazy”オプションが設定されていて、未読み込みのプラグインをすべて読み込む	:NeoBundleSource vimproc.vim、:NeoBundleSource
NeoBundleDisable	指定したプラグインを無効にする	:NeoBundleDisable vimproc.vim
NeoBundleInstall	プラグインをインストールする。プラグイン名が指定されなかった場合はすべてのプラグインをインストールする。“!”を付けた場合はNeoBundleUpdateと同じ	:NeoBundleInstall vimproc.vim、:NeoBundleInstall! vimproc.vim、:NeoBundleInstall、:NeoBundleInstall!
NeoBundleUpdate	プラグインを更新する。プラグイン名が指定されなかった場合はすべてのプラグインをアップデートする。“!”を付けた場合は“stay_same”オプションを無視して更新を行う	:NeoBundleUpdate vimproc.vim、:NeoBundleUpdate! vimproc.vim、:NeoBundleUpdate、:NeoBundleUpdate!
NeoBundleClean	プラグインを削除する。プラグイン名が指定されなかった場合はプラグインのインストールパスに存在するneobundle.vimで管理されていないプラグインをすべて削除する	:NeoBundleClean vimproc.vim、:NeoBundleClean
NeoBundleReinstall	指定したプラグインを再インストールする	:NeoBundleReinstall vimproc.vim
NeoBundleList	neobundle.vimで管理されているプラグインの一覧を出力する	:NeoBundleList
NeoBundleDocs	neobundle.vimで管理されているプラグインのヘルプタグを作成する	:NeoBundleDocs
NeoBundleLog	前回のインストール時のログを表示する	:NeoBundleLog
NeoBundleUpdatesLog	前回の更新時のログを表示する	:NeoBundleUpdatesLog

Text objectsを拡張（textobj-user）

Vimにはtext objectsという機能があり、特定の範囲に対して編集を行う際にとても便利です。標準では単語、文、段落、タグ、記号("、'、(、[、{、<)などが用意されていますが、textobj-user[注8]というプラグインを使用すると標準では用意されていない部分をtext objectsとして扱うことができます。簡単な例として、vimrcに**リスト4**の設定を追加すると、“aR”または“iR”とキー入力することで“-”を使用した罫線部分をtext objectsとして扱うことができるようになります。

筆者はこの設定を、**図3**のようなreSTドキュメントの表を編集する際に使用しています。

reSTの表は行の結合をする際に、対象となるカラムの中間の罫線を空白で表現していますが、罫線を削除するたびに削除キーを複数回押すのが面倒なため、一括で選択できるtext objectsを定義しました。ただし、この設定だけでは効率的に罫線を削除できないため、後述のoperatorと組み合わせて使用することになります。

また、textobj-userにはtextobj-userを拡張するためのプラグインが数多くあります。

注8）https://github.com/kana/vim-textobj-user

▼リスト3　この先紹介するプラグインのインストール例

```
NeoBundle 'kana/vim-textobj-user'

NeoBundle 'kana/vim-operator-user'

NeoBundle 'Shougo/unite.vim', {'lazy': 1,
           ¥ 'depends': ['Shougo/vimfiler',
           ¥             'Shougo/vimshell',
           ¥            ],
           ¥ 'autoload': {
           ¥     'commands': [{'name': 'Unite',
           ¥                  'complete': 'customlist,unite#complete_source'},
           ¥                  'UniteWithBufferDir',
           ¥                  'UniteWithCurrentDir',
           ¥                  'UniteWithCursorWord',
           ¥                  'UniteWithInput'],
           ¥ }}

if has('lua')
    " if_luaが使用可能であればneocompleteを読み込む
    NeoBundle 'Shougo/neocomplete', {
                ¥ 'autoload': {
                ¥     'insert': 1,
                ¥ }}
else
    NeoBundle 'Shougo/neocomplcache', {'lazy': 1,
                ¥ 'autoload': {
                ¥     'insert': 1,
                ¥ }}
endif

NeoBundle 'thinca/vim-quickrun', {'lazy': 1,
           ¥ 'autoload': {
           ¥     'commands': [{'name': 'QuickRun',
           ¥                  'complete': 'customlist,quickrun#complete',}],
           ¥     'mappings': ['nxo', '<Plug>(quickrun)'],
           ¥ }}
```

operatorを拡張（operator-user）

続いて紹介するのはtextobj-userのように、operatorをユーザが定義できるようになる、operator-user[注9]です。operatorとは変更や削除、ヤンクといった編集操作のことです。標準で用意されているoperatorをtext-objectsに組み合わせるだけでも柔軟な編集ができますが、少し複雑な編集作業を行おうとすると標準のoperatorでは物足りなくなってくることがあります。

たとえば、先のtextobj-userの紹介で定義したtextobj-ruledlineと組み合わせて使用する場合ですが、標準のoperatorの機能だけで“-”を“ ”（半角スペース）にしようとした場合、“daR”とキー入力していったん“-”を削除した後、削除した“-”と同じ数だけ“ ”（半角スペース）を入力する必要があります。カラムの文字数が少ないうちはすべて手で打ってしまっても良いかもしれませんが、文字数の多いカラムで同じことをやろうとすると手で入力する方法は非常に効率が悪いです。そこでvimrcに**リスト5**の設定を追加し、対象の範囲を空白で置き換える新たなoperatorを作成します。

リスト5ではこの機能を“<Leader>b”キーに設定したため、textobj-ruledlineの“aR”キーと組み合わせて“<Leader>baR”とキー入力することでカーソルのある場所の罫線をすべて空白で置き換えることができます。

単純にreSTの罫線を空白で置き換えるだけであれば、textobj-betweenと置換（“r”キー）の組み合わせでも行うことができますが、繰り返し操作（“.”キー）ができず、また半角文字と全角文字が混在している環境では操作後の表示が崩れてしまうため、operator-userの機能を用いることでより汎用性の高い操作を定義できます。

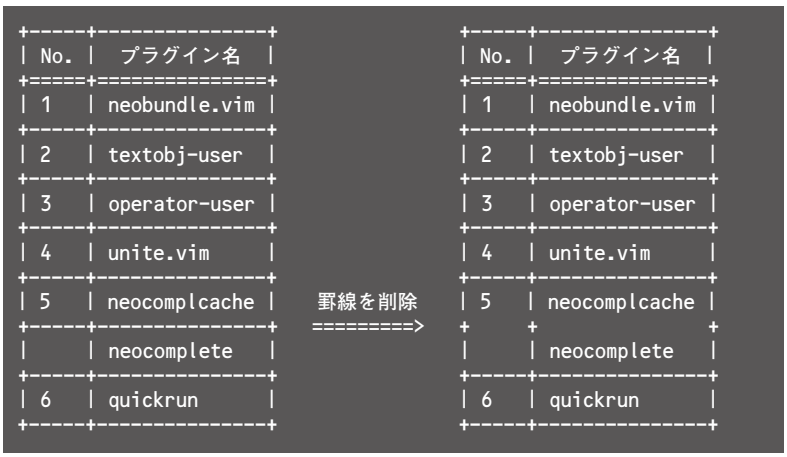
また、operator-userもtextobj-userと同様にoperator-userを拡張するためのプラグインが多数あります。

注9）https://github.com/kana/vim-operator-user

▼リスト4　textobj-userの使用例

```
" 第1引数: 作成するtext objectsの名称、
" 第2引数: 対象とする正規表現パターンと動作
call textobj#user#plugin('ruledline', {
            ¥     '-': {
            ¥         '*pattern*': '-¥+',
            ¥         'select': ['aR', 'iR'],
            ¥     },
            ¥ })
```

▼図3　textobj-ruledlineの利用シーン



▼リスト5　operator-userの使用例

```
" 対象範囲を空白で置き換える関数
function! OperatorFillBlank(motion_wise)
    " 選択時のモードを取得
    let v = operator#user#visual_command_from_wise_name(a:motion_wise)

    " 選択範囲を取得
    execute 'normal! `[' .v. '`]"xy'
    let text = getreg('x', 1)

    " 空白で埋める
    let text = s:map_lines(text,
                \ 'substitute(v:val, ".", "¥¥=s:charwidthwise_r(submatch(0))", "g")')

    " 対象範囲を置換
    call setreg('x', text, v)
    normal! gv"xp
endfunction

function! s:charwidthwise_r(char)
    " 半角・全角文字によって空白の数を変える
    return repeat(' ', exists('*strwidth') ? strwidth(a:char) : 1)
endfunction

function! s:map_lines(str, expr)
    return join(map(split(a:str, '¥n', 1), a:expr), "¥n")
endfunction

" 第1引数: 作成するoperatorの名称、
" 第2引数: 呼び出される関数
call operator#user#define('fillblank', 'OperatorFillBlank')

" 作成したoperatorを<Leader>bキーに割り当てる
map <Leader>b <Plug>(operator-fillblank)
```

インターフェースを拡張（Unite）

ファイラーとして紹介されることのあるunite.vim[注10]ですが、ファイラーはあくまでもunite.vimの1機能に過ぎず、実際には「すべてを破壊し、すべてをつなげ」のコンセプトどおり、インターフェースを拡張するプラグインです。unite.vimの動作を簡単に説明すると、専用バッファで指定した情報源（source）から対象となる候補を絞込んで候補の種類（kind）ごとに設定されている実行可能な動作（action）を選択して実行する、とい

▼図4　uniteのコマンド

```
" 通常起動
:Unite {source名}

" 絞込みの初期値として、現在のディレクトリのパスが入力された状態で起動
:UniteWithCurrentDir {source名}

" 絞込みの初期値として、現在のバッファのパスが入力された状態で起動
:UniteWithBufferDir {source名}

" 絞込みの初期値として、カーソル下の単語が入力された状態で起動
:UniteWithCursorWord {source名}

" 絞込みの初期値として、ユーザが入力した単語が入力された状態で起動
:UniteWithInput {source名}
```

注10）https://github.com/Shougo/unite.vim

うものです。unite.vimは標準で**表2**のsourceが用意されており、**図4**のコマンドの引数として1つ以上のsource名を指定すると起動できます。

各sourceは**表3**のkindに対応しています。sourceによっては個別にactionが定義されていることもありますが、基本的には対応しているkindに設定されているactionを実行できます。

また、すべてのkindはcommonを親としており、それ以外の親kindが設定されている場合はその親のkindが実行可能なactionも実行できます。

このように、単体でも非常に多くの機能を持っ

▼表2　unite.vimのsources

source名	kind	対象となる候補
bookmark	directory, jump_list	:UniteBookmarkAddコマンドなどで設定したブックマーク
buffer	buffer	起動中のVimで開いたバッファ名
buffer_tab	buffer	現在のタブで開いたバッファ名
tab	tab	起動中のVimで開いたタブ名
window	window	起動中のVimで開いたウィンドウ名
change	jump_list	:changesコマンドの結果
jump	jump_list	:jumpsコマンドの結果
line	jump_list	現在のバッファの行
line/fast	jump_list	現在のバッファの行の一部
directory	directory	ディレクトリ名
directory/new	directory	新規作成するディレクトリ名
directory_mru	directory	最近使用したディレクトリ名
directory_rec	directory	サブディレクトリ配下も含むディレクトリ名
directory_rec/async	directory	サブディレクトリ配下も含むディレクトリ名(候補は非同期に収集)
runtimepath	directory	runtimepathに設定されているディレクトリ名
file	file	ファイル名
file/new	file	新規作成するファイル名
file_mru	file	最近使用したファイル名
file_rec	file	サブディレクトリ配下も含むファイル名
file_rec/async	file	サブディレクトリ配下も含むファイル名(候補は非同期に収集)
file_point	file, uri	カーソル下のファイル名やURI
jump_point	jump_list	カーソル下の"ファイル名:行番号"形式の文字列
find	command, directory, find	findコマンドの結果
grep	file, jump_list	grepコマンド(agやackに変更可)の結果
vimgrep	file, jump_list	:vimgrepコマンドの結果
launcher	guicmd	$PATH配下の実行ファイル名
command	command	Vimのコマンド名
function	command	Vimの関数名
mapping	command	Vimのマッピング
alias	-	g:unite_source_alias_aliasesで設定したsource名
menu	source, command	g:unite_source_menu_menusで設定した内容
history_yank	word	yankした内容
output	word	Vimコマンドの実行結果
register	word	レジスタの内容
process	common	プロセス情報
undo	common	undo情報
resume	command	中断したsource名
source	source	unite.vimのsource名

▼表3　unite.vimのkindsとactions

kind/action	動作
common	
nop	何もしない(actionとして選択不可)
yank	候補の文字列をヤンクする
yank_escape	候補の文字列から記号を\でエスケープしてヤンクする
ex	候補の文字列をコマンドラインに挿入する
insert	候補の文字列をカーソル下に挿入する
insert_directory	候補のディレクトリ名をカーソル下に挿入する
preview	候補の文字列をコマンドラインで表示する
echo	候補の情報をコマンドラインで表示する
openable	
tabopen	候補を新規タブで開く
tabdrop	候補が開いていればそのタブに移動し、開いていなければ新規タブで開く
split	候補を水平分割して開く
vsplit	候補を垂直分割して開く
left	候補を垂直分割して左のウィンドウで開く
right	候補を垂直分割して右のウィンドウで開く
above	候補を水平分割して上のウィンドウで開く
below	候補を水平分割して下のウィンドウで開く
persist_open	候補を開いた後、uniteバッファに戻る
cdable	
cd	候補のパスをカレントディレクトリとして設定する
lcd	候補のパスをローカルカレントディレクトリとして設定する
project_cd	候補のパスのトップディレクトリをカレントディレクトリとして設定する
tabnew_cd	候補のパスをカレントディレクトリとして設定したタブを新規に開く
narrow	候補を候補のディレクトリ名で絞り込む
edit	narrowと同じ
vimshell	候補のディレクトリでvimshellを起動する※1
tabvimshell	候補のディレクトリで新規タブを作成し、vimshellを起動する※1
vimfiler	候補のディレクトリでvimfilerを起動する※2
tabvimfiler	候補のディレクトリで新規タブを作成し、vimfilerを起動する※2
uri	
start	候補をブラウザで開く※3
file	**openable, cdable, uriを親とする**
open	候補のファイルを開く
preview	候補のファイルをプレビューウィンドウで表示する
mkdir	候補のファイル名をデフォルト値としてディレクトリを作成する
rename	候補のファイル名をリネームする
backup	候補のファイルのバックアップを作成する
wunix	候補のファイルをunixフォーマットで書き込む
diff	候補のファイルとカレントバッファ、または候補のファイル同士のdiffを表示する
dirdiff	候補のディレクトリ同士でdiffを表示する※4
grep	候補のファイルを対象にgrepする
grep_directory	候補のディレクトリを対象にgrepする
move	候補のファイルを移動する
copy	候補のファイルをコピーする
directory	**fileを親とする**
diff	候補のディレクトリ同士でdiffを表示する※4
tabopen	候補のディレクトリで新規タブを作成し、vimfilerを起動する※2
buffer	**fileを親とする**
open	候補のバッファを開く
goto	候補のバッファがあるタブを開く
delete	候補のバッファをメモリから取り除き、バッファリストから削除する
fdelete	候補のバッファが編集中でもメモリから取り除き、バッファリストから削除する
wipeout	候補のバッファをマークやオプション設定も含めて完全に削除する
unload	候補のバッファをメモリから取り除く
preview	候補のバッファをプレビューウィンドウで表示する
rename	候補のバッファ名を変更し、ファイルが存在する場合はリネームも行う
window	**cdableを親とする**
open	候補のウィンドウを開く
only	候補のウィンドウを開き、ほかのウィンドウを閉じる
delete	候補のウィンドウを閉じる
preview	候補のウィンドウのカーソル行をハイライトする
tab	**cdableを親とする**
open	候補のタブを開く
delete	候補のタブを閉じる
preview	候補のタブを表示し、uniteバッファに戻る
rename	候補のタブ名を変更する
edit	renameと同じ
jump_list	**openableを親とする**
open	候補の位置を開く
preview	候補の位置をプレビューウィンドウで表示する
highlight	候補の位置をハイライトする
replace	候補の位置を:Qfreplaceで置換する※5
rename	replaceと同じ
command	
execute	候補のコマンドを実行する
edit	候補のコマンドを編集したあとに実行する
guicmd	
execute	候補の外部コマンドを実行する
edit	候補の外部コマンドを編集したあとに実行する
source	
start	候補のsourceを実行する
edit	候補のsourceの引数を編集して実行する
word	
-	固有のactionなし

※1　vimshell(https://github.com/Shougo/vimshell.vim)が必要
※2　vimfiler(https://github.com/Shougo/vimfiler.vim)が必要
※3　open-browser.vim(https://github.com/tyru/open-browser.vim)が必要
※4　DirDiff.vim(https://github.com/vim-scripts/DirDiff.vim)が必要
※5　vim-qfreplace(https://github.com/thinca/vim-qfreplace)が必要

ているunite.vimですが、unite.vim用のプラグインを導入することでそのインターフェースをさらに拡張できます。

最初に紹介したneobundle.vimと組み合わせることでオンライン上のプラグインを検索できます(**図5**)。unite.vimをさらに拡張したい方はぜひ試してみてください。

補完を拡張(neocomplcache/neocomplete)

Vimには標準で非常に多くの補完機能があります(**表4**)。

しかし、これらの補完機能は最近のIDEとは異なり、自動で起動してくれず、入力中にユーザが補完機能を呼び出すキーを押す必要があります。また、それぞれの補完機能ごとに呼び出すためのキーが異なっているため、よほど熟練したユーザでなければVimの補完を使いこなすのは難しいかもしれません。

そこで、neocomplcache.vim[注11]またはneocomplete.vim[注12]を用いることで、これらの補完機能を自動で呼び出すことができるようになります。neocomplete.vimはneocomplcache.vimの後継として作成されたプラグインで、if_luaの機能を用いることによりneocomplcache.vimと比べて高速で補完を行うことができるようになっています。もしお使いのVimがif_luaに対応しているのであればneocomplete.vimをインストールすることをお勧めします。

またneocomplcache.vimとneocomplete.vimは別のプラグインを用いて補完機能を拡張できます。本誌では補完プラグインの作成方法については解説しませんが、興味のある方は挑戦してみてはいかがでしょうか。

それ以外にもオムニ補完を拡張するプラグインを組み合わせて使用することもできます。

▼図5　unite_neobundle_searchの使い方

```
:Unite neobundle/search
↓
Please input search word:
↓
Please input search word: unite<Enter>
```

注11) https://github.com/Shougo/neocomplcache.vim
注12) https://github.com/Shougo/neocomplete.vim

▼表4　Vimの入力補完

補完機能	キー	概要
行(全体)補完	Ctrl-x Ctrl-l	現在行のカーソルの前にあるのと同じ文字で始まる行で補完
局所キーワード補完	Ctrl-x Ctrl-n、Ctrl-x Ctrl-p	カーソルの前にあるキーワードで始まる単語をファイルの前方(後方)から検索して補完
辞書補完	Ctrl-x Ctrl-k	"dictionary"で与えられたファイルで補完
シソーラス補完	Ctrl-x Ctrl-t	"thesaurus"で与えられたファイルで補完
パスパターン補完	Ctrl-x Ctrl-i	編集中と外部参照(インクルード)しているファイルのキーワードで補完
タグ補完	Ctrl-x Ctrl-]	カーソルの直前と同じ文字で始まるタグで補完
ファイル名補完	Ctrl-x Ctrl-f	カーソルの直前と同じ文字で始まる最初のファイル名で補完
定義補完	Ctrl-x Ctrl-d	カーソルの直前と同じ文字で始まる最初の定義(もしくはマクロ)名で補完
コマンドライン補完	Ctrl-x Ctrl-v	コマンドラインでの入力時と同様にVimコマンドを補完
ユーザ定義補完	Ctrl-x Ctrl-u	"completefunc"で設定した関数で補完
オムニ補完	Ctrl-x Ctrl-o	"omnifunc"で設定した関数で補完
スペリング補完	Ctrl-x Ctrl-s、Ctrl-x s	カーソル前の単語の正しい綴りの候補で補完
キーワード補完	Ctrl-n Ctrl-p	カーソルの直前と同じ文字で始まる単語を"complete"で指定された場所から補完

プログラム実行を拡張（quickrun）

Vimでちょっとしたスクリプトなどを編集している際、スクリプトを実行するのにコンソールに戻るのが面倒だと思ったことがある方は多いのではないかと思います。もちろん":!%"コマンドや":r!%"コマンドなどで実行しても良いのですが、コンソール画面の出力だと結果が見づらいことがあり、かといって編集中バッファに結果を出力するのもあまりスマートではありません。

そんなときに便利なのがquickrun[注13]です。quickrunはファイルタイプごとに設定されているプログラムを実行し、実行結果を新規バッファなどに出力します。quickrunはデフォルトで非常に多くのファイルタイプに対応しており、対応しているファイルタイプであればとくに設定をすることなく、:QuickRunコマンドや"<Leader>r"キーで実行するプログラムを判別し、結果を出力してくれます。このときに、実行対象のバッファは保存されている必要はなく、ファイルを変更したあとに保存前の動作確認や、ビジュアルモードで選択した範囲のみを実行するといったこともできます。

また、quickrunは出力先をバッファだけでなく、ファイルやブラウザなどに指定できます。筆者はスクリプト言語の実行環境だけでなく、markdownやreST、textileといったドキュメントを書く際にもquickrunを使用し、ブラウザにプレビューを表示するといった使い方をしています。

quickrunは先に紹介したunite.vimのようなユーザが複数の候補から対象を選択して動作を実行するといった思想とは異なり、次のように実行する内容があらかじめ決まっている場合には非常に使い勝手の良いフレームワークを提供してくれます。

- 何を（バッファ、ファイル、ディレクトリなど）
- どう実行し（perl、make、gccなど）
- どこに出力するか（Vim、ファイル、ブラウザなど）

また、quickrunはquickrunの機能を利用してシンタックスチェックを行うwatchdogs.vim[注14]と、quickrunにさまざまなフック処理を追加するshabadou.vim[注15]を組み合わせることで多言語に対応したシンタックスチェッカーとしても使用できます。このとき、hier[注16]とquickfixstatus[注17]をインストールしておくと、Vim上でIDEのようにエラーや警告のある行のハイライトやエラー詳細の表示が可能になります。

おわりに

今回紹介したプラグインはいずれも非常に高い拡張性を持っているものばかりのため、本誌で解説した内容以上に高度な設定ができます。また、独自のプラグインを作成することで"プラグインのプラグイン"としての利用が可能になっています。基本的な設定で物足りなくなってしまった方はさらに高度な設定や新たなプラグインを作成してみてはいかがでしょうか。

なお、具体的なプラグインの紹介やプラグインの設定例など、今回誌面で紹介しきれなかった内容は、本誌のサポートページ[注18]からダウンロードできますのでご利用ください。SD

注13）https://github.com/thinca/vim-quickrun

注14）https://github.com/osyo-manga/vim-watchdogs
注15）https://github.com/osyo-manga/shabadou.vim
注16）https://github.com/jceb/vim-hier
注17）https://github.com/dannyob/quickfixstatus
注18）http://gihyo.jp/magazine/SD/archive/2013/201310/support

COLUMN 3

Vimで Objective-Cプログラミング

Writer 所 友太(ところ ゆうた) クックパッド㈱ Twitter: @tokorom Web:www.tokoro.me

基本的な考え方

Objective-Cを書くということはiOSアプリケーションやMacアプリケーションを開発するということだと思います。その場合、Xcodeで開発するのが普通でVimを使って書くというのはある意味で邪道です。とはいえ、Vimを使い慣れている開発者にとってVim以外のエディタでプログラミングをすることは苦痛を伴いますのでなんとか使いたいという気持ちはわかります。そんな中、筆者がこれまで3年間、VimでObjective-Cを書き続けてきた結論は、

- すべてをVimでやろうと思わない
- Xcodeでやるべきことは Xcodeでやり、Vimでやるべきことを Vimでやる

のが良いということです。

XcodeにXVimやViEmuなどのプラグインをインストールして、Xcode上にVimっぽい環境を実現する方法も考えられます。しかし筆者の経験上では、そのプラグインのできがよければ良いほど不気味の谷に落ちていってしまう(微妙な違いにストレスを感じるようになる)ものと思います。

VimとXcodeを瞬時に切り替える

具体的には筆者の場合、**表1**に表すような使い分けをしており、Objective-Cのコーディングはすべて Vimで行い、それ以外のことはXcodeに切り替えて行っています。そのため、コーディングとビルド&実行の繰り返しのリズムを崩さないためにVimとXcodeを瞬時に切り替えるためのツールが必須です。そのために利用しているのが**TotalTerminal**です。

筆者が利用しているVimは**macvim-kaoriya**[注1]で、それを**Terminal.app**上で動かして使っています。TotalTerminalは、そのTerminalを瞬時に表示したり非表示にしたりできる(デフォルトでは Ctrl キーを2回叩くだけ)ツールです(**図1**)。

これにより、Vimでコーディング→Xcodeでビルド&動作確認→おかしなところをふたたびVimで修正→修正個所をXcodeで実行して確認といったことが流れるようにできるようになりました。

Objective-Cのコード補完

VimでObjective-Cを書くうえでよく問題視されるのがコード補完です。Xcodeのコード補完はとてもパワフルなため、同様のことがVimでも実現できるかのかという話です。結論としてはVimでも一定のところまでできます(ただし、コード補完だけ見るとXcodeのほうが快適なことはたしかです)。

注1) macvim-kaoriya(http://code.google.com/p/macvim-kaoriya/)

▼表1 おもなOIP

Vim or Xcode	タスク
Vim	Objective-Cのコーディング
Xcode	ビルド、実行、Storyboard編集、プロジェクト設定

Objective-Cのコード補完はclangによって実現できますのでVimからclangを呼び出せばそれができます。これについてはclang_complete[注2]というずばりそのもののプラグインがありますのでこれを使うのが一番簡単と思います。

なお、筆者自身はというとコード補完はそれほど重要視しておらず、むしろコードスニペットのほうをおもに活用しています。コードスニペットのためのプラグインとしてはneocomplcache[注3]とneosnippet[注4]を使っています。

clang_completeの利用方法

clang_completeですが、Objective-Cの場合にはインストールして即利用できるわけではありません。

まず、2013年8月19日時点のclang_completeにはObjective-Cのメソッドを補完するときに引数がたくさんあるときちんと補完できないという問題があるので、それが気になる方は筆者のほうでForkしたバージョン[注5]を使ってみてください。

次に、clang_completeに対してビルド時に指定するオプションを教えてあげる必要があります。具体的にはプロジェクトのルートディレクトリ（Vimのカレントディレクトリになる場所）に.clang_completeというファイルを設置して、そこに必要なオプションを記述する必要があります（**リスト1**）。

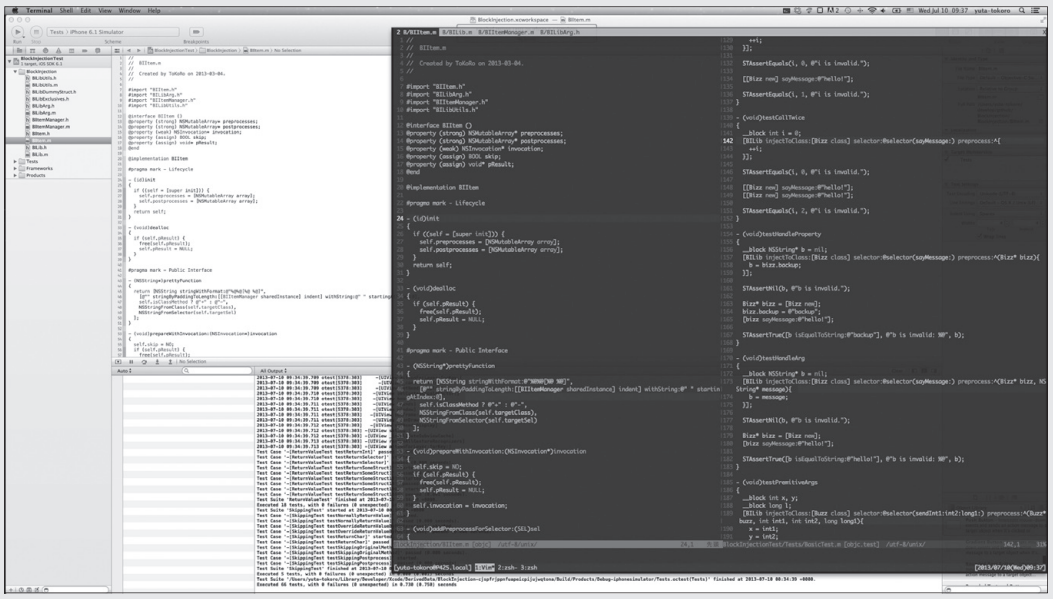
表2でそれぞれのオプションについて説明します。

なお、カレントディレクトリ以外にインクルードすべきヘッダファイルが存在する場合はそのディレクトリも**-I**オプションで追記してやる必要があります。たとえば、Sampleというサブディレクトリ以下にいくつかヘッダーファイルがある場合は**-I Sample**を追記する必要があります。

これらの設定がうまくいけば、Objective-Cのファイルの中で、たとえば「**self.v**」と入力した時点で**図2**のように入力候補が表示されるはずです。

注2）https://github.com/Rip-Rip/clang_complete
注3）https://github.com/Shougo/neocomplcache.vim
注4）https://github.com/Shougo/neosnippet.vim
注5）https://github.com/tokorom/clang_complete

▼図1　筆者の開発環境



neocomplcacheとの併用

neocomplcacheをご利用の方はneocomplcacheのdocに従って、vimrcに**リスト2**の設定を追加する必要があります。

Objective-Cのシンタックスハイライト

VimでのObjective-Cのシンタックスハイライトですが、デフォルトでは**図3**のように限られた部分しかハイライトされません。せっかくですから**図4**のようにメソッド名やクラス名などもハイライトされてほしいものです。このシンタックスハイライトですが、古き良き**cocoa.vim**[注6]できれいにサポートされていました。しかし、このcocoa.vimは残念ながら3年以上前からメンテナンスが止まってしまっており、ここからシンタックスハイライトの部分だけ抜き

注6） https://github.com/msanders/cocoa.vim

▼リスト1　.clang_completeの具体例

```
-w
-fblocks
-fobjc-arc
-D __IPHONE_OS_VERSION_MIN_REQUIRED=40300
-include ./**/*-Prefix.pch
-F /Applications/Xcode.app/Contents/Developer/Platforms/iPhoneSimulator.platform/Developer/SDKs/iPhoneSimulator6.1.sdk/System/Library/Frameworks
-I /Applications/Xcode.app/Contents/Developer/Platforms/iPhoneSimulator.platform/Developer/SDKs/iPhoneSimulator6.1.sdk/usr/include
```

▼表2　clang_completeのオプションの解説

オプション	説明
-w	すべての警告を無視する（コード補完では警告はいらない）
-fblocks	ブロック構文を利用している場合に指定するオプション
-fobjc-arc	ARCを使用している場合に指定するオプション
-D __IPHONE_OS_VERSION_MIN_REQUIRED=40300	ターゲットがiOS 4.3以上であることを示す（必要なら変更）
-include ./**/*-Prefix.pch	プロジェクト内のpchファイルを指定（具体的なファイル名を指定しても良い）
-F /（略）/iPhoneSimulator6.1.sdk/System/Library/Frameworks	iOSのフレームワークがあるディレクトリの指定（利用するSDKが違う場合は変更必須）
-I /（略）/iPhoneSimulator6.1.sdk/usr/include	iOSのヘッダファイルがあるディレクトリ指定（利用するSDKが違う場合は変更必須）

▼図2　入力コードの補完機能を加える

```
15 @implementation ViewController
16
17 - (void)viewDidLoad
18 {
19     [super viewDidLoad];
20     self.v
21 }
22
23 - (void)
24 {
25     [sup
26 }
27
28 @end
```

```
view                    14 UIView * view
viewWillUnload          16 void viewWillUnload
viewDidUnload           16 void viewDidUnload
viewDidLoad             16 void viewDidLoad
viewWillLayoutSubviews  16 void viewWillLayoutSubviews
viewDidLayoutSubviews   16 void viewDidLayoutSubviews
version                 17 NSInteger version
```

出して使わせていただくのが良さそうです。

実際にForkしてその作業をやってみたものを筆者のGitHubのリポジトリに`syntax-only`というブランチで置いています[注7]。NeoBundleをご利用の方であれば、

```
NeoBundleLazy 'git://github.com/tokorom/cocoa.vim.git', 'syntax-only', {'autoload': {'filetypes': ['objc']}}
```

という設定をvimrcに追記していただければ、Objective-Cのコードを書くときだけこのシンタックスハイライトの設定が読み込まれるようになります。

ソースファイル(.m)とヘッダーファイル(.h)の切り替え

Objective-Cのコーディングをする際にはソースファイルとヘッダーファイルとの行き来が頻繁に発生します。これをスムーズに行うのに**vim-altr**[注8]が便利です。

vim-altrをインストールしたら、Vimの設定ファイル用ディレクトリ以下の**after/ftplugin/objc.vim**あたりに

```
call altr#remove_all()
call altr#define('%.h', '%.m')
```

という設定を追記します。また、これに加えて

```
nmap <Space>a <Plug>(altr-forward)
```

というキーマップを設定することでspace→Aと打鍵することで**XXX.h**と**XXX.m**を瞬時にトグルすることが可能になります。

オマケで

```
call altr#define('en.lproj/%', 'ja.lproj/%')
```

も追記しておくと、ローカライズの際に英語用のファイルと日本語用のファイルをトグルできて便利だったりします。SD

注7）https://github.com/tokorom/cocoa.vim/tree/syntax-only

注8）https://github.com/kana/vim-altr

▼リスト2　vimrcの追加設定

```
if !exists('g:neocomplcache_force_omni_patterns')
  let g:neocomplcache_force_omni_patterns = {}
endif
let g:neocomplcache_force_overwrite_completefunc = 1
let g:neocomplcache_force_omni_patterns.c =
    ¥ '[^.[:digit:] *¥t]¥%(¥.¥|->¥)'
let g:neocomplcache_force_omni_patterns.cpp =
    ¥ '[^.[:digit:] *¥t]¥%(¥.¥|->¥)¥|¥h¥w*::'
let g:neocomplcache_force_omni_patterns.objc =
    ¥ '[^.[:digit:] *¥t]¥%(¥.¥|->¥)'
let g:neocomplcache_force_omni_patterns.objcpp =
    ¥ '[^.[:digit:] *¥t]¥%(¥.¥|->¥)¥|¥h¥w*::'
let g:clang_complete_auto = 0
let g:clang_auto_select = 0
```

▼図3　cocoa.vim適用前

```
 7 #import "ViewController.h"
 8
 9 @implementation ViewController
10
11 - (void)viewDidLoad
12 {
13   [super viewDidLoad];
14
15   // コメント
16   UILabel* label = [UILabel new];
17   label.text = @"LABEL";
18   [label sizeToFit];
19 }
20
21 - (void)didReceiveMemoryWarning
22 {
23     [super didReceiveMemoryWarning];
24 }
```

▼図4　cocoa.vim適用後

```
 7 #import "ViewController.h"
 8
 9 @implementation ViewController
10
11 - (void)viewDidLoad
12 {
13   [super viewDidLoad];
14
15   // コメント
16   UILabel* label = [UILabel new];
17   label.text = @"LABEL";
18   [label sizeToFit];
19 }
20
21 - (void)didReceiveMemoryWarning
22 {
23     [super didReceiveMemoryWarning];
24 }
```

第4章

生産性を向上させるVimのTips

Writer mattn Twitter@mattn_jp

いわゆる商用アプリケーションと違い、Vimを使いこなすには主体的に情報を探し、使用目的に合ったプラグインなどを取り入れ、自分なりに工夫を加えることが重要です。本章がガイドとなり、生産性を大幅に向上させるために必要な手がかりを紹介します。

Vimの使いこなしのアイデア

一口にVimと言ってもVimをテキストエディタとして使っている人もいれば、Vimを環境として使っている人もいます。人によってさまざまな使い方があるのです。知らない世界を知ることにより、新たなVimの使い方を得ることができます。

これまで何となしに使っていたVimも、ひと工夫するだけで目から鱗、皆から一目置かれるような使い方になるかもしれません。

一般的にVimは設定ファイルを編集したり、プログラミングに使われることが多いと思います。もちろんVimはプログラミングにはとても有用ですが、プログラミング以外の目的でも使える非常に強力な**環境**でもあります。

Vimの中でシェルを開く「vimshell」

ソースコードの編集のためにVimを使っている人の中には、コンパイルのために毎回Vimを終了している人もいます。もちろんVimは本来高速に起動するテキストエディタですがプラグインを大量に導入している人であれば、Vimの起動時間も気になる程度になっていたりもします。そういった場合、VimShell[注1]を使うと便利です。

Vim上でインタラクティブシェルを実行しようというプロジェクトで、一般的なコマンドや、Vimと親和性の高い連携が行えます(**図1**)。こちらは外部プロセスの起動やファイル入出力にvimprocというライブラリを使用して実現しています。作者はShougo氏。

ファイルを高速に開く「ctrlp.vim」

Vimを使っていると、画面は常にVimが前面に来ており、できればマウスは使わずいろんな操作をしたくなります。筆者の場合、ファイルの選択にプラグインを使用します。fuzzyfinderやunite.vim、ctrlp.vimなど、多々あります。筆者は高速性と安定性を理由にctrlp.vim[注2]と

▼図1 vimshell

```
vimshell% ls -la
合計 32
drwxrwxr-x   5 mattn mattn  4096  5月  1  2012 ./
drwxr-xr-x 119 mattn mattn 12288  7月 28 21:22 ../
-rw-rw-r--   1 mattn mattn    43  4月 29  2012 .modulebuildrc
drwxrwxr-x   2 mattn mattn  4096  6月  9 23:24 bin/
drwxrwxr-x   3 mattn mattn  4096  4月 29  2012 lib/
drwxrwxr-x   4 mattn mattn  4096  5月  1  2012 man/
vimshell% cd lib/perl5/Amon2
~            lib/perl5/Amon2/                              [sh]
~            lib/perl5/Any/                                [sh]
~            lib/perl5/AnyEvent/                           [sh]
~            lib/perl5/App/                                [sh]
~            lib/perl5/AppConfig/                          [sh]
~            lib/perl5/CPAN/                               [sh]
~            lib/perl5/Cache/                              [sh]
~            lib/perl5/Capture/                            [sh]
~            lib/perl5/Carp/                               [sh]
~            lib/perl5/Class/                              [sh]
~
~
```

注1) https://github.com/Shougo/vimshell.vim

注2) http://kien.github.io/ctrlp.vim/

いうプラグインを使っています(**図2**)。

ランチャーとして使う「ctrlp-launcher」

上記のctrlp.vimは拡張が書けるのですが、それを利用してランチャーとして使えます。筆者の場合、ブラウザを起動したりGIMPを起動したりといった操作はこのctrlp-launcher[注3]から行っています。

コンパイルと実行「QuickRun」

プログラムを書き始めるとき、Vimで新しいファイルを開くかと思います。そのあと、動作確認をしながら作っていくことになりますが、そのつどVimを終了させてコンパイルと実行を行ったり、「:make」して「:new +w!./prog」するのも面倒に感じたりします。

Vimを立ち上げたまま、プログラムのコードを開いたまま実行して結果出力させ、その内容をVimで確認したい場合もあります。そのような場合、thinca氏が作ったQuickRun[注4]を使うと便利です(**図3**)。

数多くのファイルフォーマットをサポートしており、ちょっとしたスクリプトを書きながら都度実行し、結果を確認できます。カスタマイズも可能で筆者も愛用しています。

ちなみにpythonでちょっとした物を書き始めるならば、入力補完にjedi-vim、動作確認にQuickRunという組み合わせが最強だと思っています。

Gistにポストする「gist-vim」

たとえばコードの断片や出力結果を誰かに伝える場合、その一部をクリップボードにコピーし、コードスニペットサービスに貼り付けるといったことをよくやります。例を挙げると、バグ報告としてQuickRunで実行した結果を誰かに確認してもらいたいといった場合もあります。そんな場合、gist-vim[注5]を使うと便利です。

```
:Gist
```

を実行するだけで簡単に https://gist.github.com に内容がポストできてしまいます。もちろんヴィジュアル選択して一部だけポストするといったこともできます。

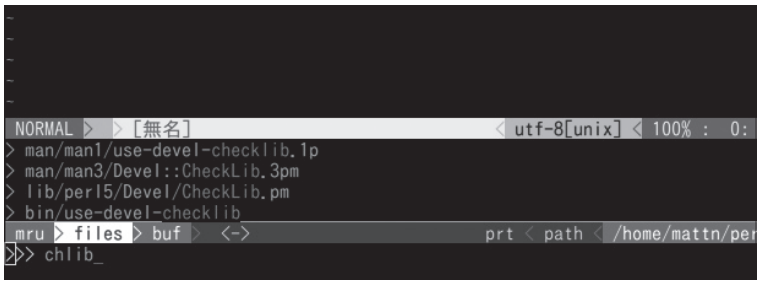
さらに、

```
:Gist -l
```

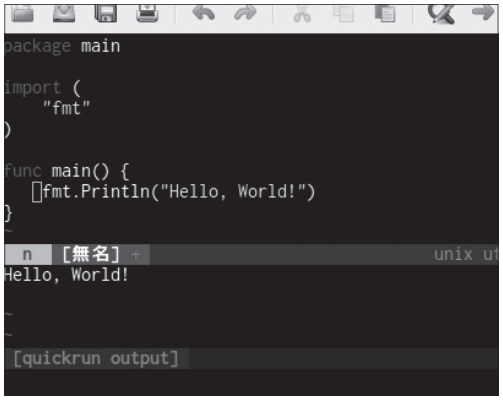
を実行して過去にポストしたGistの一覧を開いて目的のGistを開き、「:w」で更新することもできます。

注3) https://github.com/mattn/ctrlp-launcher

▼図2　ctrlp.vim



▼図3　quickrun



注4) https://github.com/thinca/vim-quickrun
注5) https://github.com/mattn/gist-vim

COLUMN ピュアVim script族

gist-vimはGitHub/GistのAPI呼び出し部分にwebapi-vimというライブラリを使用しています。このwebapi-vimなどもそうですが、基本的に筆者が作るVimプラグインは、perl拡張やpython拡張は使いません。UbuntuやCentOSなどの各ディストリビューションにデフォルトで入る言語拡張が有効でないVimでも動くようにと考えてのことですが、その他にも「外部のモジュールに依存せず、ピュアVim scriptだけで書きたい」という**遊び心**であったりもします。

Vimプラグインを作る人の一部には、こういった「ピュアVim script族」とも言える人達が何人かいて、「まさかこれがVim scriptだけで！？」と言われるような物を書いている方もいます。

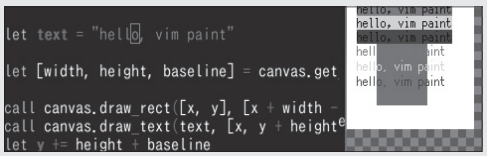
とくにVim script作者界隈ではynkdirさんが有名で、氏のコードを使っているプラグインもいくつかあります。上記で述べたwebapi-vimでも使わせていただいています。

●vim-paint

（https://github.com/ynkdir/vim-paint）

Vimから画像ファイルを出力できるライブラリです（**図4**）。bdfフォントファイルを使って文字列の描画もできます。そもそもバイナリが扱えないVim scriptでここまでできているのがすごいです。

▼図4　vim-paint



●vim-guess

（https://github.com/ynkdir/vim-guess）

日本語エンコーディングの判定はGaucheのロジックが優秀であると言われていますが、そのロジックをVim scriptに移植した物です。筆者も書こうと思いましたが、そのときはすでに先を越されていました。

●vim-patch

（https://github.com/ynkdir/vim-patch）

Vim scriptでpatchコマンドを実装しています。

●vim-diff

（https://github.com/ynkdir/vim-diff）

Vim scriptでdiffコマンドを実装しています。patchコマンドもそうですが「C言語で書かれた実装だと難しいけれどVim scriptなら読めるかも」という人はもしかしたらいるかもしれません。

●vim-vimlparser

（https://github.com/ynkdir/vim-vimlparser）

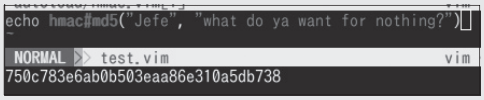
Vim scriptは「変態言語」と呼ばれるほど特殊な仕様の多い言語ですが、これを解析してAST（抽象構文木）に分解しようというプロジェクトです。ここまで来ると変態という言葉が似合います。

●vim-funlib

（https://github.com/ynkdir/vim-funlib）

MD5やSHA1、HMAC、MersenneTwisterなどをVim scriptで実装しようという変態的なプロジェクトですが、参考になるところはかなり多く一部Vim script作者の経典的な存在でもあります（**図5**）。

▼図5　vim-funlib



●lisper-vim

（https://github.com/mattn/lisper-vim）

こちらは筆者のプラグインです。ピュアVim scriptで実装したLispエンジンです。ynkdirさんも同様の物を作っておられました。VimでLispが動いて得する人はまずいません。しかしこの「誰得」がピュアVim script族にとっては最高の褒め言葉でもあるのです。

このように、いろんな物がピュアVim scriptだけで作られています。しかし簡単にピュアVim scriptで実装できるわけではありません。人しれない苦労もあったりします。その多くはバイナリの扱いです。筆者のプラグインで例えます。

●msgpack-vim

（https://github.com/mattn/msgpack-vim）

MessagePackはバイナリを扱います。しかしVim scriptはバイナリを扱えません。そこで関数「msgpack#pack」ではバイナリのhex文字列表現を返す仕様としています。Vim scriptでのバイナリ表現方法には一般的に2種あり、ynkdirさんは255を最大値とした数値の配列で表現しています。さてMessagePackの仕様として浮動小数点型をビッグエンディアンのIEEE754でバイナリを格納する必要があるのですが、これを実現するためにピュアVim scriptでIEEE754を実装しています。

これらはもちろんif_pythonなどを使用すれば簡単に書けてしまうと言うのは筆者もynkdirさんも承知です。

これは一種のスポーツに似た感覚です。あなたもこんな場合に「if_xxx使ったら負け」と思うようになったら、きっと「ピュアVim script族」になってしまったと言えるのではないでしょうか。

コミュニケーションする

さて伝えたい内容がGistにポストできたら実際に伝えたい人とコミュニケーションを行いましょう。VimからTwitterするならTwitVimが便利です。

■TwitVim（http://vim.sourceforge.net/scripts/script.php?script_id=2204）

gist-vim でポストしたURLを

```
:PosttoTwitter
Tweet: @kaoriya これです！ https://gist.
github.com/mattn/6093989
```

このようにしてポストできます。RTやFav、検索などもできます。その他VimからTwitterできる物としてはbasyuraさんが作っているTweetVimがあります。

■TweetVim（https://github.com/basyura/TweetVim）

こちらはGVimでアイコンが表示できたり、ストリーミングAPIを使ってリアルタイムにツィートを表示するといったことができます。

ちなみにVimはTwitterだけでなく、チャットサービスであるLingrもできます。tsukkeeさんが作ったlingr-vimを使います。

■lingr-vim（https://github.com/tsukkee/lingr-vim/）

Vimのpython拡張を使い、LingrAPIのロングポールを別スレッドで実行することで非同期なUIを実現しています。

Twitterやブログ、GitHub等で自作のVimプラグインを紹介することで、ほかのユーザから評価が得られたり間違いを教えてもらえたり、機能追加パッチをもらえたりします。ぜひプラグインを書いたら公開しましょう。

プロジェクトごとの設定を使う

最近ではvimrcをGitHubで公開する人もいます。環境を変えた際には非常に有用なのですが、仕事で使う設定を公のGitHub上に公開できない場合もあります。そんな場合はthincaさんが作っているvim-localrcを使うと便利です。

■vim-localrc（https://github.com/thinca/vim-localrc）

vim-localrcを使うと各ディレクトリごとに任意のvimrcを配置することができ、「~/.vimrc」に書いていた業務に特化した内容を各プロジェクトフォルダに配置することができます。

Vim情報の集め方

Vimをこよなく愛する人達を、一般的にVimmerと呼びますが、そのVimmerが集まりVimの話ばかりしている場所が何ヵ所かあります。

vim-jp（http://vim-jp.org）

日本のVimユーザ達が、日本のユーザに情報を発信しようと作られたハブサイトです。日本ではおそらくこのvim-jp.orgが一番活発に情報を発信しています。

vim-jpができるまでの間、Vimの不具合を見つけたユーザは個々にパッチを作成し、各々がvim-dev（Vimの開発者ML）にパッチを送るという作業を行ってきました。しかし英語で報告するという障壁があってか、多くのユーザが手元のVimを直すだけに止まってしまったり、不具合をどこにも報告できずにいたというケースも多くみられました。vim-jpではそういった不具合報告を集めることを目的に、日本語で報告していただき有識者で議論したり、パッチを書いてvim-devへ報告しています。vim-jp発足後、数多くのパッチが日本人から提供されマージされました。今日でも多くのパッチがvim-jp[注6]から提出されています。

Vimプラグインに関する相談でもかまいません。お気軽にどうぞ。

注6） https://github.com/vim-jp/issues/issues

lingr（http://lingr.com/room/vim）

Vim好きなユーザがVimのことばかり話しているチャットルームです。ここには多くのVim有識者が集まっており、わからないことがあれば気軽に質問できますし、結構な割合で誰かが答えてくれたりします。

このVim部屋では毎週土曜23時からvimrc読書会というイベントを行っており、さまざまな場所から誰かのvimrcを見つけて来ては「この設定いいな」「この設定おかしい」などと約1時間程度の読み合わせを行います。今まで知らなかったオプションやテクニックを得られるチャンスです。筆者もたまに参加しています。

Twitter（http://twitter.com）

TwitterでVimプラグイン作者をfollowしておくとVimに限らずいろんな情報が得られます。Vimに関するジョークが流れてきたり、海外でのVim情報を日本人が紹介してくれていたりもします。

GitHub（https://github.com/）

そしてやはりGitHubでいろんなプロジェクトを探しコードを見ることが、Vim scriptを覚えるうえでも一番の情報収集方法と言えます。日本人が開発している物も多くあります。たとえば最近Vim script作者の中でライブラリとして使われはじめているVital[注7]という物があります。

これまでのVim scriptにはモジュールなどといった概念がなく、Vim script作者が自前で作ったり、ライブラリとしては存在するけれど新しくなって動かなくなったり、Vim scriptに限らない問題が多々発生しました。しかしVitalはプラグイン自身がバージョン指定で必要な分だけを取り込むことで安定した、かつ可読性の高いコードを提供できるようになっています。サンプルを**リスト1**として挙げます。

Vitalは日本人が作っていることもあって、機能要望があれば割に簡単に取り込まれたりします。GitHubには筆者も調べきれないほどの情報が詰まっています。見つけたらぜひ皆さんにも教えてあげてください。その他、筆者の場合はredditで海外の情報を収集したり、stackoverflowでVimに関する質問に答えたりすることで、Vimユーザがいったいどんなことで困っているのかを調べたりしています。そこから新しいプラグインのアイデアが生まれることもあります。

注7） https://github.com/vim-jp/vital.vim

▼リスト1　Vitalのサンプルスクリプト

```
" vital を使って vital の機能を呼び出す。
" ここは各プラグインの名称となる。
let s:V = vital#of('vital')

" 各モジュールをインポートする。
let s:P = s:V.import('Prelude')
let s:H = s:V.import('Web.HTTP')
let s:J = s:V.import('Web.JSON')

" github の API を使って vim-jp に報告されている issues を取得する。
let url = 'https://api.github.com/repos/vim-jp/issues/issues'
let content = s:H.get(url).content
let issues = s:J.decode(content)

" 最大70文字の範囲でタイトルを表示する。
for issue in issues
  echo s:P.truncate(issue['title'], 70)
endfor
```

日本でのVimの広まり

筆者はほどほどVimに関する情報を集めている方だと思っていますが、世界的に見ても日本人が作るVimプラグインは品質が良く、Vim scriptに関する知識も高いと思っています。もちろんこれはLingrでのvimrc読書会などの影響も大きいと思っています。

先日、Vimプラグイン開発者の中でOmnisharpというプラグインが話題になりました。

■Omnisharp（https://github.com/nosami/Omnisharp）

このプラグインはVimでC#のコード補完ができるプラグインで、これまでC#のコード補完ができるプラグインがなかったこともあってか、LingrのVim部屋で一気に話題となりました。作者は日本人ではありませんが、日本のVimプラグイン作者が怒涛のようなプルリクエストを送り、マージされ改良されていきました。

筆者がこのプラグインを見つけたころは、Vimからなんとなく C#のコード補完ができる程度のプラグインでしたが、ちょっと見ない間に一気に使える代物になりました。現在ではUIのなしのC#を書くならVisualStudiは使わなくなったという人もいるようです。

昨今のVim事情

Vim 7.3のパッチレベルはこれまで3桁までしかリリースされないであろうと予測されてきました。そしてVimの作者であるBram Moolenaar氏も1,000行くまでには次のバージョンを出すだろうと言ってきました。しかしその言葉は見事に裏切られ、4桁の大台に乗りました。そしてそのころからVimに新しい正規表現エンジンが取り込まれました。

2013年8月10日、Vim7.4がリリースされました。1000を超える修正と新しい正規表現NFAエンジンが取り込まれました。しかしこの新しい正規表現エンジンも取り込まれてまだ日が経ちません。バグも残っているかもしれません。

もしバグを見つけた方がいらっしゃれば、ぜひvim-jpに不具合報告をください。ほかの不具合報告でもかまいません。ユーモア溢れるVimmerが一緒に解決方法を考えてくれるかもしれません。SD

COLUMN Vim温故知新

最近でこそVim scriptでできる可能性は広がりましたが、筆者が使い始めたころのVimは今と比べ物にならないくらい機能が少なく、Vim scriptはとてもバギーでした。

マルチバイトを扱う機能にもバグがあり、今日のVimのようにアプリケーションを作れるレベルではありませんでした。しかしKoRoNさんが正規表現エンジンのマルチバイト化パッチを送り始めたころを期にVimはどんどん生まれ変わり、バージョン6のころから有用なプラグインが現れ始めます。そのころに筆者が書き始めたのがcalendar.vimです。このころはまだVim scriptにListやDictionaryすらなく、さらにはautoloadも使えなかったという状況でした。

そしてバージョン7となり、autoloadが取り入れられたころからVim scriptの成長は加速し、バージョン7.3となった現在ではVim scriptは立派な言語となり、環境を作れるプログラミング言語となりました。最近ではVimを日常起動したままにしている人も多く存在します。

Vimの知名度は昔からありましたが、Vim scriptの成長はバージョン7で一気に広まったと言って良いでしょう。

ynkdirさんやKoRoNさんとは昔存在していたvim-jpというメーリングリスト上で知り合いましたが、そのころのユーザは皆「Vimプラグインはピュア Vim scriptだけで書くもの」といった意識があり、それが今日の「ピュアVim script族」の根源になっているのかもしれません。

その後vim-users.jpというサイトでVimに関するhack記事が数多く公開され、Vimに憧れる人が増え、Vimに関する勉強会も多く開催されるようになりVimの人気がどんどん広がり始めます。

COLUMN 4

XcodeをVimライクにして作業効率を上げる!?

Writer 森 拓也(もり たくや) ㈱ユビキタスエンターテインメント takuya.mori@uei.co.jp

XcodeとVimプラグイン

筆者は開発環境を構築する場合には、必ずVimコマンドが扱えるプラグインをインストールします。現在は、XcodeにVimプラグインをインストールし、iOSアプリケーションを中心に、開発を行っています。Xcode用のVimプラグインはいくつか存在しますが、筆者がお勧めしたいのは**ViEmu**[注1]というプラグインです。以前から、Visual Studioで愛用しているプラグインで、Xcodeも同じプラグインを使っています。ViEmuは、vi/Vimで使えるほとんどのコマンドを扱うことができ、.vimrcの設定もできます(**リスト1**)。有料で少し高額ですが、相応の効果が見込めると思います。

Vim活用ポイント

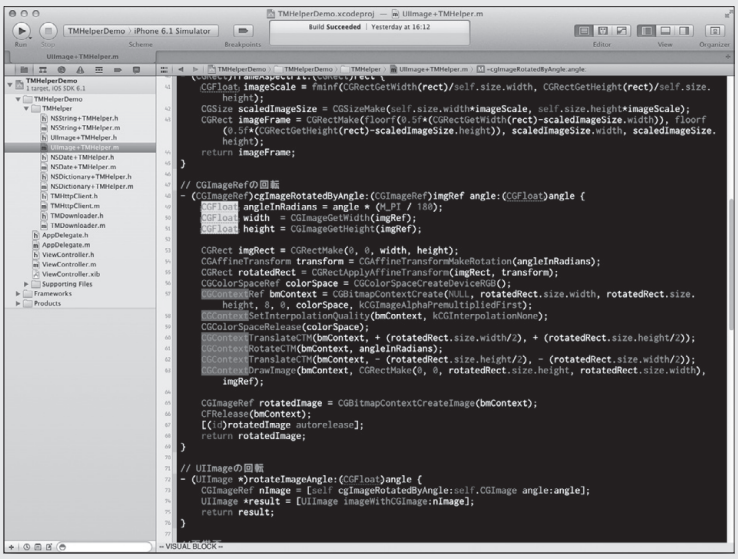
一番便利だと思うコマンドは、矩形選択 Ctrl +V です。Xcode標準でも一応矩形選択はできますが、optionボタンを押しながら狙いを定めて、そのポイントをマウスでドラッグして選択しないといけなく、とてもたいへんで誤動作もしやすいです。そして、そこから置換をする場合にも、検索バーを出してoptionボタンを押しながら隠されたボタンを押す、というわけのわからない手順が必要です。Vimコマンドを使うと、矩形選択からの挿入や置換が一瞬で行え、作業効率が大幅にアップします。とくにプロパティの宣言で、オプションをまとめて追加、置換する場合によく使います(**図1**)。

注1) http://www.viemu.com/

▼リスト1 .vimrc設定例

```
set ignorecase
set smartcase
set incsearch
set hlsearch
set nowrapscan
set noterse

nnoremap n   nzz
nnoremap N   Nzz
nnoremap *   *zz
nnoremap #   #zz
nnoremap g*  g*zz
nnoremap g#  g#zz

nnoremap Y   y$
vnoremap Y   y$
nnoremap p   gp
vnoremap p   gp
nnoremap P   gP
vnoremap P   gP

nnoremap k       gk
nnoremap j       gj
vnoremap k       gk
vnoremap j       gj
nnoremap h       <backspace>
nnoremap l       <space>
nnoremap <Left>  <backspace>
nnoremap <Right> <space>

vnoremap < <gv
vnoremap > >gv

cnoremap <C-v>  <C-R>+
inoremap <C-v>  <C-R>+
nnoremap <Esc><Esc> :noh<CR><Esc>
```

▼図1 矩形選択



その他によく使うコマンドは、カーソル下の単語検索(*)です。Xcodeのショートカットを使うと、文字列を選択して⌘＋Cのあと⌘＋Gと、よく使う割に意外と手順が多いです。Vimでは「*」や「#」だけで済むため、一瞬で検索でき、すぐに次の動作に移ることができます。

筆者は、Interface BuilderやStoryboadなどをほとんど使わずにUIを作成するため、コーティング量が通常より多いです。コーディング量が多いほどVimコマンドの活躍の場が多くなるため作業効率もアップします。とくにiOSのUIを作成する場合は似たコーディングパターンをすることが多いため、検索や置換を使う場面が多いです。

このようなVimコマンドを使うことによって、以前はマウスで行っていた動作をすべてコマンドで補えるので、時間が短縮できます。まさに、かゆいところに手が届く感じの便利さです。また、Vimコマンドを扱ううえで、より作業効率を上げるポイントとしてフォントとキーボード選びも重要です。

フォント

フォントはRictyを使っています。等幅で見やすく、プログラミングに適したフォントがRictyです(**図2**)。導入が若干たいへんですが、相応の効果が見込めると思います。Rictyの良いところは、等幅なので矩形選択がしやすいところや、英数字が見やすいところなどがありますが、一番は日本語が大きめで見やすいところです。ほかの標準で入っているフォントと比べても、日本語がより大きくきれいに表示されるためとても重宝しています。コーディング内のコメントがきれいな日本語フォントで読みやすいと、それだけで気持ちよくコーディングが行え効率もアップします。

▼図2　Rictyフォントによる画面表示

```
//日本語が綺麗！！

//URIエンコード
- (NSString *)encodeAsURIComponent {
    if (!self.length) return @"";

    static const char* characters = "0123456789ABCDEF";
    const char* src = [self UTF8String];
    if (!src) return @"";

    NSUInteger len = [self lengthOfBytesUsingEncoding:NSUTF8Str
    char buf[len*3];
    char* dest = buf;

    for (NSInteger i=len-1; i>=0; --i) {
        unsigned char c = *src++;
        if (('a' <= c && c <= 'z')
            || ('A' <= c && c <= 'Z')
            || ('0' <= c && c <= '9')
            || c == '_'
            || c == '-'
```

▼写真1　Happy Hacking Keyboard Professional 2

キーボード

Vimコマンドを快適に扱うためのキーボード選びは、とても重要です。筆者のお勧めはHappy Hacking Keyboard Professional 2(HHK)です(**写真1**)。HHKはVimで重要なキーであるEscキーとCtrlキーがとても押しやすい位置にあります。Vimを使っているとよくEscキーを意味もなく連打しますが、HHKはEscキーの位置が近いため、ほとんどホームポジションを崩さずに押すことができます。また、通常のキーボードだとAの左隣というベストポジションに、悪名高きCaps Lockが陣取っていますが、HHKはCtrlキーがあり、押しやすいです。HHKとVimはとても相性がよく、ホームポジションを崩すことが少ないため、コーディング速度が向上します。SD

COLUMN 5

男は黙ってVim！

Writer 田中 邦裕（たなか くにひろ） さくらインターネット㈱ 代表取締役社長

viとの苦い出会い

筆者が初めてviに触ったのは20年近く前のことでした。学校の電算機室にあったSunOS 4.xのUNIXサーバ上のviでした。最初に起動した時は編集モードに入ることができず、挙句の果てには終了すらできずに教官に泣きついた……などなど、viとの出会いはあまり良いものではなかったことを思い出します。

このようなこともあって、viのようなコマンドを覚えないといけないエディタではなく、今でいうnanoのようにコマンドを必要としないテキストエディタを、cursesを使って作ろうと頑張っていました。

しかし、その開発をviの上でやっていたものですから、しばらくするうちにviの使い勝手に慣れてしまって、viより使いやすいエディタを作ろうという本来の目的を失ってしまったというエピソードがありました。今ではviのほうが使いやすくなってしまいました。プログラミングやHTMLコーディング、ドキュメント作成から、雑誌原稿執筆に至るまで、viの上で行うようになっています。

vi互換のエディタ jelvisからVimへ

初めて触ったviはSunOS上で動作するオリジナルに近いものでした。しかし、日本語が文字化けしてしまうので、設定の変更やちょっとしたスクリプトを書くぐらいにしか利用していませんでした。そのため、elvisというvi互換エディタに日本語版があると聞き、ノーマルのviからjelvisに乗り換え、純粋なviを使うことはなくなりました。

ちなみに、jelvisを作ったのはIPv6のプロトコルスタック開発でも有名なitojun氏でしたが、四十歳を前に夭折されたことは本当に残念なことでした。

その後、Red Hat系OS（Linux）をメインに触るようになってからは、Vimに乗り換えることになります。多くのLinuxディストリビューションにおいてはVimが標準であり、viからエイリアスされていることが多く、いわばLinux標準エディタと呼んでも過言ではありません。

Vimを使い始めた時は、あくまでviの代わりとして利用していただけにすぎなかったのですが、使っているうちにVimの魅力に取りつかれ、ほかのvi系エディタではストレスを感じるほどになってしまいました。

そんなVimに魅せられて

Vimの魅力は機能性の高さと、設定ファイルで変更できる動作の豊富さにあります。

Vimは**.vimrc**という設定ファイルによってカスタマイズが可能ですが、さまざまなブログ記事などを見ていると、チューンナップされた.vimrcが数多く存在しているのがわかります。筆者も、Vimを使い始めてすぐくらいのときに――どのサイトかはすでに忘れてしまったのですが――評判となっていた.vimrcをダウンロードし、今でも自分なりに改造しながら使い続けています。

筆者の使用している**.vimrc**は次のサイトにアップしているので、よろしければ使ってみてください。

```
% wget http://tanaka.sakura.ad.jp/vimrc
% mv vimrc ~/.vimrc
```

カラフルな文字表示と自動インデントで可読性アップ!

それでは、筆者の思うVimの魅力について紹介しましょう。まず挙げられるのが、ファイルの拡張子に応じたカラフルな文字表示と自動インデントです。

カラフルな表示をするためには、.vimrcにおいてsyntax onと設定することで可能になります。

```
syntax on
```

図1に示したのは、test.phpという名前のファイルを編集しているところです。PHPコードが埋め込まれたHTMLソースです。本誌の誌面が白黒ではわかりにくいかもしれませんが、HTMLタグ、JavaScript、PHPコードが、それぞれのシンタックスに基づいて色分けされ、たとえば関数であれば桃色、コメントであれば赤色といったように色が付けられています。

また、インデントが自動で付与されるのも便利です。たとえば、関数のブロック内(例示している**図1**の場合では9行目や18行目)において改行を入力したり、**o**を押して挿入モードにしたりすると、次の行は自動でインデントされた状態となり、わかりやすくインデントを行うことができます。

このように、可読性を高めてミスの少ないコードを書くために便利な機能が備わっているのが魅力といえます。

ちなみに、WindowsやMacintoshなどの端末からサーバに接続しているときは、クライアント側からペーストすることもあるでしょう。しかし、インデントがすでにされているテキストをペーストすると、**図2**のようにVim側でもインデントをしようとして、表示がズレてしまいます。このようなときに便利なのが**:a**もしくは**:i**であり、コマンドを入力して改行すると文字列の入力モードとなり、入力後に Ctrl + D で終了させると、入力された文字列がそのままカーソル位置に挿入されます。

```
:a
挿入する文字列
^D (Ctrl+D)
```

UNDOとREDOの技

これはVim独自の機能ではありませんが、自動インデントとクライアントからのペーストを利用するときには押さえておきたいコマンドです。

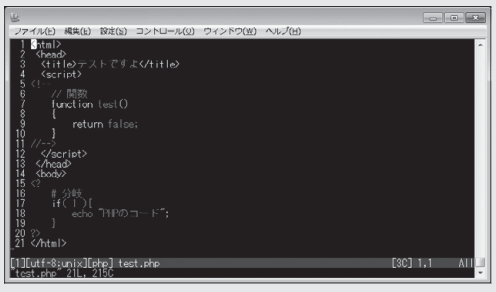
次に魅力として挙げたいのが、UNDO/REDOを何回もできることです。通常のviであればコマンドモードで**u**を入力するとUNDO動作を行い、もう一度入力するとREDO動作となり、**u**を入力するたびにUNDOとREDOが繰り返されます。

しかし、Vimであればあらかじめ設定された値を上限に、何度でもUNDOやREDOを行うことができます。ちなみに、UNDOが**u**なのは変わりませんが、REDOする場合は Ctrl + R を入力して行います。

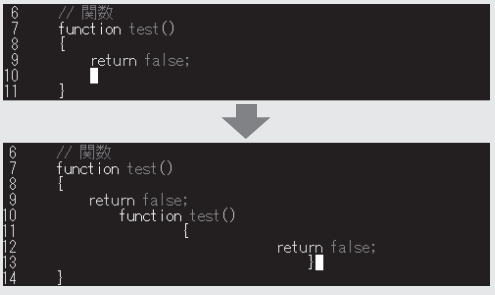
UNDOできる数の上限は、**.vimrc**においてundolevelsを設定することで可能です。

```
set undolevels=1000
```

▼図1 HTMLファイルを編集するときに色分けがされている



▼図2　インデントのずれを解消する技



マッピングとスクリプト機能の技

ここまでは、あらかじめVimに備わっている機能面の話でしたが、Vimを自分流にできる代表的なものが、マッピング機能とスクリプト機能です。

たとえば、次のように大文字のAを**:a**にマッピングするように.vimrcを設定しておけば、Shift + A を押すだけで**:a** Enter が実行され、テキストの貼り付けを行うモードへ移行されます。

```
map A :a<CR>
```

また、スクリプトと併用することで、文字を入力するだけで一連のコマンドを実行することもできます。たとえば、筆者の環境では大文字のRを入力するだけで、編集中のプログラムを実行できるようにしています。

実際に.vimrcへ設定しているのが**リスト1**の内容で、大文字Rを入力することで**:CommandRun**が実行され、保存をした後でファイルの種別に応じたコマンドの実行が行われます。

ちなみに、Rは文字の置換のコマンドですが、使用頻度が低いためコマンド実行に割り当てることにしています。

このほかにも、よく利用するコマンドについてはマッピングや、スクリプトによる定義を行うことで、毎回の入力を簡略化することが可能です。

▼リスト1　.vimrc設定の抜き出し

```
" スクリプトの実行
if has("autocmd")
  command! CommandRun call s:CommandRun()
  map R :CommandRun<CR>
  function! s:CommandRun()
    :w
    if &filetype == "php"
      :!php %
    elseif &filetype == "perl"
      :!perl %
    elseif &filetype == "ruby"
      :!ruby %
    endif
  endfunction
endif
```

文字コード設定の技

最後に触れておきたいのが文字コードの設定です。

VimではUTF-8をはじめとして多バイト文字を扱えるようになっています。しかし、文字コードの判定に失敗することも少なくありません。このような場合には、**:e**を使って文字コードを変更することが可能です。

たとえば、読み込んだファイルの文字コードをUTF-8に変更したければ、以下のように入力します。

```
:e ++enc=utf-8
```

ちなみに、私の使用している**.vimrc**ではnmapで次のように , + U で文字コードをutf-8に変更できるようにしています。そのほかにも、 , + E でEUC、 , + S でShift-JISに変更できて、非常に便利です。

```
nmap ,U :set encoding=utf-8<CR>
nmap ,E :set encoding=euc-jp<CR>
nmap ,S :set encoding=cp932<CR>
```

.vimrcファイルは秘伝のタレだ!

ここまで、筆者がいつも便利だと思っているVimの機能について紹介してきました。

今回のコラムを執筆するなかで、あらためてどれがviのオリジナル機能で、どれがVimで加えられた機能なのか、判断に迷うことも少なくありませんでした。しかし、**.vimrc**におけるマッピングやスクリプトなどの機能についてはVimによってもたらされた素晴らしい機能であることは間違いありません。

老舗のタレのように、誰かからのれん分けされた**.vimrc**を自分の使いやすいように育て続けてみてはどうでしょうか。本稿に触発されて、普通のviでは不便だと思えるくらいにVimを使いこなしてくれたら望外の喜びです。SD

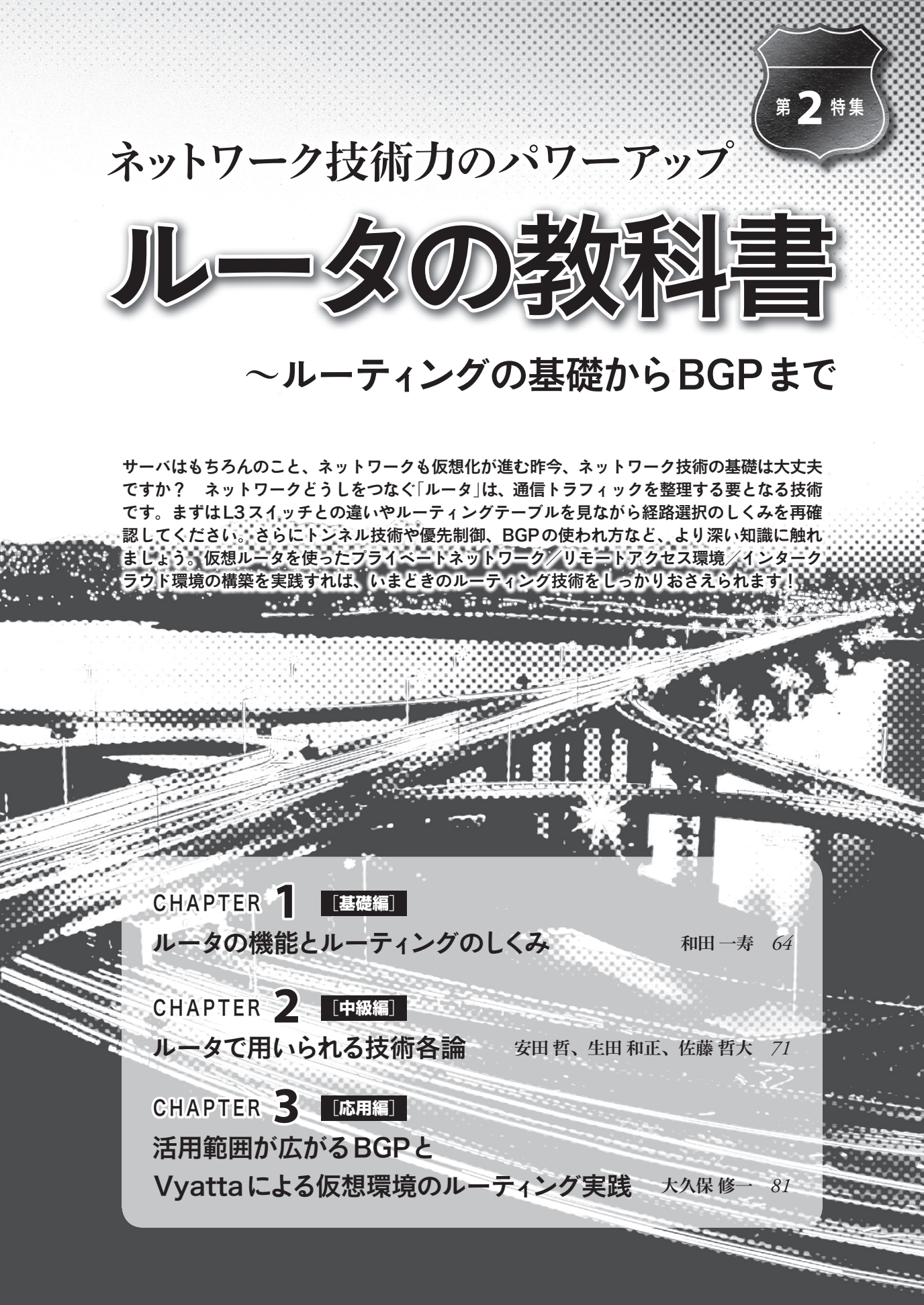
第2特集

ネットワーク技術力のパワーアップ

ルータの教科書

～ルーティングの基礎からBGPまで

サーバはもちろんのこと、ネットワークも仮想化が進む昨今、ネットワーク技術の基礎は大丈夫ですか？　ネットワークどうしをつなぐ「ルータ」は、通信トラフィックを整理する要となる技術です。まずはL3スイッチとの違いやルーティングテーブルを見ながら経路選択のしくみを再確認してください。さらにトンネル技術や優先制御、BGPの使われ方など、より深い知識に触れましょう。仮想ルータを使ったプライベートネットワーク／リモートアクセス環境／インタークラウド環境の構築を実践すれば、いまどきのルーティング技術をしっかりおさえられます！



CHAPTER 1

ルータの機能とルーティングのしくみ

［基礎編］

和田 一寿（わだ かずとし）
シスコシステムズ合同会社

ルータに対して「インターネットに接続するもの」「パケットを転送するもの」のように漠然としたイメージを持っている方が多いと思います。本章ではL3スイッチと比較したり、ルーティングテーブルの中を見たりしながら、ルータの役割やルーティングのしくみを解説します。ルータの理解をより明確なものとしましょう。

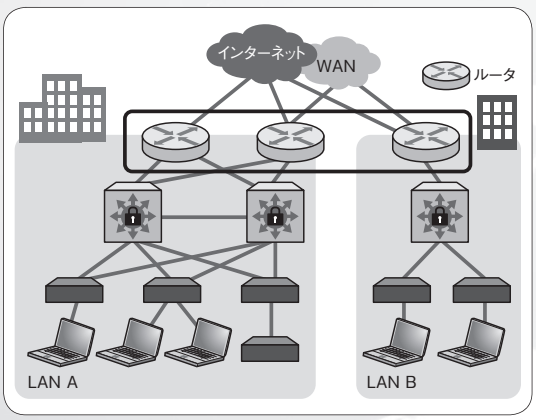
ルータとは

WAN／インターネット接続に使われている

まず、ルータがどのように利用されているかについて説明したいと思います。一例として、ルータはLAN（Local Area Network）とWAN（Wide Area Network）やインターネットとの接続に利用されています。自宅でルータを利用して、パソコンをインターネットに接続することはもはや常識ですので、これは容易にイメージがつくかと思います。

図1は企業ネットワークをシンプルに図示したものです。2つのLANがルータを介してWANやインターネットに接続されています。見方によっては、ルータがWAN／インターネットとLANの境界となっています。

▼図1　ルータを介してWAN／インターネットに接続



多様なインターフェース、プロトコル、機能に対応

WAN／インターネットのサービスは世の中にいろいろとあり、そのサービスごとに接続するためのインターフェース、プロトコルはさまざまです。ルータはそのような多種多様なサービスに接続できるように多様なインターフェース、プロトコルに対応しています。

また、NAT（Network Address Translation）[注1]、ファイアウォール、拠点間を安全に接続するVPN（Virtual Private Network）[注2]やQoS（Quality of Service）[注3]などのさまざまな付加機能を備えている場合が多く、これらはWAN／インターネットへの接続と併用することが多い機能です。これらの機能は、専用機器を別で用意するケースもありますが、1台のルータでまとめて実装

注1） IPアドレスを変換する機能。1つのグローバルIPアドレスで複数のホストを接続する（グローバルIPアドレスとプライベートアドレスの変換）といった用途がよく見られる。家庭で利用されているルータもこの機能を持っているので、複数端末を接続することが可能となっている。

注2） 共有ネットワーク（WANやインターネット）上で、仮想的にプライベートネットワーク間を接続する機能のことで、仮想の専用線を共有ネットワーク上に展開するイメージ。個別に専用線を利用するよりもコストを低く抑えることが可能なので、よく使われる。

注3） ネットワーク上で伝達されるパケットの扱いに差をつけたり、帯域を制御したりすること。たとえば、音声系や動画系などのパケットを優先して処理するといったこと。

する場合も多々あります。

このような理由から、ルータはWAN／インターネットに接続するためのデバイスとして利用され、ルータがLANと、WAN／インターネットを接続することにより、世界中に広がるネットワークが形成されています。

L3スイッチとの違い

このようにルータの話をしていると「L3スイッチと何が違うのか？」といった質問をいただくことが多いです。みなさんの中にも同じような疑問をお持ちの方は多いと思います。ここではこの「よくある疑問」について触れたいと思います。

ソフトウェアベースか ハードウェアベースか

ルータとL3スイッチとは何が違うのでしょうか？　実は「これ！」という定義はないのですが、一般論として**表1**のような違いがあります。

ルータのほうが対応している機能、プロトコルが多く、拡張性が高いです。ソフトウェアベースの製品ですので、後々のアップグレードで機能が拡張／強化されることも多いです。L3スイッチはやはりスイッチですのでポート数が多く、ハードウェアでの作り込みがなされているので処理速度はルータを上回ることが多いです。反面、L3スイッチの機能や拡張性は限定的となっています。

これらの差異は、元来マルチプロトコルに対応するための「ルータ」、ネットワーク機器（パソコン、サーバ、プリンタなど）を相互に接続するための「スイッチ」、という背景から生じています。もう少し主観を交えて踏み込むと、ソフトウェアベースでさまざまな機能やインターフェースに対応し、将来的な拡張性を持たせている「ルータ」。対して、特定処理に特化したハードウェア[注4]をベースに高速なパケット処理を行うが、対応機能数や拡張性は絞っている「L3スイッチ」というのが大きな違いです。

ただし、**表1**は実情と照らし合わせると少し過剰な表現になっています。とくにエンタープライズ、サービス事業者規模で導入する高価な製品は、ルータであってもハードウェア処理が可能な製品もあれば、L3スイッチだとしてもルータのように多機能な製品もあります。高価な製品になればなるほど、処理速度や対応機能数の違いはなくなっている、というのが実情だと思います。逆に、比較的低価格な製品においては、まだまだ**表1**のような差異は残っています。

また、ルータに限ったことではありませんが、最近ではソフトウェア処理のベースとなる汎用CPUの性能が向上していることもあり、ソフトウェアベース製品のパフォーマンスが向上しています。これによりハードウェアベースから、ソフトウェアベースへの転換を図る製品も出てきています[注5]。

注4） 一般にASIC（Application Specific Integrated Circuit）と呼ばれていて、特定用途向けに設計された電子部品のこと。特定用途においては動作が速く、低コストといったメリットがある一方、汎用性がない、修正が難しい、開発コストが高いなどのデメリットがある。

注5） この辺りの最新動向は中級編でも触れています。

▼表1　ルータとL3スイッチの違い

項目	ルータ	L3スイッチ
アーキテクチャ	ソフトウェア処理がメイン	ハードウェア処理がメイン
ポート数	少	多
処理速度	低	高
対応機能数	多	少
対応インターフェースの種類	多	少
拡張性	多	少

ルータはネットワークとネットワークをつなぐもの

ルータは、ユーザネットワークとWAN／インターネットを接続するものと書きましたが、見方を変えればWANやインターネットも1つの大きなネットワークですので、ネットワークとネットワークを接続しているとも言えます。そして、これこそがルータの本質的な役割です。

ルーティングとは

適切なネットワークへパケットを転送

具体的にルータはどういった処理を行っているのかを説明します。**図2**のようなネットワークで考えてみます。

ホストAがホストDにパケットを送信するとき、ルータAがホストD宛のパケットをEthernet0からEthernet1に転送し、192.168.2.0/24ネットワークに届けます(**図2-①**)。

また、ホストFがホストCにパケットを送信するとき、ルータAはホストC宛のパケットをEthernet1からEthernet0へと転送し、192.168.1.0/24ネットワークに届けます(**図2-②**)。

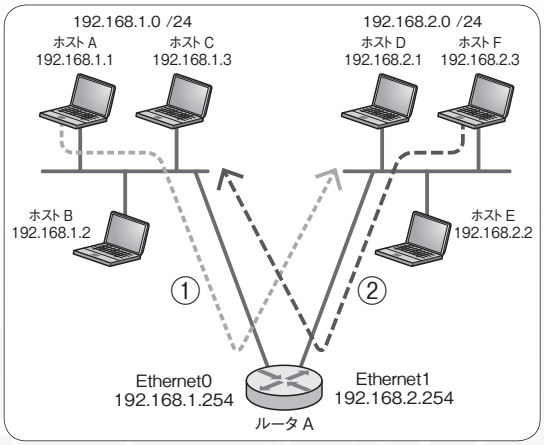
このとき、ルータは各パケットの宛先を読み取り、その宛先から、どのネットワークへと転送すれば良いのかを判断します。この処理が「ルーティング」と呼ばれるもので、パケットの転送先を判断する情報を「ルーティングテーブル」といいます。

図2のルータAが持つルーティングテーブルは、**図3**です(show ip routeコマンドの出力)。

❶と❷に注目してください。先の例で、ホストD(192.168.2.1)宛のパケットを受け取ったルータは❷を参照し、転送先がEthernet1だと判断し、パケットを転送します。同様に、ホストC(192.168.1.3)宛の場合には❶を参照して、Ethernet0に転送します。

以上がルーティングとルーティングテーブル

▼図2　ルータで接続されている2つのネットワーク



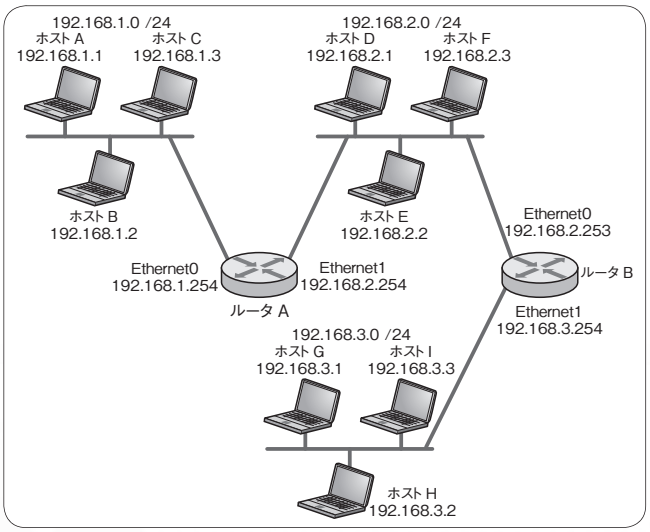
▼図3　図2のルータAのルーティングテーブル

```
RouterA#show ip route
Codes: L - local, C - connected, S - static, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2
       i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-IS level-2
       ia - IS-IS inter area, * - candidate default, U - per-user static route
       o - ODR, P - periodic downloaded static route, + - replicated route

Gateway of last resort is not set

      192.168.1.0/24 is variably subnetted, 2 subnets, 2 masks
C        192.168.1.0/24 is directly connected, Ethernet0  ←❶
L        192.168.1.254/32 is directly connected, Ethernet0
      192.168.2.0/24 is variably subnetted, 2 subnets, 2 masks
C        192.168.2.0/24 is directly connected, Ethernet1  ←❷
L        192.168.2.254/32 is directly connected, Ethernet1
```

の基本的なしくみになりますが、意外にシンプルだったのではないでしょうか？　次は、接続ネットワークを増やしてみます。

先ほどのネットワークにルータBと192.168.3.0/24ネットワークを追加しました(**図4**)。このネットワークで、ホストAからホストHへとパケットを送信する場合を考えてみましょう。ルータAはEthernet1からパケットを転送します。次にルータBがそのパケットをEthernet0で受け取り、Ethernet1からパケットを転送します。このときのルータAのルーティングテーブルは**図5**です。

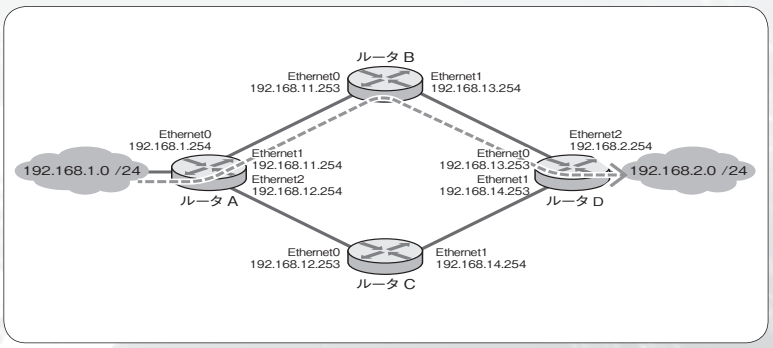
▼図4　2つのルータで接続されている3つのネットワーク



▼図5　図4のルータAのルーティングテーブル

```
RouterA#show ip route
(中略　前半は図3と同じ)
Gateway of last resort is not set

      192.168.1.0/24 is variably subnetted, 2 subnets, 2 masks
C        192.168.1.0/24 is directly connected, Ethernet0
L        192.168.1.254/32 is directly connected, Ethernet0
      192.168.2.0/24 is variably subnetted, 2 subnets, 2 masks
C        192.168.2.0/24 is directly connected, Ethernet1
L        192.168.2.254/32 is directly connected, Ethernet1
S     192.168.3.0/24 [1/0] via 192.168.2.253   ←❸
```

▼図6　スタティックルーティング



先ほどと違い、❸が増えているのがわかります(なぜ増えたのかはのちほど説明します)。ルータAは直接192.168.3.0/24ネットワークに接続されていませんが、192.168.3.0/24ネットワークが宛先のパケットは192.168.2.253、すなわちルータB宛に転送すべきだということをこのルーティングテーブルから判断します。これにより、ルータAは直接接続されていない192.168.3.0/24ネットワークへの適切な転送先を判断できます。

では、なぜ直接接続されていないネットワークへの適切な転送先を知っていたのでしょうか？　管理者がルーティングテーブルを構築する方法には、大きく分けて2つの方法があります。1つはスタティックルーティング、もう1つはダイナミックルーティングと呼ばれるものです。

スタティックとダイナミック

スタティックルーティングとは、手動でルーティング情報を設定することです。「このネットワーク／ホストに対しては、このインターフェースから、もしくはこのネットワーク宛にパケットを転送する」という情報を管理者がルータに設定します。**図5**もこのスタティック

ルートを設定して❸の情報を追加していました。

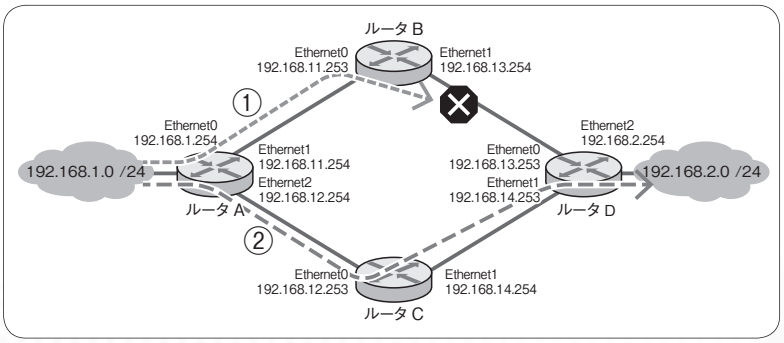
たとえば**図6**で、192.168.1.0/24ネットワークから192.168.2.0/24宛のパケットをルータB経由で転送したい場合、ルータA、B、Dに192.168.2.0/24宛のパケットはルータB経由で転送するように手動で設定します。

スタティックルーティングの長所は管理者の意図するパケット転送を容易に実現できることと、ルータへの負荷が軽いことです。短所としては、運用管理負荷が大きいことです。たとえば、どこかのネットワークが変更されたとすると、その変更に伴いネットワーク内のルータが持つスタティックルーティングの設定を変更しなければなりません。管理デバイス／ネットワーク数が1桁程度であれば問題ありませんが、数十、数百となってくると現実的には不可能です。

また、ネットワーク内に障害が生じた場合、自動でルーティング情報が切り替わらないので、管理者が対策するまで障害が続いてしまいます。

図7の例で、ルータB－D間のリンクに障害が発生したとします。このとき、ルータC経由でネットワークAとネットワークBはつながっているのですが、ルータAは設定によりルータBにパケットを転送してしまいます(**図7-**①)。

▼図7　ネットワーク障害時におけるルータC経由のパケット転送



▼図8　図7のルータAのルーティングテーブル(障害発生前)

```
RouterA#show ip route
Codes: L - local, C - connected, S - static, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2
       i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-IS level-2
       ia - IS-IS inter area, * - candidate default, U - per-user static route
       o - ODR, P - periodic downloaded static route, + - replicated route

Gateway of last resort is not set

      192.168.1.0/24 is variably subnetted, 2 subnets, 2 masks
C        192.168.1.0/24 is directly connected, Ethernet0
L        192.168.1.254/32 is directly connected, Ethernet0
O     192.168.2.0/24 [110/30] via 192.168.12.253, 00:00:31, Ethernet2  ←❹
                     [110/30] via 192.168.11.253, 00:00:31, Ethernet1  ←❺
      192.168.11.0/24 is variably subnetted, 2 subnets, 2 masks
C        192.168.11.0/24 is directly connected, Ethernet1
L        192.168.11.254/32 is directly connected, Ethernet1
      192.168.12.0/24 is variably subnetted, 2 subnets, 2 masks
C        192.168.12.0/24 is directly connected, Ethernet2
L        192.168.12.254/32 is directly connected, Ethernet2
O     192.168.13.0/24 [110/20] via 192.168.11.253, 00:00:41, Ethernet1
O     192.168.14.0/24 [110/20] via 192.168.12.253, 00:01:06, Ethernet2
```

こういった事態を回避するために、ダイナミックルーティングがあります。

ダイナミックルーティングでは、ルータ同士がルーティング情報を交換し、自身のルーティングテーブルを更新します。**図7**の例で、ルータB－D間のリンクに障害が発生した場合でも、ダイナミックルーティングを利用していればルータC経由の経路を自動で選択します(**図7-②**)。実際にダイナミックルーティングを利用しているときの、ルータAのルーティングテーブルを見てみましょう(**図8**)。

先頭に「O」[注6]と付いているのがダイナミックルーティングで学習した経路です。Oがついている4つの経路の内、❹と❺に注目してください。192.168.2.0/24ネットワークに対して、ルータB、C経由どちらの経路も登録されているのがわかります。

では、ルータB－D間のリンクに障害を発生させてみましょう。すると、ルーティングテーブルは**図9**のようになります。

先ほどの個所を見ると、ルータB経由の経路がなくなって、ルータC経由の経路(❻)だけになっているのがわかります。この経路を利用することで、部分的には障害が生じてもネットワークとしての疎通性は保つことが可能になります。

このように、ダイナミックルーティングを利用すると、運用管理面や障害時の代替経路などの設定が非常に容易になります。実情としても、ダイナミックルーティングを利用するネットワークが圧倒的に多いです。ただし、スタティックルーティングは検証時や、限定的な用途でダイナミックルーティングと併用して利用されることも多いので、どちらも理解して、設定できることが重要です。とくに小規模な検証を行う際には、不要なパケットを発生させたくないことが多々あるので、スタティックルーティングを利用することが多いです。このあたりは個人のお好みでもありますが。

ルーティングプロトコル

ここまでの説明で、どのようにルータが経路情報を交換しているか気になると思います。ダイナミックルーティングでは経路情報を交換するためにプロトコルを利用しており、それらは「ルーティングプロトコル」と呼ばれます。ルーティングプロトコルにはいくつかの種類があり、そのネットワークの種類、管理／設計コンセプトによって適切なプロトコルを選択します。

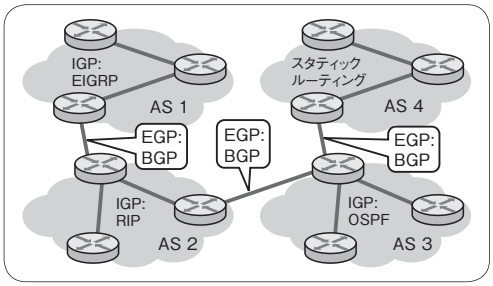
ルーティングプロトコルには大別して、EGP(Exterior Gateway Protocol)とIGP(Interior Gateway Protocol)の2つがあります(**図10**)。

注6) O：OSPFの頭文字。OSPFとは後述するルーティングプロトコルの一種。この頭文字はルートの由来を示すもの。たとえばSはStaticでスタティックルート、Cはテーブルを保持しているデバイスに接続されている経路など。

▼図9　図7のルータAのルーティングテーブル(ルータB－D間のリンクダウン)

```
RouterA#show ip route
(中略　前半は図8と同じ)
Gateway of last resort is not set

      192.168.1.0/24 is variably subnetted, 2 subnets, 2 masks
C        192.168.1.0/24 is directly connected, Ethernet0
L        192.168.1.254/32 is directly connected, Ethernet0
O     192.168.2.0/24 [110/30] via 192.168.12.253, 00:11:38, Ethernet2  ←❻
      192.168.11.0/24 is variably subnetted, 2 subnets, 2 masks
C        192.168.11.0/24 is directly connected, Ethernet1
L        192.168.11.254/32 is directly connected, Ethernet1
      192.168.12.0/24 is variably subnetted, 2 subnets, 2 masks
C        192.168.12.0/24 is directly connected, Ethernet2
L        192.168.12.254/32 is directly connected, Ethernet2
O     192.168.13.0/24 [110/30] via 192.168.12.253, 00:00:11, Ethernet2
O     192.168.14.0/24 [110/20] via 192.168.12.253, 00:12:13, Ethernet2
```

▼図10　EGPとIGPの利用イメージ



EGPはAS(Autonomous System)[注7]間の経路情報を交換するためのプロトコルで、現在はBGPの一択ですが、昔はEGP[注8]というプロトコルが利用されていました。IGPはAS内部で利用されるプロトコルで、RIP、OSPF、EIGRPなどがあります。現在は、OSPFやEIGRPを採用しているネットワークが多いです。

・BGP(Border Gateway Protocol)

AS間で利用されるプロトコルで、大規模ネットワーク(数十万の経路数)に対応するように設計されています。そのため、BGPを動かしているルータは比較的高価な製品になることが多いです。とくにサービス事業者がサービス用途で利用する製品の場合、数百万、数千万クラスだったりします。BGPを仕事で扱う方々もサービス事業者関係の方が多いです

・RIP(Routing Information Protocol)

歴史のあるプロトコルで、実装が容易ですのでよく利用されていました。経路選択には、ホップ数(目的ネットワークまでのルータ数)を基準とし、ディスタンスベクタ型プロトコルと呼ばれています。今では、後述のOSPFやEIGRPの採用が増えていますが、小規模ネットワークでは今でも利用しているところもあります

・OSPF(Open Shortest Path First)

RIPの後継として開発され、RIPでは実現できない高速な経路切り替えや、経路の帯域幅を考慮した経路選択などの実現が可能で、マルチベンダも可能です。大規模ネットワークに対応できるプロトコルで、リンクステート型プロトコルと呼ばれています。リンクステート型では、ディスタンスベクタ型より正確なネットワークトポロジを把握するしくみがあり、それを基にルーティングテーブルを構築します

・EIGRP(Enhanced Interior Gateway Routing Protocol)

Cisco独自のIGRP(Interior Gateway Routing Protocol)を改良したプロトコルで、経路選択を複数要素で決定し、非常に高速な経路切り替えを実現できます。リンクステートとディスタンスベクタの性質を兼ね備えているため、ハイブリッド型や拡張ディスタンスベクタ型と呼ばれています。小〜大規模ネットワークに対応でき、ハブ&スポーク型の大規模ネットワークに向いています。Cisco独自ですので、Ciscoオンリーのネットワーク内で実装されていることが多いです[注9]

最後に

本稿が、ルータの役割や現状のポジショニングについて少しでも理解する一助になれば幸いです。「ルータはネットワークを接続するもの」と一言にいっても実際には接続性を担保してルーティングだけをやっているわけではなく、さまざまな機能を併用しています。次の中級編では、ルータでしか実現できないサービスや機能について紹介します。SD

注7) ある一意のポリシーに基づき、管理されているネットワーク(自律システム)のこと。ISP、エンタープライズ企業、公共機関のような組織がASに該当。インターネットはAS同士が接続されることにより形成されている。

注8) EGP/IGPという区分けでの総称ではなく、単一プロトコルの名称。EGPという区分の中に、EGPとBGPというプロトコルがある。

注9) ここまで読んで、ルーティングプロトコルがどのように動作しているのか気になった方がおられると思いますが、本稿ではあえて触れませんでした。プロトコル詳細や設定については、書籍やネット上にある数々のサイトでかなり詳細に解説されているのでそちらをご参照ください。

CHAPTER 2

ルータで用いられる技術各論

［中級編］

安田 哲（やすだ さとし）
生田 和正（いくた かずまさ）
佐藤 哲大（さとう てつひろ）
シスコシステムズ合同会社

ルータには単純なルーティング以外にもさまざまな機能があります。中級編である本章では、トンネル技術、トラフィックの優先制御／可視化、次世代ルーティングアーキテクチャなど、ルータの特徴的な技術項目について解説します。

仮想的な経路で柔軟なネットワークを実現するトンネル技術

ルータは外部境界

歴史的に、ルータは多種多様なネットワークをつなぐための（物理層を含む）プロトコル変換装置として発展してきました。つまり外のネットワークを中と接続するためのゲートウェイの役割を担っていたのです。

時代は変わり、EthernetとIPに世界制覇された今となっては、プロトコル変換装置としての意味は廃れました。それでも外部との境界はルータが担っています。アドレス変換を行うNAT（Network Address Translation）やセキュリティ（ファイアウォール）機能、さらにここで紹介するトンネルの取り扱いなどです。

通信におけるトンネル技術

道路や鉄道のトンネルは、その外側の構造（山の中や川底の下など）を気にすることなく、入口から入って出口に到着します。

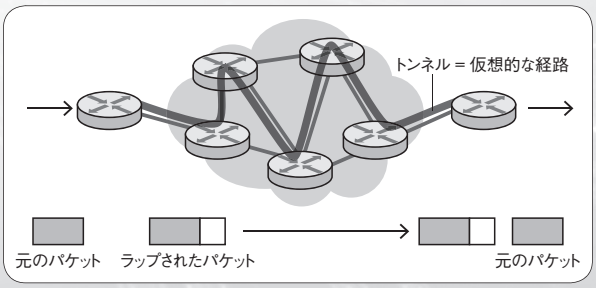
通信の世界では、物理的なネットワーク上に配置した仮想的な経路をトンネルと呼び、トンネルを形成することをトンネリングといいます。この仮想的な経路を通過するパケットは、物理的な接続構成やトポロジーを意識することなく、入口から入って出口に出ていきます。この仮想的な経路であるトンネルの入口と出口はルータの役目です（**図1**）[注1]。

トンネルの入口ルータは、パケット自体を別のプロトコルにラップ[注2]します。これにより元のパケットはトンネル外の物理ネットワーク上では透過的に転送されます。トンネルの出口ルータでは、ラップした部分を取り除き、元のパケットとして扱います。元のデータを別のプロトコ

注1） トンネリングを行うのはルータだけではありません。スイッチでもVLANタグを多重化する方式（IEEE802.1ad）は一種のトンネリング技法ですし、SSHによるポートフォワーディングもSSHで作った暗号化トンネル上で別の通信プロトコルを利用するトンネリング技術です。ただし、ルータが扱うトンネリングのほうが多種多様です。

注2） 別のプロトコルヘッダ（場合によってはフッターも）を付与すること。

▼**図1　トンネリング**



ルでラップする点がカプセルに物を入れるように見えることからカプセル化とも言います[注3]。

トンネルは、インターネットを含む外部のネットワーク上に仮想的なネットワークを構築できるため広範囲に使われています。用途に応じてさまざまな種類がありますが、ここでは2つの例を紹介します。

PPPoE

NTT東西が提供しているフレッツなどのISP接続サービスではPPPoE(PPP over Ethernet)というトンネルが使用されています。PPPoEはPPP(Point-to-Point Protocol)をEthernet(IEEE802.3)でラップしています(**図2**)。EthernetもPPPもレイヤ2のプロトコルであり、レイヤ2を多重していますが、その処理は一般的にはルータの仕事です。ところで、なぜこのようなことをするのでしょうか。

PPPは古くはダイヤルアップ接続で利用されていたプロトコルであり、ユーザ認証機能を有しています。ただし、Ethernetでは動作せず、ピア・トゥ・ピアのシリアル回線が必要です。一方でEthernetはLANのプロトコルのため認証機能がなく、ISP接続で使うには不十分でした[注4]。ISP側ではダイヤルアップ接続時代に培ったノウハウも流用できることからPPPoEが採用されたのでしょう。

IPsec

通信の暗号化で用いるIPsecもトンネルの1つです。IPsecには、トランスポートモードとトンネルモードがあり、どちらもパケットの暗号化に対応していますが、ここではトンネルモードを紹介します。トンネルモードでは、もともとの宛先／送信元IPアドレスを含むIPヘッダも含めて暗号化されるため、ルータは自身のIPアドレスとトンネルの出口を宛先にして新たなIPヘッダを付与します[注5](**図3**)。

途中のネットワークは、暗号化されている元のIPパケットの内容はヘッダも含めて気にすることなく、新たに付与されたIPヘッダに従ってルーティングを行います。そのためLANで使用しているプライベートIPアドレスをグローバルIPアドレスにNATすることなく、インターネット経由でのLAN間通信が可能となります。

トンネル利用時の留意点

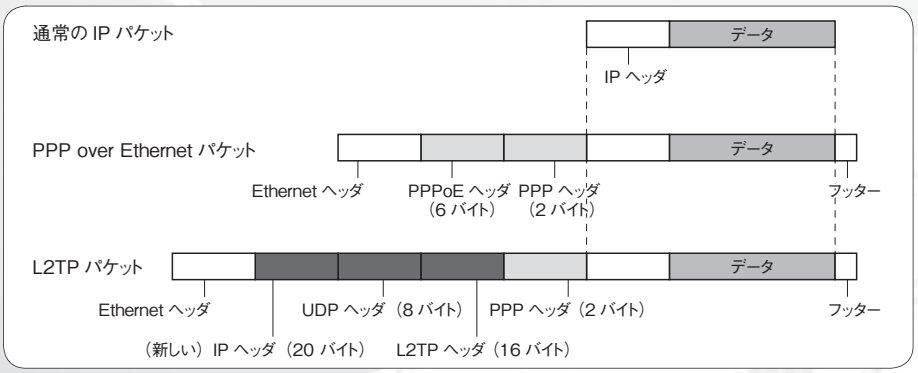
仮想的な経路を使うことで柔軟なネットワークを構築できるトンネルですが、注意すべき点

注3) カプセル化は、異なるレイヤ間でのデータ受け渡し時に広く使われている手法ですが、この場合はトンネルとは呼びません。同一レイヤのヘッダを複数扱う場合にトンネルと呼ぶことが多いです。

注4) Yahoo!BBのように、認証を必要としないADSLサービスではあえてPPPoEを使用せず、Ethernetを直接使用するケースもあります。

注5) IPSecはユニキャストの通信しかサポートしないため、OSPFなどのマルチキャストを使用したルーティングプロトコルが使用できません。そのため、大規模なネットワークにおいてはGRE(Generic Routing Encapsulation)トンネルとの併用するなどの工夫が必要です(図3)。

▼図2 PPPoE/L2TPのパケット



もいくつかあります。ここでは2つ述べておきます。

1つ目はスケーラビリティです。トンネルは入口／出口ルータで1対必要です。したがってn台のルータ間でトンネルを利用する場合、トンネルの数はnの2乗のオーダーで増えてしまい、設定や管理が煩雑になってしまいます[注6]。

2つ目はMTU（Maximum Transmission Unit）です。Ethernetのデータペイロードサイズの上限が1500バイトであるため、通常IPパケットのMTUは1500バイト（IPヘッダ含む）です。前項で見たように、トンネリング時はカプセル化によりパケットのサイズは元のサイズより付与したヘッダ分大きくなり、1500バイトを超えてしまいます。

MTUを超えたパケットは、ルータで分割（フラグメント）[注7]され、2つのパケットに分けられます。その際、ルータの処理負荷の増大や、伝送効率の悪化により、スループットが劣化することがあります。

また、OSやアプリケーションの設定でパケットの分割を禁止している場合[注8]があり、そうなると、カプセル化によりMTUを超えたパケットは、ルータでは分割できず、捨てられてしまうため通信そのものができません。そのためトンネル使用時にはMTUの調整は重要です[注9]。

たとえば、NTTのフレッツサービスにおいては、前述のとおり、PPPoEを使用しています。PPPoEでは、通常のEthernetフレームと比べると6バイト多くなってしまうためMTUを1494バイトとすれば良いように思えますがそれでは不十分です。

PPPoEのデータは、フレッツ網内でL2TP（Layer 2 Tunneling Protocol）という別のトンネリング方式で処理されており、L2TPはIP/UDPでのペイロードとして扱われるため、もともとのIPパケットと比べると、46バイト多くなってしまいます（**図2**）。そのためフレッツサービスではMTUが1454バイトということになります。このようにMTUはネットワーク全体を考慮しなければいけません。

◆ ◆ ◆

トンネリングは、インターネット／IP接続では欠かせない技術であり、データセンター間のレイヤ2延伸技術でも重要な要素として注目を浴びています。その中核を担うのがルータです。ルータはこれからも外部との境界として発展していくことでしょう。

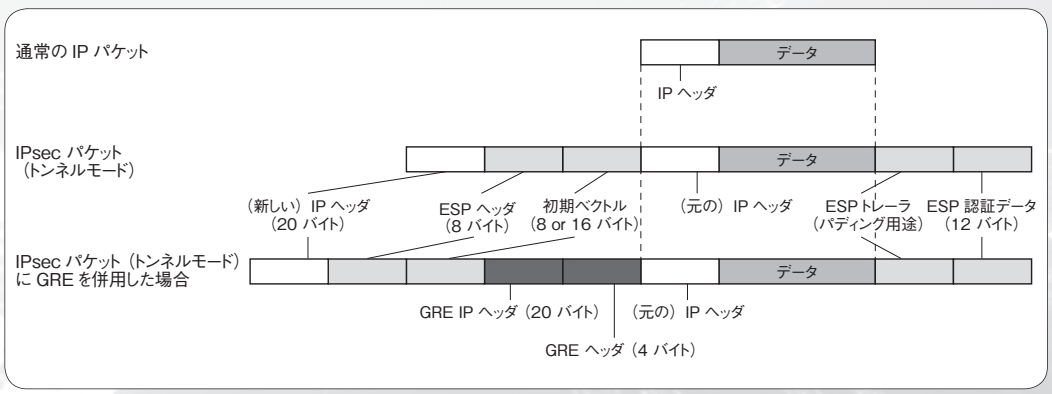
注6） 対策として、オンデマンドでトンネリングを行う、ダイナミックマルチポイントトンネリングという技術もあります。また、本章の後半で紹介しているLISPはこの点も考慮したプロトコルとなっています。

注7） IPv6では経路途中のルータはフラグメントしないという仕様に変わっています。そのためますますMTUが重要となります。

注8） IPv4ヘッダ中のDon't Fragmentビットに1が付与される。

注9） 組込み機器やスマートデバイスではMTUのような細かいチューニングが簡単にできないことも多く、トラブルのもとです。TCPのMSS（Maximum Segment Size）をルータで上書きさせることで回避できる場合があります。

▼図3　IPsec（トンネルモード）のパケット



トラフィックの優先制御と可視化

WAN終端装置としてのルータ

L3スイッチはルータと比べてパケット転送性能は高いものの、ハードウェアの制約上、暗号化や複雑なトラフィック分類／優先制御といった高度な機能は必要最小限であることが多いのが実情です。大規模LANを構築する際には、L3スイッチ／L2スイッチの組み合わせで十分なことがほとんどですが、WANを介してほかの拠点と接続しようとするとルータの出番となります。WAN終端装置という観点では、広域LANサービスや専用線サービス利用時には、「L3スイッチ＋QoSや暗号化を行う専用機」で構成される場合もありますが、インターネットVPNやMPLS-VPN(IP-VPN)では、多くの場合、ルータで構成されています。家庭利用でも、インターネット接続にはルータを利用しているはずです。

業務用途では必須の優先制御

さて、企業用ルータの場合、事業部門の配置や業務内容から大まかに転送対象となるトラフィック種別／パターンやアプリケーションの見当がつきます。WAN接続の帯域が100Mbpsや1Gbpsが一般的になってきたとはいえ、LANはさらに帯域が太くなっていますので、WAN出口では輻輳が発生し得ます。したがって、業務用途ではルータでのQoSは必須です。

トラフィック種別やアプリケーションの優先処理を行うには、事前にトラフィックを識別し、マーキングを行います。具体的には、特定のアクセスリスト(ACL)にマッチするパケットに対してIPヘッダーやEthernetフレームの優先フラグを設定します。マーキングされたパケットは、輻輳個所の装置(おもにルータ)にて、特別なキューで処理されることになり、音声系などの遅延を最小限にとどめつつ、業務系は事前に設定された一定帯域幅を確保するといった優先制御が行われます。

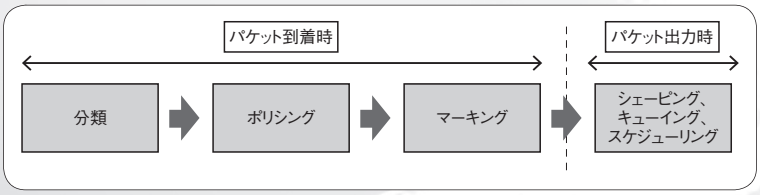
パケットの到着から出力までの処理イメージは**図4**のようになります(あくまでも概念図であり、詳細は機器の実装に依存します)。

マーキングおよび処理に利用される優先フラグの一例として、IPv4ヘッダおよびL2ヘッダのイメージを示します(**図5**)。

最近では、さまざまな要件に応えるため、QoS機能も高度化し複雑になっています。たとえば、トラフィック分類をACLではなく、ルータが備えたDPI(Deep Packet Inspection)機能[注10]を用いて識別したアプリケーションごとにマーキングしたり、多段階のポリシー(階層型QoS)と多くのキュー[注11]を組み合わせるなど、スイッチと比べてソフトウェアベースのルータの強みが活きている部分でもあります。これらの機能は、業務用途では一般的に利用されています。

注10) CiscoではNBAR(Network Based Application Recognition)という機能が該当する。
注11) LANスイッチではハードウェアの制約上キューの数が数個に固定されている場合が多い。

▼図4　パケットの到着から出力までの処理イメージ



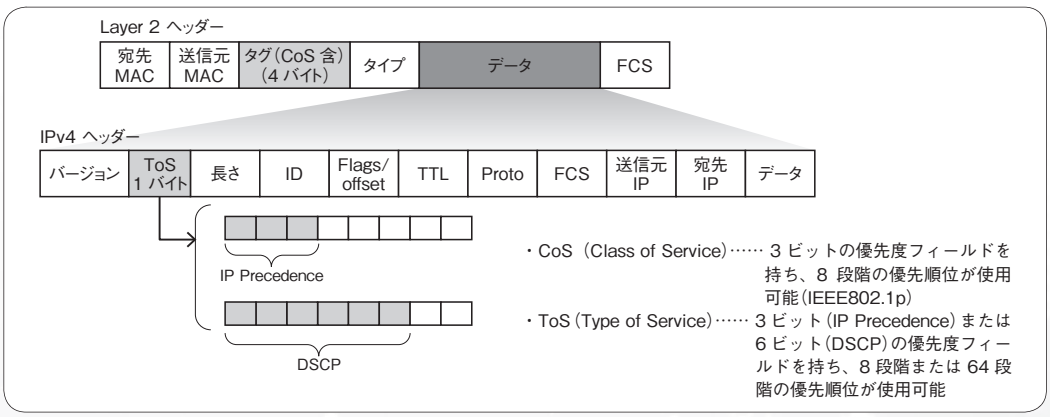
トラフィックの可視化

近年、ビデオ会議アプリケーションや仮想デスクトップなど、企業アプリケーションも変わってきており、ネットワーク設計／運用管理もそれに対応していく必要があります。今では定期的なポーリング、SNMPやSyslogに加えて、「トラフィック可視化」も一般的な要件になりつつあります。通常、平均トラフィック量やピーク時に必要な帯域幅を基準に、必要な回線や機種を選定します。トラフィック可視化によって、無駄なトラフィックが大切なトラフィックを圧迫していないか、無駄なものとしてポリシングして良いのかWAN回線を増速すべきなのか、といった投資判断の材料を得ることができ、経営者にとっても価値のある情報が得られます。

さて、一般的な企業利用で、可視化目的で実際にパケットをすべてキャプチャして中身を分析するわけにもいかないので、ルータやスイッチで設定を追加して利用できるNetFlowやsFlowといった技術（まとめてxFlowと呼ばれています）が広く利用されています。NetFlow、sFlowともにベンダが開発した仕様ですが、古くから仕様が公開されている[注12]ため、特定のベンダにとらわれず各社の製品やOSSに実装され

注12) NetFlowはCiscoSystems社、sFlowはInMon社によって開発された。RFC3176、RFC3954など。

▼図5　マーキングおよび処理に利用される優先フラグの例



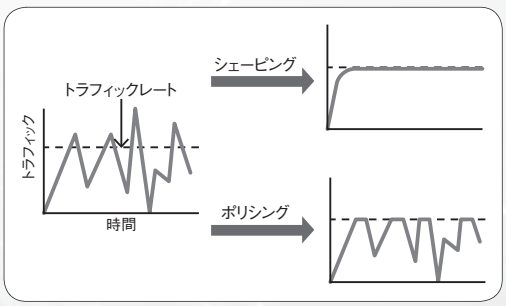
Column

シェーピングとポリシング

出力帯域幅を制限する主な方式として、シェーピングとポリシングの2種類があります。

トラフィックシェーピングでは超過パケットはキューに保持され、一定の時間間隔でスケジュールされてから遅れて伝送されます。一方、トラフィックポリシングではトラフィックレートが最大レートの設定値に達すると、超過トラフィックが廃棄（またはマーキング）されます。シェーピング方式は出力レートが平滑化されますが、ポリシング方式では出力レートはギザギザ状態となります（**図6**）[注13]。

▼図6　シェーピングとポリシングにおける出力レートの違い



注13) 詳細はこちらのURLを参照ください。 http://www.cisco.com/cisco/web/support/JP/100/1008/1008147_policevsshape-j.html

ています。現在では、NetFlow version 9をもとにしたIPFIX(IP Flow Information Export)やパケットサンプリング技術としてPSAMPの標準化が進んでいます。

NetFlowを使ってみよう!

最後に、もし手元にCiscoルータをお持ちの方であれば、手っ取り早く可視化を試せるNetFlow version 5を紹介します。可視化したいルータのインターフェースで、次のコマンドを有効にします。

```
Router(config-if)#ip flow ingress
```

コンソール上での確認コマンドはこちらです。

```
Router#show ip cache flow
```

誌面のスペース上、出力結果を載せられませんが、送信元/宛先IPアドレス、送信元/宛先ポート番号、入出力インターフェースといった情報のペア(フロー)ごとに、パケットカウントが表示されます。また、NetFlow v5に対応したアプリケーション(通称NetFlowコレクタ)は、体験版や安価なものなど広く入手できます[注14]。ルータに次の設定を追加いただき、アプリケーションを立ち上げれば、トラフィック可視化の最初のステップが完了です![注15]

```
Router(config)#ip flow-export
destination <collector-ip> <port-number>
```

企業IT担当者の方は、結果を確認することで、全体的なトラフィックパターンだけでなく、ポート番号からおおよその傾向を把握できる可能性があります。削減対象となる不要なトラフィックが確認できるかもしれませんし、可視化システムを導入していることを周知することで、不要なトラフィックが減るといった抑止効果があるかもしれません。

注14) SevOne、Plixer、Zohoなど、比較的手軽に開始できます。
http://info.sevone.com/terilogy-download.html
http://www.plixer.com/Scrutinizer-Netflow-Sflow/scrutinizer-download-now.html
http://www.manageengine.jp/trial/

注15) 自宅ラックにC892やC1812をお持ちの方も、ぜひ試してみてください! さらに応用は次のURLを参照ください。https://learningnetwork.cisco.com/blogs/jp-networkmanagement

次世代ルーティングアーキテクチャ「LISP」

はじめに

「計算機科学のいかなる問題も他のレベルの間接によって解決できる」とは、サブルーチンの発案者であるデビッド・ホイーラーの言葉です。LISP(Location Identifier Separation Protocol)は、ルーティングアーキテクチャにおけるある問題を解決するために、ある間接を導入しました。結果として、LISPは本来の目的とは異なる問題を解決する手段として利用されるという予期せぬ発展をみせるようになります。

LISPが生まれた背景

2006年10月、Internet Architecture Board主催のRouting & Addressingワークショップにおいて、「インターネット上の経路が爆発的に増加しつつあり、ルーティングスケーラビリティこそが今日のインターネットが直面する最も重要な問題である」と結論されます[注16](**図7**)。インターネット上で利用されるネットワーク機器は、大容量のトラフィックを転送する必要性から、TCAMやSRAMのような転送テーブルに高速にアクセスできる記憶装置を必要とします。しかし、これらは非常に高額で、かつ、大容量化

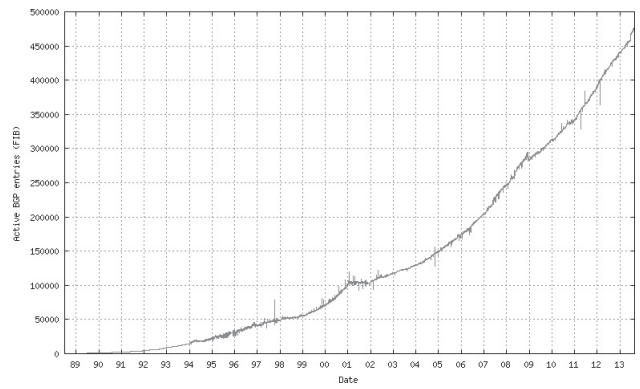
注16) この報告は、RFC4948としてまとめられています。

が困難でした。そのため、このままいくと公共のインフラであるインターネットの維持に膨大なコストがかかり、そのアクセスが非常に高価なものになってしまう恐れがありました。こういった状況を受けて検討されたさまざまな解決方法の1つがLISPです。翌2007年、LISPの最初のドラフトが提案されます。

LISPにおける間接――ID（識別情報）とLocator（位置情報）の分離

現在のインターネットルーティングおよびアドレッシングのしくみにおいて、IPアドレスはデバイスの識別情報と位置情報という異なる2つの役割で、一緒くたに用いられています。LISPはIPアドレスに2つのセマンティックスを導入します。1つは、従来どおりエンドノードに割り当てられるIPアドレスで、エンドポイントID（EID）と呼ばれます。もう1つは、グローバルルーティングシステムを構成するデバイス（おもにルータ）に割り当てられるIPアドレスで、ルーティングロケータ（RLOC）と呼ばれます。

▼図7　インターネット経路数の推移[注17]

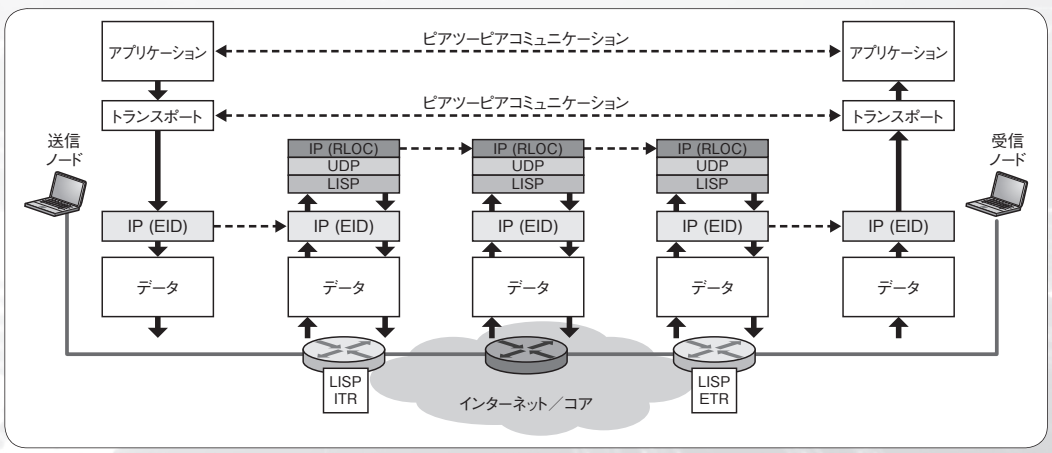


注17）「AS65000 BGP Routing Table Analysis Report　Active BGP entries (FIB)」
http://bgp.potaroo.net/as1221/bgp-active.html

LISPのしくみ

LISPはIPをIPでカプセル化するルータベースのソリューションです。**図8**にあるとおり、送信側のLISPトンネルルータ（ITR：Ingress Tunnel Router）は、EIDのIPをRLOCのIPでLISPカプセル化して、インターネットもしくはコアネットワーク上をトンネル方式で通過させます。宛先側のLISPトンネルルータ（ETR：Egress Tunnel Router）は、LISPカプセル化を解いて、最終の宛先にトラフィックを届けます。コアネットワーク上で中継するネットワーク機器では、RLOCのIPさえ転送できれば良いので、EIDのIPは意識する

▼図8　LISPパケット転送の流れ



必要がありません。そのため、ルータが保持する転送テーブル上の経路数を大幅に減らすことができます。

LISPマッピングサービス

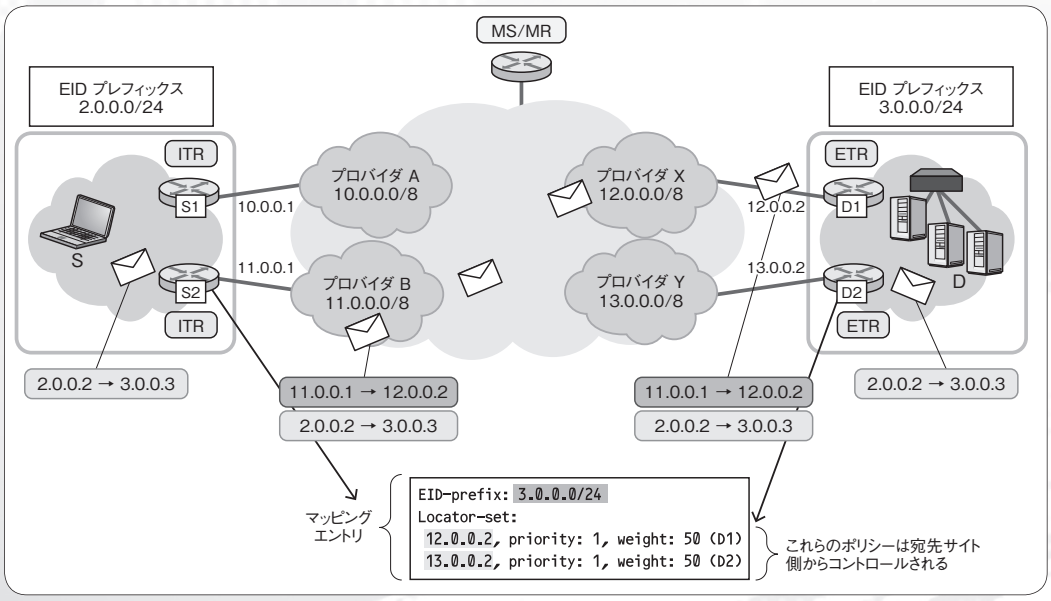
ITRは、EIDをLISPカプセル化するのにRLOCの情報を必要としますが、このEIDからRLOCの解決は、LISPマッピングサービスにより動的になされます。LISPマッピングサービスは、マッピングサーバ(MS：Mapping Server)とマッピングリゾルバ(MR：Mapping Resolver)の2つのインターフェースを介して提供されます。ETRはあらかじめ、管轄するEIDとそれに対応するRLOCのマッピング情報をMSに登録をしておきます。ITRは宛先EIDに対応するRLOCを解決するため、必要に応じてMRにマップリクエストと呼ばれる問い合わせを行います。MRはマップリクエストを受けると、対象のEIDについて登録があったMSにマップリクエストを転送します[注18]。MSはマップリクエストを受けると、対象のEIDについて登録したETRへマップリクエストを転送します。最終的に、ETRはマップリプライとしてEID-RLOCマッピング情報をITRへ返します。ITRでは一度、解決したマッピング情報をキャッシュしておき、後続するパケットの転送には、そのキャッシュされたマッピング情報を利用します。

LISPによるマルチホーミング

EID-RLOCマッピング情報は、同一のEIDに対して複数のRLOCを対応させられます。さらに、これらのマッピング情報には、Priority(優先度)とWeight(重み)を設定できます。宛先のEIDに対して、複数のRLOCが存在するときには、Priorityの高いほうを優先します。また、Priorityが等しいRLOCが複数あるときには、Weightの比率に応じて、複数のRLOCを利用します。たとえば、**図9**において、左サイト(2.0.0.0/24)から右サイト(3.0.0.0/24)へのトラ

注18) MRが問い合わせのEIDを管轄するMSを知るには、LISP ALT(Alternative Logical Topology)やLISP DDT(Delegated Database Tree)などの分散マッピングシステムを必要とします。ここではMRはMSを兼ねていて、EID-RLOCマッピング情報を登録してきたETRを知っていると思ってください。実際、1つの組織でLISPを利用するケースでは、MSにMRを兼ねさせて、ALTやDDTを用いないのが一般的です。

▼図9 LISPマルチホーミング



フィックを転送するには、12.0.0.2と13.0.0.2の2つのRLOCを利用できますが、それぞれ50：50の割合で負荷分散されることになります。LISPを用いることで、高価なWANアクセス回線を複数回線契約している場合に、入力トラフィックを柔軟かつ容易にコントロールして、それらの回線を有効に活用することができます[注19]。

LISPの利用例

ここからは、LISPの利用例を見ていきたいと思います。導入部でも触れたように、IPアドレスに識別情報(EID)と位置情報(RLOC)の2つのセマンティックを導入するというLISPの特徴は、さまざまな応用を可能としました。

IPv6適応

ここまでは暗にIPv4を前提にお話ししてきましたが、実はEID、RLOCはIPv4、IPv6の任意の組み合わせで利用できます。これにより、サイロ化されたIPv6サイト間をIPv4コア上でトンネル方式によって接続できるようになります。それだけでなく、現状、利用者が少ないため、比較的広帯域な利用が期待できるIPv6 WANサービスを利用して、IPv4トラフィックを(必要に応じてIPv6トラフィックも)トンネリングさせることも可能です[注20]。

VPN

LISPのパケットに利用されるLISPヘッダには、24ビットのインスタンスID情報を含むことができます。このインスタンスIDはVRF(Virtual Routing and Forwarding)[注21]の情報とをマッピングできるので、物理的に1台のトンネルルータ上で複数のLISPネットワークを論理的に多重することができます。また、MSおよびMRにおいても、複数のインスタンスに対するLISPマッピングサービスを構築できます。これにより、1つの企業や1つの組織において、共通のWAN上で、部門やシステムごとにVPNを構築できますし、同様にサービス事業者が共通のインフラ上で、複数の顧客向けのマネージドVPNサービスを展開することもできます。

仮想マシンおよびモバイルノードのモビリティ

LISPでは識別情報(EID)と位置情報(RLOC)が分かれているため、マッピングサービス上のEIDに対応するRLOCを更新してあげることで、割り当てられているIPアドレスを維持しながらエンドノードのロケーションを移動するといったことを容易に実現できます。ディザスタリカバリやワークロード分散などの目的で、仮想マシンを地理的に離れたデータセンターへ移動させられます。また、モバイルノードに軽量なLISPトンネルルータ機能を持たせることで、IPポータビリティやローミングなどの目的で、モバイルノードのモビリティも実現できます[注22]。

◆ ◆ ◆

当初の目的であったインターネットの経路増加を抑制するという用途でのLISPの利用は、今のところ広まっていません。しかし、応用として紹介した利用例は、幸いにも複数の先進的なユーザからの関心を惹くことに成功して、国内外を問わず、商用サービスおよび本番ネットワークでの利用が始まってきています。紙幅の制約もあって、詳細な内容に触れることができませんでしたが、LISPに興味を持った方は、ぜひ、LISP関連のRFC6830～6836を一読することをお勧めします。SD

注19) 従来ですと、マルチホーミングで入力トラフィックの負荷分散を実現するには、BGPのMED属性を用いたり、異なるプレフィックス長で経路を広報したりと、比較的、煩雑な方法をとる必要がありました。

注20) 公衆網を利用する場合、IPsecと併用してトラフィックを暗号化することもできます。

注21) 物理的に1台のルータ上で、ルーティングテーブルおよび転送テーブルを論理的に分離するための技術。MPLS VPNを始め、各種VPNサービスの実現に利用される。

注22) このようなLISPのモバイルノード用途のために、LISPmobと呼ばれるオープンソース実装があります。LISPmobは、LinuxやAndroidなどで利用できます。

Column

ルータに用いられるプロセッサの進化

フリーランチの終焉

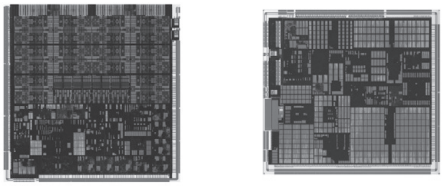
かつては、「ムーアの法則」による恩恵で、プロセッサの製造プロセスが刷新されれば、ソフトウェアは手を加えなくても自然と性能の向上を期待できました。しかし、2000年を過ぎたあたりから、「ムーアの法則」は、シングルプロセッサでは妥当しなくなります。以降、ソフトウェアの性能を向上させるには、マルチコアプロセッサを活用して、複雑になりがちな並列処理を意識しながらプログラミングする必要が出てきます。こういった状況をMicrosoft社のソフトウェアアーキテクトであるHerb Sutter氏は「フリーランチの終焉」と呼びました。もちろん、これは汎用プロセッサで作られてきたソフトウェアルータにとっても無縁の話ではく、その開発手法のあり方に大きな影響を及ぼしていくことになります。

シングルプロセッサの性能限界もあって、各ネットワーク機器ベンダは、ネットワークプロセッサ[注23]ベースのソフトウェアルータの開発を模索するようになります。当初は、固定機能ごとにステージを分けてパイプライン処理するもので、その多くはアセンブラのような低レベル開発言語で機能を実装する必要がありました。そのため、機能の追加は、困難で時間のかかるものでした。性能はそれなりの期待ができるものの、従来のソフトウェアルータが持っていた機能を部分的にしかサポートできなかったため、その普及は限定的なものにとどまりました。

QFP——次世代ネットワークプロセッサ

筆者が勤めるCisco Systemsでは、5年の歳月をかけて、2008年、QFP（QuantumFlow Processor）と呼ばれるネットワークプロセッサを自社開発しました。

▼図10　QFPエンジン（左）とQFPトラフィックマネージャ（右）



QFPは、パケット処理を行うQFPエンジンと出力スケジューリングを行うQFPトラフィックマネージャの2つのシリコンからなります（**図10**）。第一世代のQFPにおけるQFPエンジンでは、40のコアが1つのシリコンに集約され、各コアでは4つのスレッドで並列処理することが可能です[注24]。つまり、40×4＝160のスレッドで同時にパケット処理を行えます。そのほか、パケット分類、パケットの出力スケジューリング処理などは、QPFエンジンを補助する専用ハードウェアで支援されます。

QFPでは、1つのパケットに対して1つのスレッドが割り当てられ、パケット処理の全体を実行します。この処理は、C言語のような高級言語で実装できるため、機能の追加や拡張を柔軟に行うことが可能です。QFPをベースとしたルータは、従来のソフトウェアルータで利用されていた主要な機能うちの多くを当初からサポートできたため、広く普及していくことになります。

汎用マルチコアプロセッサへの応用

QFPのような専用ネットワークプロセッサによるルータは、必然的にコスト高を伴います。最近、Cisco Systemsでは、QFPの代わりに、広く流通するようになった汎用マルチコアプロセッサを用いて、パケット処理を並列実行するプラットフォームの出荷を開始しました。専用ハードウェアで支援されていた処理を、汎用マルチコアプロセッサが持つハードウェアで代替するか、もしくは、ソフトウェア的にエミュレートすることで、豊富な機能の多くをそのまま移植できるようにしました。もちろん、多量のコアとそれを支援する専用ハードウェアを持つ専用ネットワークプロセッサベースのプラットフォームほどの性能は期待できませんが、それでも、シングルプロセッサルータよりもはるかに優れた性能を比較的安価に実現できます。

注23) パケット処理に特化したプロセッサ。多量のコアを積み、パケット処理に付随する処理（暗号化など）を支援するためのハードウェアを併せ持つ。

注24) 1つのスレッドが外部リソース（DRAM、TCAM、暗号エンジンなど）からのレスポンス待ちでストールしている間、ほかの3つのスレッドの処理を続けられることで、1つのコア内での使用率を上げることができます。

CHAPTER 3

活用範囲が広がるBGPとVyattaによる仮想環境のルーティング実践

［応用編］

大久保 修一（おおくぼ しゅういち）
さくらインターネット（株）
Twitter @jq6xze_1

応用編となる本章は、「BGP」と「クラウド／仮想環境でのルーティング実践」の2本立てです。これまで使う機会は少なかったであろうBGPですが、最近では組織内やVPNで利用されるなど、クラウドとのかかわりが深まることにより活用範囲が広がってきています。実践では仮想ルータのVyattaを使い、VPNによるiPhoneでのリモート接続環境やインタークラウド環境の構築方法を解説します。

組織間ルーティングの要「BGP」

インターネットを支えるBGP

一般の方にとっては、ルーティングプロトコルの中でも比較的馴染みのない技術かもしれませんが、BGPはインターネット全体の経路制御に用いられており、インターネットを動かすうえで非常に重要なプロトコルの1つです。

BGPは“Border Gateway Protocol”の略で、最初の仕様は1989年にRFC1105として提案されました。その後改訂が重ねられ、1995年に発行されたRFC1771を元に、当時インターネットで使用されていたBGPバージョン3からバージョン4（BGP-4）へのマイグレーションが行われて現在に至っています。

BGP-4の基本仕様はその後も改訂され、2006年に発行されたRFC4271が最新となっています。また、時代の変化に合わせてさまざまな拡張仕様が定義されてきました。主要なものを挙げると、

- IPv6やVPNに対応するためのマルチプロトコル拡張
- BGPセッションのセキュリティを向上するためのTCP MD5の規定
- AS番号枯渇対策である4バイトASへの対応

などがあります。

最近のトピックとしては、経路収束の高速化を目的としたADD-PATHと呼ばれる新たな属性を定義する動きや、BGPメッセージのエラーハンドリングを規定する動きもあり、BGPは今後も進化を続けることになりそうです。

インターネットはASの集まり

BGPでは組織間のルーティングを行うため、AS番号というもので組織を特定しています。インターネットで使用されるAS番号はグローバルAS[注1]と呼ばれ、全世界でユニークでなければならないため、インターネットレジストリであるAPNIC（アジア太平洋地域を担当）やJPNIC（日本国内を担当）などが矛盾のないように割り当てを行っています。

国内の主要事業者が運用しているAS番号の一例を**表1**に示します。事業者によっては企業統合などによっていくつかの異なるネットワークが存在し、一社で複数のASを運用している

注1）閉じたネットワークで実験的に利用できるプライベートASも定義されています。

ケースもあります。最近ではISPだけでなく、データセンター事業者やコンテンツプロバイダが回線の調達コストを抑えたり、高可用性を実現したりといった目的で、自らBGPを運用するケースも増えています。

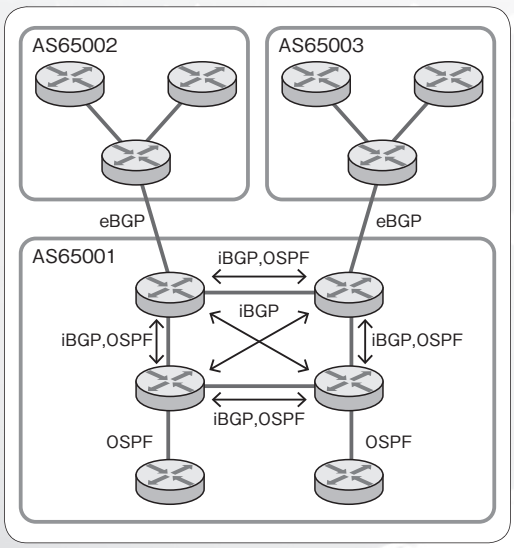
組織間、組織内のルーティング

BGPのセッションは、AS間(異なる組織のルータ間)で設定するものと、AS内(同じ組織内のルータ間)で設定するものとに区別されています。前者はeBGPセッション、後者はiBGPセッションと呼ばれ、eBGPは外部のASから経路を得たり、逆に自ASの経路を外部に広報するために使用されます。

一方iBGPは、各eBGPルータが得た経路情報を同一AS内のルータ間で交換し、同期をとるために使用されます。BGPでは、基本的に外部ASの経路のみを扱うため、AS内の細かいルーティング(内部経路)については、別途OSPFなどのIGPを併用しなければなりません[注2]。**図1**にAS、BGP、OSPFの関係を示します。

注2) 大手ISPの中には、内部経路がOSPFでハンドリングできる規模を越えてしまい、内部経路もBGPでルーティングしているところがあります。

▼図1 AS、BGP、OSPFの関係



なお、BGPを経由してインターネット上でやりとりされているすべての経路情報をフルルートと呼びますが、年々増加の一途を辿っています(**図2**にフルルート数の推移を示します)。2013年8月現在、その数は45万経路を超えており、ルータのメモリやTCAMなどのハードウェアリソース不足に悩まされるオペレータもいます。

ポリシーを反映して経路を最適化

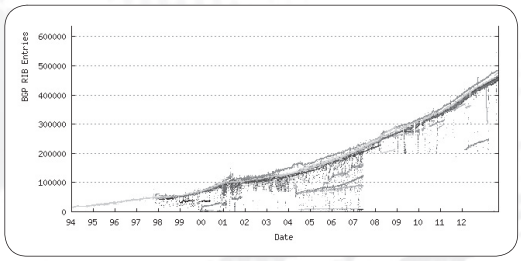
IGPではコストやホップ数といった指標でのみ経路が決定され、ネットワーク管理者の意図を反映できる余地がほとんどありません。逆に言うと、IGPはネットワーク管理者自身ですべて把握可能な組織内ネットワークで運用することが想定されているため、コストの調整のみでも事足りるといえます。

一方、BGPは異なる組織間で使用されるため、たとえば次のようなネットワーク管理者のルーティングポリシーを反映できるしくみが備わっています。

▼表1 グローバルAS番号の例

AS番号	事業者名(サービス名)
AS2497	インターネットイニシアティブ
AS2516	KDDI
AS2518	NECビッグローブ
AS4713	NTTコミュニケーションズ(OCN)
AS9370	さくらインターネット
AS15169	Google
AS17676	ソフトバンクBB
AS38651	ミクシィ
AS55394	グリー

▼図2 フルルート数の推移
(http://bgp.potaroo.net/より引用)



(a)ある組織A向けのトラフィックを組織B経由で流したい
(b)ほかから自組織宛てに流れてくるトラフィックをなるべく組織C経由にしたい
(c)組織Dとの接続点が複数あり、品質の高い回線を優先したい

このようなポリシーを実現するためのしくみとして、BGPでは経路情報に宛先のIPアドレスブロック(192.0.2.0/24のような)以外にも、さまざまな属性を付与できるようになっています。送受信する経路のBGP属性を操作することによって、管理者の意図したトラフィックフローにすることができます。

よく利用される属性と使用例を挙げます。

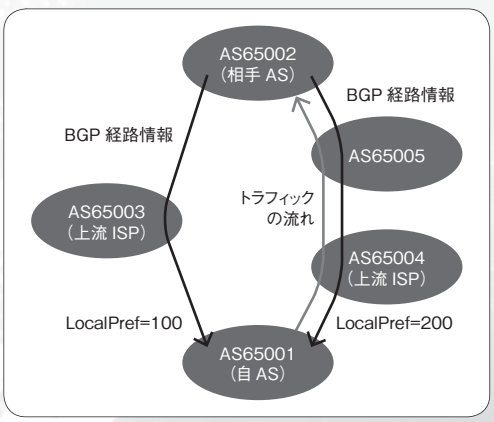
①Local Preference属性

最も評価優先度が高いBGP属性です。回線コストを抑えるなどの目的で、上記(a)のようなポリシーを反映するために使用されます。このLocal Preferenceは32bitの整数で表され、数値の「大きいほう」が優先されます(デフォルトは100です)。

たとえば、**図3**のようにトラフィックの多いAS65002に至る経路を、上流ISPAS65003とAS65004から受信しているとします。このとき、回線コストが安いなどの理由で、あえて遠回りとなるAS65004を選択したい場合には、AS65004から受信したAS65002の経路情報に高いLocal Preference値を付与します(たとえば200など)。

なお、Local Preference値を上げると使用する回線がほぼ決め打ちとなります。複雑な構成では思わぬ経路に副作用が生じることもありますので、使用する際は十分に検討が必要です。

▼図3　Local Preferenceを用いた経路制御



②AS-PATH属性

AS-PATH属性には、ASを越えて経路情報が伝えられる際、AS番号のリストが順次追記されるようになっています。実際の例を見てみましょう。**図4**はさくらインターネットが公開しているLooking Glass[注3]にて検索した、Apple社のWebサイト(apple.com)の経路情報です。これによると、Apple社(AS714)から広報された経路情報が、AS3356(Level3)とAS2914(NTTコミュニケーションズ)を経由して伝わってきていることがわかります(下線部)。

BGPで経路の優先度が決定される際、Local Preference値が同じ場合にはAS-PATHが短い経路情報が優先されます。この性質を逆手にとって、自身の経路情報を近隣ASに広報するときに、通ってほしくないAS向けに自身のAS番号

注3) そのASが持っている経路情報を外部に公開しているサイト。世界中の多くのサイトが公開しており、広報した経路情報がどのようにインターネット全体に伝わっているかを確認できる。

▼図4　AS-PATHの例

```
> show ip bgp 17.172.224.47
BGP routing table entry for 17.168.0.0/13
Paths: (2 available, best #1, table Default-
IP-Routing-Table)
  Advertised to non peer-group peers:
  192.41.192.119 210.173.172.66 210.188.224.38
  2914 3356 714

    59.106.248.3 from 59.106.248.22
(59.106.248.3)
      Origin IGP, metric 3100, localpref
100, valid, internal, best
      Community: 9370:100
      Originator: 59.106.248.3, Cluster
list: 59.106.248.22
      Last update: Tue Jun 18 12:04:22 2013
```

※http://as9370.bgp4.jp/より

をあえて余分に付加し、AS-PATHを長く見せることで(AS-PATH Prependと呼ばれる手法)、前に挙げた(b)に相当するようなポリシーを実装することもできます。

③MED属性

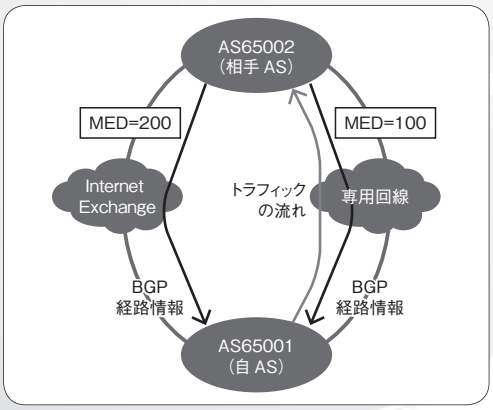
Local Preference値もAS-PATH長も同じ経路が複数あった場合、MED属性が優先度決定に参照されます[注4]。前に挙げた(c)のような、おもに隣接ASとのポリシーを実装する際に用いられます(**図5**)。

MEDは32bitの整数で、数値の「小さいほう」が優先度が高くなっています(Local Preferenceと逆なので注意してください)。なお、MEDはデフォルトでは「空」になっています。この空のMEDを最小値(0である)とみなす機器と、最大値(4,294,967,295=2^{32}-1である)とみなす機器があり、オペレーションの現場では混乱するケースがあります。

また、通常MED値は経路を広報する側がトラフィックを受信したい回線を相手に伝えるために使用するのですが、相手側の事情により意図したとおりにならない場合もあります(MED値を相手が上書きしているケース)。MED値を自分と相手のどちらで付与するのか、どの値を設定するのか、隣接ASのオペレータとの密な連携が必要な属性の1つです[注5]。

注4) 正確にはMED属性の前にORIGIN属性が評価されるのですが、あまり使用例を見たことがないので割愛します。

▼図5 MEDを用いた経路制御



インターネット以外でも使われるBGP

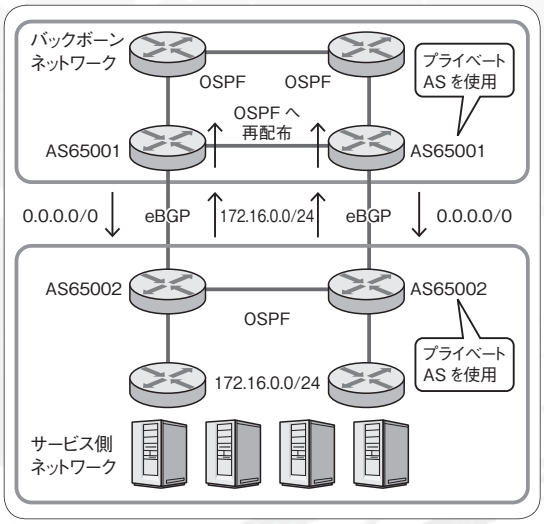
これまで解説してきたBGPの動作は、ISPやデータセンター事業者のインターネットバックボーンに接する部分を想定したものでした。しかし、BGPの用途はそれだけにとどまらず、組織「内」でのルーティングや、一般企業におけるVPN接続など、身近なところで利用されるケースも多くなってきています。

まずは組織内でのBGPの利用例を紹介します。異なる部門が管理しているネットワーク同士を接続し、ネットワークの管理ドメインを分割したいようなケースで、BGPが活用されることがあります。**図6**に弊社(さくらインターネット)のクラウドサービスで実際に運用している構成を示しました。

弊社ではデータセンターバックボーンとサービスネットワークを運用している部門が異なっています。このような状況でバックボーンとサービスネットワークの間に回線を2本引いて冗長構成を組むにはどうしたらよいでしょうか?
やはり最初に思いつくのは、単純にサービス収

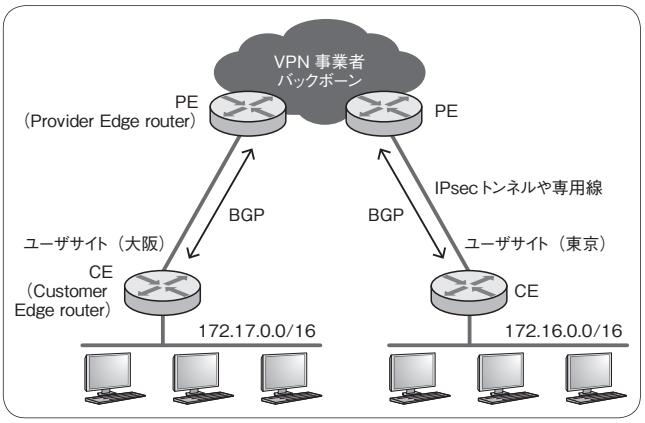
注5) ほかにもMEDにまつわる苦労話がJANOG19で発表されています。URL http://www.janog.gr.jp/meeting/janog19/2006/10/bgp.html

▼図6 組織内でのBGP接続例



容ルータをバックボーンのOSPFドメインに組み込んでしまう方法です。しかし、OSPFを使用する場合、①バックボーン側で扱っている多くの経路情報がそのままサービス収容ルータにもインストールされるためハードウェアリソースが厳しくなってしまう。②サービス収容ルータで不正な経路情報を生成してしまった場合、バックボーンに影響してしまう。などといった課題があります。そこでプライベートASを使用したBGPによる冗長構成をとることにしました。バックボーンからはOSPFの細かい経路情報は渡さずに、デフォルトルート(0.0.0.0/0)のみをサービス収容ルータに渡してあげることで、①の問題を解決できました。また、受け入れるべきでない経路情報をバックボーン側でフィルタすることで、②の問題も解決できました。

▼図7　IP-VPNエッジでBGPを使用する構成



IP-VPNエッジで使われるBGP

IP-VPNとは、キャリアやデータセンター事業者が保有する広域のIPネットワークを用いて、離れた拠点に存在するユーザのプライベートネットワークを相互接続するサービスです。IP-VPN事業者は、バックボーンネットワーク上で複数ユーザの経路を論理的に分離された状態でルーティングします。

IP-VPNを利用する際、ユーザが各拠点で使用しているIPアドレスブロックをIP-VPN事業者がルーティングする必要がありますが、ここで**図7**のようにBGPを利用するケースがあります。

また、同様の事例としてクラウド環境とオンプレミス環境を相互接続するVPNでも使用されています。1つ例を挙げてみますと、Amazon社が提供しているIaaSのVPC(Virtual Private Cloud)という機能で、ユーザサイトの経路情報をクラウド側に注入するためにBGPが使われています[注6]。

BGPのまとめ

駆け足になってしまいましたが、BGPの概要と実際のユースケースについて紹介しました。超大規模ネットワークであるインターネットのルーティング技術で、少々とっつきにくいところもあるかもしれませんが、組織内やVPNでのルーティングに適している部分もあります。なお、次のテーマ(クラウド環境でのルーティング)でVyattaにおける簡単なBGPの設定例も紹介していますので、併せてご覧いただければと思います。

注6) http://aws.amazon.com/jp/vpc/faqs/

クラウド／仮想環境でのルーティング実践

大規模なシステムもクラウド上に

クラウドがこれほど発達していなかった数年前までは、サーバ数百台を稼働させるような大規模なシステム構築には、非常に大きな労力を要していました。しかし、ここ数年のクラウド技術、サービスの発展には目をみはるものがあ

り、実際に大規模なソーシャルゲームプラットフォームをはじめ、ダウンすると即業務停止に陥るようなクリティカルな基幹システムなどもクラウド上にマイグレーションされる事例も現れています。数年前までは考えられなかったことです。

そうしたクラウド化の流れの中で、サーバ仮想化のみならずネットワーク仮想化の知識が必要とされてきています。ネットワークを仮想化することで、物理的なルータやスイッチ、ファイアウォールやロードバランサといったハードウェアアプライアンスの管理作業から開放され、大規模なシステムでも少人数で運用が可能です。構築にかかる日数は短縮され、瞬時にデプロイできますし、不要となったシステムはそれこそ即座に消去することもできてしまいます。

ここからのテーマは、そうしたクラウドなどの仮想環境で必要となる仮想ルータの活用方法について紹介します。

仮想ルータとVyatta

仮想ルータとは、従来のハードウェアルータと同等、もしくは一部機能を搭載し、クラウドをはじめとするサーバ仮想化環境上で動作する仮想アプライアンスのことを指します。メジャーなところでは、現在**表2**のような製品があります。クラウド環境上にこれらをインストールすることで、ルーティングやVPN、NAT、ファイアウォールといった高度なネットワーク機能を利用することが可能となります。

本稿では**表2**の中でもさまざまなクラウド事業者にて提供されているVyattaを紹介します。

Vyattaは、米Vyatta社[注7]が開発、提供しているオープンソースの仮想ルータ製品です。

無償版である「Vyatta Core(以下、VC)」と、有償サポートが受けられる「Vyatta Subscription Edition(以下、VSE)」の2種類のラインナップがあり、VCはWebサイト[注8]からダウンロードできます。なお、VSEにはVCでは実装されていないGUIやAPI、VPNの拡張機能が提供されています。

Vyattaは、Linuxをベースに各種オープンソースソフトウェアをパッケージングしたディストリビューションとなっており、さまざまな機能を統一されたCLI(JUNOSライク)にて簡単に設定できるようになっています。日本国内にもJapan Vyatta Users Group[注9]という有志によるコミュニティがあり、盛んに活動しています。

Vyattaを使って仮想プライベートクラウド環境を構築しよう

ここからは、NAT、DHCP、リモートアクセス、VPNといったVyattaの持つさまざまなネットワーク機能を活用し、自分好みの仮想プライベートクラウド環境を構築する方法を紹介していきます。なお、Vyattaのインストール方法、基本的な操作方法については、すでに本誌の過去の記事にて解説されていますので、今回は割愛したいと思います。また、手前味噌で恐縮ですが、筆者も執筆に参加した書籍『オープンソース・ソフトウェアルータVyatta入門——実践ルーティングから仮想化まで』(技術評論社刊)もよろしければ併せて参照ください。

注7) Vyatta社は昨年末にネットワーク機器メーカであるBrocade社に買収されました。
注8) http://www.vyatta.org/downloads
注9) http://www.vyatta-users.jp/

▼表2 仮想ルータの例

提供元	製品名
Vyatta(Brocade)	Vyatta Core、Vyatta Subscription Edition
IIJ	SEIL/x86
Cisco	Cloud Services Router 1000V
Juniper	JunosV Firefly
Electric Sheep Fencing	pfSense

Step1. プライベートネットワークを設定する

では早速、**図8**のような仮想プライベートクラウド環境を例に、Vyattaにプライベートネットワークの設定をしていきます[注10]。

まず、インターフェースのIPアドレスとデフォルトゲートウェイの設定をします。IPアドレスは適宜実際の環境のものに読み替えてください。ここでは、次のパラメータを例に説明します。

・デフォルトゲートウェイ：203.0.113.1
・グローバル側IPアドレス：203.0.113.5/24
・プライベート側IPアドレス：10.103.1.1/24

設定モードに移行し、**図9**のように入力してください。Vyattaのインストール、および初期設定は完了しているものとします。

これだけではプライベート側のサーバが外部と通信できませんので、続いてNATの設定を行います(**図10**)。

さらに内部に存在するサーバを外部に公開する場合は、リバースNATの設定を行います。ここでは次のようにIPアドレスを紐付ける1対1NATを説明します。

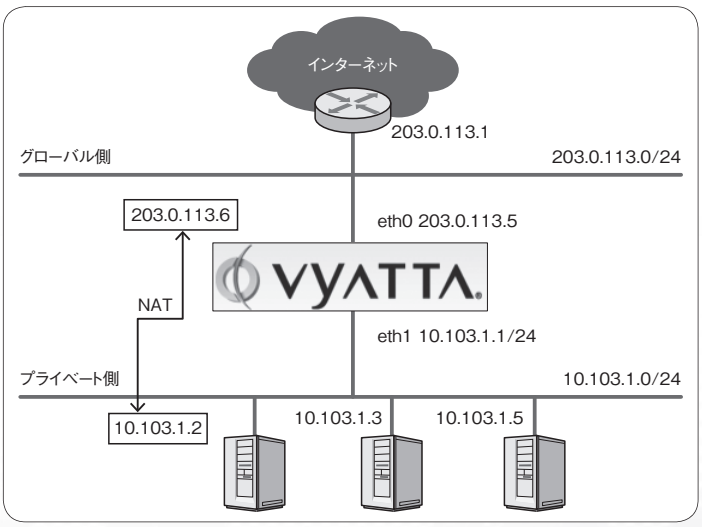
・グローバル側IPアドレス：203.0.113.6
・プライベート側サーバ：10.103.1.2

設定モードに移行し、**図11**のように入力してください。

これで、インターネットから203.0.113.6に宛てた通信が10.103.1.2に転送されるようになります。

ところで、内部サーバに対してDHCPを用いてアドレス配布をする場合、VyattaにDHCPサーバの設定を行います。DHCPを用いるとサーバにIPアドレス設定を行う必要がなくなる

▼図8　仮想プライベートクラウド環境の構築例



▼図9　IPアドレスの設定

```
set interfaces ethernet eth0 address 203.0.113.5/24
set interfaces ethernet eth1 address 10.103.1.1/24
set protocols static route 0.0.0.0/0 next-hop 203.0.113.1
commit
```

▼図10　NATの設定

```
set nat source rule 999 outbound-interface eth0
set nat source rule 999 translation address masquerade
commit
```

▼図11　リバースNATの設定

```
set interfaces ethernet eth0 address 203.0.113.6/32
set nat destination rule 10 destination address 203.0.113.6
set nat destination rule 10 inbound-interface eth0
set nat destination rule 10 translation address 10.103.1.2
set nat source rule 10 outbound-interface eth0
set nat source rule 10 source address 10.103.1.2
set nat source rule 10 translation address 203.0.113.6
commit
```

注10）本稿で紹介する設定は、執筆時点で最新版であるVyatta Core 6.6R1にて動作検証を行っています。

ため、管理しやすくなります。なお、通常は指定した範囲から動的にアドレスが払い出されますが、MACアドレスに対応してIPアドレスを固定設定する場合は、static-mappingの設定を台数分入力します(**図12**)。

プライベートネットワークに存在するホストは、DHCPクライアントとして動作するように設定します。CentOSやRHEL系のOSの場合、/etc/sysconfig/network-scripts/ifcfg-eth0 に次のような設定を書き込みます。

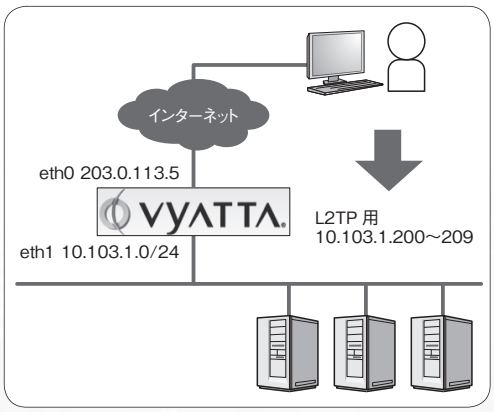
```
DEVICE=eth0
BOOTPROTO=dhcp
```

Step2. リモートアクセス環境を設定する

この環境では、プライベート側に存在するサーバには外部から直接アクセスできません。そこで、リモートアクセスの設定を続いて行います。リモートアクセスとは、**図13**のようにVPNを用いて内部サーバに対して安全にアクセスするしくみです。ここでは、次のパラメータを例に、VyattaをL2TP/IPsecサーバとして設定する方法について紹介します。

・ユーザ名／パスワード：
　user1/aaabbb
・Pre-Shared Key：cccddd

▼図13　L2TP/IPsecによるリモートアクセス構成



▼図12　DHCPサーバの設定

```
edit service dhcp-server shared-network-name
private subnet 10.103.1.0/24
set default-router 10.103.1.1
set dns-server 8.8.8.8
set dns-server 8.8.4.4
set start 10.103.1.100 stop 10.103.1.199
set static-mapping host1 ip-address 10.103.1.2
set static-mapping host1 mac-address ↵
00:11:22:33:44:55
commit
top
```

▼図14　リモートアクセスの設定

```
set vpn ipsec ipsec-interfaces interface eth0
set vpn ipsec nat-networks allowed-network 0.0.0.0/0
set vpn ipsec nat-traversal enable

edit vpn l2tp remote-access
set authentication local-users username user1 password aaabbb
set authentication mode local
set client-ip-pool start 10.103.1.200
set client-ip-pool stop 10.103.1.209
set dns-servers server-1 8.8.8.8
set dns-servers server-2 8.8.4.4
set ipsec-settings authentication mode pre-shared-secret
set ipsec-settings authentication pre-shared-secret cccddd
set mtu 1280
set outside-address 203.0.113.5
set outside-nexthop 203.0.113.1
commit
top
```

▼図15　iPhoneによるL2TP/IPsec接続設定例

・払い出しIPアドレス：10.103.1.200～209
・DNSサーバ：8.8.8.8/8.8.4.4

設定は**図14**のようになります。

アクセスするユーザが複数ある場合は、authentication local-usersにて必要なだけのユーザ名、パスワードを定義してください。

接続を行うクライアントは、WindowsやMac、そのほかL2TP/IPsecに対応しているOSであればとくに問題なくつながるはずですが、ここでは簡単な例として、iPhoneから接続する設定を紹介します。

iPhoneで、［設定］－［一般］－［VPN］－［VPN構成を追加］と選択すると**図15**のような画面が表示されます。VyattaのIPアドレスを「サーバ」欄に、ユーザ名とパスワードを「アカウント」と「パスワード」欄に、Pre-Shared Keyを「シークレット」欄にそれぞれ入力するだけで設定は完了です。

Step3. マルチクラウドで運用する

最近、クラウド界隈での流行りのキーワードに「インタークラウド」というものがあります。これは、既存のオンプレミスの環境とクラウド環境を接続して並行運用したり、耐障害性の向上やディザスタリカバリといった目的で、複数の事業者のクラウド上に構築した環境を相互接続する概念です。

このインタークラウドは、一部事業者がすでにサービス提供していますが、もし使用しているクラウドが対応していない場合でも、VyattaのVPN機能を活用して自身で構築することができます。ここでは、**図16**のように異なるクラウド上に構築した仮想プライベートクラウド環境を接続する方法について紹介します。

次に示すパラメータをもとに、一番手軽に設定できるsite-to-siteモードのOpenVPNを使った方法を紹介します[注11]。

クラウドAのVyatta
・グローバルIPアドレス：203.0.113.5
・トンネルIPアドレス：10.103.3.1/24
・プライベートIPアドレス：10.103.1.0/24

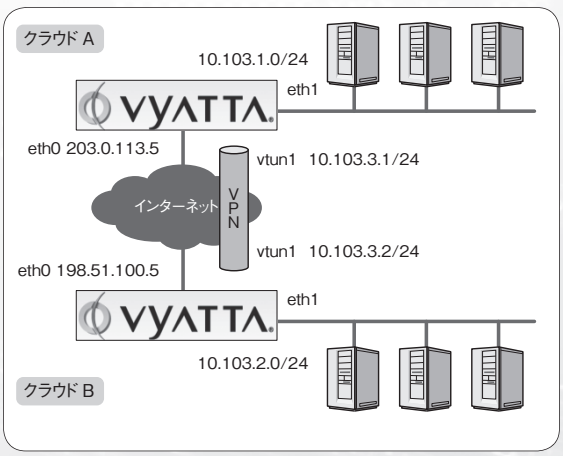
クラウドBのVyatta
・グローバルIPアドレス：198.51.100.5
・トンネルIPアドレス：10.103.3.2/24
・プライベートIPアドレス：10.103.2.0/24

トンネルIPアドレスはVPNインターフェースに設定するものですので、使用していないプライベートアドレスを適当に割り当てれば問題ありません。まず、片方のVyatta（クラウドA）でOpenVPNの共通鍵を生成して、もう一方のVyatta（クラウドB）にコピーします。**図17**のコマンドは一般モード（設定モードを抜けた状態）にて行ってください。

続いてOpenVPNの設定を行います。**図18**は

注11）手軽に設定できる反面、オーバヘッドが大きかったり、OpenVPN同士でしか接続できないといった制限もあります。パフォーマンスが問題になるような用途や、一般的なVPN機器と接続する場合は、IPsecの利用を検討してください。

▼図16　インタークラウド環境の構築例



▼図17　共通鍵の生成とコピー

```
$ generate openvpn key /config/auth/secret
$ sudo scp /config/auth/secret ➡
vyatta@198.51.100.5:/config/auth/
```

設定モードで入力してください。対向側のVyattaも同様に設定します(**図19**)。

以上でお互いのサイトの内部サーバ間で通信できるようになります。

ところで、多くのサイトを接続する場合、経路設定(protocols static)が煩雑になってしまいます。そのような場合、動的ルーティングを用いるのも1つの手です。せっかくですので、前のテーマにて解説したBGPを使った設定例を紹介しましょう(**図20**)。

▼図18　OpenVPNの設定

```
edit interfaces openvpn vtun1
set local-address 10.103.3.1 ↵
subnet-mask 255.255.255.0
set mode site-to-site
set remote-address 10.103.3.2
set remote-host 198.51.100.5
set shared-secret-key-file /config/↵
auth/secret
top

set protocols static interface-route ↵
10.103.2.0/24 next-hop-interface vtun1
commit
```

▼図19　対向側のOpenVPNの設定

```
edit interfaces openvpn vtun1
set local-address 10.103.3.2 subnet-mask ↵
255.255.255.0
set mode site-to-site
set remote-address 10.103.3.1
set remote-host 203.0.113.5
set shared-secret-key-file /config/auth/secret
top

set protocols static interface-route ↵
10.103.1.0/24 next-hop-interface vtun1
commit
```

65001は自分のAS番号を、65002は相手のAS番号を指定します。対向側は逆に入力してください。また、networkの部分は、自身のプライベートネットワークのアドレスを指定します。なお、BGPを使う場合は前に設定したstatic routeは削除してください(**図21**)。

正常に動作すると**図22**のようにBGPで経路情報が交換されていることが確認できます。

まとめ

本稿では仮想環境やクラウド環境で利用できるVyattaを用いてプライベートクラウド環境を構築する方法を紹介しました。従来、高価なファイアウォールやVPN機器を導入しなければならなかったシステムが、手軽に構築できることがおわかりいただけたでしょうか？　若干CLIがとっつきにくい、という方もいらっしゃるかもしれませんが、ぜひチャレンジしてみてください。SD

▼図20　BGPを使った動的ルーティングの設定例

```
edit protocols bgp 65001
set neighbor 10.103.3.2 remote-as 65002
set neighbor 10.103.3.2 soft-↵
reconfiguration inbound
set network 10.103.1.0/24
commit
top
```

▼図21　static routeの削除

```
delete protocols static interface-route
commit
```

▼図22　動作確認

```
vyatta@test-vc66-1:~$ show ip bgp
BGP table version is 0, local router ID is 133.242.72.116
Status codes: s suppressed, d damped, h history, * valid, > best, i - internal,
              r RIB-failure, S Stale, R Removed
Origin codes: i - IGP, e - EGP, ? - incomplete

   Network          Next Hop            Metric LocPrf Weight Path
*> 10.103.1.0/24    0.0.0.0                  1         32768 i
*> 10.103.2.0/24    10.103.3.2               1             0 65002 i

Total number of prefixes 2
```

BOOK no.1 実践Vim
思考のスピードで編集しよう!

Drew Neil 【著】、 新丈 径 【訳】
A5判、 400ページ／価格＝2,940円 （税込） ／発行＝アスキー・メディアワークス
ISBN＝978-4-04-891659-2

本書はVimのレシピ集だが、プラグインの紹介本ではない。Vimを最大限に活用するための基本コマンドの組み合わせサンプル集といった内容だ。Vimには“i”と“a”のような似ているけれど動作が微妙に異なるコマンドが多々ある。けれど、その必要性がわからない、という人は本書を読むことで各コマンドの活用方法がわかるだろう。第1章は直前の変更を繰り返すための“.”コマンドの解説だが、まずはここだけでも読んでみると良い。細かなコマンドが使い分けられると、いかに“.”の活用の幅が広がるかがよくわかる。Vimの実力を発揮するには、他エディタにはない独特の考え方を知る必要がある。本書でその勘どころをつかんでほしい。

BOOK no.2 数学ガールの秘密ノート
式とグラフ

結城 浩 【著】
四六判、 232ページ／価格＝1,260円 （税込） ／発行＝ソフトバンク クリエイティブ
ISBN＝978-4-7973-7414-8

主人公の「僕」と3人の少女が数学についての会話を繰り広げる数学ガールの新シリーズ。恒等式、連立方程式、1次関数、正比例／反比例、2次関数、それらを題材に話は展開する。しかし、これらはバラバラの話題ではない。正比例／反比例の話あたりから、だんだんと話題は1つに収束していく。「数式の世界」と「図形の世界」はつながっていること、さらには数学は数式と図形を通じて現実の世界ともつながっていること、それらが感覚的にではなく論理的に理解できる。きちんと腑に落ちる。また、本書では1つ1つ理解を積み重ねていく心地よさや、その積み重ねがもっと大きな理解につながったときの新鮮さ、そういう感覚も味わえる。

BOOK no.3 Software Design plus シリーズ
はじめての3Dプリンタ
3Dデータ作成／出力まるごと体験ガイド

水野 操、 平本 知樹、 神田 沙織、 野村 毅 【著】
B5判、 128ページ／価格＝2,604円 （税込） ／発行＝技術評論社
ISBN＝978-4-7741-5973-7

昨今話題の3Dプリンタは、多彩な機能を持った個人向け製品の低価格化など、本体（ハードウェア）について語られることが多い。しかし、実際に「何かを作ってみたい」と思い立ったら、3Dプリンタの本体以外に、造型方式や素材、さらに3Dデータを作成するソフト（サービス）などの幅広い情報が必要になってくる。そんなときに本書は役に立ちそうだ。内容は「3Dプリンタ入門」から「出力サービスの利用法」「3Dプリンタカタログ」「3D FABスペースの紹介」「メタセコイアによるロボットモデリング」など多くの記事で構成されており、ひととおりの基礎知識からデータ作成方法、FABスペースの最新事情まで習得できる。

BOOK no.4 Software Design plus シリーズ
独習Linux専科サーバ構築/運用/管理
あなたに伝えたい技と知恵と鉄則

中井 悦司 【著】
B5変形判、 384ページ／価格＝3,129円 （税込） ／発行＝技術評論社
ISBN＝978-4-7741-5937-9

本書を一言で言えば、Linuxの新定番教科書だ。40代のエンジニアならばアスキーの『たのしいUNIX』（坂本文）が定番ではなかろうか。かくいう評者もこの本には助けられた。catコマンド1つとっても、昔はconCATenate（連結）と教えてくれる人すら少なかった。坂本氏の本により、大きく理解の幅をのばしUNIXという環境が好きになったことを覚えている。本書はそのくらいの入門者が対象である。クラウド上で当たり前のようにUNIXインスタンスが稼働する時代になったが、その基礎を知らねば障害発生時に正しい処置ができない。今はそれがニーズとなって本書の存在価値がそこにある。若手を育てる本としてぜひ一読いただきたい。

分散KVS「okuyama」の開発から現在そしてこれから

Key-Value Storeをゼロから創る

㈱神戸デジタル・ラボ　岩瀬高博（いわせ たかひろ）　Twitter@okuyamaoo
イラスト：高野 涼香

はじめに

はじめまして、㈱神戸デジタル・ラボの岩瀬です。今回は国産のNOSQLデータベースである分散Key-Value Store（以降：分散KVS）okuyamaの紹介と開発中の出来事、気づきなどを紹介させていただきます。

なぜ分散KVSを開発しはじめたのか

NOSQLとは?

皆さんはNOSQLをご存じですか？　NOSQLはNot Only SQLのことであり、SQLを利用するRDBMSとは違ったアプローチを用いたデータベースです。シンプルなデータ構造と操作、そして高速な応答速度が特徴です。okuyamaもこれらの特徴を備えています。まず筆者がこのokuyamaを開発しはじめた動機から紹介します。

開発の動機とは?

筆者がokuyamaを開発し始めたのは2009年頃です。当時は、NOSQLといえばKVSが活用されていました。その中でもMemcachedに関しては、筆者もWebサイトのレスポンス改善に活用し、その処理性能に驚かされました。それでNOSQLというソフトウェアの存在が頭の片隅に茫漠とありました。

偶然、ある案件でNOSQLであるHBaseの評価をすることになりました。それはGoogle社が開発を行い、利用しているBigTableの思想を元に開発されたOSS（Open Source Software）です。このHBaseを検証していくうちに、もともとデータベースに強い技術的興味を筆者が持っていたこともあり、その他のNOSQLについても調べるようになりました。そうするうちに、NOSQLはいくつかの種類に分類されていることや、用途に合わせて利用することで非常に高い性能を発揮するデータベースであることがわかってきました。

とくに性能の部分には興味をそそられました。NOSQLの性能を示す値としてQPS（Queries Per Second）という単位が用いられることが多いのですが、これは秒間当たりの処理回数です。この性能値が10,000QPSというNOSQLソフトウェアも多く、当時はその処理能力に驚くばかりでした。

現在では、HBaseやCassandra、MongoDBなどは書籍やWeb上で多くの紹介記事が出ており、性能やアーキテクチャ、その使い方などを知ることが可能ですが、その当時はまだ情報が乏しく、設定し稼働させるだけでもひと苦労でした。そこでNOSQLの概念を手っ取り早く確実に理解するには、自身で開発を行えばいいのではないかと思いました。

最初は業務としてではなく趣味でokuyamaの開発に取り掛かりました。当時は所属会社でプログラミング業務からマネージメント業務へ担当業務が移行していた時期でもあり、自分でプログラムを書き、何か作りたいという漠然と

した欲求も強かったのです。

開発初期の活動

開発環境の整備

実装を始めるにあたり開発環境を整えます。開発をするうえでソースコードやモジュールのバージョン管理が重要です。それらを手軽にホスティングできる環境として以前から利用していたSourceforge.jpを選択しました。Sourceforge.jpを選択することでソースコードのホスティングだけでなく完成後もオープンソースとしての公開管理ができます。

そして開発言語は最も使い慣れているJavaを選びました。あくまで趣味の開発なので、すでに使い慣れている技術に絞り、開発初期の取り掛かりを容易にしました。また何よりJavaが好きだったのです。

開発前の下調べ

これで開発環境は整ったわけですが、すぐに始められません。まず、NOSQLとしてどのような機能を備えていればよいのかを整理することにしました。幸いなことに業務でHBaseの検証を行い、その他のNOSQLも比較のために調査していたため、しくみの部分は完全に理解できていなくても、その機能は理解していました。そこでokuyamaに実装したい機能の基本であるデータアクセスメソッドや構成は、検証での成果を参考に設計をしました。**表1**のようにまとめました。

okuyamaの機能選定

これらの調査から、okuyamaの基本概念を決めました。まずデータ構造ですが最もシンプルなKey-Value型としました。

これはすでに実装言語であるJavaには連想配列であるMap型の実装が取り込まれており、コアのデータ保存部分をゼロから実装しなくて良いと考えたためです。そしてセカンダリーのKey情報としてTagを持つこととしました。まだこの時点では可用性やサーバ構成などに関しての構想は考えられておらず、データベースの基本的な部分から実装に取り掛かることにしました。

開発をはじめてわかってきたこと

初期リリース

開発のとりかかりは想像以上にスムーズでした。それはJavaにネットワークやデータを保持するしくみ、データを永続化するためのファ

▼表1　NOSQLの特徴

データ構造	・シンプルなデータ構造であるKey-Value型を基本としている ・Key以外にもセカンダリーのKeyとも言える構造を備えていることがある
データアクセスメソッド	・データへのアクセスはKeyでのアクセスを基本としたシンプルなもの ・DBMSの得意とするJOINのような操作が存在しない ・トランザクションの概念が存在しない場合が多い
データ管理	・メモリのみでデータを管理し高速化を実現している方式 ・メモリ以外にディスクにデータを記録し、データ永続性を備えた方式 ・NOSQLには大きくこれら2つのデータ管理方式を備えたソフトウェアが存在する
サーバ構成	・複数台のサーバ上で稼働し、それらを連携させ1つのデータベースとする分散管理の機能を備えている
可用性	・サーバ障害などに強く、複数のサーバで構成されていることで一部に不具合が発生しても、データベースとして停止しない機能（SPOF：Single Point of Failureの除外）を備えている
性能	・シンプルなデータ構造のため、非常に高速なアクセス性能を備えている ・サーバを追加することで、データベースの性能を向上させるスケールアウトの機能を備えている

▼図1　okuyamaのロゴ

イルI/Oのしくみが備わっていたため、すみやかに初期リリース版0.0.1を完成させることができました。

リリース初期の反応

0.0.1が完成したので、早速リリースしました。しかし、予想どおりというべきか、やはり利用者からの反応がありませんでした。リリースしただけで、アピールしたわけではないので、当然知ってもらう機会もありません。ただこの当時はokuyamaを作ることに面白みを感じていたため、宣伝的な活動はほとんど行わず開発に集中していました。

継続開発

そして次に取り掛かった機能は可用性の部分でした。一般的にNOSQLのデータベースは高い可用性を備えています。可用性とはシステムの継続性を指しますがNOSQLの場合は大きく次のような機能を指します。

・データベースを構成する一部が故障しても全体として停止しない
・故障した場合のシステム側の対応が自動化されている

このそれぞれの具体的な機能が以下になります。

・データベースを構成する一部が故障しても全体として停止しない
　→冗長構成を作ることが可能であり、その機能を標準で備えている
・故障した場合のシステム側の対応が自動化されている
　→故障時に故障ノード自動的に切り離すフェイルオーバーを行い、冗長化された別ノードを利用し処理を継続する

初期のokuyamaにはこれらの機能はなかったため、それぞれを実装し始めました。ただこれらの機能は当然Javaの機能として存在するものではなく、自身でそのアルゴリズムを考えて実装する必要がでてきました。

そこで行ったことがほかのNOSQLのしくみを理解することでした。すでにOSSとして公開されているNOSQLにはこれらの機能が搭載されているものが多く、そこから実装のアイディアを得ることができると考えました。

そこでアルゴリズムを最も参考にしたのがCassandraでした。Cassandraは個々のノードが独立して稼働し、SPOFの存在しない高い堅牢性を備えたNOSQLデータベースです。データ構造はokuyamaとは異なりカラム型を採用していますが、冗長化部分は参考になると考えました。

Cassandraを調べるうちにNOSQLの冗長化のセオリーのような次のしくみが見えてきました。

「冗長化のしくみ」

・データをあらかじめコピーしておくことで、データ保存ノードの障害時にデータ損失率を減少させる
・ノード障害時に迅速に切り離しを行い処理の持続性を高める

上記の2つの機能を行うことによりノード障害発生時も処理が継続できる、それがわかりました。

このように既存のNOSQLの影響を強く受けてokuyamaの開発は進みました。すでにいくつものNOSQLが登場していたため、開発の方向性などに迷うことは少なく進めていくことができました。

開発での課題

okuyamaを実装するうえで最も苦労したことにテストがあります。その中の2つの課題を紹介します。

自動化

初期のテスト作業ですが、恥ずかしながらすべて手作業でした。テストユニットなどを使わず手動でデータを登録し、削除し、取得し、すべて正しい結果になるかを試していました。そしてそこで出てきた課題が「これは寝れないな……」でした。

たまらず自動化に踏み切りました。単純にデータを登録して結果を確認する類のものは、すべてJavaで実装して自動化しました。そして冗長化などの機能があるため、自動的にノードを停止、起動させるバッチなどを作成し、フェイルオーバーやリカバリなどのテストも徐々に自動化を行いました。

今になって振り返ると自動化は開発初期の時点で着手するべきでした。どうしても処理と結果が単純なため、自動化の手間を惜しんで手動に頼ってしまい、自動化への着手が遅くなっていたのです。まさに「急がば回れ」な課題でした。

性能テスト

機能の妥当性確認はテスト自動化によりかなり手間を減らすことができたのですが、okuyamaのテストでもう1つ重要なのが性能テストでした。開発前からNOSQLの性能には非常に関心があったので、このテストがある意味メインでした。テストクライアントを並列化し性能を計測するわけですが、okuyama自体の性能を見る際に1台のテスト機の中でテストプログラムとokuyamaを動作させると、すぐにリソース不足になってしまいました。これでは性能の計測になりません。そこで複数のPCを使ってテストを実施するのですが、PCを何台も買う資金があるわけではありません。自分で使っていたBTOデスクトップやノートPCなどをLANケーブルでつなぎ、手作り感満点な環境でテストをしていました(**写真1**)。

ただこの過程で低い性能のマシンを使った場合の挙動や、LANケーブルやハブが性能のボトルネックになりえることがわかりました。たとえばノード間の通信ネットワークのタイムアウト時間をokuyamaの挙動に合わせて可変にしたのは、ここでの経験が元になっています。後に筆者の会社にテスト環境を構築した際に、ノウハウとして非常に役立ちました。

▼写真1　自宅での開発環境

okuyamaの啓発活動

知名度への悩み

okuyamaの開発は技術的な難点はいくつも出てきましたが、先に述べたとおりすでに参考となるNOSQLがたくさんあったので、つまずきつつも進めていくことができました。しかしそのころ新たな課題が出はじめました。それはokuyamaの認知度です。あくまでの趣味で開発し公開しているソフトウェアなので、第三者かが宣伝してくれるわけではありません。公開前は、多くの人の目に触れることになるので少し怖くもあり、足踏みしていましたが、いざ公開してみるとそんなことは、まったくありません

でした。そこで認知度向上のための啓発活動を始めることにしました。

啓発活動を開始

基本的にはNOSQLのイベントやオープンソースカンファレンスに代表されるようなOSS関係のイベントでokuyama紹介の時間をいただき、セミナーの実施や時にはハッカソンなども行いました。これらのイベントへの参加によりokuyamaの知名度も徐々にではありますが上がっていきました。初期のイベントではWebサービスのように見た目を伴わないシステムであるため、イベント会場にタワー型のサーバを持ち込んでデモをさせてもらったりしていました。最初のデモのときはまったくうまくいかず冷や汗をかいたことを、今でもよく覚えています。

またこういったイベントに参加することで参加者の方々からokuyamaに対しての意見をもらえたことも非常に良かったです。参加者からもらった意見で次期開発内容が決まるということも頻繁にありました。

事業化

開発を継続的に続け、啓発活動を行っている中で自然と所属会社のメンバーにもokuyamaの名前が知られるようになりました。またその頃、セミナーなどでokuyamaを知った別の企業の方などから、自社サービスで使用したいというオファーももらいました。そこで所属会社で、正式にokuyamaの事業化・推進が決まりました。

okuyamaの機能紹介

okuyamaで何ができる?

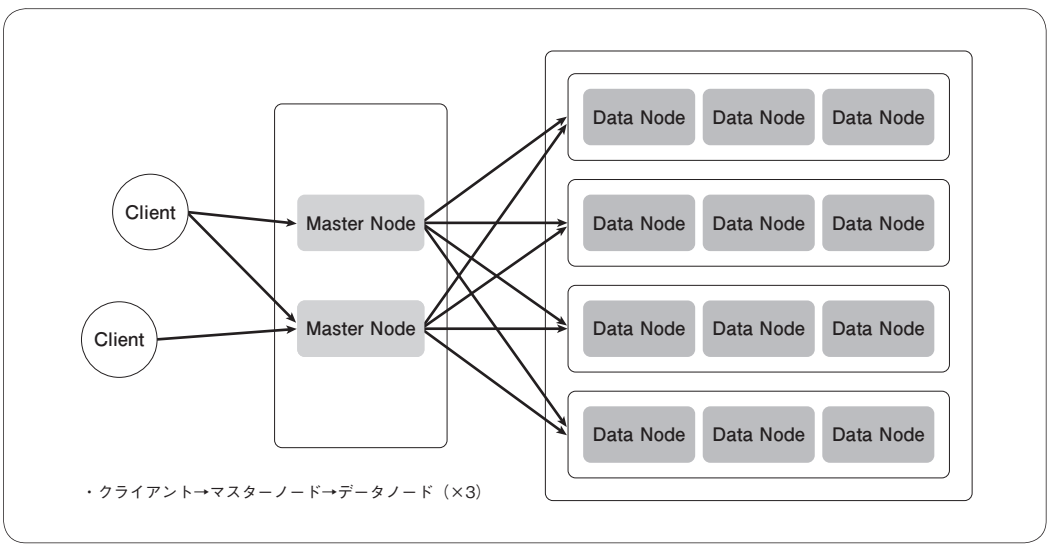
さて、開発動機から開発経緯、公開してからの活動など、おもに今までのokuyamaの経緯を紹介しました。これからは機能を詳しく紹介します。

okuyamaの基本機能

okuyamaは先ほど紹介した通り既存のNOSQLを参考に開発してきた経緯もあり、基本的なNOSQLの機能はほぼ備えています。まずそのような基本的な機能を紹介したいと思います(**図2**)。

okuyamaは大きくDataNode、MasterNode、Clientの3つの要素で構成されます。それぞれ

▼図2　okuyamaの全体図



の役目は次のようになります。

DataNode

データを保存する役目を担うノード。完全に独立して稼働し、自発的に他のノードに通信を行いません。

MasterNode

DataNodeの管理を行い、ClientとDataNodeの中継を行います。DataNodeが停止した際のフェイルオーバーや復帰時のリカバリ処理などを担当します。また、データを保存する際の保存先DataNodeの決定や冗長化のためにデータをコピーするのもすべてMasterNodeが行います。そのため、ClientからはDataNodeの存在が隠蔽され、直接DataNodeとネットワークで接続する必要がありません。

Client

okuyama専用のクライアント。開発言語別に実装が存在します。筆者が開発したものではJavaとPHPがあります。その他にも有志の方が開発されたRuby版やPHP版もあります。MasterNodeと通信を行い、okuyamaへ処理命令を発行します。接続時にあらかじめ複数台のMasterNodeを指定することで、接続中のMasterNodeが停止した場合もフェイルオーバーを行い処理を継続します。

分散アルゴリズム

okuyamaの分散方式はConsistent Hashを用いたアルゴリズムに従った方式となります(この方式はもともとキャッシュデータを保存する際によく用いられる方式で、たえばMemcachedのクライアント分散などではこの方式が用いられる)。アルゴリズム方式のメリットはデータの保存場所を記録する必要がなくリクエスト時にデータの保存ノードを即座に特定することが可能です。デメリットとしてはデータの割当先を一定のアルゴリズムにより決定するため、登録されるデータの種類によってはどうしても割り当てられるノードに偏りが出てしまい、すべてのノードに均等な件数のデータが保存されない可能性があります。この対策としてokuyamaではConsistent Hashでもっとも標準的な仮想ノードの採用とその数の調整ができます。

冗長化機能

処理継続性を高めるしくみとして、すべてのデータや構成ノードは1つ以上の冗長化が可能です。また冗長化をしておくことで障害発生時などに自動的にフェイルオーバーが行われ処理を継続します。そのため、一部のノードが停止したとしてもokuyama全体が停止することはなくすべての処理を正しく継続できます。このしくみによりokuyamaは単一障害点(SPOF：Single Point of Failure)が存在しないデータベースになっています。また、障害ノードが復帰した際のデータ同期などのリカバリ処理もすべて自動化されているので、障害復帰は当該ノードをただ起動するだけで復旧できます。

スケールアウト機能

処理能力を高めるしくみとして動的なノード追加をサポートしています。これはMasterNode、DataNodeどちらも追加できます。またノード追加時のどのタイミングであってもすべての処理を継続できるので、システムを運用しながらokuyamaの処理能力を徐々に高めていくなども可能です。また、ノードを追加した場合のデータのリバランシング(再配置)もすべて自動的に行われます。

データ保存方式

データを保存するストレージにさまざまな特性が用意されています。データベースの利用シーンを考えた場合、大容量を保存し続けたいシーンと、高速にデータの出し入れを行いたいシーンで大きく分別できると思いますが、その2つ

▼表2　データ保存方式

Key	Value	永続性	特性
メモリ	メモリ	なし	高速なアクセス性能を持ったストレージです。ただし、保存されたデータはメモリのみに保存されるため、永続化されることはありません。おもにアプリケーションキャッシュなどに利用される想定です
メモリ	メモリ	あり	高速なアクセス性能を持ったストレージですが、保存されたデータは操作記録ログと言われるログに記録され、再起動時に復元されます。おもにアプリケーションのキャッシュやセッション情報などに利用される想定です
メモリ	ディスク	あり	Keyをメモリとし、Value部分をディスクに書き出すことでレコード数が比較的多く、容量の大きなデータを保存することに向きます。ディスクを使うことで速度が低下しますが、指定されたKeyに応じてディスク上のどこに保存されているかを探す際に、メモリ上にKeyとセットでValueのディスク上の位置を保存しているため、ディスクへのアクセスは一度で済むようになっています。そのためある程度の高速性を維持することが可能です。とくにディスク上のランダムリードが高速なSSDなどの場合、高速に動作する設計となっています
ディスク	ディスク	あり	Key、Valueともにディスク上に保存されるため、メモリを消費せずに大容量のデータを保存することができます。保存可能な容量はディスクの上限までとなりますが、アクセス速度は全ストレージ上でもっとも低速になります

の利用シーンに用途別のデータベースソフトウェアを用意するのではなく、okuyama1つで実現できるようにと考えてこの機能を搭載しました。機能としてはKeyとValueそれぞれを保存する場所をメモリもしくはディスクのどちらかを選択できます。ストレージの種類とそれぞれの特性は**表2**のようになります。

ユニークな機能

okuyamaにはいくつかのユニークな機能が実装されていますが、それらを紹介します。

Tag機能

okuyamaはKVSなので、データ構造はKeyとValueからなるシンプルなものです。アクセス方式がシンプルなため、扱いやすい反面かゆいところに手が届かいないところもあります。たとえばKeyなどをハッシュ値などで生成しValueを登録した場合、同様のハッシュ値が作れない限り、登録したデータは二度と取り出せなくなります。SQLでいうところのあるテーブルのデータをすべて取得するようなことができません。

そこでokuyamaはTagというデータ構造を持っています。これはKeyのように全データ中でユニークである必要はなく、複数のKey-Valueに対して横断的に付加することができるグループ化のKeyのようなものになります。Tagを使うことで登録するデータをグループ化して取得することが可能になります(**図3**)。

また、Tagは1つのKey-Valueに無制限に付加することが可能なため、用途に応じて複数個付加することで利用側のユースケースに沿ったグループ化が可能です。

プロトコル

okuyamaへのアクセスは最適化された独自プロトコル以外に、Memcachedプロトコルでのアクセスが可能となっています。そのため、okuyamaにMemcachedの代わりをさせることで冗長化された分散キャッシュを作成することができます。ただし、Memcachedクライアントには接続中のサーバが停止した際に、別のサーバへフェイルオーバーする機能がないため、筆者がMemcachedの代わりとしてokuyamaを利用する際はロードバランサを前面に配置し、フェイルオーバーできるようにしています。またこの2つのプロトコルそれぞれで登録した値は互換性があるため、どちらのプロトコルのクライアントからも取得可能です。

▼図3　Tag機能

Key	Tag	Value
memcached	KVS	“The World’s most popular open source kvs”
MySQL	RDBMS	“The World’s most popular open source database”
okuyama	KVS	“Distributed Key Value Store Implemented in Java”

タグ“KVS”を指定することで、｛“memcached”，“okuyama”｝に関係するキーをまとめて取得できる

全文検索機能

KVSは通常Keyでのアクセスに特化しているため、Valueを使った検索を行う場合別途Apache Solrなどを用いて検索のしくみを外部に構築する必要があります。そこでokuyamaにはValueによる検索機能を実装しました。検索のしくみとしてはValueを1件ずつ調べていくのではなく、事前に検索Indexを作成します。そのため、検索に関しては高速な応答を返すことが可能です。もちろん日本語にも対応しています。

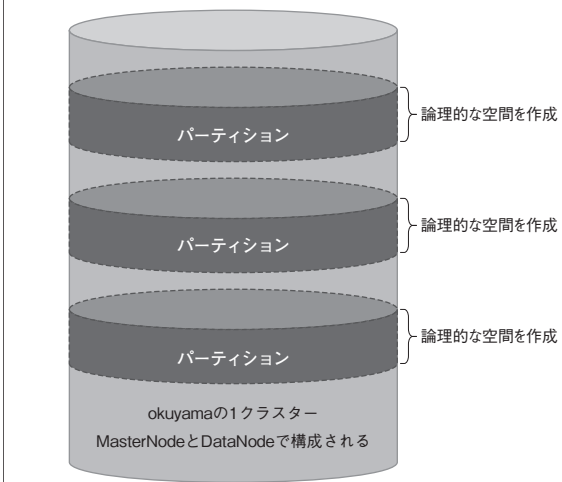
また、全文検索で問題となる意図しないデータを検索結果に含んでしまうノイズ問題への対応として、グルーピングの概念を持っています。これは検索Index作成時にグループ名を指定することで、検索時にグループの範囲内で検索を可能とするものです。たとえば、Twitterへの投稿情報とFacebookへの投稿情報を登録し検索Indexを作成した場合に、どちらか一方の情報群に対してのみ検索可能です。

パーティショニング機能

パーティショニング機能はKVSの持つデータ構造上の問題を緩和してくれます。通常RDBMSなどを利用する場合スキーマやテーブルなどの概念でデータ領域を分割すると思います。しかしKVSはKeyの重複が許されないため、どれだけ大きなクラスタを構築しても必ずKeyはユニークである必要があります。そのため、利用者は設計時にかならずKeyを作成するルールを決定する必要があります。

しかしテスト環境や開発者の環境などで同じKVSを複数の環境から共有利用すると、このKeyの重複が問題になってしまいます。そこでokuyamaではパーティションと呼ばれる領域を構築する機能をもっています(**図4**)。この機能を使えば1つのクラスター内で同一Keyを使っ

▼図4　パーティション機能



た場合でもパーティション単位で分離されるため、Keyの重複が可能です。開発環境とテスト環境の共存などで有効な機能になっています。

とくに力を入れて実装した機能

保存データサイズとメモリ量について

データ保存方式の部分で説明したとおりokuyamaはKey-Valueそれぞれをメモリに保持可能です。しかしこれには大きな問題がありました。たとえばKeyとValueを保存する際に最も速度を考えて保存すると、バイト情報でMapなどの連想配列にそのまま登録することになります。そうすると多少の誤差が出たとしてもJavaに割当てたメモリサイズ程度のKey-Valueが保存できます。ただ保存できる量を計算すると、1GBのメモリに対して5KB程度のKey-Valueを保存しようとすると、おおよそ20万件程度しか保存できません。キャッシュやSessionなどのデータ保存することを考えるとこの件数は少なすぎます。

Valueを圧縮する

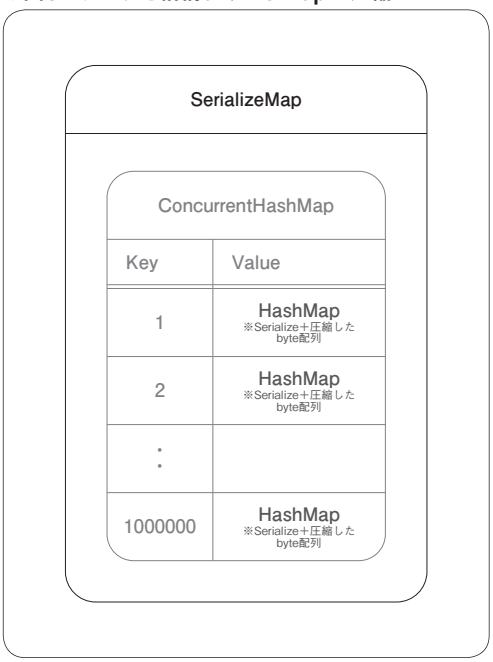
20万程度のデータを保存するのに1GBのメモリを消費することになると、100万のユニークユーザを持つWebサービスのSessionに採用した場合、最大で5GB以上のメモリが必要な計算になります。メモリは一昔前と比べると相当安価にはなりましたが、まだまだ容量単価を考えると高価です。そこで考えられる対応策としてデータの圧縮があります。特に圧縮を行うことでWeb系システムがSessionやキャッシュなどで用いることが多い文字データは非常に効率良く圧縮されます。

そこでまずValueだけをZipアルゴリズムで圧縮する機能を実装しました。しかし、Valueのみを圧縮しただけでは思ったほどの効果を得ることができませんでした。そこであるテストをしました。Valueに登録するJavaのString型の文字列の長さを変化させ登録を行い、文字の長さと総容量の関係を調査しました。すると、ある一定の文字の長さを半分にしてももともとの長さの場合の2倍の容量を保存することはできませんでした。そこである仮説を立てました。実際は文字の長さ分でメモリを専有している部分よりもString型やデータを格納しているMap型が多くのメモリを使用しているのではないかと考えたのです。

SerializeMap機能

その仮説をもとにValueを単体で圧縮するのではなく、データを格納しているMapのバケット(要素の入る個所)自体を圧縮することとしました。そこで行った実装が**図5**の実装です。これはMapのバケット数を一定しか用いず、1つのバケット内にさらにMapを格納するという実装です。ですが、ただ親のMapのValueをMapにしたのでは何も圧縮されないため、この子のMapをシリアライズした直列化情報をZipにて圧縮する実装になっています。これによりある程度の集合で圧縮され、型情報などもすべて圧縮されるため、非常に高い圧縮効率を

▼図5　データを格納しているMapの圧縮



実現できました。ただしデータのアクセスごとにこの圧縮、解凍やシリアライズ、デシリアライズが行われるので、速度としては低速になります。しかしディスクを使う場合に比べればまだまだ高速に稼働するため、キャッシュなどの使用にも十分活用可能です。

実際の利用事例の紹介

さてここまではokuyamaの基本機能やユニークな機能の紹介をしてきましたが、ここからは実際にokuyamaが利用されているシーンを紹介したいと思います。

キャッシュデータベースとしての利用

最も代表的な利用方法になるのはアプリケーションが利用するキャッシュ用データベースとしてです。これはアプリケーションが利用するメインのデータベースが、検索時などの負荷により応答速度が劣化する場合などにとくに有効です。利用方法は比較的簡単で、一度メインのデータベースに検索を行い、そこで得た結果を検索パラメータなどをKeyとしてokuyamaに登録します。そして再検索時はまずokuyamaに問い合わせ、結果があればそれを返し、なければ従来どおりメインデータベースへ検索を行います。ごく単純な実装ですが、効果は絶大です。とくに検索が集中しがちなEコマースサイトのセール時などは、特定のページへのアクセスが集中します。そのページが利用するデータを取得する部分がキャッシュ化されるため、多くの検索がキャッシュ化されメインデータベースへの負荷の低減につながります。

この利用方法ではokuyamaのストレージ保存方式はKeyもValueもメモリを選択します。速度を重視した利用方法のため、データの永続性よりも速度を優先した選択になります。

画像ファイルデータベースとしての利用

参照頻度の高い画像やCSS、JavaScriptなどのWebページで利用するファイルを格納するストレージとして利用するケースです。画像などのデータはサイズも大きいため、負荷が広範囲にわたって分散して発生すると、ディスクへの参照負荷が問題になってきます。

そこでそうしたデータをあらかじめokuyamaに格納し、参照時にokuyamaから取得することで画像配信を担当するWebサーバのディスク負荷を軽減しようとするものです。これはokuyamaがデータを冗長化して持つことができるため、データ参照時はこの冗長化したデータも活用して応答するためです。1つのデータファイルがokuyama側では実際3台のノードで担当することが可能となり自然とディスク負荷も分散され応答速度も向上します。

この利用方法ではokuyamaのストレージ保存方式はKeyがメモリ、Valueはディスクを選択しています。これはある程度サイズが大きなデータであるため、メモリに全て格納するのは難しく、速度と容量のバランスを考えた選択になります。

全文検索エンジンデータベースとしての利用

okuyamaの持つ全文検索機能を利用し全文検索エンジンのデータベースとして利用するパターンです。Webクロールを伴うようなWWW用の検索エンジンではなく、ユーザの持つデータベースのデータを対象とした検索エンジンになります。例を挙げるとEコマースサイトなどの商品データなどがそれになります。全文検索機能のもつグルーピングの機能を使い、商品の名前や商品説明などを別の項目として全文検索することで、高速に目的の情報にたどり着けます。また、検索Indexはどうしても膨大になりがちですが、スケールアウトの特性をもつためデータ量の増加にあわせてokuyama側を追加することでデータの増加にも対応可能です。

この利用方法ではokuyamaのストレージ保存方式はKeyもValueもメモリを選択しますが、SerializeMapですべてのデータを圧縮します。

速度を重視した設定ですが、検索Indexデータが大量になるため、圧縮を行うように設定しました。

今後考えているokuyamaへの変更など

ここからは、今後の開発予定などを紹介します。

次期バージョン

ここまでにご紹介したとおり、分散KVSとしての基本機能はある程度実装が完了したため、今後は機能追加と周辺ソフトウェアの開発に注力したいと考えています。執筆時のokuyamaの最新バージョンは0.9.4になりますが、直近でリリースを予定している0.9.5に実装される代表的な機能を2つ紹介します。

List型の追加

Key-Value型の以外のデータ構造としてList型を追加します。方式としては双方向リストで実装されており先頭および末尾へのアクセス時間が常にO(1)になるように実装されているListです。このリストを使うことで高速なFIFOやFILOアクセス可能なジョブキューなどを構築できます。

全Keyの取得

今まで最も多くの追加リクエストがあった機能、すなわちKeyの全件取得を搭載します。この機能を使うことで開発やテスト時に、プログラムから意図したKeyが登録されているかをすべてのKeyをデバッグして調べることが可能になります。

周辺ソフトウェア

周辺ソフトウェアとしては0.9.4より含まれるようになったokuyamaFuseが代表的です。これはokuyamaをストレージとしたLinux上のユーザファイルシステムです。まだβ版ですが、ランダムI/Oでの検証ではHDDに比べて非常に高速なI/O性能が出ることがわかっています。0.9.5では性能向上と安定性向上のための改修が主になっています。

今後の予定「管理系機能拡充」

okuyamaが最も弱い部分になる管理機能の拡充を実施していきたいと考えています。コマンドベースではバックアップやノードの追加、情報取得などはそろっていますが、管理系ソフトウェアであるMuninやZabbixなどとの連携はできないので、デファクトな管理系ソフトウェアとの連携を、今後の開発課題としたいと考えています。

おわりに

いかがだったでしょうか？今回初めてokuyamaを開発した動機から開発過程での課題や啓発活動、okuyamaの機能を通して、紹介させていただきました。

振り返ると、開発当時から壮大な目標を立てて作り上げてきたものでは決してなく、その時々の環境に影響を受けて進んできたことがわかります。今後もトレンドに影響を受けつつokuyamaの開発を続けます。よろしくお願いします。SD

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。

2013年9月号

第1特集
今からはじめる
sed/AWK再入門

第2特集
開発するなら やっぱりMacですよね?
9人9色のデスクトップ拝見＋新OS傾向と対策

一般記事
・小規模プロジェクト現場から学ぶJenkins活用（3）
・最終段階に入ったFedora 19

1,280円

2013年8月号

第1特集
理論から実践まで
3Dプリンタが未来を拓く理由

第2特集
バグを狙い撃つ技術
システムを見通す力でソフトウェア開発を楽にしませんか!

一般記事
・小規模プロジェクト現場から学ぶJenkins活用（2）

1,280円

2013年7月号

第1特集
データの価値を見直しませんか?
ここからはじめるデータ分析学習

第2特集
自分の定規を持っていますか?
ボトルネックを探れ! 「ベンチマーク」活用テクニック

一般記事
・小規模プロジェクト現場から学ぶJenkins活用

1,280円

2013年6月号

第1特集
わかった人だけメキメキ上達
ちゃんとオブジェクト指向できていますか?

第2特集
研修じゃ教えてもらえない!?
あなたの知らないUNIXコマンドの使い方

一般記事
・リアルタイム分散処理「Storm」ほか

1,280円

2013年5月号

第1特集
IT業界ビギナーのための お勧め書籍55+α
新しい季節に君へ

第2特集
覚えておきたい、ちゃんと使いたい!
正規表現をマスターしていますか?

一般記事
・バーチャルネットワークコントローラ2.0開発の実際

1,280円

2013年4月号

第1特集
僕（私）の言語の学び方
裏口からのプログラミング入門

第2特集
オブジェクト指向再入門
ソフトウェア開発に効くSmall Objectをご存じですか?

特別企画
・スクウェア・エニックス＋Skeed 「ゲーム開発の舞台裏」

1,280円

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%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。

マニュアルどおりでホントに使える?

小規模プロジェクト現場から学ぶJenkins活用

第4回 Skypebotの制作とJenkinsからGUIツールを使う方法

今回は、まず第2回で紹介したSkypebotをもう少し詳しく解説していきます。それから、GUIツールしかないけれど自動化したいという状況をどうやって解決したかについて紹介します。

嶋崎 聡(しまざき さとし)(株)XVI Twitter @sato_c

SkypebotをRubyで作る

第3回までで、Skypebotを使ってJenkinsの運用をしやすくする方法を紹介しました。誌面の都合で駆け足になっていましたので、今回はWindows環境に特化してしまいますが、Ruby4SkypeやSwin.soといったライブラリのインストールもまとめて紹介、運用設定についても説明します。今回紹介するSkypebotは、グループチャットを監視してチャット上に送られる内容に応じてJenkinsのJOBを起動するものです(図1)。

環境を準備する

Skypebotを使うためには、Ruby環境といくつかライブラリが必要です。

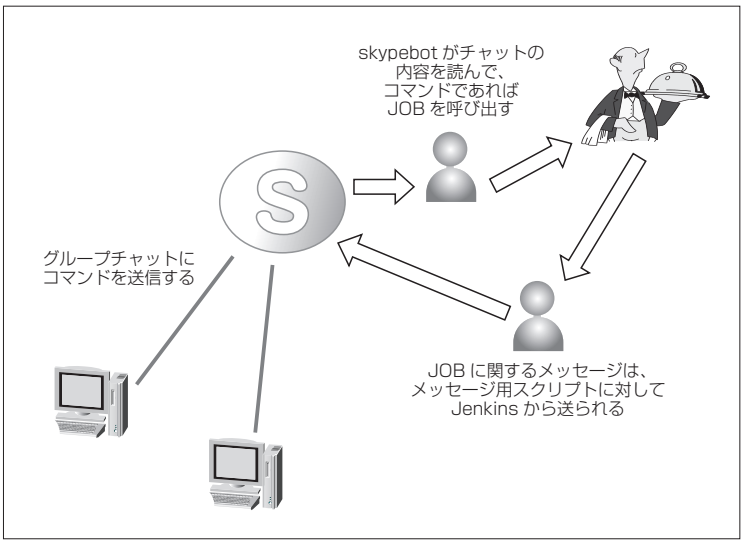
●Ruby

筆者の環境では1.8.7を使っています。パッチレベルは一番数字の大きいものにしてください[注1]。なぜ、1.8.7という古いバージョンを使っているかというとSkypebotで使っているSkype4RubyとSwin.soを問題なく使えるようにするためです。RubyのWindows環境用バイナリをダウンロードできるRubyInstaller[注2]からインストールしてください。Cygwinでは、Setupの段階で1.8系を選んでインストールしてください。

●Swin.so

Ruby4Skypeで必要になるライブラリです。RubyBinaries[注3]からダウンロードしてsite_rubyの

●図1 Skypebotの動作イメージ



注1) 2013年7月現在はp374
注2) http://rubyinstaller.org/downloads/
注3) ftp://ftp.ruby-lang.org/pub/ruby/binaries/

下にコピーします。

RubyBinariesにはCygwinやWindows、mingwなど環境に応じたライブラリ用フォルダがありますので、自分の環境に合ったフォルダを選択してください。extというフォルダにある「vrswin」で始まるバイナリをダウンロードします。このとき、最後に1.8と付いているのが1.8処理系向けのバイナリです。1.6系用も同じ場所にありますので間違えないようにしてください。

ダウンロードしたファイルはWindows向け環境では"lib/ruby/site_ruby/1.8/i386-msvcrt"、Cygwin環境では"/usr/lib/ruby/site_ruby/1.8/386-cygwin"というフォルダにコピーしましょう。もしもRubyが見るフォルダがこの場所以外になっている場合は、正しい場所にインストールしなおしてください。Rubyがデフォルトでライブラリを読み込む場所を確認するには、コマンドラインから次のように入力してください。これで出てくるsite_ruby/1.8以下のフォルダに各環境のフォルダがありますので、そこにコピーします。

```
ruby -e "puts $:"
```

●Ruby4Skype

RubyからSkypeをコントロールするために使うライブラリです。これは次のようにgemを使ってインストールします。

```
gem install Ruby4Skype
```

大文字小文字を正確に書かないと正しいものがインストールされない可能性がありますので注意してください。

インストールできたら、環境は準備完了です。

Skypebotの動作

各処理の説明をする前に、準備した環境がどのように振る舞うのかの全体像を把握しておきましょう。

SkypebotはRuby4Skypeを使って、直接Skypeを操作します。Skypebotの動く環境でSkypeを起動しておいてください。テスト時は自分のアカウントでもいいでしょうが、運用時には専用のアカウントを作るといいでしょう。

Ruby4SkypeがWindowsのメッセージ処理を行い、Skypeのウィンドウを操作します。実際はSkypeクライアントでチャットの文章として文字列を送受信しますので、Skypebotの動いているPC上でSkypeを起動しておかなければSkypebotは動作しません。

Skypebotが起動するとSkypeのグループチャットに接続します。Skypebotはグループチャットに流れてきたコマンドに応じた処理を行います。コマンド以外の文章はすべて無視しています。グループチャットを使っている理由はJenkinsに仕事をさせたい人がSkypebotに対して送ったコマンドを、それ以外の人からも確認できるようにするためです。また、Skypebotはコマンドの実行に関するメッセージやヘルプなど、固定の文字列を返すようになっています。それ以外の文章を送りたいときもありますので、後述するチャットへのメッセージ送信用プログラムも動かしています。

それでは各処理を説明していきます。

●Skypeへ接続する

Skypeに接続するには、まずRuby4Skypeの初期化を行い、そのあとで接続するための関数を呼び出します(**リスト1**)。

接続が完了したら、次にどのチャットに常駐して監視するかを決めます。ここではトピックに「Jenkins」と付いているものにしています(**リスト2**)。

まず、トピック名とグループチャットに接続している人数を調べます(**リスト2**の003～008)。メンバー数などは実際の動作には関係ないのですが、デ

●リスト1　Ruby4Skypeの初期化

```
001 : require 'rubygems'
002 : require 'Skype'
003 : require 'kconv'
004 : require "socket"
005 :
006 : Skype.init 'Skypebot'
007 : Skype.start_messageloop
```

バッグも兼ねてここでは残しました。接続するチャットを決めたら(**リスト2**の010～)、接続確認の意味もかねて「接続しました」というメッセージを送ります(**リスト2**の013)。グループチャットにメッセージが送られたら成功です。

●リスト2　監視対象のチャットに常駐

```
001 : chats = Skype.searchRecentChats
002 : chats.each { |n|
003 :   topic   = n.get_topic
004 :   members = n.get_members
005 :
006 :   no = members.size
007 :
008 :   print"Member:#{no} #{topic.tosjis} n"
009 :
010 :   if topic =~ /Jenkins/
011 :     puts( n )
012 :     $chat = n
013 :     $chat.send_message(" 接続しました ")
014 :   end
015 : }
```

●Skypebotにコマンドを理解させる

Skypeへの接続ができたら、次にグループチャットに流れてくる文字を受け取って、内容の解析をしていきます。すべての文字列を解析していけば話し言葉で処理を動かすこともできると思いますが、通常時にそこまで処理が必要ないので、「@」の後ろに続く半角英数(大文字小文字関係なく)はコマンドとして認識するようルールを決めました。チャットから取り出した文字列の行頭が@であればコマンドとして処理、それ以外は無視という流れが作れます。

コマンドは次のように決めました。コードは**リスト3**です。

●リスト3　コマンド分解処理

```
001 : WGET="wget -o NUL"
002 : JENKINS_HOST="http://localhost:8080/"
003 : JOBS={
004 :   'cnv'    => 'DATA_Compile',
005 :   'color'  => 'DATA_ReduceColor',
006 : }
007 : Skype::ChatMessage.set_notify do |chatmessage, property, value|
008 :   if (value == 'RECEIVED' || value == 'SENT') && property == :status
009 :     bd = chatmessage.get_body
010 :     if /^ @/ =~ bd
011 :       bd = bd.slice!(1,bd.size)
012 :       if bd.downcase == 'help'
013 :           chatmessage.get_chat.send_message " @cnv  : データをまとめます "
014 :           chatmessage.get_chat.send_message " @color: 減色します "
015 :       elsif
016 :         if JOBS.key?(bd)
017 :           job_name=JOBS[bd]
018 :           chatmessage.get_chat.send_message "#{bd} を開始 "
019 :           kick_job="#{WGET} #{JENKINS_HOST}/job/#{job_name}/build?"
020 :           system(kick_job)
021 :         end
022 :       end
023 :     end
024 :   end
025 : end
026 : $tcp = TCPServer.open(6001)
027 : while true
028 :   Thread.start($tcp.accept) do |s|
029 :     while s.gets
030 :       $chat.send_message($_)
031 :     end
032 :   end
033 :   s.close
034 : end
```

例)

@help …… コマンドの内容を表示
@cnv …… 複数の画像データから動画を作成
@color …… 画像を減色

連想配列(**リスト3**の003～)のキーをコマンドにしておいて、そのキーが存在するかをチェック(**リスト3**の016～)します。こうしておけば(自分で)ループを回して配列をチェックする手間もありませんし、増設する際にもどんなコマンドがあるかをすぐに確認できます。

キーが存在したら、そこに紐付けられているデータを取得します。ここにJOB名が書かれていますので、その値を使ってJOB呼び出し用のURLにし、kick_jobを作ってwgetを呼び出します(**リスト3**の020)。デバッグ中は、systemの前にpやwarnでkick_jobを表示すれば、文字列がどうなっているか確認できます。

●Skypebotに結果を報告させる

JOBが終わったときやエラーが出たときなど、JOB実行以外でチャットに対して文字列を送信する部分が必要になることもありますので、その部分について考えます。Skypeへの接続などはSkypebotと同じになります。監視する対象をチャットではなくTCPポートにして、書き込まれた文字列はとくに加工せずにSkypeへ送信することにしました。

接続処理はSkypebotと変わりませんので、そちらを見てください。**リスト4**はTCPポートから受け取った文字列をそのまま送信する部分です。ポートへの書き込みがあったら、文字列を取り出してsend_messageします。終わったらポートを閉じるといった処理を繰り返しますので、起動後はそのままプロセスを常駐させて同じ処理を繰り返します。

送信側も内容はシンプルです(**リスト5**)。ポートが開けたら(**リスト5**の005)コマンドライン引数で指定された文字列をすべて取り出して連結(**リスト5**の009～)して書き込みます(**リスト5**の014)。begin～rescue～else～endは例外処理で、begin内の処理が失敗したときにrescue以下を、成功したらelse以下を実行します。

実際の利用では呼び出しを簡単にするために、次のようにバッチファイルからRubyスクリプトを呼ぶ形にしました。

例) バッチファイル例

```
001 : ruby write_to_skype.rb %*
```

送信プログラムの応用として、Rubyスクリプト内でエラー通知などで使う場合は、直接TCPポートを開いて、エラーメッセージを送信します(**リスト6**)。**リスト6**の005～011で、TCPポートが開けなかった場合(受け側のスクリプトが起動していない、など)は、標準出力にエラーを出力して終わります(**リスト6**の008)。

複数の処理でも単一のスクリプトでもやろうと思えばできると思いますが、取り回しを楽にするために複数スクリプトに分けて実装しました。

●リスト4　受信側

```
001 : $tcp = TCPServer.open(6601)
002 : while true
003 :   Thread.start($tcp.accept) do |s|
004 :     while s.gets
005 :       $chat.send_message($_)
006 :     end
007 :     s.close
008 :   end
009 : end
```

●リスト5　送信側

```
001 : HOST = "localhost"
002 : PORT = 6601
003 :
004 : begin
005 :   sock = TCPSocket.open( HOST, PORT )
006 : rescue
007 :   puts "error : #$! n"
008 : else
009 :   puts "sending...  n"
010 :   cmdline = ""
011 :   ARGV.each do |s|
012 :     cmdline += "#{s} "
013 :   end
014 :   sock.write(cmdline)
015 : end
```

●リスト6　スクリプト改造例

```
001 : def  send_msg_to_bot(msg)
002 :   host = "localhost"
003 :   port = 6601
004 :
005 :   begin
006 :     socket = TCPSocket.open(host, port)
007 :   rescue
008 :     puts("error : TCPSocket")
009 :   else
010 :     socket.write(msg)
011 :   end
012 : end
```

Skypebotを実際に利用する

まず、WindowsでSkypebot用のユーザアカウントを作ります。ユーザができたら、Skypebot起動用のバッチファイルをスタートアップに登録しました。これでログイン時に自動的に起動します。Windowsでバッチファイルから常駐プロセスを作る方法がわからなかったので複数のコマンドプロンプト画面が開いていますが、コマンドプロンプトから状況を確認できるのでそのまま利用しています。

例) Skypebot起動

```
ruby c:¥server¥skypebot.rb
```

例) レスポンス用スクリプト起動

```
ruby c:¥server¥skypemsg.rb
```

SkypebotからJenkinsのJOBを呼び出す際、ユーザ認証設定を行っている場合はJOBに設定したトークンを付けないと正しく実行されません。今回のスクリプトで修正を行う場合は、**リスト7**と**リスト8**のようにします。

リスト8では、TOKEN[bd]と直接配列名を書かずに、いったんjob_nameと同じように変数job_tokenを作って書いてもいいでしょう。

●リスト7　次の連想配列を追加

```
007 : # アクセス制限時の外部起動に必要
008 : TOKEN={
009 :   'cnv'    => ' 設定したトークン ',
010 :   'color'  => ' 設定したトークン '
011 : }
```

JOBからSkypeに対して、開始時と終了時にバッチファイルでメッセージ送信用スクリプトを呼び出しています。メッセージは次の例のような環境変数を使って、JOBの名前やビルド番号、URLをまとめたものにしました。このメッセージからコンソール出力をすぐに開けるようにしたかったので、メッセージの最後にBUILD_URLをconsoleと付けて出力しています。これでURLをクリックすればコンソール出力が見られるようになりました。

例) JOB開始時に送信するメッセージ例

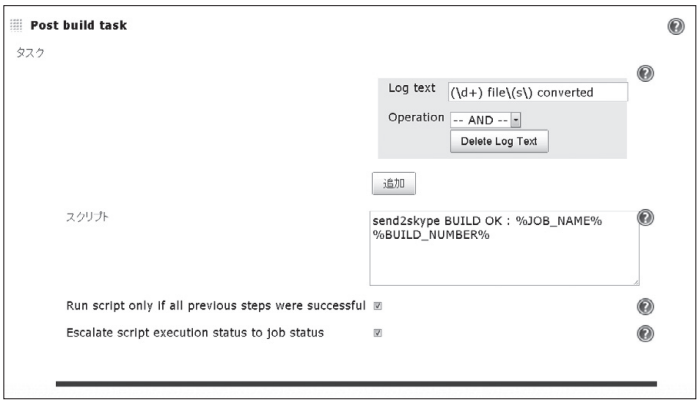
```
%JOB_NAME%#%BUILD_NUMBER%(Rev.%SVN_
REVISION%) %BUILD_URL%console
```

また、終了時にPost Build Taskプラグインで状況によってメッセージを変える処理もしています(**図2**)。

GUIツールを動かすには

JenkinsのJOBからGUIツールを動かしてもその

●図2　JOBでの設定例



●リスト8　kick_jobを次のように変更

```
028 :          kick_job="#{WGET} #{JENKINS_HOST}/job/#{job_name}/build?token=#{TOKEN[bd]}"
```

ままでは起動できず、起動できても動作が確認できませんでした。それでしばらくGUIツールの利用をあきらめてた時期もありました。しかし、そうも言ってられない状況になって、なにか方法はないかというわけで試してみたのが「UWSC[注4]」です。

UWSCはGUIのアプリケーションでの定型作業を自動化するために作られたツールです。対象となるアプリケーションの起動から、ボタンを押したり、アプリケーション側からのリアクションを待つといった動作を時系列に沿う形でマクロとして記録／再生できます。

マクロの記録と編集

マクロはテキストファイルに保存されます。まず、UWSCで動作を記録して基本となるマクロを作成します。記録方法はほとんどすべてのWindowsのイベントを記録する「低レベル記録」とクリックイベントを中心にあまり細かいイベントに関しては記録しない「高レベル記録」があります。とくに理由がない限りは高レベルで記録しておいても大丈夫でしょう。どれくらい違いがあるか比べるためにChromeのGmailタブをクリックするマクロを作りました（**リスト9、10**）。

低レベル記録では、ほぼすべての記録をとっているのでMMVというマウス移動のコマンドが大半になります。高レベルでは、もう少しわかりやすいマクロになってWindowIDをとってから、アイテム（この場合、タブ）をクリックする動作が入っています。

繰り返しになりますが、普段使う分には高レベル記録で充分です。低レベルでは実時間で動作を記録／再生していますので実際に使う場合にウィンドウの開く時間やディスクアクセスといった理由で時間がずれると正しく実行できなくなることがあるからです。高レベルでは何か動作を起こすときに必要な情報が取れるまで待たせたり、SLEEPコマンドでわざと待ち時間を取るように対策することで、低レベルよりも手間をかけずに実行時の環境変化に強いマクロを作れます。

使えるマクロを作るまで

実際に利用する場合は、まずマクロを記録状態にしてから、一通りいつもどおりの作業をしていきます。記録ができたらマクロを動かして、動きのおかしい場所を探しては直し、また動かしてを繰り返します。メニューやダイアログボタンのクリックは、次のようにCLKITEMという関数が実行されます。

●リスト9　低レベル記録

```
MMV(2117,48,390)
ACW(GETID("受信トレイ - shimazaki@xvi.co.jp - XVI Inc. メール - Google Chrome","Chrome_Widge
tWin_1"),2016,21,1601,1129,0)
BTN(LEFT,DOWN,2117,48,31)
MMV(2117,48,31)
BTN(LEFT,UP,2117,48,219)
MMV(2117,48,515)
MMV(2117,48,249)
MMV(2115,55,10)
MMV(2106,73,16)
```

●リスト10　高レベル記録

```
id = GETID("受信トレイ - shimazaki@xvi.co.jp - XVI Inc. メール - Google Chrome", "Chrome_
WidgetWin_1", -1)
SLEEP(1)
CLKITEM(id, "受信トレイ - shimazaki@xvi.co.jp - XVI Inc. メール", CLK_ACC)
```

注4）http://www.uwsc.info/

例) クリックの取得

```
CLKITEM(id, "ファイル(F)", CLK_TOOLBAR)
CLKITEM(id, "開く(O)", CLK_BTN)
CLKITEM(id, "ファイル作成(F)<#TAB>Ctrl+H",
CLK_ACC or CLK_BACK or CLK_MUSMOVE)
```

この例で言うと、(F)や(O)と付いている部分はAlt + Fや Alt + 0、3つ目はCtrl + Hを押せばその機能が使えます。こうした動作は、大抵のものがキーボードから入力できるようになっています。そこで、次のように書き換えて、なるべくクリックに頼らないような形にしていきました。

例) キー入力に書き換える

```
SCKEY(id, VK_ALT, VK_F)
SCKEY(id, VK_ALT, VK_0)
SCKEY(id, VK_CTRL, VK_H)
```

クリックのタイミングやウィンドウ位置によってクリックが有効にならないといったことはUWSCのマクロではあまり起きません。しかしまったく起こらないというわけではありませんので、自動実行に使う場合は極力こうした書き換えをしておくほうがいいでしょう。ファイルを保存する場合はCtrl + S、アプリケーションの終了にはAlt + F4といった具合に、よほどのことがない限りデフォルトの設定がありますから、失敗してほしくない動作については、なるべく安定して動作する方法を選ぶといいでしょう。

●図3 常駐スクリプトと呼び出しスクリプトの関係

JenkinsでUWSCを動かす

マクロが準備できたら、次は実際にJenkinsのJOBから呼び出すための準備をします。UWSCはマクロのファイル名を指定してコマンドラインから動かすことができます。しかし、JOBに直接UWSCの呼び出しを書いてもうまく動きません。

そこでSkypebotと同じアカウントからUWSCの起動スクリプトを常駐させて、TCPポートへアクセスがあったら起動という形をとっています。JOBからは常駐スクリプトのTCPポートに対してアクセスするように設定しました(**図3**)。

スクリプトとしては、Skypebotの常駐部分と同じような作りになっています(**リスト11**)。TCPポート8801を監視して、そこに何かアクセスがあったら、systemメソッドを使ってuwscを起動します。マクロファイルのあとに引数を付けるとコマンドライン引数としてそのままアプリに渡されます。

呼び出し側も文字列を送信するスクリプトと同じような形でポートを開いてアクセスしています(**リスト12**)。TCPポートへアクセスしてデータを読み込みます。このスクリプトでは、その読み込んだデータにWarningやErrorといったエラーメッセージが含まれているか確認します。メッセージが含まれていた場合はその行を出力して、スクリプトをエラー終了させるために「1」を返しています。

こうして呼び出しスクリプトをJOBに含めることで、GUIツールが使えるようになりました。GUIツール呼び出し以外のスクリプトについては、そのまま通常どおりに記載していきましょう。

まとめ

第2回では駆け足になっていたSkypebotのもう少し詳しい話とJenkinsからGUIツールを使う方法の1つを紹介しました。

Skypebotについては、便利なライブラリを使お

●リスト11　常駐側

```
001 : #!ruby -Ks
002 : require 'kconv'
003 : require 'socket'
004 : tcp = TCPServer.open(8801)
005 : while true
006 :   Thread.start(tcp.accept) do |s|
007 :     result = system( "uwsc.exe tool_kick.uws [ 引数 ]" )
008 :    s.puts(result)
009 :     sleep( 0.1 )
010 :     s.close
011 :   end
012 : end
```

●リスト12　呼び出し側

```
001 : #!/usr/bin/ruby -Ks
002 : require "socket"
003 : require 'kconv'
004 : HOST = "localhost"
005 : PORT = 8801
006 : begin
007 :   sock = TCPSocket.open( HOST, PORT )
008 : rescue
009 :   puts "エラー : #$! n".tosjis
010 : else
011 :   puts "【tool を起動しています(#{Time.now})】".tosjis
012 :   result = sock.read
013 :   err = false
014 :   result.each_line do |line|
015 :     next if line =~ /Warning/
016 :     err = true if line =~ /Error/
017 :     puts line
018 :   end
019 :   puts "【tool 動作終了】".tosjis
020 :   sock.close
021 : end
022 : exit(1) if err
```

うとしたら最新のRubyには対応していなかったという理由で、古いRubyをそのまま使っています。最新のものも含めて複数のRubyをRubyInstallerを使ってインストールして、必要に応じてPATHを通して切り替えています。複数のRubyを切り替えるにはrvm[注5]やpik[注6]をインストールして使う方法もあります。どちらが優れているというものではないので、実際に使う環境に合わせて選んでください。

GUIツールについては、JenkinsがWindowsのサービスで起動しているため、ユーザ側からは動作しているかどうかを画面で確認できません。そのためにユーザプロセスで動かせるような対処とハードウェアメーカーから提供されるツールにコマンドライン版がなくて困っていた末に考えた方法です。マクロの記録→再生やJenkinsの動いている環境での動作確認、JOBから呼び出して使えるかのチェックといろいろと時間がかかる作業が必要にはなりますが、その後の作業はかなり楽になります。自動化した意味はかなりあったと考えています。SD

注5) https://rvm.io/
注6) https://github.com/vertiginous/pik

分散データベース「未来工房」

第4回 スケールアウトする全文検索
――Yokozuna in Riakでログを掘る

Writer 上西 康太（うえにし こうた） Bashoジャパン株式会社 kota@basho.com

現行のRiakの機能では、複雑な条件による全文検索は難しい。次期バージョンのRiakでは、Solrとのインテグレーションによって高機能な全文検索もできるようになる予定だ。本稿では、それを実現する新しいRiakのモジュール“Yokozuna”を解説しつつ、Apacheのログ検索のユースケースを挙げ、実際のクエリを例に紹介していきたい。
注）本稿は、筆者の意向により常体で表記している。

Riakでデータを探す

本誌6月号からの連載でRiakに関してさまざまな紹介をしてきたが、Riakそのものの機能についてはあまり解説しなかった。というのも、網羅的な機能の紹介については、この連載ではなくSoftware Design plusシリーズの『データベースエンジニア養成読本』にほとんどのことを執筆したので、そちらをご覧いただきたい。

そのとき、そちらでも1つだけ記載しなかった機能が全文検索である。実はRiakにも全文検索の機能は以前から存在しており、いくつかのシステムが商用で稼働している。この既存の全文検索機能はErlangで実装されており、それ自体はとても安定している。

しかしRiakの次期バージョン2.0でそれがリプレースされ[注1]、検索エンジンにSolrを利用したものとなる。Bashoではこの新しいモジュールを“Yokozuna”と呼んでいる[注2]。Yokozunaによって、すでにあるSolrの豊富な機能と、Riakの可用性やスケーラビリティが強力な組み合わせとなることが期待できるので、その一端を本稿で紹介したいと思う。

これまでRiak内のデータを見つけて効率的に読みだすためには、全文検索を除くと、キーによるアクセス、セカンダリインデックス（2i）、キーフィルタ、MapReduceなどをうまく組み合わせてアクセスするしかなかった。そのため、

- クエリ最適化の機能はない
- 構造化されていないデータに対して効率的にデータを呼び出す手段がない

などの問題点があった。ここでは例として、ログ検索のシステムをRiakを使って構築してみようと思う。いまはWebシステムを運営するうえでアクセス解析は必須であり、証拠保全の役割も果たす。その一方で、そのためのログの保存や解析は運用者にとって大きな負担になり得る。ログの収集にはFluentdが昨今の定番であるから[注3]、FluentdでApache httpd形式ログを収集し、Riakに保存することを考えよう。Fluentdは、いったん集めたJSON形式のログをさまざまなストレージやデータベースに保存することができる。今回の構成は**図1**のようになる。

fluent-plugin-riakは、そのFluentdが集めて

注1）現在の最新版Riak1.4系にもriak_searchというモジュールがあるのだが、マルチバイト文字をサポートしていない、カスタムスキーマやextractorを設定できないなどいくつか問題がある。

注2）諸説あるが、Yokozunaのレポジトリには「横綱とは、相撲のチャンピオン＝最上位であり、この製品も、与えられたクエリに対して最上位の回答を返すことを目指している」と書かれている。

注3）『データサイエンティスト養成読本』に詳しい解説があるので参照いただきたい。

きたデータをRiakに書き込むプラグインである。fluent-plugin-riakが利用するRiakのRubyクライアントは接続先のRiakのノードのアドレスを複数管理できるようになっており[注4]、Riakのノードが動作するサーバが故障した場合には自動的に他のRiakのノードへ再接続する。したがって、Riakのノードが故障した際に、Fluentd側で接続をフェイルオーバーさせるなどの手間がかかる操作は不要である。

Yokozuna入りRiakの準備

前述したとおりYokozunaはまだ開発中であり、正式版どころかビルド済みのパッケージすらリリースされてはいない。現在はまだベータ版であり、バージョンは0.8.0である。正式版ではないが、Yokozuna入りのRiakを動かすための現状の手順をここに簡単に書いておく[注5]。

事前の準備とパッケージ取得

OSはRiakが動作する64bitのものを用意しておくのがよいだろう。筆者はDebianGNU/Linuxを使用してこの手順を検証した。ビルドには、

- Erlang/OTP R15B02 またはR15B03[注6]
- Oracle Java JRE 1.6以降
- GNU make、GCC

が必要である。

まだソースコードが配布されているのみでビルド済みのパッケージはないため、**図2**のようにソースコードをダウンロードする[注7]。

コンパイルと設定、起動

取得したソースコードを次のように展開、ビルドする。

```
$ tar xzf riak-yokozuna-0.8.0-src.tar.gz
$ cd riak-yokozuna-0.8.0-src
$ make stagedevrel
```

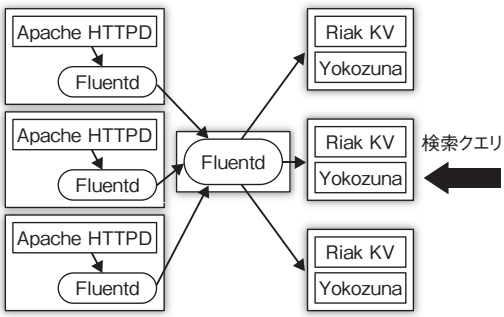
これで起動可能なRiakのノードが**dev/dev{1-5}**のディレクトリに5個作成されていることと思う。さらに、デフォルトではYokozunaはオフになっているので、設定で有効にする必要がある。各**dev/dev{1-5}/etc/app.config**のyokozunaセクションで、デフォルトを変更して**{enabled,true}**と設定する。これで、全文検索可能な状態でRiakが起動する。

また、fluent-plugin-riakのために、バックエンドをeleveldbに変更しておく必要がある。バックエンドを変更するには、設定ファイルで同様に**riak_kv**セクションで、デフォルトのbitcaskを変更して、**{storage_backend , riak_kv_eleveldb_backend }**を設定する。

注4）これは非常に便利な機能だが、いくつかある言語のRiakクライアントのなかでも実装されている言語は限られている。

注5）https://github.com/basho/yokozuna/blob/master/docs/INSTALL.md

▼図1　FluentdとRiakを使ったログの収集アーキテクチャ



▼図2　Yokozuna入りRiakのダウンロード

```
$ wget http://data.riakcs.net:8080/yokozuna/riak-yokozuna-0.8.0-src.tar.gz
$ md5sum riak-yokozuna-0.8.0-src.tar.gz
59ef004f32ef040fb8bfcec7430053d3 riak-yokozuna-0.8.0-src.tar.gz
```

注6）Erlang/OTPの最新版はR16B01だが、0.8.0では未対応である。R15B03は正確にはR15B03-1というバージョンが配布されているので、そちらを利用した方がよい。また、HiPEはオフにしたほうがよいなど、コンパイルオプションはいくつか推奨のものがある。もしもOSのパッケージインストーラがこれに該当するものでなければ、ソースコードからインストールするしかない。詳しくは「http://erlang-users.jp」を参照するとよい。そこでソースパッケージもミラーされている。

注7）余談だが、このファイルは先月号で紹介したRiak CSでホストされている。BashoがRiak CSのデモ用に構築したシステムであり、ドッグフードだ。

起動はRiakと全く同じだ。ここでは1台だけ起動しておく。複数台での運用はあとで説明する。

```
$ ulimit -n 4096
$ dev/dev1/bin/riak start
$ dev/dev1/bin/riak ping
pong
```

まずは**図3**のように、Fluentdのログを検索する内容を指定したインデックスを作成する[注8]。

ここではとくに何も明示的にスキーマを指定していないので、デフォルトのスキーマが採用される。各ノードの**dev/dev{1-5}/data/yz/**配下にスキーマ関連のファイルがあるので、詳細はそちらを参照するとよい。今回はデフォルトスキーマを使用するので必要ないが、デフォルトのスキーマで不足がある場合には、ここで自分のアプリケーションで必要なスキーマを指定する。スキーマをRiakに登録するには、

```
$ curl -X PUT \
http://localhost:10018/yz/schema/SCHEMA \
-d @yourschema.xml \
-H 'content-type: application/xml'
```

などとしておいて、インデックスを最初に作成する際にこのスキーマを指定する(**図3**)。

ここでは、fluent-plugin-riakがfluentlogというBucketしか利用しないので、さきほど作成したfluentlog_indexと結びつける。このとき、1つのインデックスに対していくらでもBucketを指定することができるため、Bucketをまた ぐ範囲の検索も可能だ。

```
$ curl -X PUT \
http://localhost:10018/buckets/ ➡
fluentlog/props\
-d '{"props":{"yz_index":"fluentlog_ ➡
index"}}'
-H 'content-type: application/json'
```

無事にこれらの設定ができているかどうかを確認するためには、**図4**のように、これまでやってきたPUTの代わりにGETを呼べば良い。これで設定は完了したので、次はデータの登録と検索について解説する。

Fluentdとfluent-plugin-riakの簡単なセットアップ

Ruby1.9とBundlerさえあれば、Fluentdとfluent-plugin-riakのセットアップは非常に簡単だ。いくつか方法はあるが、基本的にはどちらもgemで入るようになっている。

```
$ sudo gem install fluentd
$ sudo gem install fluent-plugin-riak
```

もし簡単に試したいだけなら、fluent-plugin-riakのリポジトリから取得するのがよいだろう。

```
$ git clone \
git://github.com/kuenishi/fluent- ➡
plugin-riak
$ cd fluent-plugin-riak
$ bundle install --path=vendor/bundler
```

とすると、ユーザ空間ですべてを完結させることができる[注9]。

設定ファイル(ここではfluent.confとする)のRiakの部分は次のようになる。in_tailの設定

注8) 以降は読みやすいように、URLエンコードされていないアドレスを文中で用いる。curlコマンドはエンコードされていないリクエストでも、自動的にURLエンコードしてリクエストして投げるので文中のコマンドをそのまま打ち込めばよい。

注9) 安定したものがよいのであれば、Fluentdの代わりにtd-agentを使うのも選択肢の1つだろう。td-agentはFluentdそのものなので、同じ設定ファイルで動作する。Windows対応も進められているようだ。

▼図3　インデックスの作成

```
$ curl -X PUT http://localhost:10018/yz/index/fluentlog_index
```

▼図4　Yokozunaが正しくセットアップされているか確認する

```
$ curl -i http://localhost:10018/yz/index/fluentlog_index
$ curl -i http://localhost:10018/yz/schema/_yz_default
$ curl -i http://localhost:10018/buckets/fluentlog/props
```

をApacheのログファイルへのパスにすると、ログをパースしてJSONに変換するようになる(**図5**)。

fluent-plugin-riakはRiakのProtocolBuffersインターフェースを利用する。今回は**make stagedevrel**でビルドしたのでポート番号は10017になるが、もし将来的にパッケージインストールや、別の方法で環境を構築した場合はその都度ProtocolBuffersのポート番号を指定する。**app.config**には**pb**のセクションで指定されているので、その値を使用する。**make stage**の場合はデフォルトの8087になる。

これで準備が整ったので、Fluentdを起動する。Bundlerを使っている場合は、次のようにする。

```
$ bundle exec fluentd -c fluent.conf
```

これで、Webサーバを用意しているなら、そこへブラウザでアクセスすればRiakにログが格納される。fluent-plugin-riakはバッファリングをしているので、Riakに入るには少し間があるだろう。

テストデータの生成と最初の検索

これだけ長々と手順を書いたが、今回は問題を簡単にするために、in_tailが出すJSONを擬似的にランダムで生成するスクリプトを使ってこれ以降の検証を行う。スクリプトはfluent-plugin-riakのレポジトリに含まれている**datagen.rb**である[注10]。

▼図5　fluent.confの設定(一部)
パッケージなどでインストールした場合は/etc/fluent.conf に存在する場合もある

```
<source>
  type tail
  format apache
  path /var/log/apache2/access.log
  tag apache.access
</source>
<match apache.**>
  type riak
  nodes 127.0.0.1:10017
</match>
```

datagen.rbは次のようなJSON形式のログをランダムに生成し、Riakに書き込む(実際にfluent-plugin-riakがRiakに書き込むデータは、バッファリングをしているためこれとは多少異なり、各行のログがリストになっているが、Yokozunaは問題なく検索できる)。

```
{
  "agent":"",
  "code":"503",
  "host":"175.183.161.211",
  "method":"GET",
  "path":"/moriyoshi/riaklogo.png",
  "rand_f":0.2060684945318313,
  "rand_i":23528,
  "referer":"",
  "size":"752",
  "tag":null,
  "time":"2013-08-19",
  "user":"-"
}
```

datagen.rbを動作させるためには、あらかじめチェックアウトしBundlerで準備してあったfluent-plugin-riakのレポジトリに移動し、

```
$ bundle exec ruby datagen.rb 1024
```

などと実行する。最後に指定した数字の個数だけログをRiakに書き込む。このスクリプトはシングルスレッドで動作するので、書き込みの速度が欲しい場合は複数プロセスを別々のコンソールで起動するとよいだろう。

Yokozunaは優秀で、データを書き込むときにContent-typeをapplication/jsonに指定しておくと、自動的にJSONとして展開してくれる。Yokozunaが受け取るクエリはSolrと互換なので、たとえば全件数と最初の1024件を取得するためには、

```
$ curl 'http://localhost:10018/search/
  fluentlog_index?wt=json&q=*:*&rows=1024'
{
  "response": {
    "docs": [
```

注10) https://github.com/kuenishi/fluent-plugin-riak/blob/master/datagen.rb

```
      {
        "_yz_ed": "20130818T142554 ....
        "_yz_fpn": "2",
        "_yz_id": "2013-08-18-31291_4",
        "_yz_node": "dev4@127.0.0.1",
        "_yz_pn": "4",
        "_yz_rb": "fluentlog",
        "_yz_rk": "2013-08-18-31291"
      },
    ……(中略)……
  ],
  "maxScore": 1.5108808,
  "numFound": 1802581,
  "start": 0
},
……(中略)……
```

となる。このとき"**_yz_rb**"はクエリにマッチしたデータがあるBucket、"**_yz_rk**"はそのキーを表す。したがって、これを使って元データを取得することができる。"**start**"はページネーションのためのオフセットである。また、"**numFound**"は**?q=...**で指定した検索条件にマッチする総件数を表す。これをうまく使えば、条件を絞っての集計がずいぶん簡単になったことがわかるだろう。

さまざまなクエリの例

ここでは、Solrのクエリの表現力[注11]を直接確かめるために、とくにGUIなどを使わず、HTTPでさまざまなクエリのサンプルを列挙する。curlを使ってHTTPインターフェースを叩くときのサンプルで、URL自体は同じなのでGETパラメータだけで表記する。

たとえば、8月以降のログをすべて日付ソートをしたい場合には

```
q=_yz_rk:[20130800 TO *]&sort=_yz_rk desc
```

とすればよい。**sort=_yz_rk desc**は、Riakのキーでソートするという意味だ。fluent-plugin-riakでは日付にユニークな識別子を追加してキーにしているため、この方法が使える。

Riakの値になっているJSONのなかのフィールドについてソートを行いたい場合は、そのフィールドがシングルバリューとなるようにスキーマを設定してやる必要があるが、本稿では割愛する。8月の間について集計をしたい場合は、このキーは文字列として認識されているため、**_yz_rk:[20130800 TO 20130899]**と指定してやればよい。

GETリクエストで404が返ったものをとりたい場合は

```
q=method:GET AND code:404
```

などとすればよい。返ってきた検索結果のJSONの中の"numFound"にこの条件にマッチする全件数が入っているので、これでSQLにおけるcount(*)相当のことができることがわかる。

また、SQLで集計を行う際に最も重宝する操作の1つがGROUP BY文だが、それに相当する機能も(Solrが)持っているので、Yokozunaのあるriakの場合でもできるようになる。

```
q=*:*&group=true&group.field=code
```

とすれば、全件のログをHTTPのステータスコードについてグループ分けを行って、結果はグループごとに集計されたものになる。

クエリ仕様はSolrのものと互換なので、ほかにもファセット検索やフィルタクエリ"**fq**"などさまざまなパラメータが利用可能なので、ぜひ試していただきたい。スキーマを適切に選んで設定すれば[注12]さらに多くのことができるだろう。

ノード追加・削除とスケーラビリティ

Riakのときと同様に追加・削除してよい。使い方はElasticSearchやSolrCloudとまったく異なるので注意が必要だ。逆に言えばRiakとほぼ同じなので、Riakに慣れている場合には安心だ。

注11) 実際にはLuceneが受け取るクエリの仕様である。

注12) 各ノードにファイルをSCPして……といったことは不要だ。

追加・削除の操作

ノードを追加する場合は、新しく追加するノードに、あらかじめ正しくIPアドレスなどを設定しておく必要がある。冒頭のビルドではmake stageを使ったが、パッケージを作成する場合はmake relを使うとよいだろう。これでrel/riak以下が、それだけで動作する完全なパッケージになるので、これをtarballなどにして各サーバで配布、解凍、設置するとよい。配布先のサーバでErlang/OTPなどをインストールしておく必要はない(図6)。

設定ファイルの変更は、冒頭にあったYokozunaの設定に加えて、riak_api、riak_core,riak_kvにあるIPアドレスの変更をするだけでよい(fluent-plugin-riakを使う場合はstorage_backendの変更も必要)。

コマンドは、Riakと同様に、新しく追加するノードから、すでにクラスタにいるノード(ここでは仮にriak@10.0.0.1とする)へ向けて

```
$ riak start
$ riak-admin cluster join riak@10.0.0.1
$ riak-admin cluster plan
$ riak-admin cluster commit
```

とすればよい。複数台追加する場合は、joinを繰り返すだけでよく、plan,commitは最後に一度だけ行えばよい。ノードを追加したあとも、特にスキーマの再構築などは必要ない。

故障などでノードをクラスタから外すときもRiakと同様の操作でよく、特にスキーマの再構築やリカバリの操作を行う必要はない。また、当然だが、これらの操作中にデータの読み書きをすることも可能だ。

スケーラビリティ

Riak自体は、100台を超えるクラスタでも安定動作するほどスケーラビリティに関しては実績がある。Yokozunaを組み込んだ場合でも、同程度にスケールすることを目指している。

7月に行われたRiak Meetup Tokyoでは、32台でも安定動作したという報告があった[注13]。

ただし、ノード台数を増やしてもいくつかの操作はスケールアウトしないので注意が必要だ。たとえば、検索クエリのAPI(/search/INDEXNAME?q=...)は内部的にはRiakの2iと同様に、1つのクエリが全ノード上に分散して送信される。したがって、ノードを追加しても単位時間あたりのクエリ処理性能は向上しない。ただし、データ量や件数の増加に対しては、ノード追加は分散処理の並列度を上げるので、ある程度の効果が期待できるだろう。

まとめ

ここまで、Riakの次期全文検索モジュールとして、ログ検索のユースケースをもとにYokozunaを紹介してきた。クラスタ構成でもセットアップや故障時の対処が容易で、柔軟なクエリを発行することができることをおわかりいただけたと思う。

残念ながら、Kibanaのようなモダンな検索UIがあるわけではない。Riakはデータの可用性にフォーカスしているので、**よりクリティカルなデータ**を保存し、柔軟に検索したい場合にYokozunaを使うべきである。また、割りきってモノリシックなアーキテクチャにしているため、マスタ、スレーブ、ZooKeeperなどサーバの役割を考える必要がなく、こういったユースケースに加えて簡単に運用したい場合に威力を発揮するだろう。SD

注13) IIJの曽我部さん、田中さんのスライドが非常にシンプルでわかりやすい。Yokozunaのよい入門資料にもなっている(http://www.slideshare.net/takashisogabe/riak-meetup2-iijbeta2)。

▼図6　配布用にパッケージを作成する簡単方法

```
$ make rel
$ sed -e '/{yokozuna,/,/]}/{s/{enabled, false}/{enabled, true}/;}' -i.back rel/riak/etc/ ➥
app.config
$ tar czf riak-yokozuna-0.8.0.tar.gz rel/riak
```

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13回 ハイパーバイザの作り方・総集編（上）

浅田 拓也（ASADA Takuya）Twitter @syuu1228

はじめに

前回は、Virtqueueとこれを用いたNIC（virtio-net）の実現方法について解説しました。今回はこれまでの総集編として、2回にわたって仮想化システムの全体像を振り返ります。

x86アーキテクチャにおける仮想化

今から40年ほど前に発表された論文“Formal Requirements for Virtualizable Third Generation Architectures”では、ハイパーバイザを実装するのに必要な命令セットアーキテクチャ上の要件が定義されています[注1]。

この定義は、ゲストマシン上で実行される命令を、実CPU上で直接実行するという前提で考えられています。これは、命令のエミュレーションを回避し、パフォーマンスの低下を避けるためです。

ゲストマシン上で実行される命令を実CPUで直接実行するには、「センシティブな命令」と呼ばれるものをトラップする必要があります。センシティブな命令には、システム資源の構成を変えようとする命令やシステム資源の構成に動作が依存している命令が含まれます。

一般的なCPUでは特権命令をユーザ権限で実行すればトラップがかかるので、センシティブな命令がすべて特権命令であれば「仮想化可能」となります。しかし、x86アーキテクチャにはセンシティブな命令であるにもかかわらず、ユーザ権限で実行可能な命令が複数存在しているため仮想化が困難でした。

VMwareはBinary Translation、Xenは準仮想化と呼ばれるソフトウェア上のテクニックによりx86向けのハイパーバイザを実装していましたが、ある程度のオーバーヘッドが生じたり、対応OSの種類が非常に限られていました。

そこで、Intelはx86アーキテクチャを拡張して前述の問題を解決しました。この拡張を「Intel VT-x」と呼びます。

Intel VT-xの概要

前節で述べたような問題を解決する簡単な方法としては、命令の挙動を変更したり、CPUのリングプロテクション機構[注2]のしくみを変更するものが考えられます。しかし、これらを変更すると、既存のソフトウェアとの互換性に問題が生じてしまいます。

そこで、Intel VT-xではそれらのしくみを変更せず、新たに「ハイパーバイザのモード」と「ゲストマシンのモード」を追加しています[注3]。この「ハイパーバイザのモード」をVMX root mode、「ゲストマシン

注1） PopekとGoldbergの仮想化要件。

注2） x86アーキテクチャではリングプロテクションと呼ばれる階層的な保護モードのしくみが導入されています。OSのカーネルとアプリケーションは異なる権限で実行され、アプリケーションがOSのメモリ空間を破壊したりハードウェアへ不正なアクセスを行うことを防ぎます。

注3） モードごとにリングプロテクションのステートが独立して存在します。図1参照。

のモード」をVMX non-root modeと呼びます（**図1**）。VT-xでは、このしくみを用いてゲストOSをVMX non-root mode上で動作させ、すべての命令をそのまま実行させます。

ゲストOSが実行した命令がセンシティブ命令だったときは、VMX non-root Modeでのプログラムの実行が中断されてVMX root Modeへモードが遷移し、ハイパーバイザが適切な処理を行います。このとき、VMX root ModeからVMX non-root Modeへ切り替えることをVMEntry、VMX non-root Modeが中断されVMX root Modeへ戻ってくることをVMExitと呼びます（**図1**）。

センシティブな命令の実行以外でもVMExitが必要な場合があります[注4]。

- ゲストOSの実行ばかりにCPUリソースを取られないよう、一定時間実行したら他のタスクに切り替えるため周期的にVMExitを行う場合
- ゲストOS実行中に入出力装置からの処理要求割り込みをハイパーバイザで受け取る場合

しかし、どのようなイベントが発生したときにVMExitを起こすべきなのかをハードウェアレベルで決め打ちしてしまうとハイパーバイザを柔軟に設計することが不可能になります。このため、VT-xではあらかじめ用意されたイベントの中から「これはVMExitする、これはVMExitしない」というようにハイパーバイザ側でコンフィギュレーション可能になっています。このコンフィギュレーション情報はVMCS（Virtual Machine Control Structure）と呼ばれ、ハイパーバイザのメモリ空間上にゲストマシンごとに別々に用意する必要があります。

VMCSは、VMExitするイベントの設定だけでなくゲストマシンのステートの保持や、VMExitされたときのVMExit Reason情報（以降、VMExit Reasonと表記）の通知などにも用いられます。また、VT-xの有効化・VMCSの読み書き[注5]・VMEntry実行などの仮想化にかかわる操作を行うためにVT-x拡張命令セットが導入されています。

VMCSの構造

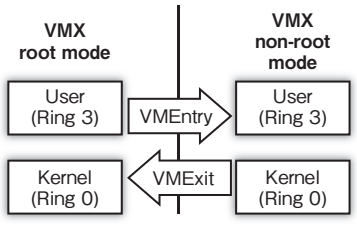
VMCSの内部構造は次の項目のようになっています（**図2**）。

- **VMCS revision identifier**
  ……VMCSのデータフォーマットのリビジョン番号。CPUにより書き込まれる
- **VMX-abort indicator**
  ……VMExitが失敗したときにCPUによりエラーコードが書き込まれる
- **Guest-state area**
  ……VMExit時にゲストレジスタを待避し、VMEntry時に復帰するための領域

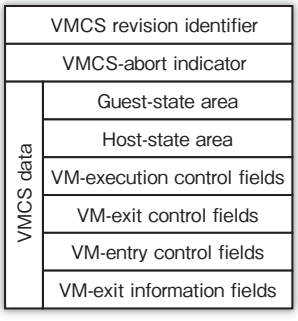
注4） ここに挙げているのは一例で、ほかにもさまざまな要因でVMExitが必要になります。

注5） VMCSはメモリ上の4KB（＝ページサイズ）アラインされた4KBのデータ構造ですが、内部構造は公開されておらず、必ずVMWRITE・VMREAD命令を介して読み書きすることになっています。

▼図1　VT-xの2つのモードとモード切り替え



▼図2　VMCSの構造



・Host-state area
……VMEntry時にハイパーバイザのレジスタを待避し、VMExit時に復帰するための領域

・VM-execution control fields
……ゲストマシン実行時のCPUの挙動を設定する（例：どのイベントでVMExitするか）

・VM-exit control fields
……VMExit時のCPUの挙動を設定する（例：外部割り込み発生時のCPUの挙動）

・VM-entry control fields
……VMEntry時のCPUの挙動を設定する（例：ゲストマシンへの割り込み挿入）

・VM-exit information fields
……VMExit時にCPUによりVMExit Reasonが書き込まれる

VT-x 拡張命令セット

ハイパーバイザからVT-xの機能へアクセスするために、次のような命令が追加されています。

・VT-x管理命令
VT-xを有効にするにはVMXON、ゲスト起動時にVMEntryするにはVMLAUNCH、VMExitした状態のゲストを再開するためにVMEntryするにはVMRESUMEを用います。

・VMCSメンテナンス命令
CPUへVMCSのアドレスをセットするにはVMPTRLD、VMCSのフィールドの読み書きにはVMREAD/VMWRITEを用います。

・ハイパーコール命令
ゲストOSから明示的にハイパーバイザを呼び出すための命令です。準仮想化を実装するのに用いられる事があります。

・EPT操作命令
EPTが有効なハイパーバイザで用いられます。EPTについては後述します。

Intel VT-x を用いたハイパーバイザのライフサイクル

Intel VT-xを用いたハイパーバイザは、次のような処理の繰り返しにより実現されます（**図3**）。なお、一連の処理はすべて特権モード（Ring0）で実行する必要があります。

①初期化

VMEntryを行いVMX non-root modeでゲストマシンを実行するには、まずVMXON命令でVT-xを有効化します。続いてVMCS用の領域をアロケート、VMPTRLD命令でVMCS用領域の先頭アドレスをセットします。そして、VMWRITE命令を用いてVMCSの初期値を書き込んでいきます。

②ゲストマシンへのモード切り替え

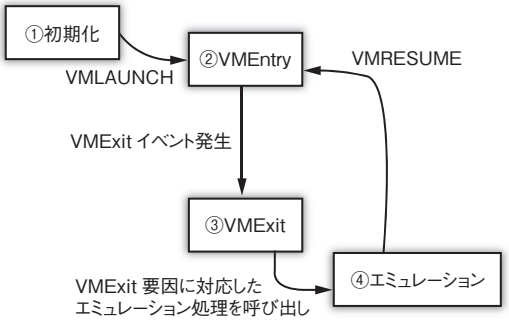
VMEntryに先立ち、ハイパーバイザのレジスタを待避しゲストマシンのレジスタを復帰します[注6]。

RIPなどハイパーバイザがソフトウェアにより待避・復帰を行うことができない一部のレジスタ[注7]に

注6）ゲストマシン起動時の場合、復帰すべきレジスタ値はブート時に設定されているべきレジスタ初期値となります。

注7）以下のようなレジスタが対象になります：CR0, CR3, CR4, DR7, RSP, RIP, RFLAGS, CS, SS, DS, ES, FS, GS, LDTR, TR, GDTR, SMBASE。また、レジスタ以外のいくつかのCPU内部ステートが待避されます。

▼図3　VT-xを用いたハイパーバイザのライフサイクル



ついては、VT-xがVMCSを用いて透過的に待避・復帰を行います。

ゲストマシンのレジスタ復帰を行ったら、VMX non-root modeへVMEntryを行いゲストマシンを実行します。このとき、VMCSを作成してから一度もVMEntryを行っていなければVMLAUNCH命令、二度目以降はVMRESUME命令を実行します。何らかのイベントによりVMExitされるまでの間CPU上でゲストマシンが実行されます。

③ VMExit

センシティブ命令の実行などの要因によりVMExitが発生したら、まずゲストマシンのレジスタを待避しハイパーバイザのレジスタを復帰します。

次に、VMCSに書き込まれたVMExit Reasonを調べ、要因に合わせた処理を行います。多くの場合、ゲストマシンが実機上で実行されていると見せかけるためのエミュレーション処理となります。たとえば、ハードウェアへのI/O命令であれば対象のハードウェアの挙動をエミュレーションする必要があります。

④ ②へ戻る

ゲストマシンを再開するには②へ戻ってVMRESUME命令を実行します。また、ここですぐにゲストマシンを再開せずに別のタスクを実行することも可能です。

連載第6回から第10回の記事では、このサイクルについてFreeBSD用のハイパーバイザ「BHyVe」のソースコードを例として挙げ、詳細に解説しました。

Intel VT-xにおけるページングとメモリの仮想化

仮想化されたシステムにおいて、CPU上でそのままゲストOSを実行するとき、ページング機構を仮想化する必要が生じます。これは、あるゲストマシンから別のゲストマシンやハイパーバイザのメモリ領域を参照できないようにするために、物理メモリの特定の領域だけをゲストマシンの物理メモリ領域として見せる必要があるからです。

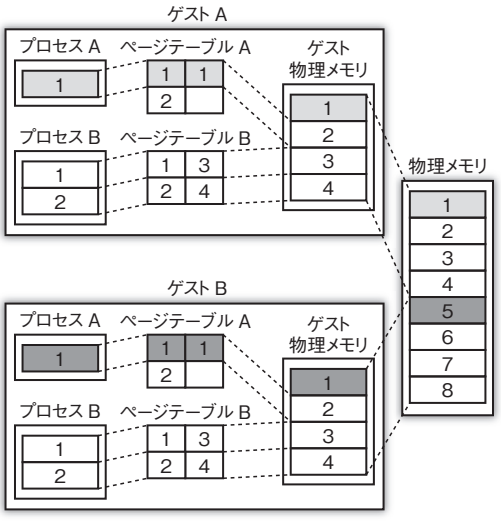
図4に具体例を示します。**ゲストA・B**はそれぞれ4つの物理ページを持ちます。**ゲストA**の物理ページ1には実機上の物理ページの1が、**ゲストB**の物理ページ1は実機上の物理ページ5が割り当てられます。それぞれのゲストマシンから異なるゲストマシンのメモリ領域は見えません。このような挙動を実現するために、ゲストマシンのページ割り当て情報の変換処理が必要になります。

この問題を解決するメモリ仮想化の手法として、ソフトウェアによる「シャドーページング」、ハードウェアによる「EPT」の2つを紹介します。

シャドーページングによるメモリ仮想化

ハイパーバイザはゲストのページディレクトリテーブルとページテーブルを複製（シャドウページテーブル）します。そしてハイパーバイザがゲストマシンに割り当てたメモリ範囲を反映した物理ページ番号をページテーブルの各エントリに書き込み、CPUのCR3レジスタに複製したページディレクトリテーブルを書き込みます（**図5**）。

▼図4　ページングとメモリの仮想化



シャドウページテーブルの作成と更新は、ゲストマシンのページテーブルへのアクセスとCR3レジスタへのアクセスを契機にVMExitさせることによって行います。これにより、ゲストマシンの実行時に仮想ページへアクセスが生じた時、ゲストOSが設定したゲストマシン内の物理ページ番号ではなく、ハイパーバイザが設定した実際の物理ページ番号を参照するようになります。

EPTによるメモリ仮想化

EPTを搭載したCPUでは、ゲスト物理アドレスからホスト物理アドレスへのアドレス変換を拡張されたMMUにより透過的に実行します。

ゲストからホストへのアドレス変換情報はExtended Page Tableと呼ばれる特別なページテーブルを作成して書き込み、CPUに設定します[注8]。

EPTをゲストマシンで有効にするには、Extended Page Tableを作成し、VMCSのVM Execution control fieldにあるExtended Page Table Pointer（EPTP）にアドレスを書き込みます。これにより、ゲストマシンから仮想メモリ空間への参照が行われたときに、CR3にセットされたページテーブルでゲスト物理アドレスへの変換が行われます。さらにEPTPにセットされたページテーブルでゲスト物理アドレスからホスト物理アドレスへの変換が行われます（**図6**）。

シャドーページングのときと異なり、頻繁なVMExitとハイパーバイザの介入を必要としないためパフォーマンスが向上します[注9]。

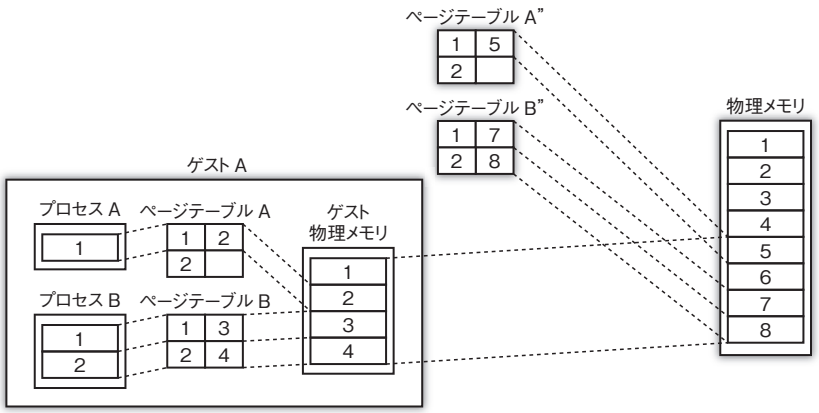
次号では

今回は、今までの総集編としてIntel VT-xの概要とメモリ仮想化について解説しました。次回は、総集編第2弾としてデバイスI/Oと割り込みの仮想化、ハイパーバイザの実装手法について解説します。SD

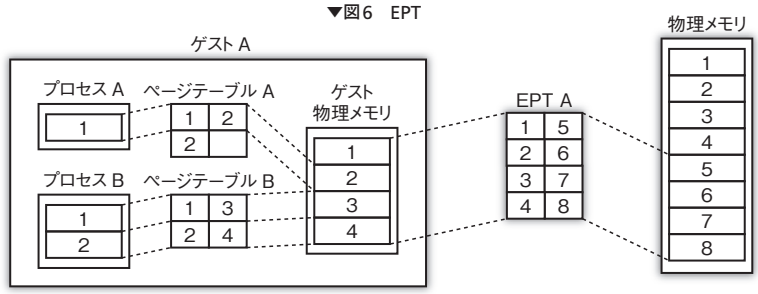
注8） Nehalem以降のEPTに対応したIntel CPUが必要です。一部のローエンド向けCPUではサポートされていません。

注9） ハイパーバイザの実装によっては30%ほど高速化すると言われています。

▼図5　シャドーページング

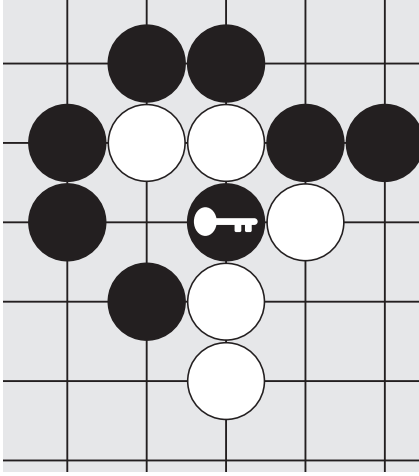


▼図6　EPT



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

【第四回】脆弱性はどう扱われるのか

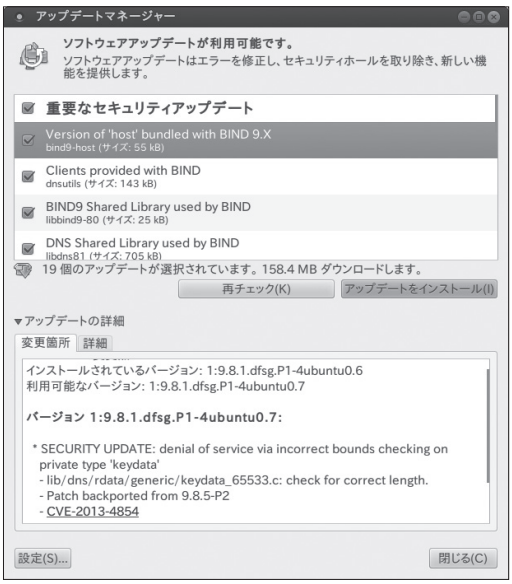
前回、脆弱性はなぜ現れるのかを述べましたが、今回は実際にソフトウェア脆弱性が見つかったとき、どのような流れで処理されるのかを考えます。まず具体的な話題から入っていくために、Ubuntuなどの Linuxディストリビューションを例にセキュリティアップデートを紹介するところからスタートします。

セキュリティアップデート

筆者はノートパソコンにはUbuntuを入れているのですが、かなり頻繁にセキュリティアップデートがかかります（**図1**）。

バグを見つけて順次修正するのとは違い、セキュリティアップデートは同時期にすべてのディストリビューション、あるいは利用しているシステムがアップデートされます。そのようなことができるのは関係者間で脆弱性情報の流通管理をしているからです。

◆**図1　Ubuntuのセキュリティアップデート画面**



「変更箇所」の説明部分に、何が問題であったかの説明とより詳しいサイトへのリンクがある。

脆弱性のハンドリング

前回、ソフトウェアに現れる脆弱性は「いつ現れるか」「どこに現れるか」「どの段階に原因があるか」の予見は難しいという説明をしました。また、脆弱性を見つける方法は、特別な能力や特殊な技術が必要なのではなく、これまでのソフトウェア品質を向上するためのテスト技法の延長線上にある、ということも説明しました。

では、その脆弱性が見つかったとして、どのような処理が行われるのでしょうか[注1]。

脆弱性ハンドリングの必要性

ソフトウェアの脆弱性は、ソフトウェアを提供している企業や開発者の問題にとどまらず、ソフトウェアを利用しているユーザの問題に直結します。

ソースコードが公開され、自由に変更、再配布可能であり、さらに“AS IS”[注2]という条件で提供されるものであれば、パッチや脆弱性の対応は、その

注1）ここではたとえばApache、BINDあるいはOpenSSLのような汎用のアプリケーション、あるいはライブラリの脆弱性についての説明に絞ります。
注2）“AS IS”とは無保証で、そのままの形で提供されること。

ソースコードを利用する者が自分の手で修正するという形があっても良いと思います。

しかしながら現実には、1つのシステムを利用する場合、そこに存在するソフトウェアの多様性を考えると、オリジナルのソースコードのアップデートも含めた形でユーザ側で管理するというのは現実的ではありません。

また、1つのソフトウェアベンダーがユーザの利用するソフトウェアすべてを提供するというのは、現実的な運用においては、極めて稀な状況です。サードパーティのソフトウェアも含めて利用するのが一般的です。たとえばWebブラウザを使う場合、そのWebブラウザがシステムに付属していたとしても、実際にはそこにはPDFのプラグインや、あるいは動画とスクリプト言語を扱うようなプラグインがサードパーティから提供されているはずです。Webブラウザ自体もサードパーティから提供されているものを使っているかもしれません。

そのうえで、企業、団体、個人にかかわらず、利用者の安全性を考えるうえで脆弱性に対応していく必要があります。この問題をどう解決するか考えていくと、最終的には中立な第三者が統一的に脆弱性情報を一元化して取りまとめ、そしてハンドリングする方法が最も効率的であるのではないか、というところに落ち着きます。

脆弱性のハンドリングを担う組織

それでは、脆弱性ハンドリングの全体像を説明していきたいと思います。

MITRE Corporationという組織があります(**図2**)。米国の非営利団体でFederally Funded Research and Development Centers(FFRDCs)という米政府が資金提供している研究組織群の中の1つです。ここが世界規模で脆弱性のハンドリングを行っています。

この組織は、直接的には米国国土安全保障省(U.S Department of Homeland Security)のサイバーセキュリティ&コミュニケーション室(Office of Cybersecurity and Communications)から資金を得ています。もっとあからさまにいえば、脆弱性のハンドリングは、米国の国家安全保障の枠組みの中で運用されています。

多くのユーザは「脆弱性はベンダーや開発者の責任範囲」と考えているのではないでしょうか。しかし、現実には国家安全保障の問題として扱われています。これには1988年にモリス・ワームが脆弱性をついて繁殖し、インターネット全体を麻痺させたことが背景にあると筆者は考えています。

MITREが番号を振ったうえで一般に公開する脆弱性の情報のことをCVE(Common Vulnerability Exposure)と言います。この情報は次のサイトで公開されています。

http://cve.mitre.org/

また、この番号のことをCVE番号と言い、これが世界共通の脆弱性の統一番号と言えるものです。CVEのことを日本国内では共通脆弱性識別子と命名していますが本連載ではCVEと呼びます。

いろいろな脆弱性ハンドリングの組織やソフトウェアベンダーから脆弱性の情報が上がってきて、最終的にCVE番号が振られるわけですが、すべてに対して番号が振られるわけではありません。脆弱性として扱うべきもののみが対象になります。

このCVEの番号の振り方は、「CVE-YYYY-NNNN」となります。YYYYは西暦、NNNNは連番です。たとえばCVE-2013-0001(2013年最初の脆弱性登録の番号)は、「.NET Frameworkの脆弱性に

◆図2 MITREのWebサイト

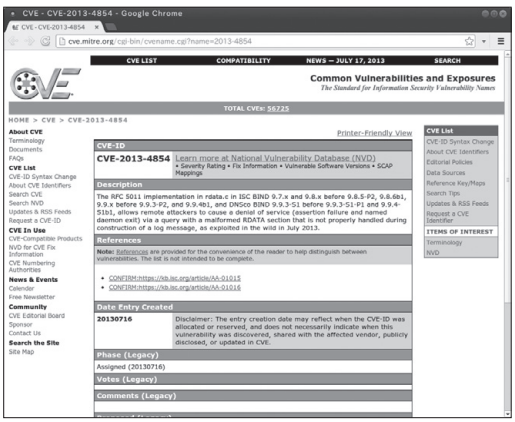


図1の画面の「アップデートの詳細」からもリンクが張られている。

◉世界規模でセキュリティに取り組むきっかけを作ったワーム

脆弱性と国家レベルでのセキュリティに関しては、1988年11月8日に発生したモリス・ワームにまで遡る必要があります。モリス・ワームとは当時コーネル大学の学生だったロバート・タッパン・モリスが作成した世界で最初のインターネット上で大規模に繁殖したワームです。これはDEC VAX上の4BSDやSUN-3の上で動作するsendmail、fingerやrshの脆弱性を介して繁殖し（そのほかにもパスワードクラッキングの能力を持っています）、その影響でインターネット全体の電子メールシステムが麻痺してしまいました。約6,000台のマシンが感染したと言われます。これは当時のインターネットに接続していたコンピュータの10%と言われますが、実際には接続しているマシン数を把握していないので、どれだけの割合になるかは正確にはわかりません。

感染したコンピュータが麻痺しただけはなく、感染を恐れてインターネットから接続を切り離したマシンも数多く、結果としてメールシステムだけはなく、インターネット全体が大規模に麻痺しました。これにより脆弱性の問題がインターネット全体に影響するということが認識されるようになりました。

このモリス・ワームがきっかけになり、国防高等研究計画局DARPAからの資金を得てカーネギーメロン大学の付属研究所であるSoftware Engineering Institute内に世界初のコンピュータ緊急対応センターCERT/CCが作られることになります。

より、特権が昇格される」というものです[注3]。

CVE番号が振られ、リストアップされることで脆弱性として「認められた」あるいは「対処すべきもの」という位置づけがされます。

ただし、この判断は米国の国家安全保障という枠組みの中でとらえられるものですから、「日本国内のみ」とか「どこかの国内のみ」といった地域的にしか使われないものにはCVE番号が振られるかどうかは、なんとも言えません。振られない確率のほうが高いと考えたほうがいいでしょう。逆もまたしかりで、CVE番号が振られない脆弱性は、Linuxディストリビューションなどでは脆弱性対応として扱われずに、通常のバグフィックスとして扱われる可能性が高いです。

一般にCVEや同様の脆弱性情報を扱う組織から脆弱性情報が公開される場合は、その脆弱性の対応が終了し、ユーザに利用してもらえる環境になっているもの、あるいは最悪、回避方法を示すこと[注4]が可能なもののみ公開します。脆弱性の対処ができていない状態での公開というものは原則ありません。

脆弱性の影響度

Common Vulnerability Scoring System（CVSS-SIG）は、その脆弱性の影響度を示すための指標の1つです。いろいろなソフトウェアに次から次へと現れる脆弱性のすべてが同様に危険なわけではありません。ですが、その危険度、問題度を客観的基準を持たずに共通に理解することはたいへん難しいことといえます。脆弱性に対し、関係者の中で共通認識を持つためにも、客観的な指標が必要となります。そこでセキュリティ対応組織が寄り集まっている国際的団体であるFIRSTが中心になってCVSS-SIGを策定しました。

脆弱性の対応の流れ

CVEにはいろいろな脆弱性ハンドリングチームからの脆弱性報告があがります。脆弱性ハンドリングチームは、いろいろな形態があります。たとえばMicrosoft社のような巨大なソフトウェアベンダーであれば、自らが、脆弱性を発見する、あるいは外部から情報がもたらされるところから始まり、脆弱性の管理、ソフトウェアの対応、セキュリティアップデート配布、そしてCVEの報告／番号の獲得まですべてを内製化して処理できることでしょう。ですが、すべてがそのような組織を内部に持つことが

注3） http://technet.microsoft.com/ja-jp/security/bulletin/ms13-004
注4） 最悪の場合「このソフトウェアは利用すべきではない」という勧告もありえます。

できるわけではありません。

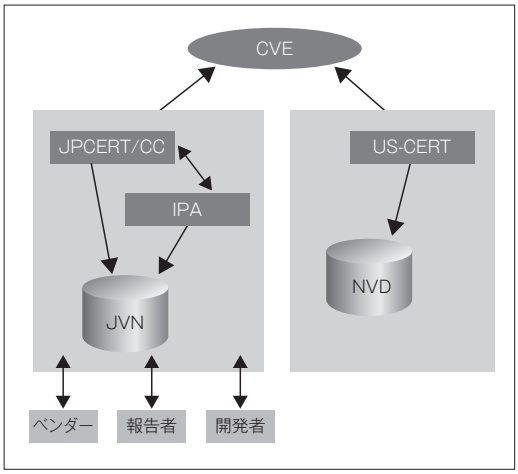
脆弱性を持つソフトウェアを分析し修正するのは、開発者かそれに相当する人たちの力が必要ですが、脆弱性情報の流通に関しては第三者でかつCVE番号の取得や脆弱性情報の管理あるいは公開など専門的に行う組織あるいは集団のほうが効率が良く確実でしょう。

日本国内ではJPCERTコーディネーションセンター（JPCERT/CC）と独立行政法人情報処理推進機構（IPA）が共同で脆弱性情報の流通に対応しています。脆弱性報告の受け付け、ベンダーや開発者との取りまとめ、脆弱性情報の管理、そして脆弱性対応告知といった範囲まで含めてサポートできる体制を整えています（**図3**）。そのポータルサイト（脆弱性情報データベース）がJVN（Japan Vulnerability Notes）[注5]です。

米国ではUS-CERTがその対応にあたり、脆弱性情報データベースはNVD（National Vulnerability Database）[注6]です。

JVNは日本国内だけではなく、海外での脆弱性情報を国内で展開する役目も持っています。たとえば次の例を見てみましょう。

◆図3　日本及び米国の脆弱性データベースとCVEとの関係



日本国内での脆弱性対応は、日本国内の組織で行う。国際的な調整が必要な場合は、各国の同様な組織、あるいはベンダーと一緒に対応する。ここでは日米の組織しか示していないが、大手ベンダーやあるいは各国に同様な組織がある。

> 公開日：2013/08/16
> JVNVU#95005184
> DellのBIOS更新処理にバッファオーバーフローの脆弱性
> 概要：Dellが提供する複数のLatitude LaptopおよびPrecision Mobile WorkstationのBIOS更新処理に、バッファオーバーフローの脆弱性が存在します。
> （略）
> ・CERT/CC Vulnerability Note VU#912156
> ・CVE: CVE-2013-3582

DellのBIOS更新処理のバッファオーバーフローの脆弱性ですが、これは米国ではCERT/CC Vulnerability NoteのVU#912156として管理されているのと、国際的にはCVE-2013-3582で管理されているということになります。

脆弱性対応を巡る議論

脆弱性対応を巡るいくつかの議論です。脆弱性情報流通の対応もすべてがうまく回るわけではありません。それに関連し、私たちが考えなければいけないことをいくつか取り上げてみたいと思います。

情報管理の複雑さ

扱う情報は脆弱性情報ですから、外部への情報流出にはかなり神経質になります。万が一、対策が存在しないまま脆弱性情報が外部に漏れてしまえば、それは、そのままゼロデイ攻撃になる可能性があるからです。

ソフトウェアの開発者が、大手ベンダーや、あるいは組織がしっかりしたオープンソースコミュニティであれば、先ほどの脆弱性情報流通の枠組みの中で情報を共有したり、対応したりすることが問題なく行えることでしょう。しかし、小規模なベンダーやオープンソースの作者グループであった場合、

注5）**http://jvn.jp/**
注6）**http://nvd.nist.gov/**　US-CERTがNVDを運営していますが、Webサイトは米国国立標準技術研究所NISTのドメイン下にあります。

それがたいへんな負荷になります。これらの負荷をどう軽減するかなど考える必要があるでしょう。

一定期間対処されない脆弱性

脆弱性が発見され、その対処について開発者に連絡をとっても反応を示さない場合、これはどうしようもなくなります。日本国内でも、いつまでも潜在的な脆弱性の存在をそのままにするのは良くないという考え方から、ある一定期間の猶予を与えたあと、脆弱性の告知を行うという運用に変化しつつあります。脆弱性を公表できないことを逆手に取られ、存在する脆弱性を密かに悪用され続ける可能性を排除するためです。

サポートが終了したソフトウェアも脆弱性は修正されないわけですから、基本的には、先の例と同じように、修正されないまま脆弱性の告知がされることになるでしょう。

見解の相違

バッファオーバーフローのようにコーディングのミスなら、それはミスとしか言いようがありませんが、仕様に基づく見解の相違ほど厄介なものはありません。有名なのが、GCCの最適化によって期待していたコードが生成されないという問題です。

- Vulnerability Note VU#162289
  C compilers may silently discard some wraparound checks
- JVNVU#162289
  ある種の範囲チェックを破棄するCコンパイラの最適化の問題

たとえば、次のようなバッファオーバーフローを起こすかどうかをチェックするコードがあります。

```
char *buf;
int len;
len = 1<<30;
 略
if(buf+len < buf)
```

GCCの場合、コンパイラが最適化するためにこのコードが目的とした動作であるバッファオーバーフロー対策のチェックができません。そのため、最初、脆弱性として位置づけられ、VU#162289とJVNVU#162289が発行されました。

しかし、筆者に言わせれば、そもそもの問題としてアドレス空間と整数とを計算させてどうなるかという話であり、コンパイラの実装依存になるのは当たり前です。この書き方自体が移植性などに非常に問題のある書き方です。他機種にコードを持っていって違う種類のコンパイラでコンパイルすると、動作が以前のコンパイラと同じかどうかは保証されません。

当初、この問題を指摘した人は、ほかのコンパイラでは問題ないので、GCCの問題だと主張していました。結局、最終的に、次のようなコードに書き改めるということになりました。

```
#include <stdint.h>
if((uintptr_t)buf+len < (uintptr_t)buf)
```

「Cの表現はキャストして型を明確にしないと危ない」というよく聞く話です。ですがGCCにも少し問題があって、当時は型があいまいなのにコンパイラとして警告が出ていなかったようなのです。それが誤解を生んだのかもしれません。

コンパイラの解釈の問題ですから、当然、正しい答えなどなく話が迷走します。そのためドキュメントの変更履歴がとんでもないことになっています。CVE番号は当然ながら取得されていません。最初にVU#162289が公開されたのが2008年4月4日ですが、最終バージョンは2008年10月8日で、そこまでに内容が61回更新されています。

2008年4月4日にCERT/CCがVU#162289を発行してから、GCCメーリングリストでどういう議論がされたか、アーカイブに残っているので興味がある方は目を通すとおもしろいかもしれません[注7]。SD

注7） http://gcc.gnu.org/ml/gcc/2008-04/msg00115.html

プログラム知識ゼロからはじめる
iPhoneブックアプリ開発

第6回

アプリにBGM・効果音を加えよう!

GimmiQ(ギミック;いたのくまんぼう&リオ・リーバス)
URL http://ninebonz.net/
URL http://www.studioloupe.com/
イラスト●中川 悠京

プログラミングをしたことのない方にもアプリを作る楽しさを味わってもらいたい本連載。今回はアプリにBGMと操作にあわせた効果音を付ける方法を解説します。

今回のテーマ

今回はアプリにBGMを付けたり効果音を鳴らす方法についての説明です。iOSアプリで音を鳴らす方法はシンプルなものから高度なものまで幅広くありますが、まずはシンプルに音を鳴らすところからはじめてみましょう。

音は大きく分けるとループさせたり、長時間流し続けたりするBGMのようなものと、動作に合わせて音を鳴らす効果音のようなものがあります。iPhoneのように真っ平らなタッチパネル上での操作が強いられるデバイスの場合、画面上のものを物理的に触ることができないため、動きや音によって擬似的に触ったかのような操作感や気持ち良さを加える手法が有効です。音がなければアプリが完成しないというものではありませんが、より良いアプリにするために不可欠な要素であることは間違いないでしょう。

音源の用意──どんなファイルが再生可能?

まずは音声ファイルを用意しなければはじまりません。どのような形式のオーディオフォーマット(音声ファイル)がiOSに対応しているかを**表1**にまとめました。まずは仮でもかまわないので、上記の形式の音声ファイルを2つ用意しておいてください。できれば、1つがBGM用に長めの曲、1つが効果音として使える短めのサウンドが望ましいです。

音源をプロジェクトに追加

音声ファイルの準備ができたら、アプリで使用できるようにプロジェクトにドラッグ&ドロップで加えましょう。ファイルの置き場所は画像ファイルと同じ場所でもいいですし、数が多い場合は管理しやすいようにすべての音声ファイルを1つのフォルダにまとめて、フォルダごとプロジェクトにドラッグ&ドロップしてもかまいません。

▼表1 iPhoneがサポートする主なオーディオフォーマット

略称(拡張子)	オーディオフォーマット名
CAF(.caf)	Apple Core Audio
ALAC(.m4a)	Apple Lossless Audio Codec
AIFF(.aif/.aiff)	Audio Interchange File Format
AAC(.m4a)	Advanced Audio Coding
MP3(.mp3)	MPEG Audio Layer-3
WAV(.wav)	RIFF waveform Audio Format

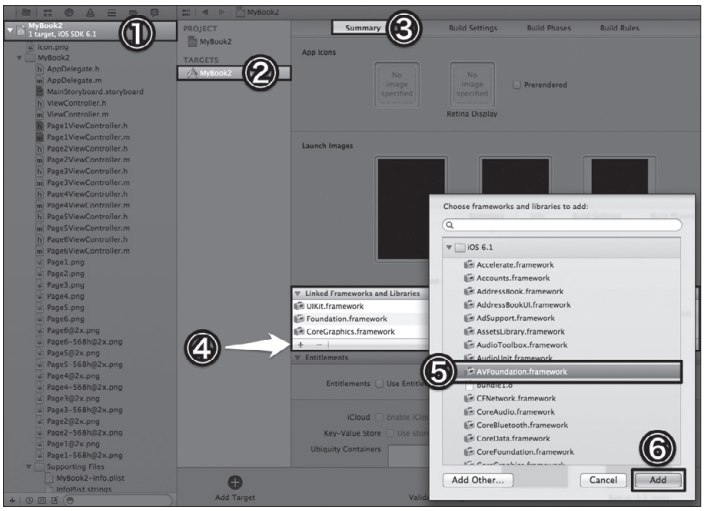
フレームワークの追加

音声ファイルを再生するためにはプログラムにコードを加えるだけでは足らず、音声を再生するた

めの専用の「フレームワーク」を追加しなければいけません。フレームワークとは特定の機能を使用するために必要なものがすべて詰まったパッケージです。

ステップ1

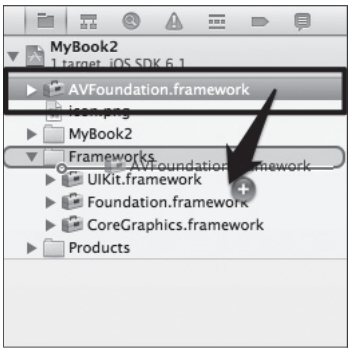
step1



フレームワークを追加するためには、まずXcode内の左カラムのプロジェクトファイル(記事例ではMyBook2)をクリックします(**図step1-**①)。続いて**図step1-**②の「TARGETS」の下のプロジェクト名を選択しましょう。**図step1-**③の「Summary」タブの中で下の方へスクロールしていくと、**図step1-**④の「Linked Frameworks and Libraries」という項目が現れるので、この中の左下の「+」ボタンを押します。すると追加可能なフレームワークのリストが現れるので、その中から「AVFoundation.framework」を選択(**図step1-**⑤)して「Add」をクリック(**図step1-**⑥)してください。

ステップ2

step2

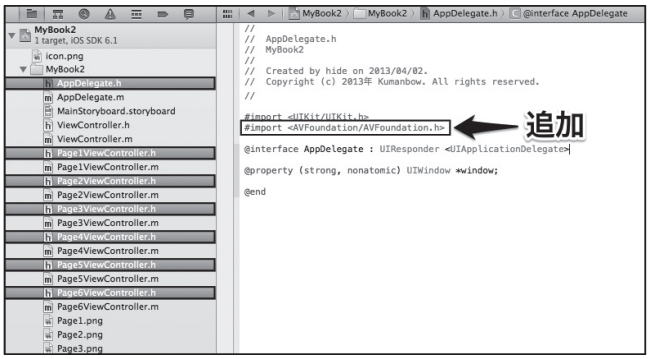


これでプロジェクトに新しいフレームワークが追加されました。フレームワークは左カラムの一番上に追加されます。そのままの場所でも問題ありませんが、下のほうにフレームワークをまとめる専用フォルダ「Frameworks」があるので、そこへドラッグ&ドロップで移動して整理しておくことをお勧めします(**図step2**)。

ヘッダファイルにライブラリの宣言

ステップ3

step3



フレームワークをプロジェクトに追加後、そのフレームワークを実際に使用することをヘッダファイルで宣言しなければなりません。「AppDelegate.h」とすべてのページのヘッダファイル("Page"からはじまり、"ViewController.h"で終わるファイル)の中の`#import <UIKit/UIKit.h>`のすぐ下に、`#import <AVFoundation/AVFoundation.h>`を追加入力してください(**図step3**)。これで音声再生のための下準備が整います。

クラスのインスタンスを宣言

次は、AVFoundation.framework内の音声ファイルをコントロールするための「クラス」、AVAudio Playerの「インスタンス」を宣言しなければなりません。

まず、クラスを簡単に説明するために、プログラム全体を「学校」としてたとえてみましょう。学校にはたくさんの「クラス」が存在します。これらのクラスは機能別に細かく分かれていて、小さなプログラムとしてそれぞれ連動し合ってプログラム全体を動かしています。つまり、プログラム(学校)はこのような小プログラム(クラス)の集合体ということです。さらにクラスの中には個別の生徒がいます。これら生徒のことを「インスタンス」(またはオブジェクト)と言います。生徒にはそれぞれ別の名前を付けてあげなければなりません。そうでなければ呼びたい生徒を呼び出すことができないからです。

つまり、これから行う「インスタンスの宣言」とは、AVAudioPlayerクラスの中には、こういう生徒(インスタンス)がいる、と宣言するということです。

ステップ4

それでは、AppDelegate.hを選択し、`@interface AppDelegate : UIResponder <UIApplication Delegate>`の後に波括弧「`{`」を加え、return を入力してください。自動で「`}`」が入力されるので、2つの波括弧の間に`AVAudioPlayer *bgm;`を追記しましょう。入力後は**図step4**のようになるはずです。

「AVAudioPlayer」がクラス、続く*(アスタリスク)の後の「bgm」がインスタンスになります。学校のたとえで言えば、「AVAudioPlayerクラスの中のbgm君」という生徒というわけです。ここではbgmとしましたが、audioやsoundなど、自分にとってわかりやすいほかの名前を付けても問題ありません。

step4

```
#import <UIKit/UIKit.h>
#import <AVFoundation/AVFoundation.h>

@interface AppDelegate : UIResponder <UIApplicationDelegate>{
    AVAudioPlayer *bgm;
}

@property (strong, nonatomic) UIWindow *window;

@end
```

BGMを鳴らす

宣言が終わったら、次は実際に音を鳴らすための処理を加えていきましょう。音を鳴らす場面はさまざまなので、どのタイミングで、何をしたときに鳴らすかによって、どこにコードを書くかが変わります。今回は、起動後すぐに自動的にBGMを鳴らすパターンと、ページが切り替わるタイミングで効果音を鳴らす2つのパターンを紹介します。

ステップ5

まずは起動時にBGMを鳴らす処理からはじめましょう。すでにAppDelegate.hにライブラリの宣言とインスタンスの宣言はできているので、AppDelegate.mを選択し、コード中の`- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {`の中に音声再生のためのコードを加えます。`return YES;`の上に**リスト1**のコードを書き込みましょう[注1]。入力後AppDelegate.mは**図step5**のようになります。

これでアプリの起動と同時にBGMが鳴り始めます。`@"bgm"ofType:@"mp3"];`の部分は事前に用意し

注1) 「//」からはじまるコメントアウトの部分は入力しなくて大丈夫ですが、メモとして残しておきたい場合は「//」だけは全角ではなく半角で入力しないとエラーの原因になるので注意が必要です。

たファイル名によって調整しなければいけません。音声ファイル名が「music.wav」である場合は、**@"music"ofType:@"wav"];**になります。

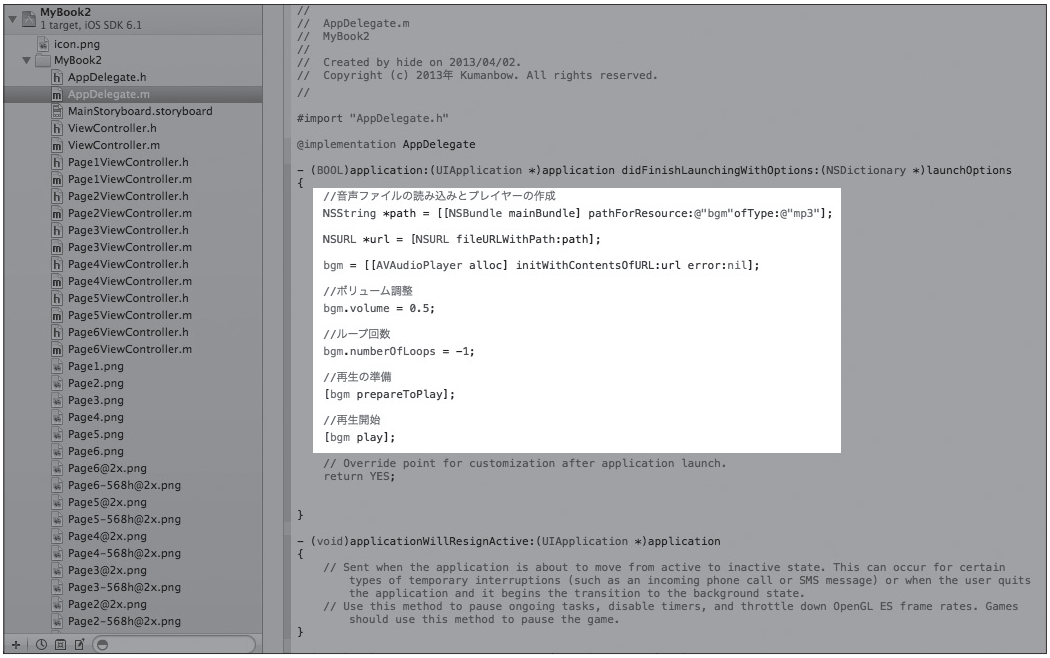
ボリューム調整は0.0(無音)から1.0(最大限)まで自由に設定できますが、元の設定のままで構わない場合は**bgm.volume = 0.5**の1行は不要です。

ループ回数の設定は、エンドレスにループさせたい場合は「-1」にしましょう。「0」なら1回、「1」なら2回、音が鳴ります。なぜ「1」で2回かというと、これは“繰り返す”回数を表しているからです。つまり1回目の再生は繰り返しには含まれていません。1回目のあとにさらに何回追加で再生させたいかの回数の設定と考えればわかりやすいと思います。

▼リスト1

```
//音声ファイルの読み込みとプレイヤーの作成
NSString *path = [[NSBundle mainBundle] pathForResource:@"bgm"ofType:@"mp3"];

NSURL *url = [NSURL fileURLWithPath:path];

bgm = [[AVAudioPlayer alloc] initWithContentsOfURL:url error:nil];

//ボリューム調整
bgm.volume = 0.5;

//ループ回数
bgm.numberOfLoops = -1;

//再生の準備
[bgm prepareToPlay];

//再生開始
[bgm play];
```

step5



効果音を鳴らす ── イベントの作成

ページの切り替えのときに音を鳴らすために使用するコードは基本的にBGMのものと同じですが、唯一違う点は音を発生させるタイミングです。今度は起動時ではなく、ユーザがページを切り替えたタイミングに合わせて音を再生させたいので、新たに専用の“イベント”を追加しなければなりません。イベントとはある条件によって発動する処理のことです。今回のケースでは、ボタンを押したと同時に発動する「音を鳴らすイベント」を作成します。

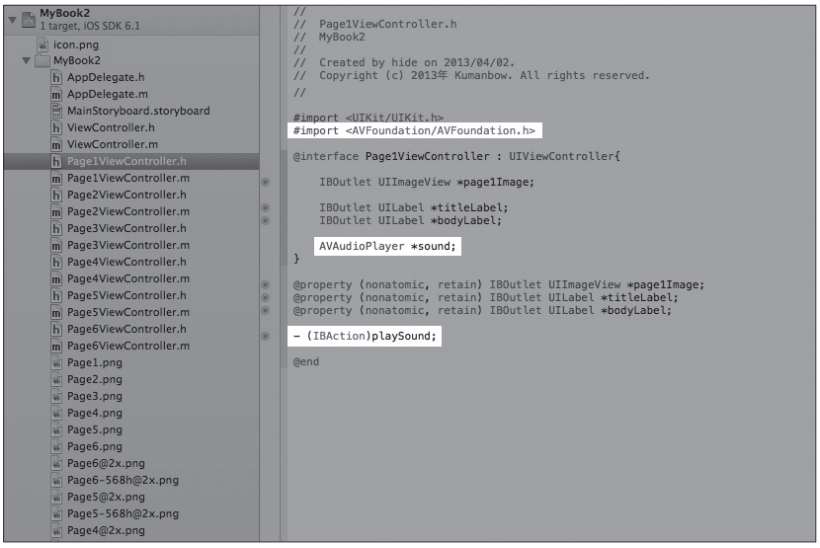
ステップ6

ページをめくる動作は各ページのViewControllerが行っているので、音を鳴らすのも同じように各ページのViewControllerのヘッダ(.h)と実装ファイル(.m)にコードを加えなければなりません。まずはPage1ViewController.hを選択してください。AppDelegate.hと同じ要領でインスタンスの宣言をしてください。`@interface Page1ViewController : UIViewController{`の中に`AVAudioPlayer *sound;`を加えましょう。今回は効果音ということで、名前はbgmではなくsoundとしました。

続いて、`@property ...`と`@end`の間に、`- (IBAction)playSound;`というイベントを追加します。IBActionはストーリーボード上でボタンなどに対して関連付けることのできるイベントです。これからこのplaySoundというイベントと、ストーリーボード上のボタンを紐付けることで、ボタンを押したときに効果音を鳴らす処理を発動させるようにします。

以上の手順を終えたら、**図step6**のようになっているはずです。

step6



ステップ7

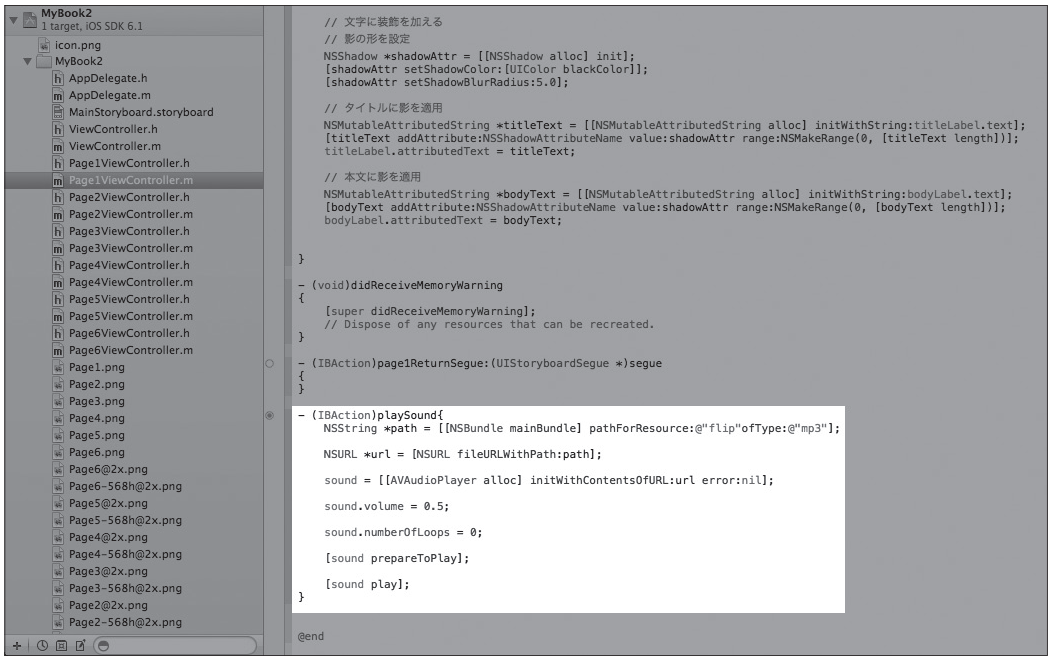
次にPage1ViewController.m(実装ファイル)のほうを選択し、イベントそのものの処理を加えます。一番下までスクロールし、`@end`の上に**リスト2**のスクリプトを書き込みましょう(**図step7**)。

これで1ページ目の効果音を鳴らす処理ができあがりました。しかし、ページごとに音を再生させる処理を加えなければいけないので、Page2ViewController以降もページの数だけ.hと.mファイルにまったく同じ手順を繰り返し入力してください。

▼リスト2

```
- (IBAction)playSound{
    NSString *path = [[NSBundle mainBundle] pathForResource:@"sound"ofType:@"mp3"];

    NSURL *url = [NSURL fileURLWithPath:path];

    sound = [[AVAudioPlayer alloc] initWithContentsOfURL:url error:nil];

    sound.volume = 0.5;

    sound.numberOfLoops = 0;

    [sound prepareToPlay];

    [sound play];

}
```

step7



効果音を鳴らす ── Storyboardで処理を紐付ける

ステップ8

BGMは起動と同時に自動的に再生される処理ですが、効果音を鳴らすにはStoryboard上のボタンに対してイベントを紐付けなければなりません。まず、Xcode左カラムの「MainStoryboard.storyboard」を選択しましょう(**図step8-**①)。Storyboard内の左カラムにある、「Page1 View Controller Scene」の中の「Page1 View Controller」をクリックしてください(**図step8-**②)。Xcodeの右カラムから「コネクションインスペクタ」を選ぶ(**図step8-**③)と、先ほど加えたイベント"playSound"が「Received Actions」の下に新しく加わっていることが確認できます(**図step8-**④)。"playSound"の右の◯をクリックし、そのままドラッグしながら1ページ目の右半分をしめるボタンの上で離しましょう。イベント発動条件の一覧が現れるので、「Touch Up Inside[注2]」を選択します(**図step8-**⑤)。これでやっとボタンを触る(ページをめくる)という行動に対して効果音を鳴らす処理(playSound)が紐付けられたことになります。この手順をすべてのページで行わなければいけません。

1ページ目は右に進むボタンしかないので、右側のボタンに処理をつなげるだけですみましたが、2ページ目以降(最後のページ以外)は左右のボタンに同じ処理をつなげましょう。1つのボタンに対して複数のイベントを加えることが可能であるように、同じイベントを複数のボタンに使用することも可能です。つなげるための方法はまったく同じようにplaySoundの右にある◯をそのままドラッグ&ドロップするだけです。

注2) Touch Up Insideを選ぶと、ボタンから指が離れたと同時にイベントが発動するようになります。Touch Downだとボタンに触れた瞬間に音が鳴ってしまうので、この場合はTouch Up Insideが適切でしょう。

step8

実行して確認

StoryBoardですべてのページのすべてのボタンにイベントをつなげることができたら、音を鳴らすための準備が整ったことになります。シミュレータで実行してテストしてみましょう。起動するとBGMが流れ、ページを行ったり来たりするたびに効果音が鳴るはずです。もしも音が鳴らないページなどがあった場合は、そのページの.hと.mファイルを確認し、記入漏れやコードに間違いがないかを確認してみましょう。

いかがでしょう？　音が加わるだけで動きにリアリティが増しますよね？　当然、どんな音を使用するかによって感じ方も変わってくるはずです。音声ファイルをゼロから作るのは専門知識や特殊なソフトがなければ簡単ではありませんが、インターネット上でフリーで使用できる素材はたくさんあります。理想の音を見つけるまでは少し時間がかかるかもしれませんが、音はアプリのユーザエクスペリエンスを左右する大事な要素の1つなので、妥協せず時間をかけて探してみることをお勧めします。SD

本連載のサポートページで記事の補足説明をしています。あわせてご活用ください！
› http://www.gimmiq.net/p/sd.html

いたのくまんぼう／Itano Kumanbow
Twitter @Kumanbow

神奈川工科大学非常勤講師。リオさんとはGimmiQ名義で「MagicReader」（手を使わずにページがめくれる電子書籍ビューワ）をリリース。個人ではNinebonz名義で「Crop It Cam!」（おしゃれな切り抜き写真カメラ）、「i列車の車窓からーそうだ! 京都行こう!ー」（バーチャル旅行アプリ）など。アプリ紹介サイト「あぷまがどっとねっと（http://appmaga.net/）」の技術サポータ。

リオ・リーバス／Leo Rivas
Twitter @StudioLoupe

iOSアプリ開発を中心に電子絵本作家・漫画家として活動中。代表作は、KDDI株式会社に社内導入され、世界で40万ダウンロードを記録する革新的な電卓アプリ「FusionCalc」と、国連主催のWSA Mobile 2013を受賞した、顔の動きで電子書籍が読める「MagicReader」。電子絵本はiBookstore/Kindleストア共に児童書カテゴリ総合1位を獲得。

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

5% OFF& 送料無料

年間定期購読のご案内

富士山マガジンサービス版

年間購読なら割引料金で購読できます!

全国どこでも直接お届けしています!

1年購読(12回)

14,580円(税込み，送料無料)1冊あたり1,215円(5%割引)

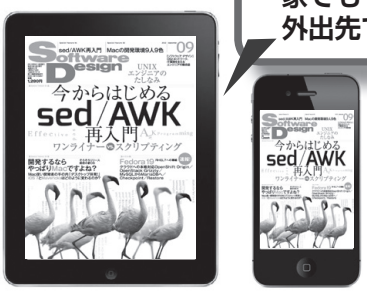
- インターネットから最新号、バックナンバーも1冊からお求めいただけます！
- 紙版のほかにデジタル版もご購入いただけます！
  デジタル版はPCのほかにiPad/iPhoneにも対応し、購入するとどちらでも追加料金を払うことなく読むことができます。

[月額払い]スタートしました！

デジタル版Software Design

Webで購入

家でも外出先でも



※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。

お申込み方法

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

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

第41回

[アプリ開発2013] 2 Android開発を体験しよう

モバイルデバイス初のオープンソースプラットフォームとして、エンジニアから高い関心を集めるGoogle Android。いち早くそのノウハウを蓄積したAndroidエンジニアたちが展開するテクニックや情報を参考にして、大きく開かれたAndroidの世界へ踏みだそう！

鈴木 圭介　SUZUKI Keisuke
Android & 組込み技術者
URL www.ksksue.com/wiki/ Mail ksksue@gmail.com

アプリ開発に役立つ3つのポイント

1回目となる前回に引き続き、これからAndroid開発をはじめようとしている方を対象に、全3回に分けて新しい機能や話題を盛り込みながら基本的な開発方法について解説していきます。前回にも書きましたが、Androidを取り巻く環境は黎明期と言える状態はとうに過ぎ去り、成熟期に入ってきています。それは言い方を変えれば、これからAndroid開発に一歩踏み出そうという人にとって、開発をはじめるための環境が整っている状態にあるということです。

また、これまでデスクトップやノートPCにあった常識がタブレット・スマートフォンによって日々変化してきています。時期尚早と様子見をしていた方も、今この時期にAndroid開発に一歩踏み出してみてはどうでしょうか。

さて、今回はTips表示アプリを題材に、Androidアプリ開発を知るにはちょうど良い3つのポイント「Activity」、「ActionBar」、「WebView」について解説していきます。

Android開発の基本

Androidの基本であるActivity、そしてGoogleの推奨するデザインパターンActionBar、またblogの記事などを修飾する際にどなたでも一度は触れたことのあるであろうHTMLを表示させるWebViewを使ってアプリ開発を体験してみましょう。そして、新しいトピックとしてAPI 18から刷新されたサポートライブラリのActionBar Activityメソッドの使い方を解説していきます。

また、今回サンプルとして用意したTips表示アプリは、作って終わりというわけではなく、後々作った自分のアプリに追加できるようなものにしました。前半で重要項目について解説し、後半で実際にプログラミングをしていきます。

最も重要なクラスActivity

Androidプログラミングで最も重要なクラスはActivityクラスでしょう。**Androidアプリの開始と終了を担うクラスがActivityクラス**だからです。通常のプログラムではmainメソッドから始まりますが、Androidの場合はonCreateメソッドから始まります。onCreateメソッドが実行されたあとはどうなるかというと、ユーザからのイベント待ち状態になります。イベントとは、たとえばボタンがタップされたときや、ある座標がタップされたときにそれに対応させた自前のメソッドが呼ばれることを指します。そしてバックキーなどを押してアプリケーションを終了させるときにonDestroyメソッドが呼ばれてアプリケーションが終了します。

このonCreate、onDestroyメソッドはアプリ

のライフサイクル[注1]と呼ばれるしくみの一部です。ライフサイクルはアプリの状態(開始、終了、停止中など)を表しており、状態遷移時にそれに対応したコールバックメソッドが呼び出されます。Activityクラスにそのライフサイクルのコールバックメソッドが実装されているため、使用時にはActivityクラスを継承し、ライフサイクルに対応したコールバックメソッドをオーバーライドします。Eclipseで新しいプロジェクトを生成すると、Activityクラスを継承したクラスと、@OverrideアノテーションのついたonCreateメソッドが生成されるのはこのためです。他人のコードを参考にするときにはライフサイクルのメソッドかそうでないかを意識して見ると読みやすいでしょう。

▼図1　Googleアプリに見るActionBar

Google推奨デザインパターン ActionBar

ActionBarはタブレットが発表されたAndroid 3.0から導入された、Googleの推奨するデザインパターンです。たとえば**図1**のようにGmailやGoogle Map、Google Playアプリに使われています。これらのアプリを見ると、アプリ画面上部／下部にあるスペースにアイコンやタブ、テキスト、検索フォームが配置されているのがわかります。この上部のスペースのことをActionBar、下部のスペースのことをSplitActionBarと言います。このデザインパターンの狙いは**UIを統一し、機能を集約することでユーザに直感的に操作を理解してもらう**ことにあります。UI設計する場合にはこのデザインパターンを意識するとよいでしょう。

そして先日の7月のアップデートでActionBarに関する重要なアップデートがありました。それがActionBarActivityの導入です。これまでAndroid 3.0以上でなければActionBarを使うことができませんでしたが、**ActionBarActivityを使うことでAndroid 2.1以上の端末からActionBarを表示させることができる**ようになりました。つまり2.x系～4.x系のユーザすべてに統一的なUIを提供できるようになったのです。

ActionBarActivityはActivityを継承したクラスです。ですので、Activityの代わりにActionBarActivityに変更するだけで使うことができます。ただし、利用するにはサポートライブラリを導入する必要があります。

古いバージョンでも新しい機能を提供するサポートライブラリ

Androidには**古いバージョンの端末で新しい機能を使う手段としてサポートライブラリ**があります。たとえばAndroid 2.1ではまだActionBarがサポートされていなかったため、ActionBarでプログラムされたアプリは2.1では走りません。つまり自動的に3.0未満の端末は排除されてしまっていたわけですが、サポートライブラリを組み込むことでそこを穴埋めすることができます(具体的な導入方法については後述)。

現在のAPI 18ではv4、v7、v13の3バージョンのサポートライブラリが提供されており、そ

注1) onCreate、onDestroyのほかにもいくつかライフサイクルメソッドがあります。
参考：TechBooster URL http://bit.ly/AndroidLifecycle

れぞれAPI 4、7、13以上の端末、つまりAndroid 1.6、2.1、3.2以上の端末で同じ機能がサポートされています。たとえばv7サポートライブラリを使えば、Android 2.1以上の端末に一貫して同じ機能を使うことができます(ただし、含まれる機能は限定されています)。v7はappcompat、glidlayout、mediarouterライブラリの3種類が用意されており、必要に応じてアプリに組み込みます。今回使用するActionBarActivityはv7のappcompatに含まれています。

HTML/CSS/JavaScriptを表示・実行できるWebView

WebViewはその名のとおりWebページを表示するための機能で、HTMLやCSS、JavaScriptを表示・実行してくれます。blogの記事やコメントなどを修飾するときにHTMLを使ったことのある人は多いでしょう。素人から玄人まで簡単に文書整形ができるのがHTMLの利点です。よりデザインを洗練させたいのであればCSSを、よりインタラクティブにしたいのであればJavaScriptを使うなど拡張性が高いのも魅力的です。もちろんWeb上のコンテンツを表示させることもできますが、アプリ内にHTMLファイルを含めることもできます。今回はアプリ内にファイルを含める方法を解説します。

Tips表示アプリの作成

アプリ開発を進めていくと、どうしてもアプリのUI/UXだけではユーザに理解されないと思われる操作が出てきます。そういったときにTipsダイアログを出すことで、その場その場でユーザにアプリの使い方を理解してもらうことができます。今回のアプリでは、右上に表示されたアクションボタンをタップすることでTipsを表示させるものを作成します。

以降の作業をしてできあがったサンプルプロジェクトを筆者のGitHub[注2]にアップしていますので、作業の参考にしてください。ただし、v7サポートライブラリの導入はご自分の環境でしてもらう必要がありますので注意してください。

プロジェクトの作成

まず、Eclipseを立ち上げて[File]－[New]－[Android Application Project]を選択し、新しいプロジェクトを作りましょう。「Application Name」では、Package Name以外はデフォルトのままで[Next]をクリックしていけばOKです。サンプルプロジェクトでは、

注2) https://github.com/ksksue/TipsApp

Column

WebViewのセキュリティ脆弱性には注意

WebViewはブラウザを実現するために使われますが、AndroidのWebViewはセキュリティの問題上、不特定多数のWebサイトに接続するのに適していません。たとえばキャッシュされているIDとパスワードが第三者から覗き見されてしまう場合があります。fileスキーマを使ってWebView権限でローカルデータベースファイルにアクセスし、平文のままのIDとパスワードを読み出すことができてしまうという方法があります。たとえばAndroidにある標準ブラウザからfile:///data/data/com.android.browser/databases/webview.db(Android 3.x以下)、file:///data/data/com.google.android.browser/databases/webview.db(Android 4.0以上)にアクセスするとそのデータベースを見ることができます(Chromeが標準ブラウザの場合は見ることができません)。とはいえ、第三者による攻撃のない閉じられた空間で使うのであればWebViewはとても書き心地の良いキャンバスといえます。

参考文献:
『Androidセキュリティ勉強会～WebViewの脆弱性編～』株式会社イエラエセキュリティ
(資料　http://ierae.co.jp/uploads/webview.pdf)

- Application Name : TipsApp
- Package Name : com.physicaloid.TipsApp
- Activity Name : TipsActivity

というプロジェクトを作りました。ちなみにサンプルプロジェクトをインポートする場合は、Eclipseの［File］－［import］から「Existing Android Code Into Workspace」を選択し、「Root Directory」にサンプルプロジェクトへのパスを設定して［Finish］をクリックすればOKです。

▼リスト1　AndroidManifest.xml変更前／変更後

```
AndroidManifest.xml変更前
…略…
    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
…略…

AndroidManifest.xml変更後
…略…
    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/Theme.AppCompat.Light" >
…略…
```

v7サポートライブラリを導入する

誌面の都合上導入方法をここで書くスペースがなくなってしまったため、サンプルプロジェクトのGitHubページに書きましたので、そちらを参考にしてください。元情報は公式ページの「Adding libraries with resources」の Using Eclipse[注3]です。

ActionBarActivityを使うための準備

さて、v7サポートライブラリを導入したらActionBarActivityを使うための変更を加えていきましょう。まずAndroidManifest.xmlのテーマを変えます。デフォルトのテーマAppThemeからサポートライブラリに含まれているTheme.AppComapt.Lightに変更します（**リスト1**）。これは古いAndroidのバージョンのテーマにAppThemeがないため、v7 appcompatのテーマを使用するための変更です。

次に、アクティビティのあるTipsActivity.javaファイルを変更します。**リスト2**にプロジェクト生成時からの変更個所すべてを太字にしています。その中でActionBarActivityを使うための変更点・注意点は次の3点です。

1. Activityを ActionBarActivityに変更する（❷）
2. 必要なパッケージをインポートする（❶）
3. メニューの表示方法設定時にMenuItem Compatを使う（❺）

3.を使う理由もテーマと同じ理由で、通常使われるMenuItemでは古いAndroidのバージョンのメニューの表示方法を設定できないため、サポートライブラリ（この場合はv4）のMenu ItemComaptを使用します。すると**図2**、**3**のようなアクションボタンをつけることができます。

Activityのコールバックメソッド

前半の解説でActivityはコールバックメソッドがいくつかあることを紹介しました。このサンプルでのActivityのコールバックメソッドは3つ、onCreate、onCreateOptionsMenu（❹）、onOptionsItemSelected（❻）とあります。そのうちonCreateメソッドがライフサイクルのメソッド、onCreateOptionsMenuがメニューを生成するときに呼ばれるメソッド、onOptionsItem Selectedメニューが選択されたときに呼ばれるメソッドです。

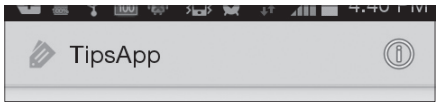
onCreateOptionsMenu内ではアクションボタンを生成しています。❸で作った定数はメニュー番号のための定数で、onOptionsItemSelectedが呼ばれたときのメニュー番号と紐付けています。また、AndroidにはActionBarに表示させるた

注3）http://bit.ly/SetupSupportLib

▼リスト2　TipsActivity.javaソースコード

```
package com.physicaloid.tipsapp;

import android.app.AlertDialog;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.webkit.WebView;
import android.support.v4.view.MenuItemCompat;
import android.support.v7.app.ActionBarActivity; ―――――❶

public class TipsActivity extends ActionBarActivity { ―――――❷

    static final int MENUID_TIPS = 0; ―――――――――――――――❸

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_tips);
    }

    // 終了時にonDestroyが実行される。ただし今回は終了処理が必要ないため省略する
/*  @Override
    protected void onDestroy() {
        super.onDestroy();
    } */

    // アプリ開始時メニュー作成のために呼ばれるコールバックメソッド
    @Override
    public boolean onCreateOptionsMenu(Menu menu) { ―――――❹
        MenuItem item = menu.add(Menu.NONE, MENUID_TIPS, Menu.NONE, "Tips"); // アクションボタンを生成
        item.setIcon(android.R.drawable.ic_menu_info_details);               // 標準のインフォアイコンを使う
        MenuItemCompat.setShowAsAction(item, MenuItemCompat.SHOW_AS_ACTION_IF_ROOM); ―❺
        // アクションボタンの数が多くなっても収まる範囲で表示させるSHOW_AS_ACTION_IF_ROOM属性にする
        return true;
    }

    // アクションボタンがタップされたときに呼ばれるコールバックメソッド
    @Override
    public boolean onOptionsItemSelected(MenuItem item) {―❻
        switch (item.getItemId()) {
        case MENUID_TIPS:
            showTipsDialog();   // Tipsダイアログを表示
            break;
        default:
            break;
        }
        return true;
    }

    /**
     * Tipsダイアログを表示
     */
    private void showTipsDialog() {
        WebView webview = new WebView(this);                    // WebViewオブジェクト生成
        webview.loadUrl("file:///android_asset/tips/index.html"); ―❼
                                                // assetsディレクトリのtips/index.htmlファイルを読み込む

        AlertDialog.Builder dialog = new AlertDialog.Builder(this); // ダイアログオブジェクト生成
        dialog.setTitle("Tips");                                    // ダイアログのタイトル設定
        dialog.setView(webview);                                    // ダイアログビューにWebViewをセット
        dialog.setPositiveButton("OK", null);                       // ダイアログにOKボタンをつける
        dialog.show();                                              // ダイアログを表示
    }
}
```

▼図2　Android 4.0.3で表示させたActionBarのようす



▼図3　Android 2.1で表示させたActionBarのようす

▼リスト3　assets/help/index.html

```
<html>
 <head>
  <meta http-equiv="Content-Type" content="text/html;
  charset=UTF-8">
 </head>
 <body>

<tt>
 <h1>Tips Sample</h1>
 This is a tips application.<p>
 <img src="android.png"><p>
 以下のようにHTMLタグを使うことができます。<p>
…略…
</tt>
 </body>
</html>
```

めのアイコン類があらかじめ用意されているので、android.R.drawable.ic_menu～から引っ張ってきて、MenuItemのsetIconでセットできます。❺のMenuItemCompatで指定しているSHOW_AS_ACTION_IF_ROOMのオプションは、ActionBarの横幅が狭くてアクションボタンが表示しきれない場合に、まとめて1つのプルダウンメニューアイコンに集約します。

WebViewを使う

WebViewを使うにはWebViewオブジェクトを生成し、ビューにセットすることで表示できます。またページをロードするにはloadUrlメソッドを使います(❼)。loadUrlメソッドの引数には通常URLを指定しますが、ローカルファイルを使う場合にはassetsディレクトリにファイルを置き、そのパスを“file:///android_asset/<*file名*>”で指定します。サンプルプロジェクトの場合は、assets/tips/index.htmlにファイルを置いたため、引数に“file:///android_asset/help/index.html”を指定しています。

プロジェクト内のassets/help/index.htmlにはHTMLファイル(**リスト3**)や、それに関連する画像ファイルを置きます。すると**図4**のように表示をさせることができます。

▶図4　AcitonBarのインフォボタンを押し、Tipsを表示させたようす

まとめ

Activity、ActionBar、WebViewを使ったサンプルプロジェクトの解説は以上です。次回は次の方にバトンタッチしてNavigationDrawerについて解説していただきます。連載2回分と短かったですが最後まで読んでいただき、ありがとうございました。SD

鈴木 圭介（すずき けいすけ）

組込み系Androidエンジニア。AndroidとArduino/mbed/FPGAなどと通信可能なUSBシリアル通信用ドライバ「FTDriver」をオープンソースで公開。現在、機能拡張させた「Physicaloid Library」を公開中 URL http://www.physicaloid.com/
フリーで仕事募集中。

テキストデータならお手のもの

開眼シェルスクリプト

USP友の会／産業技術大学院大学　上田 隆一 UEDA Ryuichi　Twitter @ryuichiueda

第22回 文章を扱うときに便利な技――sed、awk、find、grepの組み合わせ

はじめに

みなさん、今年の夏は暑かったでしょうか？執筆時点で夏の入り口にいますが、急に暑くなったので脳が溶けております。

そんな季節の挨拶とはまったく関係ありませんが、今回と次回はメモ書きや原稿など、不定形のテキストファイルのハンドリングを扱います。今回はシェルスクリプトというよりは、便利なコマンドの使い方を羅列していきます。

ドキュメントを扱う方法は、筆者のようにテキストファイルでHTML、reStructuredText、TeXを書く玄人気取りのやり方と、表計算ソフトに方眼紙を作って書く[注1]（別の意味で玄人っぽい）方法に二分されます。いや、嘘です。ワープロソフトを使う人が多数派のような気がします。

どの方法が良いかはその人の使い方にもよるのですが、おそらくこれらの方法の良し悪しが大きく現れるのは検索するときです。ワープロソフトや表計算ソフトにテキストを書いた場合、後で複数のファイルから文字を探すときは、基本的にそのソフトウェアのベンダが作った親切なツールを使うことになります。しかし、ベンダの気まぐれで何が起こるかわからないという怖い部分もあります。一方、テキストファイルで持っていれば、ベンダの干渉を受けないでしょうが、findやgrepといったコマンドを使いこなす必要があります。

この議論はやり出すとキリがないので、これ以上はやめておきます。とにかく今回は、「テキストデータならお手のもの」と連載タイトルに書いてあるように、「テキストファイルを便利に使うスキルを上げないとね！」という立場で話を進めます。

この連載の主張に呼応するように（嘘）、国もこういう流れになってきたようです。

(2) オープンデータ推進の意義

これまでも政府は、各府省のホームページ等を通じて保有するデータを公開してきており、情報提供という観点では一定の成果が出ている。

ただ、これまでのホームページによる情報提供は、基本的に、人間が読む（画面上で又は印刷して）という利用形態を念頭に置いた形で行われており、検索も難しく、大量・多様なデータをコンピュータで高速に、横断的に又は組み合わせて処理・利用することが難しい。[注2]

いつもと違ってお役所文章ですが、これを格言代わりに本編に進みます。余談ですが、これを引用したWebの記事の1つに「これを受けてデータはExcelで公開すべきだ」という文章があってひっくり返りました。そうじゃないでしょう……。原本はプレインテキストで管理する、というのが大事だと筆者は考えています。

注1）ピンとこない人は「Excel方眼紙」で検索を。

注2）「二次利用の促進のための府省のデータ公開に関する基本的考え方（ガイドライン）（仮称）（案：WG後修正版）」より http://www.kantei.go.jp/jp/singi/it2/densi/dai3/siryou6.pdf

環境

今回もMacです。gsed、gawkのバージョンとともに**図1**にMacのバージョンを示します。コマンドの用例ではgsed、gawkで統一してありますが、Linuxの多くのディストリビューションでは、gsedはsed、gawkはawkで大丈夫です。

日本語原稿の文字数を数える

まずは簡単なところから。作文をしていて、何文字書いたか調べたいときがありますね。え？ない？　……あるということにしてください。たとえば、**図2**のファイルmistery内の文字数は、段落の先頭の全角スペースを入れてちょうど60文字です。

こういうときは、**図3**のようにやります。wcにオプション-mを付けると、今のロケールに合わせて文字数を数えてくれます。ロケールを変えると**図4**のように出力に違いが出ます。

しかし、これだと改行も記号も文字数にカウントされてしまっています。**図5**のようにtrやsedで字を削っておくと正解が出るので、正確に数えたいならこのようにします。

もう1つ2つ数え方を**図6**に紹介しておきます。gsedを使う方法は、筆者の手癖になっているものです。gawkの方法は、ロケールが日本語でもインストールされているAWKの種類によってはバイト数になってしまうので注意が必要です。

▼図2　例題ファイルその1

```
$ cat mistery
　朝目覚めると、私は全身を繭で
覆われた蛹になっていたのです。
私は大変困ってしまいました。「
会社に休みの連絡ができない。」
```

▼図3　文字数を数える

```
$ cat mistery | wc -m
      64
```

▼図4　ロケール（環境変数LANG）で挙動が変わる

```
$ echo $LANG
ja_JP.UTF-8
$ cat mistery | LANG=C wc -m
     184
$ cat mistery | LANG=ja_JP.UTF-8 wc -m
      64
```

▼図5　文字数を正確に数える

```
$ cat mistery | tr -d '¥n' | wc -m
      60
  全角スペースも数えたくない場合
$ cat mistery | tr -d '¥n' |
gsed 's/　//g' | wc -m
      59
```

▼図6　文字数を正確に数える

```
$ cat mistery | gsed 's/./&¥n/g' |
wc -l
      64
$ cat mistery | gsed 's/./&¥n/g' |
gawk 'NF!=0' | wc -l
      60
$ cat mistery |
gawk '{a+=length($0)}END{print a}'
60
  Macなどではgawkを明示的に指定しないと
  以下のようになってしまうので注意
$ cat mistery |
awk '{a+=length($0)}END{print a}'
180
```

▼図1　環境

```
$ uname -a
Darwin uedamac.local 12.2.1 Darwin Kernel Version 12.2.1: Thu Oct 18 16:32:48 PDT 2012; ➪
root:xnu-2050.20.9~2/RELEASE_X86_64 x86_64
$ gawk --version
GNU Awk 4.0.2
Copyright (C) 1989, 1991-2012 Free Software Foundation.
  （以下略）
$ gsed --version
GNU sed version 4.2.1
Copyright (C) 2009 Free Software Foundation, Inc.
  （以下略）
```

もっと長い文章について、どれだけ書いたかざっくり知りたい場合は、バイト数で考えても良いでしょう。たとえば筆者はこの原稿を毎月6ページずつ書くのですが、ほかの月と比較してどれだけ書いたか、wcコマンドで**図7**のように調査しています。この例では、あと4ページくらい書かなければ原稿料をいただけないようです。これでバイトあたりの原稿料が計算できますが、雑念が入るので計算しないでおきます。

文章の抜き出し

テキストファイルの場合

次に扱うのは、テキストファイルの一部分を抜き出すテクニックです。たとえば、**図8**のような連絡先メモがあるとします。

たとえば濱田さんの電話番号が知りたい場合、普通にgrepを使おうとしても、

```
$ grep 濱田 address
- 濱田
```

▼図7　どれだけ書いたかバイト数や行数でざっくり調べる

```
$ wc 201???.rst
     549     803   24653 201201.rst
     428    1064   19469 201202.rst
      （中略）
     514     945   20703 201307.rst
     554     948   18805 201308.rst
     482     905   20520 201309.rst
     165     314    6616 201310.rst
   11016   21368  448677 total
```

▼図8　例題ファイルその2

```
$ cat address
<幹事会>

- 鎌田
        - 略称:（鎌）
        - TEL: 090-1234-xxxx
        - email: kama@kama.gov

- 濱田
        - 略称:（ハ）
        - TEL: 080-5678-xxxx
        - email: ha@haisyou.ac.jp
```

となります。このような残念な目にあったことのある人もいると思います。

実は、grepには-Aというオプションがあります。これを使うと**図9**のように、検索で引っかかった行の後ろも出力してくれます。これでいちいちlessを使ったりエディタ開いたりしなくて済みます。lessを使うことをそこまで面倒くさがる理由もないのですが……。

逆に、電話番号から人の名前を検索してみましょう。**図10**のようにします。

HTMLファイルの場合

次はHTMLファイルを扱います。HTMLの中から狙ったところをワンライナーで切り出してみましょう。これから扱うような処理は、ブラウザでソースを表示してマウスでコピー&ペーストでも良いのですが、何十、何百も同じ処理を繰り返すことになったらそうもいきません。

まず、筆者のブログからコードの部分だけを切り取るということをやってみます。2013年7月14日現在で、筆者のブログのトップページ[注3]にはいくつかコードが掲載されているのですが、コードはHTML上で<pre>と</pre>に囲まれ

注3）http://blog.ueda.asia

▼図9　grepの-Aオプション（検索でヒットした後ろの行を出力）

```
$ grep 濱田 -A 3 address
- 濱田
        - 略称:（ハ）
        - TEL: 080-5678-xxxx
        - email: ha@haisyou.ac.jp
```

▼図10　grepの-Bオプション（検索でヒットした前の行を出力）

```
$ grep 080-5678-xxxx -B 2 address
- 濱田
        - 略称:（ハ）
        - TEL: 080-5678-xxxx
  補足：-Aと-Bを併用することも可能
$ grep 080-5678-xxxx -B 2 -A 1 address
- 濱田
        - 略称:（ハ）
        - TEL: 080-5678-xxxx
        - email: ha@haisyou.ac.jp
```

ています。**図11**のようにcurlコマンドでHTMLを取得してlessで読んでみましょう。このような部分がいくつか出現します。

このような抽出はsedの得意技で、**図12**のようにコマンドを書けばコード(pre要素)だけを抽出できます。

ここでのポイントは、sedの使い方とcurlしたらすぐにnkfをすることの2点でしょう。

sedについては、本連載では文字列の置換で使うことがほとんどですが、***/<正規表現1>/,/<正規表現2>/p***(pコマンド)で、<正規表現1>にマッチする行から<正規表現2>にマッチする行まで抜き出すことができます。この処理は<正規表現2>のマッチが終わると再度実行されるので、**図12**の例ではいくつもpre要素を抜き出すことができています。sedはデフォルトで全行を出力するのですが、オプション-nはそれを抑制するために使っています。-nを付けておかないと、pコマンドの出力対象行が2行ずつ、そのほかの行が1行ずつ出力されてしまいます。

curlの出力は、たとえ読み取ったHTMLがUTF-8で書いてあっても改行コードがUNIX標準のものと違っている可能性があるので、このようなときは必ず通します。オプションは-wLuxが筆者の場合は手癖になっており、

- w：UTF-8に変換
- Lu：改行コードをLF(0x0a)に
- x：半角カナから全角カナへの変換を抑制

という意味があります。

ただ、このようにHTMLがきれいに改行されていればあまり苦労もないのですが、実際はそうもいきません。**図13**のようなHTMLもあるでしょう。

こういうときは、**図14**のように自分で掃除す

▼図13　例題のHTMLその2

```
$ cat kitanai.html
<pre>#!/bin/bash

echo "きたない"</pre>あははは<pre>
#!/bin/bash

echo "きたなすぎる"
</pre>
```

▼図11　例題のHTML

```
$ curl http://blog.ueda.asia | less
 (略)
<pre class="brush: bash; title: ; notranslate" title="">
Python 2.7.2 (default, Oct 11 2012, 20:14:37)
 (略)
&gt;&gt;&gt; round(-1.1,-1)*1.0
-0.0
</pre>
 (略)
```

▼図12　コードだけを取り出す

```
$ curl http://blog.ueda.asia 2> /dev/null |
nkf -wLux | gsed -n '/<pre/,/<¥/pre/p' > ans
$ less ans
 (略)
<pre class="brush: bash; title: ; notranslate" title="">
Python 2.7.2 (default, Oct 11 2012, 20:14:37)
 (略)
</pre>
<pre class="brush: bash; title: ; notranslate" title="">
$ cat hoge.sh
#!/bin/bash -xv
 (略)
</pre>
 (以下略)
```

るしかありません。このsedのワンライナーはお世辞にもきれいとは言えないので、ちゃんとプログラムを書いたほうがいいかもしれません。ただ、結局このほうが早いことが多いです。

この例題の最後に便利な小ネタを。先ほどpreで抜き出したHTMLには

```
&gt;&gt;&gt; round(-1.1,-1)
```

などと、記号の一部が文字実体参照に変換されています。たとえば**>**は**>**が置き換わったものです。

また、次のように数値参照になっているときもあります。

```
&#x4e0a;&#x7530;&#x53c2;&#x4e0a;
```

HTMLから抜き出してきたら、このままにするより元に戻したほうが良いでしょう。数値参照のほうは**図15**のようにnkfでできます。

文字実体参照のほうはnkfではできません。残念。しかし、**"**、**&**、**<**、**>**とスペース程度ならあまり個数がないので**図16**のようにsedスクリプトを書くと良いでしょう。コマンド化してもいいですね。

ほかの文字実体参照も変換しなければならないときは、ほかの言語のライブラリを使って変換コマンドを書くのが一番簡単な方法です。しかし、みなさんには文字実体参照の一覧を掲載したWebサイトからtableを抜き出し、**図16**のようなスクリプトをワンライナーで作ることをお勧めしておきます。

▼図14　きたないHTMLを掃除するワンライナー

```
$ cat kitanai.html |
  <pre>の後に何か文字があると改行を差し込む
gsed 's;¥(<pre[^>]*>¥)¥(..*¥);¥1¥n¥2;g' |
  <pre>の前に何か文字があると改行を差し込む
gsed 's;¥(..*¥)¥(<pre[^>]*>¥);¥1¥n¥2;g' |
  </pre>の前に何か文字があると改行を差し込む
gsed 's;¥(..*¥)</pre>;¥1¥n</pre>;g' |
  </pre>の後に何か文字があると改行を差し込む
gsed 's;</pre>¥(..*¥);</pre>¥n¥1;g'
<pre>
#!/bin/bash

echo "きたない"
</pre>
あははは
<pre>
#!/bin/bash

echo "きたなすぎる"
</pre>
```

▼図15　数値参照をnkfでデコードする

```
$ echo '&#x4e0a;&#x7530;&#x53c2;&#x4e0a;' |
nkf --numchar-input
上田参上
```

find、grep、xargsの組み合わせ

最後にファイルの検索をやってみます。ディレクトリの中から何かテキストを探すときは、findとxargsを組み合わせると自由自在な感じになります。

findは名前で誤解を受けやすいのですが、単

▼図16　文字実体参照を置換するsedスクリプトを作って使う

```
  このようなsedスクリプトを作る
$ cat ref.sed
s/&lt;/</g
s/&gt;/>/g
s/&quot;/"/g
s/&amp;/¥&/g
s/ / /g
$ curl http://blog.ueda.asia 2> /dev/null |
sed -n '/<pre/,/<¥/pre/p' | gsed -n '1,/<¥/pre/p' |
sed -f ./ref.sed
<pre class="brush: bash; title: ; notranslate" title="">
  (略)
>>> round(-1.1,-1)*1.0
-0.0
</pre>
```

に指定したディレクトリの下のファイルやディレクトリを延々と出力するだけです。findはオプションが多いことでも知られていますが、**図17**のような使い方だけ知っておけば良いと思います。オプションの`.`はカレントディレクトリ、`-type f`はファイルだけを表示しろということです。

出力は1レコードに1ファイル／ディレクトリと、UNIXの教科書どおりですので、ファイル名で検索するときはパイプでgrepをつなげば良いということになります。

たとえば、ミーティング中にとっさにとったメモをどこに保存したか忘れたが、何を書いたかはうっすら覚えている場合(そしてメモを取るときは必ずファイル名にmemoかMEMOを入れている場合)、**図18**のようなワンライナーで探し出すことができます。

fgrepは正規表現をオフにしてgrepするコマンドです。`grep -F`でも同じです。この`fgrep -v /.`で、隠しファイル(先頭が`.`で始まるファイル)を除去しています。次の`grep -i memo`の出力で、メモのファイルの一覧ができます。これでも余計なファイルがたくさん引っかかると思いますので、続けて`grep -v`でどんどん除去していっても良いでしょう。

それに続く`xargs grep 徹夜`ですが、これはパイプから流れてきたファイルのリストをgrepにオプションとして渡しています。最後のgsedは、grepの出力(***ファイル名：マッチした行***)からファイル名の部分だけ残し、直前のgrepに引っかかったファイル名のリストを作るためのものです。

このような操作が自在にできれば、某OSで検索のときに出てくる犬に頼る必要はありません。findやgrepで検索をかけるときによく使うイディオムを挙げておきます(**表1**)。

終わりに

今回はテキストファイルをハンドリングするノウハウをいくつか紹介しました。この手のノウハウは無数にあり、ほんの一部分をつまみ食いでだらだら紹介してしまった感もありますが、CUIで自分の文章を管理するときに実際にどんなコマンドの使い方をしているのか、雰囲気くらいはお伝えできたかと思います。

次回は表記揺れに的をしぼって、何か作り物をしてみる予定です。表記ゆれのテストスクリプトを書きます。原稿書きもテストファーストの時代へ……(大げさ) SD

▼図17　findを使う

```
$ find . -type f | head -n 4
./.201203.rst.swp
./.201310.rst.swp
./.DS_Store
./.git/COMMIT_EDITMSG
```

▼図18　find、grep、xargsを組み合わせる

```
$ find ~ -type f | fgrep -v "/." | grep -i memo | xargs grep 徹夜 | gsed 's/:.*//' > hoge
$ cat hoge
/Users/ueda/Dropbox/USP/memo/memo
$ cat hoge | xargs cat
徹夜で仕事しろと言われた(このメモはフィクションです。)
```

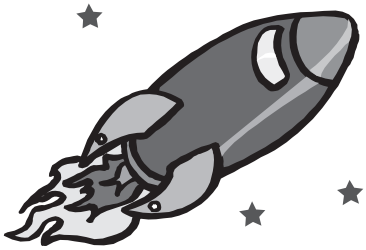
▼表1　findやgrepでよく使うイディオム

イディオム	意味
`find . | grep hoge`	ファイル名の検索
`grep -r hoge ./`	ディレクトリ下の全ファイルの中身を検索
`grep -r hoge ./ | gsed 's/:.*/' | uniq`	ディレクトリ下の全ファイルの中身を検索し、ファイル名のリストを抽出
`grep -r hoge ./ | gsed 's/:.*/' | uniq | xargs cat`	ディレクトリ下の全ファイルの中身を検索し、ファイル名のリストを抽出し、抽出したファイルの中身を表示

8

Debian JP Project　佐々木 洋平（ささき ようへい）
uwabami@debian.or.jp　Twitter：@uwabami

Ruby in Debian (2)



Debian Hot Topics

はじめに

2013年8月11日から18日の期間に、毎年1回、世界中のDebian開発者が集うDebconfがスイスで開催され、この会議のセッションの1つである「Ruby BOF」において、Debianにおける Ruby関連の状況のサマリと今後のロードマップについて議論が行われました[注1]。このセッションの結果として、次の安定版（現状のテスト版）「Jessie」に向けた状況のサマリ／タスクがDebian Wiki - Teams/Ruby/Jessie[注2]にまとめられています。都度アップデートされていく予定ですので、興味のある方はぜひ一度ご覧ください。

さて今回は、前回に引き続き、DebianにおけるRubyとRubyに関連するソフトウェアのパッケージングについて紹介します。

RubyのHEADを追いかけるには？

Debianに限らず状況に応じて複数のRuby処理系を切り替える方法として最近メジャーなのは、前回紹介したrbenvとruby-buildを用いる方法ではないでしょうか。ruby-buildもWheezyではパッケージとなっており、aptでインストール可能です（**図1**）。

Wheezyで提供されているruby-buildは、2012年5月24日リリース版ですので多少古く、Ruby2.0系列は2.0.0-devまでしかありません。ですので、ruby-buildの本家にあるdefinition[注3]を持ってくるか、新しくdefinitionを用意することになります。ruby-buildの本家で提供されているdefinitionは（依存関係の解消のため）OpenSSLのビルドも行っていたりしますが、ここではDebianパッケージのruby1.9.1のBuild-Depends[注4]を用いて多少楽をしましょう。たとえば、ruby-2.0.0-p247をインストールする場合には**リスト1**の2.0.0p247

注1）当日の議論の様子を収録したビデオが次のURLで公開されています。URL http://penta.debconf.org/dc13_schedule/events/1065.en.html
注2）URL http://wiki.debian.org/Teams/Ruby/Jessie
注3）URL https://github.com/sstephenson/ruby-build
注4）パッケージを構築する際の依存関係の定義のこと。

▼図1　rbenv、ruby-buildをaptでインストール

```
$ sudo apt-get install rbenv ruby-build
...略...
$ ruby-build --help
ruby-build 20120524
usage: ruby-build [-v|--verbose] definition prefix
       ruby-build --definitions

  -v/--verbose      Verbose mode: print compilation status to stdout
  --definitions     List all built-in definitions
```

▼リスト1　2.0.0p247を用意する

```
install_package "ruby-2.0.0-p247" "ftp://ftp.ruby-lang.org/pub/ruby/2.0/ruby-2.0.0-p247.tar.gz#c351450a0bed670e0f5ca07da3458a" autoconf standard
```

を用意して依存するパッケージを**apt-get build-dep**で導入して**ruby-build**でrbenv管理下にインストールします(**図2**)。インストールしたら、さっそくrbenvでRuby2.0.0を使ってみましょう(**図3**)。

gem2debによるDebianパッケージ作成

Wheezyからは、Gemで配布されているソフトウェアをDebianパッケージに変換するためのソフトウェアであるgem2debが用意されています。インストールは次のように行います。

```
$ sudo apt-get install gem2deb
```

さっそくパッケージを作成してみます。例としてcolorator[注5]をDebianのパッケージにしてみます(**図4**)。

図4を見てわかるとおり、gem2debは

①gemファイルを展開して、ruby-colorator-0.1.tar.gzを生成
②ruby-colorator-0.1ディレクトリにおいてDebianパッケージ用のひな形を作成
③パッケージのビルド

を行っています。もともとのgemにおいてtestやspecが提供されている場合、パッケージのビルド時にはこれらを実行します。今回のcoloratorでは、このテストに失敗したため、パッケージの作成を継続するか問われました。パッケージにしたいgemの挙動がわかっている場合や、とりあえず試しに使ってみたい場合には、今回のように"Y"を入力し、とりあえず進めるのも良いでしょう。

作成されたパッケージはruby-colorator_0.1-1_all.debというDebianパッケージとなっています。これをインストールして、実際に試してみましょう(**図5**)。

注5) URL http://rubygems.org/gems/colorator

▼図2　ruby-buildでRuby 2.0.0-p247をインストール

```
buildに必要なパッケージを導入
$ sudo apt-get build-dep ruby1.9.1-dev
rbenv管理下にインストール
$ ruby-build --verbose 2.0.0-p247 `echo ↵
$HOME`/.rbenv/versions/2.0.0-p247
...略...
```

▼図3　rbenvで確認

```
$ rbenv rehash
$ rbenv versions
  1.8.7-debian
  1.9.3-debian
  2.0.0-p247
$ rbenv global 2.0.0-p247
$ ruby -v
  ruby 2.0.0p247 (2013-06-27 revision ↵
41674) [x86_64-linux]
```

▼図4　gem2debでcoloratorをDebianパッケージにする

```
$ gem fetch -V colorator    ←colorator.gemのダウンロード
...略...
Fetching: colorator-0.1.gem (100%)
200 OK
Downloaded colorator-0.1
$ gem2deb colorator-0.1.gem    ←gem2debを実行
-- Creating source tarball from colorator-0.1.gem ...
...略...
-- Successfully created colorator-0.1.tar.gz

-- Creating Debian source package from colorator-0.1.tar.gz ...
...略...
-- Generated Debian source tree in ruby-colorator-0.1
-- Building Debian package ...
...略...
Test "ruby1.8" failed. Continue building the package? (Y/N) Y    ←とりあえず進める
...略...
Test "ruby1.9.1" failed. Continue building the package? (Y/N) Y    ←とりあえず進める
...略...
dpkg-deb: `../ruby-colorator_0.1-1_all.deb'にパッケージ`ruby-colorator'を構築しています。
...略...
-- Debian package successfully built!
```

テストを通すには?

さて、coloratorにはRSpec用のspecディレクトリが存在します。gem2debはこのディレクトリを検知し、ruby-coloratorパッケージの構築時のテスト対象としてspecディレクトリ以下のファイルをruby-test-files.yamlに列挙しています(**図6**)。

また、gem2debはパッケージの作成にdh_rubyを使用するひな形を作成します。dh_rubyにおけるパッケージ作成時のテスト実行方法は3通り用意されており、

- **debian/ruby-test-files.yaml**に列挙されているファイルを1つずつ実行
- **debian/ruby-tests.rb**を実行
- **debian/ruby-tests.rake**を実行

のいずれかを行います[注6]。coloratorの場合はRSpecを実行すれば良いので、debian/ruby-test-files.yamlを削除し、debian/ruby-tests.rakeを作成します(**図7**)。

最後に、debian/control内のBuild-Dependsに**rake, ruby-rspec**を追加しましょう(**リスト2**)。

では、再度パッケージを作成してみます(**図8**)。エラーが変わりました。RSpec自体は実行されていますが、Ruby1.8での実行に失敗しています。

▼図5　作成したruby-coloratorパッケージを試す

```
$ sudo dpkg -i ruby-colorator_0.1-1_all.deb
$ ruby -rcolorator -e 'puts "Hello".green'
Hello      ←緑色で表示される...ハズ
```

▼図6　gem2debが出力したひな形

```
$ cd ruby-colorator-0.1
$ ls
Gemfile  README.markdown  colorator.gemspec  lib/          spec/
LICENSE  Rakefile         debian/            metadata.yml
$ ls debian
changelog  control    ruby-colorator.docs   rules*   watch
compat     copyright  ruby-test-files.yaml  source/
$ cat debian/ruby-test-files.yaml
---
- spec/colorator/core_ext_spec.rb
$ ls -R spec
spec:
colorator/  spec_helper.rb

spec/colorator:
core_ext_spec.rb
```

▼図7　パッケージ作成中にRSpecを実行する設定

```
$ rm -f debian/ruby-test-files.yaml
$ vim debian/ruby-tests.rake ←編集
...略...
$ cat debian/ruby-tests.rake
require 'rspec/core/rake_task'
task :default => :spec
RSpec::Core::RakeTask.new(:spec)
```

▼図8　修正後のソースツリーでパッケージの作成(1)

```
$ sudo apt-get install rake ruby-rspec    ←追加した依存パッケージのインストール
$ cd ruby-colorator-0.1
$ debuild -rfakeroot -uc -us
...略...
/usr/bin/ruby1.8 -I/usr/lib/ruby/vendor_ruby /usr/lib/ruby/vendor_ruby/gem2deb/test_runner.rb
/usr/bin/ruby1.8 -S rspec ./spec/colorator/core_ext_spec.rb
F.

Failures:
   1) String contains all the methods from Colorator::ANSI_COLORS
      Failure/Error: Unable to find matching line from backtrace
...略...
```

注6) /usr/lib/ruby/vendor_ruby/gem2deb/test_runner.rb によってパッケージ作成時のテストが実行されます。テスト実行時に設定されるLOAD_PATHや環境変数などは実ファイルを参照ください。またruby-tests.rbの実例については dh_ruby(1)に例示されています。

サポートすべきRubyのバージョンは?

gem2debがサポートしているRuby処理系は、Debianが提供しているCRuby、すなわちRuby1.8.7とRuby1.9.1です。gem2debが生成したひな形のうちdebian/control内のXS-Ruby-Versionsにおいて、パッケージがどのRuby処理系で動作するかが指定されます。これを修正してみましょう(**リスト3**)。

この修正を加えたあと、再度パッケージをビルドしてみます(**図9**)。めでたくruby-coloratorのパッケージが生成されました! 最後にlintian(ポリシーチェックツール)により、公式パッケージとする場合のチェックが実行されており、debian/copyrightの中身がテンプレートのままですので、Eが出ています。Debianの公式パッケージとすることを考える場合には、これらのチェック項目をさらに修正していくことになります。

ライブラリの依存関係は?

今回、gem2debの例で取り扱ったcoloratorは、依存するほかのライブラリが比較的少ないGemでした。本来、Gemには依存関係が記載されています。今後はgem2debによってこの依存関係を適切にパースし、テスト時や実行時の依存関係まで含めたひな形を生成することで、より簡単にDebianパッケージが作成できるようになる、ハズです[注7]。

最後に

今回はDebianでRubyのHEADを追従する方法(ruby-buildのちょっとしたTips)と、gem2debによるGemからDebianパッケージの変換について解説しました。みなさんからのご意見、ご感想をお待ちしています。編集部(sd@gihyo.co.jp)宛にお送りいただくか、@gihyosdや@debianjp宛に一言いただければ幸いです。それではまた次回! SD

▼図9 修正後のソースツリーでパッケージの作成(2)

```
$ cd ruby-colorator-0.1
$ debuild -rfakeroot -uc -us
...略...
/usr/bin/ruby1.9.1 -I/usr/lib/ruby/vendor_ruby /usr/lib/ruby/ ↵
vendor_ruby/gem2deb/test_runner.rb
/usr/bin/ruby1.9.1 -S rspec ./spec/colorator/core_ext_spec.rb
...略...
Finished in 0.00577 seconds
2 examples, 0 failures
...略...
dpkg-genchanges: including full source code in upload
dpkg-source --after-build ruby-colorator-0.1
dpkg-buildpackage: full upload (original source is included)
Now running lintian...
W: ruby-colorator: wrong-bug-number-in-closes l3:#nnnn
W: ruby-colorator: new-package-should-close-itp-bug
E: ruby-colorator: helper-templates-in-copyright
Finished running lintian.
```

▼リスト2 debian/controlの修正

```
...略...
Uploaders: Youhei SASAKI <uwabami@gfd-dennou.org>
Build-Depends: debhelper (>= 7.0.50~), gem2deb (>= 0.4.1~), rake, ruby-rspec ←追記
Standards-Version: 3.9.3
...略...
```

▼リスト3 ruby1.9.1のみをサポートするようにXS-Ruby-Versionsを変える

```
...略...
#Vcs-Browser: http://git.debian.org/?p=pkg-ruby-extras/ruby-colorator.git;a=summary
Homepage: https://github.com/octopress/colorator
XS-Ruby-Versions: ruby1.9.1 ←allからruby1.9.1に変更
...略...
```

注7) DebianのRuby関連のパッケージは、先月触れたとおり**ruby-[Gemの名前]**というパッケージ名へ変更されている最中です。この移行が一段落したあとに、gem2debによる依存関係の解析/処理が有効になる予定です。

OpenStack as a 酒の肴

内藤 聡
NAITOU Satoshi
レッドハット㈱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グループ
ソリューションアーキテクト

恵比寿からこんにちは

こんにちは、はじめまして。内藤です。Linuxやクラウドといったプラットフォームのプリセールスとしてエンドユーザ担当をしています。さまざまな業種のさまざまなユーザ担当をしているので、先延ばしにしてきたUNIXシステムの大物をLinuxに移行したいというお話から、たくさんのRHELがあるので効率化して運用負荷を下げたい、さらにはCI(Continuous Integration)にDevOps的な要素を取り込みたい、といろいろな領域が楽しめるお仕事です。

レッドハットのプラットフォーム系エンジニアは、根っからの叩き上げOSSエンジニア(プロプラのソースコードを見るとOSSのパッチが書けなくなるから絶対見ない！——と宣言している人たちですね)やメーカー系が多い中で、筆者はユーザ系SIer出身と少し変わったバックグラウンドです。とは言え、元予備校の先生という方が2名いたり、筆者の隣の席には3年以上椅子の代わりにバランスボールに座って仕事している人(もはやバランスボールの仙人です)がいたりと、異彩を放つファンタジスタ達は実に多くのネタを提供しているので些細なことかもしれません。

やっぱりオープンソースが好き！

前職でもOSS(Open Source Software)は利用していましたが、とくにOSSが好きなんて気持ちはありませんでした。レッドハットに転職したのは、OSSだからというよりも面白い人たちがいて面白そうだったから、なんです。

そんな筆者もすっかりOSS漬けになり、ソフトウェアをより早くより良くするにはオープンソースが一番だ、と考えるようになりました。仕事PCもMacBook AirにFedoraを入れて利用しています。もちろんFedoraのシングルブートです。今なら「オープンソースベンダだからレッドハットにいる」と言える気がします(笑)。

エンジニアとしてOSSが素晴らしいと思う点をたった1つだけ挙げるとすると、レッドハットが滅んでもOSS(たとえばLinux)は不滅である、という点です。技術と会社が疎結合なのは良いことです。

この「結局OSSの何がうれしいの？」というネタはいくら語っても語り尽くせない、実に味わい深い酒の肴になります。OSSなんて好きでも何でもないというのが普通だと思いますが、なぜOSSが増えたのか、なぜMicrosoftやアップルのようなソフトウェアのビッグネームがOSSと言い出したのか、あらためて話のネタにしてみると意外と面白いかもしれないですよ？

きてますOpenStack

ここではOpenStackの特徴や機能、ロードマップのような話ではなく、OSSコミュニティという少し異なる視点から「きてる感」を観察してみます。OpenStackのプロジェクトはRackSpace社とNASAによって2010年に始まりました。実は、今からたったの3年前の話で比較的最近の出来事なんです。OpenStackは今も拡

張／成熟の過程にありますが、この原動力になっているのがまさしくコミュニティです。

それではOpenStackコミュニティの現状を見てみましょう。OpenStack.org[注1]によれば、すでに100ヵ国以上の国々から7,000名以上の参加があり、参加者の所属する団体／組織は850以上にもなるそうです（参考までにLinux Kernelの2004年から2009年のおよそ5年間における開発者の人数がおよそ5,000名でした）。最新リリースのGrizzlyでは、517名以上の参加者によって、230以上の新機能追加を含む7620ものアップデートがされました。OpenStackはおよそ6ヵ月のリリースサイクルを持っているので、毎日40程度のアップデートが取り込まれる計算になります。凄まじい開発力です。

OpenStack Foundationにも注目すべき点があります。2012年に設立されたこの組織は現時点（2013年7月末）では、プラチナメンバー8社、ゴールドメンバー16社、コーポレートスポンサー37社という大所帯です。規模もさることながら、注目すべきはメンバーへの加入条件[注2]です。プラチナメンバーへの加入には、3年間の継続を約束したうえで毎年$500,000を資金提供し、さらに運営スタッフや開発環境のインフラなどを提供することが条件として明記されています。ちなみに、ゴールドメンバーは、企業収益の0.025%（$50,000～$200,000）の資金提供が条件です。プラチナ／ゴールドメンバーの企業は資金提供とは別に多数の開発者をOpenStackにアサインしているわけですから、かなりの投資額になると思われます。ベンダーの本気度と熱気が伝わってくる気がしますね。当然ながら、個人レベルの参加に厳しい条件などはありませんし、資金提供を伴わない企業の参加も可能であることを追記しておきます。OpenStackのR＆Dはとてもオープンで誰もが参加できます。

OpenStackの明るい未来

OpenStackの「きてる感」は伝わったでしょうか？　OSSなもの、OSSでないもの、いろいろなIaaS（Infrastructure as a Service）ソフトウェアがある現状において、たった今この時点でOpenStackが一番良いか？　という問には意見が分かれることでしょう。しかし、ソフトウェア開発に終わりはなく、常に発展し続けることが要求されます。IaaSソフトウェアの場合では、ハードウェアや仮想化技術の進化、ユースケースの拡大といったものが発展することを要求します。この点において、さまざまな目的を持った多様かつ多数の開発者が参加するOpenStackコミュニティの開発力は大きな原動力になります。それはかつてLinux Kernelとそのコミュニティが切り開いた成功モデルとも言えるでしょう。

OpenStackが3年後10年後にどうなっているか、そんなことは誰にもわかりません。もしかすると、あなたのアイデアや貢献がOpenStackの今後に大きな影響を与えるかもしれません。OSS、OpenStackの話を酒の肴に大いに盛り上がってもらえるとうれしいです。あるいは、一緒に盛り上がりましょう！

恵比寿のごはん処

最後に恵比寿のランチネタを。紹介するのは恵比寿駅徒歩5分ほどにあるシンガポール料理屋さん新東記です。レッドハットはアジア地区の拠点がシンガポールにあるのですが、皆さんシンガポール料理を食べたことはあるでしょうか？　有名なのは海南チキンライスという、茹でた鶏肉とジャスミンライスをニンニクと生姜のタレでいただく料理です。シンガポールのあらゆるフードコートで食べられますが、新東記のチキンライスは現地よりも断然美味しいオススメの一品です。あ、がっつりニンニクなので、ニオイが大丈夫な日にぜひ。SD

注1）http://www.openstack.org/foundation/
注2）https://wiki.openstack.org/wiki/Governance/Foundation/Funding

第42回 Ubuntu Monthly Report

インプットメソッドの変遷とIBus 1.5

Ubuntu Japanese Team
あわしろいくや　AWASHIRO Ikuya　ikuya@fruitsbasket.info

今回はいつもとは若干指向を変えて、インプットメソッドの変遷の歴史と、Ubuntu 13.10から採用するIBus 1.5の大きな変更点を紹介します。

Ubuntuとインプットメソッド

Ubuntuで、日本語などの言語を入力するフレームワークをインプットメソッドといい、現在のデフォルトではIBusがこれに相当します。Ubuntu 13.10ではこのIBusが1.5になり、今までとはあり方がかなり変わることになります。というわけで、これまでのインプットメソッドの変遷と、IBus 1.5ではどのように変わるのかについて述べます。

前提知識

前述のとおりUbuntuではIBusがデフォルトのインプットメソッドなのですが、もう少し分解すると狭義のインプットメソッドのIBus、変換エンジンのAnthy、その2つをブリッジ[注1]するIBus-Anthyの3つで構成されています。ほかにも技術的な解説はあるのですが、本題とはかけ離れてしまうのでここでは触れません。興味がある方は、今回の執筆の参考にした本誌2007年3月号の第1特集をご覧ください[注2]。

前史

■Anthy

Anthyは、Ubuntuでインプットメソッドをサポートするようになったときから現在まで、変わらずデフォルトの変換エンジンです。2000年5月19日、当時京都大学のKMC（京都マイコンクラブ）に在籍していた一部のメンバで構成されるProject Hekeによって開発が開始されました。Anthyが好んで使われるようになった理由はいくつかあり、それ以前によく使われていたCannaやFreeWnnはシステムで動作するサーバで、同時に使用するユーザが1人しかいないような場合には大掛かり過ぎること、また1990年代以降コアにはあまり手を入れられておらず、変換精度もあまりよくなかったことがネックでした。その後Mozcが登場してもAnthyがデフォルトのままなのは、Ubuntuだとデフォルトにする場合はmainリポジトリに入る必要があり、さらにビルドに必要なパッケージもすべてmainになければいけません。ですがMozcは規模が大きいので、ビルドに必要なパッケージもいろいろとあり、交渉するだけの余力がないというのが理由です。もっとも、14.04を目指してMozcをデフォルトにする動きもあるのですが。

■SCIM

SCIMはIBusの前にデフォルトだったインプットメソッドです。いつ頃開発が開始されたのかはわかりませんでしたが、少なくとも2002年には存在しています。開発者は、当時Turbolinux China、Novellを経て現在Googleに在籍するJames Su氏です。当時はほかにもIIIMFやuimがありましたが、もっと

注1）IMModuleと言ったりもします。
注2）『Software Design総集編【2001～2012】』にも収録されています。

も多言語対応が優れていたのがこのSCIMでした。当初はAnthyのブリッジが存在せず、uimをライブラリとして使用するscim-uimをブリッジにしていました。uimではもちろんAnthyを使用できるので、[SCIM]-[scim-uim]-[uim]-[uim-anthy]-[Anthy]というまわりくどい使い方をしていました。

Ubuntu 4.10のころ

Ubuntuの最初のバージョンである4.10は、2004年10月にリリースされました。しかし、インプットメソッドは含まれていませんでした。そこで、筆者が2004年11月8日に4.10用のインプットメソッド一式のパッケージを公開しました[注3]。筆者は当時Debianを常用しており、パッケージングの知識があったのでUbuntuのビルド環境を用意し、それまでに作ったソースパッケージをリビルドすることによって用意できたのです。実際に用意したのは、前述のscim-uimを使用する構成です。

Ubuntuのリリース直後ですので、当然日本語のコミュニティはありませんでした。このパッケージをもとに筆者がコミュニティを作ることもできたのですが、それはしませんでしたし、もしできたとしてもこの時点では参加する気はなかったのです。

scim-anthyの開発開始

SCIMとAnthyのブリッジであるscim-anthyの開発が始まったのが、Ubuntu 4.10のリリース直後ぐらいです。開発者は当時、株式会社グッデイ[注4]、現在は株式会社クリアコードに在籍する足永拓郎氏です。同社は当時IPAの公募でインプットメソッド関連の開発を行っており[注5]、その一環で開発されました。筆者も同社で勤務しており、scim-anthyのテスター兼Debianパッケージのメンテナーという感じでした。scim-anthyが画期的にすばらしかったのは、Microsoft IMEやATOKなどとほぼ同じ使い勝手を提供できたことです。

Ubuntu 5.04のころ

4.10から5.04の間にUbuntu Japanese Teamの原型ができあがっていました。代表は当時株式会社グッデイ、現在合同会社コバヤシステムの小林準氏で、いうまでもなく筆者も巻き込まれていました。5.04用のインプットメソッド関連パッケージも用意しており、そこにはscim-anthyも含まれていました。さらに、のちに日本語Remixとなるローカライズ版の作成を開始したのもこの時期で、インストールから使用するまでの手間が大幅に軽減し、またインプットメソッドに関してもストレスなく使用できるようになるという、現在のUbuntuの基礎ができたのです。

Ubuntuの多言語サポート

Ubuntuが多言語サポートを行うことになったのは2006年2月頃で、日本語ではSCIMとscim-anthyとAnthyが使われることになりました[注6]。すなわちローカライズ版と同じで、そこでの実績も評価の1つになったのです。

また、Ubuntu Japanese TeamがUbuntuのオフィシャルコミュニティになったのもおおよそこの時期です。

Ubuntu 6.06のころ

6.06は2006年6月にリリースされました。初のLTS（Long Term Support）で、初の多言語サポート版です。本来は4月にリリースされる予定でしたが、その2つの「初」が主な理由で2ヵ月延期され、6.06というバージョンになりました。

IM-BUS構想とIBus

2007～2008年のことですが、James Su氏らを中心として新しいインプットメソッドの開発プロジェ

注3） http://blog.goo.ne.jp/ikunya/e/501df3776605901cfda8cfcb412fb814

注4） すでに倒産しています。

注5） http://ossipedia.ipa.go.jp/doc/82/

注6） https://wiki.ubuntu.com/MainInclusionReportscim

クトが開始しました。それがIM-BUSです[注7]。IM-BUSはD-BUSから着想を得て、GUI（設定画面）やブリッジ、各アプリケーションをプロセス間通信で結ぶというものでした[注8]。この構想自体は文句なくすばらしかったのですが、何らかの理由[注9]によって開発は進みませんでした。というのも、SCIMにはいくつかの問題があったのです。C++で書かれていたので開発コストが高いことと、stdC++にリンクする必要があったこと、1プロセスだったので、たとえばブリッジ（IMEngine）のバグで設定画面がセグメンテーション違反を起こした場合、SCIMまるごと落ちてしまう、ということがありました[注10]。

プロセス間通信なら別途新しく開発するのではなく、そのままD-BUSを使えばいいのではないかという発想から生まれたのがIBusです[注11]。開発者は当時Red Hat、現在Googleに勤務するHuang Peng氏です[注12]。（記憶では）最初から実用レベルだったこと、Red Hatの強力なプッシュがあったこと、SCIMのメンテナンスが滞っていたことなどにより、かなりの勢いで発展していきました。

Ubuntu 9.10のころ

2009年10月にリリースされたUbuntu 9.10から、SCIMがIBusになりました。すなわち、IBus、IBus-Anthy、Anthyという現在の構成になったのです。IBus-Anthyよりもscim-anthyのほうが優れている面も多いのですが、SCIMとIBusを比較すると言わずもがなであり、わざわざSCIMをインストールしなおしてまで使うことは少なくなっていったように思います。とくにIBus 1.4は極めて安定していたようにも思います。

Mozcの登場

2009年12月にGoogle日本語入力がリリースされ、そのオープンソース版としてMozcが2010年5月にリリースされました。とはいえ辞書を始めとしてオープンソース版とそうでないのとは差異が多く[注13]、またオープンソース版の辞書であるIPAdicは、当初はDebian的にnon-free（Ubuntu的にはMultiverse）でもありました。これは2011年11月に解釈しなおすことによってDebian的にmain（Ubuntu的にはuniverse）となりました。

開発スタイルもかなり独特で、外部のパッチはいっさい受け入れず[注14]、バージョン管理もGoogle内部のリポジトリからのコピーで、個々のコミットログなどはありません。よって、このリビジョンだけの差分を取得するということができません[注15]。

前述のとおりビルドの依存関係も複雑だったりするのですが、Anthyとは比べ物にならないほどの変換効率であり、現在では広く使われています。もちろんこの原稿もMozcで書いています。

Ubuntu 13.10のころ

IBusとGNOMEの統合

2012年9月にリリースされたGNOME 3.6とIBus 1.5[注16]を組み合わせることによって、GNOME とIBusを統合できるようになりました。「統合」というとわかりにくいですが、今まではIBusの設定は「キーボード・インプットメソッド」で行っていました

注7） https://code.google.com/p/imbus/

注8） http://www.slideshare.net/guestd8ac24/imbus

注9） IM-BUSに参画していた故・樋浦秀樹氏にお聞きしたところ、JamesさんがGoogleに転職したあたりからコミュニケーションが難しくなったとのことです。スモールスタートを図ることができなかった時点で（厳密に言えばスモールスタートだったのですが、継続できなかったので）、プロジェクトがうまく行くことはなかったように思います。

注10） http://2008.gnome.asia/static/upload/event_file/ibus-GNOME.pdf が詳しいです。

注11） とはいえこれは仮説ですが、状況証拠からそうだったとしか考えられません。

注12） Googleに転職したのはおそらくChrome OSでもIBusを採用することになったためと思われます。実際に採用されたのですが、現在のChrome OSではIBusは採用していません。確かにChromeしか動かさないのであれば、IBusのような汎用的なものを使用しないというのは合理的な判断です。ただ、それもあってIBus-Mozcのメンテナンスの重要度が下がりました。

注13） http://code.google.com/p/mozc/wiki/AboutMozc

注14） Mozcのパッチであれば Google内部のリポジトリにもコミットされることになりますが、Googleのポリシーで社外の人が書いたソースコードは一切Google内部のリポジトリに入れてはいけないのだそうです。

注15） 今のところはとくに問題になっていませんが、パッケージで困ることになる可能性が若干ながらあります。

注16） あるいはIBus 1.5の開発版であるIBus 1.4.99。

が、これが「システム設定」の「地域と言語」で行うようになりました(**図1**)。ようするに、IBusの設定がキーボードの設定と統合されたのです。そこまではよかったのですが、IBus自身が大幅に変更されました。具体的には後述します。

UbuntuではGNOMEが3.6になってもIBusは1.4のままだったので、この統合は有効になりませんでした。しかし、13.10からはIBusも1.5になり、覚悟が必要になりました。

IBus 1.5

IBus 1.5はGNOMEのポリシーとWindows 8の影響を受け、さまざまな変化がありました。まずはIBusの起動は、これまでは半角/全角キーで行えていましたが、今度からはSuper[注17]＋スペースキーで行うようになりました。ほかにもツールバー(言語パネル)がなくなりました。「すべてのアプリケーション間で同じインプットメソッドを共有する」がオンというのも大きな変更点です。これらはいずれもWindows 8での変更を受けてとのことです。それは確かに事実なのですが、Windows 8ではあくまでデフォルトであり、設定自体を変更することはできます[注18]。しかし、IBus 1.5では設定でどうにかすることはできません。これは、おそらくGNOMEのオプションは極力少なくするという方針に沿ったものだと思われます。その結果、Windows 8からPCを使い始めた人にとってはいいかもしれませんが、それ以外の人には極めて使いにくいものになってしまいました。

13.04の日本語RemixではMozcをデフォルトの変換エンジンにしました。これはIBus-Anthyのパッケージに手を加えることによって実現していますが、GNOMEと統合すると、このあたりの設定はGNOMEが持つようになるので、デフォルトの変換エンジンを変更する場合はgnome-settings-daemonに手を加えることになります。とはいえそれを日本語Remixでやるのは難しいので、13.10では同じようにできない可能性が極めて高いです[注19]。

UbuntuというかUnity的にも大きな変更があります。これまではIBusにUnityのIndicatorをサポートするパッチを適用していましたが、1.5ではこれをやめてindicator-keyboardというパッケージを使用するようになりました。

あと、まだよくわかりませんが起動時の挙動が変わる可能性があります。これまでは言うまでもなくログイン直後はIBusはオフだったのですが、今後はログイン直後からオンになる可能性もあります。ただし、半角/全角キーで英数切り替えはできます。

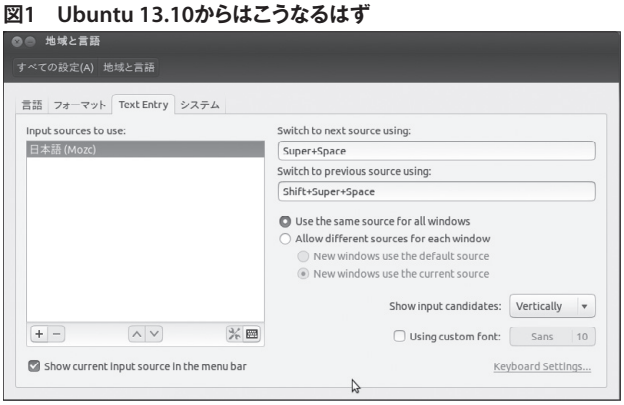
あくまで執筆の段階(2013年8月中旬)でのことですが、いろいろ不具合が多くてまともに使用できる状態になっていません。Unityの不具合でSuper＋スペースキーが動作しない(ので、Shift＋Super＋スペースキーを使用する)とか、indicator-keyboardが正しく動作しないとか、IBusやIBus-AnthyやIBus-Mozcにも不具合があるという満身創痍状態です。これは筆者も含め、リリースまでに改善する必要があります。

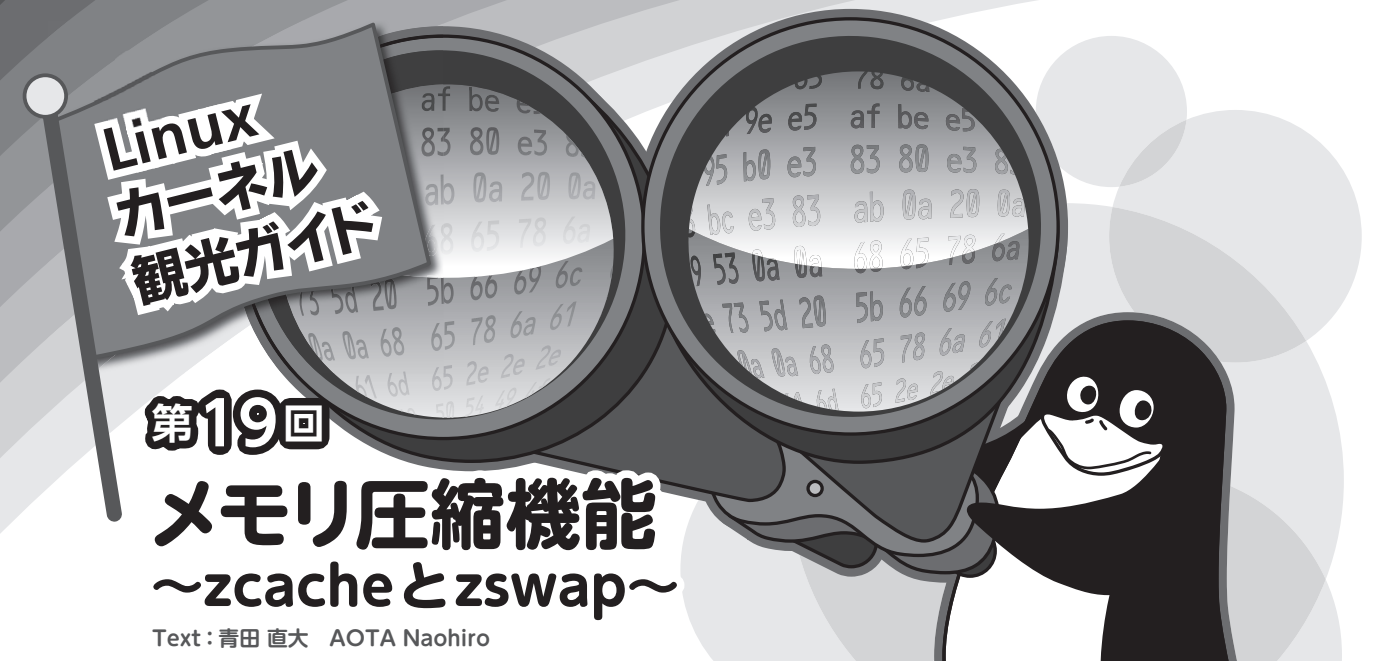
頼みの綱のFcitx

原則としてはIBus 1.5の挙動に慣れる必要があるとは思うのですが、どうしてもそれができない場合はFcitxというインプットメソッドを使用することになると思います。若干違う部分はあるものの、おおむねIBus 1.4以前と同じ使い方ができます。これはこれで不透明な部分も若干あるので、別途どこか[注20]で取り上げられたらいいと思っています。SD

注17) WindowsだとWindowsキーです。
注18) 変更するところは極めてわかりにくいですが。
注19) もちろんほかに手はないか考えてみます。
注20) かなり高い確率でgihyo.jpのUbuntu Weekly Recipeになります。

図1　Ubuntu 13.10からはこうなるはず





第19回 メモリ圧縮機能 ～zcacheとzswap～

Text：青田 直大　AOTA Naohiro

zcacheとzswap

ディスクへのI/Oというのは SSDでさえもCPUの速度と比べて、かなり遅いものです。すべてをメモリに載せてしまえれば良いのですが、そういうわけにもいきません。どうしてもしばらくは必要ではないであろうメモリ上のデータを、ディスクに追い出す必要があります。この作業で必要となるI/Oがシステムのパフォーマンスの足をひっぱってしまうこともあります。

そこでディスクへのI/Oを行わずにメモリを空ける方法として、しばらく必要のないメモリを圧縮しておく、という手法が考えられました。この実装としてzcacheとzswapという2つの実装がLinuxにはありました[注1]。これら2つのどちらをメインラインへとマージするのかが議論になってきましたが、3.11ではzswapのほうがマージされることとなりました。今月はこの2つのメモリ圧縮実装について見ていきます。

メモリ圧縮の大枠

zcacheもzswapも目的は、しばらく不必要なメモリを圧縮しておくことですから、その実装の大枠は似ています。つまり、メモリ圧縮システムがやるべきことは次の3項目にまとめることができます。

- しばらくは不必要なページを決めること
- 圧縮されたページを管理すること
- 圧縮されたページ用のメモリ領域を確保すること

これをふまえて、zcacheとzswapの中身を見ていくことにしましょう。

2種類の「不必要なページ」

「不必要」なページ（メモリの管理単位。Linuxがサポートする多くのアーキテクチャで4,096バイト）には2種類のものがあります。1つはanonymous page（anon page）などと呼ばれるファイルと関連付けされていないページです。もう1つはpage cacheなどと呼ばれるファイルとの関連付けがあるページで、一度ディスクから読み込んだファイルの内容が再度アクセスされたときに、ディスクからの読み直しを行わなくても良いようにキャッシュしているものです。

anon pageのほうはその内容がディスクにはありませんから、これをメモリから追い出したいときにはswapに書いて（スワップアウトして）どこかで保管しておかなければいけません。一

注1）厳密に言えばzramというものもあります。

方で、page cacheのほうはcleanであれば(そのページへの書き込みがなく、ディスク上の内容と一致していれば)、いつでも廃棄できます。

この2種類の不必要なページを抽象的に扱い自由に管理できるようにするAPIがLinuxカーネルにはあります。次にこの2つのAPI、frontswapとcleancacheを順番に見ていきましょう。

frontswap API

frontswapはその名前から想像できるとおり、「swapの前」に配置されて、スワップアウトされそうなページを扱うためのAPIです。

zswapなどのbackendのドライバはfrontswap APIに“struct frontswap_ops”で指定される関数を登録します。frontswapはスワップアウトされそうなページを横取り(もちろん実際にはスワップアウト処理の中でfrontswap_store()関数が呼ばれているだけです)し、❶backendが登録されていればbackendにそのページがスワップアウトされていくスワップデバイスの番号(frontswapのコード中では“swap type”や“type”と呼ばれています)と、ページのオフセットとページそのものの3つの引数で、登録されているbackendの“store”関数を呼び出します。❷backendのstore関数は渡されたページの内容を保存し、成功を示す0を返します。backendが保存に成功した場合、frontswapはswap typeごとに保持されているbitmapのoffsetに対応する部分のビットを立てておいて、backendにswapされるはずだったデータが保存されていることを覚えておきます。また、backendが失敗した場合は、通常のページをディスクにスワップアウトする作業が行われます。❸スワップアウトのときと同様にページがスワップインされるべきときにも、frontswapのfrontswap_load()関数が呼び出されています。この関数が先ほどのbitmapを見て、backendにそのページが保管されているかどうかをチェックし、保管されていればbackendのload関数を呼び出して、swap typeとページのオフセットに対応するデータをそのページに書き出してもらいます。

cleancache API

cleancache APIもfrontswap APIとほぼ同様に働きますが対象とするページが違っています。cleancacheは、もう一方の不必要なページであるcleanなpage cacheを対象にしています。こちらもzcacheなどのbackendのドライバが“struct cleancache_ops”で指定される関数を登録し、それにしたがって処理が行われます。

cleancacheに対応したファイルシステムがmountされるときには、cleancache_init_fs()関数が呼び出され、各ファイルシステムごとのキャッシュプールが作られ、このプールのIDがファイルシステムのスーパーブロックに記録されます。

page cacheが削除されるタイミングでこのページがcleanであれば、ページを引数にしてcleancache_put_page()関数が呼び出されます。この関数はページからプールのIDとinode(もしくはサポートされていればfile handle)とindex(そのページがinodeの何番目のページであるか)の情報を取り出し、これら3つ(まとめてハンドルと呼びます)を引数にして、backendのput関数を呼び出します。このページはcleanであるため、失われても何も問題はないのでbackendが保存に成功しても失敗しても、(frontswapのときに通常のスワップアウトを行ったような)特別な処理は必要はなく、ただいつも通りにpage cacheからそのページを削除します。

同様に、ディスクからの読み直しをするタイミングでcleancache_get_page()関数が呼び出されます。この関数が先ほどと同様のハンドルをbackendに渡します。backendにそのハンドルに対応するデータが保管されていれば、backendはそのデータをページに書き出し保管していたデータを(共有されているプールでなければ)削除します。ここでデータが削除されるのでcleancacheとpage cacheの両方に無駄に同じデータ

が入っている、ということはなくなります。

こういったfrontswapとcleancacheのおかげで、zswapとzcacheは「何が不必要なページであるのか」の判断をこれまでにあったカーネルの機構にお任せし、frontendから渡されてくるハンドルに対して保管と復帰を行えば良いということになります。

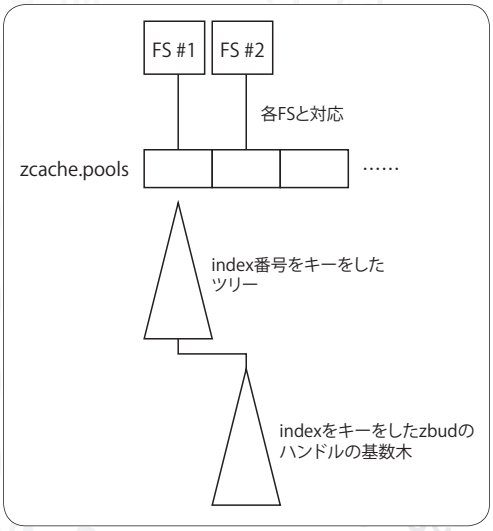
zswapのデータ管理

それでは、zswapがどうやってハンドルと圧縮されたページとを関連付けしているかを見ていきましょう。zswapはその名のとおり、swapだけを対象としていますので、frontswapから渡されるswap typeとページのオフセットのペアがキーとなります。zswapでは“zswap_trees”という配列で、swap typeごとに(つまりswap deviceごとに)1つの赤黒木[注2]を保持しています。この木にswap pageのoffsetをキーにして、“struct zswap_entry”のデータを保持します。“length”は圧縮されたページの長さ+“struct zswap_header”の長さとなっています。“handle”は後述するメモリアロケータzbudから返されるハンドルです。ここでは圧縮されたデータのアドレスを直接zswapが管理しているのではなく、ハンドルを管理していて、このハンドルを使ってメモリアロケータに適切にメモリをマップしてもらうことによって、初めてデータのアドレスがわかるということを気にとめていてください。

```
struct zswap_entry {
    struct rb_node rbnode;
    pgoff_t offset;
    int refcount;
    unsigned int length;
    unsigned long handle;
};
```

注2) 連想配列の実装に用いられている平衡二分木の一種。

▼図1 zcacheのデータ管理



zcacheのデータ管理

zcacheのほうはLinuxカーネルツリーに入っているコードも状況もいくぶん複雑です。zcacheはすでにdrivers/stagingには長らく入ってはいます。しかし、あくまでも“staging”の名のとおり、まだまだ準備途中で未完成なものです。この入っているzcacheのコードはzswapがswapだけを扱っていたのに対して、swapもpagecacheも扱っています。一方で、zswapがstagingを出てマージされたあとに、同様にstagingを出ることを狙ったzcacheのコード[注3]は、完全にpagecacheだけに集中し、コードも簡略化されています。今回はこのpatchをもとにzcacheがどのようにcleancacheから受け取ったページを管理していくのかを見ていきます(**図1**)。

zcacheでは“zcache.pools”という配列で、ファイルシステムに関連付けられたプールごとに1つの赤黒木を保持しています。これはinode番号をキーとしたツリーになっています。このツリーの要素が基数木のルートとなっています。この基数ツリーはindex(inodeの何番目のページであるか)をキーとしています。基数木の要素はメモリアロケータzbudから返されたハンドルとなります。

注3) https://lkml.org/lkml/2013/7/20/90

kmallocの問題点

それでは最後に、圧縮されたページ用のメモリの確保を行っているメモリアロケータzbudについて見ていきましょう。zbudについて話すには、まずそもそもなぜzbudのような新しいメモリアロケータを書く必要があったのかという疑問に答える必要があります。

Linuxカーネルの中でもユーザランドのプログラムでmalloc()するように、kmalloc()を使って任意のサイズのメモリを確保できます。このメモリの確保にはスラブアロケータが使われています。スラブアロケータはあらかじめある程度のメモリ領域を確保し、それをそれぞれ同じサイズに切り分けて必要とされたメモリを提供するシステムです。

メモリ領域がすべて同じサイズに分割されているので、あるサイズの領域があいているかどうかをチェックするのは使用中かどうかをチェックすればいいだけですし、フラグメンテーションの問題も起こらなくなります。/proc/slabinfoでどのような種類のslabがあるのかを見ることができます(**図2**)。その中に"size-n"というものがあります。"n"の部分がそのままメモリ分割のサイズになっています。kmalloc()では指定されたサイズ以上の最小の"size-n"を探し出し、そのslabからメモリを割り当てています。たとえば32バイトの割り当ては"size-32"から、50バイトの割り当ては"size-64"から割り当てされます。

ここで見たkmallocの性質がメモリの圧縮には不都合に働いています。たとえば、あるページが圧縮した結果2,049バイトになったとします。これをkmalloc()で確保すると"size-4096"から4,096バイトの領域が与えられます。なんとせっかくページをほぼ半分のサイズに圧縮できたのに、圧縮済みデータを書いておくのに結局1ページまるごと使ってしまいました！　これでは圧縮した意味がまったくありません。かといって、2,048バイト以下に圧縮できなければ却下するようでは多くのページが圧縮できないままになってしまいメモリ圧縮機能の利点を発揮できなくなってしまいます。

▼図2　/proc/slabinfoを表示したところ

```
$ sudo cat /proc/slabinfo|grep -v DMA
slabinfo - version: 2.1
# name            <active_objs> <num_objs> <objsize> <objperslab> <pagesperslab> : tunables <limit>
<batchcount> <sharedfactor> : slabdata <active_slabs> <num_slabs> <sharedavail>
...
size-4194304           0      0 4194304    1 1024 : tunables    1    1    0 : slabdata      0      0      0
size-2097152           0      0 2097152    1  512 : tunables    1    1    0 : slabdata      0      0      0
size-1048576           0      0 1048576    1  256 : tunables    1    1    0 : slabdata      0      0      0
size-524288            0      0  524288    1  128 : tunables    1    1    0 : slabdata      0      0      0
size-262144            0      0  262144    1   64 : tunables    1    1    0 : slabdata      0      0      0
size-131072            2      2  131072    1   32 : tunables    8    4    0 : slabdata      2      2      0
size-65536             0      0   65536    1   16 : tunables    8    4    0 : slabdata      0      0      0
size-32768             2      2   32768    1    8 : tunables    8    4    0 : slabdata      2      2      0
size-16384           277    282   16384    1    4 : tunables    8    4    0 : slabdata    277    282      0
size-8192             22     22    8192    1    2 : tunables    8    4    0 : slabdata     22     22      0
size-4096            437    437    4096    1    1 : tunables   24   12    8 : slabdata    437    437      0
size-2048            526    540    2048    2    1 : tunables   24   12    8 : slabdata    270    270     52
size-1024           1384   1384    1024    4    1 : tunables   54   27    8 : slabdata    346    346      0
size-512            3557   4544     512    8    1 : tunables   54   27    8 : slabdata    568    568    216
size-256             282    300     256   15    1 : tunables  120   60    8 : slabdata     20     20      0
size-192            1775   1880     192   20    1 : tunables  120   60    8 : slabdata     94     94      0
size-64            11629  22656      64   59    1 : tunables  120   60    8 : slabdata    384    384      1
size-128            5850   5970     128   30    1 : tunables  120   60    8 : slabdata    199    199      0
size-32            18514  36848      32  112    1 : tunables  120   60    8 : slabdata    329    329      0
...
```

メモリアロケータzbud

こうした問題に対処するためにzsmallocとzbudというメモリアロケータが開発されました。

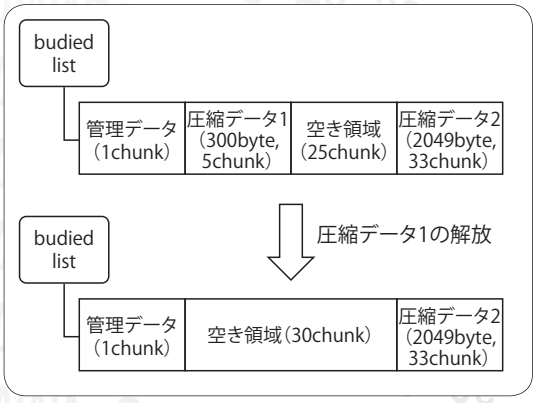
もともとzswap、zcacheはzsmallocを使用していたのですが、その複雑さからよりシンプルなzbudがLinux 3.11にはマージされ、現在のzswap、zcacheはzbudを使用しています。ここではzbudについて見ていきます。

zbudはその名の“bud”、つまり“buddy”から想像されるとおり1つのページに最高で2つの圧縮されたページデータを保持するようにしています(**図3**)。2つのうち一方のデータ領域はページの(管理用データを除けば)先頭に確保され、もう一方のデータ領域はページの末尾に確保されます。zbudは1つのページを64バイトごとのchunkに分割して、これを単位にして割り当てを行っています。

zbudは内部に2つのデータ領域が割り当てられているページのリストbudiedと、空きchunkがnであるページのリストの配列unbuddied[n]とを持っています。新しくデータ領域を割り当てるときは、まずunbuddiedを見て、どこにも空きがなければ新しいページを確保します。つまり、zbudから2,049バイトの領域を確保しようとするとunbuddied[33]のリスト、unbuddied[34]のリスト……という順に新しいデータ領域のためのページがないかどうかをチェックしていく、というわけです。

zbudではページのlruリスト(least recently usedリスト)も管理していますが、これについては次の項で触れたいと思います。

▼図3　メモリアロケータzbud



メモリの回収

ここまでで、swapされるページとpage cacheがfrontswap、cleancache APIを通じてzcache、zswapに渡され、圧縮されたデータがzbudによって確保された領域に書き込まれる様子を見てきました。一見これですべてのようですが、もう1つ大事な処理が欠けています。それはメモリの回収処理です。メモリのサイズも有限ですから、無尽蔵に圧縮されたメモリを貯めこんでおくわけにもいきません。とくにzswapの場合はswapされそうだったページを圧縮して保管しておいたわけですから、圧縮データをメモリ上から削除するときには、しっかりと元のスワップデバイスに書いておかねばなりません。zswapでのメモリ回収がどのように行われるのかを見てみましょう。

frontswapからzswapにページが渡されてきたタイミングでzswapは、そのときのzswapによるメモリ使用量のパーセントをチェックします。これがしきい値(デフォルトで20%)を超えるとzbudのzbud_reclaim_page()が呼び出されます。これが呼び出されるとzbudはlruを使って、最も長く使われていなかったページを選び出してそこに保持されているデータを解放しようとします。このときにzbudの使用側、つまりここではzswapが登録している“struct zbud_ops”のevict関数を、そのページに保管されているデータのハンドルを引数として呼び出します。このevict関数の中でzswapはデータを取得し、スワップデバイスへの書き込みを行い、書き込みが成功すればデータをzbudから削除します。1つのページに割り当てられている2つのデータ領域の両方が削除されればzbudはそのページを解放します。

これでzswap、zcacheのしくみがすべて読みとけました。ここまでの関係をまとめると**図4**のようになっています。うまくfrontswap、cleancacheとzswap、zcacheとzbudとが分業しています。結局zswap、zcacheは、frontswap、cleancacheのハンドルとzbudのハンドルとを

マップする役割をしているわけです。

メモリアロケータ zsmalloc

さて、zbudの解説を読んでシンプル過ぎると思われた方も多いのではないでしょうか。実際そのシンプルさを実現する「1つのページに2つまで」の制限ゆえにデータの密度(1つのページにいくつのデータ領域を確保できるか)は、たかだか2にとどまってしまいます。すでに軽く触れていましたが、実はzbudの前にzsmallocというメモリアロケータがありました。こちらはより高いデータの密度を実現できるのですが、その複雑さ、メモリ回収時の煩雑さゆえに今のところマージが見送られています。ここではzsmallocがどのような特徴を持ったメモリアロケータなのかを簡単に紹介します。

zsmallocもslabと同様に確保された領域を同じサイズに区切って、メモリの割り当てを行うというアプローチをとっています。kmallocから使われるslabと異なっているところは、kmallocのslabのサイズが2のべき乗であるのに対して、zsmallocではより細かく32バイトから16バイト刻みで32バイトのクラス、48バイトのクラス、64バイトのクラス……というように確保します。こうすることでたとえば、kmallocでは2,049バイト確保するのに4,096バイトの領域が割り当てられてしまっていたのが、zsmallocでは2,080バイトの領域が割り当てられるので無駄になるメモリ量が劇的に少なくなります。

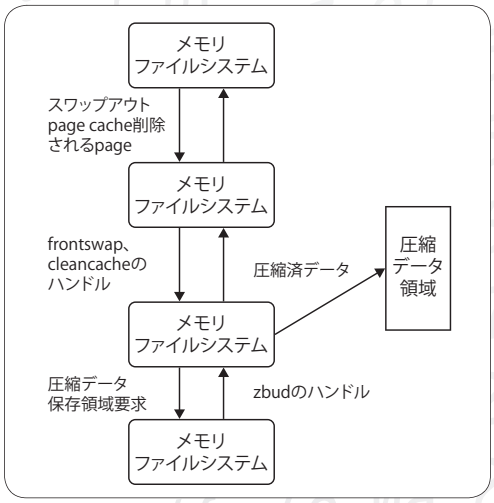
もう1つの大きな特徴は物理的に連続していない2つのページをまたいだメモリ割り当てを行うことができる点です。slabではsize-8192のような複数のページを必要とする割り当てを行うときには物理的に連続したページが割り当てられます。このzsmallocの特徴から、メモリが少ない状況でも連続したページを探しだす必要がないわけですので、メモリの割り当てが成功しやすくなります。一方でこの特徴がzsmallocから返されるものがそのまま使えるアドレスではなく、ハンドルになるという制限も生んでいます。zsmallocを使うときには、このハンドルを引数にしてzs_map_object()関数を呼び出し、適切にページをマップしてもらって返ってくるアドレスを使う手間が必要となります。

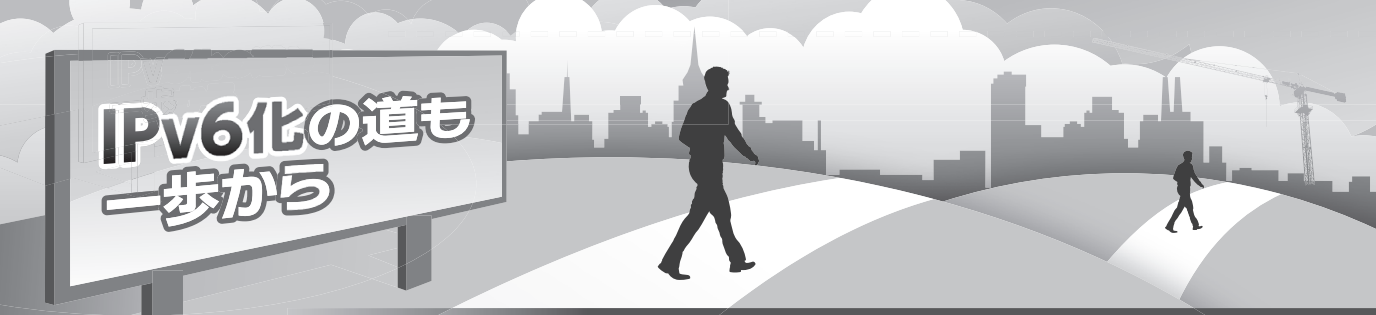
また問題視された特徴としてメモリ回収の難しさがあります。zbudの場合には1つのページに、たかだか2つの圧縮データしかありませんでしたから、その2つさえうまくスワップデバイスに書きだしてしまえばそのページを解放できました。しかし、zsmallocの場合は最悪の場合1つのページを解放するのに128ページ分のデータをスワップデバイスに書きだすことになります。こういった挙動の予測の困難さもマージの障害になっているようです。

まとめ

今月はメモリ圧縮機能であるzcacheとzswapについて紹介しました。まだzswapしかマージはされていませんが、zcacheのマージも早く行われるとうれしいところです。また、zswapもzcacheもzsmallocからzbudに移った経緯からかメモリアロケータの変更に対応しやすいように書かれているので、将来的にはzsmallocもマージされていくかもしれません。SD

▼図4　関係のまとめ





第9回 ネットワーク構築時の注意点と落とし穴（セキュリティ編）

IPv6普及・高度化推進協議会　IPv4/IPv6共存WG　アプリケーションのIPv6対応検討SWG
廣海 緑里 HIROMI Ruri　渡辺 露文 WATANABE Tsuyufumi　新 善文 ATARASHI Yoshifumi　藤崎 智宏 FUJISAKI Tomohiro

IPv6のアドレス選択

前回は、アプリケーションのIPv6対応について説明しました。今回は、IPv6のセキュリティについて説明します。

◆ ◆ ◆

セキュリティの話をする前に、セキュリティにも関係するIPv6アドレス選択について説明します。IPv6では1つのインターフェースに複数のアドレスを割り当てて利用します。IPv4との共存環境では、IPv4アドレスも加味してアドレスの選択を行います。アドレス選択には、「宛先アドレス選択」と「送信元アドレス選択」の2種類があり、「アドレス選択ルール」に基づいてIPv6で通信するか、IPv4で通信するかが決まります。

このアドレス選択のルールはRFC3484で規定されており、2012年9月にRFC6724で改定されました。IPv6とIPv4の共存環境ではIPv6の通信が優先されますが、これもこのルールに則ったものです。

厳密には、送信元アドレス選択で8個、宛先アドレス選択で10個のルールがあります。ここではポイントとなるものを抜粋して説明します。

送信元アドレス選択ルール（抜粋）

- ルール②：適切なスコープを選ぶ
- ルール⑥：ラベルが一致するものを選ぶ
- ルール⑧：共通するプレフィックスが長いものを選ぶ（Longest match prefix）

宛先アドレス選択ルール（抜粋）

- ルール①：使用できない宛先は避ける
- ルール②：スコープが一致するものを選ぶ
- ルール⑤：ラベルが一致するものを選ぶ
- ルール⑥：優先度が高いものを選ぶ
- ルール⑧：より小さいスコープのものを選ぶ
- ルール⑨：共通するプレフィックスが長いものを選ぶ（Longest match prefix）

通信を行う際、まず正引き名前解決のDNS問い合わせを行い、結果（IPアドレス）をリスト（宛先アドレスリスト）で取得します。続いて宛先アドレス選択ルールに基づき、宛先アドレスが決定されます。

上記の宛先アドレス選択ルールの②と⑧にある「スコープ」とは、アドレスに応じた通信可能範囲を指します。たとえば、リンクローカルアドレスのスコープは同一リンク内ですし、グローバルユニキャストアドレスのスコープはグローバル（インターネット全体）です。宛先アドレス選択ルール⑤にある「ラベル」とルール⑥にある「優先度」は、同じRFCで規定される「ポリシーテーブル」で管理されています。RFC6724におけるポリシーテーブルの初期値は**表1**のとおりです。このテーブルにおいて、デフォルト値ではIPv6の優先度がIPv4（マップドアドレス）の優先度より高いため、IPv6がIPv4より優先となります。

このポリシーテーブルはカスタマイズが可能であり、IPv6のグローバルな通信路がなくIPv4

へのフォールバックに時間がかかる場合は、カスタマイズして端末側で対処することもできます。

宛先アドレスが決まったあと、送信元アドレスを送信元アドレス選択ルールに基づき決定します。

IPv6とセキュリティ

少し古い資料を読むと、「IPv6はIPv4よりセキュリティが強化された」という記述を目にすることがあります。これは、IPsecの仕様が標準化される前にすでに広く利用されていたIPv4とは違い、IPv6ではIPsecを標準搭載するとされていたことに起因します。しかし、実際はIPsecが使用されないケースもあるため、必ずしもIPv4よりセキュアであるとは言えない状況です。逆に、IPv6だからセキュリティレベルが下がるということもありません。

RFC4301にて、IPsecに準拠するシステムの基本的なアーキテクチャについて定義されています。IPsecはIPv4およびIPv6で利用可能なIP層での暗号化通信を提供します。そして、通信はIPv4同士、IPv6同士である必要があります。IPv4とIPv6とでIPヘッダ構成やパスPTU探索の処理が異なるため注意が必要です。

組織としてのIPv6対応を考えるうえでは、個別のプロトコルのことよりも、むしろ、IPv4とIPv6の共存により注意すべき点が増えることを認識しなければなりません。

「IPv6は当面運用しない」と言い切る人であっても、注意が必要です。というのも、最近のクライアントOSはデフォルトでIPv6のアドレス自動設定が有効になっていて、知らぬ間にIPv6を利用できる環境になっていることがあるからです。管理監督していないために、IPv6を悪用した通信が行われる可能性があります。

2012年10月に公開された「IPv6技術検証協議会セキュリティ評価・対策検証部会最終報告書」[注2]にIPv6環境における脅威と対策がまとめられています。個々の脅威と対策についてはこの報告書を参照ください。そのほかの参考文献はコラム「IPv6のセキュリティに関する参考文献」を参照ください。

▼表1　ポリシーテーブル初期値

プレフィックス	優先度	ラベル
::1/128	50	0
::/0	40	1
::ffff:0:0/96 注1	35	4
2002::/16	30	2
2001::/32	5	5
fc00::/7	3	13
::/96	1	3
fec0::/10	1	11
3ffe::/16	1	12

注1）::ffff:0:0/96がIPv4を表すマップドアドレスです。

IPv6における脅威の種類

IPv6に関する脅威の種類としては、通信の妨害、盗聴、改ざん、情報漏洩などがあります。ここからは、IPv6に関する脅威として、通信の妨害と盗聴を引き起こす例を取り上げて解説します。

IPv4同様に同一リンク内の脅威への対策がなされていない場合、非正規のノードの通信が可能になります。万が一、悪意を持った非正規のノードがネットワークに参加すると、過剰なアクセスや、一連の通信を途中で止めることにより、ネットワーク機器のリソースを過剰に消費させ、サービスを妨害することが可能になります。また、悪意を持った人が、アドレス自動設定の機構を（正しく）利用し、通信を誘導することで盗聴するといった脅威もあります。前者は導入するネットワーク機器で対応されるべきものですが、後者はネットワーク構築で対策する必要があります。組織のIPv6対応としては、サービス妨害の対策が行われた機器を導入し、盗聴されないしくみを構築することが重要となります。

注2）http://ipv6tvc.jp/documents/20121023Report.pdf

組織内のネットワークについては、物理的にそこに立ち入れる人物が限定されるため、外部の人物によるリスクは低いです。しかし、情報流出の約半数が内部犯行と言われる昨今、IPv6を用いた盗聴対策は、IPv4のみのネットワーク運用者にも必要だと考えます。また、最近利用が増えている公衆無線LANやイベントネットワークなど、不特定多数の人が接続する環境での考慮も必要となります。

ここでは、IPv6の基本技術を用いた脅威である不正RAについて説明します。

不正RA問題

故意もしくは過失にかかわらず、不正にルータを設置し、そのルータのRA(ルータ広告)が流れることで、アドレス自動設定機構[注3]を利用して外部アクセスの経路が操作される問題を「不正RA問題」と言います。では、この原理を説明しましょう。まずは本連載5回目[注4]で紹介したアドレス自動設定のおさらいです(**図1**)。

ネットワークに接続された端末は、自身のインターフェースのインターフェースIDを生成し、fe80::/64のプレフィックスをつなげてリンクローカルアドレスを生成し、ネットワーク上に重複するアドレスがないかを確認します(**図1-①**)。重複アドレス確認(DAD)の応答がなければリンクローカルアドレスが確定します。続いて、ルータを探すためRS(ルータ要請)を同一リンク内に問い合わせます(**図1-②**)。この問い合わせに対し、ルータがRA(ルータ要請)を送信し、デフォルトルートやプレフィックスの情報を送信します(**図1-③**)。RAを受信した端末は、RAで通知されたプレフィックスと自身のインターフェースIDからアドレスを生成し、重複ア

注3) 最近のOSではデフォルトで有効になっています。

注4) 本誌2013年4月号。

COLUMN IPv6のセキュリティに関する参考文献

世界的にIPv6が利用、展開されつつあり、セキュリティに関する調査報告の公開や問題指摘などが盛んに行われるようになってきました。本文中に挙げた、IPv6技術検証協議会の報告書のほかにも、この2、3年に公開された文献をいくつか挙げておきます。

- **IPv6普及・高度化推進協議会**[注5]
  セキュリティWG「IPv6対応セキュリティガイドライン(第1.0版)」
  IPv4/IPv6共存WG IPv6導入に起因する問題検討SWG「IPv6導入時に注意すべき課題(2011年9月30日発行)」
- **日本セキュリティオペレーション事業者協議会**[注6]
  「IPv6検証報告書」
- **情報処理推進機構セキュリティセンター/IPA**[注7]
  「TCP/IPに係る既知の脆弱性に関する調査報告書 改訂版第5版」
- **ITU/IETF**
  「Draft Recommendation ITU-T X.1037(X.ipv6-secguide)Technical security guideline on deploying IPv6」
- **米国商務省/NIST**
  「Guidelines for the Secure Deployment of IPv6」

セキュリティに関しては、各組織のネットワークに関するポリシーが先行してあるべきであり、それに沿った形で実現されるものです。それぞれの文献に提示されている対策は、やや利便性を犠牲にしてでも、脅威に対して堅牢であるという考えに傾いたものが多く、自組織に必要な対策と予算に合わせて検討してみてください。

注5) http://www.v6pc.jp/jp/archive/index.phtml
注6) http://isog-j.org/activities/result.html
注7) http://www.ipa.go.jp/security/vuln/vuln_TCPIP.html

ドレス確認を行い（**図1-④**）、応答がなければIPアドレスが確定します。

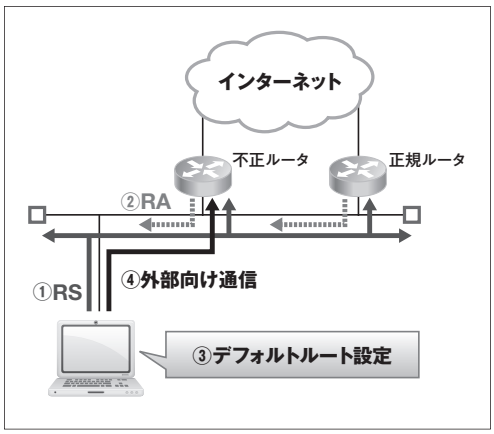
不正RAの流れの例を**図2**にまとめました。不正RA問題では、本来RAを送信する正規のルータとは別に不正なルータが設置され、端末から送信されるRS（**図2-①**）に対するRA（**図2-②**）を送信します。端末は不正なルータから送信されたRAに基づき、プレフィックスとデフォルトルートを設定します（**図2-③**）。以後、端末からリンク外への通信は不正なルータを経由することになります（**図2-④**）。不正なルータの先の構成によっては、情報を盗聴される危険性もある重要な問題です。

じつはこれと同様の問題は、IPv4のDHCPでも起こり得ます。みなさんもある日、突然外部と通信できなくなったという経験がないでしょうか？　そして、同じフロアの誰かが勝手にDHCPサーバを建てていたことが原因だった。そんな経験ありませんか？　この勝手DHCPサーバと原理は同じです。勝手DHCPサーバも外部と通信できれば、利用者が気づかないうちに、途中で通信を盗聴することも可能です。IPv6に限った話ではありません。

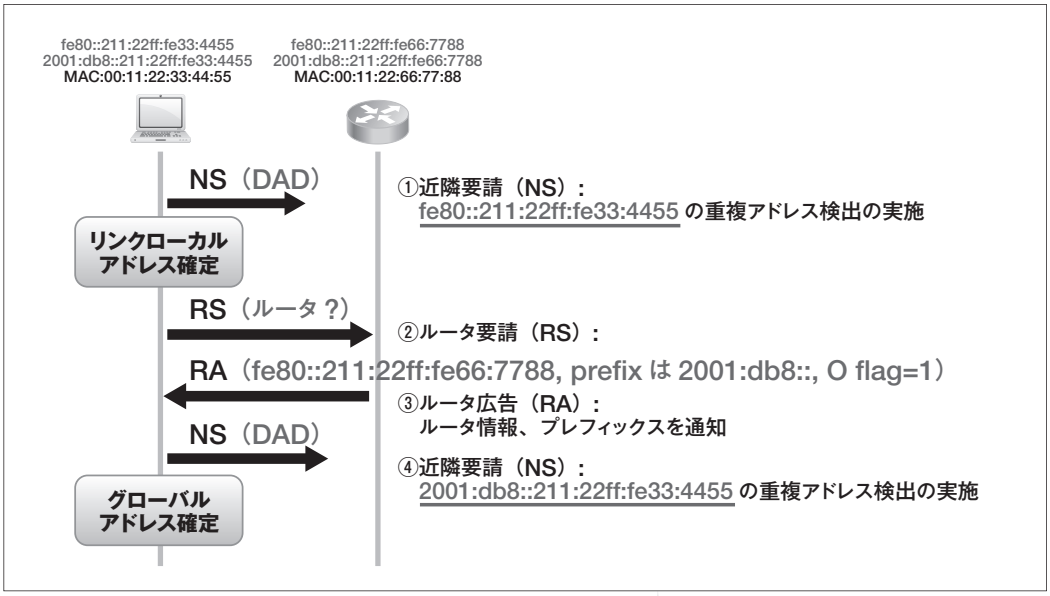
不正RAで厄介なのは、IPv6が有効な環境においてはIPv6がIPv4より優先されるため、IPv6の運用をしていないネットワークでも発生し得るということです。IPv4しか運用していないネットワークでは気づきづらいものです。

不正RAの対策としては、RA Guardや、SEND（SEcure Neighbor Discovery）、NDPのモニタリング、RAのフィルタリング、IEEE 802.1x認証などが挙げられます。また、IPv6アドレス自動設定を無効にしたり、IPv6対応しないネットワークでは先述のポリシーテーブルに

▼図2　不正RA



▼図1　SLAACによる端末のIPv6アドレス自動設定



てIPv6通信の優先度を下げるという対策も有効です。PC個々に設定する対策については、運用開始後も抜け漏れが発生しないように取り組む必要があります。

フィルタリングに要注意

IPv6においても、各サービス(ポート)へのアクセスは、フィルタリングで制御します。このフィルタリングについては注意が必要です。

まず、IPv4とIPv6とで、フィルタの設定方法が別になっているケースがあります。たとえば、Linuxのパケットフィルタがこれにあたります。IPv4のフィルタリングはiptablesというツールで行い、IPv6のフィルタリングはip6tablesというツールで行います。なお、iptablesとip6tablesは、それぞれ個別に設定を行う必要があります。IPv4とIPv6とでポリシーが同じとなるよう設計し、設定で抜け漏れが生じないように気をつけましょう。セキュリティインシデントの情報管理や対策を提示しているCERTが設定例を公開しており、参考になります[注8]。

続いては、フィルタリングにおけるIPv6に関する注意点です。IPv6では、同一のインターフェースが複数アドレスを有し、通信相手に応じて適切な送信元アドレスを選択して通信を行います。同一ホストの同一インターフェースでも、接続先によって送信元アドレスが異なるため、フィルタリングを行う際には適切な送信元アドレスでルールを記述する必要があります。デフォルトのアドレスポリシーテーブルを利用していて、グローバルアドレスとユニークローカルアドレスが付与されたノードでは、通信開始時に、先述したアドレス選択ルールに従い通信に用いるアドレスを決定します。たとえば、接続先がグローバルアドレスの場合には送信元もグローバルアドレスが選択され、宛先が同一サイト内のユニークローカルアドレスの場合には送信元もユニークローカルアドレスが選択さ

注8) ip6tables_rules.txt(www.cert.org/downloads/IPv6/ip6tables_rules.txt?)

COLUMN IPv6無効化の是非

最近のOSでは、IPv6アドレス自動設定がデフォルトで有効になっています。これに対し、IPv6が運用されていない環境では、「IPv6を無効にしたい」という要望が多いようです。LinuxでIPv6を無効にする方法を書いたブログエントリもいくつか散見されます。

GoogleやFacebookなどでサービス提供され、知らない間に使っていることもあるため、無効にすべきでないという時代になってきました。しかし、IPv6を使用しない環境では、IPv6は無効にするべきなのでしょうか？　OSを提供する各社／各団体の見解を見てみましょう。

まず、Microsoft社ですが、「Microsoft WindowsのIPv6：よく寄せられる質問」[注9]に、「IPv6を有効のままにしておくことをお勧めします」と明記されています。それどころか「Microsoftの見解では、IPv6はWindowsオペレーティングシステムの必須部分です」とまで明記されています。

続いて、CentOSです。FAQの中で、IPv6を無効化しないことを推奨しています[注10]。

最後にApple社のMac OS Xですが、サポートページにてIPv6を無効にする設定を記載しています[注11]が、その是非については言及されていません。

各社／各団体とも表現に違いがあります。みなさんもご自分がお使いのOSがどうとらえているかを調べてみてはいかがでしょうか？

注9) http://www.microsoft.com/ja-jp/mic/interop/ipv6/faq.aspx

注10) http://wiki.centos.org/FAQ/CentOS6#head-d47139912868bcb9d754441ecb6a8a10d41781df

注11) http://support.apple.com/kb/HT4667?viewlocale=ja_JP&locale=ja_JP

れます。

IPv6で利用されるICMPv6は、IPv4で利用されるICMPから変更点があり、セグメントを越える通信においても、Type1～4で規定されるエラーメッセージは、通信を許可しないと、通信に不具合を及ぼす可能性があります。**表2**にセグメントを越える通信でも許可すべきICMPv6を示します。

内部ネットワークにプライベートアドレスを利用し、組織の出口でアドレス変換を実施することが主流だったIPv4では、フィルタの設定が甘くても、外部からホストに直接通信することが困難でした。IPv6では、インターネットに接続し通信する際に、ゲートウェイでアドレス変換するのではなく、ホストにグローバルアドレスを割り当てることが主流です。そのため、DMZなどのIPv4でグローバルアドレスを利用しているセグメントと同じように、外部からホストに直接通信することが可能であり、フィルタの設定が甘いと不慮の侵入を許すこととなってしまいます。心して設定してください。

グローバルアドレスのプライバシー保護

IPv6では、インターネットと通信を行う端末にはグローバルアドレスを割り当てることが主流です。外部にサービスを公開するサーバについては、グローバルアドレスが外部に知られても何ら問題ないと思います。しかし、クライアント端末では、プライバシー保護の観点からグローバルアドレスが固定され、外部に知られることが嫌がられるケースがあるかと思います。また、アドレス自動設定においてインターフェースIDをMACアドレスから生成するEUI-64方式では、MACアドレスからインターフェースIDが一意に決定するため、逆にインターフェースIDからMACアドレスを一意に特定できます。MACアドレスが外部から特定されてしまうことは、機器ベンダの特定につながる可能性があり、セキュリティ上好ましくありません。

これを解決するのがPrivacy Extension for Stateless Address Autoconfiguration in IPv6（RFC4941）です。ランダムな初期値のMD5ハッシュからインターフェースIDを生成します。生成されたアドレスの推奨有効期限は1日、最大有効期限は1週間となっています。アドレスのプライバシーを気にすべき環境では、クライアント端末にはPrivacy Extensionを有効にしたアドレス自動設定を行うようにし、環境によって使い分けましょう。

終わりに

今回は、IPv6とセキュリティについて取り上げました。いかがだったでしょうか？　本連載は、次回で最終回です。これまで伝えきれなかったことを綴ります。お楽しみに。SD

▼表2　通過させるべきICMPv6パケット

ICMPv6 Type	パケットの意味	遮断した場合の影響
1：Destination Unreachable	宛先に到達できない。これを受信した送信元ホストは、IPv6での通信を断念しフォールバックする	遮断されるとフォールバックが遅くなる
2：Packet Too Big	送信パケットのMTU値が経路の上限を超えた場合に経路上のルータからパケット送信元へ送信される。これを受信した送信元ホストはMTU値を下げて再送する	遮断されると通信不安定、到達不可になる
3：Time Exceed	宛先に到達できずホップリミットに到達した	遮断されるとフォールバックが遅くなる
4：Parameter Problem	IPv6ヘッダの異常を通知する	遮断されるとフォールバックが遅くなる

October 2013

NO.24

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

これまでの30年、これからの30年?

jusは毎年7月に定期総会と併設行事を行っています。今年は7月20日（土）にハロー貸会議室四谷にて実施しました。今回はその模様を報告します。

定期総会

はじめに定期総会を行いました。議題は2012年度の活動報告と会計報告、および2013年度の役員選出と活動計画と予算案で、いずれも承認されました。jusは慣例で2年ごとに役員、とくに会長を交代することになっており、今年度は会長が代わる年にあたるため、新会長を選出しました。今年度の役員は次の方々です。

- 会長：岡松伸太郎（ティ・エイチ・アイ）
- 副会長：波田野裕一（運用設計ラボ）、片山喜章（名古屋工業大学）
- 事務局長：井上尚司（武蔵野美術大学）
- 理事：高野光弘（グリー）、松澤太郎（Georepublic Japan）、榎真治
- 監事：砂原秀樹（慶應義塾大学）

併設イベント「日本UNIXユーザ会のこれまでとこれから」

併設行事として、jus設立から30周年を迎えることを記念してトークイベントを行いました。第1部はこれまでのjusを支えてきた面々が、そして第2部はこれからのjusを担っていく人材が登場し、それぞれの想いを語り合いました。参加者は35人でした。

■第1部：これまでの30年を振り返る

登壇者：井上尚司、小山哲志、砂原秀樹、法林浩之

第1部は歴代会長や事務局長が登壇者となり、30年を振り返りました。登壇者のうち井上さんが事務局長、ほかの3人が元会長です。まず各々が自己紹介やjusとのかかわりを話したあと、司会を務めた筆者からjusの活動に関連するキーワードを次々に提示し、登壇者が思い出を語りました。

黎明期である1980年代の主な活動としてはUNIXシンポジウムとUNIX Fairがありました。シンポジウムを大阪で開催するときは幹事達が新幹線で宴会しながら移動したこと、シンポジウムの併設展示会が大規模化してUNIX Fairとなり、UNIX Fairも巨大化したのでイベント業者に委託したことなどの話がありました。また、UNIX Fairで行ったX11やTCP/IPの相互接続実験からインターネット関連の話題に転換し、JUNETやWIDEとの関わり、東京のシンポジウムがインターネットコンファレンスになったこと、Internet Weekにおけるjus提供企画の数々が紹介されました。

1990年代にはPC-UNIX関連のイベントや各種ワークショップを実施しました。勉強会もこの時期にスタートしたのですが、まだコミュニティ主催の勉強会が珍しかった時期で、運営においてさまざまな工夫を行いました。そして、これらの動きがオープンソースまつりにつながり、現在は関西オープンフォーラム（KOF）に受け継がれています。また、2000年以降の主な活動としてLightweightLanguage（LL）イベントと、その前身となったYARPC 19101も併せて紹介されました。各イベント誕生の経緯に

加えて、YARPCで日本初のライトニングトークが行われたこと、LLイベントを契機にチケット販売をチケット会社に委託するのが定着したことなどの話がありました。

このほかに、地域密着の活動としてjus関西UNIX研究会がありました。jus幹事の齊藤明紀さんが10年以上に渡る連続発表記録を作ったり、探偵ナイトスクープ風の構成をとったUNIX探偵団などが好評でした。2007年からは研究会の全国ツアーも行われています。さらに、テープやCD-ROMによるフリーソフトウェア配布、黎明期に行われていた分科会活動、事務局の変遷などの話がありました。最後に登壇者の感想として、「UNIXが30年も生き残っていて、しかも今でも多くのアーキテクチャに採用されているのはすごい」、「インターネット以前の話はWebに残っていないものが多いので、このようなイベントを開催し、当時の状況を懐述することで記録が残るのは貴重である」などのコメントがありました。

■第2部：これからの30年(?)を考える

登壇者：岩井雅治、榎真治、岡松伸太郎、
高野光弘、松澤太郎

休憩をはさんで後半は、幹事の中で年齢の若い方から5人が登場し、今後のjusについて考えるひとときを持ちました。

はじめに各自が自己紹介やUNIXとの出会い、jusに参加したきっかけを話しました。ほぼ全員が学生のころからUNIXを知っていて、最初からPC-UNIXを触っていた人が多いところに世代の違いが感じられました。また、jus以外にほかのソフトウェアコミュニティでも活動しているのも特徴でしょう。

その後、本題として、現在のjusについて所感を語り合いました。まったくのフリートークだったので、話題は多岐に渡ったのですが、その中からいくつかを紹介します。

- 現在はコミュニティ主催のイベントが増えたのでjusが勉強会をする必然性が薄れている。しかし、jusの行事は特定テーマに縛られないので横断的に勉強できるのが良い
- シェル関係の話題は意外に需要がある。たとえば美大生でも1割ぐらいの人は興味を持つ。近年はデザイン系の人がITに進んだり、逆に情報系の人がデザイン系に進んだりする傾向があるので、そういった層を対象にすると良いのではないか
- ニッチな話題やマニアックなテーマをもっと取り上げれば良い。2000年代はおもしろそうだと思ったら何でもやってみていた。たとえばT-code合宿など
- 一方的な講義になるセミナーではなく、参加者同士で議論できるワークショップ形式のイベントをもっとやると良い。Happy Hacking KeyboardやPOBoxもjusのワークショップに持ってきてデモして見せたところから製品化につながった
- おもしろいことをやっている団体であることを維持しつつ、その存在をもっとアピールしたい。最近はjusの存在が知られていない。オープンソースカンファレンスに出展したらどうか？
- 世代を超えて、技術分野を超えて、学術や産業といった境目も超えてメタコミュニティ的なアプローチができるのはjusの強みである。若い人ばかりの団体ではベテランからのコメントはもらえない

こんな話を繰り広げるうちにあっという間に時間がなくなり、最後に登壇者が一言ずつ今後の抱負を述べました。「マニアックな話を提供したい」、「関西での活動を強化したい」、「昔の物を残す作業を推し進めたい」、そして「UNIXはまだ生き残ると思うので文化や考え方も含めて伝えていきたい」といったコメントでイベントを締めくくりました。

◆ ◆ ◆

併設イベント終了後は、近くの店で30周年記念パーティーを行いました。こちらにも31人の参加がありました。古くからjusに関わっている人から最近jusと付き合いのできた人まで幅広い世代が集まり、思い出話や今後のことまでさまざまな話題を、酒を酌み交わしながら語り合いました。記念すべき1日にふさわしいイベントを実施できてよかったです。SD

Hack For Japan
エンジニアだからこそできる復興への一歩

●Hack For Japanスタッフ
及川 卓也　OIKAWA Takuya
Twitter @takoratta

第22回 Spending Data Party 2013と「税金はどこへ行った?」の広がり【その1】

社会的課題をテクノロジで解決するためのコミュニティ、Hack For Japanの活動をレポートする本連載。今回は税金の使い道を可視化するための取り組みについて紹介します。

自治体のオープンデータへの取り組み

Hack For Japanは昨年より、活動を震災復興支援にとどまらず、ITを通じて社会的な課題を解決することを目指しており、その一環として昨今力を入れているのが、オープンデータの活用です。政府や公共機関の持つデータを公開し、技術者が自由に活用することで、より開かれた政府を実現し、市民の政治参加も活性化されます。日本でも震災以降オープンデータをより積極的に活用しようという機運が高まっています。

このSoftware Designでの連載でも何度か紹介させていただいているように、税金の使い道を可視化する「税金はどこへ行った?(Where Does My Money Go?)」というサービスが日本で立ち上がっています。昨年、「オープンデータ活用ハッカソン」というイベントで横浜市版が作成されたのを皮切りに、有志によりいくつもの地方自治体向けサイトが立ち上がりました。このムーブメントをさらに加速しようと7月20日と7月21日に「Spending Data Party 2013」が開かれましたので、その模様を今回と次回の2回に分けてご紹介します。

Spending Data Partyでの講演

Spending Data Party 2013は「税金はどこへ行った?」の元となるOpenspendingプロジェクトをホストする、Open Knowledge FoundationのAnders Pedersenの呼びかけで、世界同時開催されたイベントです。

日本では、7月20日の午後と7月21日の1日半のスケジュールで開催されました。会場となったヤフー株式会社に、約50名の技術者やデザイナー、自治体担当者、地方財政に興味のある人たちなどが集まったほか、オンライン(Google+ ハングアウト)経由でも数名が参加しました。

7月20日は、主催者である関治之氏(Georepublic代表社員・Hack For Japanスタッフでもある)による挨拶の後、5人の方からの講演が行われました。

WDMMG の到達点と将来像

「WDMMG(Where Does My Money Go?の略)の到達点と将来像」というタイトルでお話をしてくださったのは、公共イノベーション代表取締役および今回のSpending.jpプロジェクト(「税金はどこへ行った?」はさまざまな呼ばれ方をしますが、本家のOpenSpendingを由来にしたSpending.jpという言い方もされます)のコーディネーターの川島宏一氏です。川島さんは昨年の「オープンデータ活用ハッカソン」のときに、日本版の「税金はどこへ行った?」の開発を呼びかけた、まさに日本版の生みの親の1人と呼べる方です。

川島さんはまず「ワニの顎」とたとえられる日本の国家としての財政状況を説明します。「ワニの顎」とは歳入が減っていくの反して、歳出が増えていく状況をグラフ化した際に、まるでワニの顎のように見える状況を揶揄した言葉です。このような状況では、国民1人1人が当事者意識を持ち、公共予算を意識することが必要となるのですが、川島さんは国際会議で知った「税金はどこへ行った?」がそのため

に有効と考え、昨年に日本版の開発を呼びかけました。

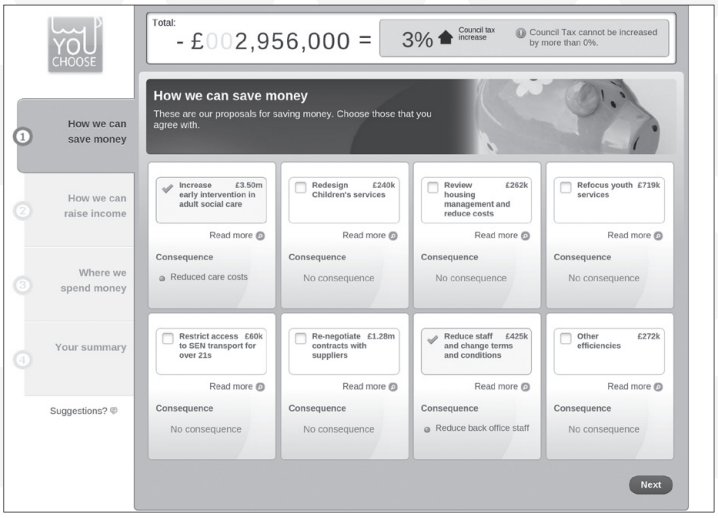
オープンデータは情報の透明性を実現し、イノベーションを加速します。税金を可視化するこのサービスも世界の自治体との比較などを可能にし、さまざまなアイデアを生み出すことが期待されていると川島さんは言います。

川島さんからは、すでに、今回のイベントの前に22都市向けの「税金はどこへ行った?」が立ち上がっていることが報告されました。川島さん自身も、現在どなたが関わっているかわからないくらいの広がりを持っているそうで、本家Open Spendingプロジェクトとの事前の打ち合わせでも日本からの技術者がプロジェクトをリードする場面も出てきていることなどが明かされました。

今後への展望としては、登録された予算(自治体によっては決算)データを用いたさまざまな応用アプリケーションなども検討していくことになるそうです。例として挙げられていたのが、英国でのYouChoose[注1]というサービスです(**図1**)。これは来年の予算をどの分野に使っていきたいか、つまり自分なりに来年度予算を組むことができるものです。このようなアプリケーションを日本でも提供していきたいと考えているそうです。

川島さんの講演は次の言葉で閉められました。「個人としての幸せ、企業の利益、社会への貢献を同時に実現できる社会をつくろう。『税金はどこへ行った?』もその一環として捉えていきたい。」

◆図1　YouChoose



税金の5W1H ー税制と財政の関係からー

次に公認会計士・税理士の山内真理氏から税金のしくみが5W1Hに沿って解説されました。

国民の義務である納税ですが、いざその内容やしくみを聞かれるとわからないものです。山内さんはそれを次のように説明します。

Why?「なんで税金を納めるの?」:

行政サービス・社会インフラ整備など社会的に必要な機能を担っている国家や自治体は税無しには成り立ちません。社会的な機能の維持に必要な税収を十分に出来ないのなら財源を債券発行で賄うしかないのですが、それは将来世代の負担で現役世代が便益を享受していることに他なりません。そのため、納税は国民の義務となっています。

Who?「税金を負担するのは誰?」:

税を負担できる能力(担税力)に応じて平等に課税しようという応能負担と受ける恩恵に見合った分だけ課税しようという応益負担の2つにより徴税は行われています。個人だけではなく、法人に対しても行われます。

How?「どうやって負担する額を決める?」:

徴税は「所得」、「消費」、「資産」に対して行われます。

Where?「税金はどこに行くの?」:

国と自治体は必要な社会的な機能を分担しており、予算調達能力や経済力には差があるなど、

注1) http://youchoose.yougov.com/redbridge2012

自治体の状況は様々のため、資源の再分配が行われています。 国の税収の一部は地方交付税、地方譲与税などを通じて地方に流れていますし、都道府県と市町村の間でも調整が行われています。

What?「目的税って？　特別会計って何？」:

目的税とは税の徴収とその使い道を直接対応させるために、特定の目的で集める税のことです。特別会計とは特定の目的で使うために、目的ごとに会計を分けることです。体の大きなお財布にまとめて色々な用途に振り分けるより、特定の目的ごとのお財布に別々にプールして計画的に使うことが合理的な場合が多いという考えに基いて、このようにされています。

（以上、山内さんの発表資料「税金の5W1H—税制と財政の関係から—」より）

山内さんから「税金はどこへ行った？」に対して、「課題と限界を知りながら、モデルづくりを」と応援のメッセージが送られました。

「税金はどこへ行った？（島根県版）」とは

一般財団法人島根総合研究所 常務理事事務局の長山根純氏および島根県からオンラインで参加のメンバーから島根県版（**図2**）の特徴が発表されました。

島根県版[注2]は島根総合研究所という非営利団体が運営しています。見るとわかるように、ほかの自治体とはまったく異なる画面になっていますが、これは県税や国税、社会保険も組み込んだものになっており、さらには市町村同士の比較も可能となっています。

使っているデータは予算ではなく決算になっていますが、その理由はデータ取得が容易であったことと、予算消化という、悪い言い方をすると、合法的な税金の無駄遣いは決算重視への転換で改善する可能性があるのではと考えたためでした。

市町村同士を比較しやすいものにしたことにより、市町村間でどの比率が高いかを想像してもらえるようになっています。たとえば、健康福祉割合が最も多いことがわかったり、老人福祉費が児童福祉費よりも多いことなどの特徴が見て取れます。

◆**図2　税金はどこへ行った？（島根県版）**

今後はデータベース共通化や他の市町村との比較など、継続的な検証可能性を世界の人たちと一緒に検討していくべきだと思うとおっしゃっていました。

「税金はどこへ行った？（千葉市版）」と今後の展望について

千葉市情報経営部 業務改革推進課課長 松島隆一氏からは千葉市版の特徴と今後の展望について発表されました。

人口減少により税収や職員数の減少に直面する千葉市は市民主体のまちづくりを目指しており、その一環として予算をよりわかりやすく公開するための試みも行っているそうです。たとえば、「もし千葉市が給料収入500万円の家庭だったら[注3]」のような形で自治体予算を家計にたとえ、想像がしやすいよ

注2）http://tax.souken.or.jp/

注3）http://www.city.chiba.jp/zaiseikyoku/zaisei/zaisei/download/chibashinokakeibo_h25.pdf

うに解説しています。

そのような背景がある中、開発された千葉市版の「税金はどこへ行った？」は予算データではなく、決算データを用いています。それは、使われる「予定」である予算よりも、使った「結果」である決算のほうが、税金の使い方を明示する意味が強いと考えられたからです。そのうえで松島さんは、予算版と決算版の両方が用意されると良いのではないかと提案されました。ほかにも、多くの自治体が2階層までで税金の使い道を可視化しているのに対して、千葉市版は3階層目まで用意しています。

また、千葉市版は横浜市版についで日本で2番目に用意されたものですが、データの分類が異なっているため、横浜市と千葉市の間だけでも、完全には自治体間の比較ができないことを指摘され、日本国内で統一した分類が望まれていることを言われていました。

千葉市は5月末に市長選が行われましたが、そこで再選を果たした熊谷市長はマニフェストとして、「自分がどれだけ税金を納め、どれだけ公的サービスを受けているのかが一目でわかるサービスの実現で、税金の行方に対して信頼が持て、納得感のある市政を推進」と掲げる[注4]など、今回の「税金はどこへ行った？」プロジェクトに大きく期待しているそうです。

今後については、単身と扶養ありの2分類以外にも、収入や家族構成などの市税を計算する項目を増やしていきたいそうです。また、公開する自治体側の負担を減らすために、予算統計や決算統計のデータを活用することを模索していると話されていました。

予算編成過程の公開とその意義

鳥取県産業振興室長の森本浩之氏からは鳥取県における予算編成過程の変遷について発表されました。

森本氏は鳥取県を「トップの強い意志でペーパーレス化から情報公開へ進んだ」と紹介されました。実際に全国市民オンブズ連絡会議が実施した予算編成過程の透明度ランキングにおいて2年連続第1位を獲得しています。

この取り組みは片山前知事時代から始まった改革に基づくもので、予算編成のペーパーレス化や予算編成過程の公開などが今でも続けられています。たとえば、平成25年度の補正予算についても、一般事業調整要求・査定の概要が県のサイトですべて公開されています[注5]。

ほかにも「とっとりWebマップ[注6]」や「鳥取県借金時計[注7]」のような面白い取り組みが紹介されました。

どうすれば納得してもらえるのだろう

埼玉県宮代町総務政策課 改革推進室長の栗原聡氏からは「どうすれば納得してもらえるだろう」と題して、行財政運営の取り組みの中で、住民に納得してもらうために、どのように情報を出していくかの取り組みが紹介されました。

宮代町は埼玉の小さな町ですが、1996年にはWebサイトを早くも開設したそうです。その後、予算や決算などのデータ公開を進め、市民を含めた議論を行っているそうです。宮代町の予算と決算というページ[注8]でその一端を見ることができます。

さらに、栗原さんは、ふるさと納税という自分の居住地以外の市町村に寄付をした場合、一定額まで税額が控除されるというしくみを紹介し、気に入った事業に対して寄付で地域を応援できるしくみをさらに活用できないかと提案されていました。

◆ ◆ ◆

7月20日は、この後にグループに分かれ、21日にかけて作業を行いました。その模様は次回ご紹介します。SD

注4) http://www.kumagai-chiba.jp/manifesto/manifesto2013/summary

注5) http://db.pref.tottori.jp/yosan/25Yosan_Koukai.nsf/h1-03.htm

注6) http://www2.wagmap.jp/pref-tottori/top/

注7) http://www.pref.tottori.lg.jp/204182.htm

注8) https://www.town.miyashiro.saitama.jp/WWW/wwwpr.nsf/%E4%BA%88%E7%AE%97%E3%81%AE%E3%83%9A%E3%83%BC%E3%82%B8?OpenPage

温故知新
IT むかしばなし

第26回

デバッガ

たけおかしょうぞう　TAKEOKA Shouzou　take@takeoka.net

初期のデバッガ

プログラムを書くと、デバッグ作業は必須です。その作業の友がデバッガです。マイコン以前のミニコンや大型コンピュータは、スタティック動作をしていたものが多いので、CPUのクロックを本当に止めてCPUの内部(レジスタ)を読み書きするデバッガが多くありました。

デバッガというソフトウェアは、8bitマイコンのころから存在します。マイコンはLSIになってしまったので、その中のレジスタを、直接、ハードウェアだけで読むことはできません[注1]。

DMAによるデバッグ

マイコン草創期のCPU、インテル8080／モトローラMC6800時代に、ハードウェアパネルからトグルスイッチをパチパチ動かして入力していたシステムは、CPUにDMA(Direct Memory Access)要求を出し、DMAでメモリを読み書きしていました。しかし、CPU内部のレジスタを読み出すことはできていませんでした。

ザイログのZ80以降は、ダイナミックRAMの時代に移り、実用システムでは、クロックを止めたり、DMAを長時間かけてデバッグする、ということはありえなくなっています。

デバッガの動作

デバッガは、デバッグ対象に影響を与えてはいけません。極端に言えば、その存在を対象に知られてはいけません。メモリ上にデバッガがいるときと、いないときで対象の挙動が変わっては、真のデバッグにならないからです。

CP/M80に標準装備のデバッガ"DDT"(**図1**)は、起動されると自身をOSの直前に転送し、OSの位置(アプリケーションが使用可能なメモリの上限)を表すグローバルなポインタを、DDTの直前に書き換え、DDTがさもOSの一部のように見せかけ、その存在を対象プログラムから隠しました。

近代的なMMU(Memory Management Unit)付きCPUで、現代的なOSが動作していれば、アプリケーションはほかから絶縁された独自の空間で動作し、ハードウェアのトラップなどで、OSが持つアプリケーション・デバッグ機構を経由し、別空間で動作するデバッガに制御を移すことは簡単です。

しかし、8bit CPU時代は、CPUにMMUもなく、CPUにデバッグ専用の命令も用意されておらず、デバッガ作り(設計)は非常に楽しい作業でした。

デバッガの議論

- 対象プログラムからデバッガに制御をどう移すか
- デバッガに入るとき、対象プログラムのレジスタなどの全情報をどうセーブするか
- デバッガから、対象プログラムに復帰＆継続するとき、全レジスタなどをどう復帰するか

といった問題は、各CPUに強く依存し、その実現に向けた議論はいつも楽しいものです。と

注1) 1990年頃からのJTAG(Joint Test Action Group)規格で、ハードウェアだけで、CPUチップ内のレジスタが読めるようになりました。

くに、レジスタなどを復帰しながら制御を対象に戻すのは、テクニックの見せところです[注2]。

対象プログラムからデバッガに制御を移すのも議論の的です。シングルステップの実行(対象プログラム1命令を実行したら、デバッガに制御を戻す)や、ブレークポイント(ある番地に来たら制御をデバッガに移す)をしかけるにはどうするか、ということです[注3]。

デバッガによる命令書き換え

ハードウェアによるデバッグの補助機能がまったくないシステムでは、デバッグのために命令書き換えを行います。つまり、実行対象命令の直後の命令を、デバッガをサブルーチンコールする命令(現代だとソフトウェア割り込み命令)に書き換えるのです。

8080(Z80)、MC6800、x86系は可変長命令(1命令が1バイトから数バイト)ですから、デバッガは、対象命令が何バイトあるかを知っていて、それによって直後の命令を書き換えます。対象命令が条件分岐命令であった場合、真/偽の両方の分岐先の命令を書き換えます。命令書き換えは現在のGDB(GNUデバッガ)でも使用されています。

ハードウェアによるデバッグ

しかし、8bit時代は、自分自身のプログラムコードを書き換えるコードが多く存在したため、命令書き換えによるデバッガは限界があり、ハードウェアの力を借りるのが普通でした。

8bit時代、有名なNECのTK-80は、デバッグのためのハードウェアを割り込みで実現していました。TK-80のデバッグ支援ハードウェアは、割り込み許可線(INTE)が有効になってから、2命令の実行(M1を2回)を検知すると割り込み要求を発生するようになっていました。TK-80のモニタ・ソフトウェアは、割り込み禁止で走行します。モニタが対象プログラムに制御を移すときは、

0.EI(割り込み許可命令;INTE線を有効にする)
1.RET(PCを復帰し、スタックポインタを戻す;割り込み受付可能)
2.対象命令(割り込み受付可能)

という流れです。したがって、2命令の実行検知が適切なのです。

多くのハードウェアは類似のことを行っていました。現在は、高級CPUの多くにはシングルステップ機構などが組み込まれていますが、安価なCPUでは今でも上記に述べたようなデバッグ方法が使用されています。SD

参考:TK80回路図
http://www.oct.zaq.ne.jp/i-garage/hiroimono/tk80_04.png

▼図1 DDTの画面

```
端末
A>
A>
A>
A>
A>
A>
A>
A>
A>
A>
A>
A>dir
A: ED       COM : DUMP     COM : DDT      COM : PIP      COM
A: MOVCPM   COM : BYE      COM : SYSGEN   COM : STAT     COM
A: ASM      COM : XSUB     COM : LOAD     COM : SUBMIT   COM
A>ddt dump.com
DDT VERS 2.2
NEXT  PC
0280 0100
-l
  0100  LXI  H,0000
  0103  DAD  SP
  0104  SHLD 0215
  0107  LXI  SP,0257
  010A  CALL 01C1
  010D  CPI  FF
  010F  JNZ  011B
  0112  LXI  D,01F3
  0115  CALL 019C
  0118  JMP  0151
  011B  MVI  A,80
-ied.com
-t5
C0Z0M0E0I0 A=00 B=0000 D=0000 H=0000 S=0100 P=0100 LXI  H,0000
C0Z0M0E0I0 A=00 B=0000 D=0000 H=0000 S=0100 P=0103 DAD  SP
C0Z0M0E0I0 A=00 B=0000 D=0000 H=0100 S=0100 P=0104 SHLD 0215
C0Z0M0E0I0 A=00 B=0000 D=0000 H=0100 S=0100 P=0107 LXI  SP,0257
C0Z0M0E0I0 A=00 B=0000 D=0000 H=0100 S=0257 P=010A CALL 01C1*01C1
-
```

注2) 今どきは、CPUによっては、メーカーがそれを提示しているものもありますが。

注3) 何のしかけもなく実行開始した対象プログラムは、ハードウェアの割り込みでもかけない限り、止められません。

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。

Report

Treasure Data社&Crocos社イベントレポート「誰も語らないデータ分析の3つの現実」

8月9日、㈱リクルートテクノロジーズ（東京都千代田区）において、Treasure Data社（以下、TD）とCrocos社の共催で「データサイエンスの始め方・データ分析のデザインパターン」と題したイベントが行われた。その中で、TDの田村清人氏がデータ分析における課題について発表を行った。

田村氏は、データ分析には3つの課題があるという。

1つめは「データ収集の問題」。いざデータ分析を始めてみると、集めたデータに間違いがあり正しく集計、分析ができないことがよく起きるとのこと。ログ分析のことをあまり意識せずにアプリケーション開発担当者が勝手にログの内容を変更したり、データを集めるしくみが複雑過ぎたりすることが主な原因だ。

これを解決するには「統一されたシステムでデータを収集する」ことが大切だという。つまり、TDのようなサービスを使えば、複数のアプリやDBから成るシステムでも、統一的にデータを収集できるというわけだ。

2つめの課題は「データ分析時の問題」。データ分析の現場では、データ収集／加工に長けた人間と、データ分析に長けた人間とは同じでない場合が多い。両方できてこそ、データサイエンティストなのだろうが、田村氏自身、そういう人は実際には少ないと感じている。

現実的には、「どちらかの仕事をアウトソースする」ことで解決策を見いだすという。つまり、データ分析に特化したサービスを利用することや、TDのようなデータを貯める基盤サービスを利用することなどだ。

たとえば、ヨーロッパで最大のモバイル系広告プラットフォームを提供するMobFox社は、TDのサービスを使って、1ヵ月に4000億レコードのデータをさばけるプラットフォームを14日間で作り上げた。

田村氏は、「インフラ技術が会社のコアでない限り、内製するのは必ずしも正しいやり方ではない」と主張する。構築の費用だけでなく、時間も無駄になるからだ。TDを使えば、そんな手間を省いて、今すぐデータの分析が始められる。

▲Treasure Data社 田村清人氏

3つめの課題は「データ基盤の整備の問題」。通常、データ分析による成果は、ある程度データが蓄積してからでないと得られない。そのためデータ基盤の整備は後回しにされやすい。しかし、田村氏は「ないデータを分析することはできない」と、サービス開始時からデータを収集することの重要性を指摘する。

解決案として挙げたのは、FluentdなどのOSSを活用してデータ収集のコストを下げること。Fluentdは TDの古橋氏が開発したデータ収集のソフトウェアだ。

田村氏の発表において、データ分析ではさまざまな場面で課題に直面することが明らかにされた。すべて自社内で解決するには大きなコストがかかる。TDのサービスやFluentdなどがどのような場面で役立つのかなど、解決の手掛かりが垣間見られた発表だった。

CONTACT トレジャーデータ㈱
URL http://www.treasure-data.com

Software

パラレルス、デスクトップ仮想化ソフトウェアの最新版「Parallels Desktop 9 for Mac」を発売

パラレルス㈱は8月29日、MacでWindowsおよびMacの両方のアプリケーションをシームレスに実行できるソフトウェアの最新バージョン「Parallels Desktop 9 for Mac」を発売した。

既存ユーザ向けアップグレード版は、同社Webサイト（www.parallels.com/jp/desktop）にて同日より販売開始、新規ユーザ向けには小売店、オンラインストア、同社Webサイトなどで9月6日より販売開始している。パッケージ版の標準小売価格は7,900円（税込）。

最新バージョンは、旧バージョンと比較し、ディスクパフォーマンスが40%向上、仮想マシンの起動とシャットダウンが最大25%迅速化、仮想マシンのサスペンドまでの時間が最大20%短縮、3Dグラフィックおよびウェブブラウジングはともに最大15%スピードアップといったパフォーマンス向上を実現している。

また、Windows 8/8.1対応として、［スタート］メニューの搭載や、メトロアプリをフルスクリーンではなく単独のウィンドウとして開くといった改良が施されている。

CONTACT パラレルス㈱
URL http://www.parallels.com/jp

Software

グレープシティ、Windowsフォーム資産をタッチ対応アプリにリメイクする「MultiTouch for Windows Forms 1.0J」を発売

グレープシティ㈱は、既存のWindowsフォームアプリケーションをタッチ操作に適したアプリケーションにリメイクするコンポーネント「MultiTouch for Windows Forms 1.0J」を8月28日に発売した。

Visual Studioで既存のWindowsフォームプロジェクトを開き、ツールボックスからコンポーネントを追加してリビルドするだけでタッチ操作に対応させられる。クリックレベルの操作だけならVisual Studioだけでもタッチ対応にできたが、本製品を使うことで「ピンチ操作でフォームを拡大／縮小させる」、「拡大したアプリケーションをスワイプ操作でスクロールさせる」といった操作が可能になる。さらに、同製品には拡大鏡やパンウィンドウなどのコントロールも収録。XPや7などの既存環境では今までどおりに利用できるので改修コストをほとんどかけずにいつものWindowsフォームをタッチアプリケーションにリメイクできる。

1開発ライセンス価格は31,500円（税込）、開発したアプリケーションを配布する場合は別途ランタイムライセンスが必要。

CONTACT グレープシティ㈱
URL http://www.grapecity.com/tools

Software

キヤノンITソリューションズ、ESET法人向けライセンス製品が新たにデスクトップ仮想化製品を正式サポート

キヤノンITソリューションズ㈱は、法人向けライセンスの総合セキュリティ製品「ESET Endpoint Protection Advanced」とウィルススパイウェア対策製品「ESET Endpoint Protection Standard」において、デスクトップ仮想化製品の「VMware Horizon View 5.2」と「Citrix XenDesktop 5.5」の正式サポートを8月5日より開始した。同セキュリティ製品は、

- Windows/Mac/Linux/Androidなど多彩なOSに対応
- クライアント管理用プログラムを利用することでWindows/Mac/Linux/Android端末を一元管理できる

といった特徴を持つ製品。今回の発表により、上記の2つのデスクトップ仮想化製品でも利用できることとなった。

また同製品は従来より「VMware ESX/ESXi 4.0/4.1/ESXi 5.0/5.1」、「Citrix XenServer 5.6」、「Windows Server 2008 R2 Hyper-V（2.0）」、「Windows Server 2012 Hyper-V」などのサーバ仮想化製品には正式対応している。

CONTACT キヤノンITソリューションズ㈱
URL http://www.canon-its.co.jp

Service

シーズホスティングサービス、「BCP対応ファイルサーバープラン」を提供

シーズホスティングサービスは、8月1日より「BCP対応ファイルサーバープラン」の提供を開始した。

本プランは専用サーバ2台とSSLサーバ証明書がセットになったプラン。2台のサーバのうち1台はファイルサーバ（東京データセンターに設置）として、もう1台はバックアップサーバ（北九州データセンターに設置）として提供される。

契約後のファイルサーバのセットアップやバックアップ設定（利用状況に合わせた日次バックアップの世代数調整など）と、運用／監視は同サービスの専任エンジニアが対応するため、サーバに関する専門の知識や技術がなくても利用できる。

万が一ファイルサーバが利用できなくなった場合は、バックアップサーバよりFTPを利用してファイルの取り出しを行う。バックアップは1日1回ファイルサーバと同期して取得する。過去のデータは日次バックアップとして、最大7日分アーカイブ形式で保存する。

価格は、初期費用が50,000円（税抜）、月額費用50,000円（税抜）。

CONTACT シーズホスティングサービス
URL http://www.seeds.ne.jp

Letters from Readers

SDの読者プレゼントはちゃんと送っていますよ！

先日、某出版社の雑誌で読者プレゼントの数を水増ししていたことが、大きな問題になりました。SDはきちんと発送していますので、ご安心を！　ちなみにTwitterなどでは、SDのプレゼントは当選しやすいと評判です。ぜひ応募してみてくださいね。ただ、この評判を聞きつけて、応募が急激に増えたら当選しにくくなってしまうかな？　もっとプレゼントを集めねば！

2013年8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 3Dプリンタが未来を拓く理由

試作品が簡単に作れる、銃や家も作れる、というように良いイメージの話題ばかりが先行しがちな3Dプリンタ。実際には、何ができて何ができないのでしょうか？　本特集ではそれを探るべく、3Dプリンタのしくみや制作における試行錯誤にまで踏み込んで、多角的に紹介しました。

魔法の道具のように言われている3Dプリンタの実際のところがわかって、非常にためになりました。まだこれからですね。個人でも買える3Dプリンタのレビューも参考になりました。

埼玉県／犬棟梁さん

頭の中で漠然と思っていた疑問が、この記事により明確になった。とくにメンテナンス関係は、「メンテフリーとはいかないか」と痛感した。

埼玉県／ぜーたさん

3Dプリンタも個人でちょっとがんばれば手に入れられるようになってきましたね。すごい時代の変化です。でも、どう使いこなすかが問題です。

愛知県／kmさん

3Dプリンタが少しわかった気になりました。3Dプリンタを購入して、3D教育模型作成に挑戦したいと思っています。

富山県／YKB84さん

ソフトウェアとずれるかも……現在必要な技術ではありますが。

千葉県／Tayuさん

「『Software Design』なのにハードウェアの記事?」というご意見も多かったです。が、元となる3Dデータはソフトウェアで作るという意味では、ギリギリ守備範囲内でしょうか……。

第2特集　システムを見通す力でソフトウェア開発を楽にしませんか!

コードを読まずにバグが潜んでいる個所を見つけるQI法や、コードをレビューするコツなど、ソフトウェアの品質をチェックするためのさまざまな技法を紹介しました。「バグを狙い撃つ技術」は身についたでしょうか?

デバッグも「科学捜査」の時代になってきたと思った。

長崎県／じゃばすぷりくとさん

読ませたい人が読みたがらない内容。

東京都／HALさん

コードのソートは興味あり。

沖縄県／當眞さん

普段は細かな技術情報のほうがおもしろくて、そちらのほうに知識が偏っているのですが、当該記事のおかげでマクロな視点が持てて、勉強になりました。

石川県／Keiさん

コードレビューといっても、もうコードをとにらめっこする時代ではないのですね。今では、レビューも種々のテクニックを駆使して、科学的／効率的に行うものなのですね。

短期連載　小規模プロジェクト現場から学ぶJenkins活用

Jenkins短期連載の第2回目です。筆者のJenkins環境におけるワークスペース構築、JOB構成、Skypeとの連携について解説しました。

Jenkinsの活用も考えているが、なかなか導入するまでにいたっていないので、情報はうれしい

静岡県／伊藤さん

Jenkins活用術は導入検討に向けての知識として有用なので、もっとやってもらいたい。

千葉県／フェニさん

Jenkinsを導入したい／しているという読者は多いようです。HDDの

容量節約やJOB構成の工夫などは、みなさんのプロジェクトでも参考になるのではないでしょうか。

連載

「仮想化ネットワークの落とし穴」で、OpenFlowについて「これといって使い道がない」「優れた点としてアピールされている点がすべて致命的欠陥」と書かれていたのには笑いました。

東京都／山下さん

3Dプリンタ同様、夢のような技術として語られがちな仮想ネットワーク。導入における問題点などを冷静に分析する本連載も、8月号で最終回でした。欠点をきちんと見据えてこそ、適切な運用ができるようになるのではないでしょうか。弊誌では今後も、OpenFlowやSDNなどをウォッチしていきたいと考えています。

フリートーク

タブレットのブームも一段落ついたと思うので、ここら辺でタブレットを使った開発や管理などの特集があるとおもしろいかもしれません。

東京都／杉原さん

ご意見ありがとうございます。編集部にも新型Nexus 7やenchant MOONの実機がぞくぞくと届き、いろいろ触っては、「○○に使えそう」「もっと□□できるといいのに」などと話し合っています。今後、これらの端末に合わせてどんなアプリケーションが出てくるのか、楽しみです。

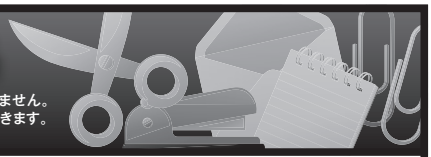
近所の家電量販店に買い物に行ったら、お目当ての商品が品切れでした。店員さんに「取り寄せできますか?」と聞いてみたら、「それよりもAmazonで買ったほうが早いですよ」と言われてしまいました。いろいろな意味で、時代は変わったのかな、と思いましたよ。

神奈川県／眞　泰志さん

商売ガタキを売りこんじゃっているのは天然ボケなんでしょうか。それはさておき、弊誌をAmazonで購入いただいている読者の方も多いです。確かに、欲しい物があるのに忙しくて買いにいけないとなると、利用してしまいますね。ただ、「Amazon中毒」にならないように気をつけないといけません。

エンジニアの能率を高める一品

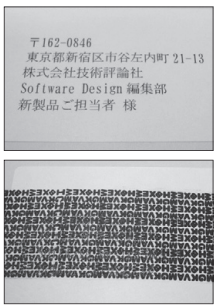
仕事の能率を高める道具は、ソフトウェアやスマートフォンのようなデジタルなものだけではありません。
このコーナーではエンジニアの能率アップ心をくすぐるアナログな文具、雑貨、小物を紹介していきます。



ローラーケシポン

735円（税込）／プラス㈱　http://www.plus.co.jp/

公共料金の明細書など個人情報が書かれた書類を処分する際、担当はシュレッダーハサミでジョキジョキ切っていました。が、これがけっこう握力を使うので、何通も処分するときは苦痛でした。そこで今回試したのが、この「ローラーケシポン」。インクの付いたローラーをコロコロと転がせば、個人情報を上から黒インクで塗りつぶせるのです（写真）。シュレッダーハサミより圧倒的に楽です。ゴシック体や手書き文字はほぼ読めなくなります。明朝体の文字はがんばれば読めるかも。紙や明細書などに使われているインクとの相性もありそうです。重ね塗りすれば、読みにくくなるかというとそうではなく、むしろ真っ黒にしてしまうと明らかに元の字が透けて見えますので、ご注意を。たいていの書類はこれで済ませられますが、確実に隠したいところでは、まだシュレッダーハサミも必要かもしれません。



▲写真　使用前（上）と使用後（下）

祝　8月号のプレゼント当選者は、次の皆さまです

①タッチパネル操作対応デジタルペンMK-WTP1 ..... 兵庫県　桑村　徹様
②Parallels Desktop 8 for Mac ..... 神奈川県　岩沢正美様

★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

2013年11月号
定価1,280円　176ページ

10月18日発売

［第1特集］思考をコード化する道具（エディタ）

我が友「Emacs」

「Emacsはエディタではなく環境である」名だたるプログラマやエンジニアから愛されているEmacs。この言葉どおりエディタという存在を超えて、むしろ創造的な開発環境として多くの技術者が支持し、手になじんだ道具として使いこなしています。それは、あたかも脳の延長であるかのようです。そんなEmacsの達人たちのノウハウを公開します。

［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テクノロジー全方位解説

今、Webサービスを支える重要な技術はサーバの高速化です。それを実現しているのは、サーバコンピュータで使用されているフラッシュメモリです。Fusion-io社のテクノロジーはこの分野で先進技術を持ち、多くのWebサービスやデータセンター、そしてクラウド事業者から支持を受け活用されています。Fusion-io社のテクノロジーを全方位から紹介&解説します。

［特別企画］ソーシャルゲームを支えるクラウド

魔法少女リリカルなのはINNOCENTの舞台裏

ユビキタスエンターテインメント＋IDCフロンティアによるソーシャルゲーム開発・運営の舞台裏を紹介!

［新連載］

FreeBSD活用「チャーリールートからの手紙」

※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

「システムで必要なことはすべてUNIXから学んだ」（第10回）は都合によりお休みいたします。

SD Staff Room

●年間計画で特集が決まっている本誌ですが、某健康雑誌でも同じように年間で決まっていることを知り、IT業界も一回りしたんだなと感じています。UNIXリバイバル特集記事の反応の良さが証左です。知らないうちに世代は変わっていきます。うっかりすると「ゆで蛙」になるので注意しています。（本）

●念願のNexus7 2013が届いた。旧Nexus7に触ると速度や重さの違いは歴然。Twonky BeamやHuluなどの動画アプリもサクサク動く。Google Mapでマイマップが使えなくなって愕然としたが、Chrome上からGoogle Mapページを開けば使えることを確認。よかった。（幕）

●今年の夏は道南で羊飼い体験のできる宿に宿泊。ミルクやりをしましたが、群れの中から仔羊が走り寄ってきてちょーカワイイー。飲んでるときは短いしっぽをぶるぶるふって、これまたカワイイー。そして夕食には仔羊のステーキ。命を頂戴するありがたさを心に刻みつつ、美味しくいただきました。（キ）

●「Googleリーダー難民」になってしまった。Googleは大胆にサービスを終了していくのは、前々からわかっていたが、まさか自分が被害者になるとは……。子供のころ、7年ほど通っていたスイミングスクールが倒産して、「スイミングスクール難民」になってしまったときのことを思い出した。（よし）

●先日沢登りに行ってきました。初心者コースだから大丈夫と言われていましたが、いざ登ってみると早速ロープの出番。これが初心者コース?? と思っていたら案の定中級者コース……。簡単じゃおもしろくないからね、とは引率者のお言葉。涼みに行ったはずが青あざ&筋肉痛のお土産つきとなりました。（ま）

本誌に記載の商品名などは、一般に各メーカーの登録商標または商標です。　© 2013 技術評論社

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
2013年10月号

発行日
2013年10月18日

●発行人
片岡　巌

●編集人
池本公平

●編集
金田富士男
菊池　猛
吉岡高弘

●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。